

Paper 292-2012

Finding Your Inner Query with SAS® Enterprise Guide®

Michael Burke and I-kong Fu, SAS Institute Inc., Cary, NC

ABSTRACT

SAS® Enterprise Guide® 5.1 introduces the concept of reusable, query-based templates. SAS Enterprise Guide users have been able to create task templates to save custom settings for some time now. Query-based templates offer the ability to create similar functionality from the SAS Enterprise Guide Query Builder. These templates allow speedy access to previously designed queries that might have included time-consuming work spent fashioning the perfect table join, computed column, or filter. SAS Enterprise Guide 5.1 also allows you to reuse certain types of query-based templates as subqueries in computed columns and filters. This paper describes how query-based templates work and how to use them to answer special problems by turning them into subqueries in the point-and-click Query Builder interface.

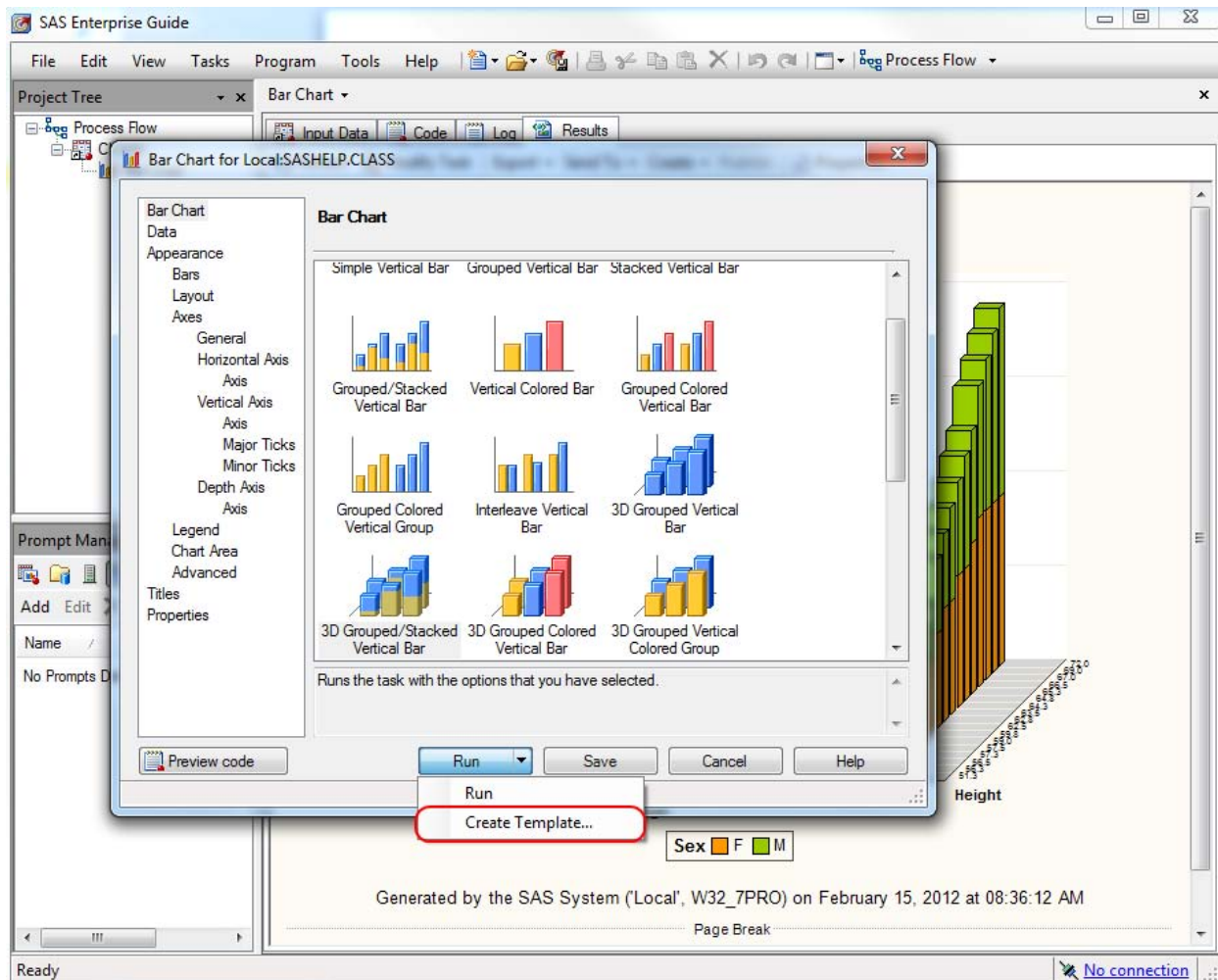
INTRODUCTION

BACKGROUND ON TASK TEMPLATES IN SAS ENTERPRISE GUIDE

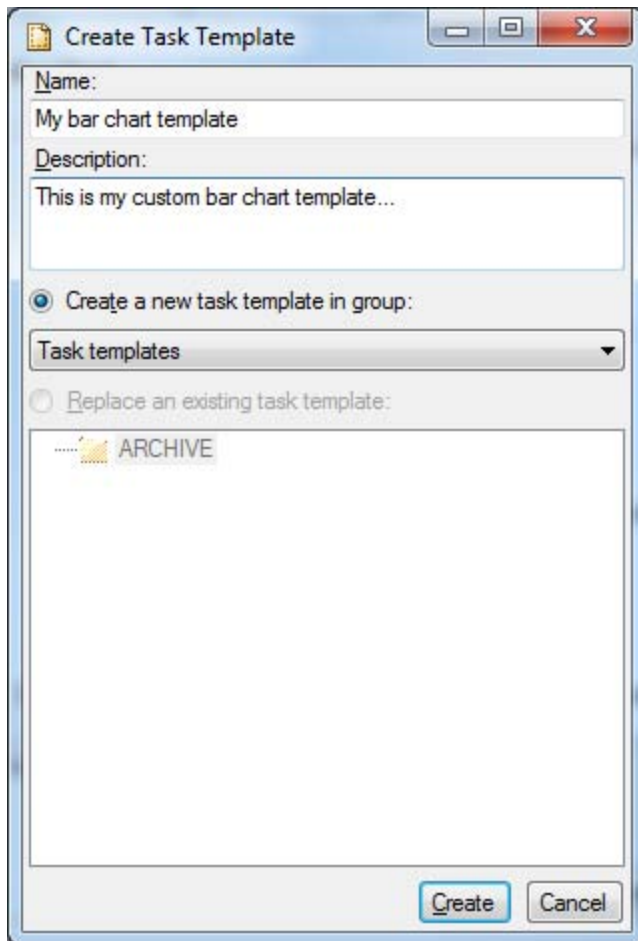
SAS Enterprise Guide 4.2 introduced the ability to create task templates. Consider the Bar Chart task. You can spend a great deal of time customizing the settings for the resulting chart's appearance. You work to modify options such as custom colors, formatting how the axes will display, and customizing the legend in order to produce an image that describes your particular data clearly and with a style matching the preferences of you and your organization.

Task templates allow you to save those choices by selecting the **Create Template** option from a drop-down list that is available in most tasks.

Finding Your Inner Query with SAS® Enterprise Guide®

**Display 1. Option to Create a Task Template in the Bar Chart Task**

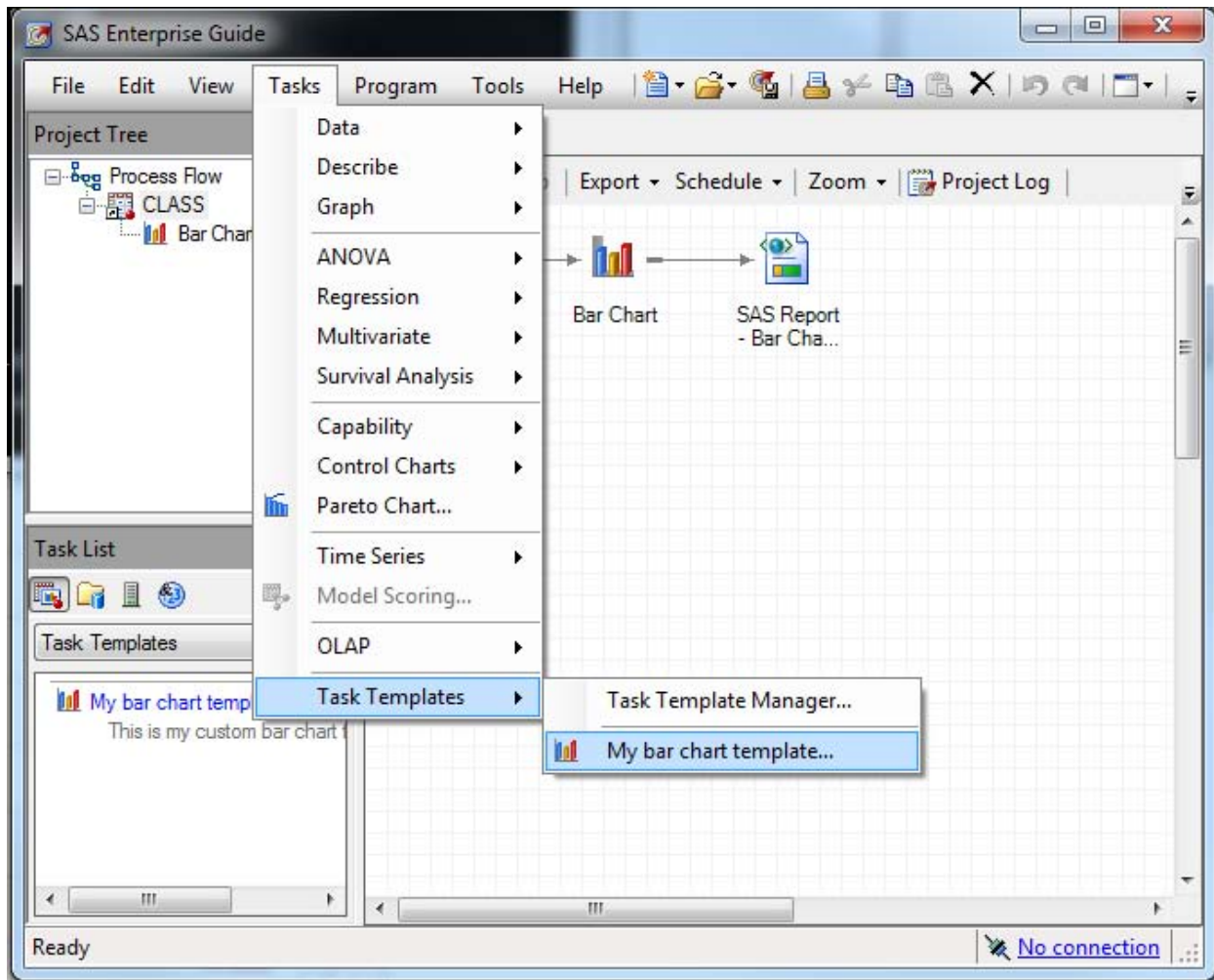
Finding Your Inner Query with SAS® Enterprise Guide®



Display 2. Dialog Box Allowing You to Describe and Create a Task Template

The Create Task Template dialog box appears after you select **Create Template**. In this dialog box, you specify the name of the template and where to save the template.

Finding Your Inner Query with SAS® Enterprise Guide®

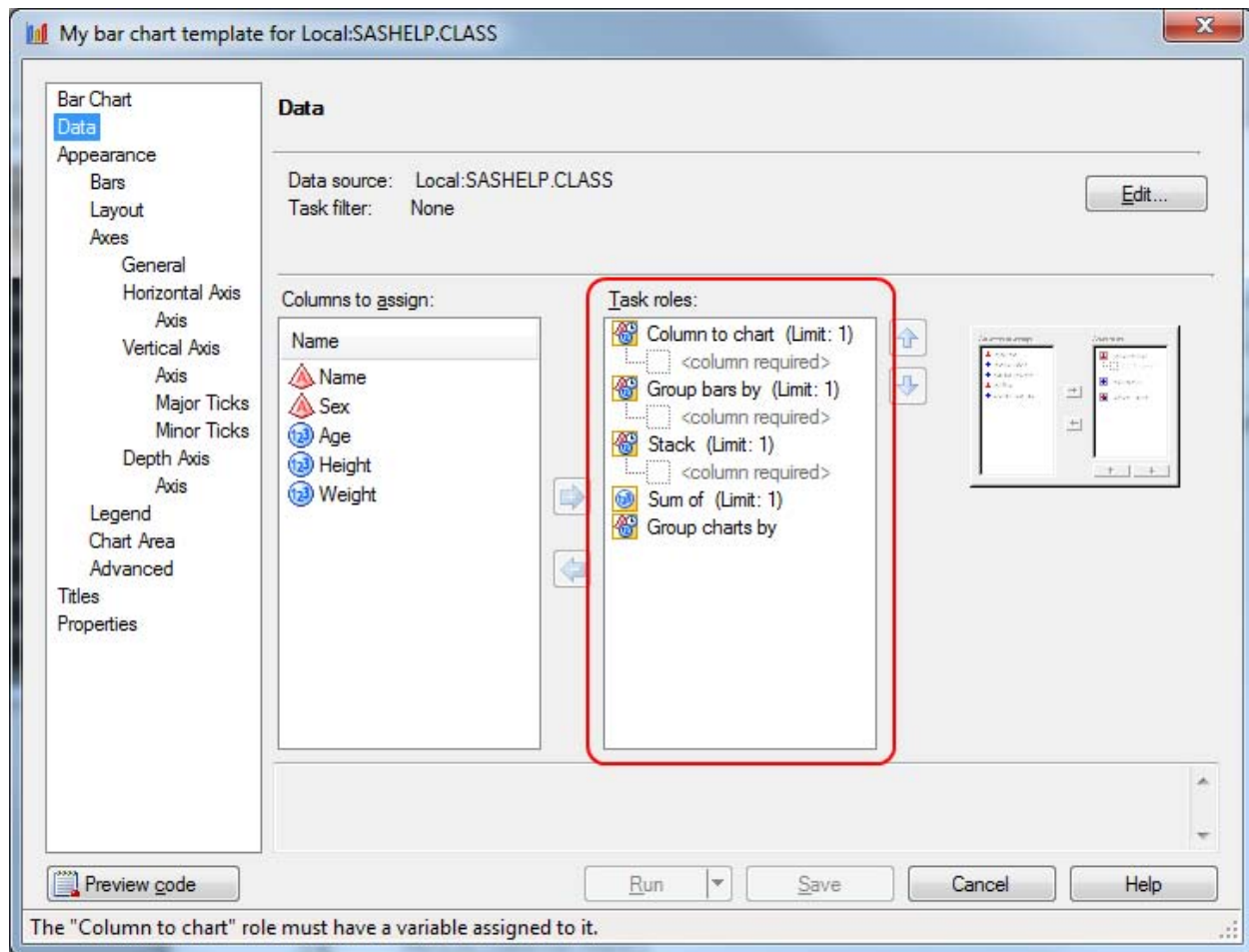


Display 3. How to Start a Task Template You Have Previously Saved

After the template is created, you can quickly reproduce similarly styled charts based on different data or with different roles. You access the saved template from the Template Manager, or by selecting the template name from **Tasks->Task Templates**.

When you open a previously saved task template, a task of the type you saved starts. The task options are already filled in with the settings that you specified for the template. However, settings that are data specific are not saved with a task template. For example, any task roles that you assigned and the actual input data are not saved as part of your template. When you select the template, the task uses the data that is currently selected in the process flow, or if there is no data is selected, a file selection dialog box appears so you can choose the input data. After the input data is selected, task roles need to be assigned before you can run the new task.

Finding Your Inner Query with SAS® Enterprise Guide®



Display 4. Task Role Settings Are Not Saved in Task Templates

QUERY-BASED TASK TEMPLATES AND SUBQUERIES IN SAS ENTERPRISE GUIDE 5.1

The initial concept of task templates in SAS Enterprise Guide did not include any work that you might have done in the Query Builder. Task templates were meant to save custom settings, but were designed to be a data independent so they could be used again quickly on any input data source. In contrast, work done in the Query Builder is all about data.

In the Query Builder, there are fewer data independent settings. Most of the time-consuming choices you make in the Query Builder are contingent on the structure of the input data sources, such as the joins involved in the query, any filters you might define, computed columns you might design, the result items you might pick, and how the result items are ordered for the query. Considering the large number of choices that can be made in a single query fashioning all of these elements into a query that exactly meets your business needs can represent a large, time-consuming effort.

SAS Enterprise Guide 5.1 extends the concept of task templates in a way to include any work that you might have done in the Query Builder. Query-based templates are offered as a way to save the entire structure of a query and retrieve it at a later time. This is not done in a purely data independent way like regular task templates. Rather, query-based templates let you modify your input data sources at the time you start the template. In this way, you can select data that could require the same query structure as the original, but which might have different names and locations from the original query.

Certain types of reusable query-based templates have been targeted in SAS Enterprise Guide 5.1 to be also useable as subqueries inside the Query Builder. Specifically, query-based templates that have a single result item, but which might return single or multiple values, can now be accessed as part of a filter condition in the Basic Filter. These subqueries can be also be used as part of a replace condition in the Recoded Column Wizard (that you use to define

Finding Your Inner Query with SAS® Enterprise Guide®

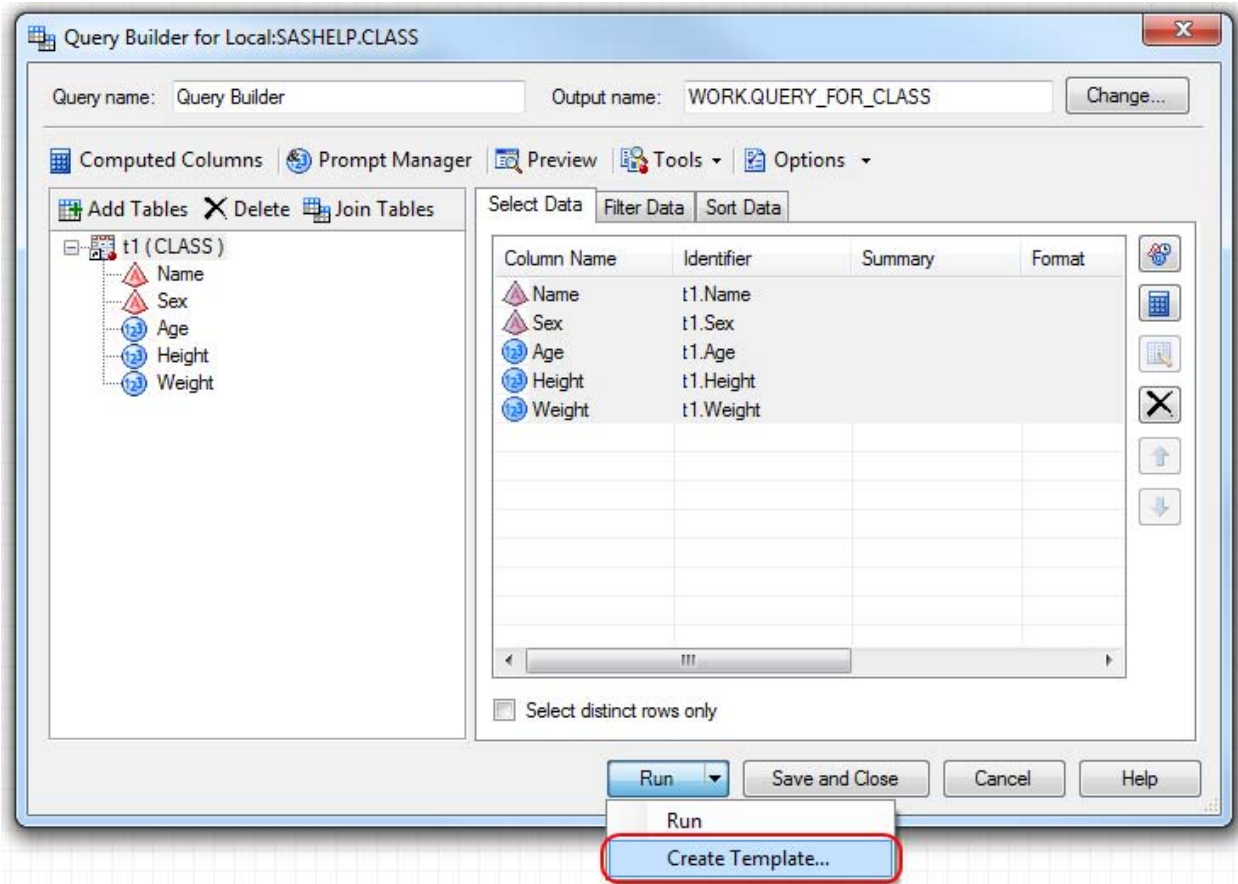
a computed column). This use of subqueries enables you to create queries that are more dynamic in the case of recoding columns, and in the case of filters, enables you to answer certain questions through queries that may be difficult to organize and answer in few steps without the use of subqueries.

The purpose of this paper is to walk through how query-based templates work and how they can save you time in SAS Enterprise Guide 5.1. Then this paper discusses how certain templates can be used to answer special problems by turning them into subqueries in the point-and-click Query Builder interface.

REUSABLE QUERY-BASED TEMPLATES IN SAS ENTERPRISE GUIDE 5.1

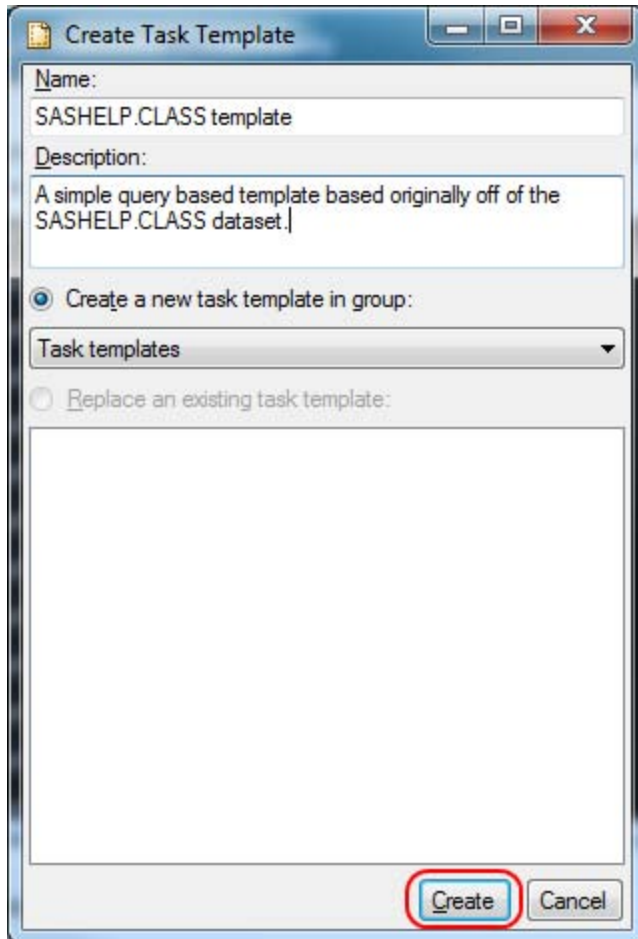
HOW TO MAKE QUERY-BASED TEMPLATES

Creating a query-based template begins with designing a regular query in the Query Builder. Creating a query might be an iterative process where you work on the query, run, review the log and results, and then work on the query some more. After you have decided that your query is in a form you would like to be able to quickly retrieve a later time (or from another SAS Enterprise Guide project), it is time to create the template. In SAS Enterprise Guide 5.1, you will notice a drop-down list under the **Run** button in the Query Builder. Selecting the **Create Template** option continues the process of making a query-based template.



Display 5. Query Builder Option to Create Template

Finding Your Inner Query with SAS® Enterprise Guide®



Display 6. Creating a Query-Based Template

After you select the **Create Template** option from the main Query Builder interface, the Create Task Template dialog box appears. (This is the same dialog box that appears if you were creating a template for a task.) Again, this dialog box gives you a chance to give your template a name and description to help you pick it out later on.

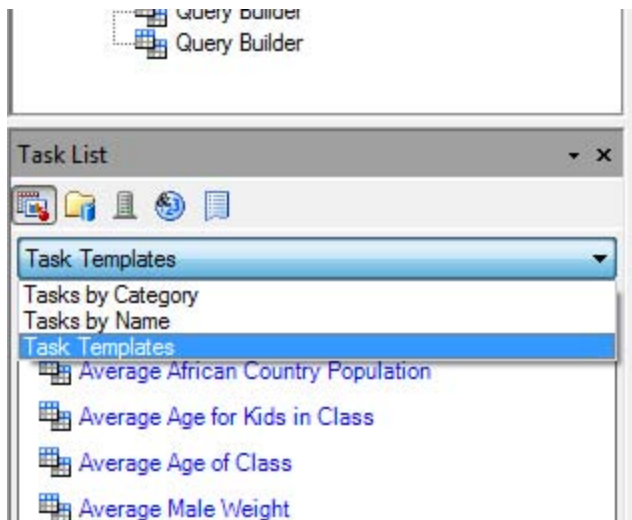
HOW TO USE QUERY-BASED TEMPLATES

After you create your query-based template by clicking **Create** in the Create Task Template dialog box, the template is ready to go. You are returned to the main interface of the Query Builder, where you can continue to work with that particular query or you can close the Query Builder altogether. Now, you can access that query template in a couple of different ways either in the same project or from a different project.

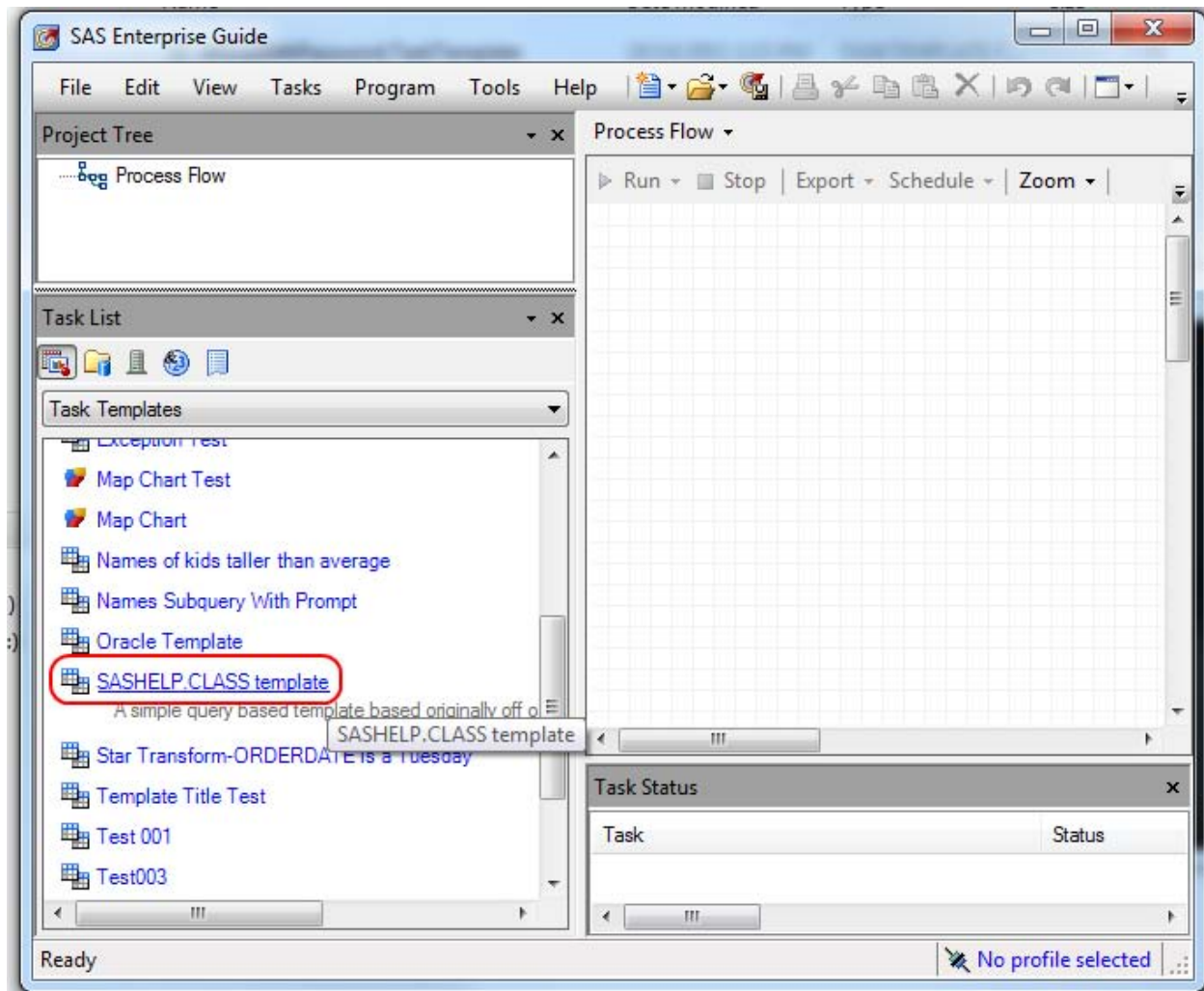
Starting a Query-based Template from the Task List

The main SAS Enterprise Guide interface features a view (in the bottom left corner by default), called the Task List. To set focus on this area, select **View->Task List**. The Task List area has a drop-down list of its own that you can use to switch between a list of available tasks sorted by name, a list of tasks sorted by category, and a list of available task templates saved on the client machine. One way to start a saved query-based template is to select the template name in the Task List.

Finding Your Inner Query with SAS® Enterprise Guide®



Display 7. Close-Up of the Task List

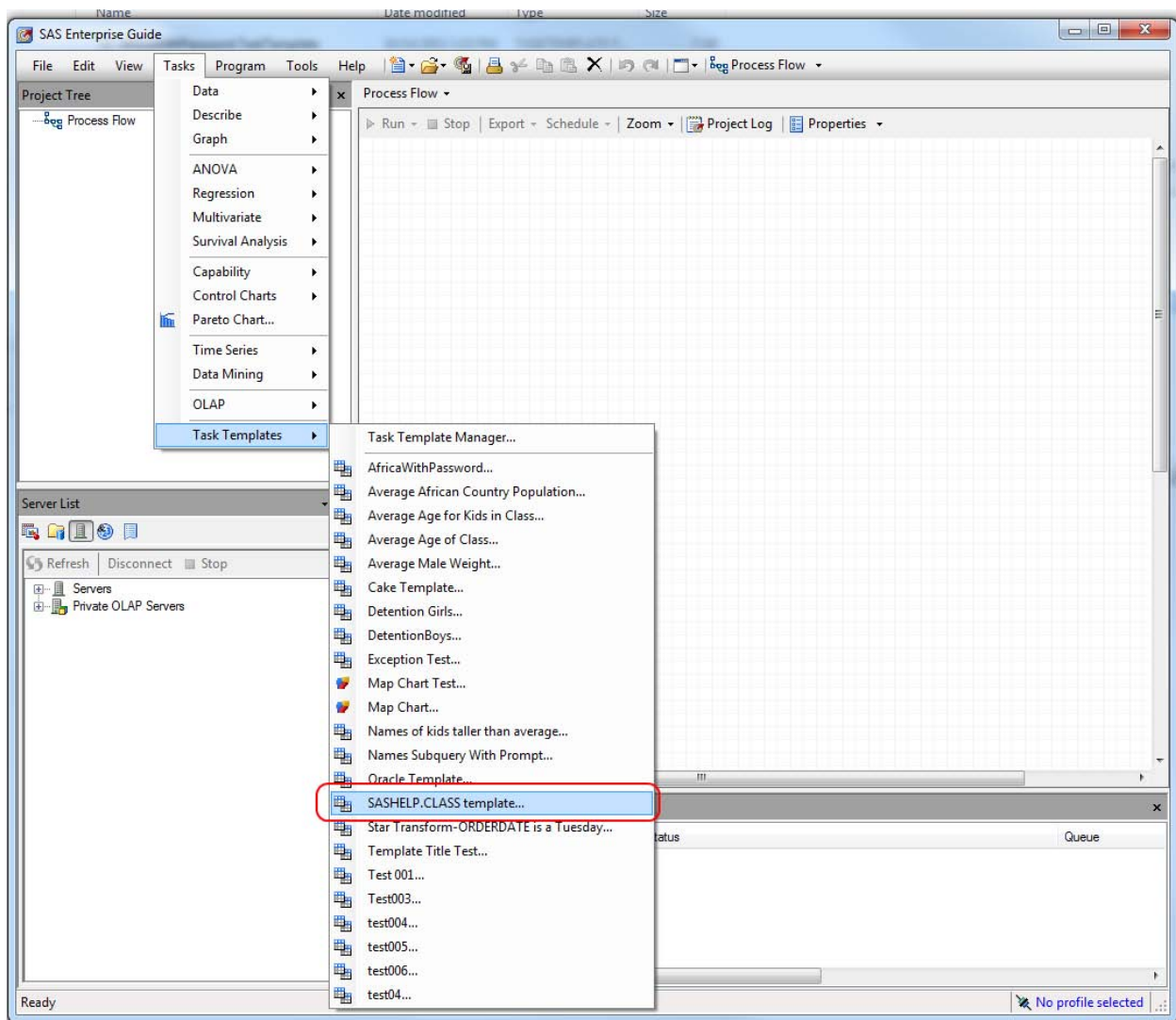


Display 8. Starting a Query-Based Template from the Task List

Finding Your Inner Query with SAS® Enterprise Guide®

Starting a Query-based Template from the Main Menu

Possibly the quickest way to initiate a query-based template and add that query to the process flow is to select its name from **Tasks->Task Templates**. The more templates you have, however, the more difficult this menu is to navigate.



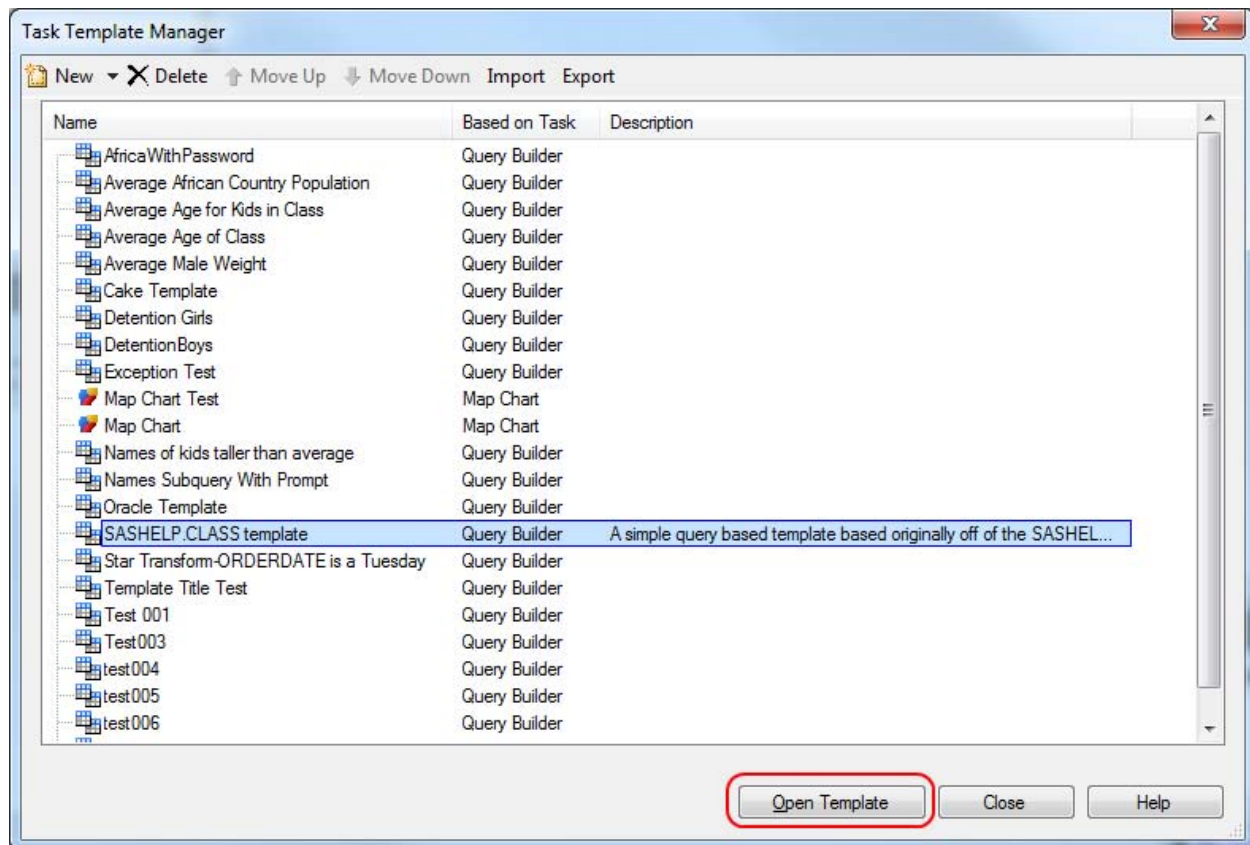
Display 9. Starting a Query-Based Template from the Main Menu

Starting a Query-based Template from the Template Manager

To open the Template Manager, select **Tasks->Task Templates->Task Template Manager**. The Template Manager might be the easiest to use if you are working with a large number of templates. The templates are presented in a list where you can sort by name, by task, and by description. Query-based templates are identified by 'Based on Task' of 'Query Builder'. You can start the template by double-clicking the name, or selecting the name and clicking **Open Template**.

When you are searching for a template, you can see the value in providing a meaningful name and description in the Create Task Template dialog box.

Finding Your Inner Query with SAS® Enterprise Guide®



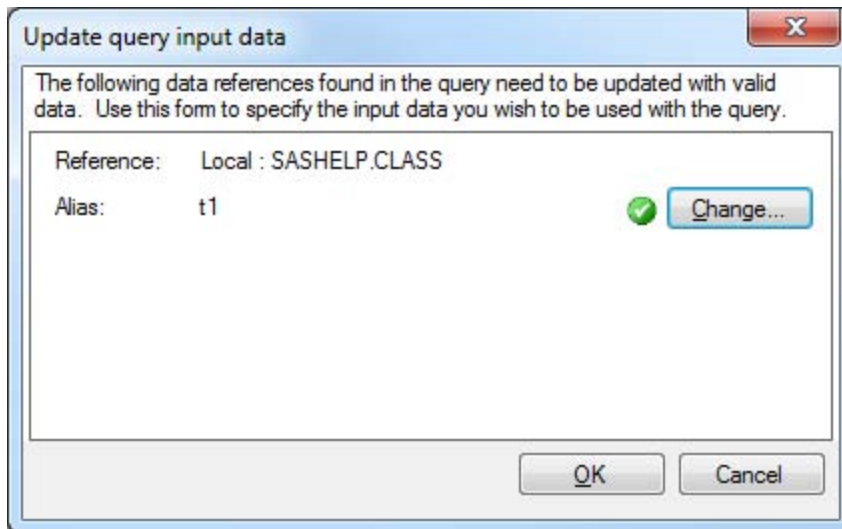
Display 10. Starting a Query-based Template from the Template Manager

Update Query Input Data

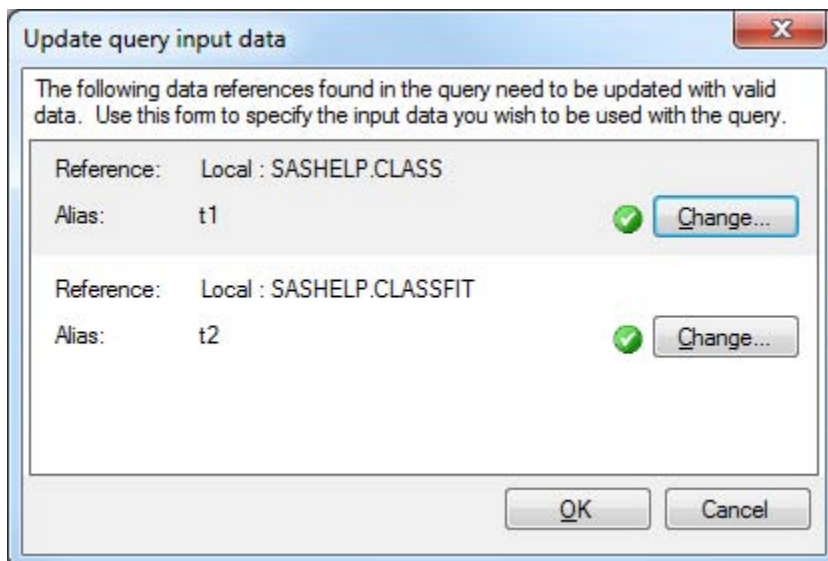
After you start a query-based template from the Task List, main menu, or Template Manager, the next step is to update the input data so that the template refers to data sources in the current project. The Update query input data dialog box, which lists all of the input data sources for the query, is shown below. The original server, libref, member name, and table alias are shown for each input data item in the query.

For each input data item, the **Change** button enables you to change the data source for that input item. After all of the input data items are detected as available on the current server, a green check mark appears, and the **OK** button is enabled.

Finding Your Inner Query with SAS® Enterprise Guide®

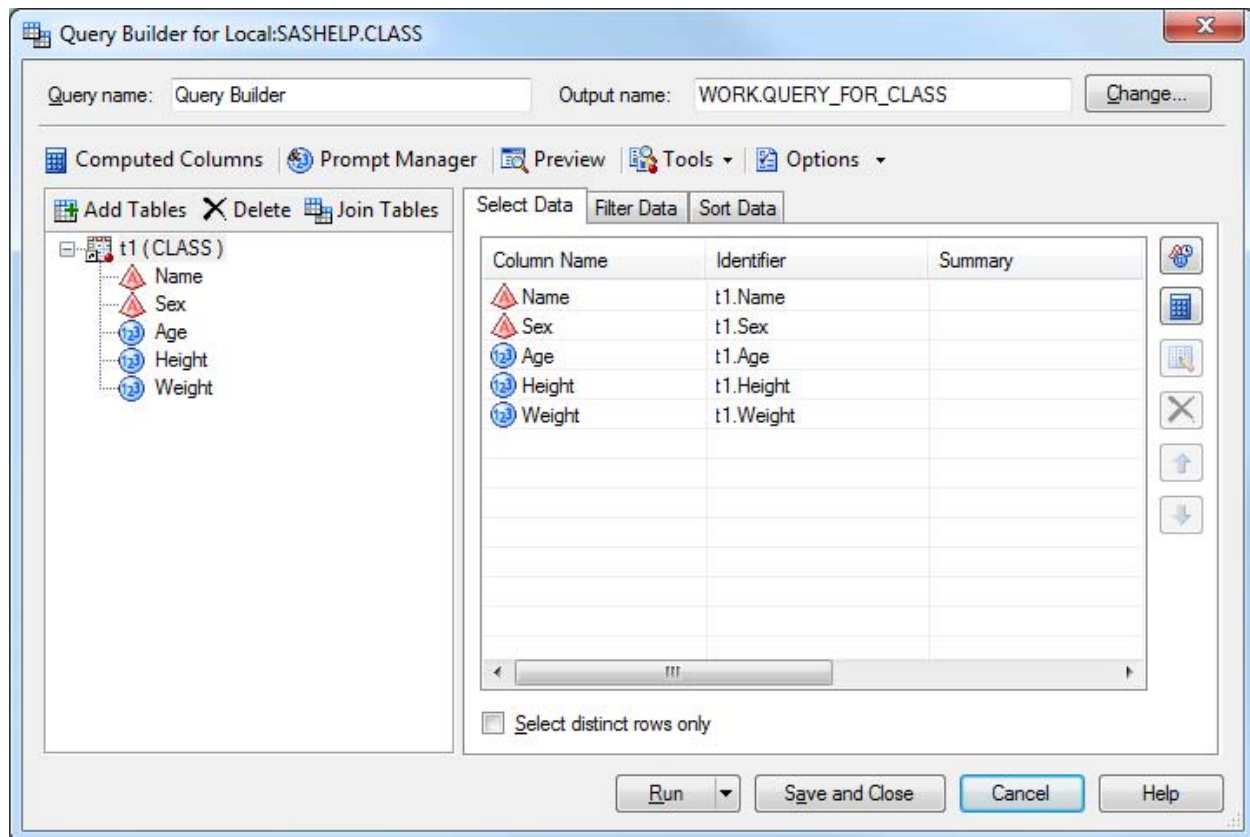


Display 11. Update Query Input Data Dialog Box with a Template Containing One Input Data Source



Display 12. Update Query Input Data Dialog Box with a Template Containing Two Input Data Sources

Finding Your Inner Query with SAS® Enterprise Guide®



Display 13. A Query Created from a Query-Based Template

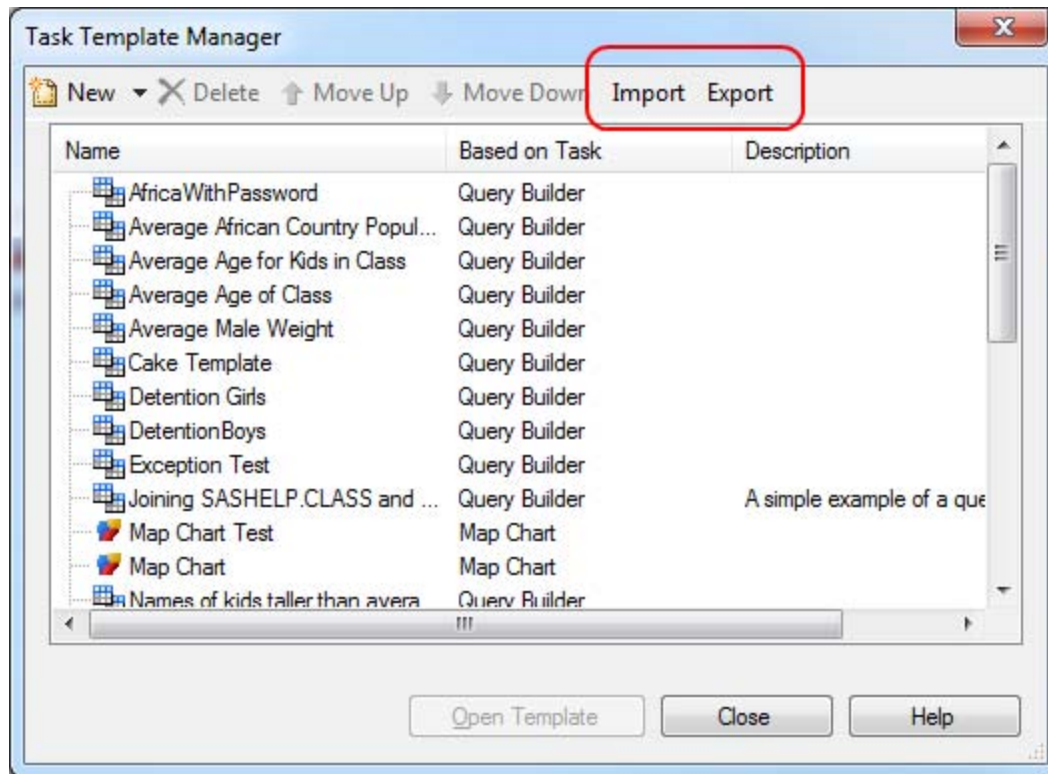
After you click **OK** in the Update query input data dialog box, the normal Query Builder interface appears. All of the query selections, computed columns, filters, sorts, joins, and query options from the original query are restored. Any data sources that refer to data not already in the project are added, and existing data items are linked to a new Query Builder node in the process flow. From this point forward, the Query Builder behaves as it normally does when creating a new query.

COLLABORATION

The Task Template Manager dialog box allows you to import and export templates in order to share work with others and to facilitate moving templates created on one machine to another. It is useful to be able to share task templates with a colleague. The ability to share query-based templates adds another convenience for groups that may have a few individuals specializing in data management while other individuals focus on data analysis or reporting.

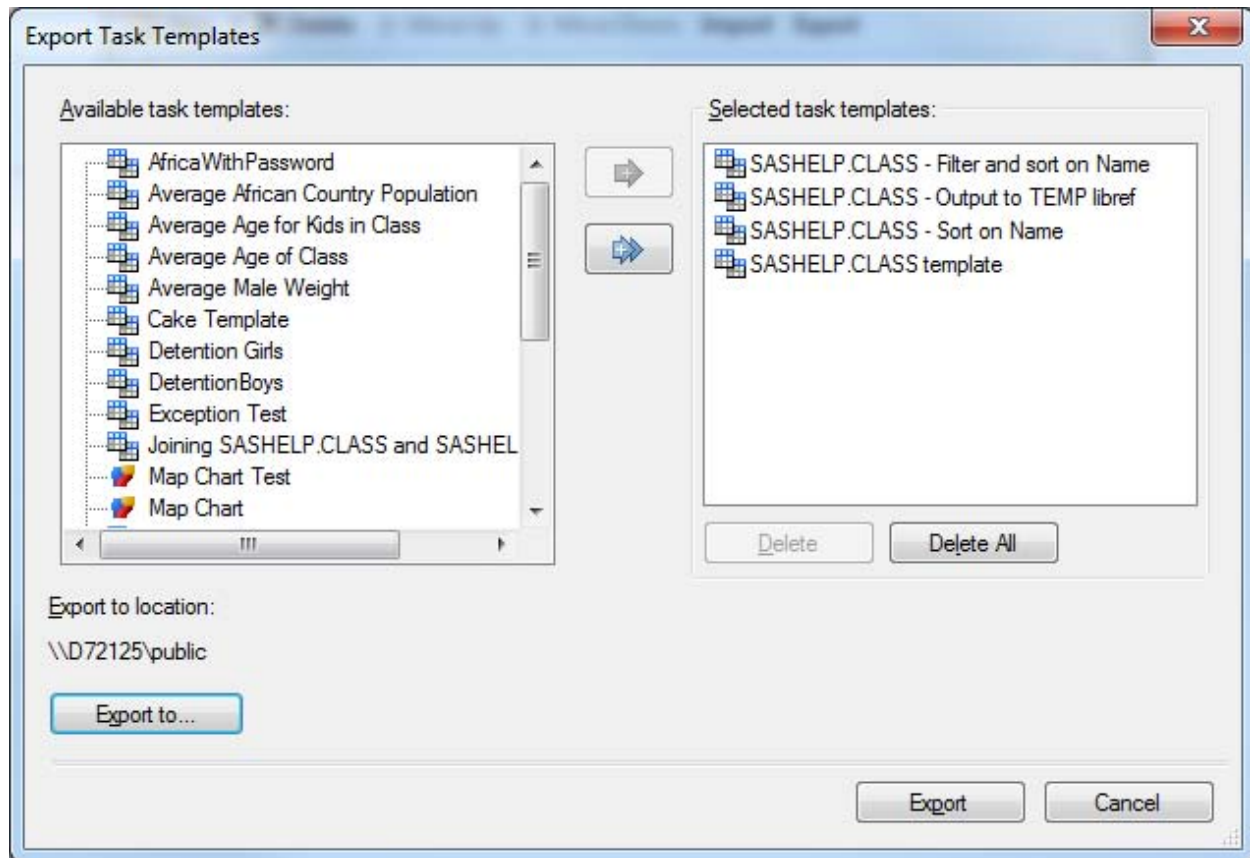
In some situations, the right answer might be for data managers to develop views or special processes to populate certain tables on an agreed upon schedule. However, for those working in a SAS Enterprise Guide environment who want a lighter weight, more flexible, and ad hoc solution, the individuals who know the data the best can use the Query Builder to develop the correct subset of data and then 'hand it off' to other consumers of that data via a query-based template. After a data analyst or report builder imports a query-based template to their client machine, the data analyst or report builder simply selects the template to quickly add a finely tuned data set to their process flow.

Finding Your Inner Query with SAS® Enterprise Guide®



Display 14. The Import and Export Options in the Task Template Manager

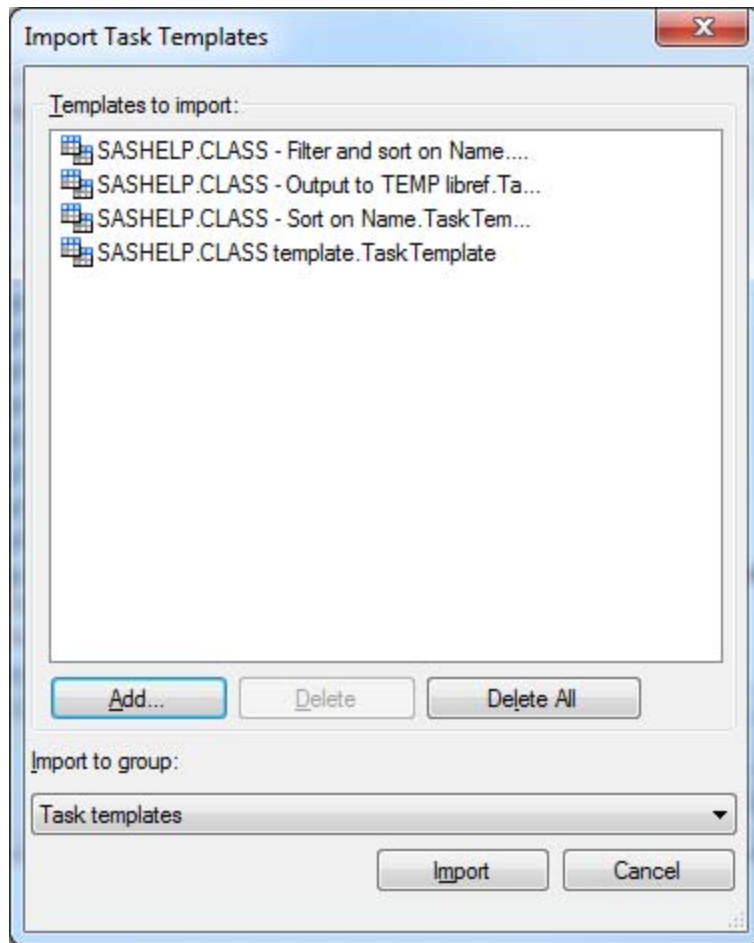
Finding Your Inner Query with SAS® Enterprise Guide®



Display 15. Exporting a Query-Based Template to a Network Location

When exporting templates, click **Export to** select the location where you want to save the template on your machine or network.

Finding Your Inner Query with SAS® Enterprise Guide®



Display 16. Importing Templates

When importing task templates you can click **Add** to browse to a location on your machine or network.

PROBLEM SCENARIOS

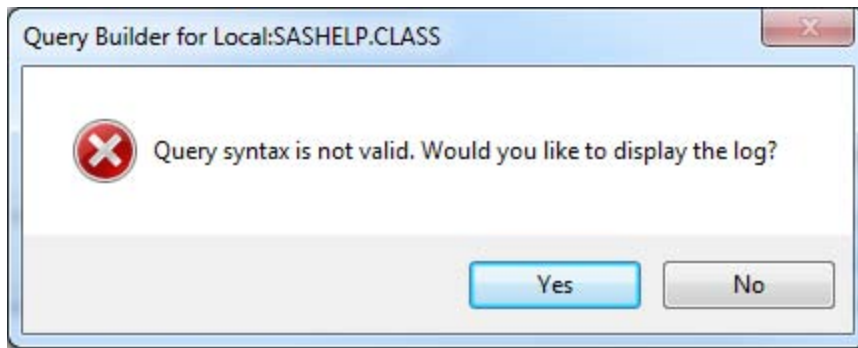
What if the new input data has extra columns that do not appear in the query-based template?

If you reuse a query-based template on data that has columns in addition to those that were included in the template, this will not affect the resulting query. So, this scenario is not really a problem. However, if you need to see those extra columns in the output, you need to manually add them to the list of result items on the **Select Data** tab of the Query Builder.

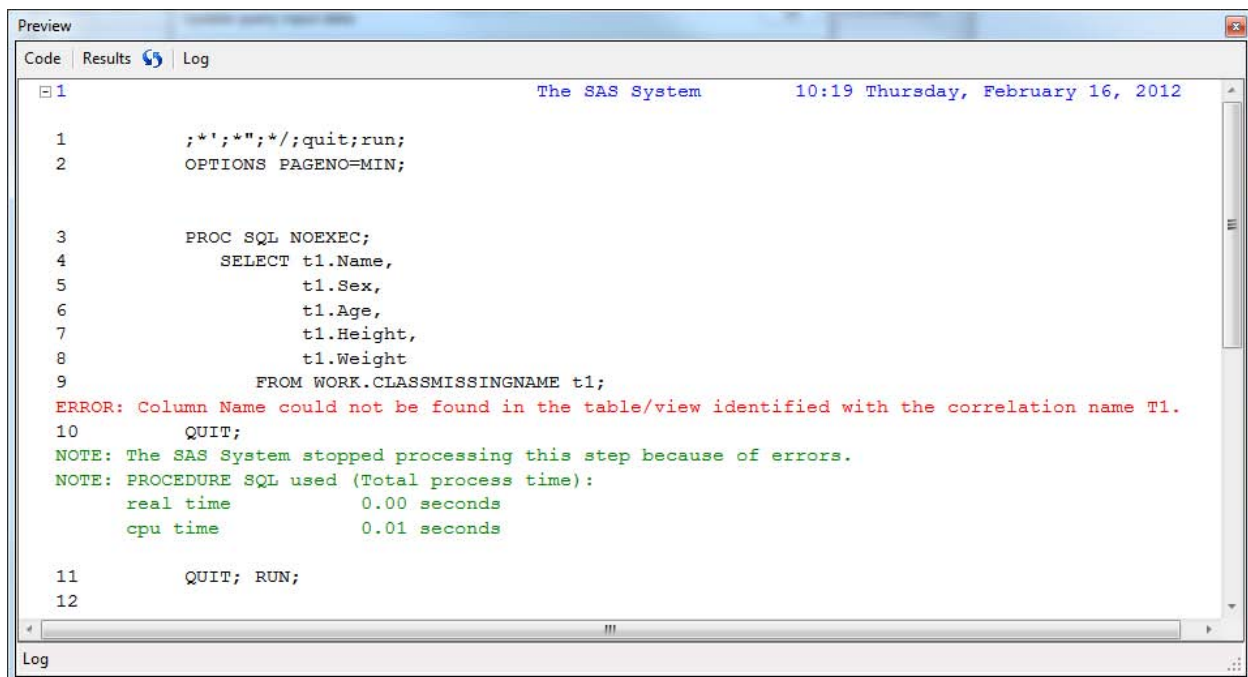
What if the new input data is missing columns that are referenced in the query-based template?

This is a problem. If you open a query-based template and change the input data to a source that does not include a column referenced in the result item list for the query, then during the load of the main Query Builder view you will get a warning that the `Query syntax is not valid`. If you examine the log, you will see a note that a certain column could not be found in the table or view involved with the query.

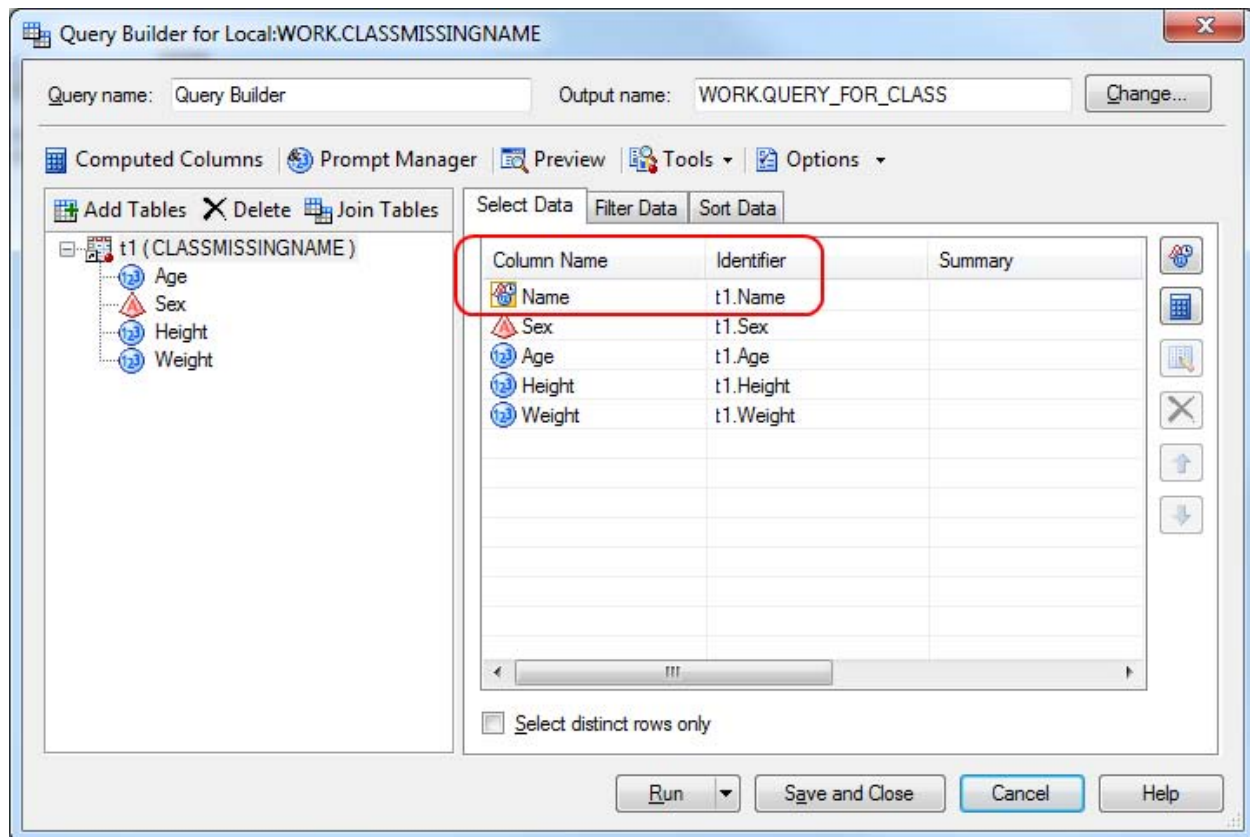
Finding Your Inner Query with SAS® Enterprise Guide®



Display 17. An Indication That Something Is Amiss

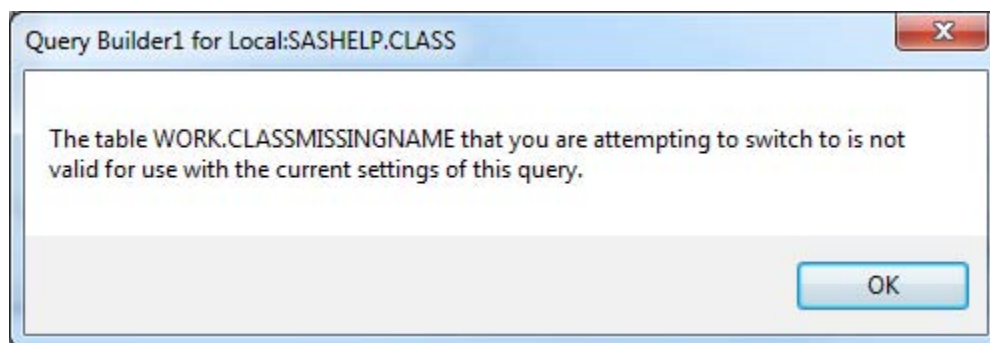


Finding Your Inner Query with SAS® Enterprise Guide®



Display 19. Placeholder for a Column Used in the Result List but Missing from the Input Data Source

A more problematic scenario is when the missing fields are referenced in a join, filter, or sort. Because of the way joins, filters, and sorts are processed in the Query Builder, columns used in these tasks are considered more important to the query itself. As a result, trying to match data with missing columns used in joins, filters, and sorts will fail with a message saying that the data is incompatible with the query.



Display 20. If the Missing Column Is Used in a Join, Filter or Sort, Then Different Data Needs to Be Used

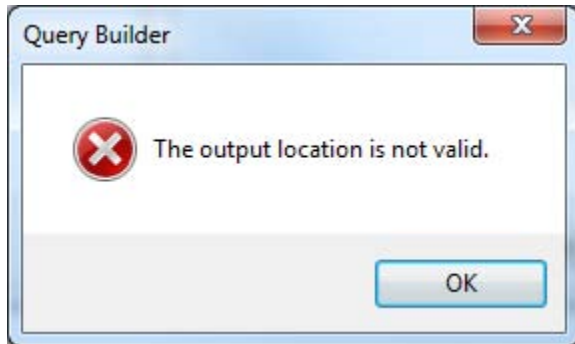
In this situation if the query-based template must be salvaged because of extensive design work, a workaround can be used to add columns if you are able to modify the input data source or copy the input data into a table which you can modify. The following example shows syntax that could be used in a program node in SAS Enterprise Guide, where a template references a column "Name" in a join, filter, or sort and where the table you want to modify is Work.ClassMissingName. After the input data contains the required columns (even if these columns are empty) the template can be opened and modified to remove the unwanted column from the query.

Finding Your Inner Query with SAS® Enterprise Guide®

```
proc sql;  
    alter table Work.ClassMissingName  
        add Name character;  
quit;
```

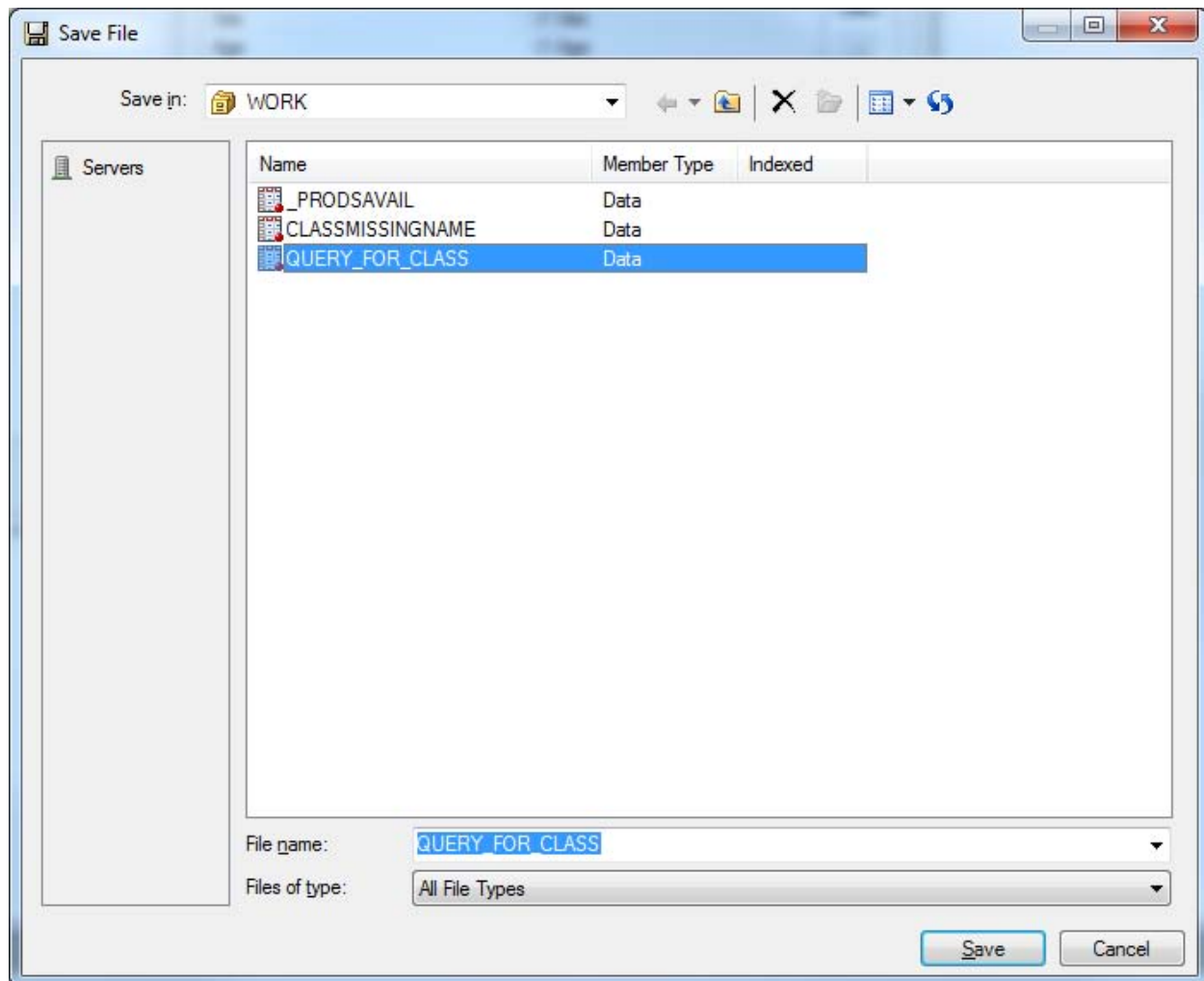
What if the template output location is no longer valid?

If a query-based template was created with an output location that is a permanent SAS library, it is possible that library will not be assigned at the time the query-based template is opened. If this happens, you are notified that the output location is not valid. However, you are immediately given the chance to select a valid output location, either in the Work library or some other existing library, in order to bring the query-based template back to life.



Display 21. Message Shown When Using a Template Writing to a Library That Does Not Exist

Finding Your Inner Query with SAS® Enterprise Guide®



Display 22. Picking a Valid Output Location

TURNING QUERY TEMPLATES INTO SUBQUERIES IN SAS ENTERPRISE GUIDE 5.1

WHAT KINDS OF SUBQUERIES ARE SUPPORTED?

A subquery, or inner query, can be thought of as a query nested inside of another query. SAS Enterprise Guide 5.1 allows the use of certain types of query-based templates as subqueries in the Query Builder. In this context, a subquery is a previously defined query-based template that can be used as an expression in the Query Builder in order to determine how to select rows in another table or to help determine when to recode a column in a computed column. In general, there are a few different types of subqueries that can be used in SQL, but all of these are not targeted in the point-and-click approach for using query-based templates as subqueries in SAS Enterprise Guide 5.1.

Here are the types of subqueries that you can create using query-based templates:

- Subqueries that return a single value
- Subqueries that return multiple values (multiple rows of a single field)
- Subqueries that appear as part of a filter of the raw data (on the WHERE clause)
- Subqueries that appear as part of a filter on the grouped data (on the HAVING clause)

Finding Your Inner Query with SAS® Enterprise Guide®

- Subqueries that appear as part of a recode condition as part of recoding a column in a computed column (on the SELECT clause)

Here are the types of subqueries that you cannot create using query-based templates:






- Subqueries that form a derived table (subqueries that appear on the FROM clause)
- Subqueries that refer to columns on the outer query (correlated subqueries)

SUBQUERIES AS FILTERS

Using query-based templates as subqueries enables you to create a subquery as a Query Builder expression that informs the outer query on how to select rows. Although the subquery might happen to refer to the same table as one involved in the outer query, its treatment will be as an independent, noncorrelated query that will return a result set to be used by the outer query in further processing. Using subqueries in this way can allow you to organize a query and answer certain questions that would be difficult to do in few steps without using subqueries.

Who is taller than the average child in class?


An interesting problem that can be solved using subqueries is when you need to compare a column's value to the scalar value of a summary function. Working with the SASHELP.Class data set, consider the question, "Who is taller than the average child in class?"

	 Name	 Sex	 Age	 Height	 Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84
3	Barbara	F	13	65.3	98
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5
9	Jeffrey	M	13	62.5	84
10	John	M	12	59	99.5
11	Joyce	F	11	51.3	50.5
12	Judy	F	14	64.3	90
13	Louise	F	12	56.3	77
14	Mary	F	15	66.5	112
15	Philip	M	16	72	150
16	Robert	M	12	64.8	128
17	Ronald	M	15	67	133
18	Thomas	M	11	57.5	85
19	William	M	15	66.5	112

Display 23. A Grid View of the SASHELP.Class Data set

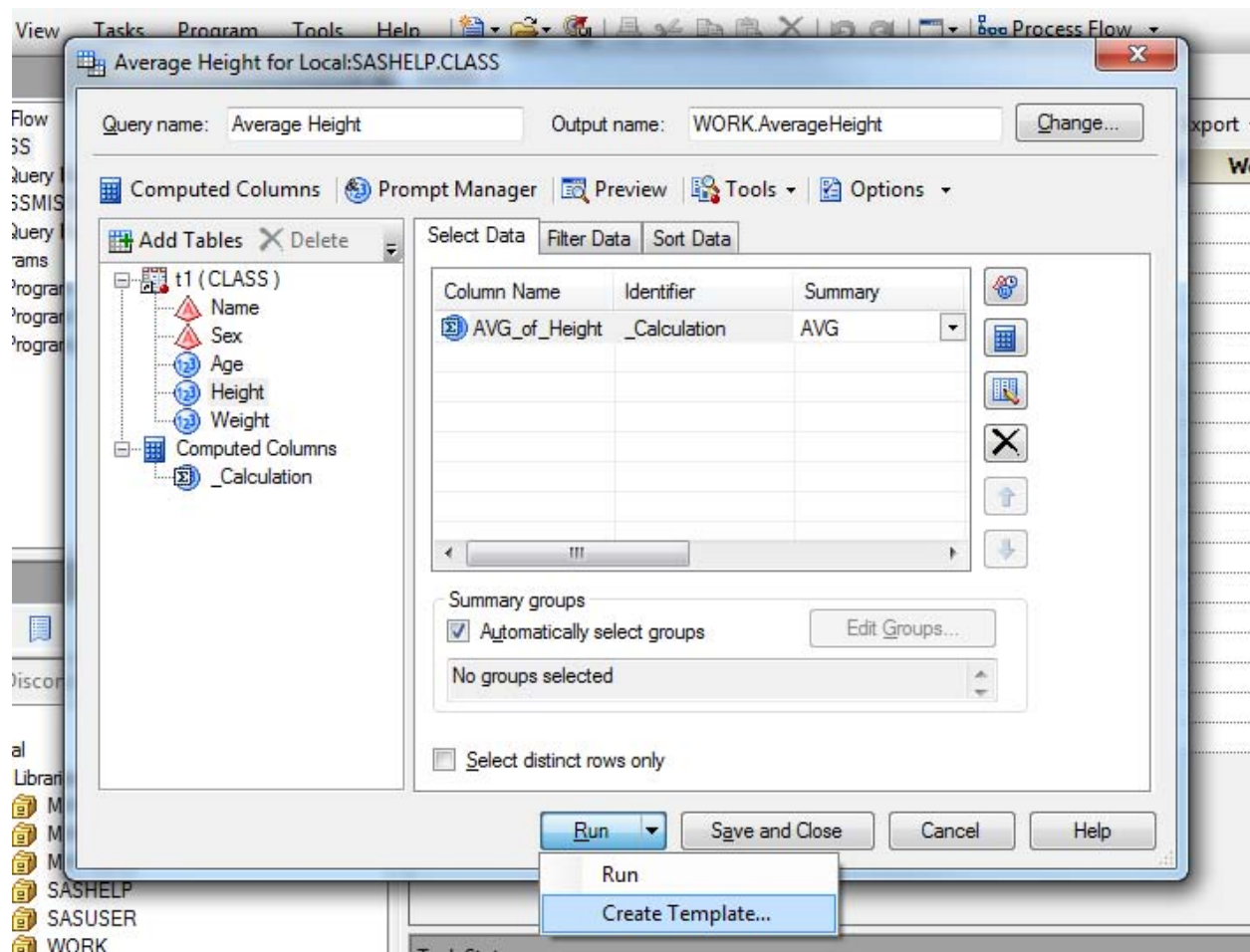
To answer this question, you need a way of determining the average for height for the whole class and at the same time compare that value with each child's individual height. Using subqueries can help answer this question. The first step in tackling this problem is to design a simple query in the Query Builder. This query calculates the average height for students in the class using a summary function. After the query is created, you can use the **Create Template** option to save this as a reusable query-based template.

When you run this query, you see that the average height comes out to be around 62.34.

 AVG_of_Height
1 62.336842105

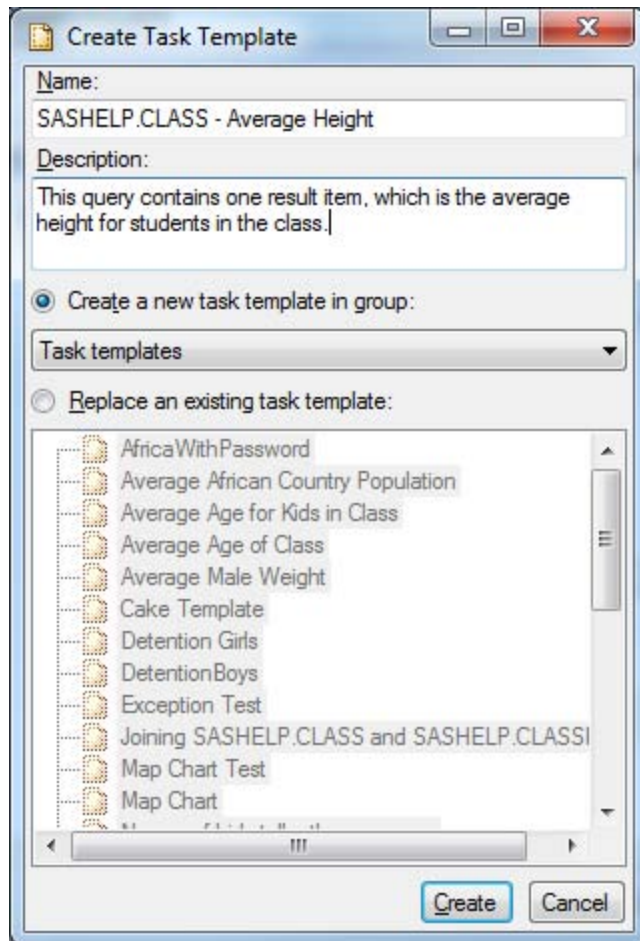
Finding Your Inner Query with SAS® Enterprise Guide®

Display 24. The Average Height of All Students in SASHELP.CLASS



Display 25. Creating a Query-Based Template That Calculates Average Height

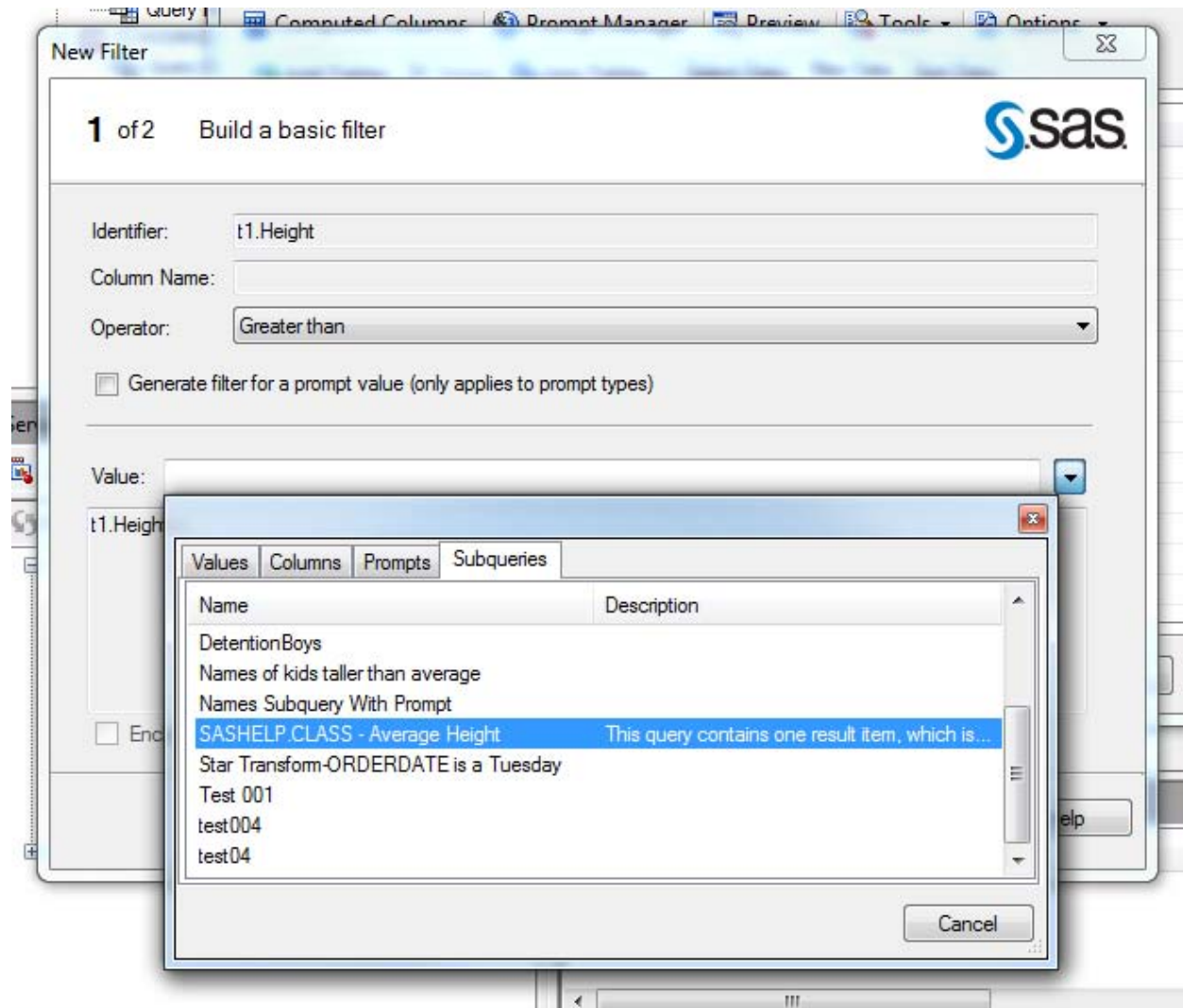
Finding Your Inner Query with SAS® Enterprise Guide®



Display 26. Specifying a Name for the New Query-Based Template

After you save the template, you can access it again in a new Query Builder node that references the SASHelp.Class data set. To find out who is taller than the average child in the class, you select the Name and the Height columns in the **Select Data** tab of the Query Builder, and then create a basic filter based on the Height column. After selecting the **Greater than** operator, click the button next to the **Value** field to see the **Subqueries** tab. This tab lists all saved query-based templates on the client machine that have a single result item in the query. Only these queries are eligible to be used as a subquery in a Query Builder filter.

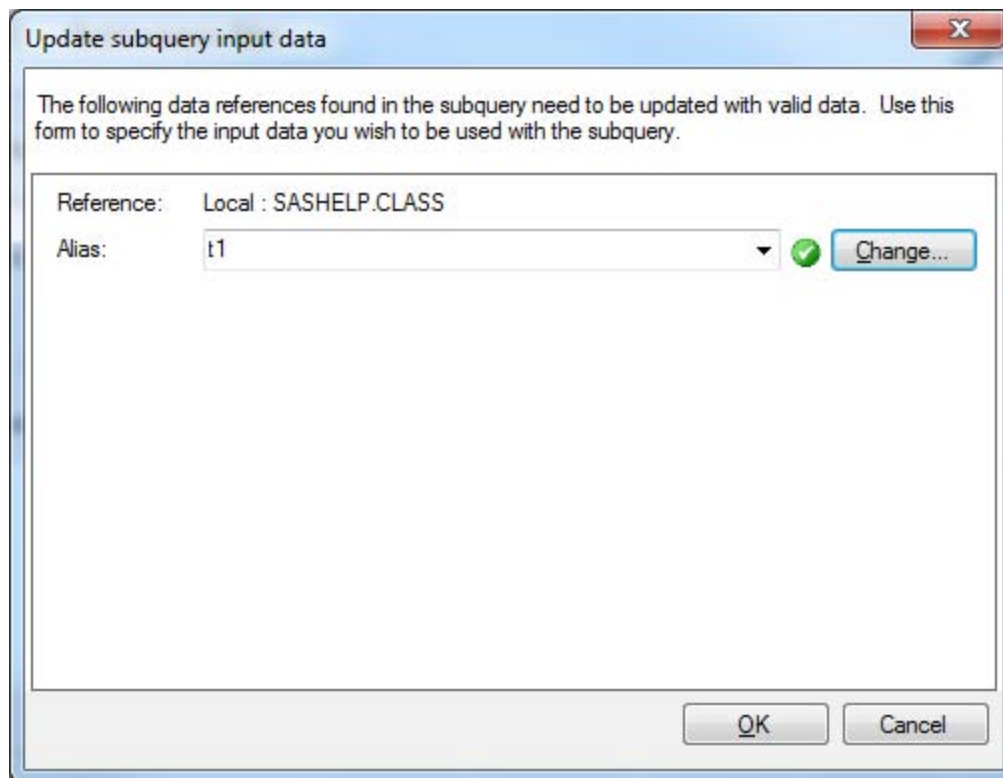
Finding Your Inner Query with SAS® Enterprise Guide®



Display 27. Reusing the Previously Saved Query-Based Template as a Subquery in a Filter

After you select the subquery, you can change its reference data source and alias in a dialog box similar to the one that you saw when you opened a query-based template. In this case, you can just click **OK** to add the subquery to the Query Builder filter.

Finding Your Inner Query with SAS® Enterprise Guide®



Display 28. Updating the Subquery Input Data

Finding Your Inner Query with SAS® Enterprise Guide®

New Filter

1 of 2 Build a basic filter

Identifier: t1.Height

Column Name:

Operator: Greater than

Subquery generated from template: SASHELP.CLASS - Average Height

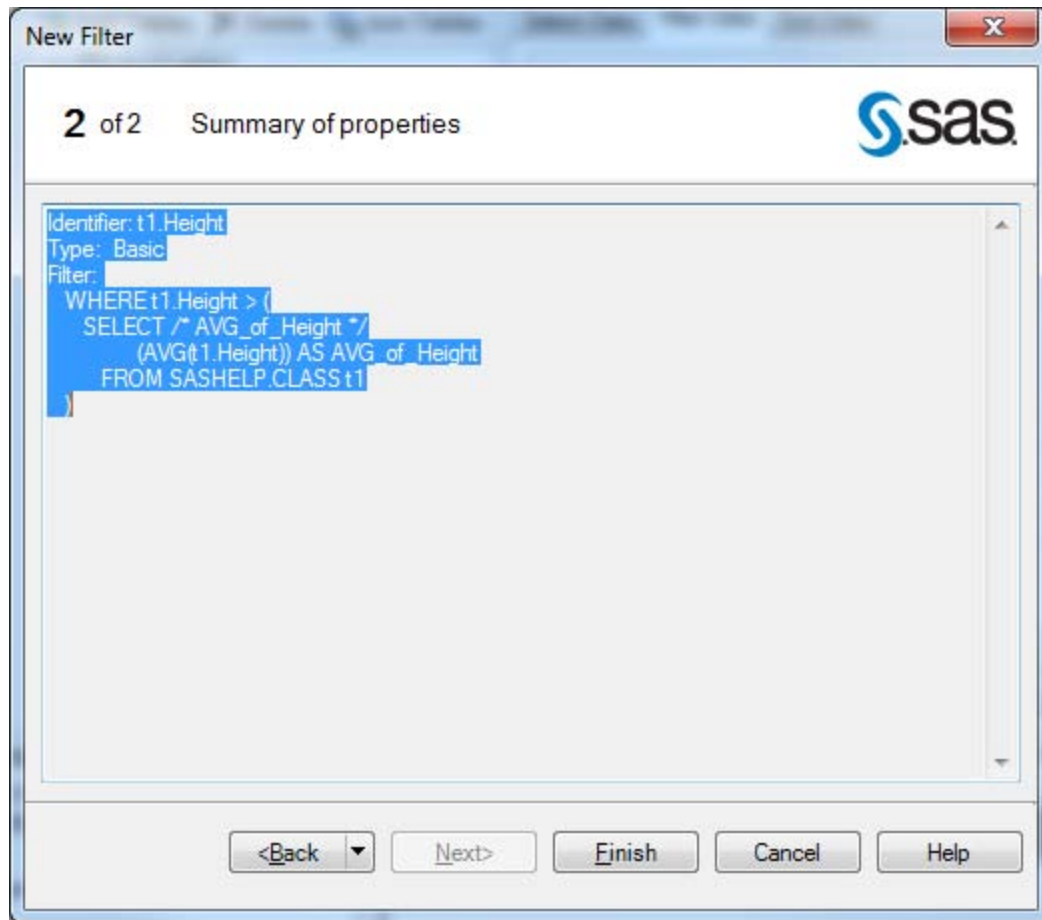
```
(
  SELECT /* AVG_of_Height */
    (AVG(t1.Height)) AS AVG_of_Height
  FROM SASHELP.CLASS t1
)
```

<Back Next> Finish Cancel Help

Display 29. The Query-Based Template Is Now a Subquery in a Filter

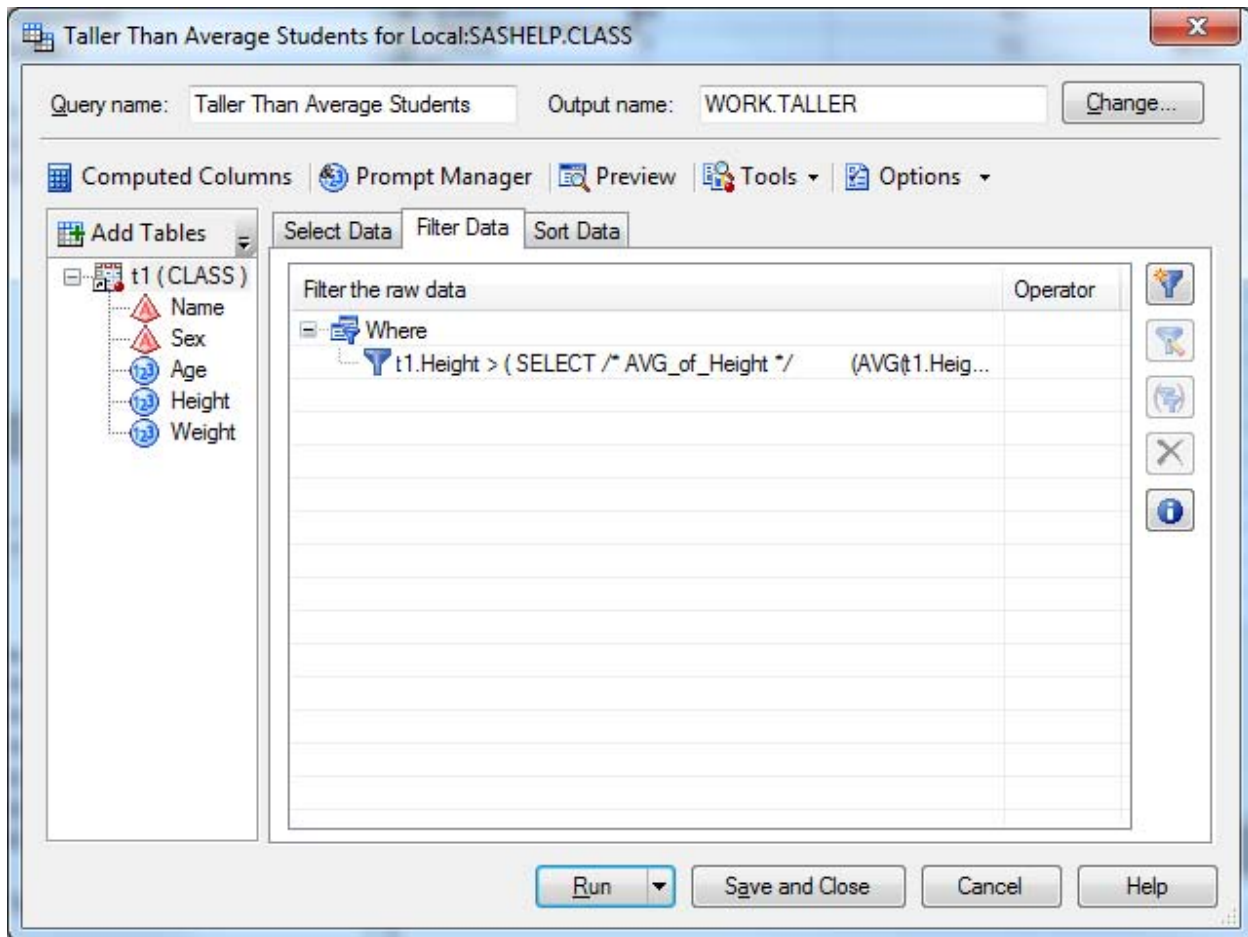
The portion of the query-based template that describes the entire SELECT statement is added to the Query Builder. From this point on, it becomes a subquery and an expression in the query filter. You can manually edit the expression from inside the filter. However, replacing the template used to import the subquery will not propagate changes to existing query nodes containing subqueries based on the original. In this example, you are working with filtering the raw data (FROM clause) but this process works just the same for filtering summarized data (HAVING clause).

Finding Your Inner Query with SAS® Enterprise Guide®



Display 30. Summary of Properties for the new Subquery

Finding Your Inner Query with SAS® Enterprise Guide®



Display 31. The Subquery as it Appears in the Filter Data Tab

Running this Query Builder node now produces a list of student names along with height. These students all are taller than the class average of 62.34.

	Name	Height
1	Alfred	69
2	Barbara	65.3
3	Carol	62.8
4	Henry	63.5
5	Janet	62.5
6	Jeffrey	62.5
7	Judy	64.3
8	Mary	66.5
9	Philip	72
10	Robert	64.8
11	Ronald	67
12	William	66.5

Display 32. Students Taller than the Average Height of 62.34

The SQL generated for this query shows the subquery wrapped in parentheses inside the WHERE clause:

```
PROC SQL;
    CREATE TABLE WORK.TALLER (label="QUERY_FOR_CLASS") AS
```

Finding Your Inner Query with SAS® Enterprise Guide®

```

SELECT t1.Name,
       t1.Height
FROM SASHELP.CLASS t1
WHERE t1.Height > (
    SELECT /* AVG_of_Height */
          (AVG(t1.Height)) AS AVG_of_Height
    FROM SASHELP.CLASS t1
);
QUIT;

```

This use of subqueries answers a unique type of problem. You can imagine this type of query being used to answer questions like the following listed below. Beside each question is the SASHELP data set that you can use to explore for the answer.

- Where has there been recorded higher than average actual sales? (SASHELP.PrdSal3, using the Actual column)
- Where have the predicted sales been more accurate than average? (SASHELP.PrdSal3, by comparing the Actual and Predicted columns)
- Out of the people who had a higher than average cholesterol and who had high blood pressure at the time of data collection, how many people were also heavy smokers? (SASHELP.Heart, using the Cholesterol, BP_Status, and Smoking_Status columns)

To answer these questions without subqueries, you might need to determine the value of the summary function and save it off to a separate table; merge the result back to match every row in the main table in question; and then create a filter that compares the individual row values to the aggregate that has now been duplicated on every row in the main table. A subquery in this case offers a more elegant solution.

In One Table but Not Another

Another interesting problem subqueries can help with is when you need to select records from one table that cannot be matched to records in another table. For example, consider a case where all the twelve-year-olds in the SASHELP.Class data set are acting up and have their names placed in a Detention data set. You could run the following code in a program node to create this data, or you could use a simple Query Builder node to create the data:

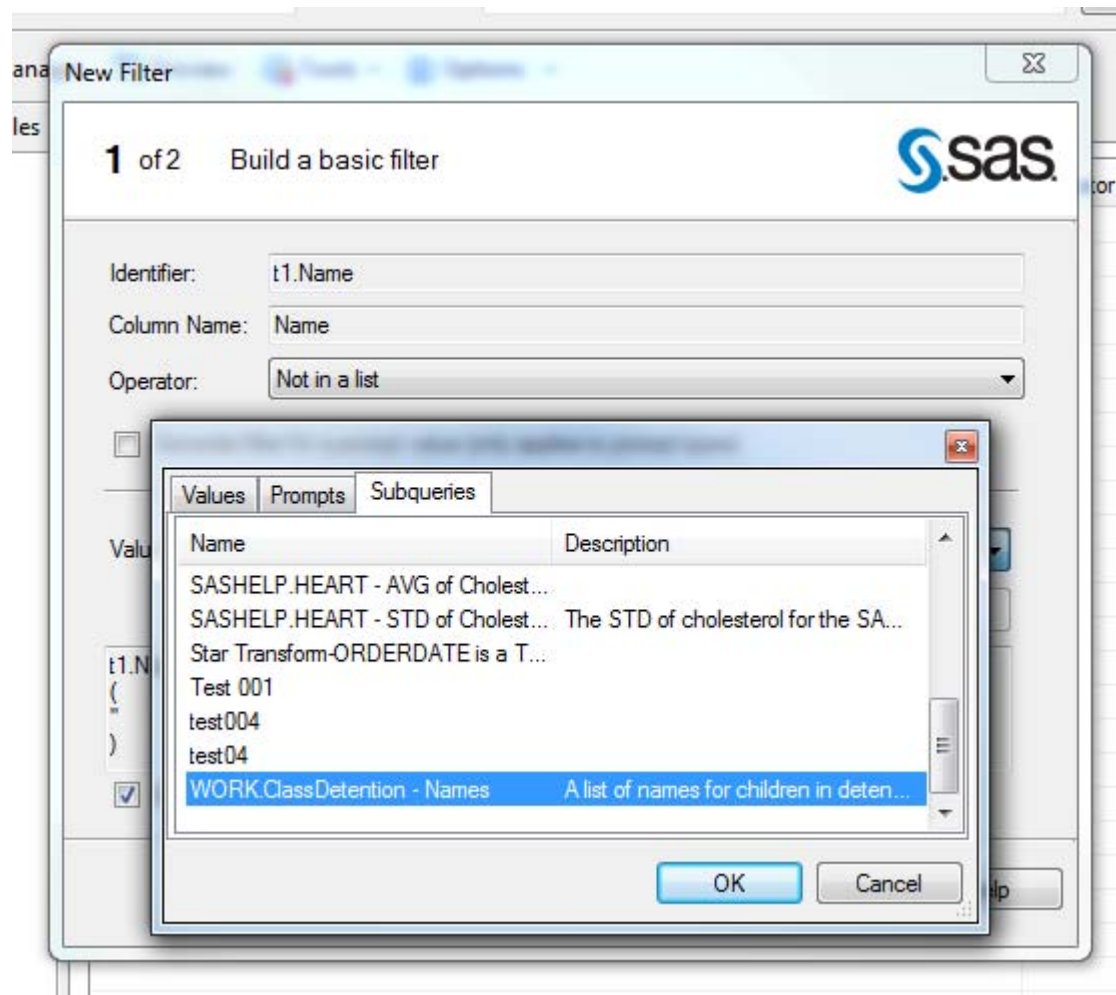
```

proc sql;
    create table WORK.ClassDetention as
    select t1.Name
    from sashelp.class t1
    where t1.Age = 12;
quit;

```

Supposing you do not know that it is the twelve-year-olds who are acting up. Instead, you want to answer the question, “Who in the SASHELP.Class data set was not put into detention?”. You can use a subquery to answer this question. The first step is to create a query-based template from a simple Query Builder node that uses the WORK.ClassDetention. In the query, no filter is applied and you select only the Name column. After that template is created, another Query Builder that runs against the SASHELP.Class data set can use this template as a subquery to determine which students are not in the Detention data set.



Finding Your Inner Query with SAS® Enterprise Guide®



Display 33. Using a Subquery That Returns More than One Row

In this example, the subquery returns more than one row. The result set includes all the names in the WORK.ClassDetention data set. To use this as part of a Query Builder filter, you need to use either the **In a list** or **Not in a list** operator. Because in this case you want to find out whom in the SASHELP.Class data set is not in the list of names in the WORK.ClassDetention data set, the operator to use is **Not in a list**. When you run the resulting query, the result shows the fourteen students do not have their name listed in the WORK.ClassDetention data set.

Finding Your Inner Query with SAS® Enterprise Guide®

	 Name 	Age
1	Alfred	14
2	Alice	13
3	Barbara	13
4	Carol	14
5	Henry	14
6	Janet	15
7	Jeffrey	13
8	Joyce	11
9	Judy	14
10	Mary	15
11	Philip	16
12	Ronald	15
13	Thomas	11
14	William	15

Display 34. The Good Kids

The resulting SAS code generated from this Query Builder shows the subquery nested inside of parentheses inside the WHERE clause. Again, this process would work the same way if the need was to filter on the summarized data (using a HAVING clause):

```
PROC SQL;

    CREATE TABLE WORK.NOTINDETENTION(label="QUERY_FOR_CLASS") AS

    SELECT t1.Name,
           t1.Age
    FROM SASHELP.CLASS t1
    WHERE t1.Name NOT IN
        (
            SELECT t1.Name
            FROM WORK.CLASSDETENTION t1
        )
    ;

QUIT;
```

Things to Remember

When you are working with a query-based template as a subquery that returns a single value, you can use the following filter operators:

- Equal to
- Not equal to
- Less than
- Less than or equal to
- Greater than
- Greater than or equal to

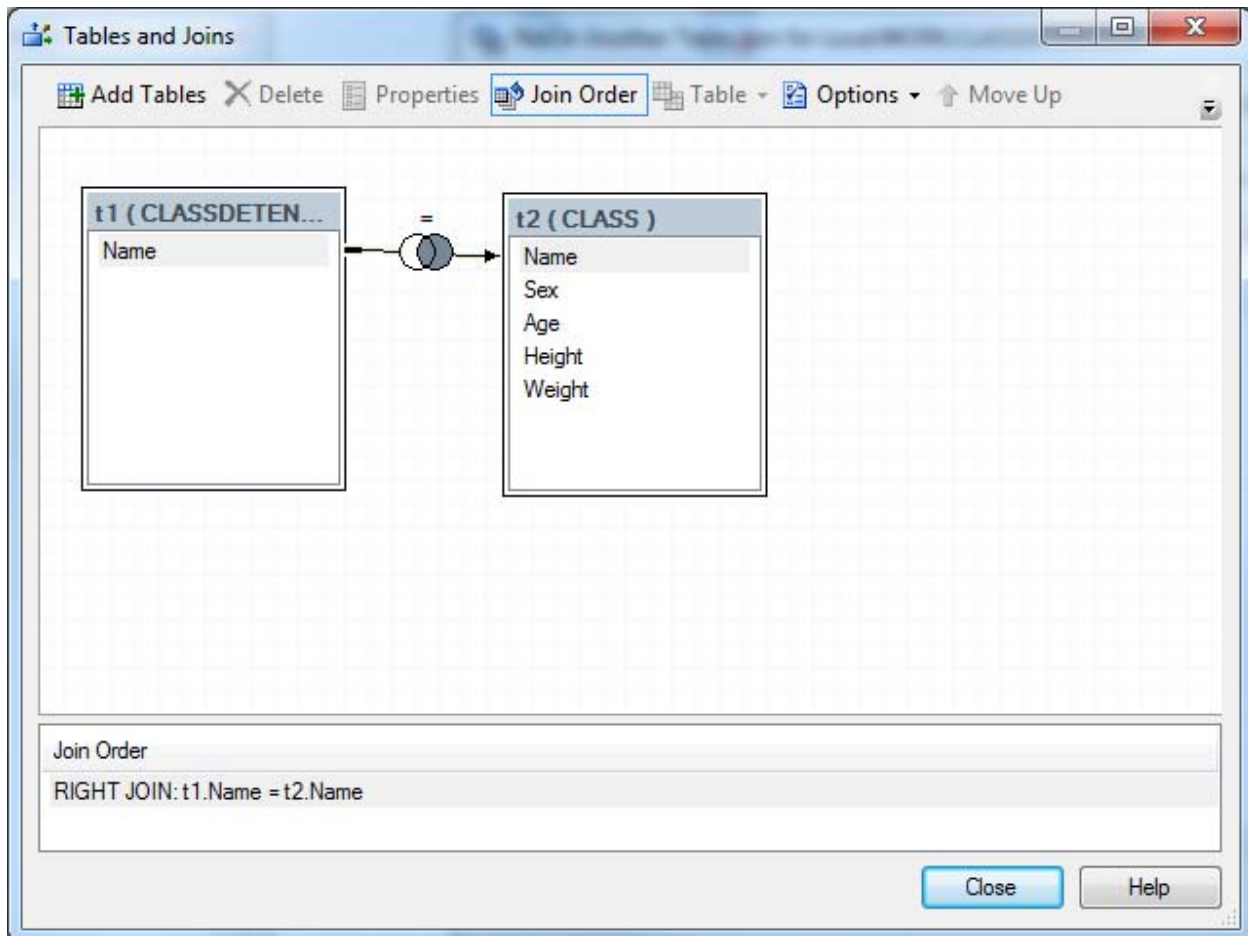
When you are working with a query-based template as a subquery that returns multiple rows for a single column, you can use the following filter operators:

Finding Your Inner Query with SAS® Enterprise Guide®

- In a list
- Not in a list

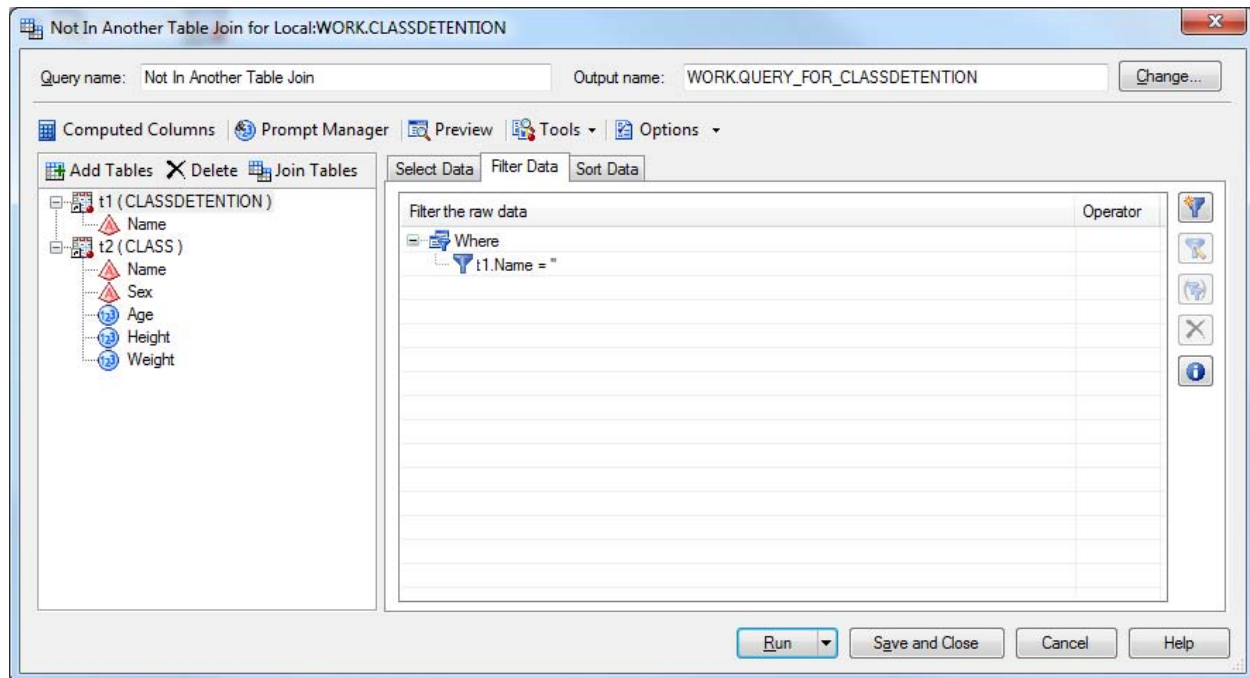
Subqueries and Joins

When working with subqueries that return multiple values, in many cases the resulting query could be redesigned to accomplish the same result by using a join in combination with a filter. If you consider the problem where you want to find rows in one table that are not in another table, the same thing could be achieved by joining the WORK.ClassDetention table with the SASHELP.Class table using a RIGHT JOIN and then including a filter to include a condition where a column from the WORK.ClassDetention table was null.



Display 35. Using a Join to Find Rows in One Table That Are Not in Another Table

Finding Your Inner Query with SAS® Enterprise Guide®



Display 36. Using a Filter to Weed Out the Records from One Table in a Query

The resulting code generated from this query returns the same fourteen 'good' kids that the earlier subquery example did. Here is the generated code:

```
PROC SQL;

    CREATE TABLE WORK.QUERY_FOR_CLASSDETENTION AS

    SELECT t2.Name,
           t2.Age,
    FROM WORK.CLASSDETENTION t1
           RIGHT JOIN SASHELP.CLASS t2 ON (t1.Name = t2.Name)
    WHERE t1.Name = '';

QUIT;
```

When working with subqueries that return multiple values and use the **In a list** operator, you often are performing the same thing as a query that contains an INNER JOIN. There might be some cases where a subquery is more efficient in retrieving data. The best choice depends on the options being used by the back-end database, the properties of the tables involved, and the query itself. However, generally speaking a join is probably going to be as efficient or more efficient for this purpose.

There are nevertheless a couple of compelling reasons to use subqueries for join-like behavior in some Query Builder filters in SAS Enterprise Guide 5.1. The first reason you might want to use subqueries is increased readability. In the previous example when comparing the Query Builder node using a subquery to the one using a RIGHT JOIN, it is easier to understand what is happening in the case of the subquery. The structure of the query seems to match how you would logically think about the problem -- first selecting from the target table you have in mind and then adding a condition that goes and checks against another table. The case of the join requires you to understand the query and its purpose as a whole before you can understand what is happening and why in the selected items, join, and filter condition.

A second reason to make use of subqueries in some cases for join-like behavior is maintainability. If you work in an established data environment where tables are organized into a knowable design like a star schema, you can create reusable query-based templates that target the key fields for tables you often work with. If you need columns in the output from multiple tables, you will always need joins. However, in cases where the complicated joins are needed

just for the sake of hunting for records in a single table, using query-based templates as ready-to-go subqueries might be easier and faster to work with from a development perspective.

The ability to quickly develop and easily use queries that perform join-like behavior through subqueries can be extended further by designing the subquery to include a filter that references a prompt. If a table includes a column representing a transaction date or perhaps a date for when the record was last updated, a subquery targeting the key for this table might also include a prompt in the filter for date. This prompt enables you to supply join information between tables at run time, which might prove very useful in ad hoc analyses on a set of large relational data tables.

Here is a simple example. Consider the earlier example table of WORK.ClassDetention. This table is comprised simply of a list of names and nothing else. If you wanted to query this table and pick out only the boys or only the girls and you wanted to make that choice at run time, then using a prompted subquery targeting the SASHELP.Class table inside a query against the WORK.ClassDetention table would let you do that. To look at this example, you first need to use the SAS Enterprise Guide prompt manager to create a text type prompt that would allow you to pick either 'F' (for girls) or 'M' (for boys) when run.

Add New Prompt

General | Prompt Type and Values

Prompt type:
Text

Method for populating prompt:
User selects values from a static list

Number of values:
Single value

Minimum length:
Maximum length:

Include Special Values
☐ All possible values ☐ Missing values

☐ Append formatted values with unformatted values

List of values:

Unformatted Value	Formatted (Displayed) Value	Default
F	F	<input type="radio"/>
M	M	<input type="radio"/>

Buttons: Add, Get Values..., Delete, Clear Default

Buttons: OK, Cancel, Help

Display 37. Setting Up a Prompt to Use with a Subquery

With the prompt created, the next step would be to create a reusable query-based template that would represent the 'join-like' behavior that you would like to make dynamic. In this case, it will be a query on the SASHELP.Class table with a filter on the Sex column that refers to the 'Sex' prompt we just created.

Finding Your Inner Query with SAS® Enterprise Guide®

New Filter

1 of 2 Build a basic filter

Identifier: t1.Sex

Column Name:

Operator: Equal to

☒ Generate filter for a prompt value (only applies to prompt types)

Value: &Sex

%_eg_WhereParam(t1.Sex, Sex, EQ, TYPE=S, IS_EXPLICIT=0)

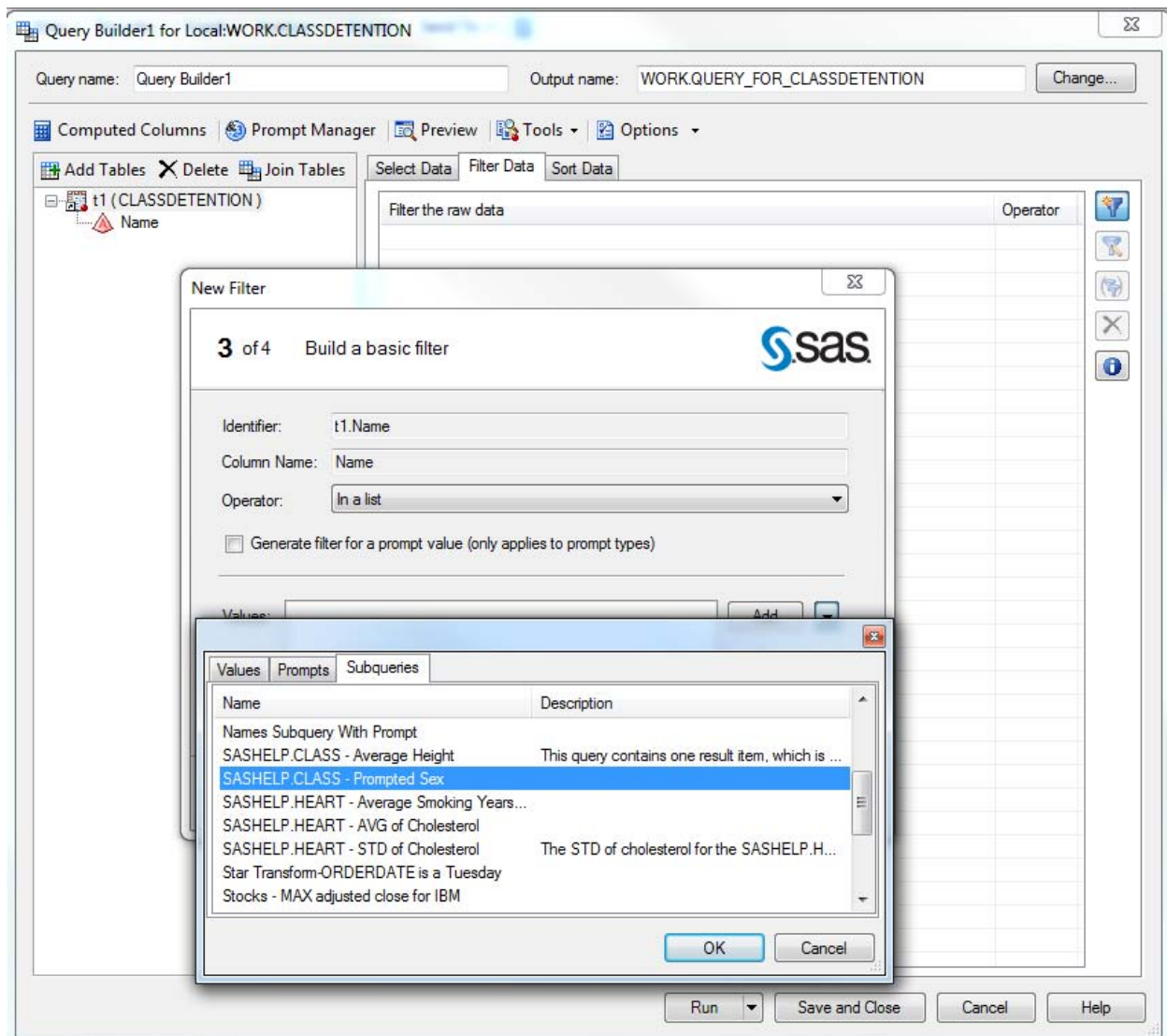
☐ Enclose values in quotes ☐ Use formatted dates

<Back Next> Finish Cancel Help

Display 38. The Filter Defined Here Will Drive the Dynamic 'Join'

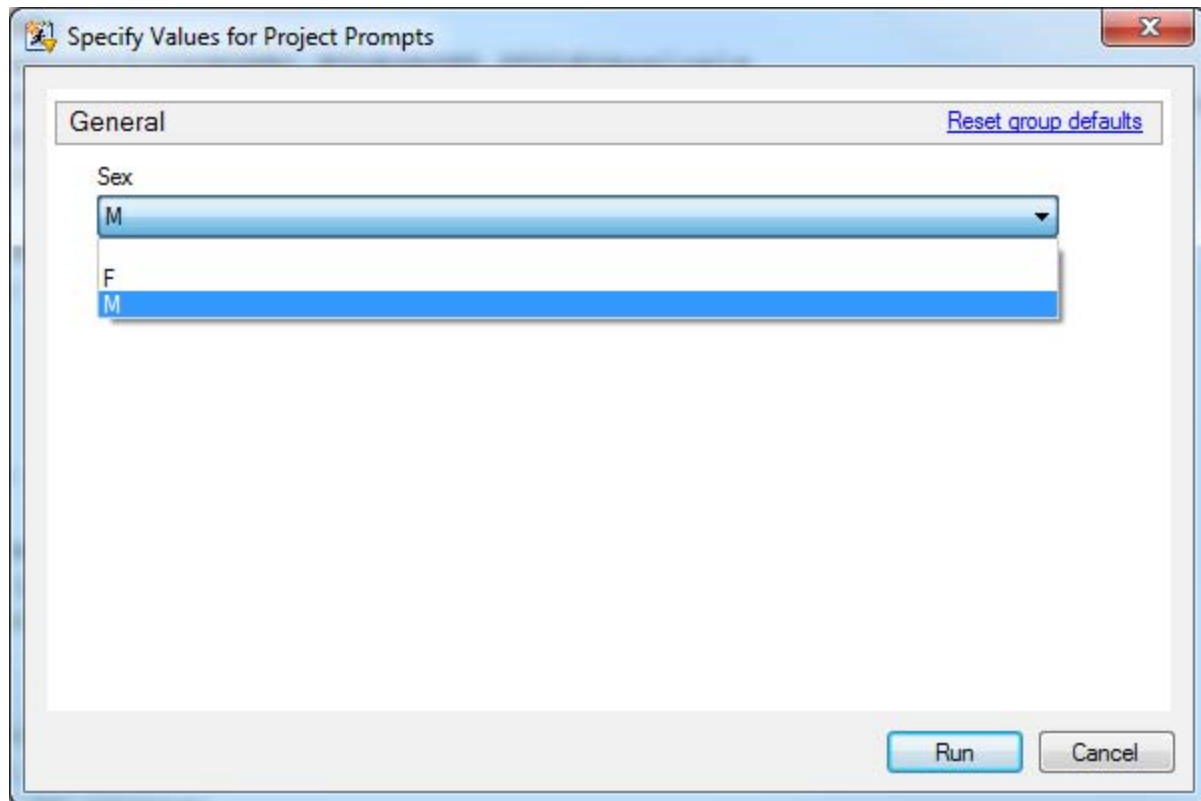
At this point, you can save the query as a template, and a new query (to come up with the final solution to this problem) can be started against the WORK.ClassDetention table. In this query, the Name value is added to the result list in the **Select Data** tab in the Query Builder, and a basic filter is initiated. In the filter, you choose the **In a list** operator and then import the query-based template that you just created for the SASHELP.Class table as a subquery. This subquery provides a list of 'keys' (which in this case are names) of either boys or girls depending on the value chosen in the prompt at run time.

Finding Your Inner Query with SAS® Enterprise Guide®



Display 39. Prompted Subqueries Used as a Dynamic Join

Finding Your Inner Query with SAS® Enterprise Guide®



Display 40. Providing the Prompt Value at Runtime

When the query is run, the value that you pick in the prompt is fed into the subquery and used to drive the logic in this 'join-like' example. Here is the code for this example:

```
PROC SQL;

    CREATE TABLE WORK.QUERY_FOR_CLASSDETENTION AS

    SELECT t1.Name,

    FROM WORK.CLASSDETENTION t1

    WHERE t1.Name IN

    (

        SELECT t1.Name

        FROM SASHELP.CLASS t1

        WHERE %_eg_WhereParam(t1.Sex, Sex, EQ, TYPE=S, IS_EXPLICIT=0)

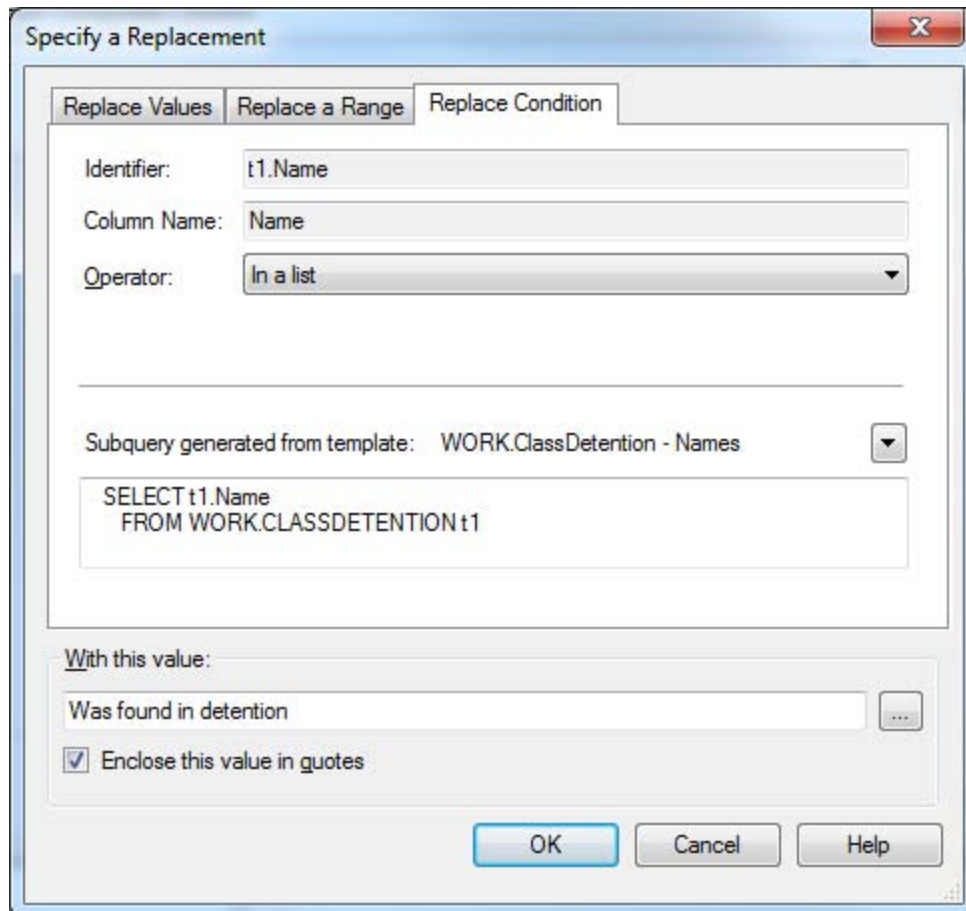
    );

QUIT;
```

RECODING COLUMNS USING SUBQUERIES

In SAS Enterprise Guide 5.1, subqueries enable you to add a dynamic element to how you can create a computed recoded column by using these subqueries as an expression in a replace condition. If you consider a query against the SASHELP.Class table, you can use this feature to create a new character column that will read 'Was found in detention' or 'Was NOT found in detention' based on what a subquery finds when run against the WORK.ClassDetention table. In the Specify a Replacement dialog box, the process for importing a reusable query-based template as a subquery functions the same as when creating a basic filter on the **Filter** tab of the Query Builder.

Finding Your Inner Query with SAS® Enterprise Guide®



The image shows a 'Specify a Replacement' dialog box with three tabs: 'Replace Values', 'Replace a Range', and 'Replace Condition'. The 'Replace Condition' tab is selected. The 'Identifier' field contains 't1.Name', the 'Column Name' field contains 'Name', and the 'Operator' dropdown is set to 'In a list'. Below these fields, a text box displays the subquery: 'SELECT t1.Name FROM WORK.CLASSDETENTION t1'. At the bottom, the 'With this value:' section contains the text 'Was found in detention' and a checked checkbox labeled 'Enclose this value in quotes'. The dialog has 'OK', 'Cancel', and 'Help' buttons at the bottom right.

Specify a Replacement

Replace Values Replace a Range Replace Condition

Identifier: t1.Name

Column Name: Name

Operator: In a list

Subquery generated from template: WORK.ClassDetention - Names

SELECT t1.Name
FROM WORK.CLASSDETENTION t1

With this value:

Was found in detention

☒ Enclose this value in quotes

OK Cancel Help

Display 41. Using a Subquery as a Replace Condition for a Recoded Column

Finding Your Inner Query with SAS® Enterprise Guide®

Edit Computed Column

1 of 2 Specify a replacement

Replacement

Replace	With
IN (SELECT t1.Na...	'Was found in detention'

Add... Edit... Delete

Other values

Replace all other values with:

☐ The current value
☐ A missing value
☒ Specify a value:

Was NOT found in detention

☒ Enclose value in quotes

Column type

☒ Character
☐ Numeric

<Back Next> Finish Cancel Help

Display 43. Specifying the Replacement Value Used when the Subquery Condition is Matched

	Name	Age	Detention
1	Alfred	14	Was NOT found in detention
2	Alice	13	Was NOT found in detention
3	Barbara	13	Was NOT found in detention
4	Carol	14	Was NOT found in detention
5	Henry	14	Was NOT found in detention
6	James	12	Was found in detention
7	Jane	12	Was found in detention
8	Janet	15	Was NOT found in detention
9	Jeffrey	13	Was NOT found in detention
10	John	12	Was found in detention
11	Joyce	11	Was NOT found in detention
12	Judy	14	Was NOT found in detention
13	Louise	12	Was found in detention
14	Mary	15	Was NOT found in detention
15	Philip	16	Was NOT found in detention
16	Robert	12	Was found in detention
17	Ronald	15	Was NOT found in detention
18	Thomas	11	Was NOT found in detention
19	William	15	Was NOT found in detention

Display 44. Results from Using a Subquery in a Recoded Column

The resulting code shows the subquery nested inside the CASE function in the SELECT statement:

```
PROC SQL;

    CREATE TABLE WORK.QUERY_FOR_CLASS AS
    SELECT t1.Name,
           t1.Age,
           /* Detention */
           (CASE
            WHEN t1.Name IN
            (
                SELECT t1.Name
                FROM WORK.CLASSDETENTION t1
            ) THEN 'Was found in detention'
            ELSE 'Was NOT found in detention'
           END) LENGTH=50 AS Detention
    FROM SASHELP.CLASS t1;

QUIT;
```

CONCLUSION

The purpose of this paper was to explore the new features of reusable query-based templates and subqueries in the SAS Enterprise Guide 5.1 Query Builder, to understand how these templates are used, and why you might want to use them.

Reusable query-based templates are offered in SAS Enterprise Guide 5.1 as an extension to the existing task template feature. Query-based templates enable you to save time-consuming work that you have done in the Query Builder designing queries. These templates also enable you to quickly add those queries to a process flow at a later time and possibly with different data. Combined with the ability to import and export templates through the Template Manager, query-based templates can increase the ease of collaboration between colleagues in an organization where some individuals might specialize in data management and others might focus more on analysis or reporting.

The SAS Enterprise Guide 5.1 Query Builder enables you to reuse query-based templates with a single result item as subqueries. As a result, you can solve special problems as well as facilitate logical query design and ease of development. Using a subquery inside a recoded computed column opens up the possibility for dynamic recoding conditions where the logic for recoding information in one table is controlled by values in another. Using a subquery that returns a single value in a filter can let you do things with your query that would be difficult to do any other way, such as filtering a table based on subsets or aggregated values from the same or another table. Subqueries also enable you to perform filtering operations in substitution for a join. Depending on your data, using subqueries might not be more efficient from a performance standpoint, but in some cases, it does allow for a more elegant query design that is easier to design, understand, and maintain. Finally, combining that type of subquery with a prompt allows for join criteria to be driven by user choices at run time. This might facilitate ad hoc analyses in a data environment where you are constantly hunting for subsets of records in a single table driven by certain date ranges or other criteria stored in another table.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors:

Michael Burke

Finding Your Inner Query with SAS® Enterprise Guide®

SAS Campus Drive
SAS Institute Inc.

E-mail: Michael.Burke@sas.com

I-kong Fu
SAS Campus Drive
SAS Institute Inc.

E-mail: I-kong.Fu@sas.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.