Paper 3440-2019

Adding Data to Real-Time Decision-Making with SAS® Decision Manager on SAS® Viya®

Dean Taplin and Ernest Jessee, SAS Institute Inc., Cary, NC

ABSTRACT

SAS® Decision Manager enables users to operationalize analytics in real-time settings. In these deployments, analytics can create efficiencies and optimize your business process. However, to properly leverage the full value of analytical decision-making processes, users need access to their data. SAS Decision Manager offers that ability, enabling database queries within their real-time flows. This paper tours this ability, showing how to configure a database connection within SAS® Environment Manager and write custom DS2 code that queries a database within SAS Decision Manager. The results of the query are added to the data stream of the decision flow for further processing.

INTRODUCTION

To demonstrate the use of data retrieved from an external database within a SAS Decision Manager decision flow, this paper explores a scenario where a car dealer wishes to calculate a selling price for a car on his lot. The dealer knows the year, make, model, and type of the car, and he knows the premium options that the car is equipped with. A third-party database can provide the base value of the car along with additional value for each premium feature. These amounts can be added together to reveal an appropriate selling price. The dealer would like a REST API that can be accessed from his internal website to calculate an appropriate price for a car given the known information.

This car pricing REST API is achieved using a decision flow created in SAS Decision Manager, which contains a custom code node and a rule set. This flow is deployed to the SAS® Micro Analytics Server, which creates the REST API.

ARCHITECTURE

The car pricing decision flow executes an SQL query against a Postgres database as shown in Figure 1. The SQL query is executed in a code node. A code node allows custom DS2 code to be inserted into a decision flow. This DS2 code queries the database and assigns the returned information to decision variables, which can be used downstream in other parts of the flow. SAS Environment Manager provides a means of defining the database connection that is referenced in the code node. The database used in this example is Postgres, but other database technologies are applicable.

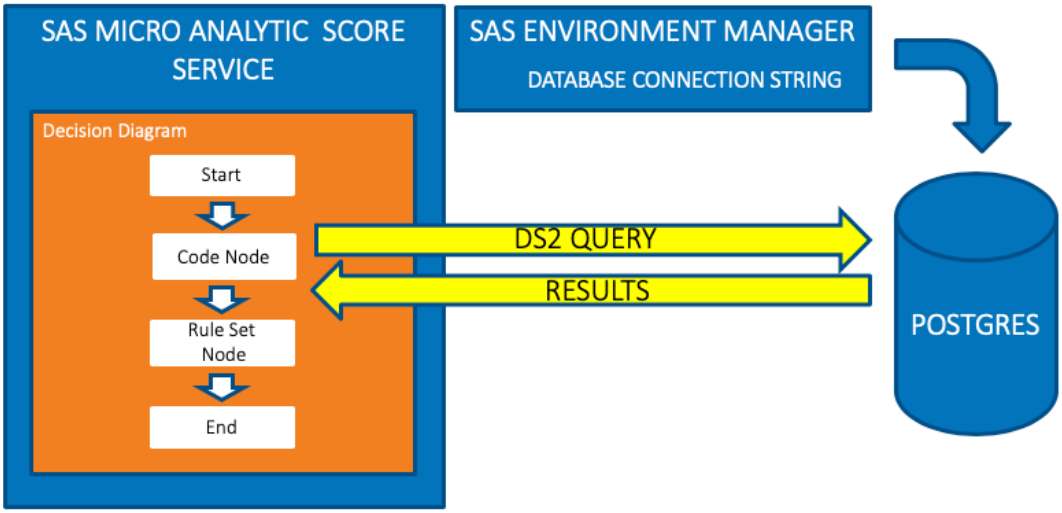

Figure 1. The Architecture of a Decision with SQL Calls

THE CAR PRICING DECISION

The car pricing decision contains 2 nodes: a code node and a rule set node. The code node queries a database using specifics of a particular car and inserts the result returned from that query into the decision flow's data stream. The rule set node combines the data gathered by the code node to result in a selling price. The car pricing decision is shown in Figure 2.

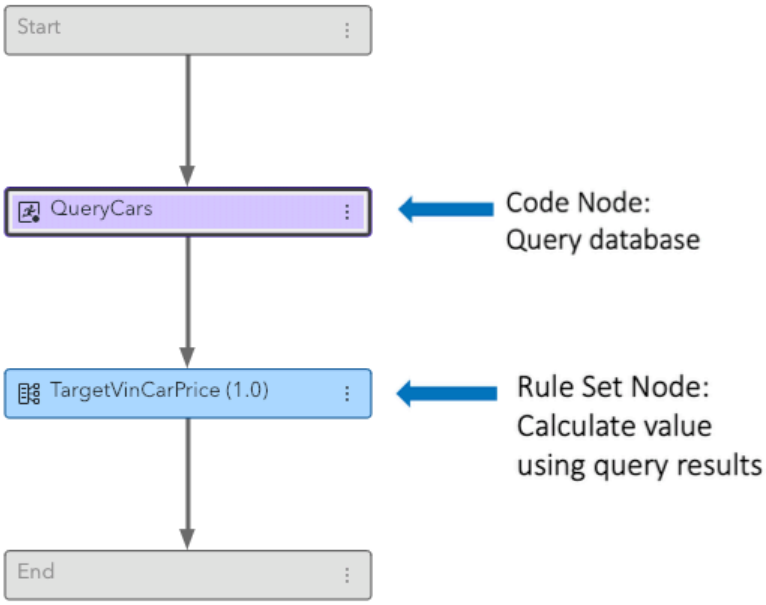

Figure 2. The Car Pricing Decision Flow

QUERYCARS CODE FILE

The code contained in the "QueryCars" code node is shown in its entirety in Appendix A and is explained in the following sections.

Input and Output Data

The code node uses the input data entered by the car dealer for year, type, make, and model of a particular car to construct an SQL query. Executing this SQL query against the database returns a base value for the car along with information about premium features. The query result becomes the output of the code file node. The input and output tables for the code file node are shown in Figure 3.

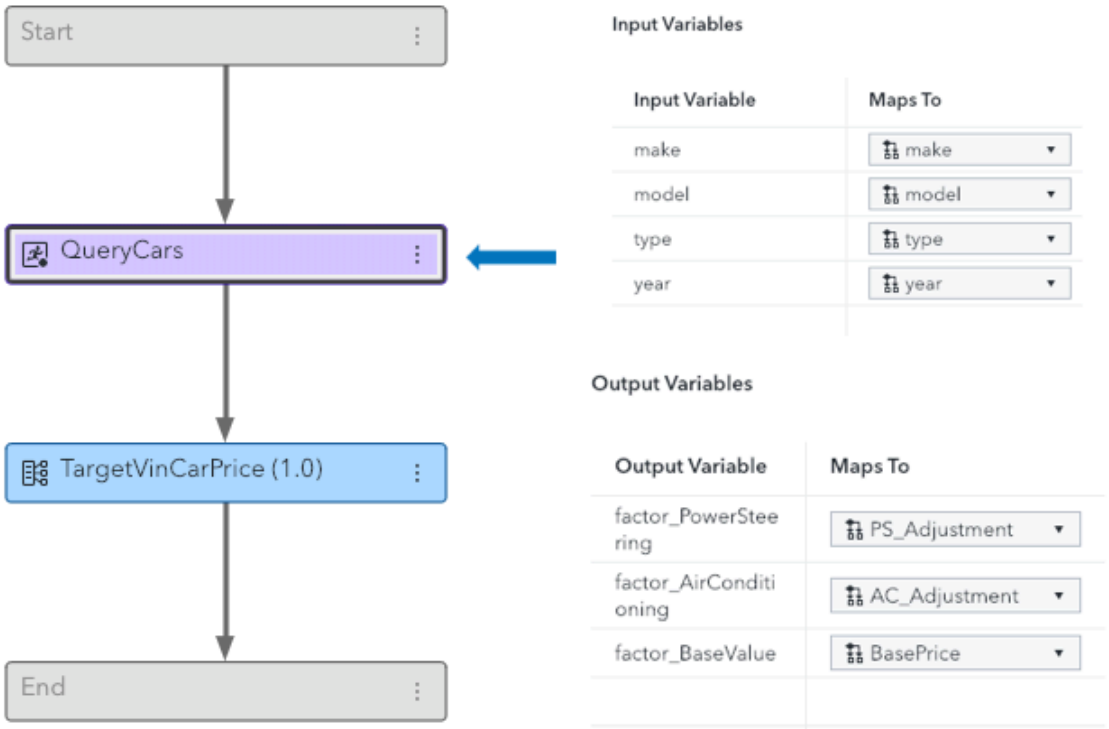

Figure 3. Input and Output for the QueryCars Code Node

Query Code

The code node, "QueryCars", uses the DS2 SQLSTMT package, which allows queries to a pre-defined data source. An SQL Select is executed against the database table, "carslookup". Here is the DS2 code that queries the database:

```
dcl package sqlstmt lookupDataSource;
dcl varchar(100) sql;
dcl int rc;
dcl int initComplete;

method loadLookup();
      myCustomLookupHash.clear();
```

```
      sql = 'SELECT make,model,type,year,baseValue,ACValue,PSValue from "carslookup"';
      lookupDataSource = _new_ sqlstmt(sql);
      rc = lookupDataSource.execute();
      if rc ne 2 then
         do while (lookupDataSource.fetch([_make _model _type _year baseValue ACValue PSValue])
                     eq 0);
            /* use strip() upcase() to make hash key matching more forgiving */
            _make = upcase(strip(_make));
            _model = upcase(strip(_model));
            _type = upcase(strip(_type));
            myCustomLookupHash.ref([_make _model _type _year],[baseValue ACValue PSValue]);
      end;

      lookupDataSource.closeResults();
      initComplete=1;
end;
```

Marshaling Code

Data marshaling takes place in the Execute method of the code node. "in_out" variables included in the Execute method signature are used to pass the data pulled from the database by the query. When SAS Decision Manager parses the signature of the execute method, all "in_out" variables are added to the node's output list while others are added to the input list. Here is the marshaling code:

```
method execute(varchar(52) make,
                     varchar(160) model,
                     varchar(32) type,
                     double year,
                     in_out double factor_BaseValue,
                     in_out double factor_AirConditioning,
                     in_out double factor_PowerSteering);

     if missing(initComplete) then loadLookup();
         /* assign keys - variable lists must reference global variables */
         /* use strip() upcase() to make hash key matching more forgiving */
         _make = upcase(strip(make));
         _model = upcase(strip(model));
         _type = upcase(strip(type));
         _year = year;
         if myCustomLookupHash.find([_make _model _type _year],
                                                  [baseValue ACValue PSValue]) = 0 then do;
             factor_BaseValue = baseValue;
             factor_AirConditioning = ACValue;
             factor_PowerSteering = PSValue;
         end;
         else do;
             /* In this example use static values */
             factor_BaseValue = 200; /* scrap metal value */
             factor_AirConditioning = .;
             factor_PowerSteering = .;
         end;
     end;
end;
```

PRICE ADJUSTMENT RULE SET

The data retrieved with the SQL query is combined for a total price using the rule set, "TargetVinCarPrice".

Input and Output Data

The input and output tables for the rule set node are shown in Figure 4.

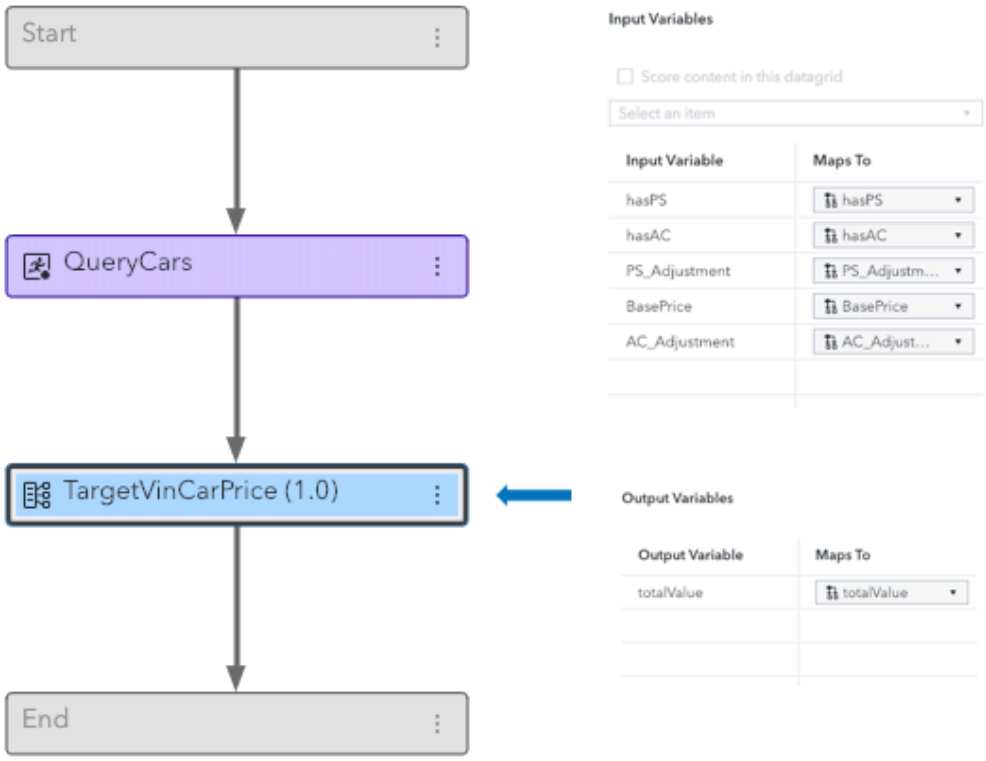

Figure 4. Input and Output Tables for TargetVinCarPrice Rule Set

Price Adjustment Rules

TargetVinCarPrice consists of a single assignment and two rules. The selling price, "totalValue", is initially assigned to be equal to "BasePrice" as returned from "QueryCars." For each premium feature that the car has, "totalValue" is increased by the value of that premium feature. The specifics of the price adjustment rule set are shown in Figure 5.

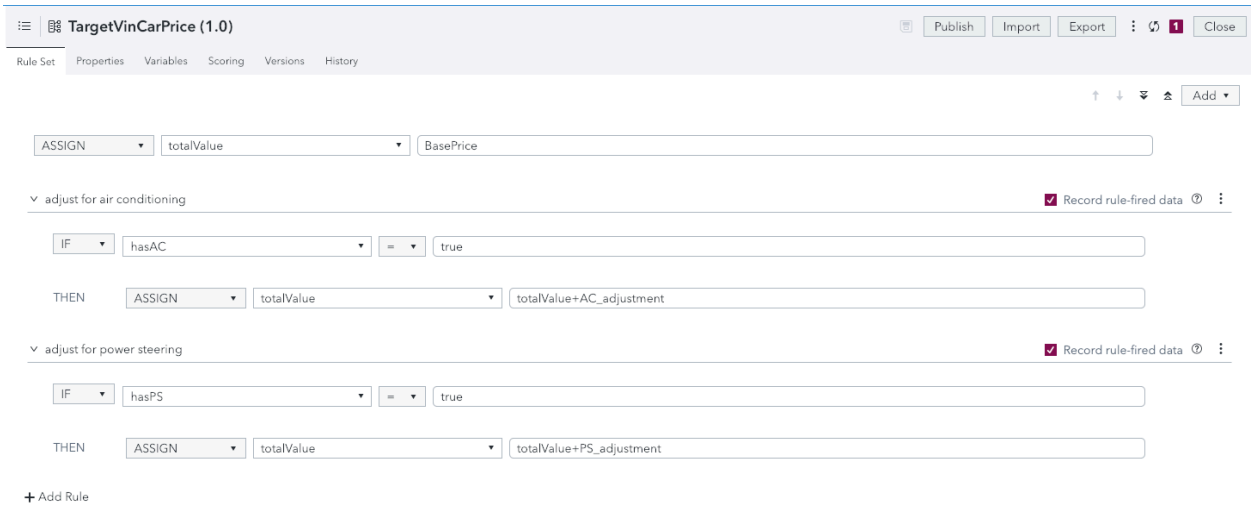

Figure 5. TargetVinCarPrice Rule Set

CONFIGURATION

In order to access a third-party party database from SAS® Micro Analytic Score Service, the SAS Micro Analytic Score Service must be configured to access that database. This configuration entails installing appropriate drivers on the host server, insuring that these drivers are loaded when the service is started and defining a connection string in SAS Environment Manager.

After following the steps in this section if there are problems accessing the database from a code node, follow the steps in Appendix B to use system logging for diagnosis of the problem.

Database Drivers

Database drivers relevant to the database to be accessed must be installed on the server that hosts the SAS Micro Analytic Score Service. See the documentation from the database vendor for instructions.

Database Connection String

At start up, the SAS Micro Analytic Score Service connects to all data sources represented in its FEDSQL connection string. For a Postgres database, the FEDSQL connection string entry follows this pattern:

```
DRIVER=SQL;CONOPTS=((DRIVER=<engine>;CATALOG=<dbms-
catalog>;UID=<userid>;PWD=<password>;SERVER=<server>;PORT=<port>;DB=<da
tabase>))
```

To modify the FEDSQL connection string for the SAS Micro Analytics Service, open SAS Environment Manager as a system administrator. Navigate to the following category within the hierarchy: **Configuration** > **All services** > **Micro Analytic Score service**. Next, enter "FEDSQL" in the search to reveal a section labeled "core". The "core" section contains the "connectionString" property. Modify this property to contain the connection string entry for your database. See Figure 6 for an example of a FEDSQL connection string in SAS

Environment Manager.

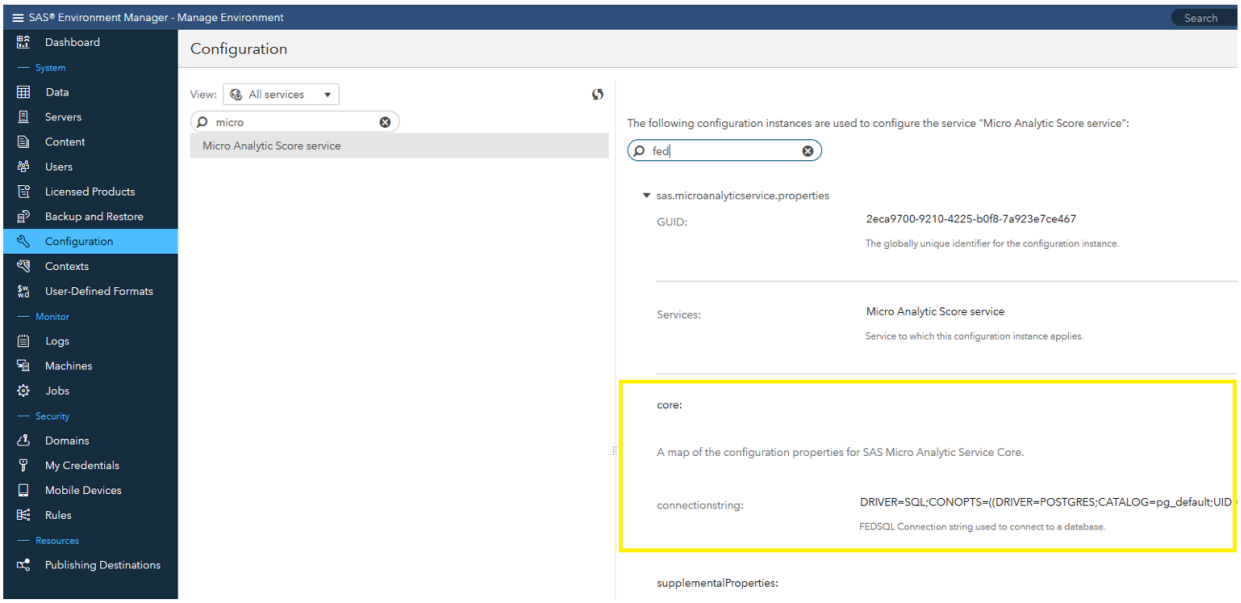

Figure 6. SAS Environment Manager Showing the MAS FEDSQL Connection String

SAS Micro Analytic Score Service Configuration File

In order for the SAS Micro Analytic Score Service to load needed drivers for accessing the database pointed to by the defined connection string, the execution environment of the service must be modified. Environment changes that must be picked up upon the start up of the SAS Micro Analytic Score Service are defined in a special configuration file. On the host server, create a text file called microanalyticservice.conf in the following directory:

```
/opt/sas/viya/config/etc/sysconfig/
```

For the Postgres database used in this example, the contents of this file are as follows:

```
PGLIENTENCODING=UTF-8
export PGCLIENTENCODING
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/sas/viya/home/lib64/
```

The contents of this file will vary depending on the database vendor.

After the configuration file is in place, the SAS Micro Analytic Score Service should be restarted. The micro analytic score service is typically named sas-viya-microanalyticservice-default.

DEMONSTRATION

This section will demonstrate the results of the process described above which has enabled a car dealer to create decision flow that retrieves data from a third-party database and combines that data to calculate an appropriate selling price for a particular car. In our demonstration, the decision is deployed to the SAS Micro Analytic Score Service resulting in a REST API that reveals the analytical power of the decision flow. The dealer then accesses this API from his own website. Known information is entered by his employees and is then sent to the REST API, which returns a price. As a result, his employees can quickly and accurately price cars.

THE DATABASE

The third-party database the dealer wishes to use is a "blue-book" database, which contains a base price for cars and adder values based on options the car might have. Typical data from the database used in this sample is shown in Figure 7.

	make character varying(52)	model character varying(160)	type character varying(32)	year double precision	basevalue double precision	acvalue double precision	psvalue double precision
1	Acura	3.5 RL 4dr	Sedan	1990	11148	1114	557
2	Acura	3.5 RL 4dr	Sedan	1991	12337	1233	616
3	Acura	3.5 RL 4dr	Sedan	1992	16794	1679	839
4	Acura	3.5 RL 4dr	Sedan	1993	11597	1159	579
5	Acura	3.5 RL 4dr	Sedan	1994	15310	1531	765
6	Acura	3.5 RL 4dr	Sedan	1995	3563	356	178
7	Acura	3.5 RL 4dr	Sedan	1996	3737	373	186
8	Acura	3.5 RL 4dr	Sedan	1997	6078	607	303
9	Acura	3.5 RL 4dr	Sedan	1998	16289	1628	814
10	Acura	3.5 RL 4dr	Sedan	1999	14250	1425	712
11	Acura	3.5 RL 4dr	Sedan	2000	7185	718	359
12	Acura	3.5 RL 4dr	Sedan	2001	3123	312	156
13	Acura	3.5 RL 4dr	Sedan	2002	3498	349	174
14	Acura	3.5 RL 4dr	Sedan	2003	2881	288	144
15	Acura	3.5 RL 4dr	Sedan	2004	2418	241	120
16	Acura	3.5 RL 4dr	Sedan	2005	3206	320	160
17	Acura	3.5 RL 4dr	Sedan	2006	2718	271	135
18	Acura	3.5 RL 4dr	Sedan	2007	3290	329	164
19	Acura	3.5 RL 4dr	Sedan	2008	2460	246	123
20	Acura	3.5 RL 4dr	Sedan	2009	2280	228	114
21	Acura	3.5 RL 4dr	Sedan	2010	2614	261	130
22	Acura	3.5 RL 4dr	Sedan	2011	2511	251	125

Figure 7. Third-Party Blue-Book Database

MICRO ANALYTIC SERVICE MODULE

After the decision has been created, the power of SAS Decision Manager is made available as a REST service by publishing the decision to SAS Micro Analytic Score Service. This is accomplished by clicking the **Publish** button from within the decision editor and selecting **maslocal** as the target destination. The Publish Decisions window is shown in Figure 8.

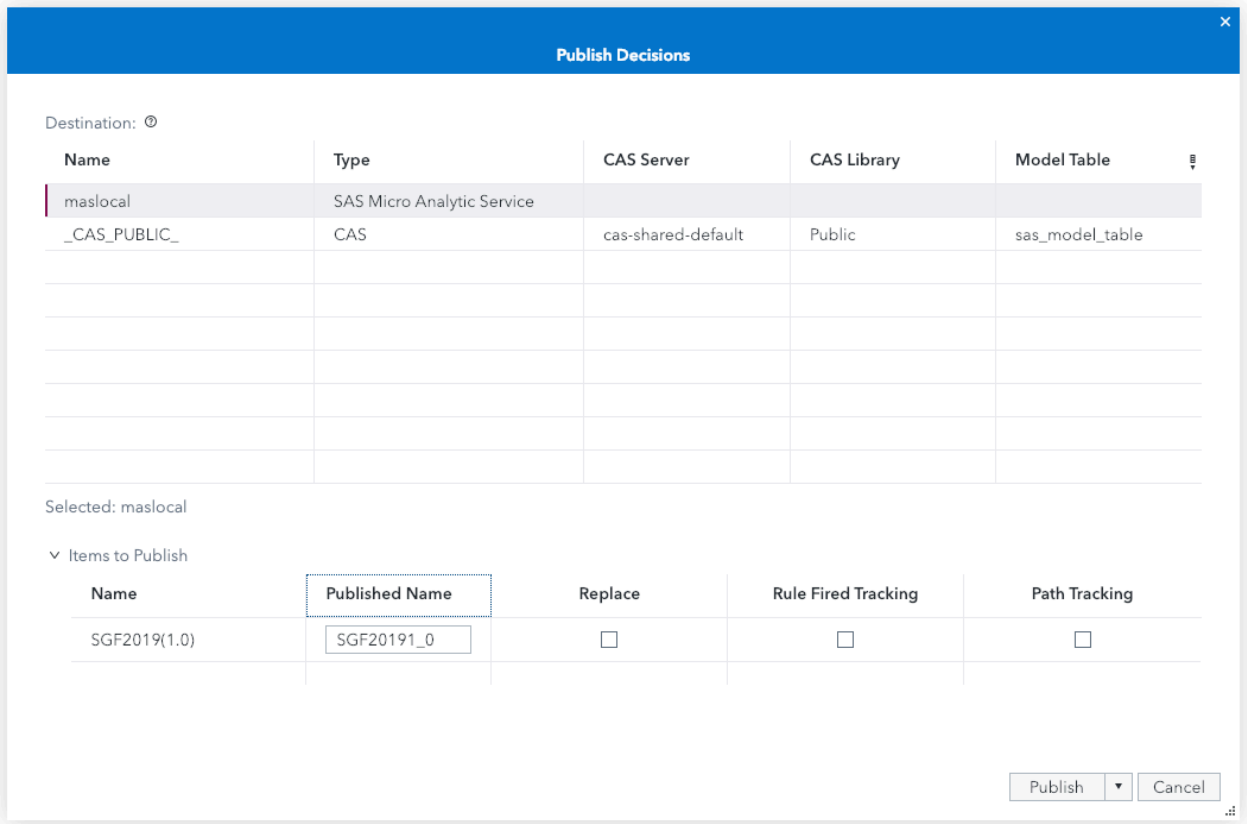

Figure 8. Publish Decisions Window

Figure 9 shows the URL of the published decision's Micro Analytic Service module in the **Module URI** column of the history table. This URL is used in your web application to access the logic of the decision.

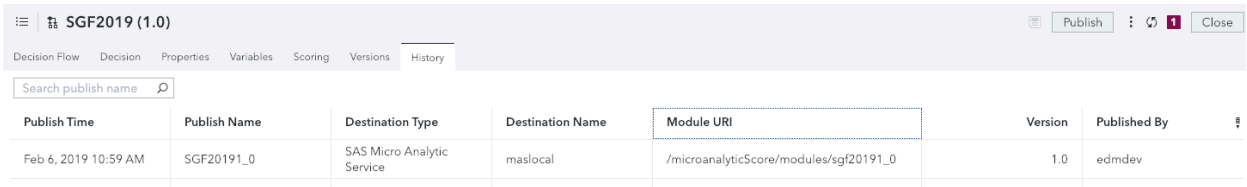

Figure 9. History Tab of the Decision Editor Showing the Published Decision's URL

Starting with the URL shown in the history table and using any REST client, one can discover the exact format of the input data that is expected by the Execute step of the published decision as shown in Figure 10.

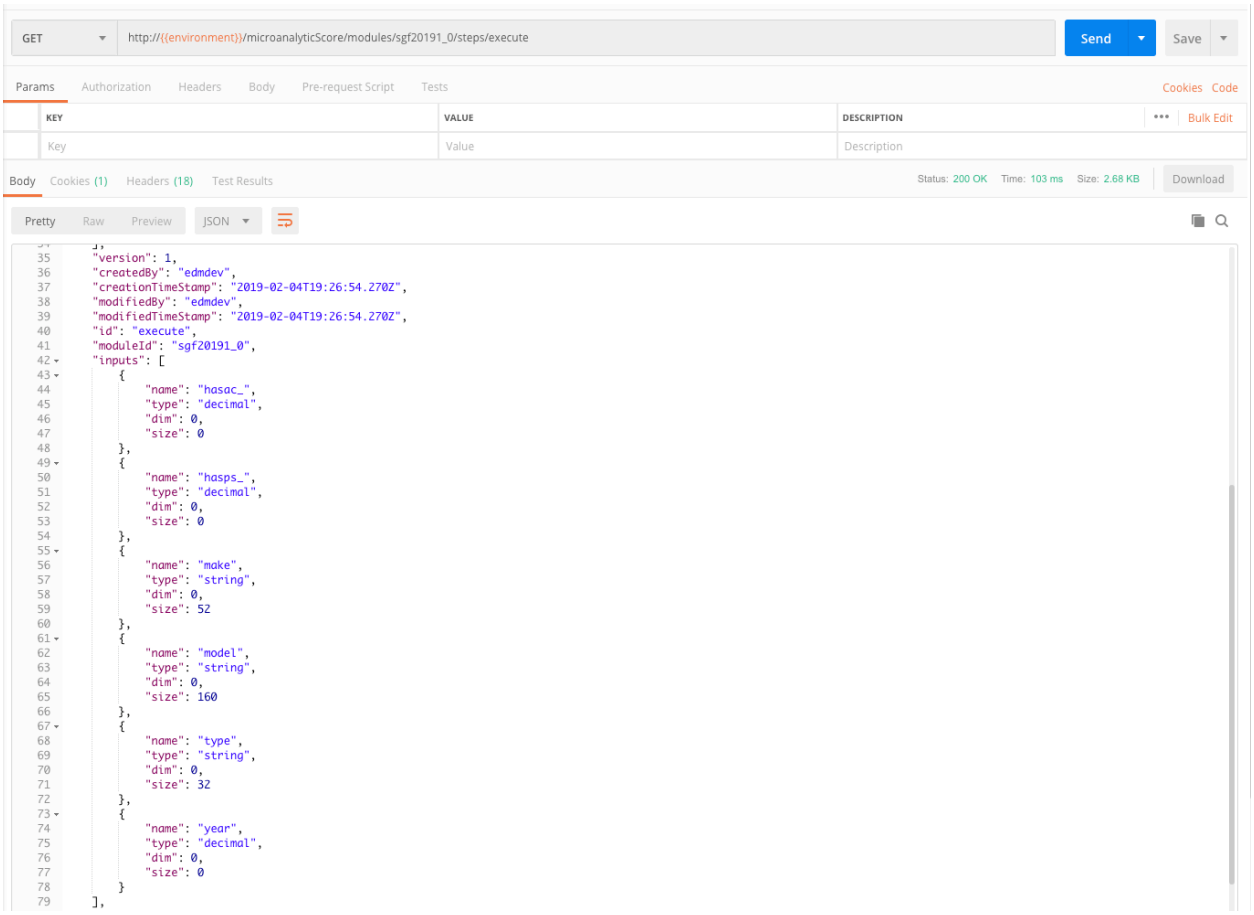

Figure 10. Discovery of the Expected Execute Inputs Using a REST Client

THE DECISION IN USE

Executing the REST API created from the car pricing decision is as simple as posting the expected input data and processing the returned payload. An example of the Car Pricing decision in action is shown in Figure 11.

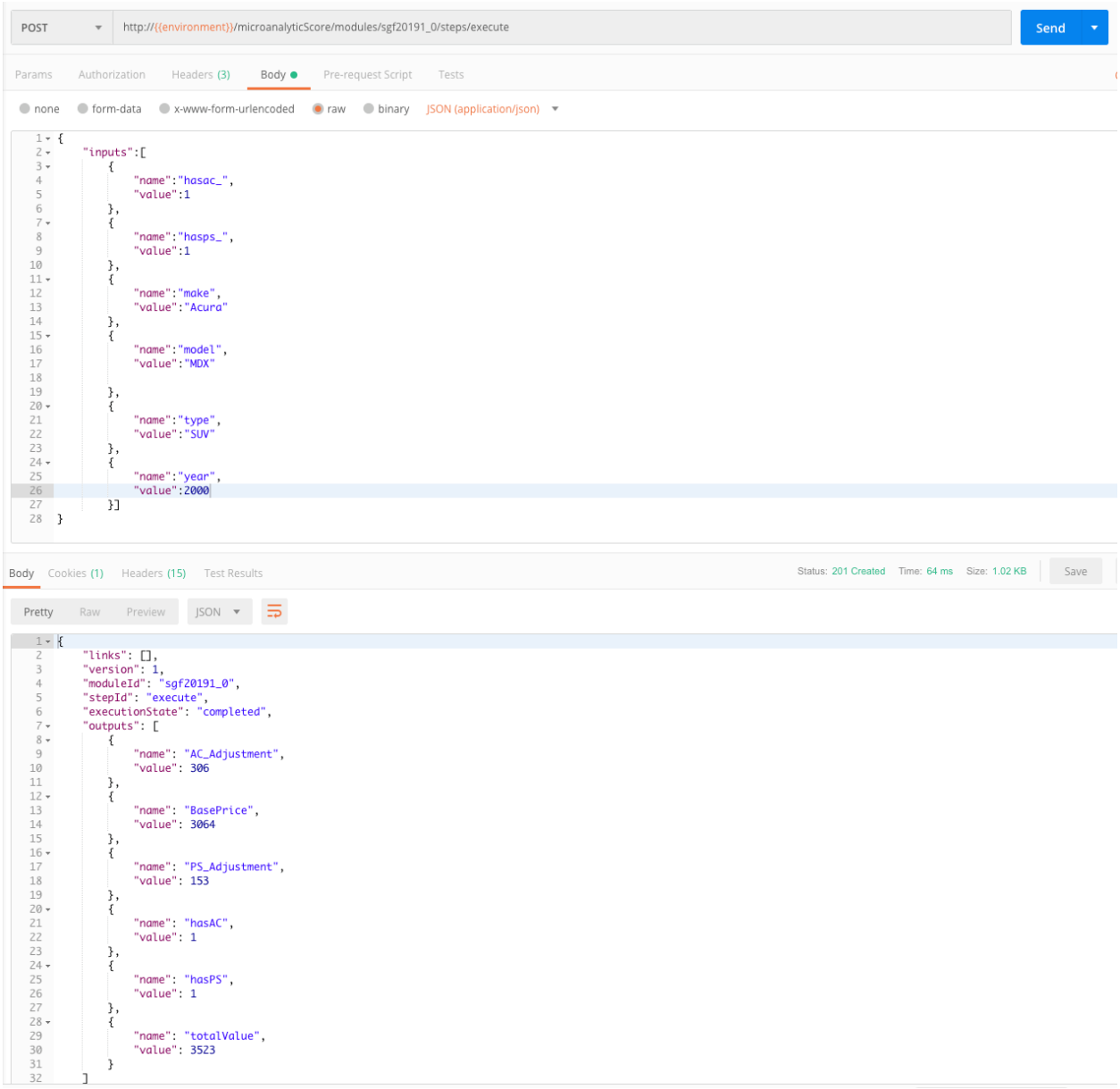

Figure 11. Exercising the Car Pricing Decision Using a REST Client

CONCLUSION

In order for a decision process to be effective, it is often necessary to reference data that does not reside within the SAS® Viya® environment. SAS Decision Manager offers a fantastic method for accessing external data within a decision. The SAS Micro Analytic Score Service can quickly transform that decision into a REST API enabling users to harness the power of SAS from their own applications.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Ernest Jessee
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
919-677-8000
Ernest.jessee@sas.com

APPENDIXES

Appendix A. The DS2 Code for QueryCars

```
package "${PACKAGE_NAME}" / inline ;

      /* lookup keys are managed internal to the package */
      dcl int _year;
      dcl varchar(52) _make;
      dcl varchar(160) _model;
      dcl varchar(32) _type;
      dcl double baseValue;
      dcl double ACValue;
      dcl double PSValue;

      dcl package hash myCustomLookupHash ([_make _model _type _year],
                                          [baseValue ACValue PSValue]);
      dcl package sqlstmt lookupDataSource;
      dcl varchar(100) sql;
      dcl int rc;
      dcl int initComplete;

      method loadLookup();
            myCustomLookupHash.clear();
            sql = 'SELECT make,model,type,year,baseValue,ACValue,PSValue from "carslookup"';
            lookupDataSource = _new_ sqlstmt(sql);
            rc = lookupDataSource.execute();

            if rc ne 2 then
                do while (lookupDataSource.fetch([_make _model _type _year baseValue ACValue
                          PSValue]) eq 0);
                  /* use strip() upcase() to make hash key matching more forgiving */
                  _make = upcase(strip(_make));
                  _model = upcase(strip(_model));
                  _type = upcase(strip(_type));
                  myCustomLookupHash.ref([_make _model _type _year],
                                         [baseValue ACValue PSValue]);
                end;

            lookupDataSource.closeResults();
            initComplete=1;
      end;

      method execute(varchar(52) make,
                     varchar(160) model,
                     varchar(32) type,
                     double year,
                     in_out double factor_BaseValue,
                     in_out double factor_AirConditioning,
                    in_out double factor_PowerSteering);

            if missing(initComplete) then loadLookup();

            /* assign keys - variable lists must reference global variables */
            /* use strip() upcase() to make hash key matching more forgiving */
            _make = upcase(strip(make));
            _model = upcase(strip(model));
            _type = upcase(strip(type));
            _year = year;

            if myCustomLookupHash.find([_make _model _type _year],
                                       [baseValue ACValue PSValue]) = 0 then do;
                  factor_BaseValue = baseValue;
                  factor_AirConditioning = ACValue;
                  factor_PowerSteering = PSValue;
            end;
            else do;
                  /* In this example use static values */
                  factor_BaseValue = 200; /* scrap metal value */
                  factor_AirConditioning = .;
                  factor_PowerSteering = .;
            end;
      end;
endpackage;
```

Appendix B. Troubleshooting Database Connection Issues

If there are issues connecting to your database, you can enable logging to help diagnose the problem. In SAS Environment Manager, navigate to the following category: **Configuration** > **All services** > **Micro Analytic Score Service**. Here, set the logger, "Audit.Table.Connection", to "DEBUG" as shown in Figure 12.

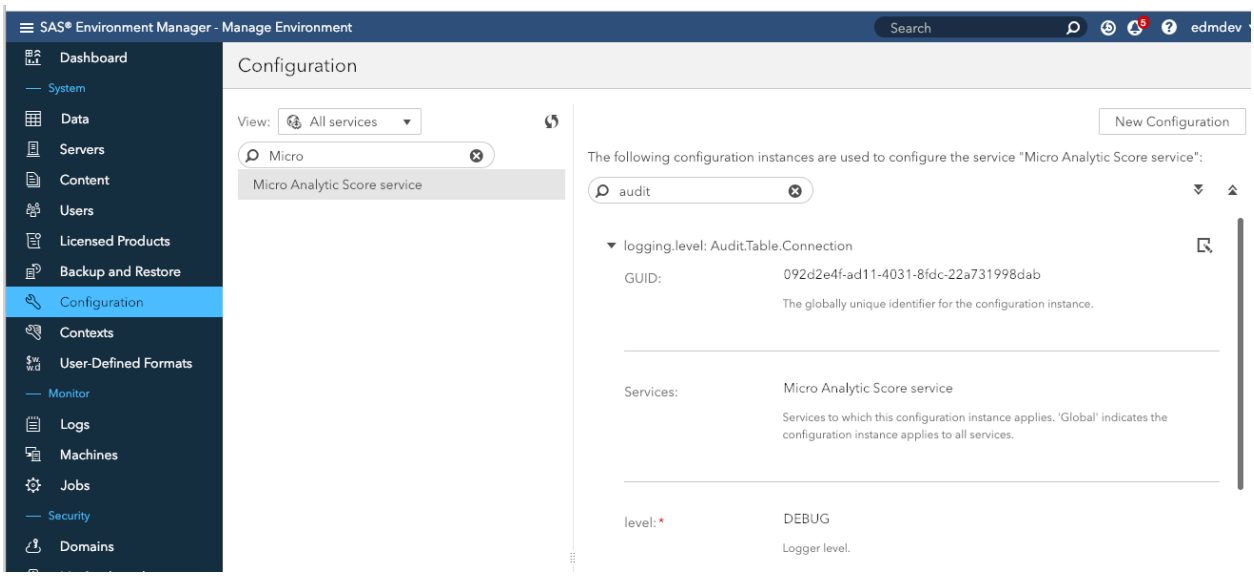

Figure 12. Micro Analytic Service Table Connection Logging

After setting this log level, observe the micro analytics service log at

/opt/sas/viya/config/var/log/microanalyticservice/default