Paper 3390 - 2015

Working with PROC FEDSQL in SAS® 9.4

Zabiulla Mohammed and Ganesh Kumar Gangarajula, Oklahoma State University
Pradeep Kalakota, Business Intelligence Developer II, FHL Bank of Des Moines

ABSTRACT

Working with multiple data sources from multiple vendors simultaneously in SAS had not been straight forward until the PROC FEDSQL was introduced in the SAS® 9.4 release. FEDSQL, Federated Query Language is a vendor independent language which provides a common SQL syntax to communicate across multiple relational databases without having to worry about vendor specific SQL syntax. PROC FEDSQL is SAS implementation of the FEDSQL language. PROC FEDSQL enables us to write federated queries which can be used to perform joins on tables from different databases with a single query, without having to worry about loading the tables into SAS individually and combining them using DATA step and PROC SQL statements. The objective of this paper is to demonstrate the working of PROC FEDSQL to fetch data from multiple data sources such as Microsoft SQL Server database, MySQL database and a SAS dataset and run federated queries on all the data sources. Other powerful features of PROC FEDSQL such as transactions and FEDSQL pass-through facility will be discussed briefly.

INTRODUCTION

FEDSQL is a new addition to SAS 9.4, providing extensive data type support, ability to access and manage large data through multiple data sources in an optimized multi-threaded way for high performance. To support data from multiple data sources, FEDSQL follows a common SQL syntax, which is neutral to all the data sources and allows the users to not submit individual queries which are specific to each data source.

BENEFITS OF FEDSQL

The benefits of FEDSQL exceed that of SQL as described below:

- The conformance of FEDSQL to ANSI SQL: 1999 core standard allows it to process queries in its own language and the native languages of other data sources which conform to the standard. This conformance makes the FEDSQL queries vendor neutral.
- FEDSQL provides precision via new data types such as bigint, decimal, double etc. Apart from providing extended data type support, FEDSQL allows translation from and to legacy SAS data types such as SAS numeric and SAS character.
- FEDSQL can handle federated queries to join tables from multiple data sources and provide a single dataset.
- FEDSQL can generate the resultant data to be in any of the supported data sources irrespective of what the input data sources. This enables the users to tailor the data sources as needed.

WAYS TO RUN FEDSQL PROGRAMS

FEDSQL code be submitted in multiple ways:

- Using PROC FEDSQL from Base SAS Language Interface
- JDBC, ODBC or OLE DB client using SAS Federation Server
- Base SAS session using SAS Federation Server LIBNAME engine and PROC SQL pass-through
- From a DS2 program
- From SPD Server using SQL pass-through facility or through an ODBC or JDBC client

DATA SOURCE SUPPORT

FEDSQL supports a wide range of data sources from multiple vendors as listed below:

FEDSQL Data Sources	FEDSQL Data Sources
Aster	SAP (Read-only)
DB2 for UNIX & PC Operating Environments	SAP HANA
Greenplum	SASHDAT files
Hadoop (Hive and HDMD)	Sybase IQ
Memory Data Store (MDS)	SAS data sets
MySQL	SAS Scalable Performance Data Engine (SPD Engine) data sets
Netezza	Teradata
ODBC databases (such as Microsoft SQL Server)	Oracle

Table 1. FEDSQL Data source support

DATA TYPE SUPPORT

FEDSQL supports a wide range of data types than the previous SAS SQL implementations

FEDSQL Data types	FEDSQL Data types
BIGINT	**SMALLINT**
BINARY	**TIME**
CHAR	**TIMESTAMP**
DATE	**TINYINT**
DOUBLE	**VARBINARY**
DECIMAL|NUMERIC	**VARCHAR**
FLOAT	**NCHAR**
INTEGER	**NVARCHAR**

Table 2. FEDSQL Data Types

PROC SQL VS PROC FEDSQL INTERNALS

PROC FEDSQL queries are executed via implicit pass through mechanism. When SQL queries are submitted using PROC FEDSQL, the queries are converted to the target data source specific code that is passed down to the target database for execution. By using implicit pass-through, FEDSQL can leverage the data source specific capabilities in order to reduce the query response time. Another advantage of implicit pass through is that the target specific data need not be transmitted to FEDSQL side for processing.

To understand PROC FEDSQL's working we compared the execution methods with those of PROC SQL query using the SAS's undocumented _METHOD option. For illustrative purposes we used the same ODBC connection to Microsoft SQL Server and the same query had been submitted using the two procedures.

PROC SQL CODE WITH _METHOD OPTION

```
LIBNAME MSSQL ODBC DSN="MSSQLSERVER";
/*Count Query using PROC SQL and _METHOD option*/
PROC SQL _METHOD;
      SELECT COUNT (*) AS FREQS, COURSEID, GRADE, SEMESTER FROM
      MSSQL.STUDENT_GRADES GROUP BY COURSEID, GRADE, SEMESTER;
QUIT;
```

PROC FEDQL CODE WITH _METHOD OPTION

```
LIBNAME MSSQL ODBC DSN="MSSQLSERVER";
/*Count Query using PROC FEDSQL and _METHOD option */
PROC FEDSQL _METHOD;
      SELECT COUNT (*) AS FREQS, COURSEID, GRADE, SEMESTER FROM
      MSSQL.STUDENT_GRADES GROUP BY COURSEID, GRADE, SEMESTER;
QUIT;
```

While both the above procedures returned the same results, the way both the queries are handled by SAS system are different. The log files of both the procedures are as follows:

```
NOTE: SQL execution methods chosen are:

      sqxslct
          sqxextr( connection to SASIOODB /* dbms=SASIOODB, connect options=( ) */ ( select
COUNT(*) as "FREQS", TXT_1. CourseID , TXT_1. Grade , TXT_1. Semester  from  STUDENT_GRADES
... ) )
16   QUIT;
```

Figure 1. SQL Execution Log

```
21   SELECT COUNT(*) AS FREQS, COURSEID ,GRADE,SEMESTER  FROM MSSQL.STUDENT_GRADES GROUP BY
21 ! COURSEID, GRADE,SEMESTER;

Methods: Full query pushdown!
22   QUIT;
```

Figure 2. FEDSQL Execution Log

The _METHOD is an undocumented SAS option which outputs the way a SQL query is processed by the SAS system. In the case of PROC SQL, the SQL execution methods SQSXSLCT and SQXEXTR are chosen by the SAS system to execute in the SAS environment where as in the FEDSQL method, the complete query is pushed down to Microsoft SQL server for execution.

PROC SQL VS PROC FEDSQL INTERNALS

One of the main advantages of FEDSQL is that it brings in a lot of data types than what is supported by previous SQL flavors. This allows FEDSQL queries to run queries and return results with more precision.

To understand how new data types can increase precision in FEDSQL, we did a little experiment on how FEDSQL works on BIGINT values on a Microsoft SQL Server and compared it with PROC SQL results. We created a table called “t_BigInteger” with a single field “Big_Integer” of type bigint. The table description and the data from Microsoft SQL Server is as follows:

	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	PRECISION	LENGTH
1	osu	dbo	t_BigInteger	Big_Integer	-5	bigint	19	8

Figure 3. Table schema from Microsoft SQL Server

	Big_Integer
1	4611686018427387904
2	2222222222222222222
3	2312346890765434687

Figure 4.Values from t_BigInteger table

PROC SQL vs. PROC FEDSQL RESULTS ON BIGINT VALUES

The following code was run using PROC SQL to find out the sum of all values of Big Integer

```
LIBNAME MSSQL ODBC DSN="MSSQLSERVER";
PROC SQL;
      TITLE 'Display using PROC SQL';
      SELECT * FROM MSSQL.T_BIGINTEGER;
      SELECT SUM (BIG_INTEGER) AS TOTAL_SQL FROM MSSQL.T_BIGINTEGER;
QUIT;
```

Display using PROC SQL

Big_Integer
4611686018427387904
2222222222222222336
2312346890765434880

Display using PROC SQL

TOTAL_SQL
9146255131415045120

Figure 5. SQL results on bigint values display and sum

PROC SQL displayed inaccurate results for the values on the table as well as their sum. Unless the programmer is aware of this situation, it is very hard to spot a limitation of this sort on PROC SQL.

The same ODBC connection was used in PROC FEDSQL and the same set of queries were submitted in a base SAS session.

```
PROC FEDSQL;
      TITLE 'Display using FED SQL';
      SELECT * FROM MSSQL.T_BIGINTEGER;
      SELECT SUM (BIG_INTEGER) AS TOTAL_FEDSQL FROM MSSQL.T_BIGINTEGER;
QUIT;
```

PROC FEDSQL on the other hand returned correct values from the table and correct sum value as the result set.

Display using FED SQL

Big_Integer
4611686018427387904
2222222222222222222
2312346890765434687

Display using FED SQL

TOTAL_FEDSQL
9146255131415044813

Figure 6. FEDSQL results on bigint values display and sum

This clearly demonstrates the power of FEDSQL when it comes to providing precision on larger values.

FEDERATED QUERIES USING FEDSQL

FEDSQL has the ability to communicate with multiple types of data sources at the same time, divide and translate the query with respect to the database it is targeted and perform joins by sending the query to the individual databases. Because the queries are sent down individually to the database, a federated query can leverage the built in abilities of the data source such as threading, query optimizing, parallel processing etc. which are specific to the database.

In this section, we will run a federated query to merge three data sources using FEDSQL. The data sources used are

1. Microsoft SQL Server (Table : Student_Grades)
2. MySQL database (Table : Grades)
3. SAS Dataset (OKSTATE course dataset)

FEDSQL CODE FOR FEDERATD QUERY

```
PROC FEDSQL _METHOD;
          SELECT    S.STUDENTID,
                    S.FIRSTNAME,
                    S.LASTNAME,
                    G.COURSEID,
                    G.SEMESTER,
                    G.GRADE,
                    O.COURSENAME
                    FROM MSSQL.STUDENT_GRADES G,
                              MYSQLDB.STUDENT S,
                           SASGF.OKSTATE O
                      WHERE
                      S.STUDENTID = G.STUDENTID AND O.COURSE_ID = G.COURSEID;
QUIT;
```

The federated query above joins data in three tables from different data sources and returns a single result set as below:

StudentID	FirstName	LastName	CourseID	Semester	Grade	CourseName
S110000	Zabiulla	Mohammed	MKTG 5983	Spr 2015	B	Database Marketing
S110001	Ganesh	Gangarajula	MKTG 5983	Fall 2014	A	Database Marketing
S110002	Pradeep	Kalakota	MKTG 5983	Spr 2015	B	Database Marketing
S110003	Naresh	Abburi	MKTG 5983	Fall 2014	A	Database Marketing
S110004	Bala Murugan	Mohan	MKTG 5983	Spr 2015	B	Database Marketing
S110005	Kushal	Kathed	MKTG 5983	Fall 2014	A	Database Marketing
S110006	Girish	Shirodkar	MKTG 5983	Spr 2015	B	Database Marketing

Figure 7. Federated query output using PROC FEDSQL

```
48       SELECT  S.STUDENTID,
49               S.FIRSTNAME,
50               S.LASTNAME,
51               G.COURSEID,
52               G.SEMESTER,
53               G.GRADE,
54               O.COURSENAME
55               FROM MSSQL.STUDENT_GRADES G,
56                     MYSQLDB.STUDENT S,
57                   SASGF.OKSTATE O
58                 WHERE
59                 S.STUDENTID = G.STUDENTID AND O.COURSE_ID = G.COURSEID;

Methods:
Number of Sorts Performed is : 2
Number of Joins Performed is : 2
Number of Merge Joins Performed is : 1
        MergeJoin (INNER)
          Sort
            NestLoop (INNER)
              SeqScan from SASGF.SASGF.OKSTATE
              SeqScan from MSSQL.dbo.STUDENT_GRADES
          Sort
            SeqScan from MYSQLDB.test.STUDENT
60   QUIT;
```

Figure 8. Federated query log

Using _METHOD option, the log returned above explains the methods FEDSQL chose to run the query. We can see that the number of sorts performed are 2, number of joins performed are 2 and number of merged joins performed is 1. The execution methods chosen will differ once we try to use a table that has indexes.

LEVERAGING DATABASE INDEXES IN FEDERATED QUERIES

FEDSQL can leverage the indexes on the data sources to perform faster joins on federated queries. The methods chosen to perform join will be slightly different than the ones used with the tables that do not have an index. More importantly, FEDSQL supports the “CREATE INDEX” SQL construct to create indexes directly on the data base.

The following code creates two indexes on “StudentID” in the tables present in the Microsoft SQL Server and the MySQL databases.

```
/*creating Indexes using FEDSQL*/
PROC FEDSQL;
      CREATE INDEX STDIDX ON MSSQL.STUDENT_GRADES (STUDENTID);
      CREATE INDEX STDIDX ON MYSQLDB.STUDENT (STUDENTID);
QUIT;
```

```
66       SELECT  S.STUDENTID,
67               S.FIRSTNAME,
68               S.LASTNAME,
69               G.COURSEID,
70               G.SEMESTER,
71               G.GRADE,
72               O.COURSENAME
73               FROM MSSQL.STUDENT_GRADES G,
74                     MYSQLDB.STUDENT S,
75                   SASGF.OKSTATE O
76                 WHERE
77                 S.STUDENTID = G.STUDENTID AND O.COURSE_ID = G.COURSEID;

Methods:
Number of Joins Performed is : 2
Number of Index Performed is : 1
        NestLoop (INNER)
          NestLoop (INNER)
            SeqScan from SASGF.SASGF.OKSTATE
            SeqScan from MSSQL.dbo.STUDENT_GRADES
          IndexScan with _pushed_ qual from MYSQLDB.test.STUDENT
            Index selected: STDIDX
78   QUIT;
```

Figure 9. Federated query execution log using indexes on the database

Once the indexes were created, we submitted the same federated query to examine if the execution steps were different while using an Index. We can clearly see in the log, that the index selected is STDIDX, which we created in the previous step. Since indexes are available on the databases, FEDSQL doesn't do the sort operation explicitly as it relies on the index information on the database. For this reason, it is important to have tuned indexes on the data source to expedite the process of joining using federated queries.

SUPPORT FOR TRANSACTIONS

A transaction is an atomic unit of work. In certain instances, there may be a need to treat a group of updates as a single unit. But, nevertheless, the ability to rollback is needed in case the updates are not successful.

One of the basic requirements while dealing with multiple sources of data bases is the ability to ensure all the related data is consistent at all times, which is a basic requirement of data integrity. Such ability is offered by FEDSQL by supporting COMMIT and ROLLBACK statements.

BEGIN statement marks the beginning of a transaction statement which consists of multiple sub statements. COMMIT statement marks the end of completion of all the statements in the transaction. ROLLBACK statement ensures that no changes are made due to the transaction.

Syntax:

BEGIN [TRANSACTION];

COMMIT [TRANSACTION];

ROLLBACK [TRANSACTION];

FEDSQL offers transaction management through COMMIT and ROLLBACK. But, by default, AUTOCOMMIT is turned ON, which makes each single statement get automatically committed and rolled back if the statement is not successful.

While, transaction management is not offered by many data sources such as SAS, it is offered by data sources that include Aster, DB2 for UNIX and PC Hosts, Greenplum, Microsoft SQL Server, MySQL, ODBC databases, and Sybase IQ.

To utilize the transaction management module, AUTOCOMMIT feature needs to be turned off.

COMMAND:

LIBNAME LIBREF SASLIBRARY AUTOCOMMIT= NO;

CONCLUSION

FEDSQL is a new and proprietary SQL implementation of SAS, available in SAS 9.4 FEDSQL is not a replacement of PROC SQL, but it has many powerful features such as precision via new datatypes, vendor independency etc. FEDSQL translates queries to vendor specific database native language which reduces the amount of work done by SAS system by leveraging database specific capabilities. With DS2 becoming a powerful language in handling data in SAS and FEDSQL's ability to merge into DS2 programs, this combination has the potential to make its mark in data processing programming alongside DATA step and PROC SQL.

REFERENCES

SAS® 9.4 Language Reference, Fourth Edition, Available at http://support.sas.com/documentation/cdl/en/lrcon/67885/PDF/default/lrcon.pdf

SAS® 9.4 FEDSQL Language Reference, Third Edition. Available at http://support.sas.com/documentation/cdl/en/fedsqlref/67364/PDF/default/fedsqlref.pdf

Lafler, Kirk Paul. 2013. "Exploring the PROC SQL _METHOD Option." *Proceedings of the SAS Global 2013 Conference.* Available at http://support.sas.com/resources/papers/proceedings13/200-2013.pdf

SAS PROC FEDSQL Documentation. Available at http://support.sas.com/documentation/cdl/en/proc/67327/HTML/default/viewer.htm#n06w5kqwkurgk2n1irjo6h94fkcq.htm

ACKNOWLEDGMENTS

We thank Dr. Goutam Chakraborty, Ralph A. and Peggy A. Brenneman Professor of Marketing & Founder of SAS and OSU Data Mining Certificate Program, Oklahoma State University, for his support, guidance and encouragement throughout our research.

AUTHORS

Zabiulla Mohammed is a Masters' student in Management Information Systems at Spears School of Business, Oklahoma State University. He holds SAS Statistical Business Analyst, Predictive Modeler, Base and Advanced Programmer for SAS 9 Credentials. He has 5 years of experience working with two Fortune 100 companies. He successfully completed SAS and OSU Data Mining Certificate program in December 2014. He has an undergraduate degree in Computer Science and Engineering and has presented various papers at conferences including JMP Discovery Summit 2014 and SAS Analytics Conference 2014

Ganesh Kumar Gangarajula, is a Masters' student in Management Information Systems at Spears School of Business, Oklahoma State University. He has an undergraduate degree in Computer Science & Engineering. He has a previous work experience of 2.5 years as a Solution Developer for a leading US Automobile Insurance client. He had successfully completed the SAS & OSU Data Mining Program in the summer of 2014. He is also a Base SAS® 9 Certified Programmer, SAS® Certified Statistical Business Analyst. He has also published papers at other conferences which include JMP Discovery Summit 2014, SAS Analytics Conference 2014 and SAS Analytics Day at OSU 2014.

Pradeep Reddy Kalakota was a Graduate Student at Oklahoma State University and completed OSU and SAS data mining certification program. He is currently working as Business Intelligence Developer II for Divisional Risk team at FHL Bank Des Moines. He is a Base SAS® 9 certified Programmer, SAS® Certified Statistical Business Analyst, SAS® certified Predictive Modeler using Enterprise Miner 7. He has 5 years of experience in ETL-Data Warehousing at Deloitte Consulting.

CONTACT INFORMATION

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

Zabiulla Mohammed
Oklahoma State University
zabiulla.mohammed@okstate.edu
https://www.linkedin.com/in/zabiullamohammed

Ganesh Gangarajula
Oklahoma State University
ganesh.gangarajula@okstate.edu
https://www.linkedin.com/in/gangarajulaganesh

Pradeep Reddy Kalakota
Federal Home Loan Bank of Des Moines
pradeep.kalakota@okstate.edu
https://www.linkedin.com/in/pradeepreddykalakota