When was my data last updated? Automating data monitoring and notification

Drew Turner, Texas Parks and Wildlife Department, Austin, TX

Abstract

Knowing when data was last updated can prove invaluable when verifying data integrity. If you have ever spent time troubleshooting a process only to find out that your root problem was outdated data, then you know the importance of keeping track of how recent your data is. Having that information about your data—how much there is and how recently it was updated—awaiting you first thing in the morning increases efficiency, and the ability to analyze and compare past results enables you to recognize trends and better predict future activity. This paper will describe a method to identify when a table was last updated (whether stored in a separate database or the SAS server), how to store that information for later analysis and finally how to automatically notify you of the results.

Introduction

At my organization, knowing when new data was last added to a table can save hours of time running reports only to find out our output isn't what we were expecting because data was missing, and then we must re-run those reports. It was with this in mind that we set about to determine how to keep our data up-to-date and notify users when data was last added to mission critical tables. Finding when a table was last updated can be accomplished several ways, but to ensure reliability and provide additional customized information, we developed the methodology explained in this paper. Detailed below are two methods to retrieve relevant data, how to process and store that data, how to get notification of the day's results, and finally how to schedule the project.

Get your data

The first step is to get the most recent data through the first method, which utilizes the PROC Contents procedure. The PROC Contents function provides descriptive information on a table located on the SAS server, in this case a table created from a scheduled Enterprise Guide project that runs daily. The Proc Contents procedure has many options; however, this method only utilizes the DATA, OUT, DETAIL and PRINT options.

PROC Contents Method

The first option is the DATA option, which defines what data will be pulled. For this paper we will be using the table BISPRO04.GL_OUTSTANDING_ENCUMBERANCES. Next, specify the level of detail needed, which (since this example only necessiates general table attributes) is set to NODETAILS, the system default. The OUT option specifies to which table to output descriptive information. Finally the PRINT option is set to NOPRINT, which disables printing the results within the project. It may be useful to exclude this option during development in order to view all the available information.

```
PROC CONTENTS DATA=BISPRO04.GL_OUTSTANDING_ENCUMBERANCES NODETAILS
OUT=WORK.GL_SUMMARY_CHECK_INPUT NOPRINT;
run;
```

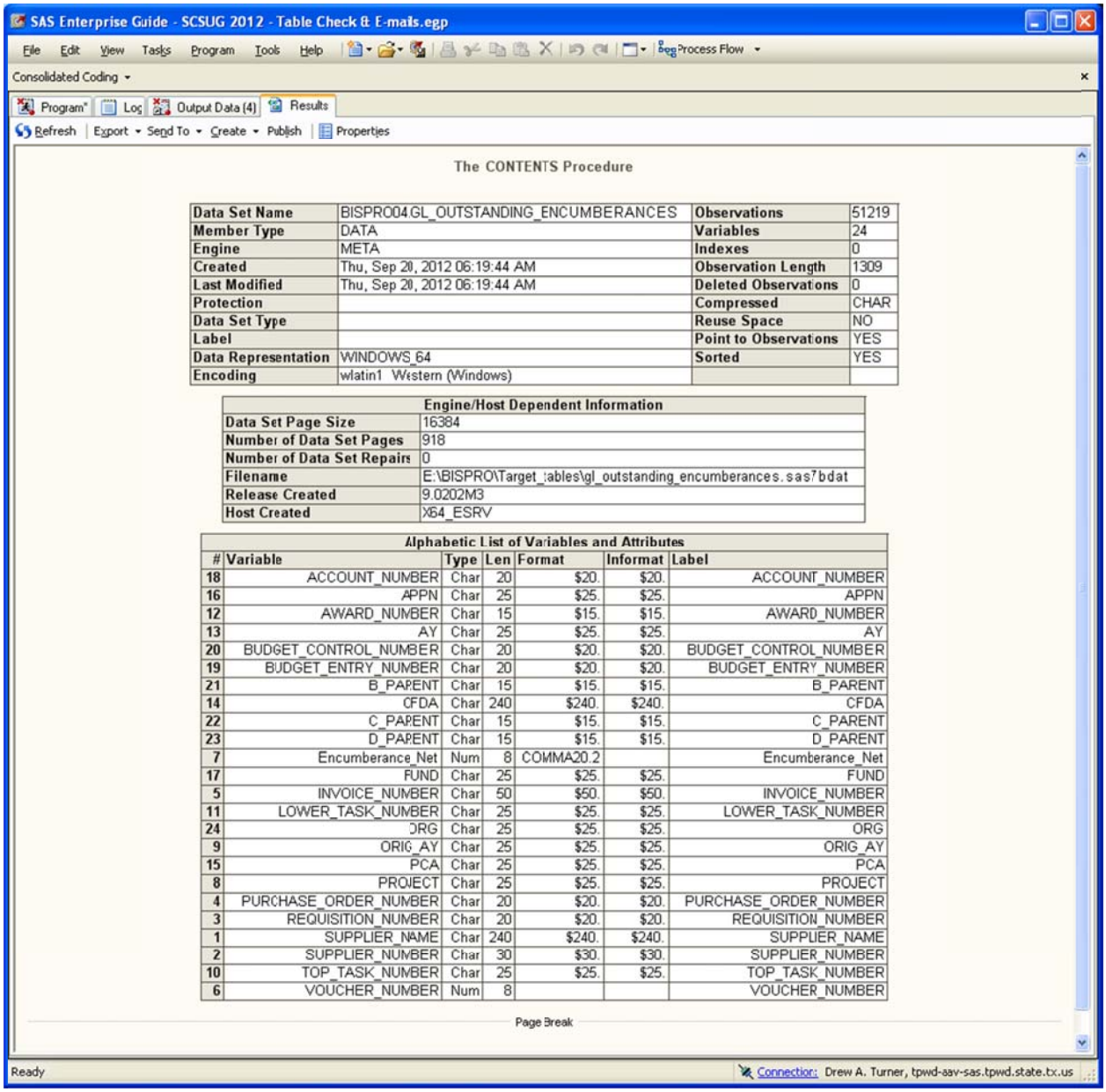

Figure 1. Results of the PROC Contents on BISPRO04.GL_OUTSTANDING_ENCUMBERANCES.

While PROC Contents can provide information on SAS Tables, another method is needed to retrieve similar data on non-SAS tables.

Pass-thru SQL Method

Utilizing pass-thru SQL allows the use of database specific coding, in this case the SCN_TO_TIMESTAMP and ORA_ROWSCN functions, which can provide information not available about a table from the SAS server. The table of interest, TPW.TPWD_USAS_HX_EXTRACT, is updated daily in an Oracle database. One alternative to this method is to query the ALL_TABLES system table to retrieve this information, but this approach only works if statistics are being run on the table in question. Statistics are not being run on the TPW.TPWD_USAS_HX_EXTRACT table, hence the need for this method.

The pass-thru function of PROC SQL can fill entire papers on its usage and options, but this paper only covers the main elements, which consist of the CONNECT statement, the CONNECTION TO statement and the DISCONNECT statement. The CONNECT statement specifies what type of database to connect to and provides the user's credentials as shown below.

```
proc sql;
   CONNECT TO oracle (user=query password=access path=BISPRO);
```

Once connected to the Oracle database, run the database specific code and retrieve the output by selecting 'CONNECTION TO oracle' as the query table. The ORA_ROWSCN function will return the last system change number (SCN) for a row while the SCN_TO_TIMESTAMP function converts this SCN to a timestamp.

```
CREATE TABLE WORK.HX_UPDATE_TABLE as
     SELECT * from CONNECTION TO oracle
          (SELECT SCN_TO_TIMESTAMP(MAX(ORA_ROWSCN)) as Update_Time,
                  count(*) as row_count
          FROM TPW.TPWD_USAS_HX_EXTRACT);
```

In this case, it is only necessary to know when the table was last updated and the number of rows in the table. Once the data is retrieved, the DISCONNECT statement closes the connection to the Oracle server.

```
DISCONNECT from oracle;
```

Create a Stage Table

The data has now been retrieved and will be formatted for later reporting but still needs the application of additional logic before storage. This paper describes how to store information about multiple database tables on one SAS table, so the Table_name value is used to identify the database table in the description. Putting the row count into the comma format makes it more readable when reporting on this data down the road, as does separating the last update timestamp into a date portion and a time portion and storing them separately. Next, determine at a glance if the table was up to date when this ran. To do this, add a case statement to determine if today's date matches the last update date, and store a character string indicating the result. And finally, in order to record exactly when these records were added to the dataset, the current date and time are recorded with the DATETIME function.

```
CREATE TABLE WORK.USAS_HX_CHECK AS
   SELECT DISTINCT
          'TPW.TPWD_USAS_HX_EXTRACT' as Table_name,
          t1.row_count FORMAT =comma20. as NOBS,
          DATEPART(t1.Update_Time) FORMAT =mmddyy10. as CRDATE,
          TIMEPART(t1.Update_Time) FORMAT =TIMEAMPM. as TIME,
          CASE
          WHEN DATEPART(t1.Update_Time) = TODAY()
          THEN 'TABLE UP TO DATE'
          ELSE 'TABLE OUT OF DATE!!!!!!!!'
          END AS DATE_CHECK,
          datetime() FORMAT =dateampm. as Run_Datetime
   FROM WORK.HX_UPDATE_TABLE t1;
```

Write Data to a SAS Table

Now that the data is formatted and processed the next step is to combine the temporary tables to simplify the creation of a final table. Append the GL_SUMMARY_CHECK table to the USAS_HX_CHECK table and create the table Append_Table.

```
CREATE TABLE WORK.APPEND_TABLE AS
    SELECT * FROM WORK.GL_SUMMARY_CHECK
       OUTER UNION CORR
    SELECT * FROM WORK.USAS_HX_CHECK
```

The first time the project runs, create a new table to store the data.

```
CREATE TABLE BISPRO04.TABLE_CHECK_SCSUG12 AS
   SELECT t1.CRDATE,
          t1.DATE_CHECK,
          t1.NOBS,
          t1.Run_Datetime,
          t1.Table_name,
          t1.TIME
   FROM WORK.APPEND_TABLE t1
```

After running this project the first time, change how new data is added to the TABLE_CHECK_SCSUG12 table to insert new rows into the existing table instead of appending the new data to the existing table and then rewriting the table. Use the INSERT INTO statement to avoid data integrity issues reading from and writing to the same table, as well as to automatically update any indexes on the TABLE_CHECK_SCSUG12 table.

```
Insert into BISPRO04.TABLE_CHECK_SCSUG12
   SELECT t1.CRDATE,
          t1.DATE_CHECK,
          t1.NOBS,
          t1.Run_Datetime,
          t1.Table_name,
          t1.TIME
   FROM WORK.APPEND_TABLE t1
```

Now that the new data has been added to the final table, send out an e-mail with the current day's data. The first step is to load data into macro variables.

Load Data into Macros

For each variable that the user wants to receive in a daily e-mail, load it into a macro variable. To do this use the CALL SYMPUT routine that creates a macro variable for each variable, and sets it to the specified value. Each variable is placed within the STRIP function to remove any leading or trailing spaces as shown below. The Number of observations (NOBS) value is converted to a character data type and put in the comma format to ensure proper formatting when displayed later in an e-mail.

```
DATA GL_SUMMARY;
SET GL_SUMMARY_CHECK;
CALL SYMPUT('RECORD_COUNT_ENCUM',STRIP(PUT(NOBS,COMMA20.)));
CALL SYMPUT('CREATE_DATE_ENCUM',STRIP(CRDATE));
CALL SYMPUT('TIME_ENCUM',STRIP(TIME));
CALL SYMPUT('DATE_CHECK_ENCUM', STRIP(DATE_CHECK));
run;

DATA USAS_CHECK;
SET USAS_HX_CHECK;
CALL SYMPUT('RECORD_COUNT_HX',STRIP(PUT(NOBS,COMMA20.)));
CALL SYMPUT('CREATE_DATE_HX',STRIP(CRDATE));
CALL SYMPUT('TIME_HX',STRIP(TIME));
CALL SYMPUT('DATE_CHECK_HX', STRIP(DATE_CHECK));
run;
```

Note that each table has a unique macro name for each column, which for the number of tables involved here isn't an issue, but may be an issue when using this method with a larger number of tables.

Notify Me

Now that the data is stored where it can be inserted dynamically, build the e-mail message. The code below inserts the newly created macros into an e-mail. Note the option of specifying the recipient, the REPLY TO and the CC addresses. Macro names must be preceded with the & sign as well as enclosed in quotes, as shown below. Apostrophes will display exactly what is entered, which in this case would be the name of the macro.

```
FILENAME OUTBOX EMAIL
TO="Drew.Turner@tpwd.state.tx.us"
FROM="Drew.Turner@tpwd.state.tx.us"
CC="Drew.Turner@tpwd.state.tx.us"
SUBJECT="SCSUG 2012 Daily Table Check E-mail";

data _null_;
FILE OUTBOX;
PUT "The SAS GL Outstanding Encumbrance Table has completed.";
PUT "There were &RECORD_COUNT_ENCUM records.";
PUT "Table Created on &CREATE_DATE_ENCUM at &TIME_ENCUM.";
PUT " ";
PUT "&DATE_CHECK_ENCUM";
PUT " ";
PUT "The USAS HX File has been updated in BISPRO.";
PUT "There were &RECORD_COUNT_HX records.";
PUT "Table updated on &CREATE_DATE_HX at &TIME_HX.";
PUT " ";
PUT "&DATE_CHECK_HX";

RUN;
```

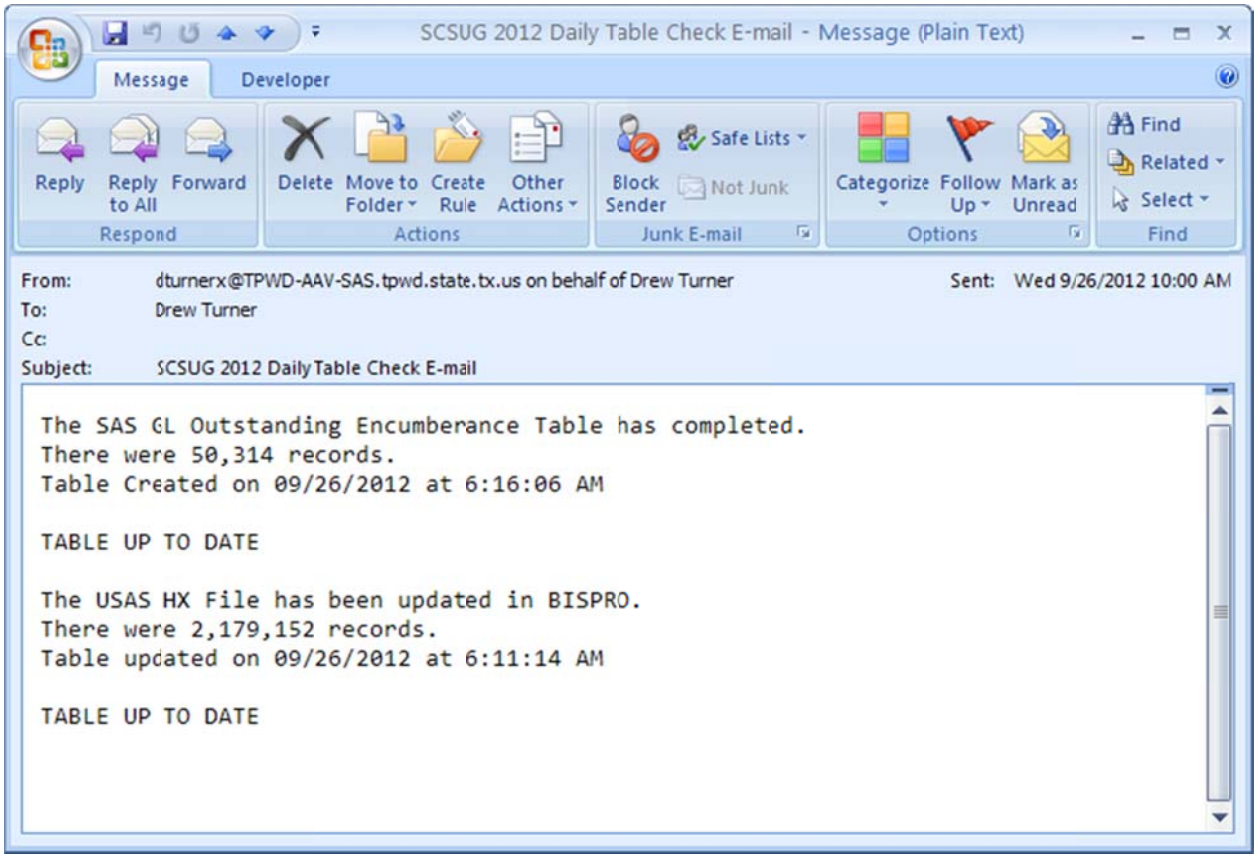

Figure 2. Example E-mail notification of table row counts and run times.

Now Automate It!

Now the e-mail has a row count and last update time for the selected tables, so the next step is to automate sending the data at a specific time. To schedule this, utilize Enterprise Guide's scheduler, which is integrated with the Windows system scheduler and is found on the file menu.

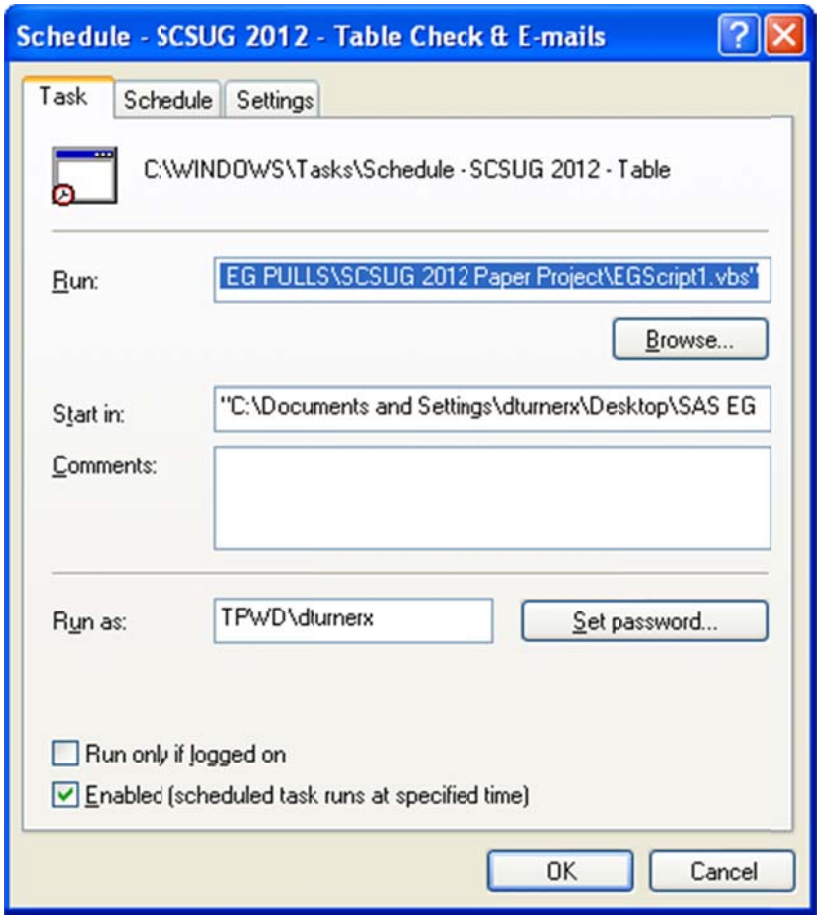

Figure 3. Runtime and user options available for scheduling a EG project.

When utilizing a network logon the user will need to enter his or her network password for the project to log into their local machine to run. With that set, move to the schedule tab and specify at what time and frequency to run the project. One thing to note with this approach is that using the Windows task scheduler on a local machine necessitates leaving it on overnight (or turning it on prior to run time) for this project to run.

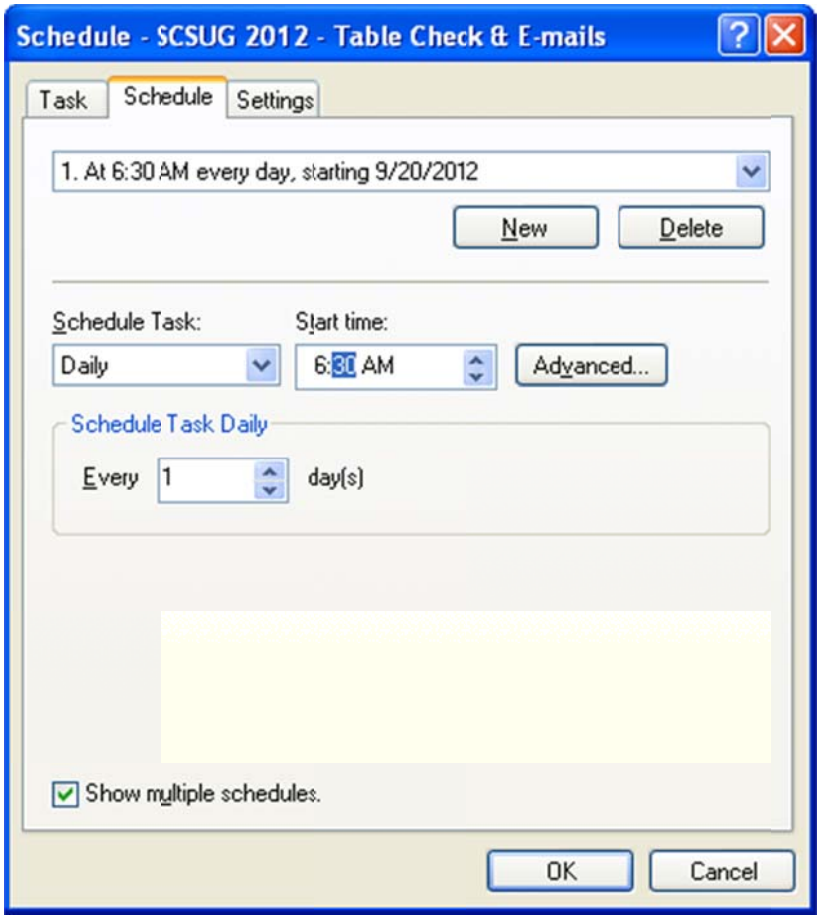

Figure 4. Scheduling options available for an EG project.

Under the settings tab is the option to set the maximum run time for the project, which defaults to 72 hours.

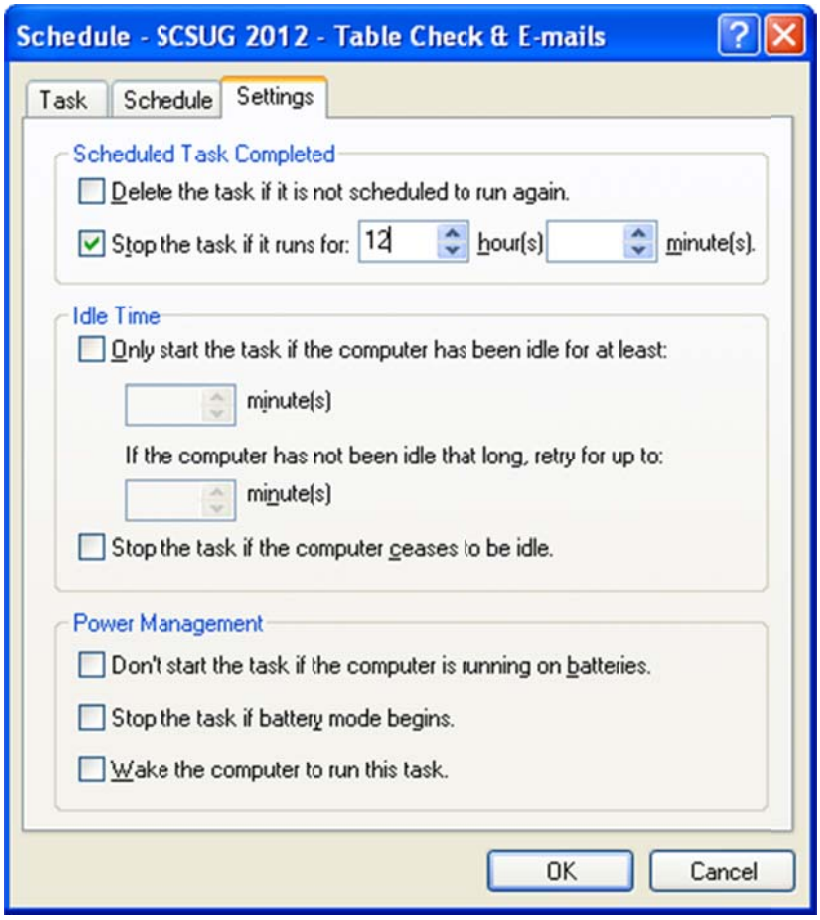

Figure 5. Example of runtime option on scheduled EG project.

Conclusion

The examples covered here demonstrate how a SAS programmer can find when new data was last added to a table using two separate methods. The data was prepared and stored for later analysis, and the user can notify him or herself of the daily results. And finally the entire process was automated. These steps should enable users to be informed of the status of their data and maybe even let their DBA know when their database is having an issue.

References

Base SAS 9.2 Procedures Guide. Available at
http://support.sas.com/documentation/cdl/en/proc/61895/PDF/default/proc.pdf

Hummel, Andy, 2012 "Sending E-mails in your sleep" Proceedings of the SAS Global Forum 2012 Conference. Available at http://support.sas.com/resources/papers/proceedings12/078-2012.pdf

Oracle database documentation on the Ora_rowscn pseudocolumn. Available at
http://docs.oracle.com/cd/B19306_01/server.102/b14200/pseudocolumns007.htm

Oracle database documentation on the scn_to_timestamp function. Available at
http://docs.oracle.com/cd/B19306_01/server.102/b14200/functions142.htm

Poling, Jeremy W., 2011, "Finding Oracle® Table Metadata: When PROC CONTENTS Is Not Enough" Proceedings of the South Central SAS Users Group. Available at
http://www.scsug.org/SCSUGProceedings/2011/poling2/FINDING%20ORACLE%20TABLE%20METADATA%20WITH%20SAS.pdf

Contact Information

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

Drew Turner
Texas Parks & Wildlife Department
Financial Analyst
Drew.Turner@tpwd.state.tx.us
512-389-8246