TERADATA CORPORATION

Teradata ADS Generator & Fuzzy Logix DB Lytix™ Integration Guide

Teradata Partner Integration Lab

8/29/2014

Teradata ADS Generator, a product within the Teradata Warehouse Miner suite, can provide a front end for Fuzzy Logix DB Lytix™ in-database library of advanced analytic components. This document provides a tutorial on how to call and manage the different types of DB Lytix™ components from ADS Generator.

Table of Contents

1. Introduction

Teradata ADS Generator, part of the Teradata Warehouse Miner family of products, was built to support both comprehensive data profiling as well as analytic data generation for Teradata customers. Neither the data profiling nor the analytic data set generation capabilities of the product require any movement of data outside of the warehouse and utilize as much of the data as the analyst or data scientist requires. Results and metadata are stored directly in the database while utilizing the parallel, scalable processing power of the Teradata platforms to perform data intensive operations.

DB Lytix™, a product developed by Fuzzy Logix, offers scalable and robust high performance analytical methods that are embedded seamlessly into database systems. The DB Lytix™ library of statistical, machine learning, and quantitative methods provide Teradata customers a rich set of in-database components which fall into one of the following categories:

1) Mathematical Functions
2) Matrix Operations
3) Basic/Sparse Statistics
4) Date Functions
5) Fit Distributions
6) String Functions
7) Hypothesis Testing
8) Data Mining
9) Sim Univariate/Copula
10) Sampling Techniques
11) PDF/CDF/Inv CDF
12) Time Series Functions

Although there is some overlap in functionality between Teradata ADS Generator and DB Lytix™, the combination of the two create an extremely powerful in-database predictive analytics platform providing data profiling, data preparation, machine learning, multivariate statistics and predictive model deployment capabilities for Teradata customers.

Recent changes made by the Teradata Applications Engineering group to the Teradata Warehouse Miner family of products, allows the Teradata ADS Generator front-end to call the DB Lytix™ database objects, and manage them for production environments. Teradata Warehouse Miner or ADS Generator Version 05.03.05 is required for this integration. This document will describe the integration points between the two products and does not attempt to replicate the information in either the Teradata ADS Generator or Fuzzy Logix DB Lytix™ user documentation. For more information on the capabilities of Teradata ADS Generator, please refer to the following:

- Teradata Warehouse Miner User Guide - Volume 1 - Introduction and Profiling Release 5.3.5 (B035-2300-064A, June 2014)
- Teradata Warehouse Miner User Guide - Volume 2 - ADS Generation Release 5.3.5 (B035-2301-064A, June 2014)

For a thorough description of all the functions available in the Fuzzy Logix DB Lytix™ library, please refer to the following:

- User Manual for DB Lytix™ on Teradata Advanced Package v1.0.1

The Teradata ADS Generator & Fuzzy Logix DB Lytix™ Integration Guide is meant to provide a functional description on how each type of database object available within the Fuzzy Logix DB Lytix™ library is called from the Teradata ADS Generator User Interface. It also includes an example of one of the Fuzzy Logix Excel Templates that is available for visualizing the results generated by their database objects. A thorough description of all the Excel Templates in a use-case oriented format is available in the following companion document:

- Teradata ADS Generator - User Guide for Integrating DB Lytix™ and Excel (Aug 2014)

Note – As of this publication, additional integration features have been added to Teradata Warehouse Miner and ADS Generator version 5.3.5.1. These features are documented within the Help system available with the product, but not within the documents listed above.

1.1. Variable Creation Analysis

The Variable Creation analysis in Teradata ADS Generator makes possible the creation of variables as columns in a table or view. It is also the integration point with Fuzzy Logix' DB Lytix™. When using the Variable Creation analysis, the end-user creates each new variable as an expression by selecting various SQL keywords and operators as well as table/view and column names. Valid elements include:

1) Columns from one or more tables or views in one or more databases
2) Aggregation functions including MIN, MAX, SUM, AVG, COUNT, CORR, COVAR_POP/SAMP, STDDEV_POP/SAMP, VAR_POP/SAMP, SKEW, KURTOSIS, etc.
3) Ordered analytical functions including Windowed AVG, COUNT, MAX, MIN, and SUM, along with PERCENT_RANK, RANK, ROW NUMBER, etc.
4) Arithmetic operators including +, -, *, /, MOD, **
5) Arithmetic functions including ABS, EXP, LN, LOG, SQRT, RANDOM, WIDTH BUCKET, etc.
6) Trigonometric functions including COS, SIN, TAN, ACOS, ASIN, ATAN, ATAN2
7) Hyperbolic functions including COSH, SINH, TANH, ACOSH, ASINH, ATANH
8) CASE expressions, both valued and searched types
9) Comparison operators including =, >, <, <>, <=, >=
10) Logical predicates including BETWEEN, NOT BETWEEN, IN, NOT IN, IS NULL, IS NOT NULL, AND, OR, NOT, LIKE, NOT LIKE, ANY and ALL
11) Custom logical predicates AND ALL, OR ALL
12) NULL operators including NULLIF, COALESCE, NULLIFZERO, ZEROIFNULL
13) Built-in functions including CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP
14) Date/Time functions including ADD_MONTHS, EXTRACT, TEMPORAL_DATE, TEMPORAL_TIMESTAMP
15) Custom Date/Time differences and elapsed time functions
16) Calendar fields based on a specified date column with all Teradata Calendar options.
17) String functions including CHARACTER_LENGTH, Concatenate (||), LOWER, POSITION, SUBSTRING, TRIM, UPPER, etc.
18) Hash functions including HASHAMP, HASHBAKAMP, HASHBUCKET, and HASHROW
19) Miscellaneous other SQL constructs, including Asterisk(*), BYTES, CAST, Parentheses ( ), Sample ID, SQL element list, etc.
20) Formulas of 1 (x), 2 (x,y), 3 (x,y,z) or any number (x1…xn) of variables
21) Free SQL Text Entry with optional arguments
22) Subqueries
23) User Defined Functions
24) User Defined Methods
25) References to other variables
26) Embedded Services User Defined Functions (more than 100)
27) Geospatial Methods
28) Geospatial System Functions
29) Table Functions
30) Table Operators

31) Run-Units

The Variable Creation function also allows the creation of various expert clauses, some with specialized elements, including the following:

1) WHERE, HAVING and QUALIFY clauses
2) GROUP BY clause, including CUBE, ROLLUP and GROUPING SETS
3) ORDER BY clause
4) SAMPLE clause
5) TOP clause
6) WITH Recursive/Seed Query clauses

Any number of variables can be defined in a single Variable Creation function, provided they conform to rules that allow them to be combined in the same table, and they do not exceed the maximum number of columns allowed in a table by Teradata. Several variable properties are used in determining which variables can be built in the same table. Some rules of combining variables in the same Variable Creation function are given below.

- Variables derived in a single table must have the same aggregation type and level.
- A number of tables may be referenced by the variables defined in a single Variable Creation function.
- Variables referenced by another variable must not be dimensioned.
- All the variables in a Variable Creation function share the same table level constraints.
- The user may request at any time that the intermediate table created by a Variable Creation function be validated using the Teradata EXPLAIN feature.

The standard result options are available with the Variable Creation function, namely SELECT, EXPLAIN SELECT, CREATE TABLE and CREATE VIEW. The choice depends primarily on whether this analysis produces a final result or an intermediate result, and if so, whether the user wants to create a permanent table or view for this intermediate result. If a permanent result is not desired, the Select option can be used to view and verify results.

Within the ADS Variable Creation Analysis, the DB Lytix™ library of UDF and XSP objects can serve as an input variable, a Table input to the FROM clause or through a Run Unit as a Stored Procedure CALL, or input query / subquery.

1.2. Fuzzy Logix DB Lytix™

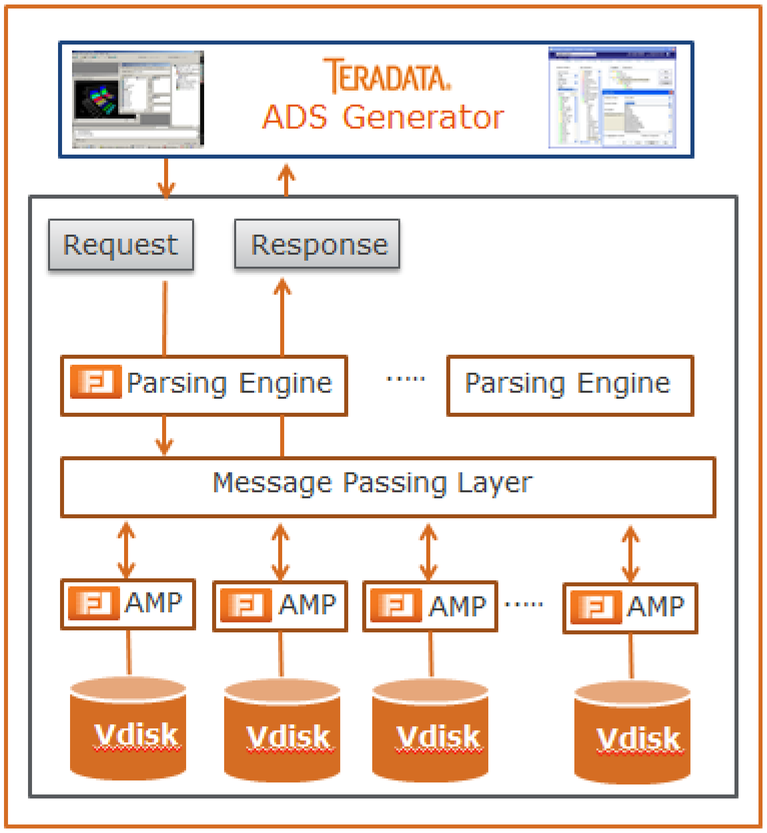
The Fuzzy Logix DB Lytix™ library of advanced analytic functions are embedded into the Teradata database as a set of database objects in the form of User Defined Functions (UDF's) and External Stored Procedures (XSP's) as follows:

1) Scalar UDF's - for those analytics that require a single result / single row as an answer set.
2) Aggregate UDF's - for those analytics that require single result of multiple aggregated rows as an answer set.
3) Table UDF's - for those analytics that require a table of results as an answer set.
4) XSP's - for those analytics that create multiple results and/or iterate over the input data multiple times. The XSP controls the iteration of the SQL executed, often with DB Lytix™ embedded UDF's in it.

DB Lytix™ comes in a Basic and Advanced package, with Basic being a subset of Advanced. The DB Lytix™ Advanced Package consists of hundreds of advanced functions across the following categories and can be deployed on Teradata 13.10, Teradata 14.00, and Teradata 14.10:

Category	**Functions**
Cumulative Distribution Functions	40
Data Mining Functions	66
Date Functions	26
Fit Distributions	78
Hypothesis Testing Functions	33
Inverse Cumulative Distribution Functions	40
Mathematical Functions	23
Matrix Operation Functions	20
Probability Density Functions	40
Sampling Techniques Functions	5
Simulate Copula Functions	7
Simulate Univariate Functions	40
Sparse Statistics Functions	11
Statistical Functions	50
String Functions	14
Time Series Functions	6
Total	**499**

The combined ADS Generator/DB Lytix™ integrated architecture is depicted below:



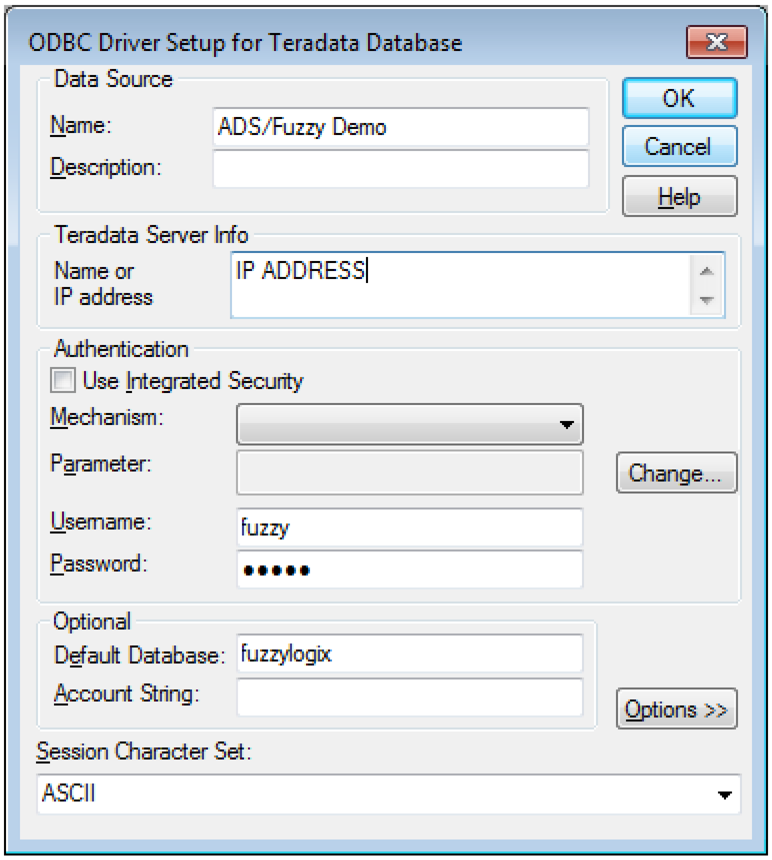
Since there are almost 500 (and counting!) DB Lytix™ functions within their library, it is not possible to illustrate how each would be called and managed from Teradata ADS Generator. What follows is a tutorial on how to call each type of DB Lytix™ function, and how to use the Teradata ADS Generator's Variable Creation Analysis to manage their execution and interrogate the results of their in-database analysis.

1.3. Importing the ADS/Fuzzy Logix Tutorial

For simplicity, this tutorial utilizes the well-known TWM Demonstration Data that is released with each version of Teradata ADS Generator. Attached below is the supporting ADS Project ".bin" file that can be imported into your Teradata ADS Generator environment:

Teradata ADS Demo Powered by Fuzzy Logix.bin

In order to import it, you must first create a user in your Teradata database named "fuzzy" with a subordinate database named "fuzzylogix." Then set up an ODBC DSN with the following properties:


ODBC Driver Setup for Teradata Database

Data Source
Name: ADS/Fuzzy Demo
Description:

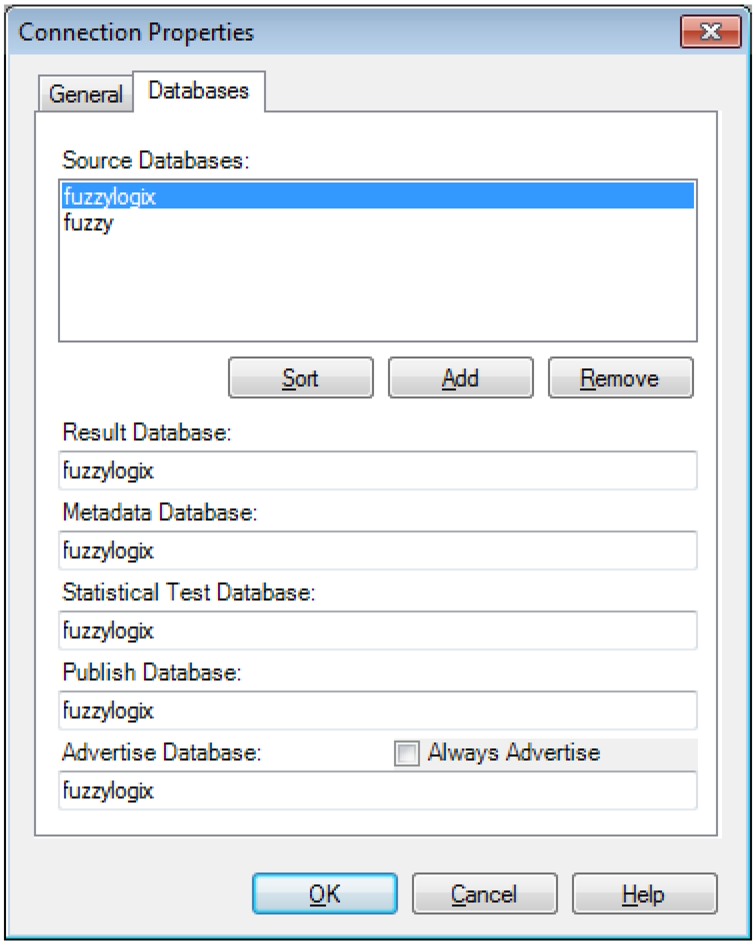
Teradata Server Info
Name or IP address: IP ADDRESS

Authentication
Use Integrated Security [ ]
Mechanism:
Parameter:
Username: fuzzy
Password: •••••

Optional
Default Database: fuzzylogix
Account String:

Session Character Set: ASCII

Connect the ADS front end to this DSN, and change your connection properties as follows:


Connection Properties

General | Databases

Source Databases:
- fuzzylogix
- fuzzy

Sort | Add | Remove

Result Database: fuzzylogix

Metadata Database: fuzzylogix

Statistical Test Database: fuzzylogix

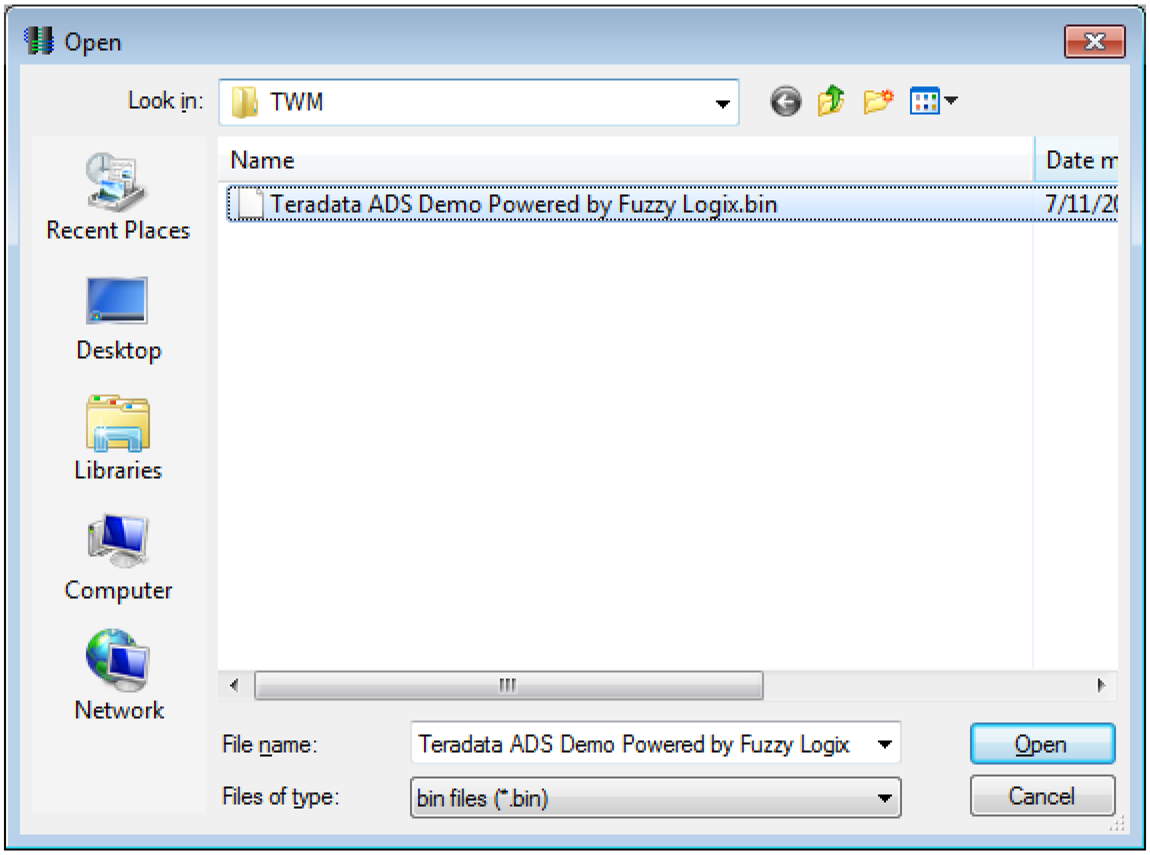
Publish Database: fuzzylogix

Advertise Database: fuzzylogix

[ ] Always Advertise

OK | Cancel | Help

Then execute the Import Wizard by selecting "Import…" from the File Menu:

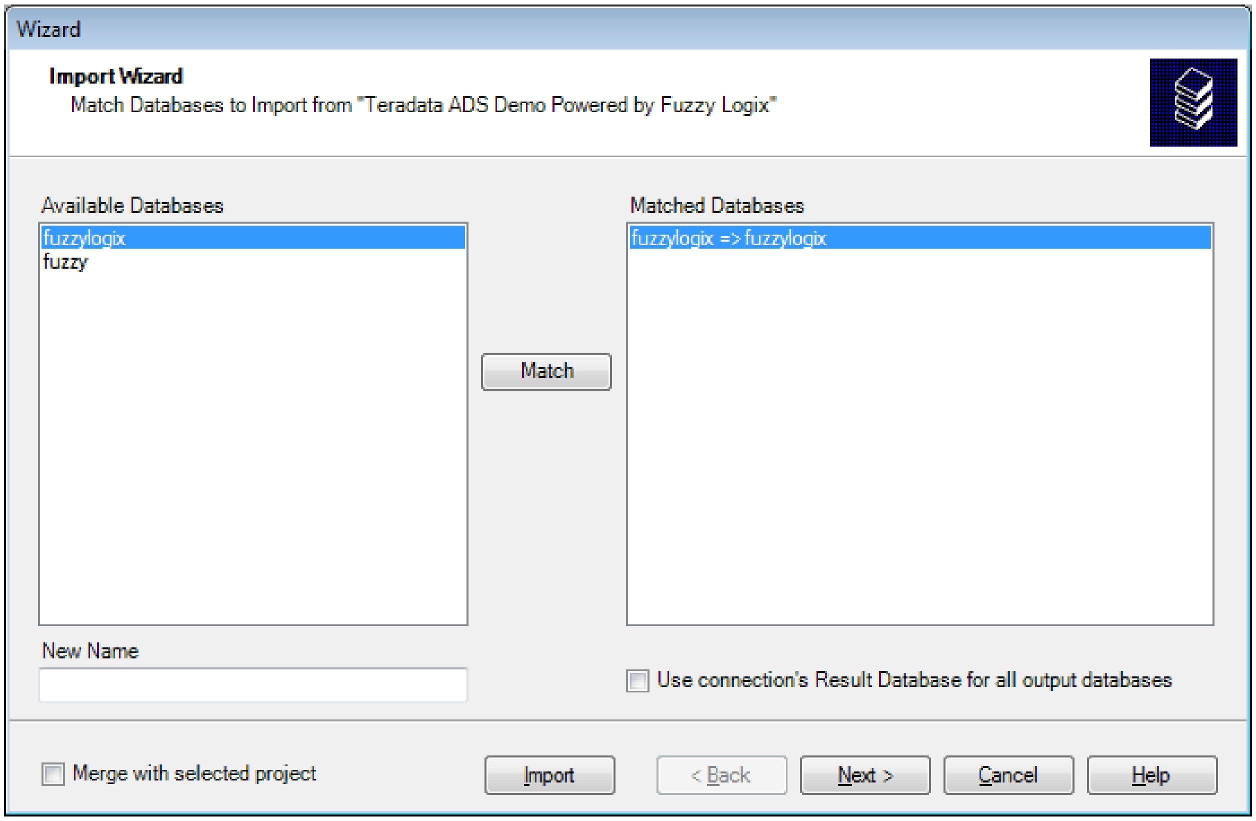


Teradata ADS Demo Powered by Fuzzy Logix.bin 7/11/2

File name: Teradata ADS Demo Powered by Fuzzy Logix

Files of type: bin files (*.bin)

Select the Teradata ADS Demo Powered by Fuzzy Logix bin file, and click Open:


Wizard

Import Wizard

Match Databases to Import from "Teradata ADS Demo Powered by Fuzzy Logix"

Available Databases

fuzzylogix
fuzzy

Matched Databases

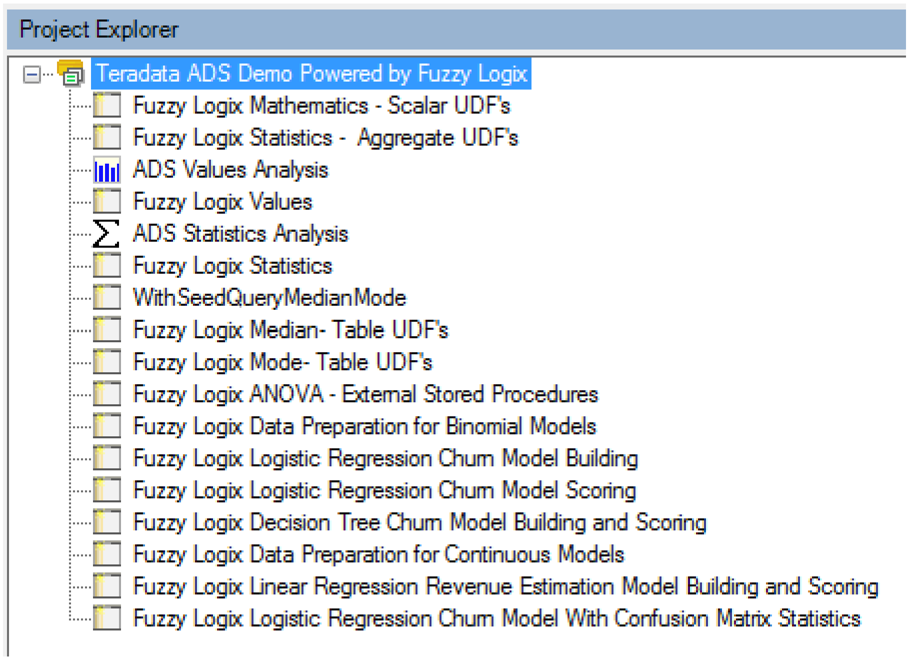
fuzzylogix => fuzzylogix

New Name

Use connection's Result Database for all output databases

Merge with selected project

If you have setup the DSN and the ADS Connection Properties as previously indicated, there is nothing more to do than to click on the "Import" button. The "Teradata ADS Demo Powered by Fuzzy Logix" Project should automatically come up as follows:


Project Explorer

- Teradata ADS Demo Powered by Fuzzy Logix
  - Fuzzy Logix Mathematics - Scalar UDF's
  - Fuzzy Logix Statistics - Aggregate UDF's
  - ADS Values Analysis
  - Fuzzy Logix Values
  - ADS Statistics Analysis
  - Fuzzy Logix Statistics
  - WithSeedQueryMedianMode
  - Fuzzy Logix Median- Table UDF's
  - Fuzzy Logix Mode- Table UDF's
  - Fuzzy Logix ANOVA - External Stored Procedures
  - Fuzzy Logix Data Preparation for Binomial Models
  - Fuzzy Logix Logistic Regression Churn Model Building
  - Fuzzy Logix Logistic Regression Churn Model Scoring
  - Fuzzy Logix Decision Tree Churn Model Building and Scoring
  - Fuzzy Logix Data Preparation for Continuous Models
  - Fuzzy Logix Linear Regression Revenue Estimation Model Building and Scoring
  - Fuzzy Logix Logistic Regression Churn Model With Confusion Matrix Statistics

And you are done!

NOTE – If you wish to attempt to map this tutorial to a different set of users and databases on Teradata, please see Chapter 3, Using Teradata Warehouse Miner in the Teradata Warehouse Miner User Guide - Volume 1 - Introduction and Profiling Release 5.3.5 document for instructions to do so. One word of warning however – we have observed very obscure errors, including syntax errors, when attempting to configure into a different environment. DB Lytix™ utilizes all of Teradata's extensibility features which have very complex security requirements.

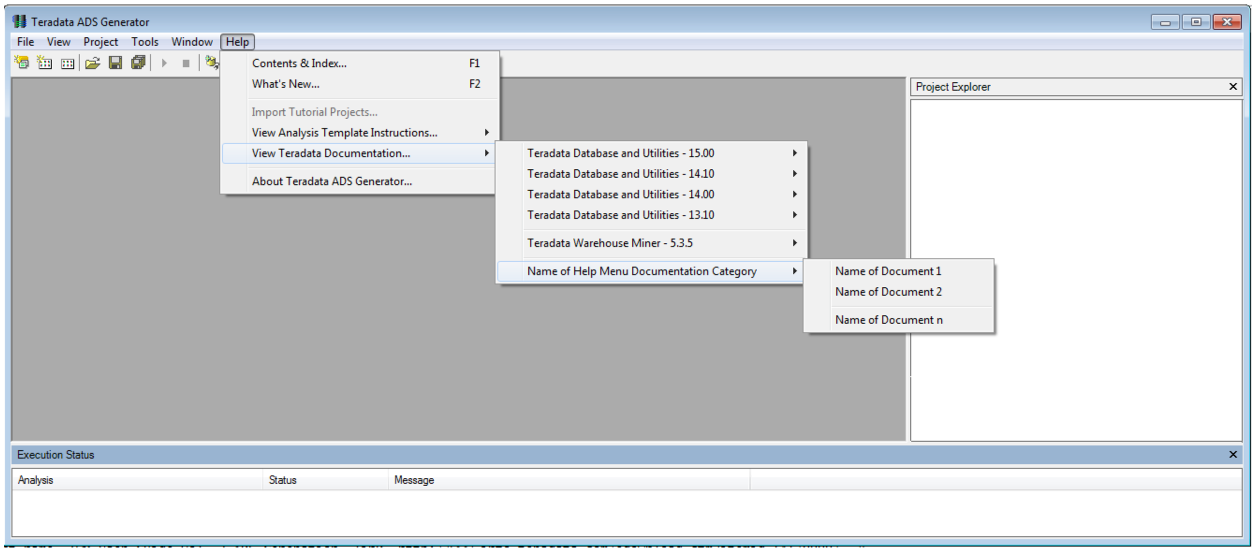
1.4. Adding Fuzzy Logix Documentation to the ADS Help Menu

For convenience, you can easily add documentation links to the Teradata Warehouse Miner or ADS Generator Help Menu. This way pertinent documentation is readily available from the user interface, without having to browse the web or your hard drive for it.

Upon installation, an XML file named "DocumentLinks.xml" is created within the Teradata Warehouse Miner or ADS Generator installation folder (by default "C:\Program Files\Teradata\Teradata ADS Generator 5.3.5" or "C:\Program Files\Teradata\Teradata Warehouse Miner 5.3.5"). The format of this file is very simple:

```
<DocumentLinks>
  <Product>
    <Product release=""/>
    <Product release="Name of Help Menu Documentation Category">
      <Document name="Name of Document 1" link="Fully qualified file location or URL" />
      <Document name="Name of Document 2" link="Fully qualified file location or URL" />
      <Document/>
      <Document name="Name of Document n" link="Fully qualified file location or URL" />
  </Product>
</DocumentLinks>
```

By copying the XML between the <DocumentLinks> and </DocumentLinks> tags from above and pasting it prior to the last line of XML (</DocumentLinks>) in the "DocumentsLinks.xml" file, you will see the structure captured in the screen below.


Teradata ADS Generator
File View Project Tools Window Help
Contents & Index... F1
What's New... F2
Import Tutorial Projects...
View Analysis Template Instructions...
View Teradata Documentation...
About Teradata ADS Generator...
Teradata Database and Utilities - 15.00
Teradata Database and Utilities - 14.10
Teradata Database and Utilities - 14.00
Teradata Database and Utilities - 13.10
Teradata Warehouse Miner - 5.3.5
Name of Help Menu Documentation Category
Name of Document 1
Name of Document 2
Name of Document n
Project Explorer
Execution Status
Analysis
Status
Message

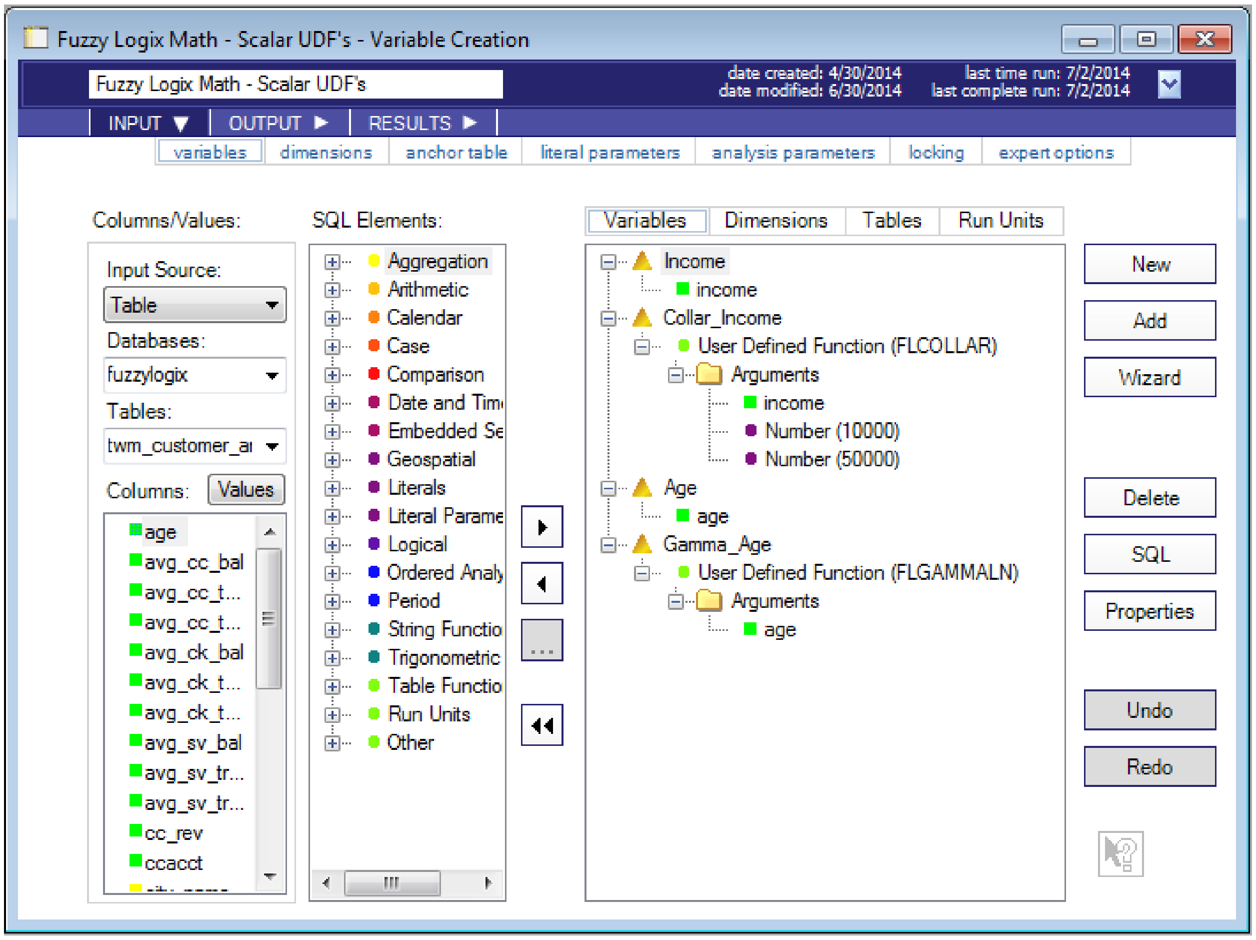
So, for example, this document or the Fuzzy Logix DB Lytix™ User's Guide could be easily added to ADS Generator's Help Menu.

2. Creating Variables by Executing DB Lytix™ Scalar and Aggregate UDF's

Although the capability to call either Scalar or Aggregate UDF's is not new to the 05.03.05 release of ADS Generator, we include it here for completeness of the tutorial. It is quite straight-forward to create variables by executing Scalar or Aggregate UDF's.

2.1. DB Lytix™ Scalar UDF's

First let's look at the Scalar UDF example. Here is what the finished analysis looks like ("Fuzzy Logix Mathematics – Scalar UDF's" Variable Creation analysis in the tutorial):



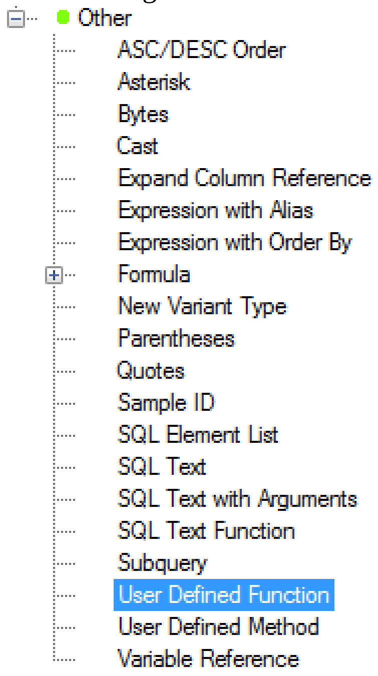
First, we select two columns from the "twm_customer_analysis" table – "income" and "age." Then we are going to execute two scalar UDF's against those columns. The DB Lytix™ function FLCOLLAR will be called to find the "collar" value for "income" (limits the values of "income" to the specified range of upper and lower bounds). Additionally, the FLGAMMALN function will be called to perform a transformation of the column "age" into the natural logarithm of the gamma function. Here are the steps to parameterize the analysis:

1) Select the "twm_customer_analysis" table in the "Tables:" pull-down list.

2) Select the “age” and “income” columns as Variables by highlighting each in the “Columns:” list and either:
   a. Double clicking each,
   b. Clicking on the > button, or
   c. Dragging and dropping them to the Variables palette.

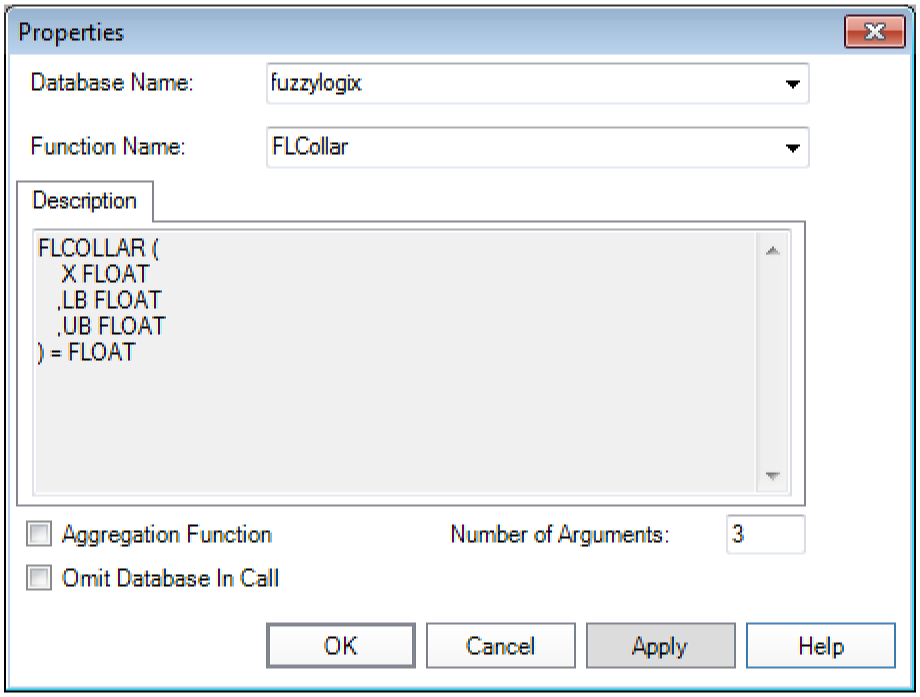
   In the example, these columns have been aliased (Capitalized) by highlighting them with a single click and hold (Windows style rename) and changing the name. Optionally, click on the “Properties” button to do this within the Properties dialogue.

3) Select the FLCOLLAR function as follows:
   a. Click on the + icon next to “Other” in the “SQL Elements” list. It will expand and show the following elements:



Other
- ASC/DESC Order
- Asterisk
- Bytes
- Cast
- Expand Column Reference
- Expression with Alias
- Expression with Order By
- Formula
- New Variant Type
- Parentheses
- Quotes
- Sample ID
- SQL Element List
- SQL Text
- SQL Text with Arguments
- SQL Text Function
- Subquery
- User Defined Function
- User Defined Method
- Variable Reference

   b. Select “User Defined Function” by double clicking on it, highlighting it and clicking on the > button, or highlighting it and dragging and dropping it on the Variables palette. This will bring up the following dialogue:


Properties

Database Name: fuzzylogix

Function Name: FLCollar

Description

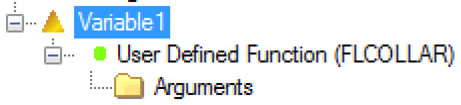
```
FLCOLLAR (
   X FLOAT
  ,LB FLOAT
  ,UB FLOAT
) = FLOAT
```

☐ Aggregation Function   Number of Arguments: 3

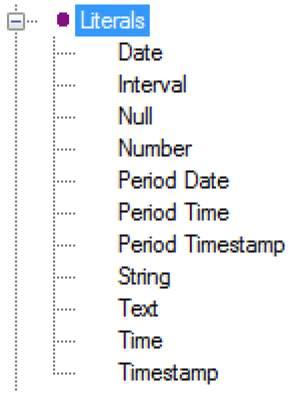
☐ Omit Database In Call

Select the database where DB Lytix™ is installed in the "Database Name" pull-down. Once you do this, the "Function Name:" dialogue will be populated with the Scalar and Aggregate UDF's that reside in the selected database. It will also show the signature of the function under the "Description" tab. Select FLCollar in the "Function Name:" pull-down. The "Aggregation Function" check-box will be disabled since this is a scalar function. By default, ADS Generator will fully qualify the UDF call by prepending it with the database name it resides in. You can turn this default behavior off by enabling the "Omit Database In Call" check-box. You may hit "OK" to save these selections and close the dialogue, or "Apply" to save these selections and leave the dialogue up. Note that this dialogue can be left up allowing you to parameterize additional Scalar and Aggregate UDF calls.

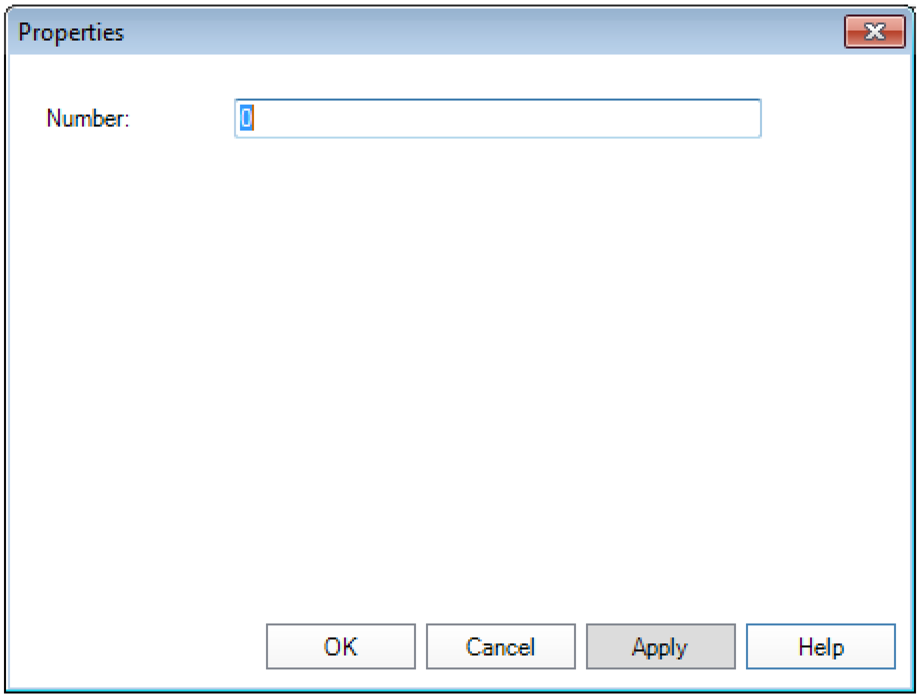
4) Parameterize the call to the FLCOLLAR function as follows:
   a. By default, the FLCollar UDF variable will look like:



   b. It is parameterized by dragging and dropping "income" from the "Columns" list and dropping it into the "Arguments" folder. Optionally, you can highlight the "Arguments" folder and then double click "income" from the "Columns:" list.
   c. Next you need to specify the upper and lower bounds for the FLCollar function. You do this by first clicking on the "+" icon next to "Literals" in the "SQL Elements" list. The expanded list looks like the following:



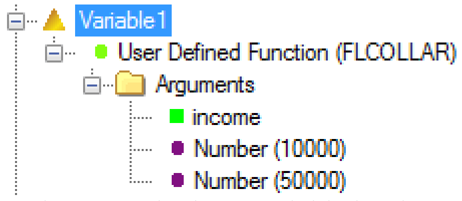
Select the “Number” literal by highlighting/dragging and dropping or using the “>” icon into the Arguments folder. The following dialogue opens when you do this:



Enter “10000” in the “Number:” text box and click OK.

d. Repeat step c. for the value “50000”, or simply highlight the “Number 10000” argument, press the “Ctrl” key on your keyboard and drag and drop it back into the “Arguments” folder. This is a quick and dirty way to copy parameters that you should find useful.

e. “Variable1” will now look like this:



In the example these variable has been given the name “Collar_Income” by highlighting “Variable1” through a highlight and single click Window’s style rename, or by changing the name in the “Properties” dialogue.

5) Select and parameterize the call to the FLGAMMALN function as follows:
   a. Follow the instructions in 3) a. - 3) c. choosing FLGammaLN instead of FLCollar in 3) c.
   b. It is parameterized by dragging and dropping "age" from the "Columns:" list and dropping it into the "Arguments" folder. Optionally, you can highlight the "Arguments" folder and then double click "age" from the "Columns:" list.
   c. Since this is the only argument to FLGammaLN, the parameterization is complete. Optionally, follow the instructions in 4) e. to rename "Variable2" to "Gamma_Age" to match the tutorial.

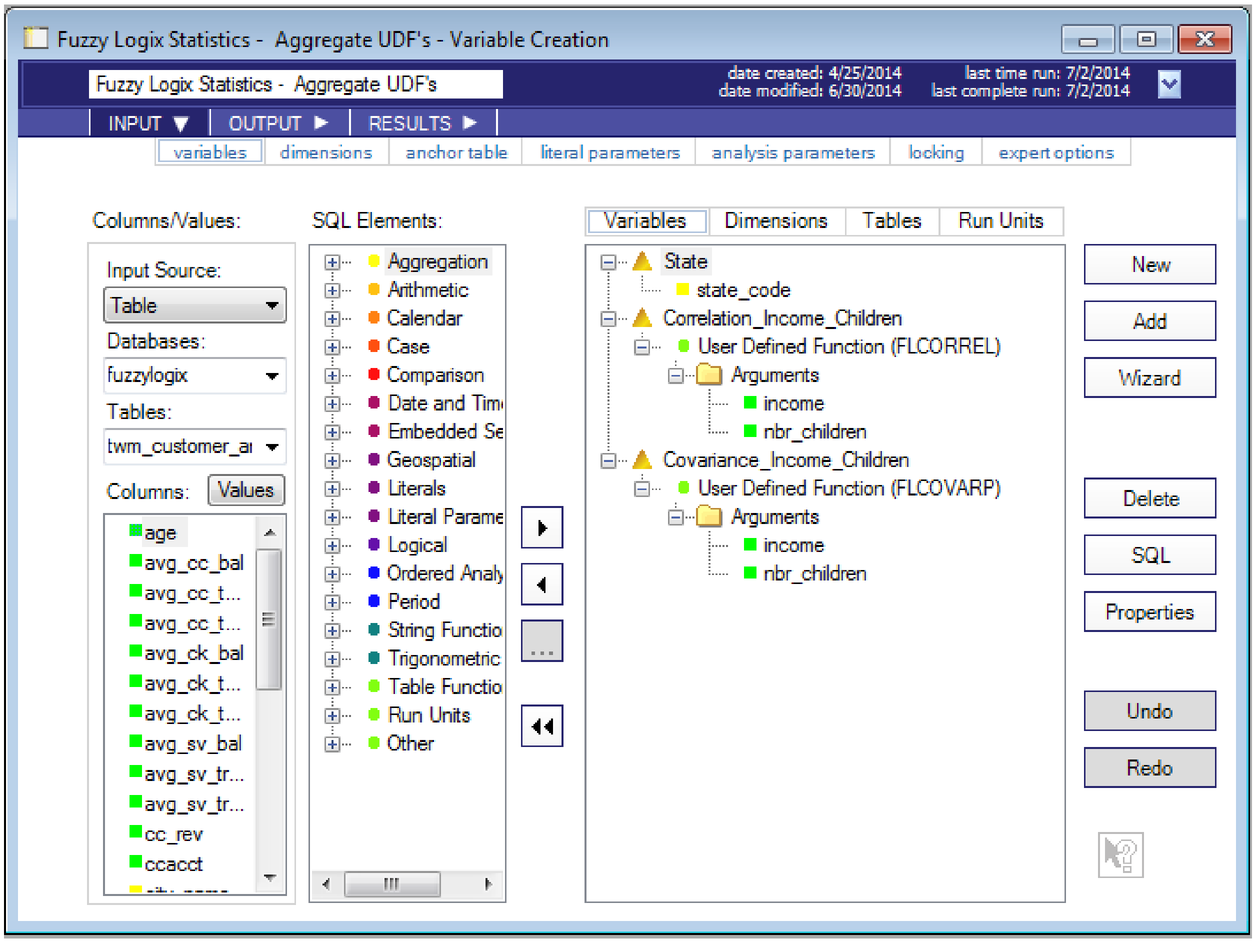
Since this is a Scalar function, nothing additional is required. Simply execute the Variable Creation Analysis by clicking on the "Run" button directly under the "Tools" menu item, or highlighting the Variable Creation analysis in the "Project Explorer," right-clicking and selecting "Run."

Once the Status of the analysis execution is "Complete" in the "Execution Status" window, click on the "Results→data" tab and you should see 747 rows returned with values for income, the collar value (10000→50000) of income, the values of age, and the gamma natural logarithm transformation of age. The SQL under "Results→sql" tab should look like this:

```
SELECT
                "_twmVC0"."income" AS "Income"
               ,"fuzzylogix"."FLCOLLAR"("_twmVC0"."income", 10000, 50000) AS "Collar_Income"
               ,"_twmVC0"."age" AS "Age"
               ,"fuzzylogix"."FLGAMMALN"("_twmVC0"."age") AS "Gamma_Age"
FROM "fuzzylogix"."twm_customer_analysis" AS "_twmVC0"
ORDER BY "_twmVC0"."cust_id"
;
SELECT
                "_twmVC0"."income" AS "Income"
               ,"fuzzylogix"."FLCOLLAR"("_twmVC0"."income", 10000, 50000) AS "Collar_Income"
               ,"_twmVC0"."age" AS "Age"
               ,"fuzzylogix"."FLGAMMALN"("_twmVC0"."age") AS "Gamma_Age"
FROM "fuzzylogix"."twm_customer_analysis" AS "_twmVC0"
ORDER BY "_twmVC0"."cust_id"
;
```

2.2. DB Lytix™ Aggregate UDF's

Next, we'll look at the Aggregate UDF example. Here is what the finished analysis looks like ("Fuzzy Logix Statistics – Aggregate UDF's" Variable Creation analysis in the tutorial):



Fuzzy Logix Statistics - Aggregate UDF's

date created: 4/25/2014
date modified: 6/30/2014
last time run: 7/2/2014
last complete run: 7/2/2014

Input Source: Table
Databases: fuzzylogix
Tables: twm_customer_ai

age
avg_cc_bal
avg_cc_t...
avg_cc_t...
avg_ck_bal
avg_ck_t...
avg_ck_t...
avg_sv_bal
avg_sv_tr...
avg_sv_tr...
cc_rev
ccacct

```
State
    state_code
Correlation_Income_Children
    User Defined Function (FLCORREL)
        Arguments
            income
            nbr_children
Covariance_Income_Children
    User Defined Function (FLCOVARP)
        Arguments
            income
            nbr_children
```

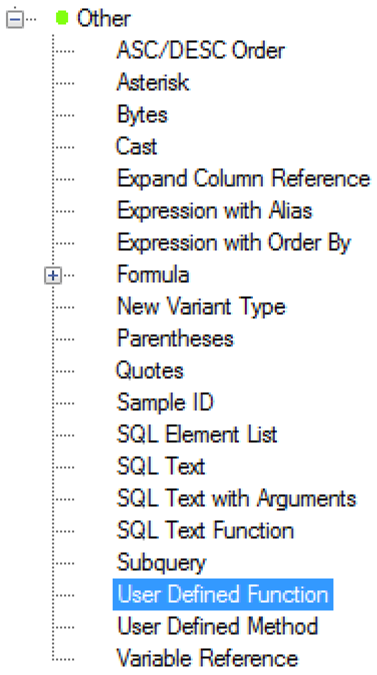
First, we select "state_code" column from the "twm_customer_analysis" table. For each state code, we are going to execute two aggregate UDF's; first the DB Lytix™ function FLCORREL will be called to find the correlation value between two additional columns in the "twm_customer_analysis" table – "income" and "nbr_children". Then FLCOVARP function will be called to calculate the population covariance of the same two columns. Here are the steps to parameterize the analysis.

1) Select the "twm_customer_analysis" table in the "Tables:" pull-down list.

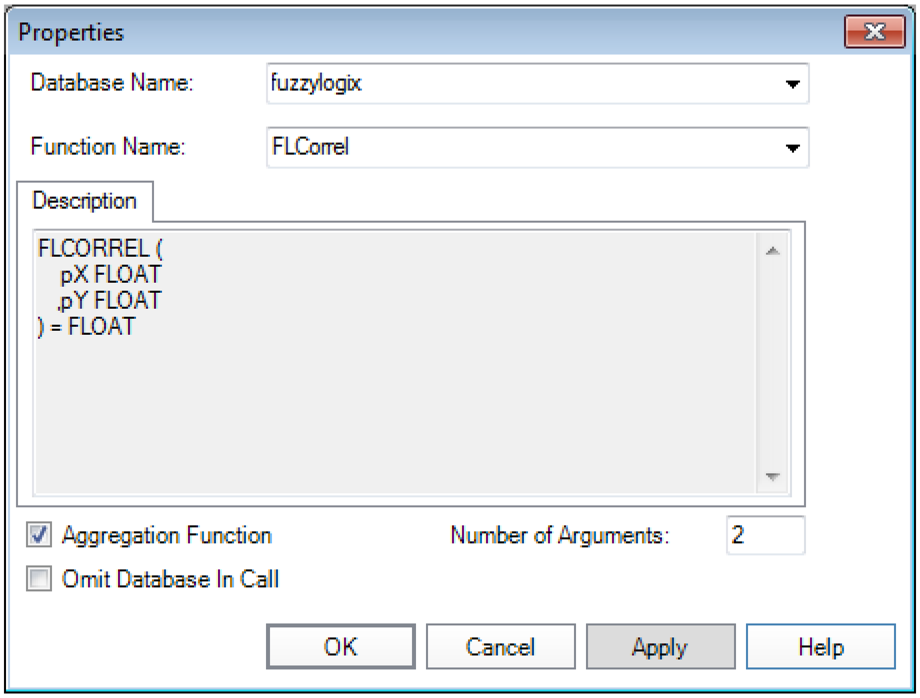
2) Select the "state_code" column as Variables by highlighting it in the "Columns:" list and either:
   a. Double clicking it,
   b. Clicking on the > button, or
   c. Dragging and dropping it to the Variables palette.

   In the example this column has been aliased to "State" by highlighting it with a single click and hold (Windows style rename) and changing the name. Optionally, click on the "Properties" button to do change the name within the Properties dialogue.

3) Select the FLCORREL function as follows:
   a. Click on the + icon next to "Other" in the "SQL Elements" list. It will expand and show the following elements:


- Other
  - ASC/DESC Order
  - Asterisk
  - Bytes
  - Cast
  - Expand Column Reference
  - Expression with Alias
  - Expression with Order By
  - Formula
  - New Variant Type
  - Parentheses
  - Quotes
  - Sample ID
  - SQL Element List
  - SQL Text
  - SQL Text with Arguments
  - SQL Text Function
  - Subquery
  - User Defined Function
  - User Defined Method
  - Variable Reference

b. Select "User Defined Function" by double clicking on it, highlighting it and clicking on the > button, or highlighting it and dragging and dropping it on the Variables palate. This will bring up the following dialogue:


Properties

Database Name: fuzzylogix

Function Name: FLCorrel

Description

```
FLCORREL (
   pX FLOAT
  ,pY FLOAT
) = FLOAT
```

[✓] Aggregation Function   Number of Arguments: 2

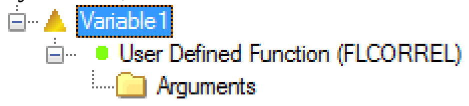
[ ] Omit Database In Call

Note that the "Aggregate Function" checkbox has been enabled as the front-end recognizes this is an Aggregate UDF vs. Scalar UDF.

Select the database where DB Lytix™ is installed in the "Database Name:" pull-down. Once you do this, the "Function Name:" dialogue will be populated with the Scalar and

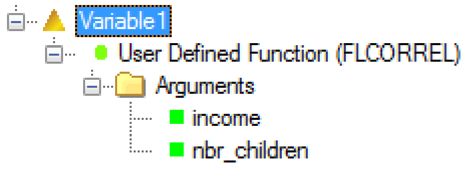
Aggregate UDF's that reside in the selected database. It will also show the signature of the function under the "Description" tab. Select FLCorrel in the "Function Name" pull-down. By default, ADS Generator will fully qualify the UDF call by prepending it with the database name it resides in. You can turn this default behavior off by enabling the "Omit Database In Call" check-box. You may hit "OK" to save these selections and close the dialogue, or "Apply" to save these selections and leave the dialogue up. Note that this dialogue can be left up allowing you to parameterize additional Scalar and Aggregate UDF calls.

4) Parameterize the call to the FLCorrel function as follows:
   a. By default, the FLCorrel UDF variable will look like:



Variable1
User Defined Function (FLCORREL)
Arguments

   b. It is parameterized by dragging and dropping "income" and "nbr_children" from the "Columns:" list and dropping it into the "Arguments" folder. Optionally, you can highlight the "Arguments" folder and then double click "income" and then "nbr_children" from the "Columns:" list or use the ">" icon.
   c. "Variable1" will now look like this:

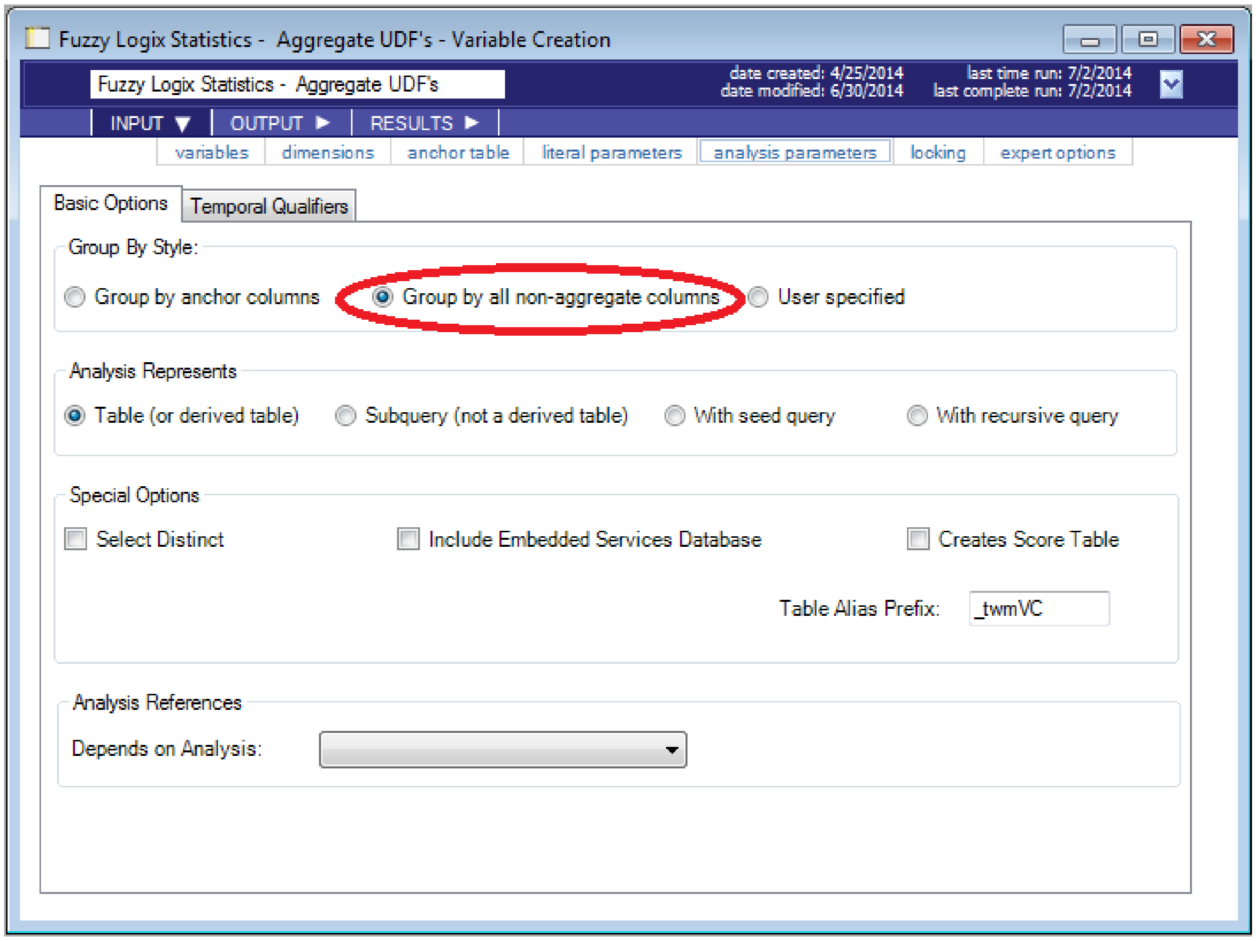


Variable1
User Defined Function (FLCORREL)
Arguments
income
nbr_children

In the example these variable has been given the name "Correlation_Income_Children" by highlighting "Variable1" through a highlight and single click Window's style rename, or by changing the name in the Properties dialogue.

5) Select and parameterize the call to the FLCOVARP function as follows:
   a. Follow the instructions in 3) a. - 3) c. choosing FLCovarP instead of FLCorrel in 3) c.
   b. It takes the same parameters as FLCorrel, so follow parameterize it following the same instructions in 4) b.
   c. Optionally, follow the instructions in 4) e. to rename "Variable2" to "Covariance_Income_Children" to match the tutorial.

6) Since this is an Aggregate function, one additional parameter is required. Click on the "analysis parameters" tab to select the "Group by all non-aggregate columns" parameter:



Group By Style:
Group by anchor columns [✓ Group by all non-aggregate columns] User specified

Analysis Represents:
[✓ Table (or derived table)] Subquery (not a derived table) With seed query With recursive query

Special Options:
☐ Select Distinct ☐ Include Embedded Services Database ☐ Creates Score Table

Table Alias Prefix: _twmVC

Analysis References:
Depends on Analysis:

By default, ADS will group by anchor column(s), which in this case is the primary index of the "twm_customer_analysis" table, "cust_id." This option allows us to generate the SQL statement grouping by "state_code."

7) Now execute the Variable Creation Analysis by clicking on the "Run" button directly under the "Tools" menu item, or highlighting the Variable Creation analysis in the "Project Explorer," right-clicking and selecting "Run."

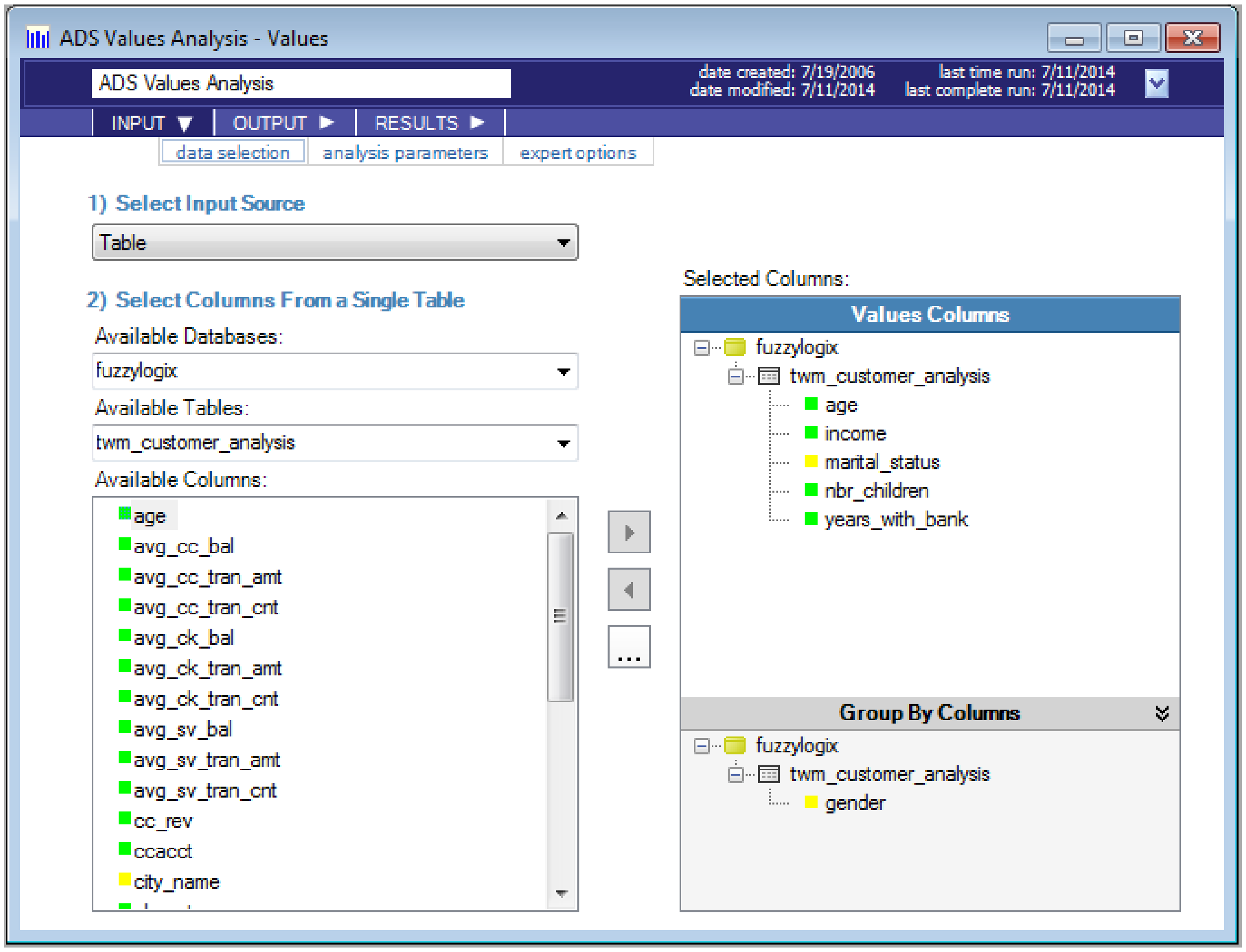
Once the Status of the analysis execution is "Complete" in the "Execution Status" window, click on the "Results→data" tab and you should see 33 rows returned with all available values for "state_code" along with the calculated correlation and population covariance of income and number of children for each State. The SQL under "Results→sql" tab should look like this:

```
SELECT
        "_twmVC0"."state_code" AS "State"
       ,"fuzzylogix"."FLCORREL"("_twmVC0"."income", "_twmVC0"."nbr_children") AS
"Correlation_Income_Children"
       ,"fuzzylogix"."FLCOVARP"("_twmVC0"."income", "_twmVC0"."nbr_children") AS
"Covariance_Income_Children"
FROM "fuzzylogix"."twm_customer_analysis" AS "_twmVC0"
GROUP BY 1
ORDER BY 1
;
```

3. Notes on Functional Overlap

The next four analyses in the "Teradata ADS Demo Powered by Fuzzy Logix" project illustrate some of the functional overlap between the two products. Note that the two products are 99% complimentary and even within an area of functional overlap, certain use cases will lend themselves to one solution or the other – the end-user ultimately gets the best of both worlds. The areas that overlap are limited to the ADS Generator Descriptive Statistics and some of the DB Lytix™ Mathematical and Statistical Scalar or Aggregate UDF's. For those functions that do overlap, the difference in processing is that the ADS Generator analysis allows you to process any or all columns within a table, whereas the DB Lytix™ functions are called once per column.

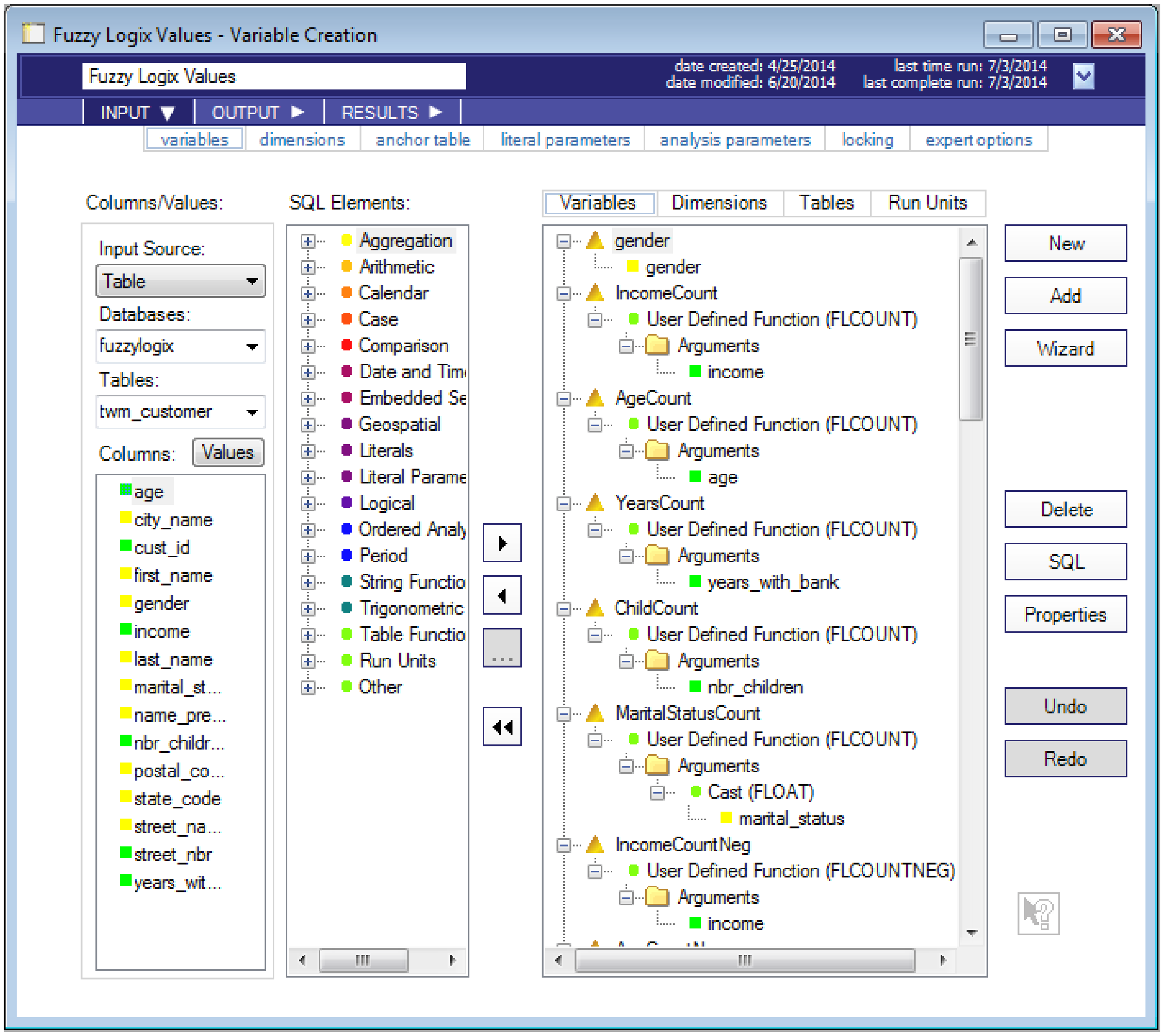
The first example compares an ADS "Values" analysis (Descriptive Statistics → Values) to a set of DB Lytix™ Statistical Aggregate UDF's. The ADS "Values" analysis provides a count of the number of rows, rows with non-null values, rows with null values, rows with value 0, rows with a positive value, rows with a negative value, and the number of rows containing blanks from a given table name and column. It can optionally group by an additional column. In the example "ADS Values Analysis" shown below, these calculations are given for five columns in the "twm_customer_analysis" table ("income", "age", "years_with_bank", "nbr_children", "marital_status"), with "gender" given as the group by column:



Fuzzy Logix provides five different aggregate functions that provide the same functionality:

- FLCount - Aggregate function which returns the count of non-NULL values
- FLCountNeg - Aggregate function which returns the count of negative values
- FLCountNull - Aggregate function which returns the count of null values
- FLCountPos - Aggregate function which returns the count of positive values
- FLCountZero - Aggregate function which returns the count of values which are zero

Each aggregate function takes a single floating point number as an argument. Therefore you need to call each function five times, for “income”, “age”, “years_with_bank”, “nbr_children”, and “marital_status”, resulting in 25 UDF calls, as shown in the “Fuzzy Logix Values” Variable Creation analysis below:



Fuzzy Logix Values

date created: 4/25/2014
date modified: 6/20/2014
last time run: 7/3/2014
last complete run: 7/3/2014

Input Source: Table
Databases: fuzzylogix
Tables: twm_customer
Columns: Values

age
city_name
cust_id
first_name
gender
income
last_name
marital_st...
name_pre...
nbr_childr...
postal_co...
state_code
street_na...
street_nbr
years_wit...

```
gender
    gender
IncomeCount
    User Defined Function (FLCOUNT)
        Arguments
            income
AgeCount
    User Defined Function (FLCOUNT)
        Arguments
            age
YearsCount
    User Defined Function (FLCOUNT)
        Arguments
            years_with_bank
ChildCount
    User Defined Function (FLCOUNT)
        Arguments
            nbr_children
MaritalStatusCount
    User Defined Function (FLCOUNT)
        Arguments
            Cast (FLOAT)
                marital_status
IncomeCountNeg
    User Defined Function (FLCOUNTNEG)
        Arguments
            income
```

Similar overlapping functionality is illustrated by the “ADS Statistics Analysis” (Descriptive Statistics -> Statistics) and the “Fuzzy Logix Statistics” Variable Creation analysis in the “Teradata ADS Demo Powered by Fuzzy Logix” project, but not described here.

4. <u>Creating Variables From DB Lytix™ Table UDF’s</u>

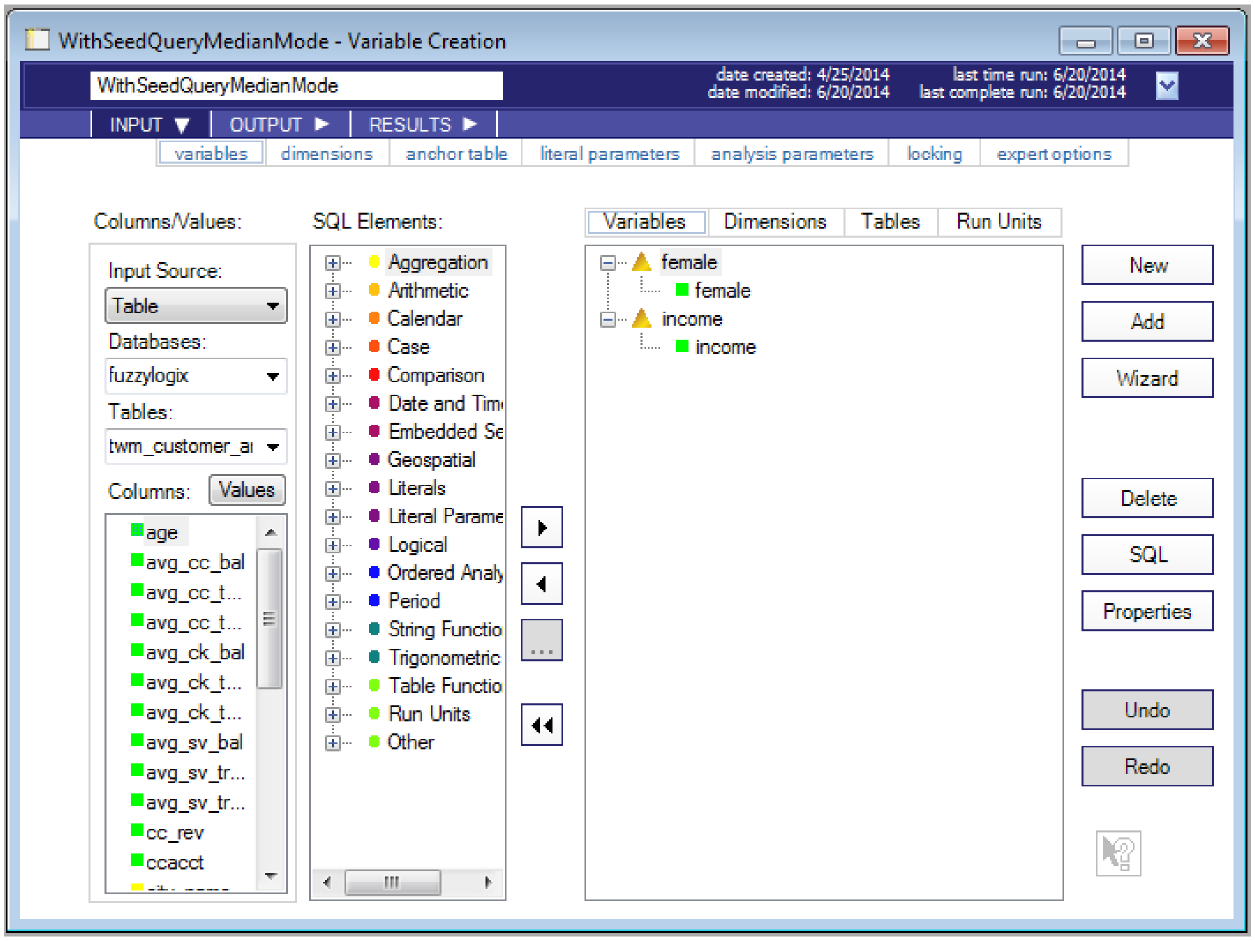
Next, we'll look at a Table UDF example. The Table Functions within the DB Lytix™ library utilize what is known as the "With Seed Query" syntax. The SQL syntax looks as follows:

```
WITH "WithSeedQueryName" ("Parameter1", “Parameter2”, … , “ParameterN”)  AS
(
   SELECT
            "_SeedQueryTable"."Parameter1" AS "Parameter1"
            ,"_SeedQueryTable"."Parameter2" AS "Parameter2"
                        …
            ,"_SeedQueryTable"."ParameterN" AS "ParameterN"
    FROM "DBName"."TableName" AS "_SeedQueryTable"
)

SELECT
            "TableFunction"."Output1" AS "Output1"
           ,"TableFunction"."Output2" AS "Output2"
                        …
           ,"TableFunction"."OutputN" AS "OutputN"
FROM

    TABLE("TableFunctionName"(CAST("WithSeedQueryName"."Parameter1" AS DataType),
                                    CAST("WithSeedQueryName"."Parameter2" AS DataType),
                                                      ,,,
                                    CAST("WithSeedQueryName "."ParameterN" AS DataType))

    HASH BY "WithSeedQueryName"."Parameter1",
                "WithSeedQueryName"."Parameter2",
                        …
                "WithSeedQueryName"."ParameterN"

    LOCAL ORDER BY "WithSeedQueryName"."Parameter1",
                          "WithSeedQueryName"."Parameter2",
                              …
                          "WithSeedQueryName"."ParameterN”)
```

In order to create this type of query within Teradata ADS Generator, a feature known as "Analysis References", in which one Variable Creation Analysis (for the With Seed Query) is input to another Variable Creation analysis (for the Table Function call and selection of the Table Function output elements as Variables).

Note that during the collaboration between Teradata and Fuzzy Logix, it was determined that a "Template" for Queries that select from a Table UDF and utilize the With Query syntax would be extremely beneficial. This template is described in section 4.1, but it is still important to understand how to do this manually. What follows is a description of how to setup the two Variable Creation analyses for a With Query reading from a Table Function manually. Feel free to review section 4.1 first, before returning back here.

The first step is to create the Variable Creation analysis that will represent the With Seed Query portion of the generated query. Here is what the finished analysis looks like ("WithSeedQueryMedianMode" Variable Creation analysis in the tutorial):


WithSeedQueryMedianMode - Variable Creation

WithSeedQueryMedianMode

date created: 4/25/2014
date modified: 6/20/2014
last time run: 6/20/2014
last complete run: 6/20/2014

Input Source: Table
Databases: fuzzylogix
Tables: twm_customer_a

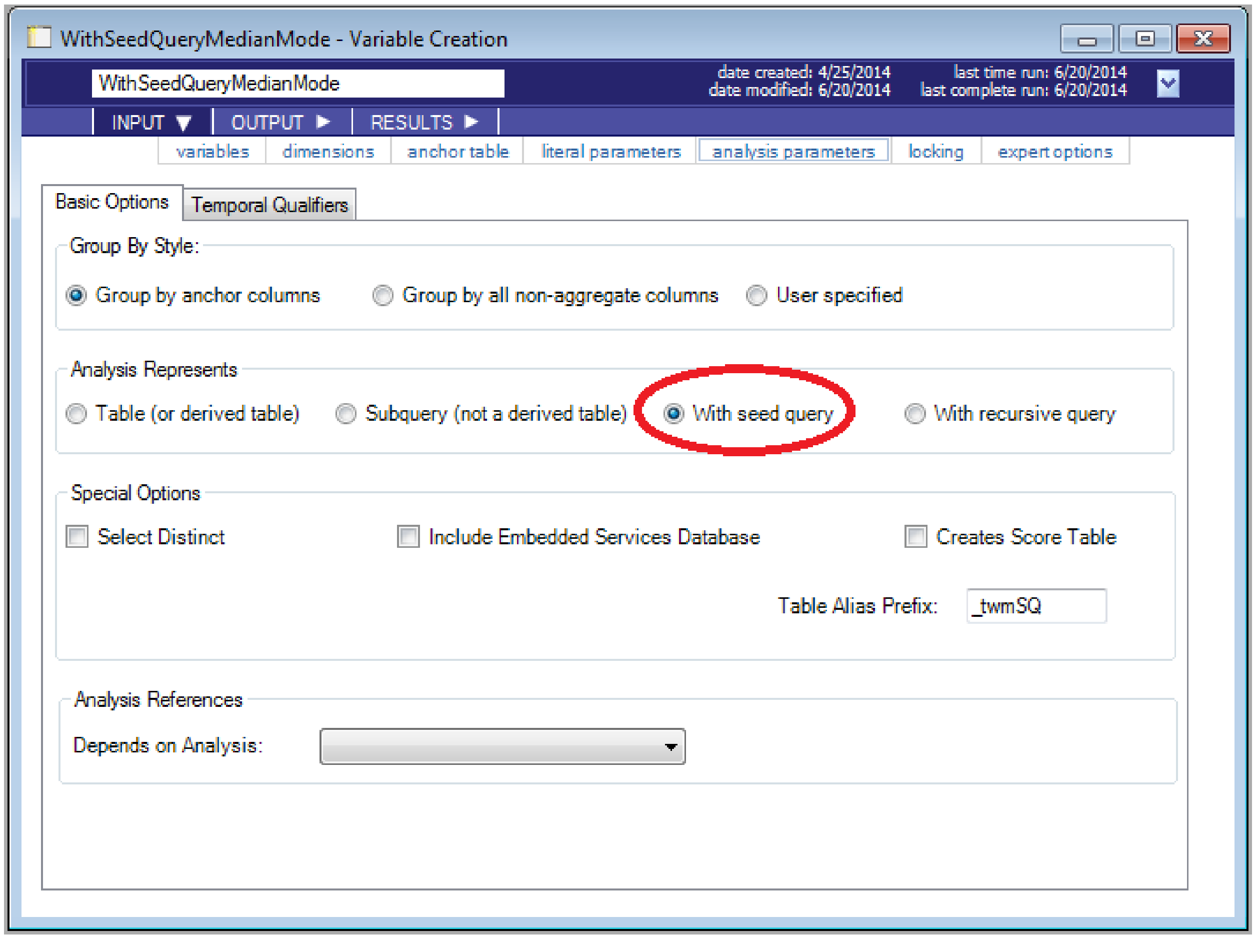
First, we select the "female" column from the "twm_customer_analysis" table. Then we select the "income" column in the "twm_customer_analysis" table. Next, you parameterize the analysis to only generate SQL and not execute it. Finally, you parameterize the analysis to specify that it is a "With Seed Query". Here are the steps to parameterize the analysis:

1) Select the "twm_customer_analysis" table in the "Tables:" pull-down list.

2) Select the "female" column as a Variable by highlighting it in the "Columns:" list and either:
   a. Double clicking it,
   b. Clicking on the > button, or
   c. Dragging and dropping it to the Variables palette.

   Next, select the "income" column as a Variable by following the directions above.

3) Next, click on the "analysis parameters" tab, and select the "With Seed Query" option as follows:


WithSeedQueryMedianMode - Variable Creation

WithSeedQueryMedianMode

date created: 4/25/2014
date modified: 6/20/2014
last time run: 6/20/2014
last complete run: 6/20/2014

Group By Style: [✓ Group by anchor columns] Group by all non-aggregate columns User specified

Analysis Represents: Table (or derived table) Subquery (not a derived table) [✓ With seed query] With recursive query

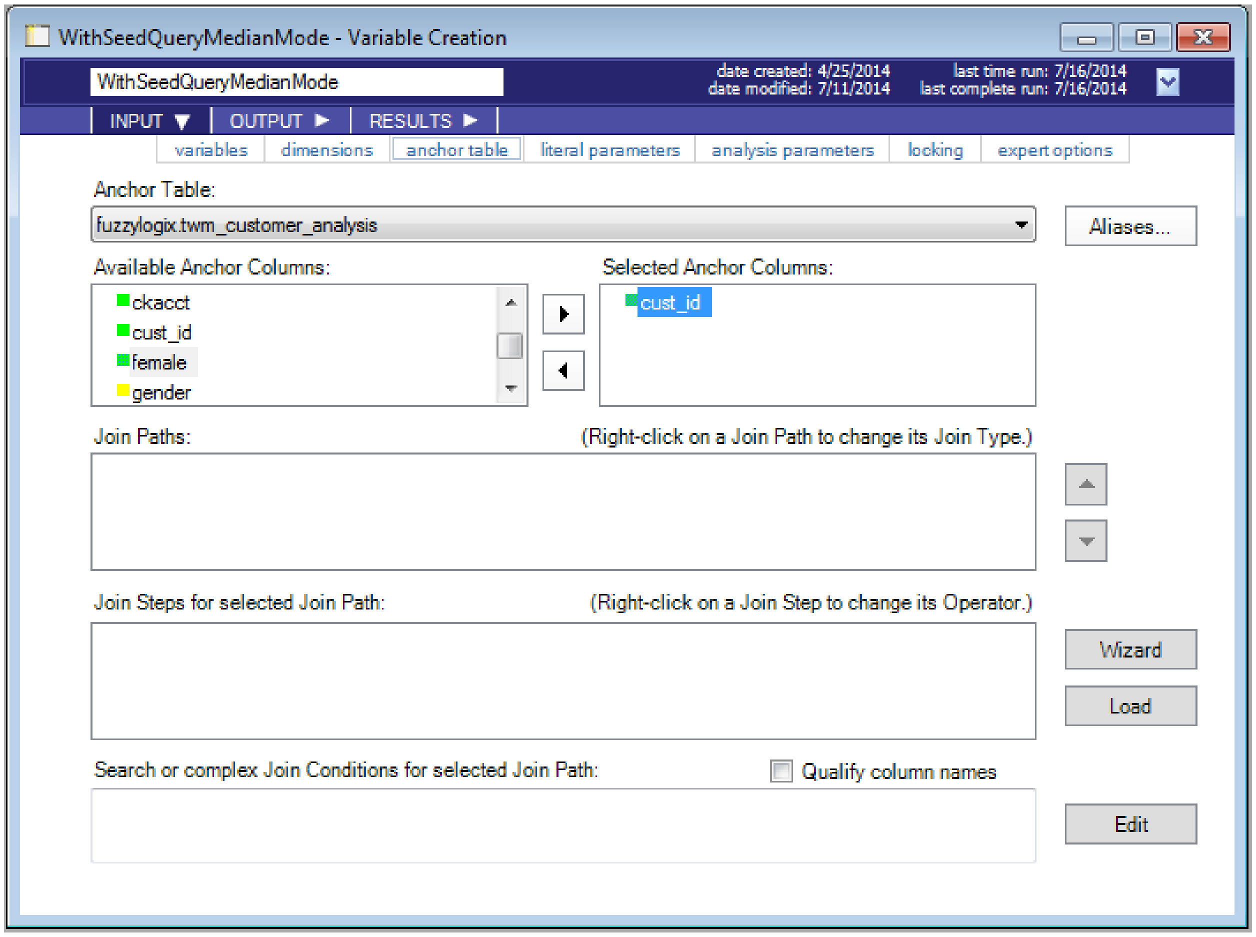
Special Options: ☐ Select Distinct ☐ Include Embedded Services Database ☐ Creates Score Table

Table Alias Prefix: _twmSQ

Analysis References

Depends on Analysis:

4) Next, click on the "anchor table" tab, and de-select the "cust_id" column as the default "Selected Anchor Column" follows:


WithSeedQueryMedianMode - Variable Creation

WithSeedQueryMedianMode

date created: 4/25/2014
date modified: 7/11/2014
last time run: 7/16/2014
last complete run: 7/16/2014

Anchor Table: fuzzylogix.twm_customer_analysis

Available Anchor Columns:
🟩 ckacct
🟩 cust_id
🟩 female
🟨 gender

Selected Anchor Columns:
🟩 cust_id

Join Paths: (Right-click on a Join Path to change its Join Type.)

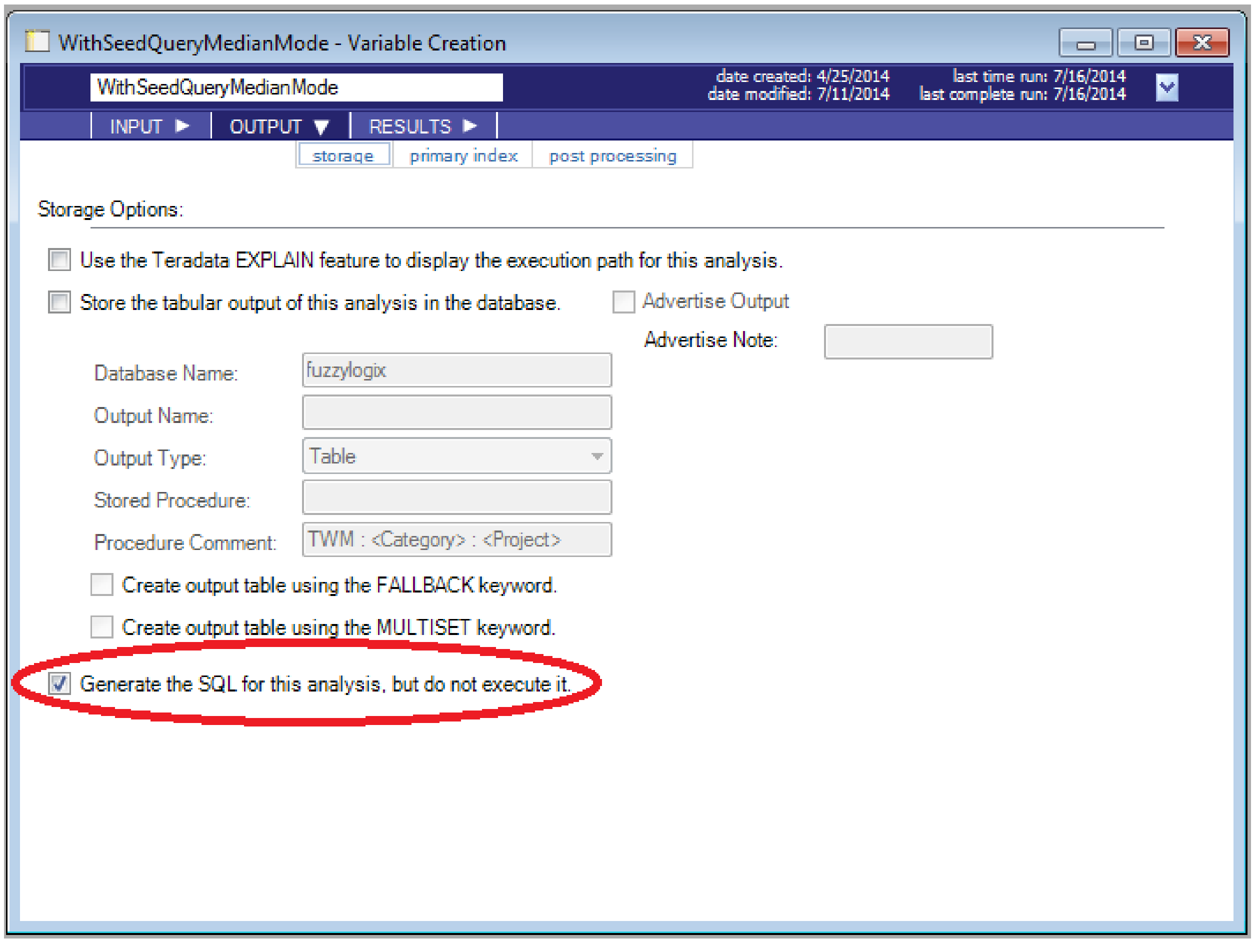
Join Steps for selected Join Path: (Right-click on a Join Step to change its Operator.)

Search or complex Join Conditions for selected Join Path:

Qualify column names [ ]

Highlight the “cust_id” column from the “Selected Anchor Columns:” list and click on the “de-select” or backward arrow button. If this is not done, an ORDER BY clause will be automatically generated which results in a syntax error for the WITH query clause.

5) Next, click on the “OUTPUT→storage” tab, and select the “Generate the SQL for this analysis but do not Execute it” option as follows:



WithSeedQueryMedianMode
date created: 4/25/2014
date modified: 7/11/2014
last time run: 7/16/2014
last complete run: 7/16/2014

Storage Options:

☐ Use the Teradata EXPLAIN feature to display the execution path for this analysis.

☐ Store the tabular output of this analysis in the database. ☐ Advertise Output

Advertise Note:

Database Name: fuzzylogix

Output Name:

Output Type: Table

Stored Procedure:

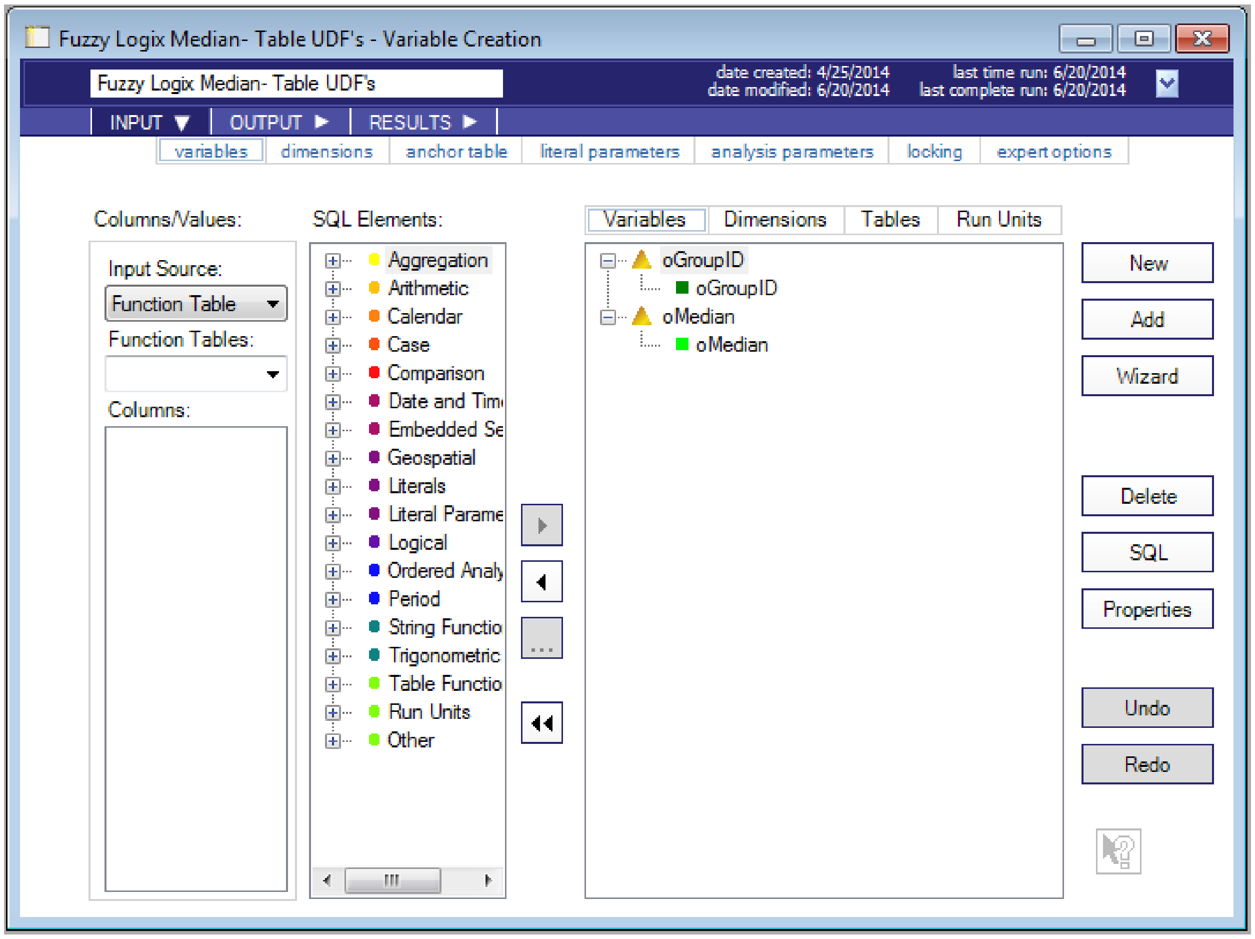
Procedure Comment: TWM : <Category> : <Project>

☐ Create output table using the FALLBACK keyword.

☐ Create output table using the MULTISET keyword.

☑ Generate the SQL for this analysis, but do not execute it.

6) The next step is to create the Variable Creation analysis that will call the Table Function, and select the elements returned from the table function as variables. Here is what the finished analysis looks like (“Fuzzy Logix Median- Table UDF's” Variable Creation analysis in the tutorial):



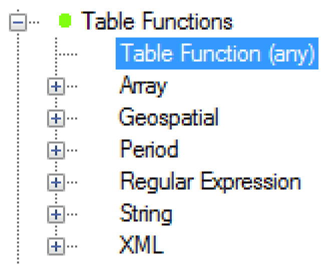
First, we change the Input Source of the analysis to “Analysis.” Then we select the “WithSeedQueryMedianMode” analysis from the “Analyses:” pull-down list. This populates the “Columns:” area with “female” and “income” – the variables generated by the “WithSeedQueryMedianMode” analysis. Next, you change palette area from “Variables” to “Tables” and select the “FLMEDIANUDT” DB Lytix™ Table Function that will calculate the median values of the data specified by the “With Seed Query.” The “FLMedianUDT” function is parameterized per its signature, and the “HASH BY” and “LOCAL ORDER BY” options are specified. Finally an expert option is specified to tell the analysis that it must generate the “With Query Analysis,” and you are ready to execute the analysis! Here are the steps to parameterize the analysis:

7) Change the Input Source of the analysis to “Analysis.”

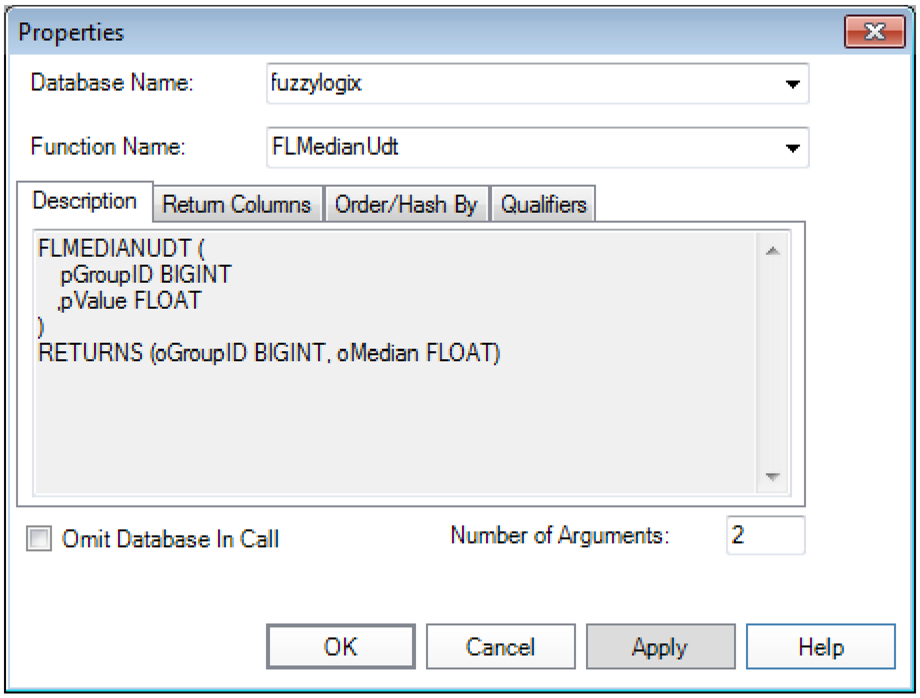
8) Select the “WithSeedQueryMedianMode” analysis from the “Analyses:” pull-down list – the “Columns:” area becomes populated with “female” and “income.”

9) Next, change the “Palette” area from “Variables” to “Tables.”

10) Select the “FLMEDIANUDT” function as follows:
    a. Click on the + icon next to “Table Functions” in the “SQL Elements” list. It will expand and show the following elements:



b. Select “Table Function (any)” by either double clicking on it, highlighting it and Clicking on the > button, or highlighting it and dragging and dropping it on the Tables palette. This will bring up the following dialogue:



Database Name: fuzzylogix

Function Name: FLMedianUdt

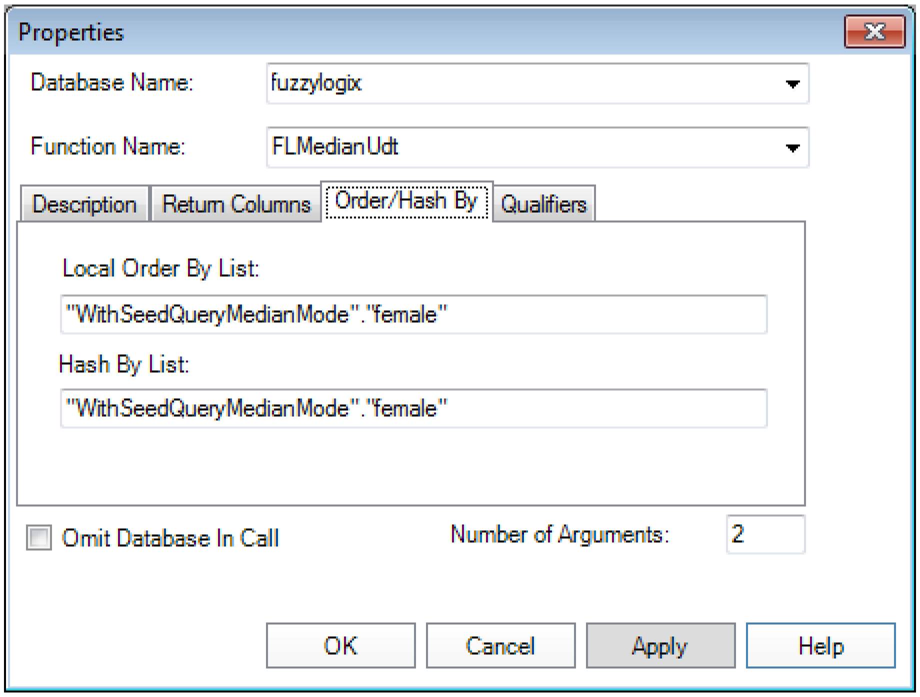
```
FLMEDIANUDT (
   pGroupID BIGINT
  ,pValue FLOAT
)
RETURNS (oGroupID BIGINT, oMedian FLOAT)
```

☐ Omit Database In Call

Number of Arguments: 2

Select the database where DB Lytix™ is installed in the “Database Name” pull-down. Once you do this, the “Function Name:” dialogue will be populated with the Table UDF’s that reside in the selected database. It will also show the signature of the function under the “Description” tab. Select FLMedianUDT in the “Function Name” pull-down. By default, ADS Generator will fully qualify the UDF call by prepending it with the database name it resides in. You can turn this default behavior off by enabling the “Omit Database In Call” check-box.

c. Click on the “Order/Hash By” tab and enter the following:



Database Name: fuzzylogix

Function Name: FLMedianUdt

Local Order By List: "WithSeedQueryMedianMode"."female"

Hash By List: "WithSeedQueryMedianMode"."female"

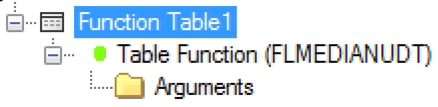
Omit Database In Call [ ]

Number of Arguments: 2

Under the "Local Order By List:" enter "WithSeedQueryMedianMode"."female." (NOTE – Include the double quotes around both object names). Enter the same under the "Hash By List:" This will ensure that we correctly generate the median income values for both Females (females = 1) and Males (females = 0). Click on "OK" or "Apply" and begin to parameterize the FLMedianUDT table function within the Tables palette.

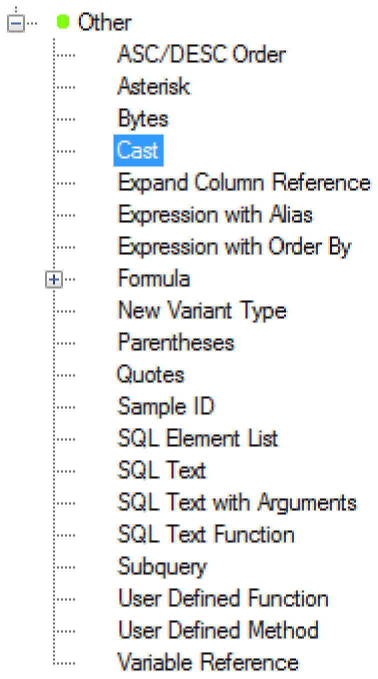
11) Parameterize the call to the FLMedianUDT function as follows:

a. By default, the FLMedianUDT Function Table will look like:

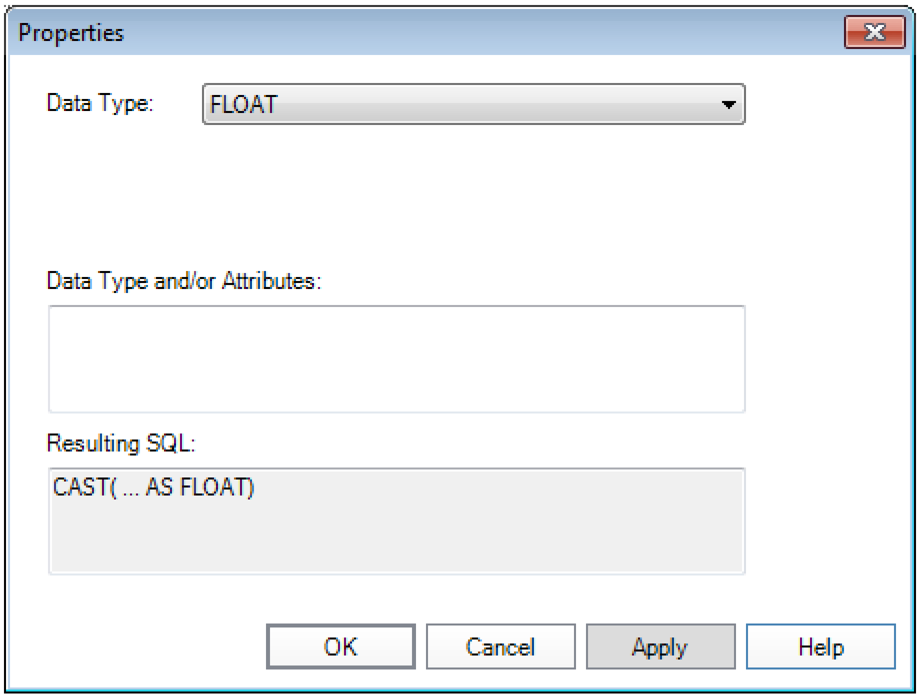


b. It is parameterized by dragging and dropping "female" and "income" from the "Columns:" list and dropping them into the "Arguments" folder. Optionally, you can highlight the "Arguments" folder and then double click "female" and then "income" from the "Columns:" list or use the ">" icon.

c. Next, you must explicitly cast the arguments to the exact type specified by the table functions signature. In this case, "female" is cast to "BIGINT" and "income" to "FLOAT". In order to do this, click on the "+" icon next to "Other" in the "SQL Elements" list. It will expand and show the following elements:



Select "Cast" by dragging and dropping it on top of the "female" and "income" arguments, or individually highlighting the "female" and "income" arguments, and double-clicking "Cast" or using the ">" button. This action will bring up the following dialogue:



Data Type: FLOAT

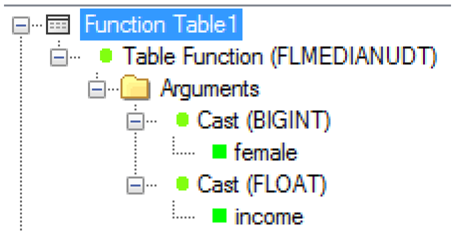
Data Type and/or Attributes:

Resulting SQL:

```
CAST( ... AS FLOAT)
```

Select "BIGINT" in the "Data Type:" pull-down for "female" and "FLOAT" for income.

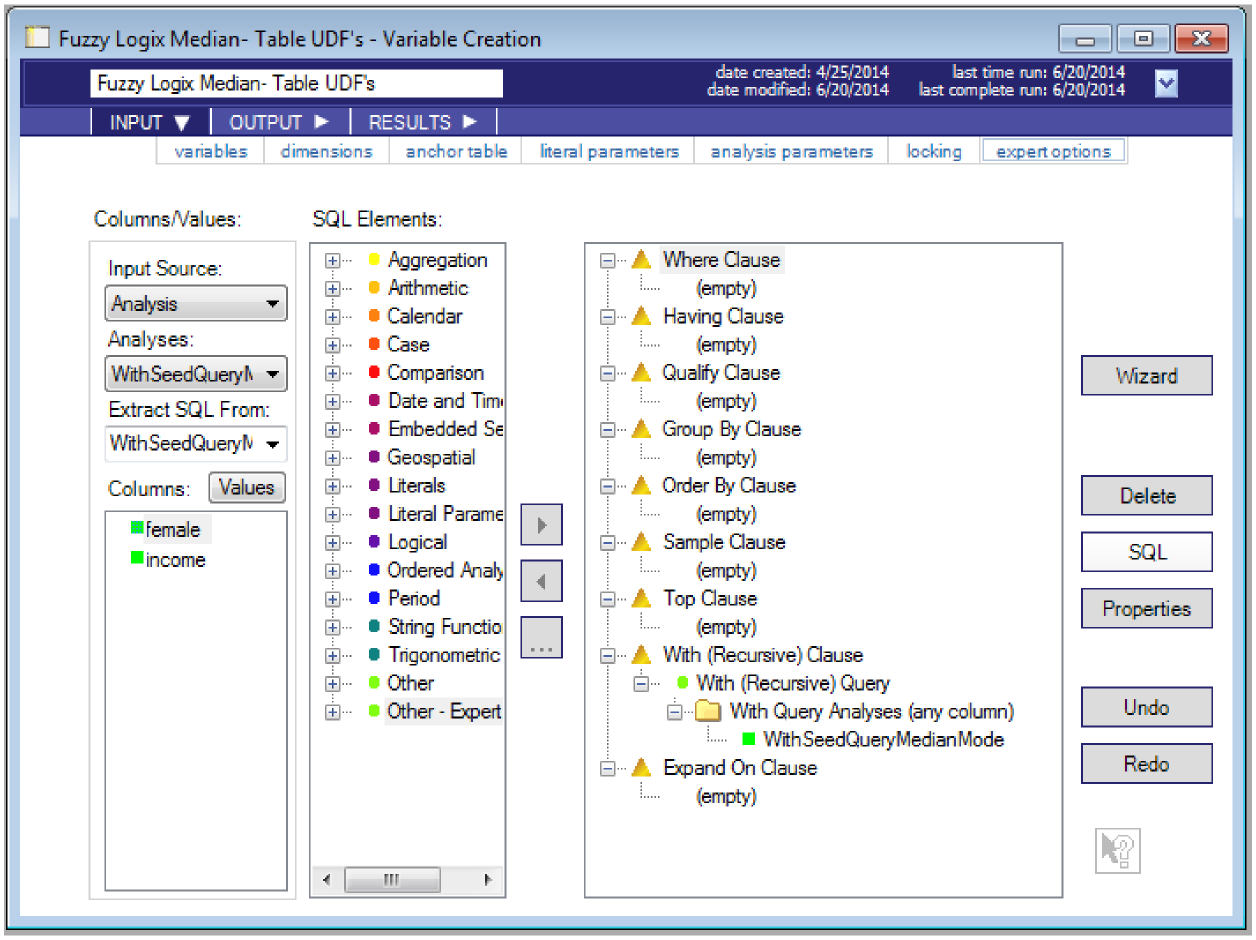
d. "Function Table1" will now look like this:


Function Table1
Table Function (FLMEDIANUDT)
Arguments
Cast (BIGINT)
female
Cast (FLOAT)
income

In the example these Function Table has been given the name “IncomeMedian” by highlighting “Function Table1” through a highlight and single click Window’s style rename, or by changing the name in the Properties dialogue.

12) In the tutorial example provided, an additional Function Table has been added and parameterized to call the “FLModeUDT” Table UDF. Follow the instructions as specified above for “FLMedianUDT” to add and parameterize that function, if desired, as its arguments are the exact same. Note, however, that only one Table Function can be selected from within a single query. For this reason, an additional Variable Creation analysis called “Fuzzy Logix Mode - Table UDF's” has been added to the tutorial project to calculate the modal value for income for both males and females.

13) Next, click on the “expert option” tab, and specify the “With Query Analysis (any column)” option as follows:


Fuzzy Logix Median- Table UDF's - Variable Creation

Fuzzy Logix Median- Table UDF's
date created: 4/25/2014
date modified: 6/20/2014
last time run: 6/20/2014
last complete run: 6/20/2014

INPUT ▼ | OUTPUT ▶ | RESULTS ▶

variables | dimensions | anchor table | literal parameters | analysis parameters | locking | expert options

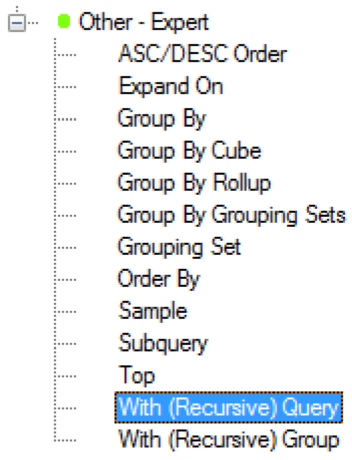
Columns/Values:
Input Source: Analysis
Analyses: WithSeedQueryM
Extract SQL From: WithSeedQueryM
Columns: Values
female
income

SQL Elements:
Aggregation
Arithmetic
Calendar
Case
Comparison
Date and Tim
Embedded Se
Geospatial
Literals
Literal Parame
Logical
Ordered Analy
Period
String Functio
Trigonometric
Other
Other - Expert

Where Clause
(empty)
Having Clause
(empty)
Qualify Clause
(empty)
Group By Clause
(empty)
Order By Clause
(empty)
Sample Clause
(empty)
Top Clause
(empty)
With (Recursive) Clause
With (Recursive) Query
With Query Analyses (any column)
WithSeedQueryMedianMode
Expand On Clause
(empty)

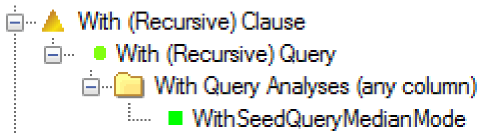
Wizard
Delete
SQL
Properties
Undo
Redo

a. First, expand “Other – Expert Options” in the SQL Elements list by clicking on the “+” icon. It will expand and show the following elements:



Select “With (Recursive) Query” by dragging and dropping it under “With Recursive Clause” in the Expert Options palette or highlighting the “(empty)” argument under “With Recursive Clause”, and double-clicking “With (Recursive) Query” or using the “>” button.

b. Next, drag and drop (or use any of the other selection mechanisms described thus far) either the “female” or “income” variables from the “Columns:” list to the “With Query Analysis (any column)” folder. This will populate the folder with the “WithSeedQueryMedianMode” analysis, telling this variable creation to use it as its seed query as follows:



14) The final step is selecting the return values from the table function as variables within the Variable Creation analysis. The process for this is as follows:

a. First, change the Input Source of the analysis to “Function Table.”
b. Select the “IncomeMedian” Function Table from the “Function Table:” pull-down list – the “Columns:” area becomes populated with “oGroupID” and “oMedian” – the return values of the call to the FLMedianUDT table function.
c. Select the “oGroupID” column and the “oMedian” columns as Variables by highlighting them in the “Columns:” list and either:
    i. Double clicking them individually, or
    ii. Clicking on the > button, or
    iii. Dragging and dropping them to the Variables palette.

15) Now execute the Variable Creation Analysis by clicking on the “Play” button directly under the “Tools” menu item, or highlighting the Variable Creation analysis in the “Project Explorer,” right-clicking and selecting “Run.” Notice that the front-end gives you a visual clue that the analysis being executed references another analysis in the project by highlighting and italicizing the analysis that is referenced:

Note also that the referenced analysis is executed first, followed by the analysis you are executing. Once the execution status of both analyses is “Complete” in the “Execution Status” window, click on the “Results→data” tab and you should see 2 rows returned with the median values for income for females (oGroupID = 1) and males (oGroupID = 0). The SQL under “Results→sql” tab should look like this:

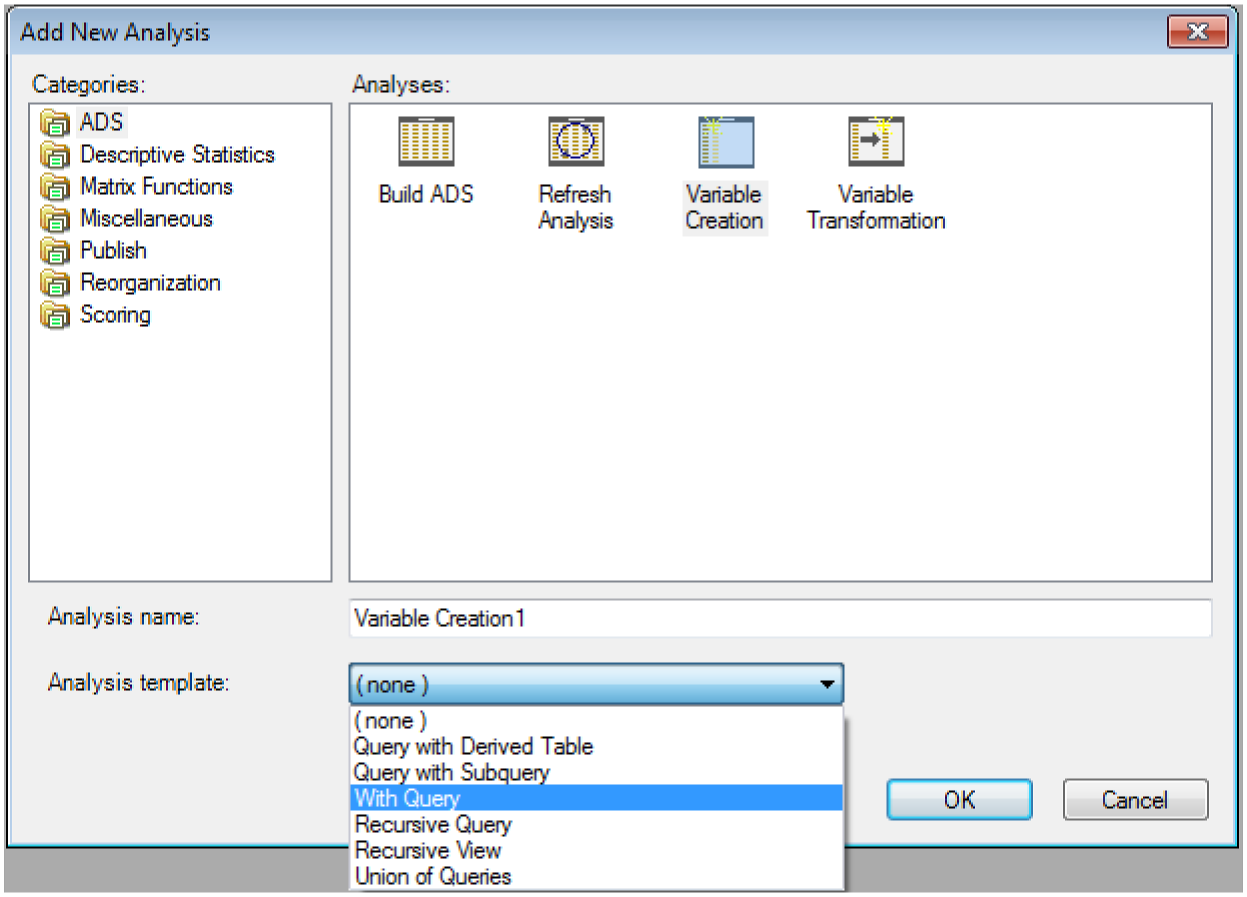
```
WITH "WithSeedQueryMedianMode" ("female", "income")  AS
(SELECT
                "_twmSQ0"."female" AS "female"
               ,"_twmSQ0"."income" AS "income"
FROM "fuzzylogix"."twm_customer_analysis" AS "_twmSQ0"
)
SELECT
                "_twmVC0"."oGroupID" AS "oGroupID"
               ,"_twmVC0"."oMedian" AS "oMedian"
FROM
TABLE("fuzzylogix"."FLMEDIANUDT"(CAST("WithSeedQueryMedianMode"."female" AS BIGINT),
CAST("WithSeedQueryMedianMode"."income" AS FLOAT))
HASH BY "WithSeedQueryMedianMode"."female"
LOCAL ORDER BY "WithSeedQueryMedianMode"."female")
 AS "_twmVC0"
;
```

4.1. Calling DB Lytix™ Table UDF’s Using the Variable Creation Analysis Templates (With Query)

During the collaboration between the ADS engineering team and Fuzzy Logix, several enhancements were made. One key enhancement was in the area of specifying and parameterizing queries that used the “With Query” syntax. The ADS engineering team introduced the concept of an Analysis Template when creating a new Variable Creation Analysis. There are templates available for:

- Derived Tables
- Subqueries
- With Queries
- Recursive Queries
- Recursive View
- Union

The With Query Analysis Template can be specified in the Add New Analysis→Variable Creation dialogue as follows:

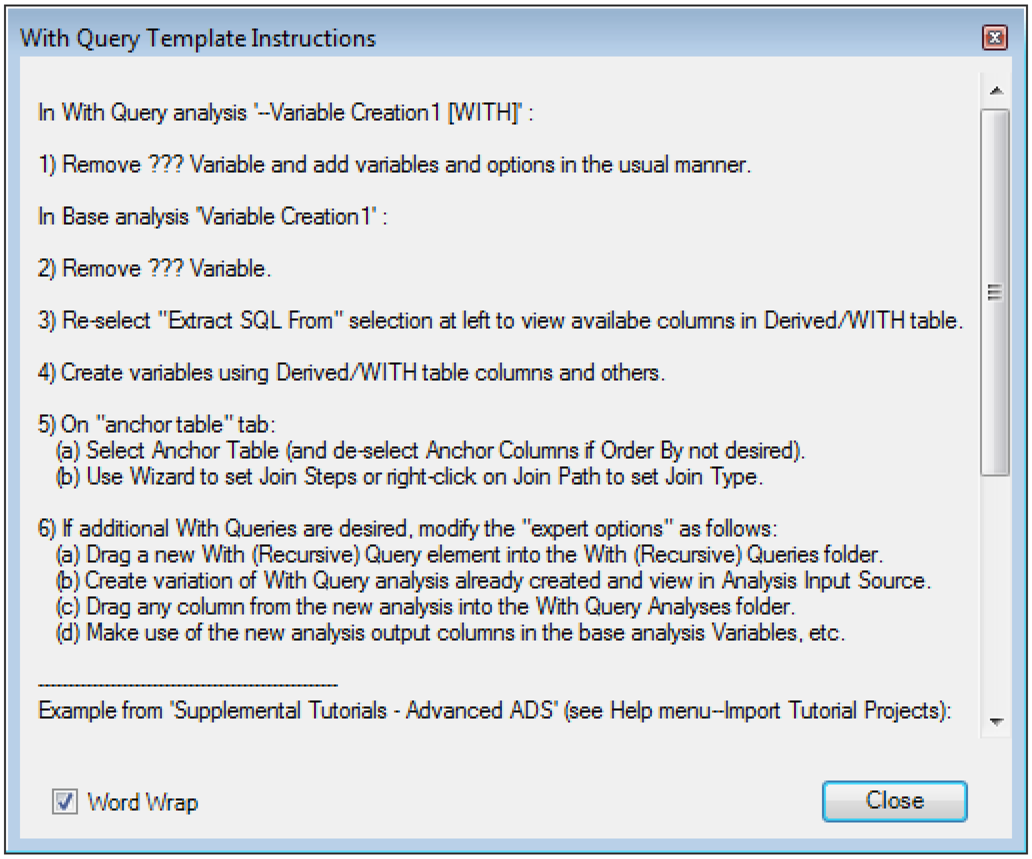


Analysis name: Variable Creation1

Analysis template: (none)

(none)
Query with Derived Table
Query with Subquery
With Query
Recursive Query
Recursive View
Union of Queries

When you select the "With Query" Analysis template, two Variable Creation analyses are created, one each for the With Query (named "--AnalysisName (WITH)") and the Table Function Query. They are parameterized in the manner described above with the "With Query" analysis Output parameters set to "Generate SQL only" and its analysis parameters set to "With Query" along with a template of data selected. The Variable Creation analysis is also parameterized with the Expert Option "With Recursive Clause" set, and a template of data selected through a Variable Reference to the "With Query" analysis. You also get the instructions for completing the template as follows:



In With Query analysis '--Variable Creation1 [WITH]' :

1) Remove ??? Variable and add variables and options in the usual manner.

In Base analysis 'Variable Creation1' :

2) Remove ??? Variable.

3) Re-select "Extract SQL From" selection at left to view availabe columns in Derived/WITH table.

4) Create variables using Derived/WITH table columns and others.

5) On "anchor table" tab:
(a) Select Anchor Table (and de-select Anchor Columns if Order By not desired).
(b) Use Wizard to set Join Steps or right-click on Join Path to set Join Type.

6) If additional With Queries are desired, modify the "expert options" as follows:
(a) Drag a new With (Recursive) Query element into the With (Recursive) Queries folder.
(b) Create variation of With Query analysis already created and view in Analysis Input Source.
(c) Drag any column from the new analysis into the With Query Analyses folder.
(d) Make use of the new analysis output columns in the base analysis Variables, etc.

Example from 'Supplemental Tutorials - Advanced ADS' (see Help menu--Import Tutorial Projects):

For purposes of the tutorial, step 4) equates to following the instructions in steps 9-11 and 14-15 in section 4. Step 5) and 6) do not apply to this Fuzzy Logix tutorial. The template does the rest for you!

5. <u>Calling Fuzzy Logix XSP's from Variable Creation</u>

The DB Lytix™ External Stored Procedures (XSP's) are called as pre-processing elements of a Variable Creation analysis known as "Run Units". Before describing the examples provided in the tutorial, a brief description of Run Units and Literal Parameters is given below.

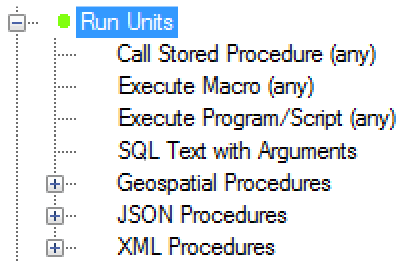
5.1. <u>Introduction to Variable Creation Run Units</u>

The term Run Unit represents one of the following "execution" SQL elements:

- Call Stored Procedure
- Execute Macro
- Execute Program/Script
- SQL Text with Arguments

Run Units have a category of their own in the SQL Elements tree, from which each of the Run Units elements may be dragged or selected onto the Run Units palette. When a Variable Creation analysis is executed, any Run Units defined on the Run Units tab are first executed in the order of appearance (unless they are marked "Skip"), and then if Variables are defined on the Variables tab, a query or sub-query is built or defined in the typical manner as described above.

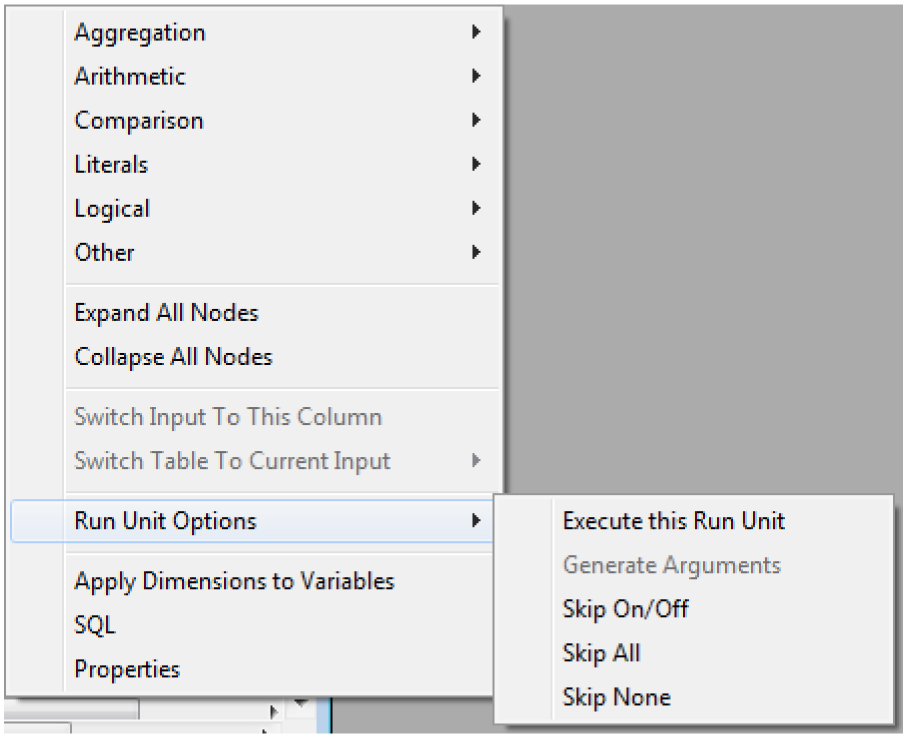
When you expand the Run Unit category of the SQL Elements tree, you see the following:



For purposes of the ADS/DB Lytix™ integration, the important run units are:

- Call Stored Procedure (any) – execute the DB Lytix™ XSP's, utilizing literal parameters to pass along resulting analysis ID's and/or volatile table names.
- SQL Text with Arguments – query to DB Lytix™ system tables populated via call to the XSP's using the analysis ID's, or query/persist the volatile tables created by a call
- Execute Program/Script (any) – execute Excel-based macros for visualizing the results of a DB Lytix™ XSP call.

As previously stated, Run Units are executed as pre-processing elements of the Variable Creation analysis, which means they are executed first when the Variable Creation analysis is executed. Optionally, Run Units have the following additional properties when you right-click on one:



- Execute this Run Unit
  A Run Unit may be executed by itself by using this option. Note that this option executes the analysis but inhibits other processing so that only this Run Unit is executed. It may not be used when the analysis or any of its Run Units is executing.

- Generate Arguments
  This option is available only when the Run Unit is of Call Stored Procedure or Execute Macro type and the Arguments folder under the Run Unit is empty. For Stored Procedures, Input arguments consist of Literal elements of the appropriate type (string, number, date etc.), while Input-Output and Output arguments consist of Literal Parameter elements of the appropriate type so that they may receive the final value of the argument after execution. Generated Macro arguments always consist of Literal elements of the appropriate type. (For more information about the use of Literal Parameter SQL elements in Call Stored Procedure Run Units, refer to the description of the Call Stored Procedure SQL Element.

- Skip On/Off
  This option toggles the selected Run Units Skip status on or off. (If on, the Run Unit name is preceded by “[Skip]”.)

- Skip All
  This option marks all Run Units as [Skip].

- Skip None
  This option removes any [Skip] status present on all Run Units.

For a very detailed description of Run Units, please refer to “Chapter 2 – Analytic Data Sets” in:

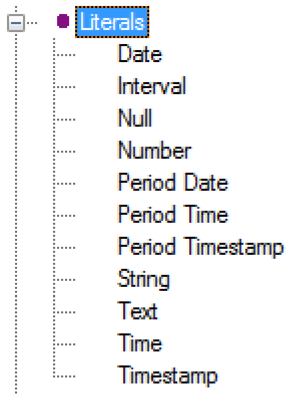
- Teradata Warehouse Miner User Guide - Volume 2 - ADS Generation Release 5.3.5 (B035-2301-064A, June 2014)

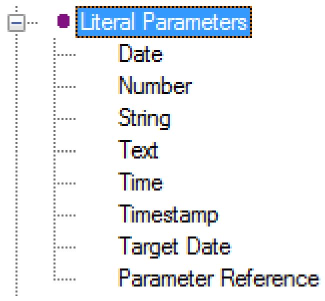
In specific the following sections:

- Variable Creation → INPUT → Variables → SQL Elements → Run Units, and
- Variable Creation → INPUT → Variables → Run Units

5.2. Introduction to Literals and Literal Parameters

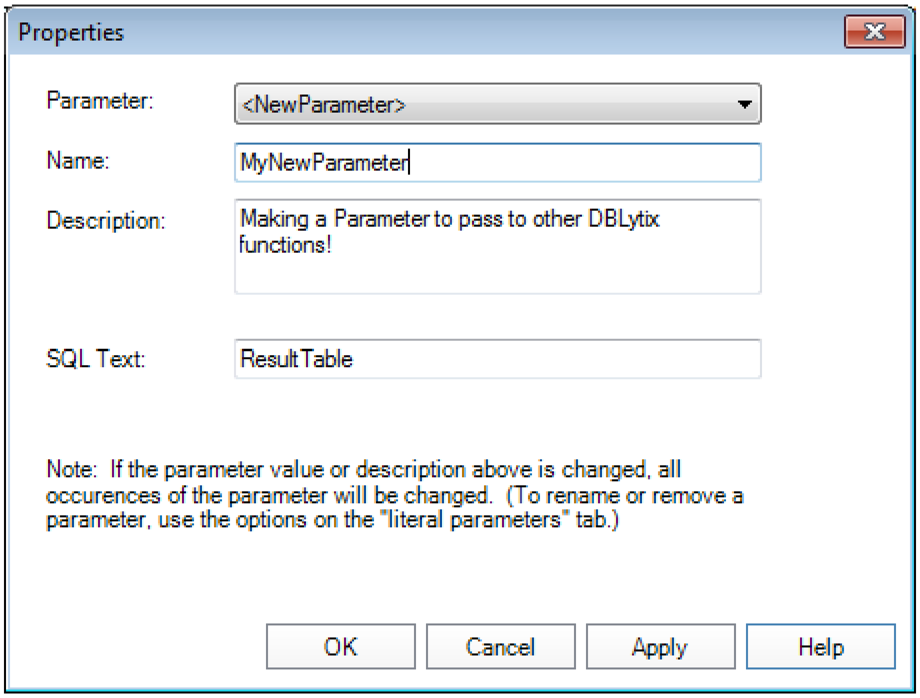
Typically the arguments to the DB Lytix™ XSP’s are either String (characters with single quotes around them in their entirety – single quotes within the string are escaped), Text (unquoted character), Null or Number “Literals” or String, Text, Number and Parameter Reference “Literal Parameters.” Valid values of each include:





Literals are hard-coded and can only be changed manually. Literal Parameters are dynamic and can change with each execution of the Variable Creation analysis or Run Units. Literal Parameters can be shared within a given analysis, or shared globally throughout a project by using the "Parameter Reference" type. Literal Parameters are what allows ADS Generator to manage the execution of and communication between the various DB Lytix™ XSP's, and SQL Statements that query either result tables (passing a volatile table name created by the XSP in a Literal Parameter) or the Fuzzy Logix System Tables (passing an analysis ID created by the XSP in a Literal Parameter), where analytic modeling results and statistics are stored.

Properties of String, Text and Number Literals are simply text boxes where hard-coded literal values can be either typed in, or dragged and dropped in some cases. Properties of Literal Parameters include:


Properties

Parameter: <NewParameter>

Name: MyNewParameter

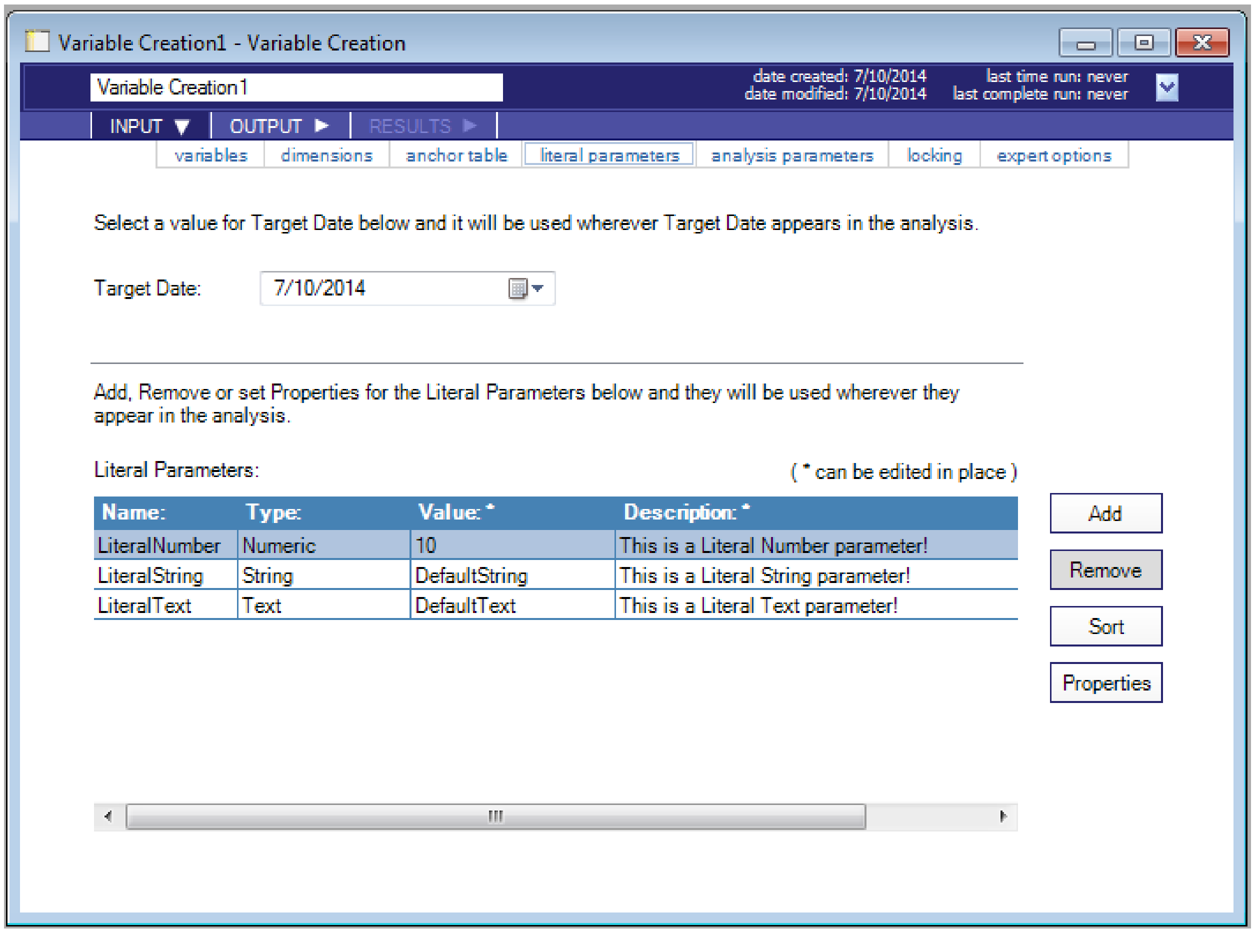
Description: Making a Parameter to pass to other DBLytix functions!

SQL Text: ResultTable

Note: If the parameter value or description above is changed, all occurences of the parameter will be changed. (To rename or remove a parameter, use the options on the "literal parameters" tab.)

By default, when adding a Literal Value of any kind, ADS attempts to create a new one (with a specification of "<NewParameter>" in the "Parameter:" pull-down). You provide a Name of the Literal Parameter, an optional Description, and the initial value of the String, Number, or in the case above SQL Text ("ResultTable"). This new Parameter "MyNewParameter" can be used now as an IN, OUT or INOUT parameter of an XSP call, in a SQL Text with Arguments Run Unit, or anywhere else within the Variable Creation Analysis. It can also be used in other analyses within your project by assigning it to a Parameter Reference.

As Literal Parameters are created within the Variable Creation analysis, they can be viewed under the "literal parameters" table under INPUT as follows:



Variable Creation1 - Variable Creation

Variable Creation1

date created: 7/10/2014 date modified: 7/10/2014 last time run: never last complete run: never

INPUT ▼ OUTPUT ▶ RESULTS ▶

variables | dimensions | anchor table | literal parameters | analysis parameters | locking | expert options

Select a value for Target Date below and it will be used wherever Target Date appears in the analysis.

Target Date: 7/10/2014

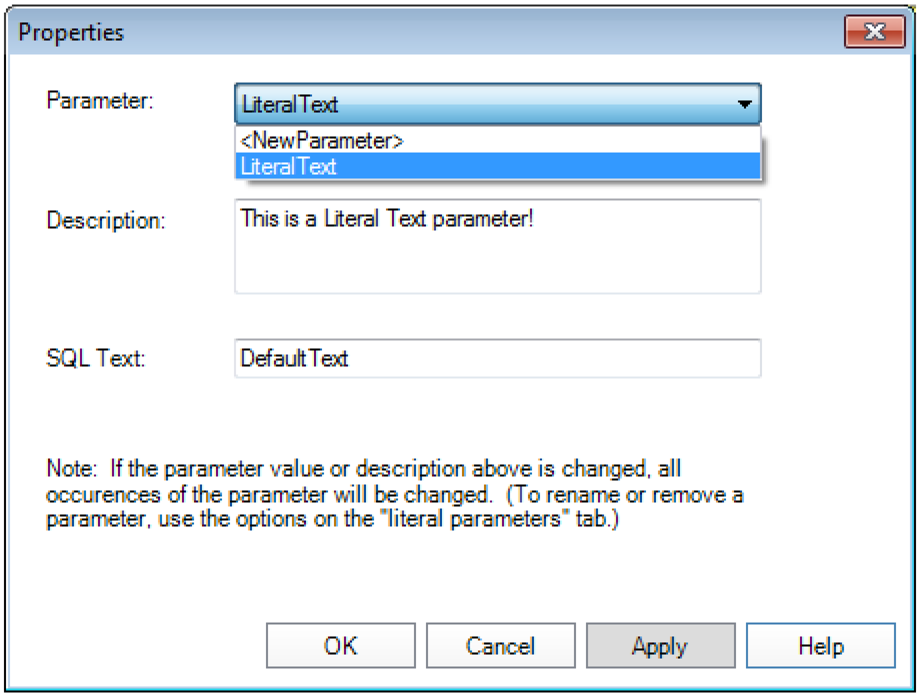
Add, Remove or set Properties for the Literal Parameters below and they will be used wherever they appear in the analysis.

Literal Parameters: (* can be edited in place)

Name:	Type:	Value: *	Description: *
LiteralNumber	Numeric	10	This is a Literal Number parameter!
LiteralString	String	DefaultString	This is a Literal String parameter!
LiteralText	Text	DefaultText	This is a Literal Text parameter!

In this case three Literal Parameters have been created for each of the types used for the calls to DB Lytix™ functions. You can add more here, remove one or more, sort or view/change them.

Once Literal Parameters have been created within the Variable Creation analysis, you can assign them directly to an argument within a Run Unit as follows:



Parameter: LiteralText

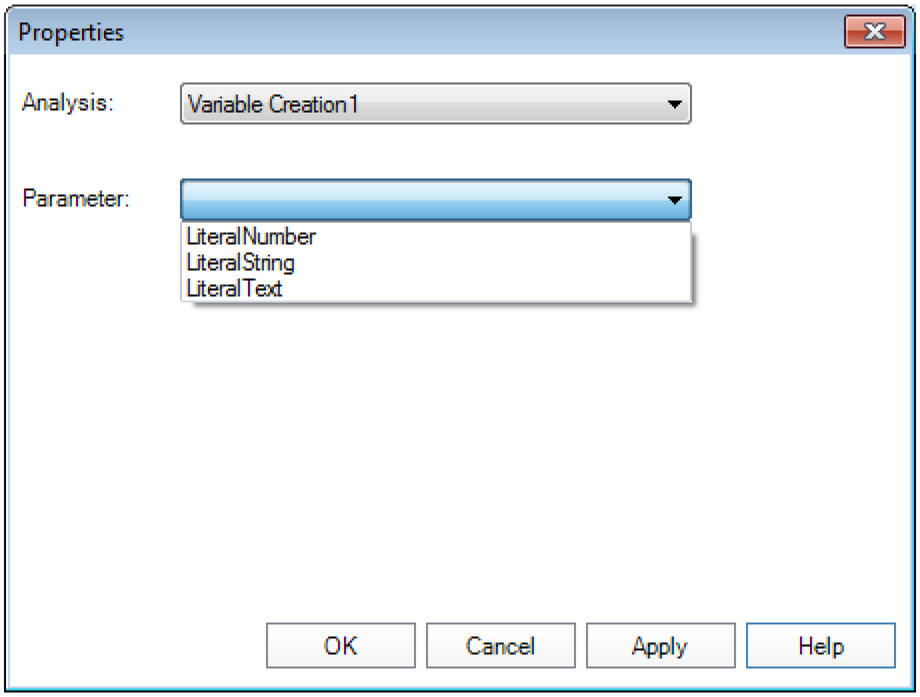
Description: This is a Literal Text parameter!

SQL Text: DefaultText

Note: If the parameter value or description above is changed, all occurences of the parameter will be changed. (To rename or remove a parameter, use the options on the "literal parameters" tab.)

Drag and drop a new Literal Parameter as an argument to a Run Unit. By default, you will see "<NewParameter>" in the "Parameter:" pull-down. If you pull down the list, you will see all of the created Literal Parameters of that same type (in this case Literal Text Parameters) – select the Literal Parameter needed for this Run Unit and it is now permanently linked to any analysis that uses this same Literal Parameter. If one analysis changes the value of this Literal Parameter, the change is reflected globally within the same Variable Creation analysis.

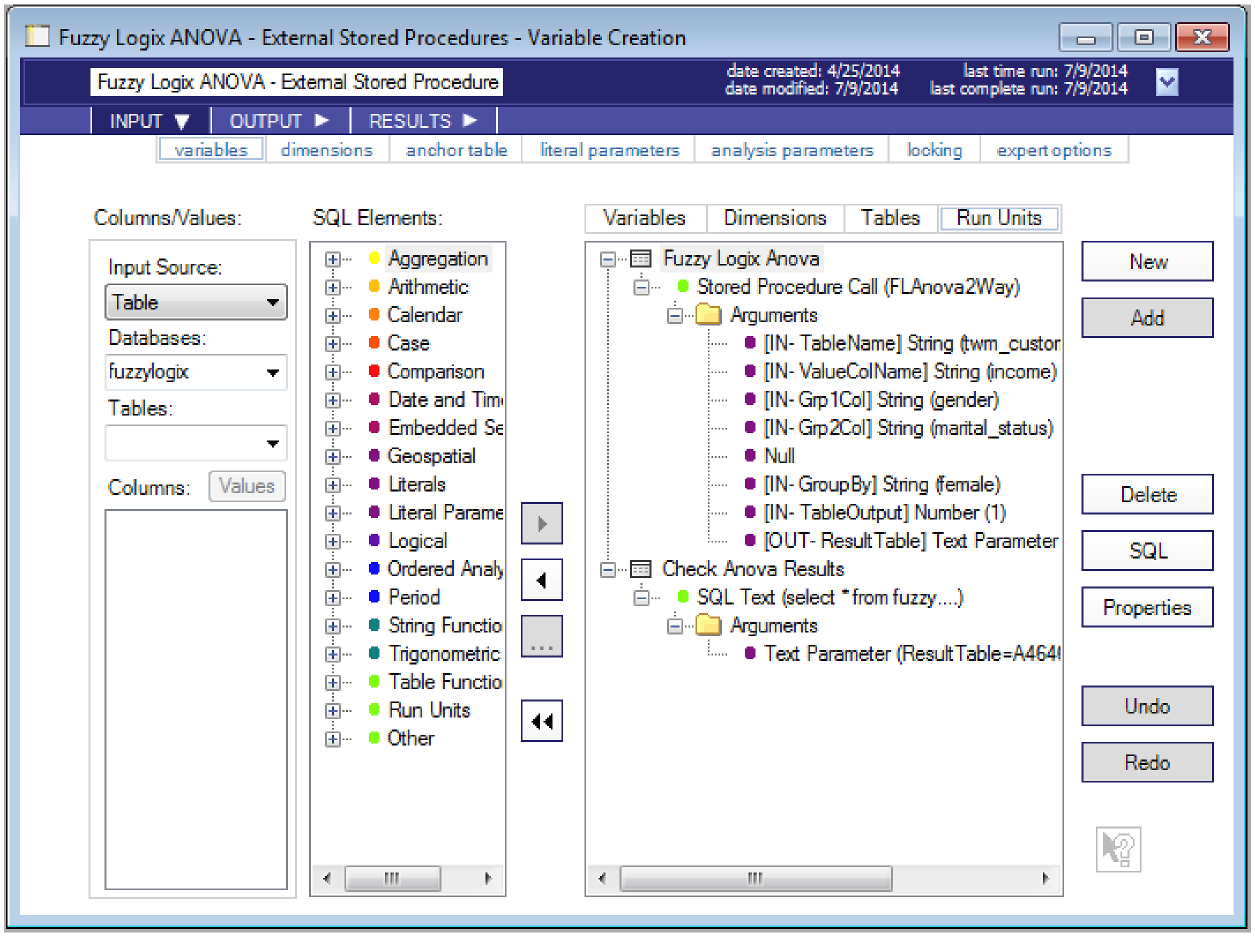
In order to make a Literal Parameter global within the entire project, use the "Parameter Reference" Literal Parameter type:



Select the Analysis where the Literal Parameter that you want to reference is defined, in the “Analysis:” pull-down. The “Parameter:” pull-down list is then populated with all the Literal Parameters defined within the analysis. Select the appropriated Literal Parameter and it will be synchronized with all the other analyses that both define or reference it.

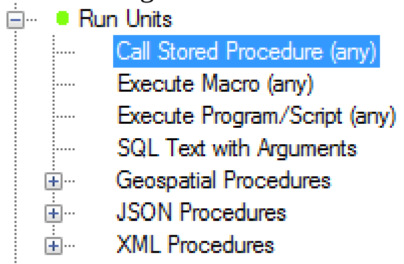
5.3. DB Lytix™ XSP’s - The Basics

First, we will start with an example of an XSP that works against a typical relational table (or “Wide” table as Fuzzy Logix refers to them). One such example is “FLAnova2Way” which performs two-way analysis of variance (ANOVA). Here is what the finished analysis looks like (“Fuzzy Logix ANOVA - External Stored Procedures” Variable Creation analysis in the tutorial):

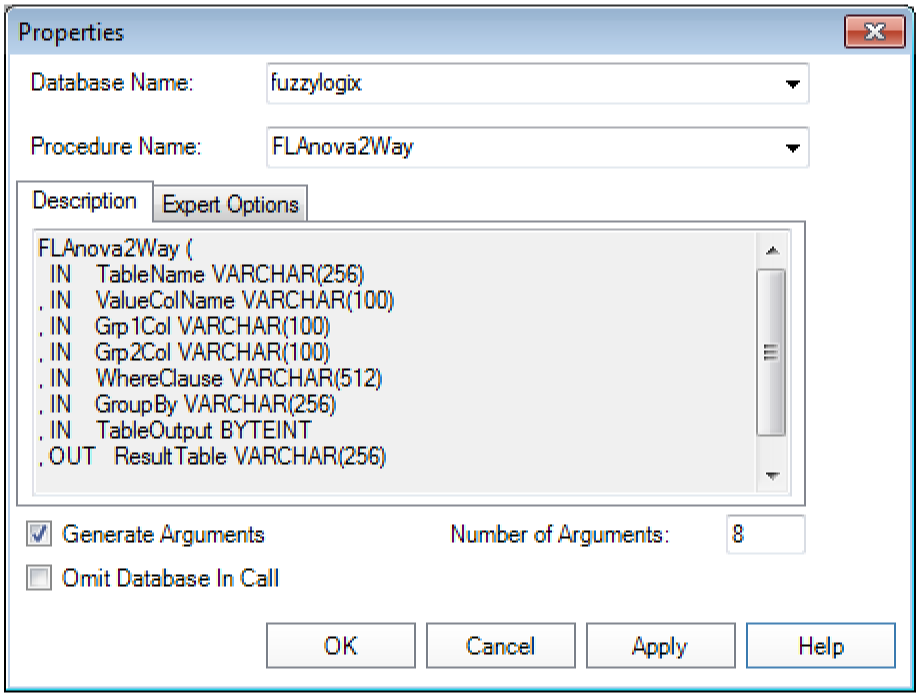


First, a Run Unit (Call Stored Procedure (any)) is added to the Run Unit palette and parameterized to execute the DB Lytix™ FLAnova2Way XSP. Then, another Run Unit (SQL Text with Arguments) is added to the Run Unit palette utilizing the OUT Parameter from the XSP call as an argument to the SQL query to view the results generated. Here are the steps to parameterize the analysis:

1) Click on the "+" icon next to "Run Units" in the "SQL Elements" list. It will expand and show the following elements:



Select "Call Stored Procedure (any)" by dragging and dropping it on to the Run Units palette or highlighting it, and double-clicking or using the ">" button. This action will bring up the following dialogue:



Database Name: fuzzylogix

Procedure Name: FLAnova2Way

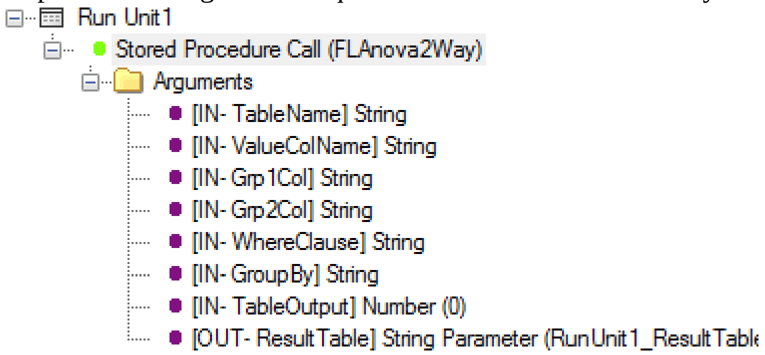
```
FLAnova2Way (
  IN   TableName VARCHAR(256)
, IN   ValueColName VARCHAR(100)
, IN   Grp1Col VARCHAR(100)
, IN   Grp2Col VARCHAR(100)
, IN   WhereClause VARCHAR(512)
, IN   GroupBy VARCHAR(256)
, IN   TableOutput BYTEINT
, OUT  ResultTable VARCHAR(256)
```

☑ Generate Arguments   Number of Arguments: 8

☐ Omit Database In Call

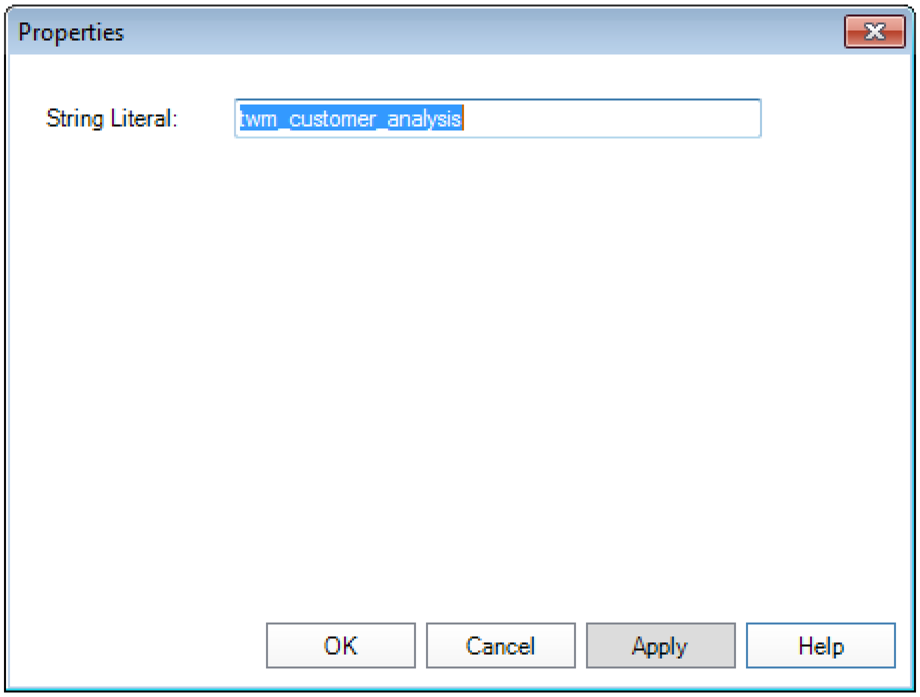
Select the database where DB Lytix™ is installed in the "Database Name:" pull-down. Once you do this, the "Procedure Name" pull-down will be populated with the Stored Procedures that reside in the selected database. It will also show the signature of the function under the "Description" tab. Select FLAnova2Way in the "Procedure Name:" pull-down to view the signature. By default, ADS Generator will fully qualify the UDF call by prepending it with the database name it resides in. You can turn this default behavior off by enabling the "Omit Database In Call" check-box.

2) Click the "Generate Arguments" checkbox and hit OK or Apply. This option will create a template of the arguments required to call the FLAnova2Way XSP as follows:



```
Run Unit1
  Stored Procedure Call (FLAnova2Way)
    Arguments
      [IN- TableName] String
      [IN- ValueColName] String
      [IN- Grp1Col] String
      [IN- Grp2Col] String
      [IN- WhereClause] String
      [IN- GroupBy] String
      [IN- TableOutput] Number (0)
      [OUT- ResultTable] String Parameter (RunUnit1_ResultTabl
```

Note that this is truly a template, providing a starting point for parameterizing the XSP call. Next, start entering values for each of the arguments. Do this by highlighting the first argument ("TableName") and hitting the "Properties" button, or double clicking on it:



String Literal: twm_customer_analysis

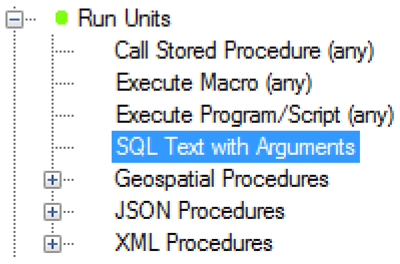
Enter the table name "twm_customer_analysis" and click on Apply, or simply click on the next argument to enter the next String Literal. Repeat for the remaining literal values:

- "ValColName" = income
- "Grp1Col" = gender
- "Grp2Col" = marital_status
- "WhereByClause" = Null (i.e. drag and drop a Literal Value "Null" on top of argument)
- "GroupBy" = female
- "TableOutput" = 1

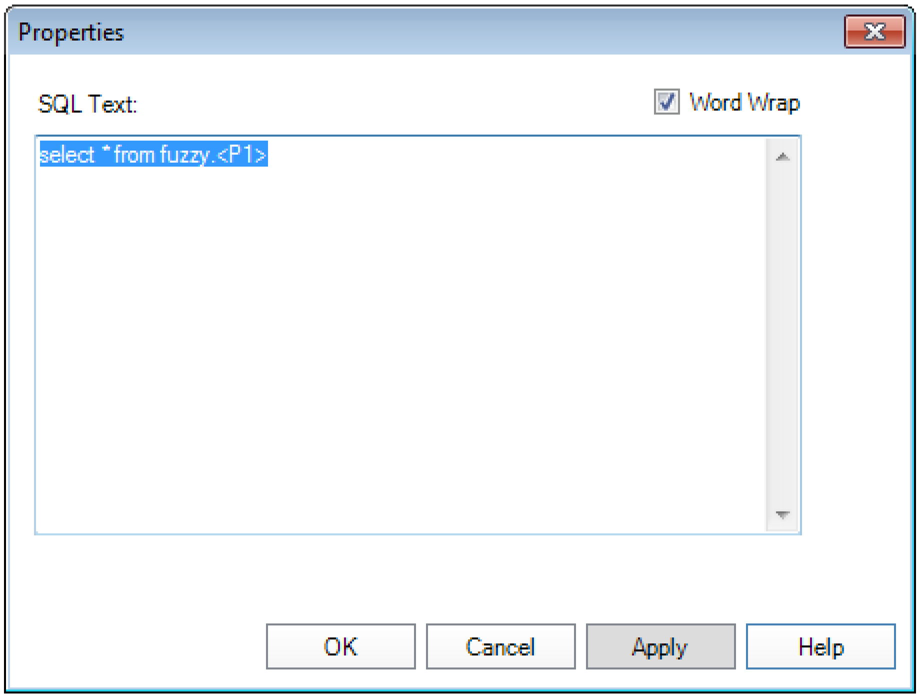
Note that the last XSP argument, the OUT parameter, has been specified as a "String Parameter." This is because the XSP returns a dynamically generated volatile table name or analysis identifier as its OUT parameter (since "TableOutput" is set to 1, it will be a volatile table in this case). This needs to be a Literal Parameter so we can use it in subsequent Run Units to query or persist the volatile table.

By default, ADS Generator constructs the argument list such that OUT XSP parameters of type VARCHAR(n) are always generated as type a "String Parameter." In the case when the XSP is generating a volatile table, you need to change this to a "Text Parameter," so it is not quoted and can be used within a SQL query. To do this simply drag and drop a "Text" Literal Parameter on top of the existing String Parameter ("[Out – ResultTable]" in the example above) and create a new Text Literal Parameter using the instructions given above. In the example given, the name of the Text Literal Parameter created was also "ResultTable."

3) Next, create another Run Unit, this time of type "SQL Text with Arguments" in order to query the result set created by the call to FLAnova2Way, using the "ResultTable" literal parameter set by the previous Run Unit:
   a. Click on the "+" icon next to "Run Units" in the "SQL Elements" list. It will expand and show the following elements:



Select “SQL Text with Arguments” by dragging and dropping it on to the Run Units palette or highlighting it, and double-clicking or using the “>” button. This action will bring up the following dialogue:

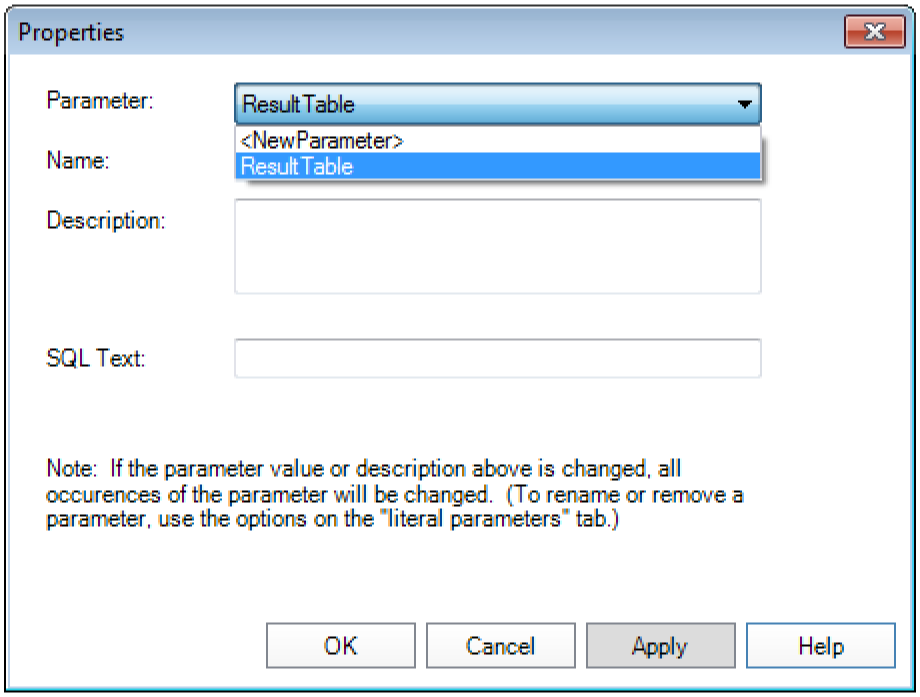


SQL Text:

```
select * from fuzzy.<P1>
```

Type in “select * from fuzzy.<P1>” (the database name is the user name/containing database as this is a volatile table). <P1> is used as a placeholder for the first literal parameter that will be specified as described below.

b. Next, drag and drop a “Text” Literal Parameter into the Arguments folder of the Run Unit. This will bring up the following dialogue:


Properties

Parameter: ResultTable

<NewParameter>
ResultTable

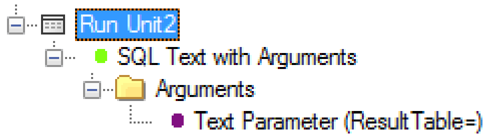
Name:

Description:

SQL Text:

Note: If the parameter value or description above is changed, all occurences of the parameter will be changed. (To rename or remove a parameter, use the options on the "literal parameters" tab.)

Select “ResultTable” from the “Parameter:” pull-down list. This Literal Parameter maps to the value of <P1> in the SQL Text. Multiple Literal Parameters can be referenced as <P2>, <P3>, … , <Pn). Initially this Run Unit will look like this:



4) Optionally, you can change the name of the Run Units by highlighting them with a single click and hold (Windows style rename) and changing the name. Optionally, highlight the Run Unit and click on the “Properties” button to do this within the Properties dialogue. Run Unit1 was renamed to “Fuzzy Logix Anova” and Run Unit2 was renamed to “Check Anova Results” in the example provided.

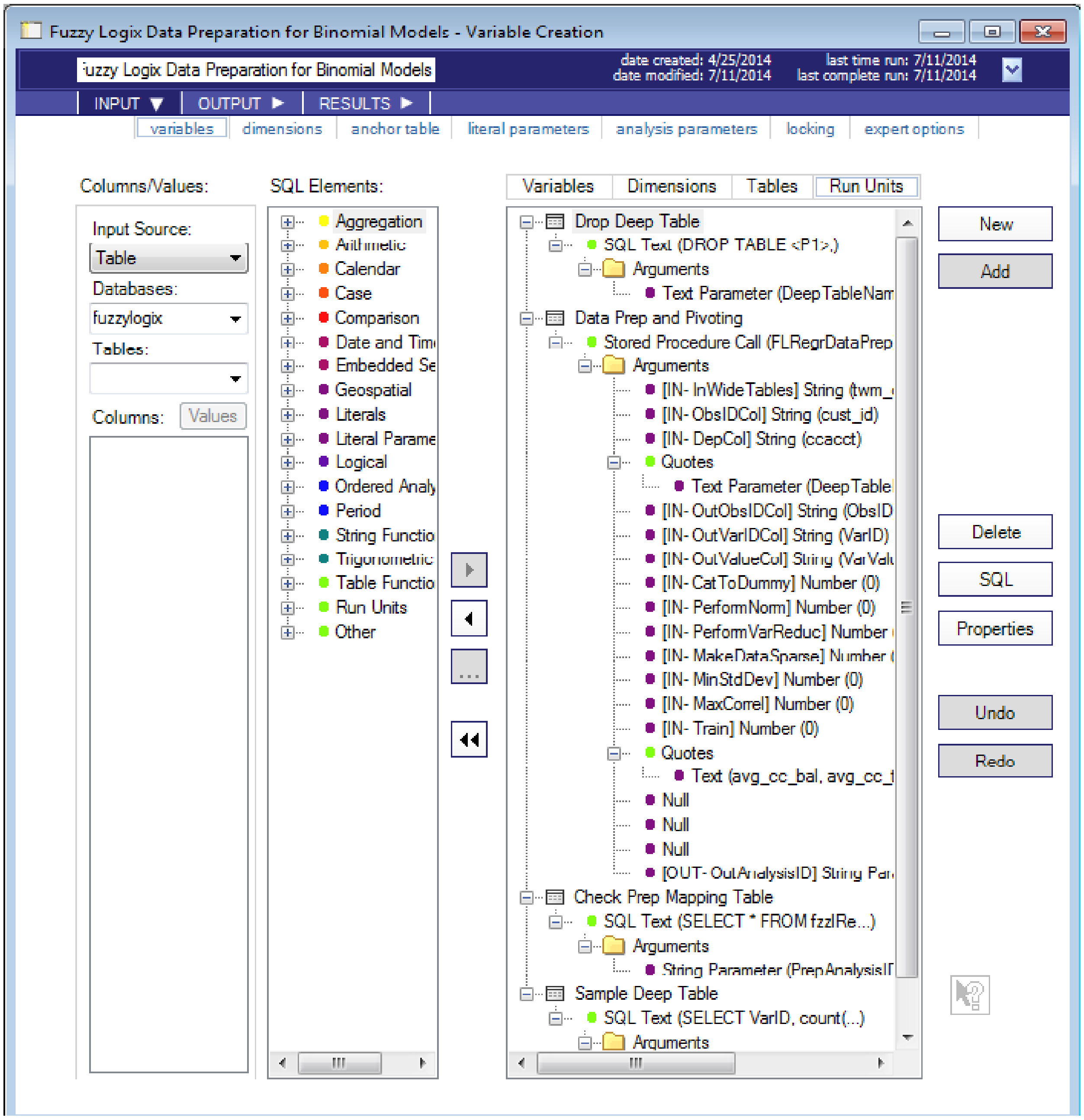
5) Now execute the Variable Creation Analysis by clicking on the “Run” button directly under the “Tools” menu item, or highlighting the Variable Creation analysis in the “Project Explorer,” right-clicking and selecting “Run.” Alternately, since there are only Run Units within this particular Variable Creation Analysis, the individual Run Units can be executed by right clicking on them and selecting the “Execute this Run Unit” option.

   As each Run Unit executes, the Run Unit name will be proceeded by “[Executing].” Note that as soon as the Run Unit that calls FLAnova2Way completes, its Text Literal Parameter is set to the return value of the XSP OUT argument. This is automatically reflected in the SQL Text Run Unit. In the example the value is “<AnalysisID>_Anova2Way.”

   Once the execution is “Complete” in the “Execution Status” window, click on the “Results→data” tab and you should see 2 result sets – the first being the OUT argument of the XSP and the second is the results of the SQL Text query - the results of the FLAnova2Way XSP (see the Hypothesis Tests → FLAnova2Way section in the User Manual for DB Lytix™ on Teradata Advanced Package v1.0.1 for a complete description of the results generated).

5.4. DB Lytix™ Data Preparation XSP's

Next we will see an example of an XSP that prepares a typical relational table or tables (or "Wide" table as Fuzzy Logix refers to them) into a form required by the DB Lytix™ Data Mining Functions (or "Deep" tables as Fuzzy Logix refers to them). The most comprehensive way of performing this data pivoting operation is to use the DB Lytix™ "FLRegrDataPrep" XSP and then analyze the results. Here is what one of finished example analyses looks like ("Fuzzy Logic Data Preparation for Binomial Models" Variable Creation analysis in the tutorial):



Fuzzy Logix Data Preparation for Binomial Models

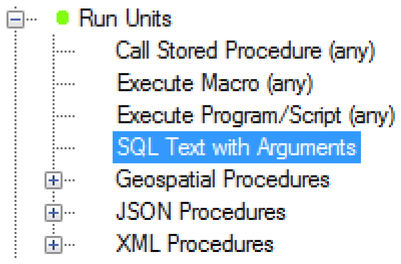
date created: 4/25/2014
date modified: 7/11/2014
last time run: 7/11/2014
last complete run: 7/11/2014

Input Source: Table
Databases: fuzzylogix
Tables:
Columns:

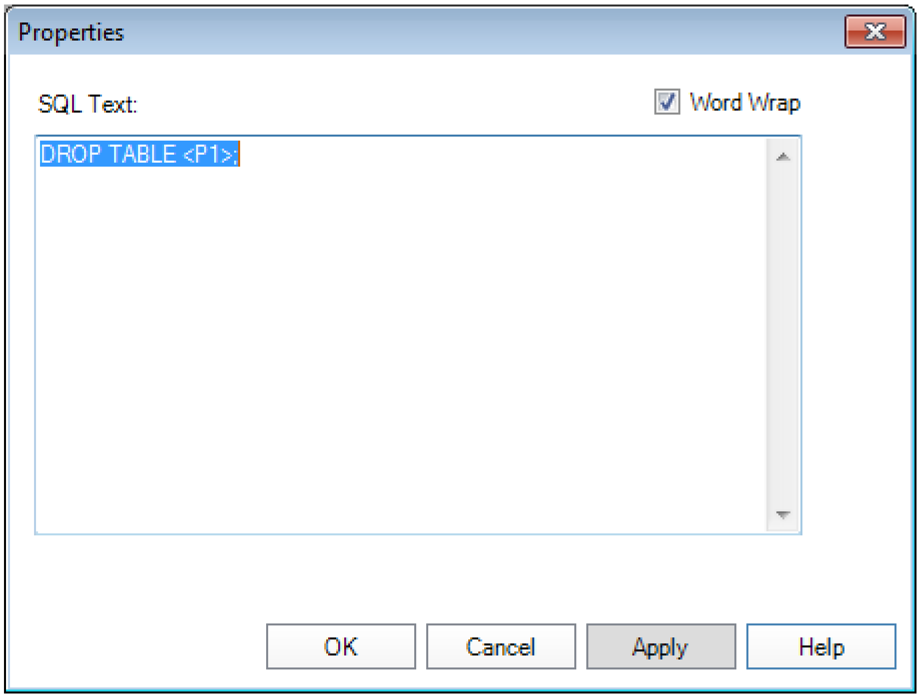
```
Drop Deep Table
    SQL Text (DROP TABLE <P1>,)
        Arguments
            Text Parameter (DeepTableNam
Data Prep and Pivoting
    Stored Procedure Call (FLRegrDataPrep
        Arguments
            [IN- InWideTables] String (twm_
            [IN- ObsIDCol] String (cust_id)
            [IN- DepCol] String (ccacct)
            Quotes
                Text Parameter (DeepTable
            [IN- OutObsIDCol] String (ObsID
            [IN- OutVarIDCol] String (VarID)
            [IN- OutValueCol] String (VarValu
            [IN- CatToDummy] Number (0)
            [IN- PerformNorm] Number (0)
            [IN- PerformVarReduc] Number
            [IN- MakeDataSparse] Number
            [IN- MinStdDev] Number (0)
            [IN- MaxCorrel] Number (0)
            [IN- Train] Number (0)
            Quotes
                Text (avg_cc_bal, avg_cc_t
            Null
            Null
            Null
            [OUT- OutAnalysisID] String Par
Check Prep Mapping Table
    SQL Text (SELECT * FROM fzzlRe...)
        Arguments
            String Parameter (PrepAnalysisI
Sample Deep Table
    SQL Text (SELECT VarID, count(...)
        Arguments
```

First, a Run Unit (SQL Text with Arguments) is added to the Run Unit palette to DROP a table created by the second Run Unit. This Run Unit (Call Stored Procedure (any)) is added to the Run Unit palette and parameterized to call the FLRegrDataPrep XSP. Finally, two other Run Units (SQL Text with Arguments) are added to the Run Unit palette utilizing the OUT Parameter from the XSP call as an argument to the SQL query to view results. Here are the steps to parameterize the analysis:

1) Click on the "+" icon next to "Run Units" in the "SQL Elements" list. It will expand and show the following elements:



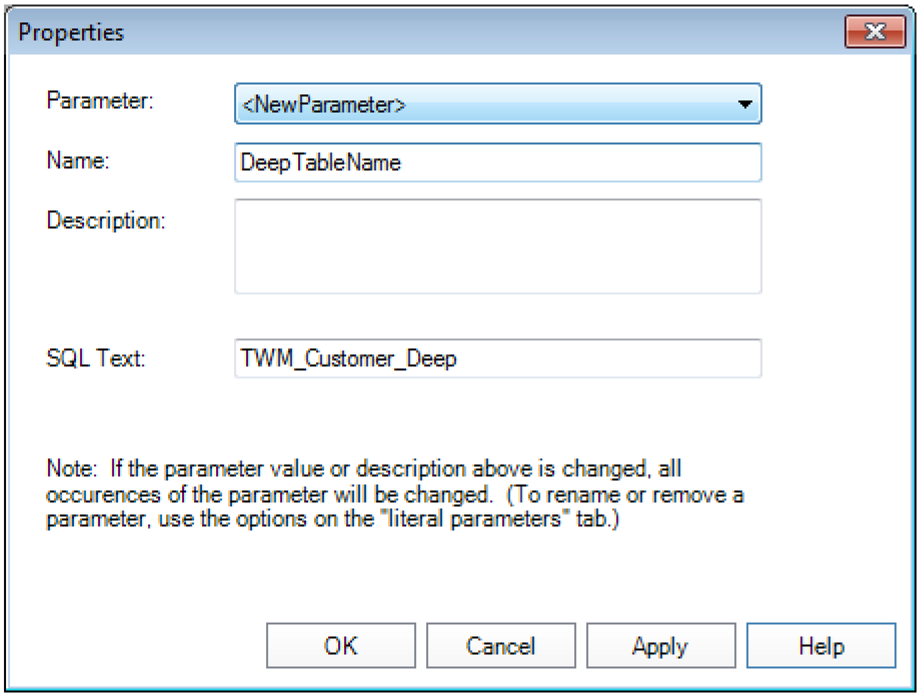
Select "SQL Text with Arguments" by dragging and dropping it on to the Run Units palette or highlighting it, and double-clicking or using the ">" button. This action will bring up the following dialogue:



SQL Text:

```
DROP TABLE <P1>;
```

In this case, we are going to use a Static Literal Parameter. It is convenient to do so as this same parameter will be used in several other Run Units. Drag and Drop a Text Literal Parameter on to the SQL Text with Arguments folder as follows:


Properties

Parameter: <NewParameter>

Name: DeepTableName

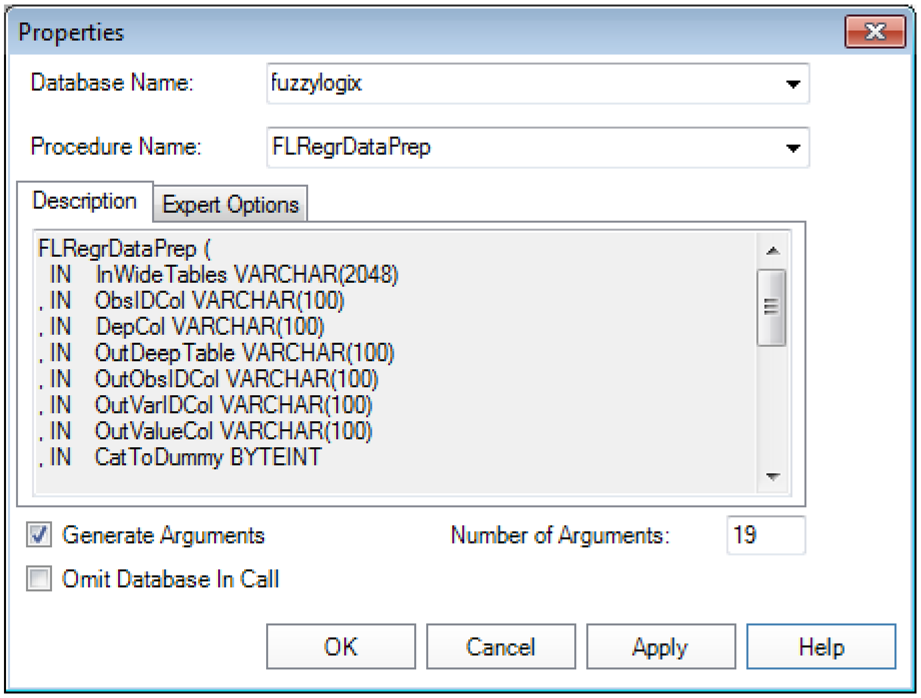
Description:

SQL Text: TWM_Customer_Deep

Note: If the parameter value or description above is changed, all occurences of the parameter will be changed. (To rename or remove a parameter, use the options on the "literal parameters" tab.)

Type in “DeepTableName” in the “Name:” text box. Note that TWM_Customer_Deep is the table created by the next Run Unit and queried in subsequent Run Units.

2) Next create another Run Unit – this time “Call Stored Procedure (any)” – by dragging and dropping it on to the Run Units palette or highlighting it, and double-clicking or using the “>” button. This action will bring up the following dialogue:


Properties

Database Name: fuzzylogix

Procedure Name: FLRegrDataPrep

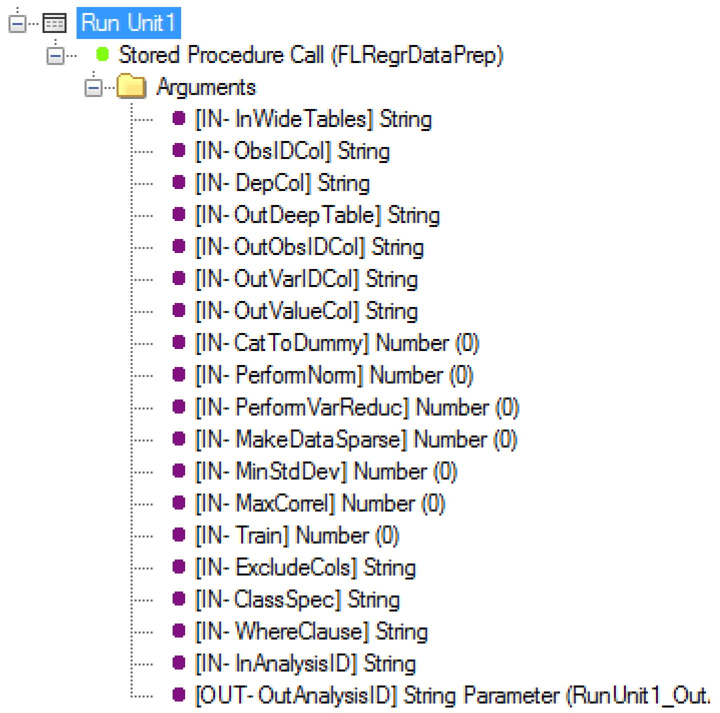
```
FLRegrDataPrep (
  IN   InWideTables VARCHAR(2048)
, IN   ObsIDCol VARCHAR(100)
, IN   DepCol VARCHAR(100)
, IN   OutDeepTable VARCHAR(100)
, IN   OutObsIDCol VARCHAR(100)
, IN   OutVarIDCol VARCHAR(100)
, IN   OutValueCol VARCHAR(100)
, IN   CatToDummy BYTEINT
```

[✓] Generate Arguments   Number of Arguments: 19

[ ] Omit Database In Call

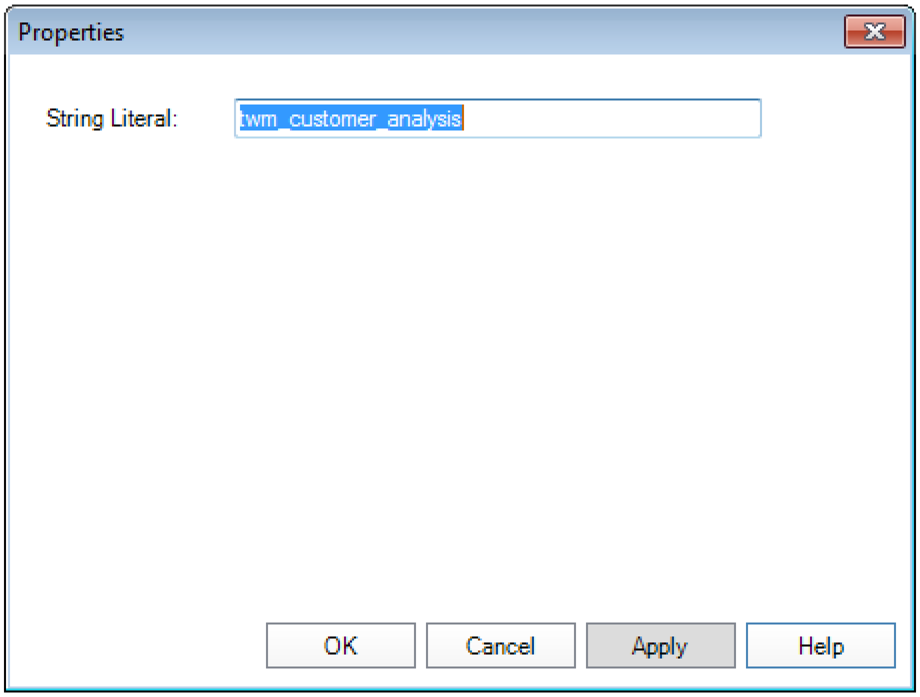
Select the database where DB Lytix™ is installed in the "Database Name:" pull-down. Once you do this, the "Procedure Name:" pull-down will be populated with the Stored Procedures that reside in the selected database. It will also show the signature of the function under the "Description" tab. Select FLRegrDataPrep in the "Procedure Name:" pull-down to view the signature. By default, ADS Generator will fully qualify the UDF call by prepending it with the database name it resides in. You can turn this default behavior off by enabling the "Omit Database In Call" check-box.

3) Click the "Generate Arguments" checkbox and hit OK or Apply. This option will create a template of the arguments required to call the FLRegrDataPrep XSP as follows:



```
Run Unit1
    Stored Procedure Call (FLRegrDataPrep)
        Arguments
            [IN- InWideTables] String
            [IN- ObsIDCol] String
            [IN- DepCol] String
            [IN- OutDeepTable] String
            [IN- OutObsIDCol] String
            [IN- OutVarIDCol] String
            [IN- OutValueCol] String
            [IN- CatToDummy] Number (0)
            [IN- PerformNorm] Number (0)
            [IN- PerformVarReduc] Number (0)
            [IN- MakeDataSparse] Number (0)
            [IN- MinStdDev] Number (0)
            [IN- MaxCorrel] Number (0)
            [IN- Train] Number (0)
            [IN- ExcludeCols] String
            [IN- ClassSpec] String
            [IN- WhereClause] String
            [IN- InAnalysisID] String
            [OUT- OutAnalysisID] String Parameter (RunUnit1_Out.
```

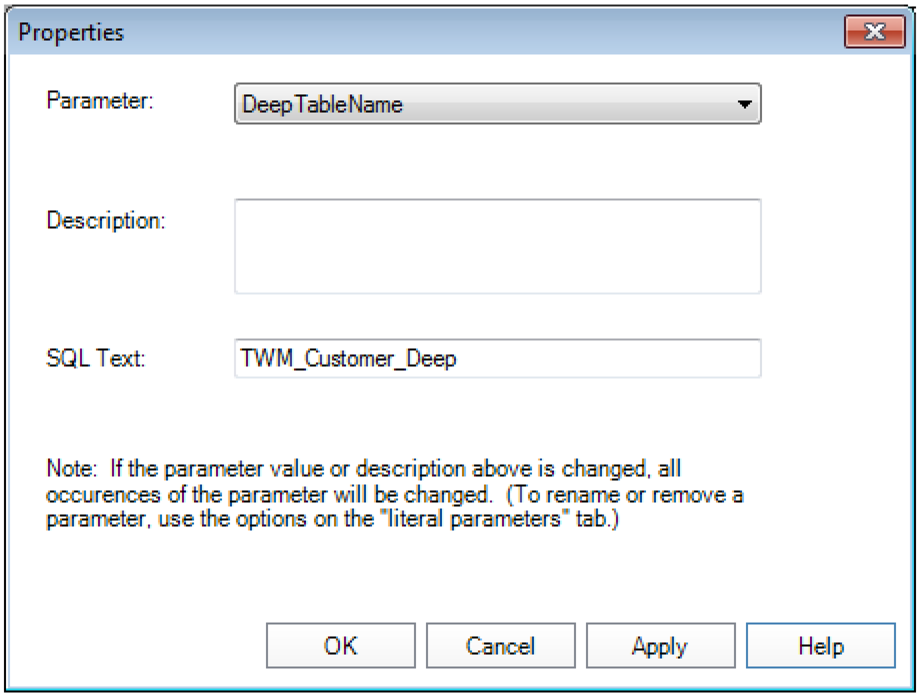
Next, start entering values for each of the arguments. Do this by highlighting the first argument ("InWideTables") and hitting the "Properties" button, or double clicking on it:


Properties

String Literal: twm_customer_analysis

Enter “twm_customer_analysis” and hit Apply, or simply click on the next argument to enter the next String Literal. Repeat for the remaining literal values:

- “ObsIdCol” = cust_id
- “DepCol” = ccacct
- For the “OutDeepTable” argument, , you can do either of the following:
  a. Specify explicitly the DeepTableName parameter value as "OutDeepTable" = TWM_Customer_Deep, or
  b. Replace the default generated String Literal with a Text Literal by dragging and dropping a Text Literal Parameter on top of “[IN – OutDeepTable] String.” Then Select “DeepTableName” in the “Parameter:” pull-down when the following dialogue is brought up:

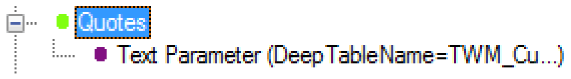


Parameter: DeepTableName

SQL Text: TWM_Customer_Deep

Note: If the parameter value or description above is changed, all occurences of the parameter will be changed. (To rename or remove a parameter, use the options on the "literal parameters" tab.)

Since this argument needs to be a quoted as it is passed to the XSP as VARCHAR, and Text Literals are not quoted, use SQL Element Other→Quotes by dragging and dropping it on to the populated Text Literal as follows:



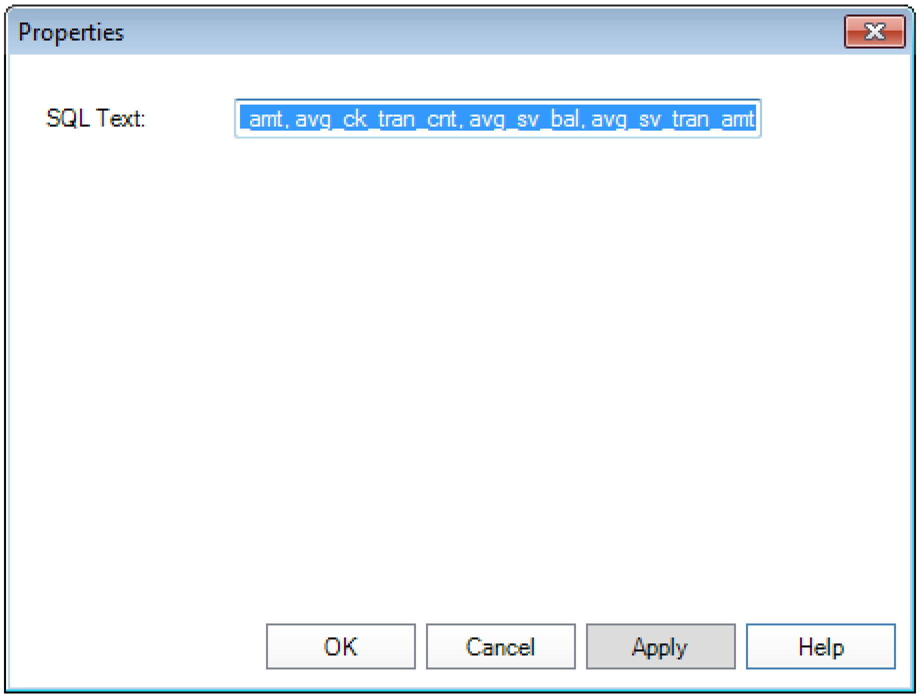
Quotes
Text Parameter (DeepTableName=TWM_Cu...)

Continue parameterizing the remainder of the arguments as follows:

- "OutObsIdCol" = ObsId
- "OutVarIdCol" = VarId
- "OutValueCol" = VarValue
- "CatToDummy" = 0
- "PerformNorm" = 0
- "PerformVarReduc" = 0
- "MakeDataSparse" = 0
- "MinStdDev" = 0
- "MaxCorrel" = 0
- "Train" = 0
- For the "ExcludeCols" argument, replace the default generated String Literal with a Text Literal by dragging and dropping a Text Literal Value on top of "[IN – ExcludeCols] String." Then from the "Columns/Values:" area, select "twm_customer_analysis" in the "Table:" pull-down list and highlight the following columns (multi-highlight using CTRL-Click):
  a. avg_cc_bal,
  b. avg_cc_tran_amt,
  c. avg_cc_tran_cnt,
  d. cc_rev,
  e. city_name,

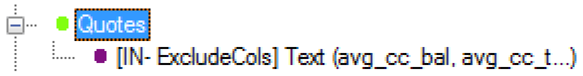
f. gender,
g. marital_status,
h. state_code

Then drag and drop these columns on top of the "[IN – ExcludeCols] Text" argument. This populates the Text Literal Properties Dialogue box as follows:



SQL Text: amt, avg_ck_tran_cnt, avg_sv_bal, avg_sv_tran_amt

Since this argument needs to be a list of quoted columns, and Text Literals are not quoted, use SQL Element Other→Quotes by dragging and dropping it on to the populated Text Literal as follows:

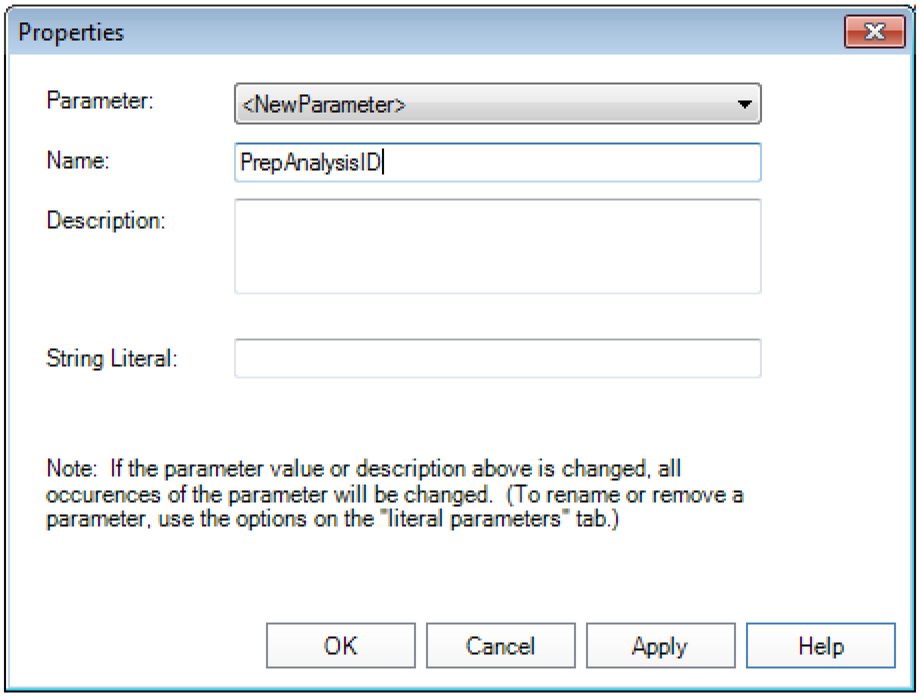


Quotes
[IN- ExcludeCols] Text (avg_cc_bal, avg_cc_t...)

Set the next three IN arguments to Null by dragging and dropping a Literal Null value on top of:

- "ClassSpec" = Null
- "WhereClause" = Null
- "InAnalysisID" = Null

Note that the last XSP argument, the OUT parameter, has been specified as a "String Parameter." In this case, we want it to be a quoted String Parameter as it is an Analysis Identifier that will be used within a WHERE clause in subsequent SQL Text Run Units. In this example, a new String Parameter has been created with the name "PrepAnalysisID" as follows:

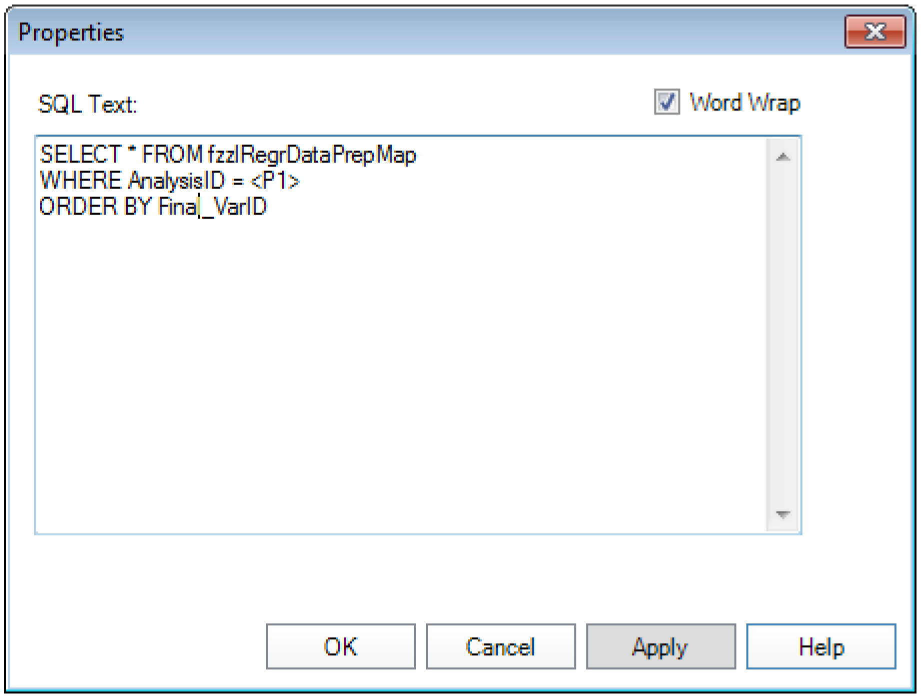


Parameter: <NewParameter>

Name: PrepAnalysisID

Note: If the parameter value or description above is changed, all occurences of the parameter will be changed. (To rename or remove a parameter, use the options on the "literal parameters" tab.)

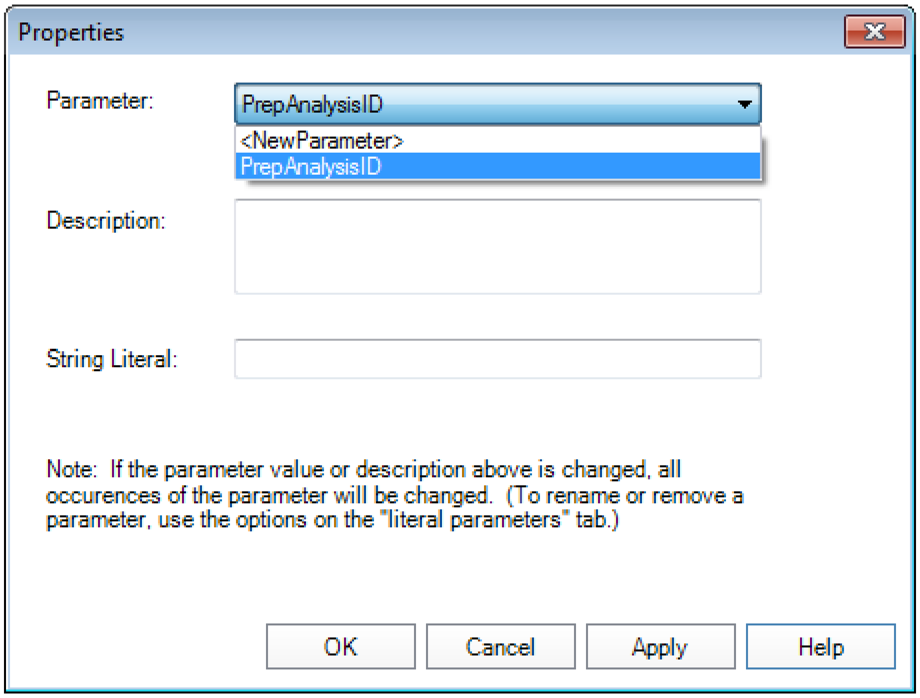
4) Next, create another Run Unit, this time of type "SQL Text with Arguments" in order to query the DB Lytix™ System tables, populated by the call to FLRegrDataPrep. Do this by dragging and dropping the "SQL Text with Arguments" Run Unit on to the Run Units palette or highlighting it, and double-clicking or using the ">" button. This action will bring up the following dialogue:



Word Wrap [✓]

```
SELECT * FROM fzzlRegrDataPrepMap
WHERE AnalysisID = <P1>
ORDER BY Final_VarID
```

For this Run Unit, the literal parameter being used is the OUT parameter from the FLRegrDataPrep call, "PrepAnalysisID." Drag and drop a String Literal Parameter in this Run Units argument folder as follows:


Properties

Parameter: PrepAnalysisID

<NewParameter>
PrepAnalysisID

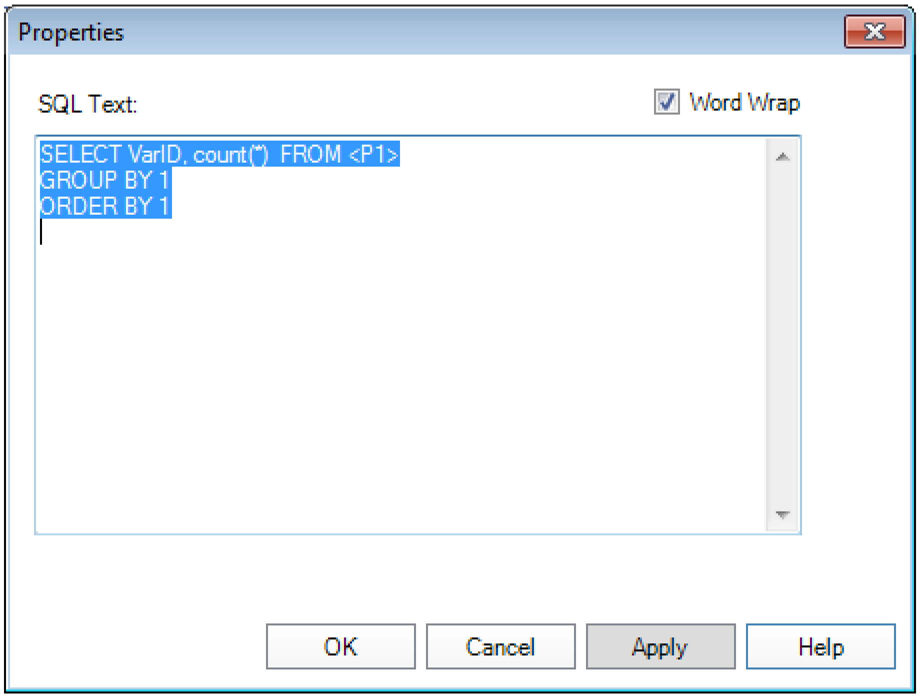
Description:

String Literal:

Note: If the parameter value or description above is changed, all occurences of the parameter will be changed. (To rename or remove a parameter, use the options on the "literal parameters" tab.)

And Select "PrepAnalysisID" in the "Parameter:" pull-down.

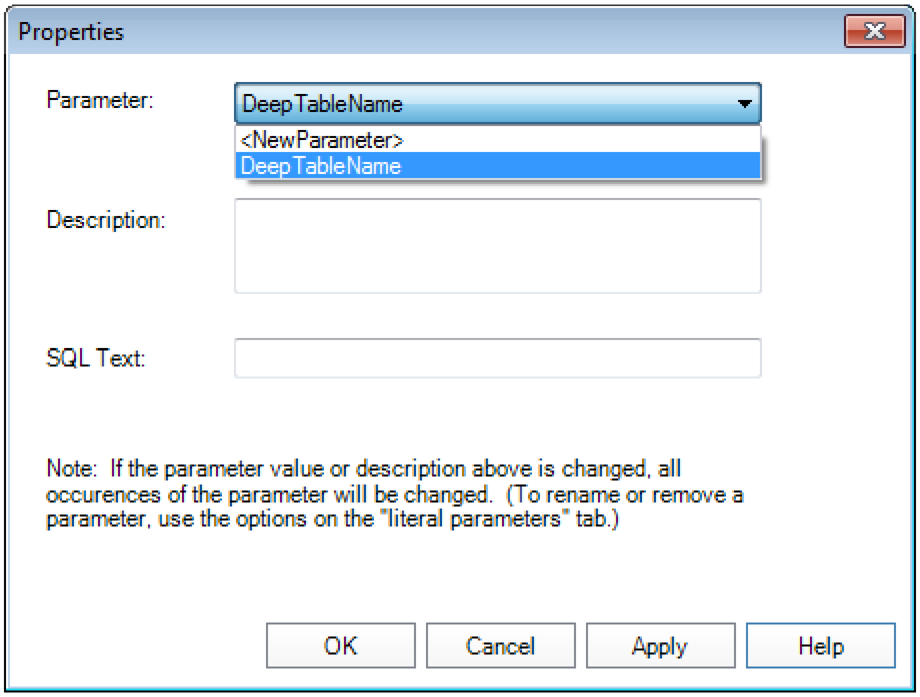
5) Finally, create another Run Unit, this time of type "SQL Text with Arguments" in order to query the resulting "Deep Table" created by the call to FLRegrDataPrep. Do this by dragging and dropping the "SQL Text with Arguments" Run Unit on to the Run Units palette or highlighting it, and double-clicking or using the ">" button. This action will bring up the following dialogue:


Properties

SQL Text: [✓ Word Wrap]

```
SELECT VarID, count(*)  FROM <P1>
GROUP BY 1
ORDER BY 1
```

For this Run Unit, the literal parameter being used is the static Text Literal parameter created in the first Run Unit, “DeepTableName.” Drag and drop a Text Literal Parameter in this Run Units argument folder as follows:



Note: If the parameter value or description above is changed, all occurences of the parameter will be changed. (To rename or remove a parameter, use the options on the "literal parameters" tab.)

And Select “DeepTableName” in the “Parameter:” pull-down.

6) Now execute the Variable Creation Analysis by clicking on the “Run” button directly under the “Tools” menu item, or highlighting the Variable Creation analysis in the “Project Explorer,” right-clicking and selecting “Run.” Alternately, since there are only Run Units within this particular Variable Creation Analysis, the individual Run Units can be executed by right clicking on them and selecting the “Execute this Run Unit” option.

As each Run Unit executes, the name will be proceeded by “[Executing].” Note that although the Text Literal Parameter is statically set to “TWM_Customer_Deep”, as soon as the Run Unit that calls FLRegrDataPrep is executed, its Text Literal Parameter is set to the return value of the XSP OUT argument. This is automatically reflected in the SQL Text Run Unit that queries the FLRegrDataPrep system and result tables. These analysis identifiers are randomly generated numbers, preceded by the letter “A” - “<Axxxxxx>.”

Once the execution is “Complete” in the “Execution Status” window, click on the “Results→data” tab and you should see 3 result sets – the first being the OUT argument of the XSP and the second two the results of querying the system table and the deep table as follows:

<u>Check Prep Mapping Table Run Unit</u>

ANALYSISID	…	**COLUMN_NAME**	…	**Final_VarID**
A209629	…	ccacct	…	-1
A209629	…	INTERCEPT	…	0

A209629	…	age	…	1
A209629	…	avg_ck_bal	…	2
A209629	…	avg_ck_tran_amt	…	3
A209629	…	avg_ck_tran_cnt	…	4
A209629	…	avg_sv_bal	…	5
A209629	…	avg_sv_tran_amt	…	6
A209629	…	avg_sv_tran_cnt	…	7
A209629	…	ckacct	…	8
A209629	…	female	…	9
A209629	…	income	…	10
A209629	…	married	…	11
A209629	…	nbr_children	…	12
A209629	…	separated	…	13
A209629	…	single	…	14
A209629	…	svacct	…	15
A209629	…	years_with_bank	…	16

Sample Deep Table Run Unit

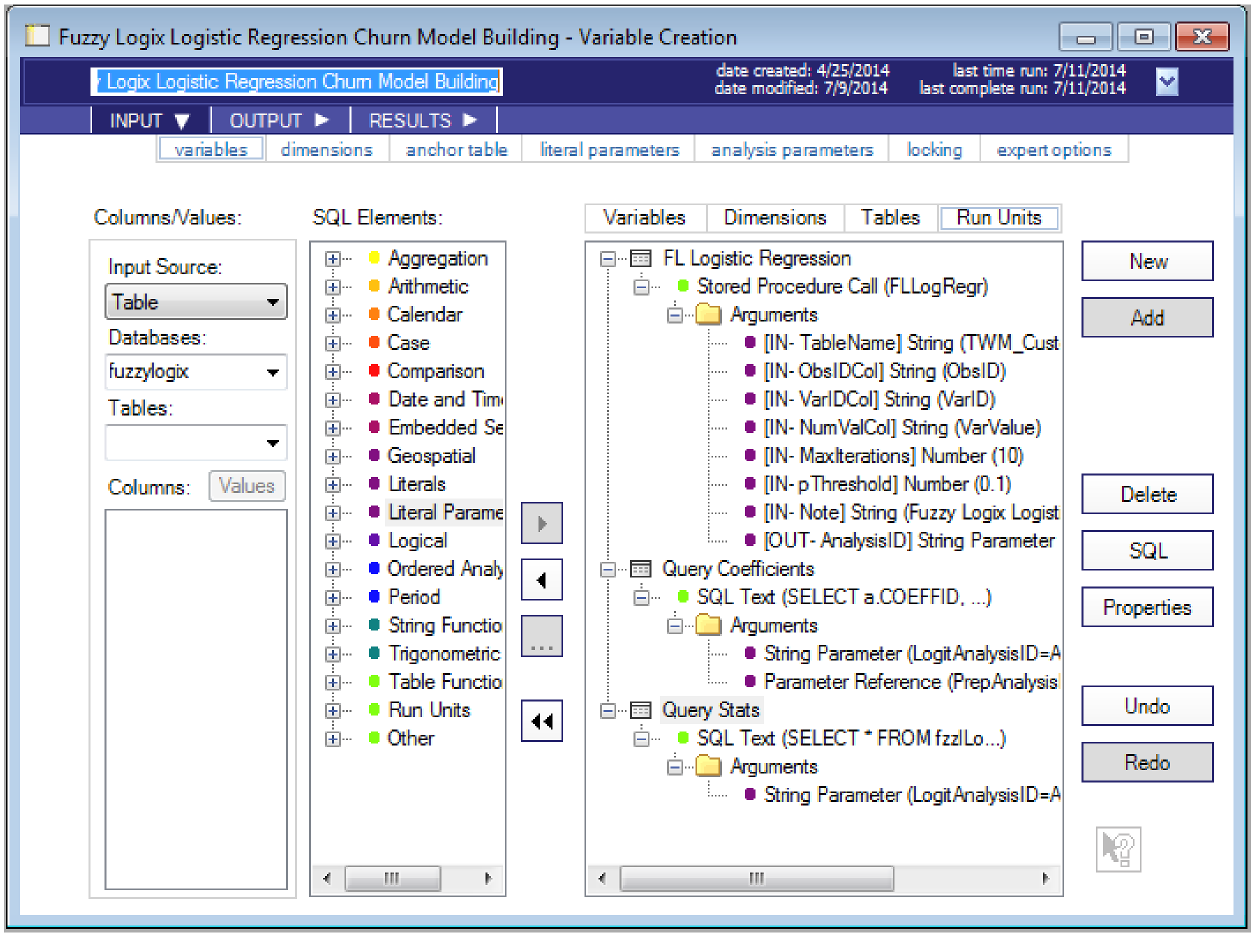
VarID	Count(*)
-1	747
0	747
1	747
2	747
3	747
4	747
5	747
6	747
7	747
8	747
9	747
10	747
11	747
12	747
13	747
14	747
15	747
16	747

This is shown here to illustrate that you will often have to join the information from the fzzlRegrDataPrep system table with tables subsequently generated by other Data Mining XSP's because DB Lytix™ will refer to "VarID's" instead of actual column names as shown in the query of the Deep Table "TWM_Customer_Deep."

See the Data Mining → FLRegrDataPrep section in the User Manual for DB Lytix™ on Teradata Advanced Package v1.0.1 for a complete description of the system table and results generated.

5.5. DB Lytix™ Data Mining / Model Building XSP's

Next we will see an example of an XSP that builds a model from the "Deep" table that was created by the previous Variable Creation analysis. As the first example, we will build a customer churn model based upon this data utilizing the FLLogRegr DB Lytix™ XSP. Then we will query the two system tables generated by FLLogRegr and view the coefficients and statistics generated. Here is what one of finished example analyses looks like ("Fuzzy Logix Logistic Regression Churn Model Building" Variable Creation analysis in the tutorial):



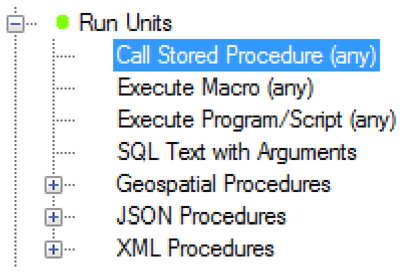
FL Logistic Regression
- Stored Procedure Call (FLLogRegr)
  - Arguments
    - [IN- TableName] String (TWM_Cust
    - [IN- ObsIDCol] String (ObsID)
    - [IN- VarIDCol] String (VarID)
    - [IN- NumValCol] String (VarValue)
    - [IN- MaxIterations] Number (10)
    - [IN- pThreshold] Number (0.1)
    - [IN- Note] String (Fuzzy Logix Logist
    - [OUT- AnalysisID] String Parameter

Query Coefficients
- SQL Text (SELECT a.COEFFID, ...)
  - Arguments
    - String Parameter (LogitAnalysisID=A
    - Parameter Reference (PrepAnalysis

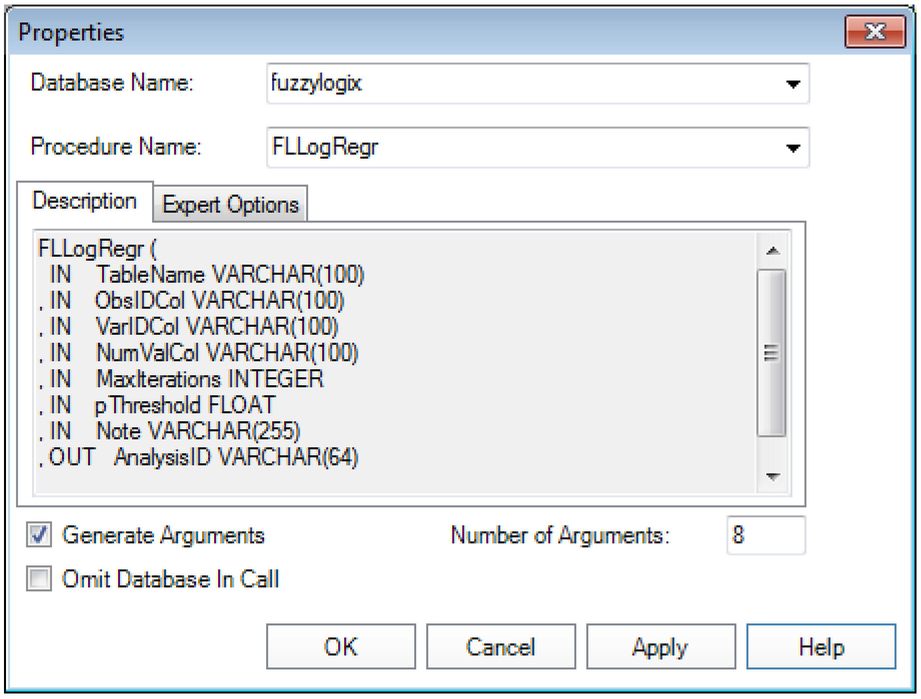
Query Stats
- SQL Text (SELECT * FROM fzzlLo...)
  - Arguments
    - String Parameter (LogitAnalysisID=A

Here are the steps to parameterize the analysis:

1) Click on the "+" icon next to "Run Units" in the "SQL Elements" list. It will expand and show the following elements:



Select "Call Stored Procedure (any)" by dragging and dropping it on to the Run Units palette or highlighting it, and double-clicking or using the ">" button. This action will bring up the following dialogue:



Database Name: fuzzylogix

Procedure Name: FLLogRegr

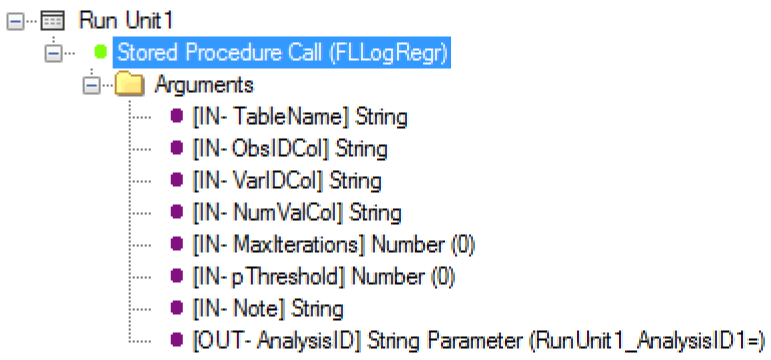
```
FLLogRegr (
  IN   TableName VARCHAR(100)
, IN   ObsIDCol VARCHAR(100)
, IN   VarIDCol VARCHAR(100)
, IN   NumValCol VARCHAR(100)
, IN   MaxIterations INTEGER
, IN   pThreshold FLOAT
, IN   Note VARCHAR(255)
, OUT  AnalysisID VARCHAR(64)
```

[✓] Generate Arguments   Number of Arguments: 8

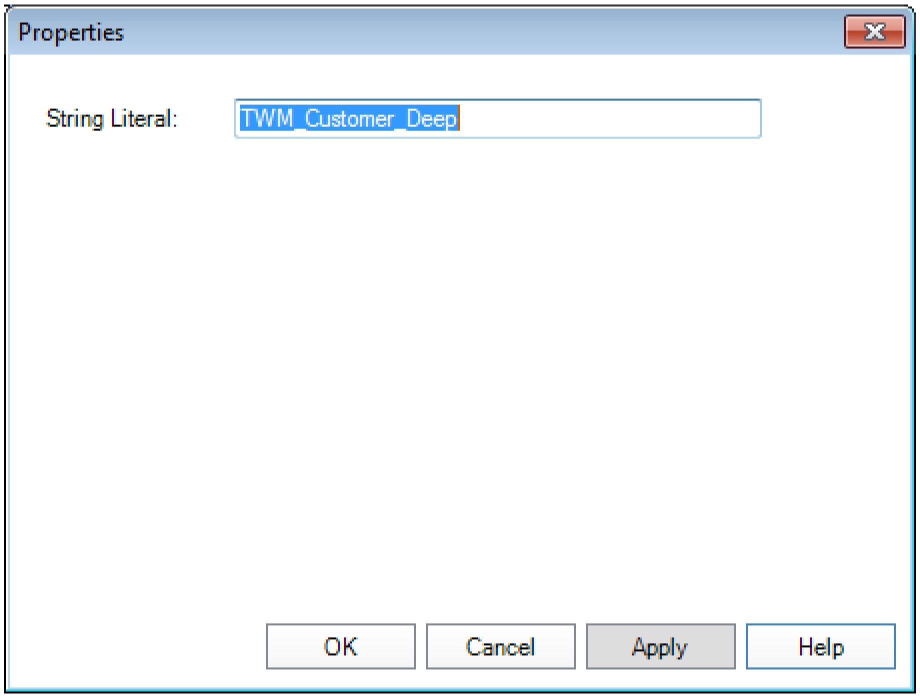
[ ] Omit Database In Call

Select the database where DB Lytix™ is installed in the "Database Name:" pull-down. Once you do this, the "Procedure Name" pull-down will be populated with the Stored Procedures that reside in the selected database. It will also show the signature of the function under the "Description" tab. Select FLLogRegr in the "Procedure Name:" pull-down to view the signature. By default, ADS Generator will fully qualify the UDF call by prepending it with the database name it resides in. You can turn this default behavior off by enabling the "Omit Database In Call" check-box.

2) Click the "Generate Arguments" checkbox and hit OK or Apply. This option will create a template of the arguments required to call the FLLogRegr XSP as follows:


Run Unit1
Stored Procedure Call (FLLogRegr)
Arguments
[IN- TableName] String
[IN- ObsIDCol] String
[IN- VarIDCol] String
[IN- NumValCol] String
[IN- MaxIterations] Number (0)
[IN- pThreshold] Number (0)
[IN- Note] String
[OUT- AnalysisID] String Parameter (RunUnit1_AnalysisID1=)

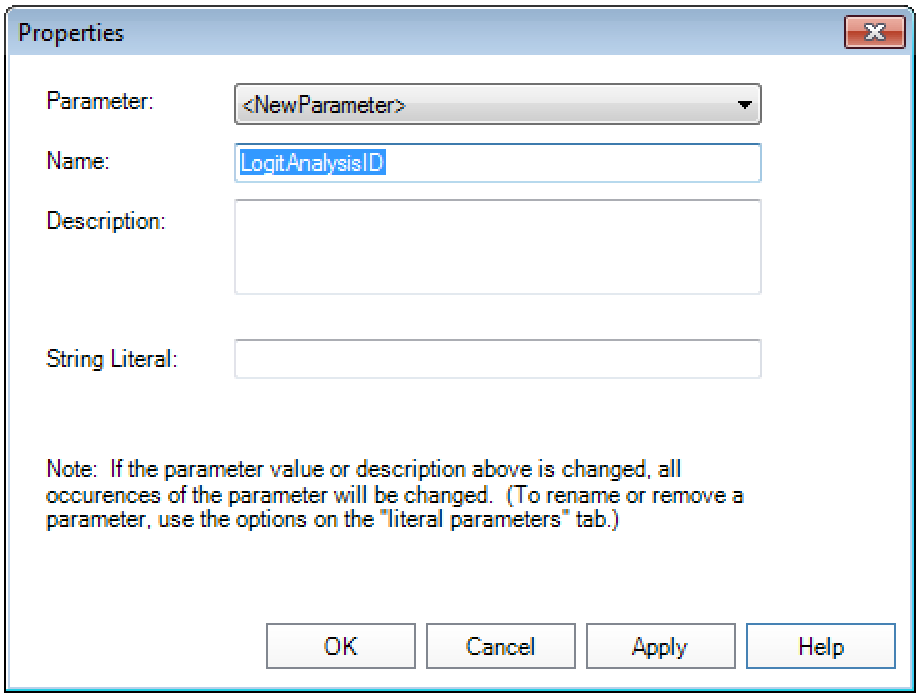
Next, start entering values for each of the arguments. Do this by highlighting the first argument ("TableName") and hitting the "Properties" button, or double clicking on it:


Properties

String Literal: TWM_Customer_Deep

Enter "TWM_Customer_Deep" and click on Apply, or simply click on the next argument to enter the next String Literal. Repeat for the remaining literal values:

- "ObsIdCol" = ObsId
- "VarIdCol" = VarId
- "NumValCol" = VarValue
- "MaxIterations" = 10
- "pThreshold" = 0.1
- "Note" = Fuzzy Logix Logistic Regression Churn Model
- Note that the last XSP argument, the OUT parameter, has been specified as a "String Parameter." This is because the XSP returns an analysis identifier as its OUT parameter. Once again, this needs to be a Literal Parameter so we can use it in subsequent Run Units to query the data generated in the system tables, and to score the resulting model. Create a new String Literal Parameter called "LogitAnalysisID" as follows:

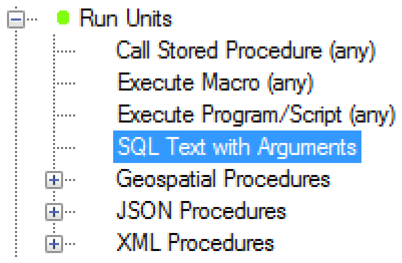


Name: LogitAnalysisID

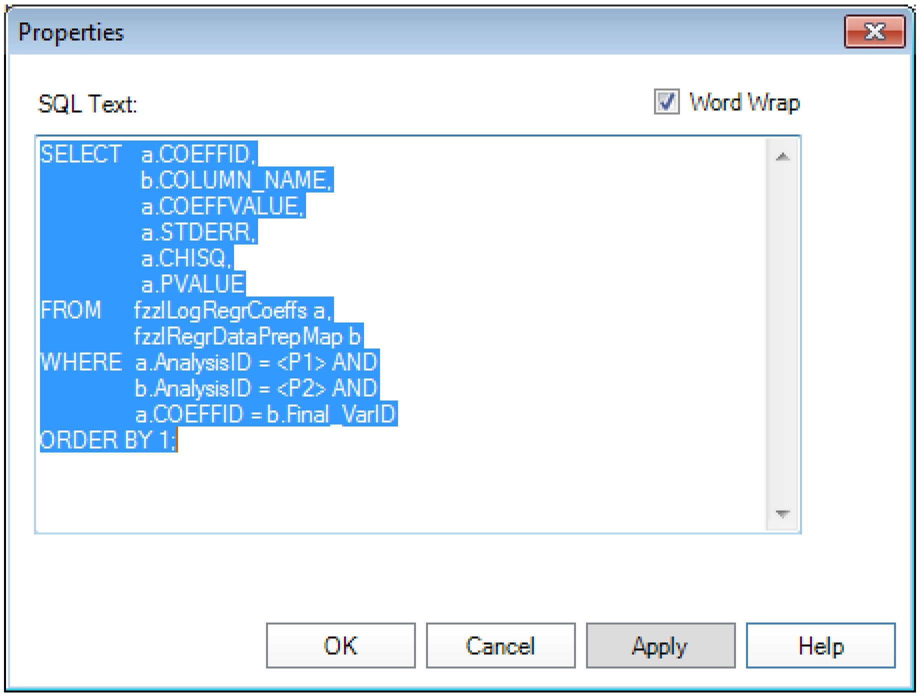
Note: If the parameter value or description above is changed, all occurences of the parameter will be changed. (To rename or remove a parameter, use the options on the "literal parameters" tab.)

Click on OK or Apply to create the Literal Parameter.

3) Next, create another Run Unit, this time of type "SQL Text with Arguments" in order to query the system tables populated by the call to FLLogRegr, using the "LogitAnalysisID" literal parameter set by the previous Run Unit as well as the "PrepAnalysisID" literal parameter set in the previous Variable Creation analysis:
   a. Click on the "+" icon next to "Run Units" in the "SQL Elements" list. It will expand and show the following elements:



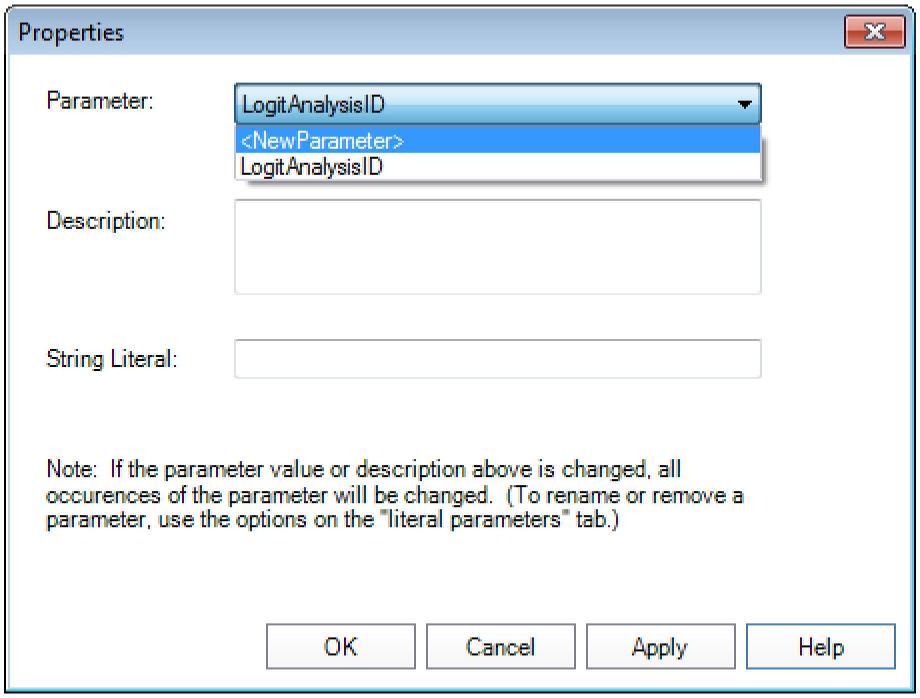
   Select "SQL Text with Arguments" by dragging and dropping it on to the Run Units palette or highlighting it, and double-clicking or using the ">" button. This action will bring up the following dialogue:



```
SELECT   a.COEFFID,
         b.COLUMN_NAME,
         a.COEFFVALUE,
         a.STDERR,
         a.CHISQ,
         a.PVALUE
FROM     fzzlLogRegrCoeffs a,
         fzzlRegrDataPrepMap b
WHERE  a.AnalysisID = <P1> AND
         b.AnalysisID = <P2> AND
         a.COEFFID = b.Final_VarID
ORDER BY 1;
```

Type in the above query and hit OK or Apply. Next, add two literal parameters to this Run Unit. Notice that P1 (the first literal parameter in the folder) corresponds to the FLLogRegr system table, while P2 corresponds to the FLRegrDataPrep system table from the previous Variable Creation analysis:

b. Add a String Literal Parameter to the "SQL Text with Arguments" folder:



Parameter: LogitAnalysisID
<NewParameter>
LogitAnalysisID

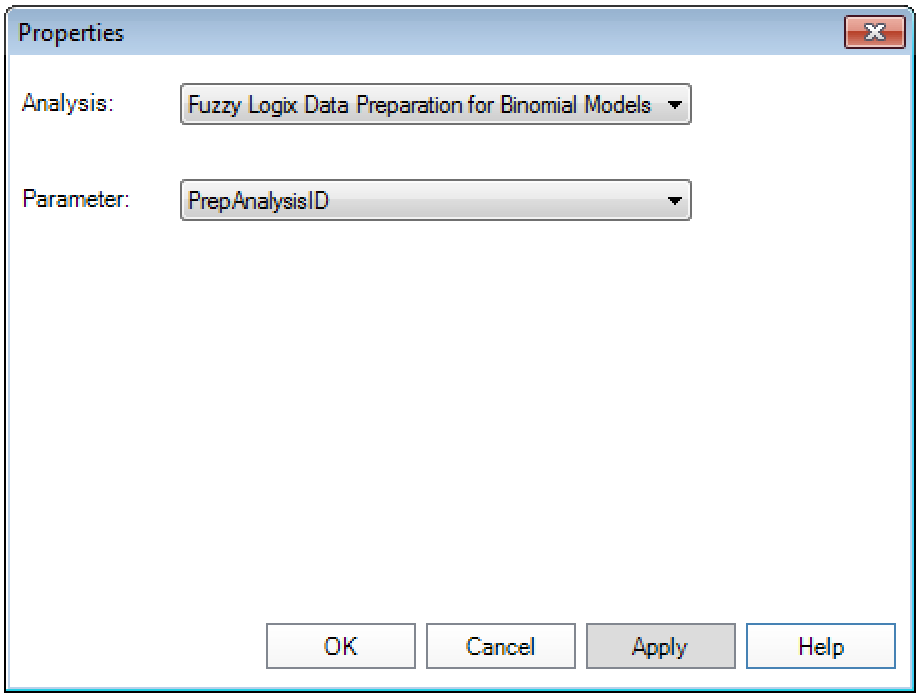
Description:

String Literal:

Note: If the parameter value or description above is changed, all occurences of the parameter will be changed. (To rename or remove a parameter, use the options on the "literal parameters" tab.)

Select “LogitAnalysisID” from the “Parameter:” pull-down list and click on OK or Apply.

c. Since the next parameter was created by a different Variable Creation analysis, a “Parameter Reference” needs to be used. Add a Parameter Reference Literal Parameter to the “SQL Text with Arguments” folder:

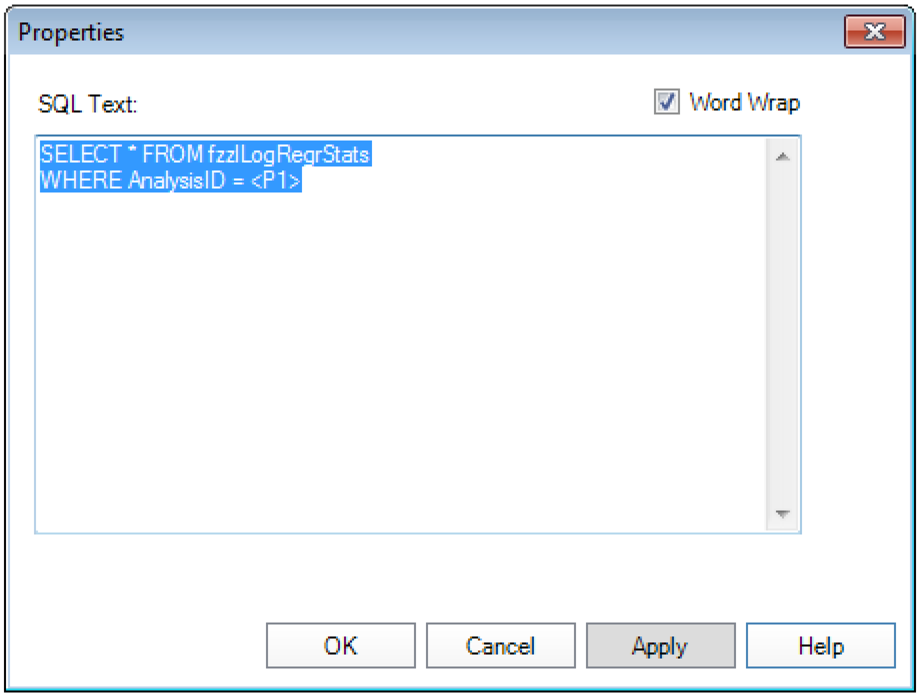


Analysis: Fuzzy Logix Data Preparation for Binomial Models

Parameter: PrepAnalysisID

Specify “Fuzzy Logix Data Preparation for Binomial Models” in the “Analysis:” pull-down list, and “PrepAnalysisID” in the “Parameter:” pull-down list.

4) Next create another “SQL Text with Arguments” Run Unit following the same instructions as above. This Run Unit will query the other system tables populated by the call to FLLogRegr, using the “LogitAnalysisID” literal parameter:
   a. This action will bring up the following dialogue:


Properties

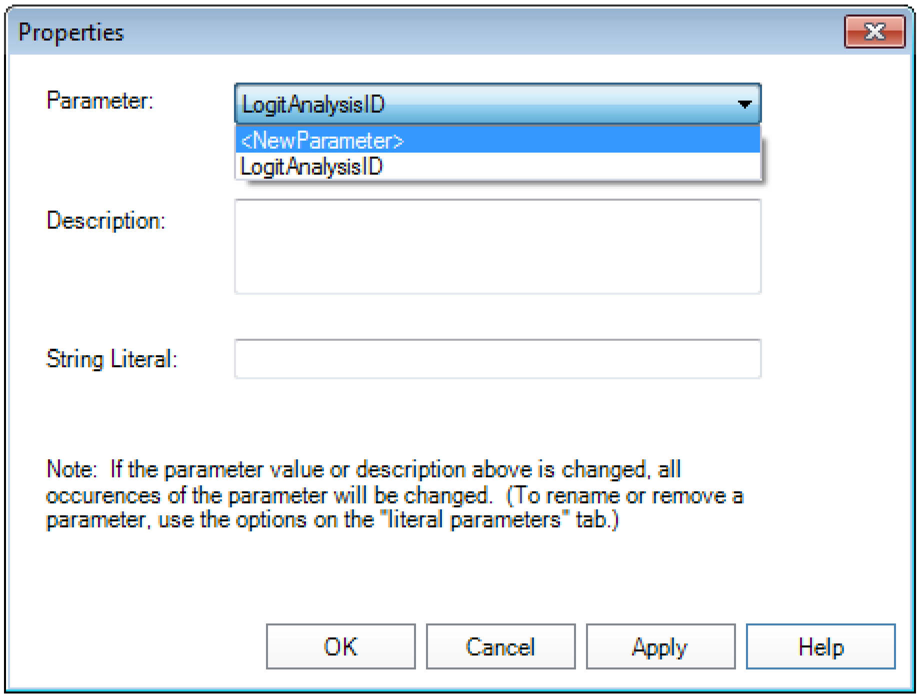
SQL Text: ☑ Word Wrap

```
SELECT * FROM fzzlLogRegrStats
WHERE AnalysisID = <P1>
```

OK Cancel Apply Help

Type in the above query and hit OK or Apply.

b. Next, add a literal parameter to this Run Unit that corresponds to the FLLogRegr OUT literal parameter.


Properties

Parameter: LogitAnalysisID

<NewParameter>
LogitAnalysisID

Description:

String Literal:

Note: If the parameter value or description above is changed, all occurences of the parameter will be changed. (To rename or remove a parameter, use the options on the "literal parameters" tab.)

OK Cancel Apply Help

Select “LogitAnalysisID” from the “Parameter:” pull-down list and click on OK or Apply.

5) Now execute the Variable Creation Analysis by clicking on the “Run” button directly under the “Tools” menu item, or highlighting the Variable Creation analysis in the “Project Explorer,” right-clicking and selecting “Run.” Alternately, since there are only Run Units within this particular Variable Creation Analysis, the individual Run Units can be executed by right clicking on them and selecting the “Execute this Run Unit” option.

   As each Run Unit executes, the name will be proceeded by “[Executing].” Note that the Parameter Reference should already to set to the same value as the Literal Parameter from the FLRegrDataPrep Run Unit, and that the String Literal Parameter set in the XSP OUT argument is automatically reflected in the String Literal Parameter in the SQL Text with Arguments Run Units.

   Once the execution is “Complete” in the “Execution Status” window, click on the “Results→data” tab and you should see 3 result sets – the first being the OUT argument of the XSP (DB Lytix™ Analysis Identifier) and the second two the results of querying the system tables populated by the XSP call:

   <u>Query Coefficients Run Unit</u>

COEFFID	**COLUMN_NAME**	**COEFFVALUE**	**STDERR**	**CHISQ**	**PVALUE**
0	INTERCEPT	-0.47868	0.686937	0.48557	0.48591
1	age	-0.00763	0.008153	0.874717	0.349653
2	avg_ck_bal	-0.00021	0.000134	2.373669	0.123397
3	avg_ck_tran_amt	0.00299	0.002232	1.794506	0.180378
4	avg_ck_tran_cnt	-0.03162	0.010777	8.608375	0.003346
5	avg_sv_bal	0.002983	0.000537	30.82613	2.82E-08
6	avg_sv_tran_amt	0.030995	0.003809	66.20402	4.44E-16
7	avg_sv_tran_cnt	-1.19806	0.212373	31.82416	1.69E-08
8	ckacct	0.404663	0.250355	2.612611	0.106017
9	female	0.059657	0.241319	0.061113	0.804745
10	income	-1.54E-05	9.70E-06	2.523791	0.112141
11	married	-0.66981	0.447473	2.240633	0.134426
12	nbr_children	-0.22261	0.157462	1.998628	0.157442
13	separated	0.459813	0.579787	0.628963	0.427736
14	single	-0.59572	0.544805	1.195637	0.274196
15	svacct	0.24019	0.356367	0.45427	0.500314
16	years_with_bank	-0.09249	0.04577	4.083411	0.043306

   This shows how we were able to join the information from the fzzlRegrDataPrep system table generated by another Variable Creation Analysis with the fzzlLogRegr coefficients system table to view actual column names as opposed to internal Variable Identifiers.

   <u>Query Stats Run Unit</u>

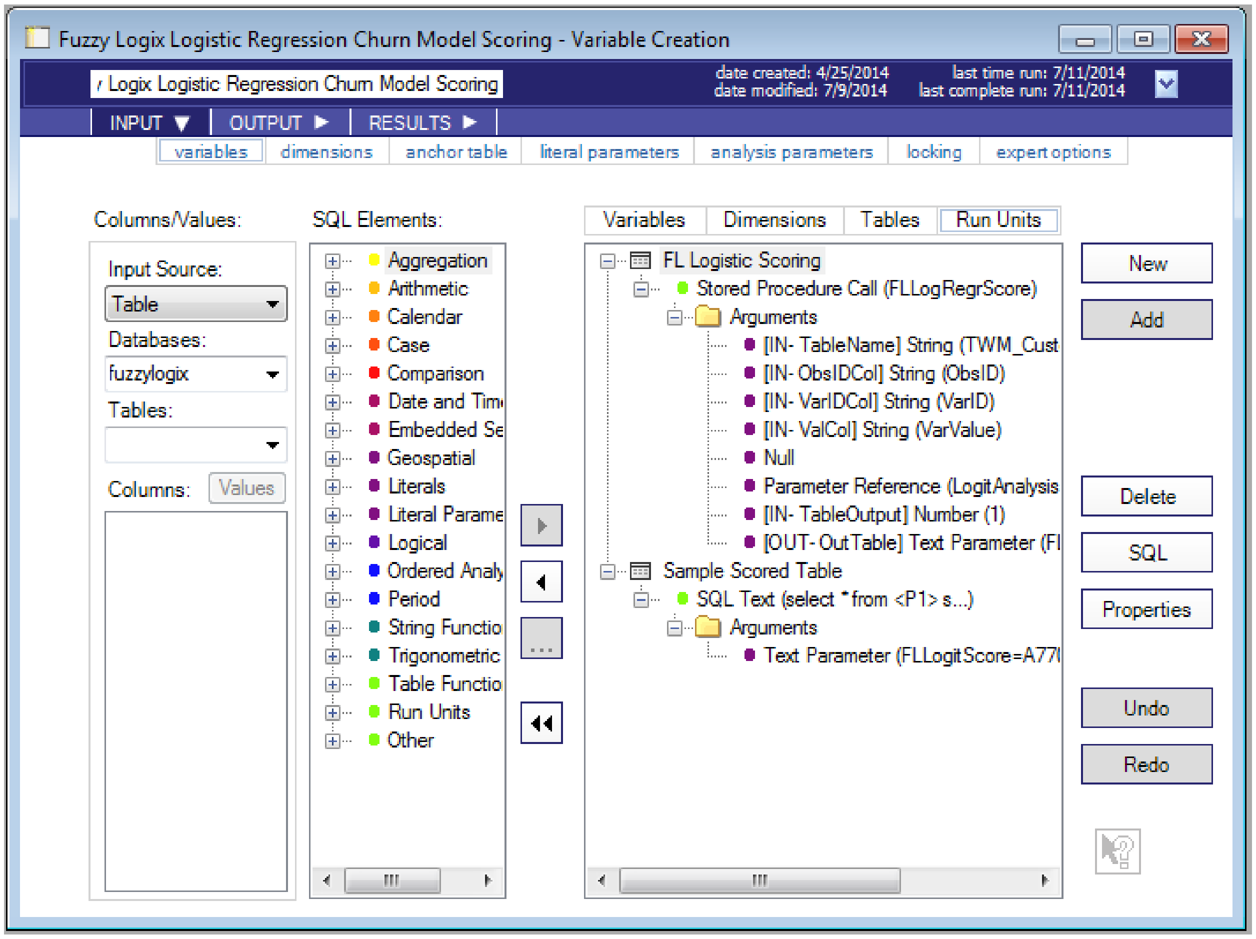
ANALYSISID	A200216
MODELID	1
NUMOFVARS	16

ITERATIONS	8
CONCORDANT	133116
DISCORDANT	6057
TIED	327
TOTALPAIRS	139500
GINICOEFF	0.910817
CSTATISTIC	0.955409
GAMMA	0.912957
HIGHESTPVALUE	0.804745
EVENTS	375
NONEVENTS	372
NUMOFOBS	747
FALSEPOSITIVE	231
FALSENEGATIVE	1

See the Data Mining → FLLogRegr section in the User Manual for DB Lytix™ on Teradata Advanced Package v1.0.1 for a complete description of the system table and results generated.

5.6. DB Lytix™ Data Mining / Model Scoring XSP's

Next we will see an example of an XSP that scores the model created by the previous Variable Creation analysis – additionally, a sample of the scored dataset is taken. Here is what one of finished example analyses looks like ("Fuzzy Logix Logistic Regression Churn Model Scoring" Variable Creation analysis in the tutorial):



r Logix Logistic Regression Churn Model Scoring

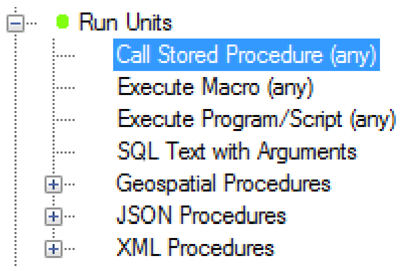
date created: 4/25/2014
date modified: 7/9/2014
last time run: 7/11/2014
last complete run: 7/11/2014

Input Source: Table
Databases: fuzzylogix
Tables:

```
FL Logistic Scoring
    Stored Procedure Call (FLLogRegrScore)
        Arguments
            [IN- TableName] String (TWM_Cust
            [IN- ObsIDCol] String (ObsID)
            [IN- VarIDCol] String (VarID)
            [IN- ValCol] String (VarValue)
            Null
            Parameter Reference (LogitAnalysis
            [IN- TableOutput] Number (1)
            [OUT- OutTable] Text Parameter (Fl
Sample Scored Table
    SQL Text (select * from <P1> s...)
        Arguments
            Text Parameter (FLLogitScore=A77(
```

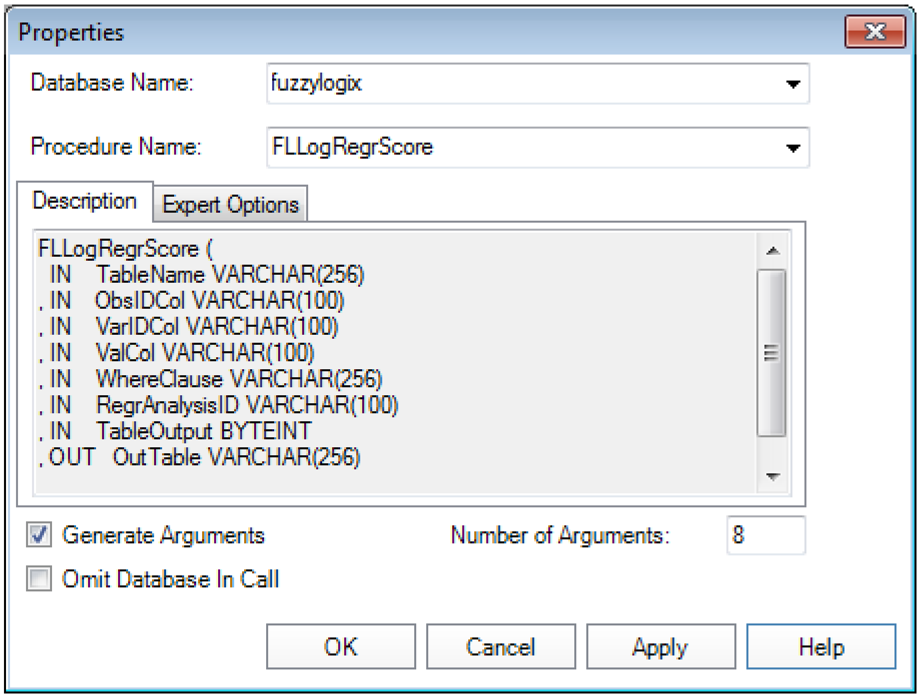
Here are the steps to parameterize the analysis:

1) Click on the “+” icon next to “Run Units” in the “SQL Elements” list. It will expand and show the following elements:



Run Units
- Call Stored Procedure (any)
- Execute Macro (any)
- Execute Program/Script (any)
- SQL Text with Arguments
- Geospatial Procedures
- JSON Procedures
- XML Procedures

Select “Call Stored Procedure (any)” by dragging and dropping it on to the Run Units palette or highlighting it, and double-clicking or using the “>” button. This action will bring up the following dialogue:



Properties

Database Name: fuzzylogix

Procedure Name: FLLogRegrScore

Description | Expert Options

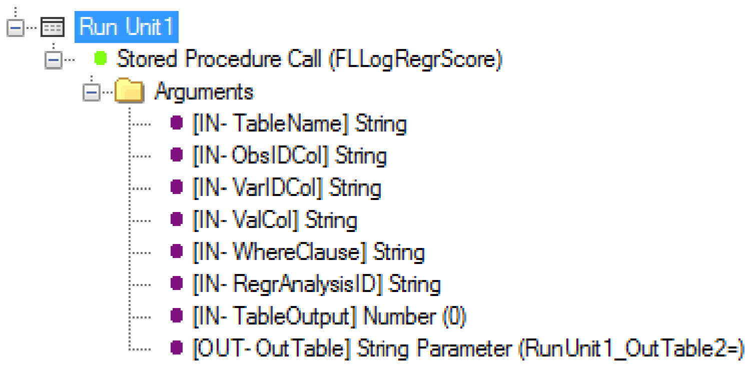
```
FLLogRegrScore (
  IN   TableName VARCHAR(256)
, IN   ObsIDCol VARCHAR(100)
, IN   VarIDCol VARCHAR(100)
, IN   ValCol VARCHAR(100)
, IN   WhereClause VARCHAR(256)
, IN   RegrAnalysisID VARCHAR(100)
, IN   TableOutput BYTEINT
, OUT  OutTable VARCHAR(256)
```

☑ Generate Arguments Number of Arguments: 8

☐ Omit Database In Call

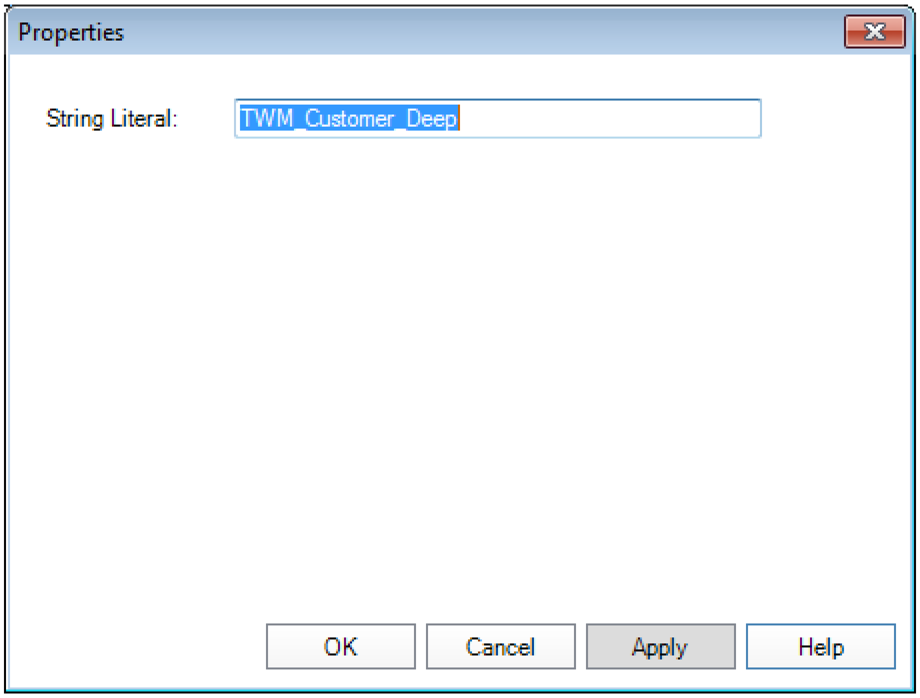
Select the database where DB Lytix™ is installed in the “Database Name:” pull-down. Once you do this, the “Procedure Name” pull-down will be populated with the Stored Procedures that reside in the selected database. It will also show the signature of the function under the “Description” tab. Select FLLogRegrScore in the “Procedure Name:” pull-down to view the signature. By default, ADS Generator will fully qualify the UDF call by prepending it with the database name it resides in. You can turn this default behavior off by enabling the “Omit Database In Call” check-box.

2) Click the “Generate Arguments” checkbox and hit OK or Apply. This option will create a template of the arguments required to call the FLLogRegrScore XSP as follows:



Run Unit1
Stored Procedure Call (FLLogRegrScore)
Arguments
[IN- TableName] String
[IN- ObsIDCol] String
[IN- VarIDCol] String
[IN- ValCol] String
[IN- WhereClause] String
[IN- RegrAnalysisID] String
[IN- TableOutput] Number (0)
[OUT- OutTable] String Parameter (RunUnit1_OutTable2=)

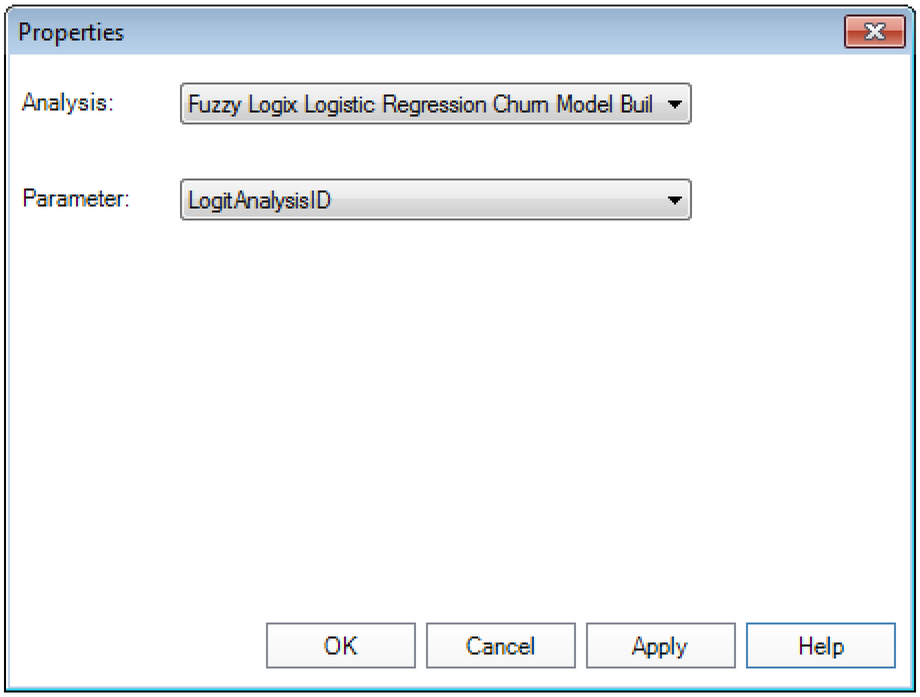
Next, start entering values for each of the arguments. Do this by highlighting the first argument ("TableName") and hitting the "Properties" button, or double clicking on it:



String Literal: TWM_Customer_Deep

Enter "TWM_Customer_Deep" and click on Apply, or simply click on the next argument to enter the next String Literal. Repeat for the remaining literal values:

- "ObsIdCol" = ObsId
- "VarIdCol" = VarId
- "NumValCol" = VarValue
- "WhereClause" = ‘’ (or drag a Literal Null value on top as in the tutorial)
- "TableOutput" = 1
- "RegrAnalysisID" is the OUT argument from the call to FLLogRegr, which was captured in a String Literal Parameter named "LogitAnalysisID" in the previous Variable Creation analysis. Change the default generated String Literal with a Parameter Reference Literal Parameter as follows:

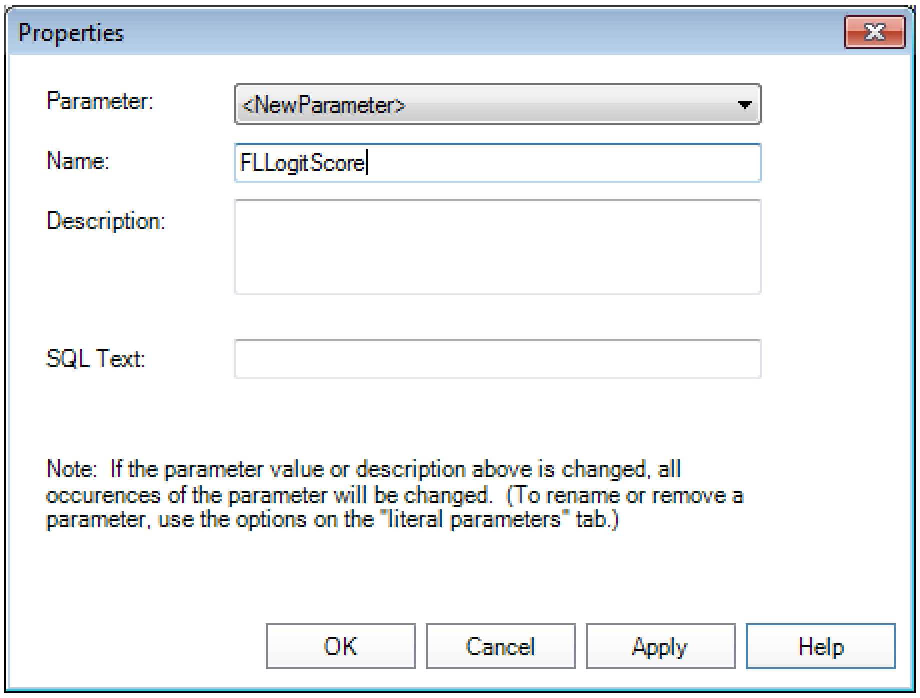


Properties

Analysis: Fuzzy Logix Logistic Regression Churn Model Buil

Parameter: LogitAnalysisID

- Once again, note that the last XSP argument, the OUT parameter, has been specified as a "String Parameter." Since we have parameterized this XSP to return a dynamically generated volatile table name as its OUT parameter (since "TableOutput" is set to 1), this needs to be changed to a "Text Parameter." To do this simply drag and drop a "Text" Literal Parameter on top of the existing String Parameter ("[Out – OutTable]") and create a new Text Literal Parameter using the instructions given above. In the example given, the name of the Text Literal Parameter created was also "FLLogitScore."



Properties

Parameter: <NewParameter>

Name: FLLogitScore

Description:

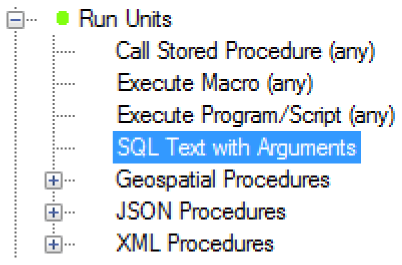
SQL Text:

Note: If the parameter value or description above is changed, all occurences of the parameter will be changed. (To rename or remove a parameter, use the options on the "literal parameters" tab.)

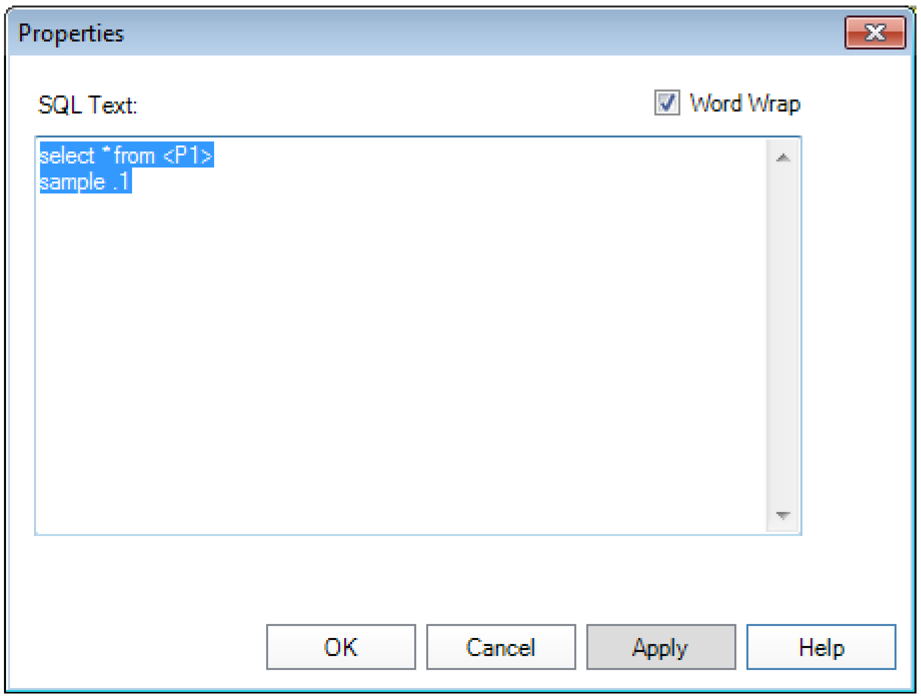
Click on OK or Apply to create the Literal Parameter.

3) Next, create another Run Unit, this time of type “SQL Text with Arguments” in order to query the scored table populated by the call to FLLogRegrScore, using the “FLLogitScore” literal parameter set by the previous Run Unit:

   a. Click on the “+” icon next to “Run Units” in the “SQL Elements” list. It will expand and show the following elements:



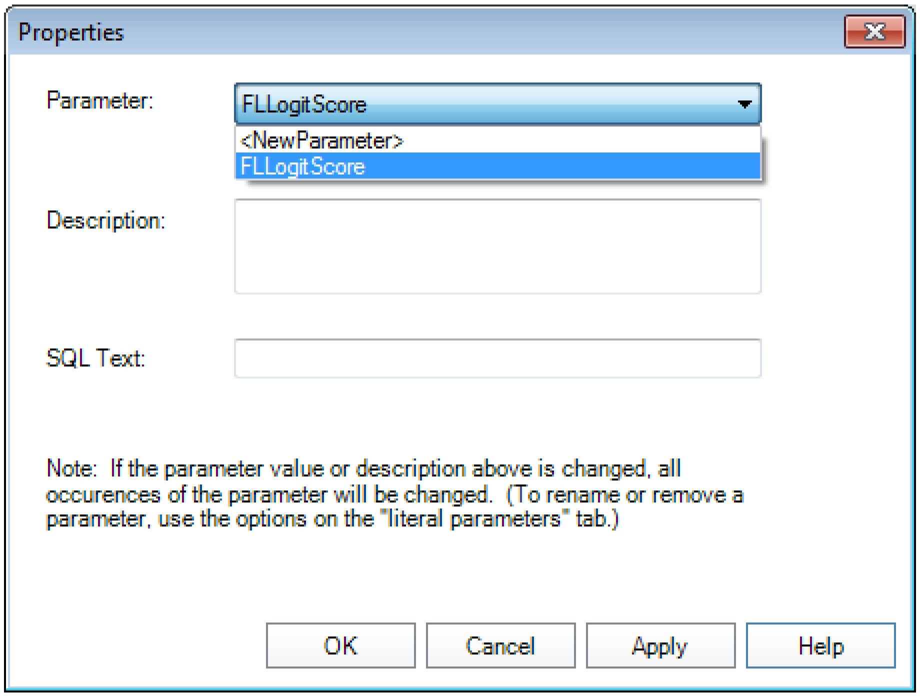
   Select “SQL Text with Arguments” by dragging and dropping it on to the Run Units palette or highlighting it, and double-clicking or using the “>” button. This action will bring up the following dialogue:



```
select * from <P1>
sample .1
```

   Type in the above query and hit OK or Apply. Next, add a literal parameter to this Run Unit.

   b. Add a Text Literal Parameter to the “SQL Text with Arguments” folder:


Properties

Parameter: FLLogitScore

<NewParameter>
FLLogitScore

Description:

SQL Text:

Note: If the parameter value or description above is changed, all occurences of the parameter will be changed. (To rename or remove a parameter, use the options on the "literal parameters" tab.)

Select “FLLogitScore” from the “Parameter:” pull-down list and click on OK or Apply.

4) Now execute the Variable Creation Analysis by clicking on the “Run” button directly under the “Tools” menu item, or highlighting the Variable Creation analysis in the “Project Explorer,” right-clicking and selecting “Run.” Alternately, since there are only Run Units within this particular Variable Creation Analysis, the individual Run Units can be executed by right clicking on them and selecting the “Execute this Run Unit” option.

As each Run Unit executes, the name will be proceeded by “[Executing].” Note that the Parameter Reference should already to set to the same value as the Literal Parameter from the FLLogRegr Run Unit, and that the Text Literal Parameter set in the FLLogRegrScore XSP OUT argument is automatically reflected in the Text Literal Parameter in the SQL Text with Arguments Run Units.

Once the execution is “Complete” in the “Execution Status” window, click on the “Results→data” tab and you should see 2 result sets – the first being the OUT argument of the XSP (DB Lytix™ Scored volatile table name) and the second the result of sampling the scored table:

Sample Scored Table Run Unit

ObsID	Y
1362654	0.991008
1362610	0.088947
1363214	0.018272
1363486	0.104522
…	…

…	…
…	…
1363026	0.042755
1363353	0.09518
1363042	0.149811

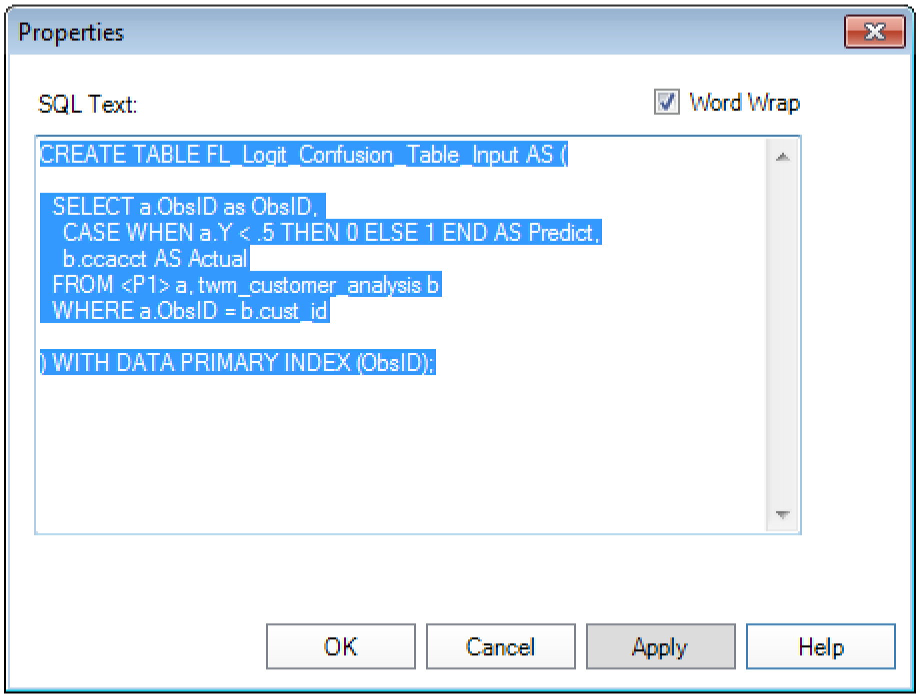
See the Data Mining → FLLogRegrScore section in the User Manual for DB Lytix™ on Teradata Advanced Package v1.0.1 for a complete description of the system table and results generated.

6. <u>Bringing it all Together – Variables, Tables and Run-Units</u>

The next Variable Creation Analysis builds upon a combination of the “Fuzzy Logix Logistic Regression Churn Model Building” and “Fuzzy Logix Logistic Regression Churn Model Scoring” Variable Creation Analysis by illustrating the creation of Variables and the use of Function Tables along with the execution of Run Units.

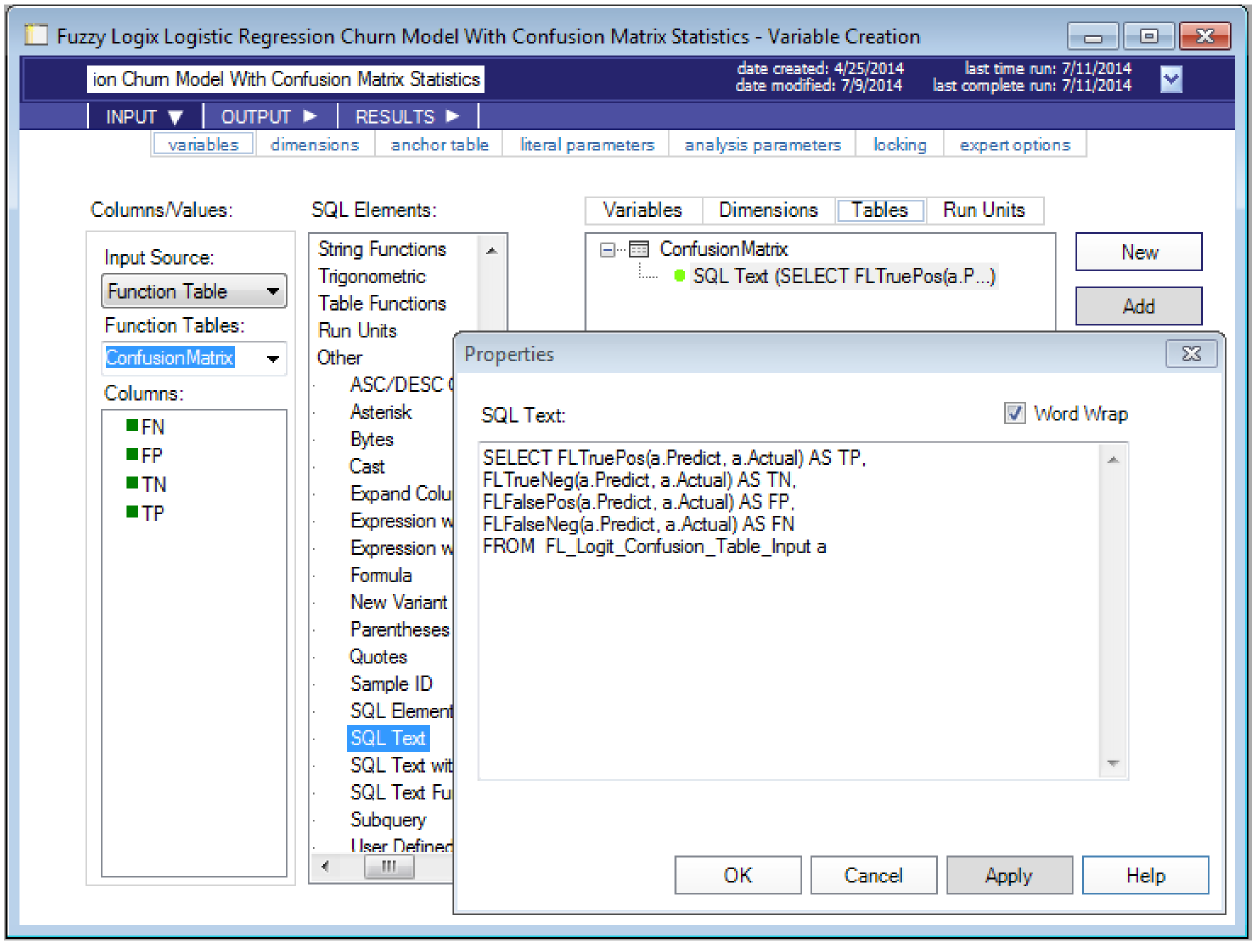
6.1. <u>Fuzzy Logix Logistic Regression Churn Model With Confusion Matrix Statistics</u>

For brevity, we will not describe the model building, scoring and supporting Run Units. After the regression model is scored, there are two additional “SQL Text with Arguments” Run Units that drop and create a table based upon the scored data as follows:



```
CREATE TABLE FL_Logit_Confusion_Table_Input AS (

 SELECT a.ObsID as ObsID,
  CASE WHEN a.Y < .5 THEN 0 ELSE 1 END AS Predict,
  b.ccacct AS Actual
 FROM <P1> a, twm_customer_analysis b
 WHERE a.ObsID = b.cust_id

) WITH DATA PRIMARY INDEX (ObsID);
```

The table created by this Run Unit (“FL_Logit_Confusion_Table_Input”) joins the original data (prior to FLRegrDataPrep) with the “Actual” model training values in it to the predicted values in the scored table using 0.5 as the threshold for a successful prediction. This table is then used to create a SQL Text Function Table:



Input Source: Function Table

Function Tables: ConfusionMatrix

Columns:
- FN
- FP
- TN
- TP

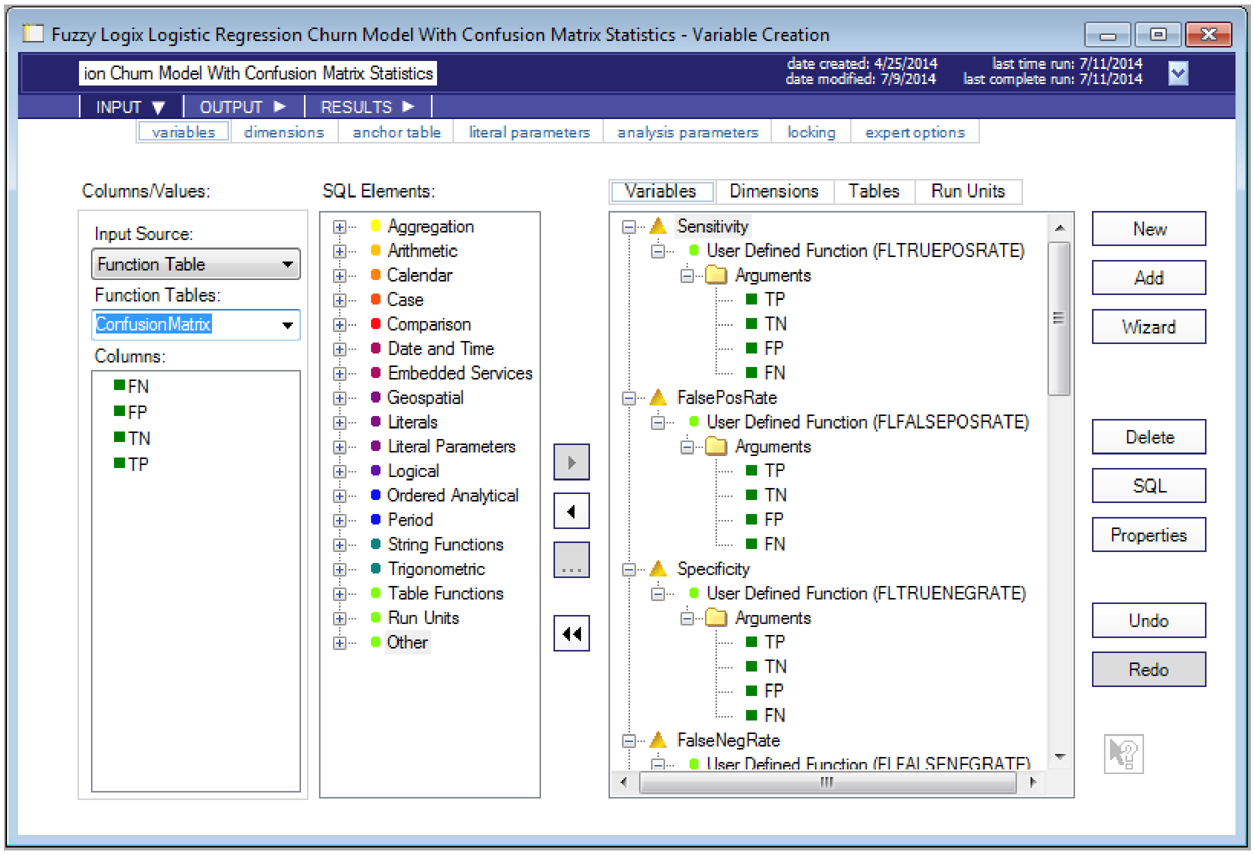
SQL Text: [✓] Word Wrap

```
SELECT FLTruePos(a.Predict, a.Actual) AS TP,
FLTrueNeg(a.Predict, a.Actual) AS TN,
FLFalsePos(a.Predict, a.Actual) AS FP,
FLFalseNeg(a.Predict, a.Actual) AS FN
FROM  FL_Logit_Confusion_Table_Input a
```

You do this by clicking on the “Tables” palette, dragging and dropping “SQL Text” object (SQL Elements→Other→SQL Text) unto the palette, and entering the query as above. Note that we are using a trick here by calling the DB Lytix™ Confusion Matrix preparation functions:

- FLTruePos (TP above): Actual value = Predicted value = 1
- FLTrueNeg (TN above): Actual value = Predicted value = 0
- FLFalsePos (FP above): Actual value = 0 and Predicted value = 1
- FLFalseNeg (FN above): Actual value = 1 and Predicted value = 0

This “Table” will then be used by changing the Variable Creation analysis “Input Source:” to “Function Table,” the “Function Tables:” drop down list to “ConfusionMatrix,” and parameterizing the DB Lytix™ Confusion Matrix UDF’s as follows:



Input Source: Function Table

Function Tables: ConfusionMatrix

Columns:

- FN
- FP
- TN
- TP

```
Sensitivity
    User Defined Function (FLTRUEPOSRATE)
        Arguments
            TP
            TN
            FP
            FN
FalsePosRate
    User Defined Function (FLFALSEPOSRATE)
        Arguments
            TP
            TN
            FP
            FN
Specificity
    User Defined Function (FLTRUENEGRATE)
        Arguments
            TP
            TN
            FP
            FN
FalseNegRate
    User Defined Function (FLFALSENEGRATE)
```

Each of the four columns returned by the “ConfusionMatrix” Function Table are dragged and dropped as parameters to the following Confusion Matrix Performance Measures:

<u>FLTruePosRate</u>	Scalar function that calculates the sensitivity or the true positive rate from the confusion matrix. It measures the proportion of actual positives which are correctly identified.
<u>FLFalsePosRate</u>	Scalar function that calculates false positive rate from the confusion matrix. It measures the proportion of actual positives which are incorrectly identified.
<u>FLTrueNegRate</u>	Scalar function that calculates the specificity or the true negative rate from the confusion matrix. It measures the proportion of actual negatives which are correctly identified.
<u>FLFalseNegRate</u>	Scalar function that calculates false negative rate from the confusion matrix. It measures the proportion of actual negatives which are incorrectly identified.
<u>FLAccuracy</u>	Scalar function that calculates the accuracy of predictions from the confusion matrix. It is the proportion of the total number of predictions that were correct.
<u>FLPosPredVal</u>	Scalar function that calculates the positive predictive value or precision rate. It measures the proportion of actual positives which are correctly identified.
<u>FLNegPredVal</u>	Scalar function that calculates the negative predictive value or precision rate. It measures the proportion of actual negatives which are correctly identified.

FLFalseDiscRate	Scalar function that calculates the false discovery rate . It measures the proportion of false positives to the total number of positives.
FLMatthewsCorr	Scalar function that calculates the Matthews correlation coefficient. It measures the proportion of false positives to the total number of positives.
FLF1Score	Scalar function that calculates the F1Score. It can be used as a single measure of performance of the confusion matrix.

7. Remaining Analyses in the Tutorial

There are three remaining analyses showcasing the ADS/Fuzzy Logix integration:

- Fuzzy Logix Decision Tree Churn Model Building and Scoring
- Fuzzy Logix Data Preparation for Continuous Models
- Fuzzy Logix Linear Regression Revenue Estimation Model Building and Scoring

What follows is a brief description of each:

Fuzzy Logix Decision Tree Churn Model Building and Scoring

This analysis is very similar to a combination of the “Fuzzy Logix Logistic Regression Churn Model Building” and “Fuzzy Logix Logistic Regression Churn Model Scoring.” In the first Run Unit, FLDecisionTree is called to build a Decision Tree model. Then a SQL Text with Arguments Run Unit is used to query the FLDecisionTree system table, fzzlDecisionTree. Note that a much more complex recursive query would be required to export the rules within the table into a readable format.

Next, the resulting decision tree model is scored with a Run Unit that calls the FLDecisionTreeScore XSP, and a sample of the scored is data queried with the final Run Unit.

Fuzzy Logix Data Preparation for Continuous Models

This analysis is akin to the “Fuzzy Logix Data Preparation for Binomial Models”, with the exception being it creates a new “Deep” table (“TWM_Customer_Deep2”) and it specifies a numeric dependent variable for the purposes of building a linear model in a subsequent Run Unit.

Fuzzy Logix Linear Regression Revenue Estimation Model Building and Scoring

This analysis is very similar to a combination of the “Fuzzy Logix Logistic Regression Churn Model Building” and “Fuzzy Logix Logistic Regression Churn Model Scoring,” except that it estimates a continuous value as opposed to predicting a binomial value. In the first Run Unit, FLLinRegr is called to build a Linear Regression model. Then two SQL Text with Arguments Run Units are used to query the FLLinRegr system tables, fzzlLinRegrCoeffs and fzzlLinRegrStats.

Next, the resulting regression model is scored with a Run Unit that calls the FLLinRegrScore XSP, and a sample of the scored data is queried with the final Run Unit.

Fuzzy Logix K-Means Visualization in Excel

See Section 8 for an overview of the Execute Program/Script Run Unit, a description of this analysis, and Fuzzy Logix Excel templates in general.

8. Using Run Units to Launch Excel and Visualize Fuzzy Logix Results

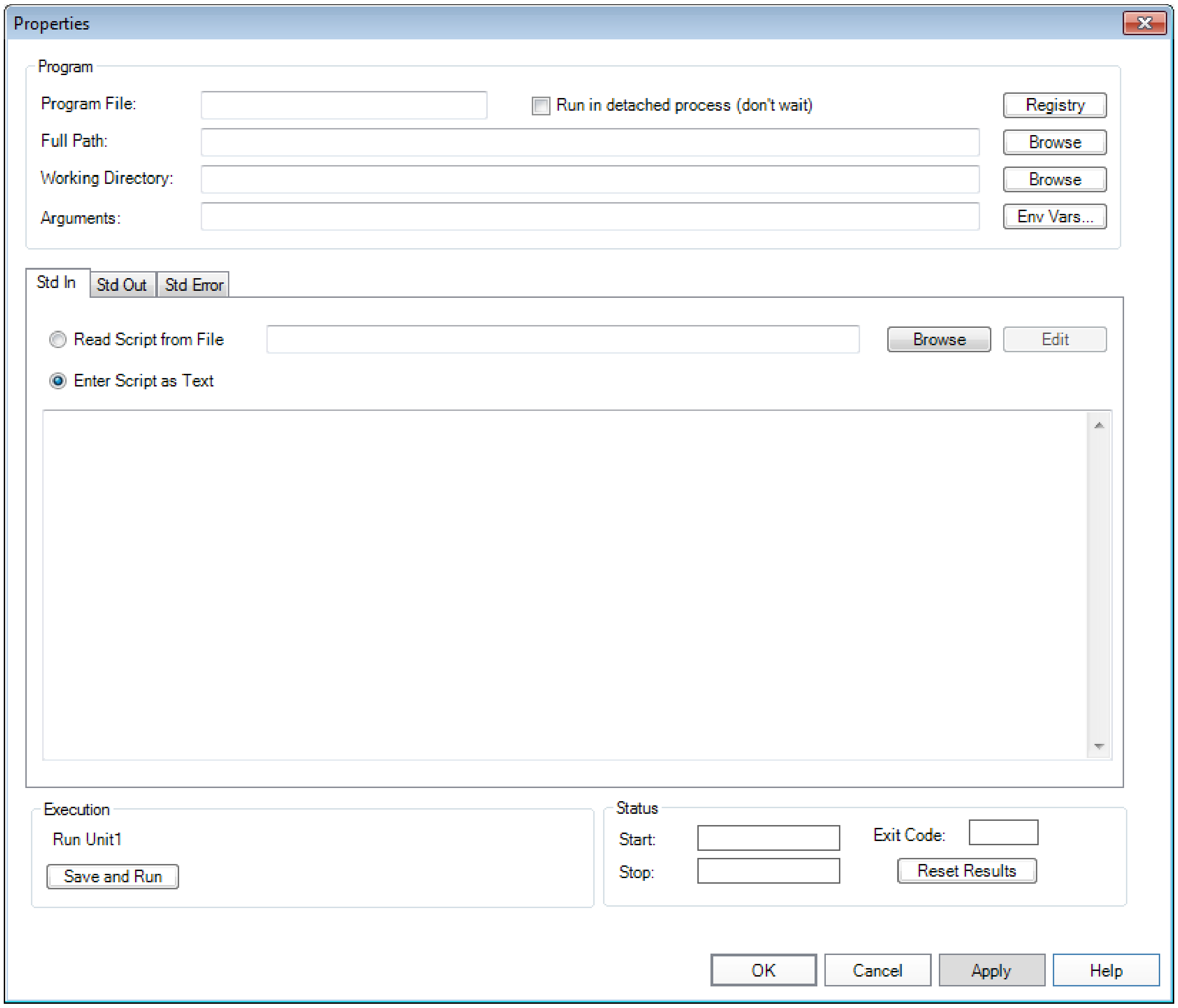
A third type of Run Unit provides a mechanism for visualizing the results of Fuzzy Logix analyses' through Excel Templates. Fuzzy Logix will provide thorough documentation on all the templates available through an appendix in their User Guide. What follows is a brief description of the Execute Program/Script Run Unit, and some examples of available templates.

8.1. Introduction to the Execute Program/Script Run Unit

A program or script may be executed using an Execute Program/Script Run Unit, which executes the program as if from a DOS standard command line, with optional command line arguments and with redirected standard input, output and error messages. Although this can be any command including the invocation of scripting languages such as R, Python, Perl and Ruby, for purposes of the integration, we will focus on Excel.

Note that an Execute Program/Script Run Unit utilizes substitution parameters in a manner similar to a SQL Text with Arguments SQL element, replacing <P1> with the first expression in the Substitution Parameters folder, <P2> for the second, etc.

When you drag and drop this type of Run Unit onto the Run Units Pallet, the following properties screen is displayed:



The fields on this dialog are described below:

<u>Program File</u> This is the name of the program file to be executed, for example "excel.exe".

<u>Run in detached process (don't wait)</u>
Check-box to provide a DOS shell around the executable so that the ADS front-end does not hang waiting upon the process specified by the executable to terminate.

<u>Full Path</u> This is the program name and full directory path, for example "C:\Program Files\Microsoft Office\Office14\EXCEL.EXE". This can be obtained using the *Browse* button, and in some cases the *Registry* button, as described below. (Note that this field is not required if the Program File is located in a directory contained in the user's *Path* environment variable.)

<u>Working Directory</u> Specify to the file being executed what the Working Directory should be. The Working Directory is folder where the Program File reads or writes to without giving a fully qualified name – in other words, that's where it is going to look for or create a file.

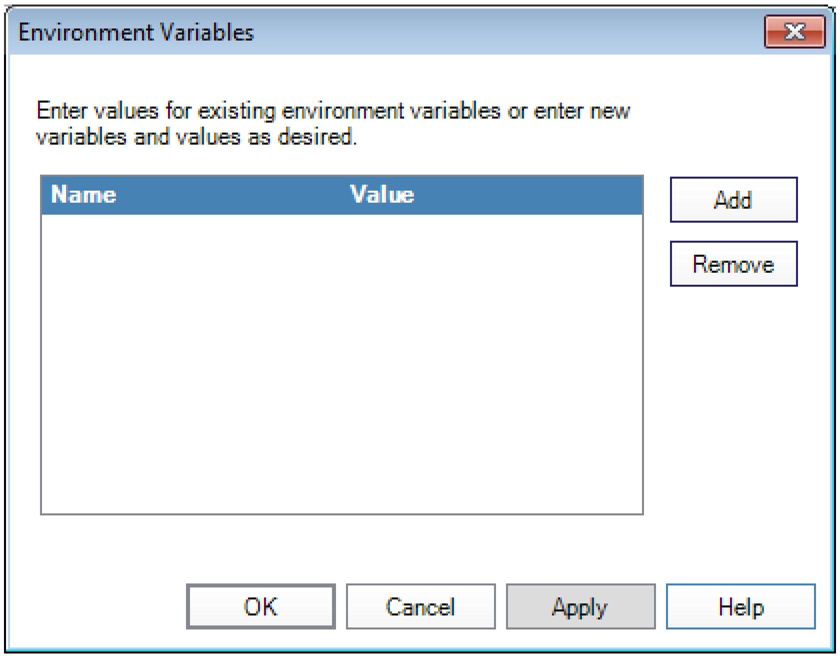
<u>Arguments</u> Command line arguments, if any, should be specified here. These arguments can be Substitution Parameters of the form <Pn> (where 'n' is a number from 1 to 'n')

and are matched up with SQL Elements contained in the folder immediately beneath the "Program/Script Execution" Run Unit node.

Additionally the user can specify a Project Attachment filename as an argument simply by enclosing the name of the Project Attachment within '<' and '>' signs. (For example, the Project Attachment MyFile.txt would be represented as <MyFile.txt>.) When the program or script is executed, the name enclosed in '<…>' is converted to the fully qualified file name (i.e. the path plus the Project Attachment name).

Additionally, a special placeholder for the data source name that the application is currently logged into is allowed. Named <DSN>, it is replaced in the previously listed fields with the current data source name. Note that if the data source name contains spaces, the user should most likely enclose the placeholder in double quotes like this: "<DSN>".

Registry	This button searches the client Registry for the Program File entered above. (Note that in order for the search to be successful, an entry for the Program File must exist in the registry area "*Software\Microsoft\Windows\CurrentVersion\App Paths*", which may not be true.)
Browse	This button presents the standard dialog for locating a file. Once located, the full file path is placed in the *Full Path* text box.
Env Vars	This button presents a dialog for adding/removing Environment Variables as follows:



Enter values for existing environment variables or enter new variables and values as desired.

Std In tab

Read Script From File	If this option is selected, program/script commands are taken from a file.
Browse	The Browse button can be used to locate the script command file above.
Edit	The Edit button can be used to alter the script command file specified, or to extract the script and place it in the *Enter Script as Text* field for additional changes.

Enter Script as Text	Script commands can be entered or copied here as an alternative to reading the script from a file. This may be useful for script development and for storing the script with the analysis.

Std Out tab

Save To File	If this option is selected, the program's Standard Output is written to a file with the specified full name and path.
Append	If checked, this option saves Standard Output to the end of an existing file, retaining the file's previous content.
Browse	The Browse button can be used to locate a file for Standard Output.
Output	This read-only text area displays the Standard Output data from executing the program/script. This data is saved with the analysis.

Std Error tab

Save To File	If this option is selected, Standard Error data is written to a file with this full name and path.
Append	If checked, this option saves Standard Error data to the end of a file, retaining the file's previous content.
Browse	The Browse button can be used to locate the path of the file that Standard Error is being written to.
Error Messages	This read-only text area displays the Standard Error data from executing the program/script. This data is saved with the analysis.

Execution group box area

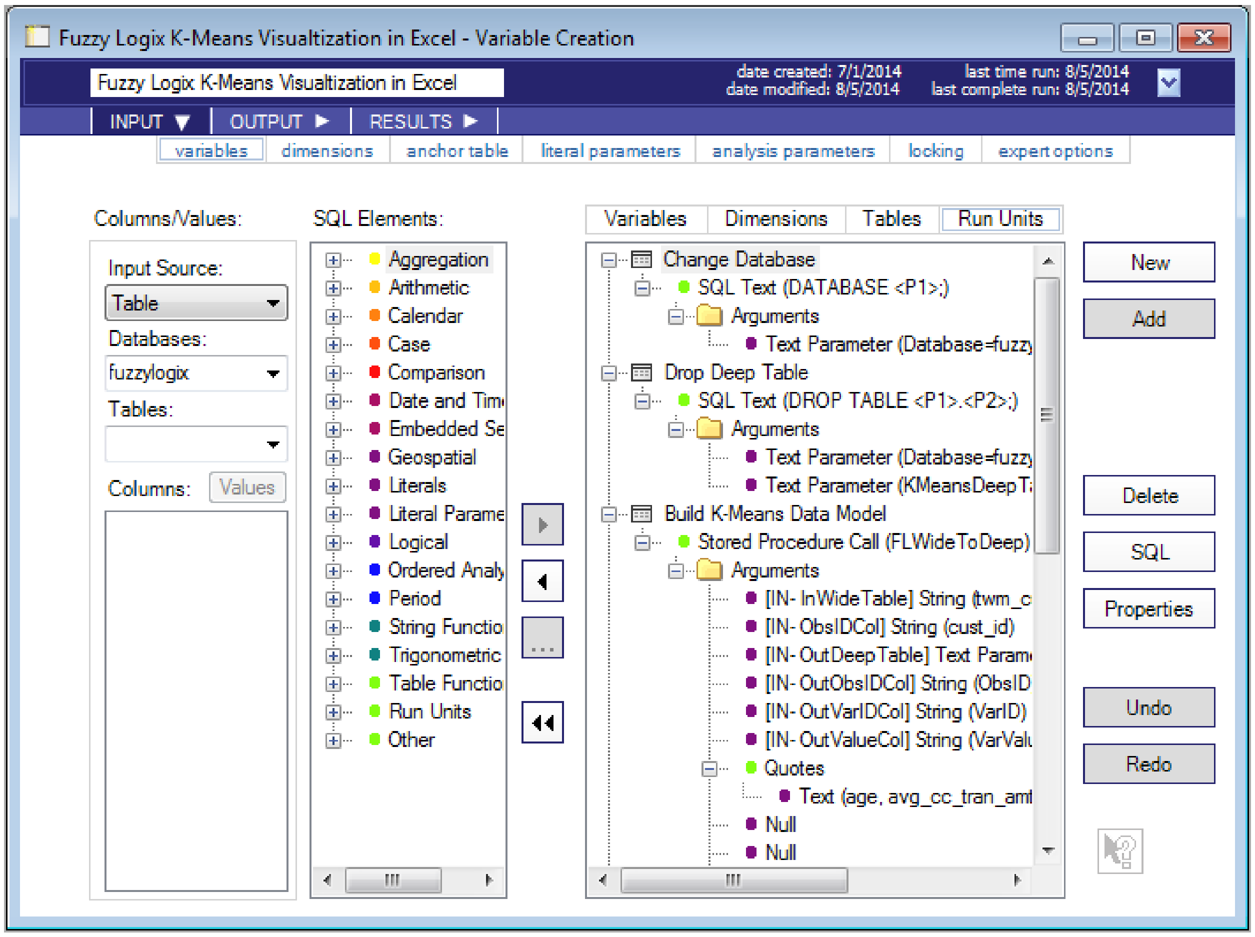
Run Unit Name	This is the name of the Run Unit whose properties are displayed.
Message	This indicates a status of Complete or Cancelled or an error message, if any.
Save and Run button	Selecting this button saves the contents of the Properties dialog and runs the program. This is not enabled if the analysis or an individual Run Unit is already executing. When execution completes, status information is returned along with Standard Output and Standard Error data. If it is necessary to cancel the execution of the script, use the Stop button on the application tool bar.

Status area

Start	This is the start time of the last run.
Stop	This is the stop time of the last run.
Exit Code	This is the exit code of the run: 0 for success or another value for error.
Reset Results	This button clears the Status and Execution fields, along with Standard Out and Standard Error data.

8.2. Fuzzy Logix Visualization Examples via Excel

The final analysis in the project illustrates the use of the Execute Program/Script Run Unit to invoke Microsoft Excel to visualize the results of a Fuzzy Logix K-Means Clustering models. Fuzzy Logix has plans to release many Microsoft Excel visualization templates, documentation for which will be provided by Fuzzy Logix. The goal here is to describe what one such analysis looks like:


Fuzzy Logix K-Means Visualtization in Excel - Variable Creation

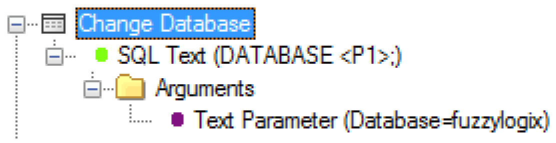
Fuzzy Logix K-Means Visualtization in Excel
date created: 7/1/2014
date modified: 8/5/2014
last time run: 8/5/2014
last complete run: 8/5/2014

```
Change Database
    SQL Text (DATABASE <P1>;)
        Arguments
            Text Parameter (Database=fuzzy
Drop Deep Table
    SQL Text (DROP TABLE <P1>.<P2>;)
        Arguments
            Text Parameter (Database=fuzzy
            Text Parameter (KMeansDeepT
Build K-Means Data Model
    Stored Procedure Call (FLWideToDeep)
        Arguments
            [IN- InWideTable] String (twm_c
            [IN- ObsIDCol] String (cust_id)
            [IN- OutDeepTable] Text Param
            [IN- OutObsIDCol] String (ObsID
            [IN- OutVarIDCol] String (VarID)
            [IN- OutValueCol] String (VarValu
            Quotes
                Text (age, avg_cc_tran_amt
            Null
            Null
```

Each Run Unit is described in detail below:

- <u>Change Database</u>

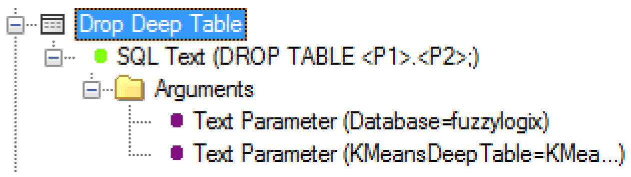
  This SQL Text Run Unit simply changes the current database to the location of the data that will be processed by the Fuzzy Logix K-Means function. It utilizes a Text Literal Parameter for the database name, which is used by subsequent Run Units, including the Run Unit that calls Microsoft Excel:



```
Change Database
    SQL Text (DATABASE <P1>;)
        Arguments
            Text Parameter (Database=fuzzylogix)
```

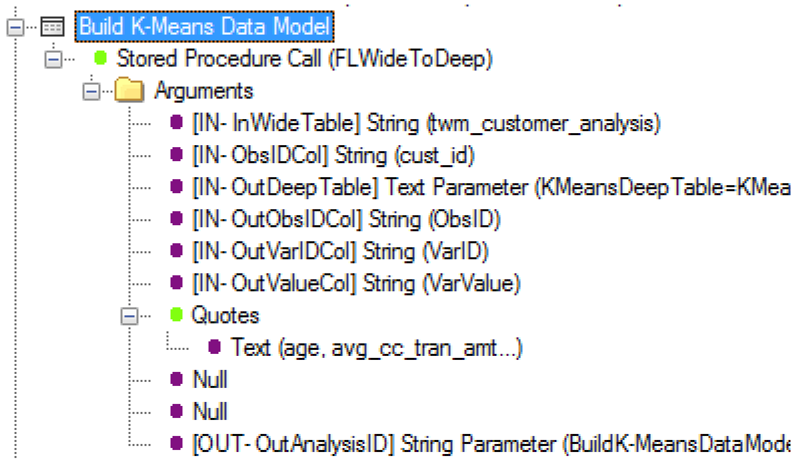
- <u>Drop Deep Table</u>

  This SQL Text Run Unit drops the deep table created in the next step. If this is not done, an error is returned by the Fuzzy Logix FLWideToDeep XSP. The “Database” Text Literal parameter is used as well as an additional Text Literal that represents the table to be dropped (also used below in the Run Unit that calls FLWideToDeep):


Drop Deep Table
SQL Text (DROP TABLE <P1>.<P2>;)
Arguments
Text Parameter (Database=fuzzylogix)
Text Parameter (KMeansDeepTable=KMea...)

- Build K-Means Data Model

  The next Run Unit calls the FLWideToDeep XSP to create the input data for the K-Means modeling algorithm. It utilized the same Text Literal Parameter for the table to create (KMeansDeepTable) as above, as well as a String Literal Parameter for the OUT argument:

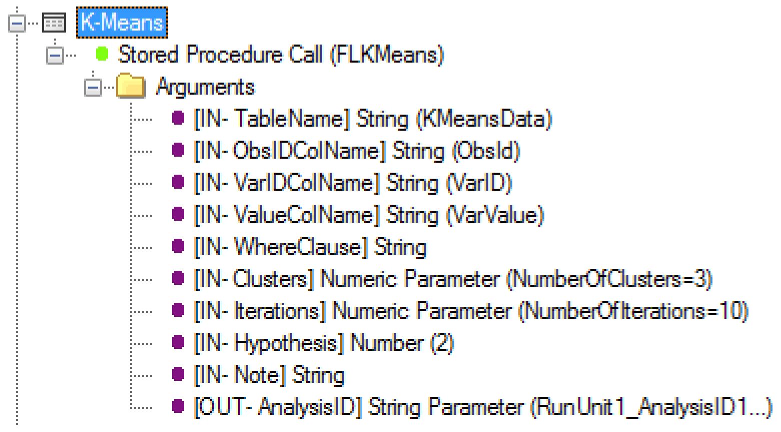

Build K-Means Data Model
Stored Procedure Call (FLWideToDeep)
Arguments
[IN- InWideTable] String (twm_customer_analysis)
[IN- ObsIDCol] String (cust_id)
[IN- OutDeepTable] Text Parameter (KMeansDeepTable=KMea
[IN- OutObsIDCol] String (ObsID)
[IN- OutVarIDCol] String (VarID)
[IN- OutValueCol] String (VarValue)
Quotes
Text (age, avg_cc_tran_amt...)
Null
Null
[OUT- OutAnalysisID] String Parameter (BuildK-MeansDataMode

The function is parameterized as follows:

- InWideTable = twm_customer_analysis
- ObsIdCol = cust_id
- OutDeepTable = KMeansDeepTable Text Literal parameter
- OutObsIDCol = ObsID
- OutVarIDCol = VarID
- OutValueCol = VarValue
- ExcludeCols = (all columns but avg_cc_bal, avg_ck_bal, avg_sv_bal in Quotes)
- ClassSpec = NULL
- WhereClause = NULL
- OutAnalysisID = String Literal Parameter to be set upon execution.

- K-Means

  The next Run Unit calls the FLKMeans XSP to create the clustering model algorithm. It utilizes the data created in the Run Unit above as well as Numeric Literal Parameters for the Number of Clusters and the Number of Iterations, both of which are required by the Microsoft Excel Run Unit. It also uses a String Literal Parameter for the OUT argument:



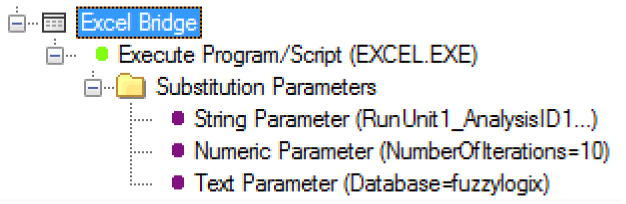
```
K-Means
  Stored Procedure Call (FLKMeans)
    Arguments
      [IN- TableName] String (KMeansData)
      [IN- ObsIDColName] String (ObsId)
      [IN- VarIDColName] String (VarID)
      [IN- ValueColName] String (VarValue)
      [IN- WhereClause] String
      [IN- Clusters] Numeric Parameter (NumberOfClusters=3)
      [IN- Iterations] Numeric Parameter (NumberOfIterations=10)
      [IN- Hypothesis] Number (2)
      [IN- Note] String
      [OUT- AnalysisID] String Parameter (RunUnit1_AnalysisID1...)
```

The function is parameterized as follows:

- TableName = KMeansData
- ObsIdColName = ObsId
- VarIDColName = VarID
- ValueColName = VarValue
- WhereClause = NULL
- Clusters = Numeric Literal Parameter = 3
- Iterations = Numeric Literal Parameter = 10
- Hypothesis = 2
- Note = NULL
- AnalysisID = String Literal Parameter to be set upon execution.

• Excel Bridge

The final Run Unit calls Microsoft Excel, passing the required information needed for graphing the results:

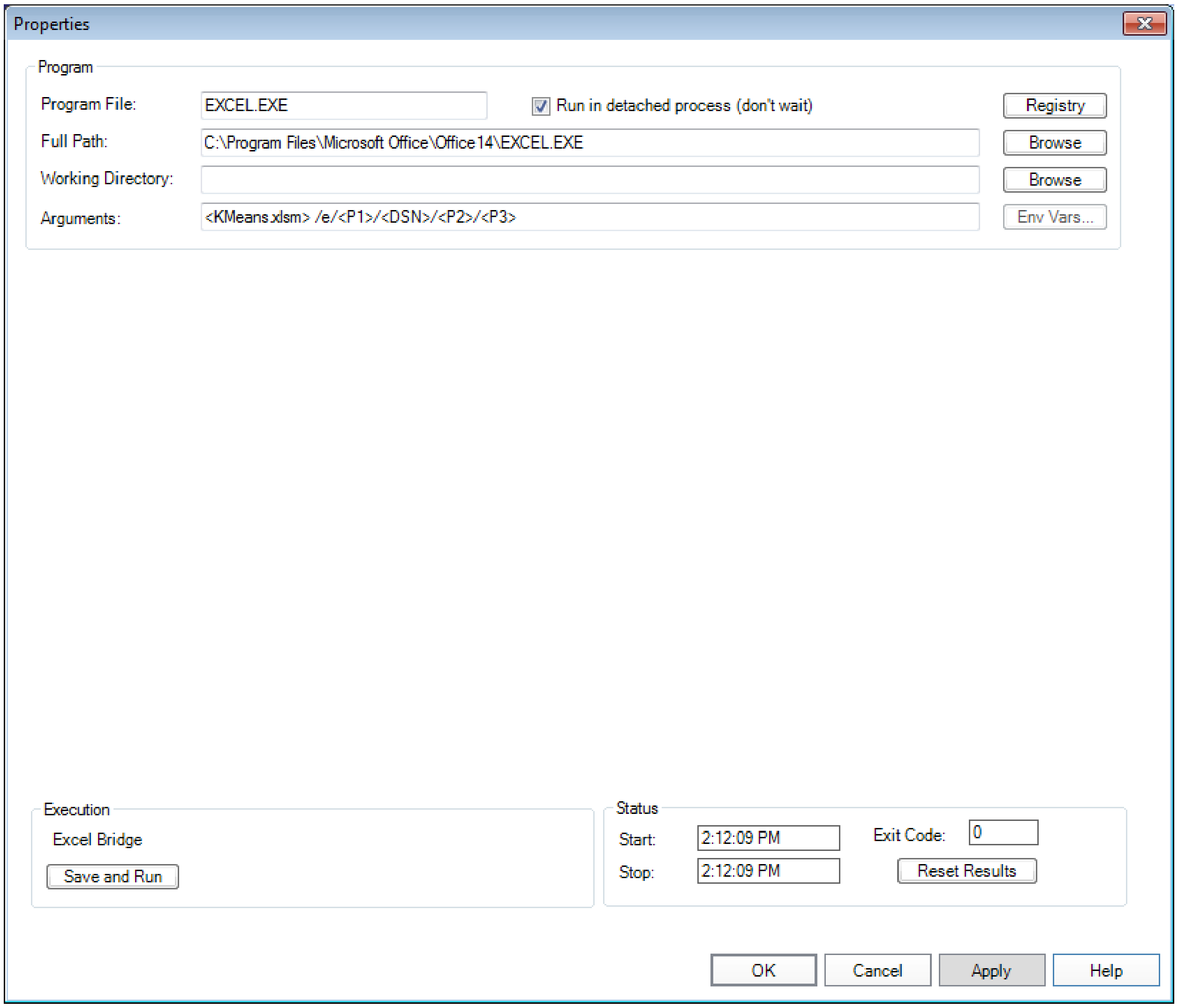


```
Excel Bridge
  Execute Program/Script (EXCEL.EXE)
    Substitution Parameters
      String Parameter (RunUnit1_AnalysisID1...)
      Numeric Parameter (NumberOfIterations=10)
      Text Parameter (Database=fuzzylogix)
```

Three Literal Parameters are passed to Microsoft Excel then used as arguments to the excel.exe command line:

- K-Mean Analysis ID (RunUnit1_AnalysisID1)
- NumberOfIterations = 10
- Database = fuzzylogix

The Run Unit is parameterized as follows:


Properties

Program

Program File: EXCEL.EXE

☑ Run in detached process (don't wait)

Full Path: C:\Program Files\Microsoft Office\Office14\EXCEL.EXE

Working Directory:

Arguments: <KMeans.xlsm> /e/<P1>/<DSN>/<P2>/<P3>

Execution

Excel Bridge

Status

Start: 2:12:09 PM

Stop: 2:12:09 PM

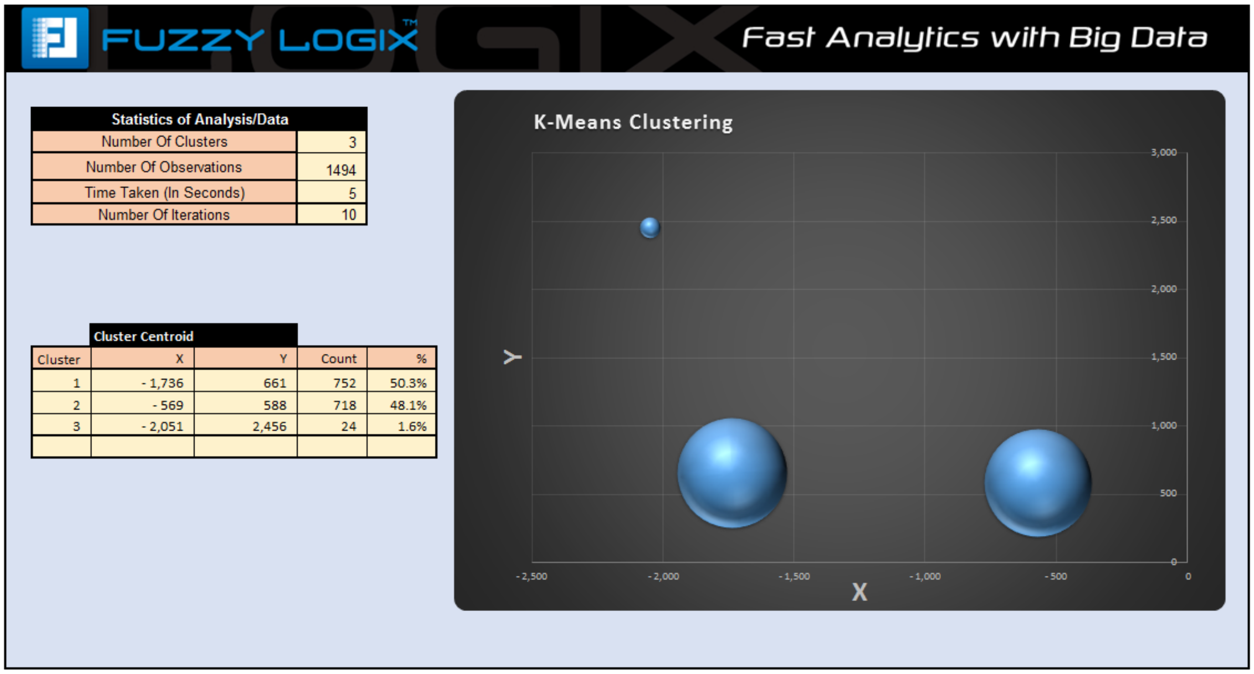
Exit Code: 0

- Program File: EXCEL.EXE
- Run in detached process (don't wait): Enabled
- Full Path: C:\Program Files\Microsoft Office\Office14\EXCEL.EXE
- <KMeans.xlsm> /e/<P1>/<DSN>/<P2>/<P3>

  Where:

  1) <KMeans.xlsm> = Attachment name
  2) /e = Embedded mode (do no bring up splash screen)
  3) <P1> = Analysis ID of K-Means
  4) <DSN> = Datasource Name ADS is currently connected to
  5) <P2> = Number of iterations (10)
  6) <P3> = Database name where results are (fuzzylogix)

Due to the random nature of centroid initialization, the results of K-Means clustering may vary. Note that because this is being run in a detached process, the analysis has an "Execution Complete" status in ADS Generator. Also note it may take some time for Microsoft Excel to connect to the DSN, and read the data required for the graphics from Teradata. Here is an illustrative screen shot of the K-Means visualization:



Statistics of Analysis/Data	
Number Of Clusters	3
Number Of Observations	1494
Time Taken (In Seconds)	5
Number Of Iterations	10

Cluster Centroid

Cluster	X	Y	Count	%
1	- 1,736	661	752	50.3%
2	- 569	588	718	48.1%
3	- 2,051	2,456	24	1.6%