SQL Logger Documentation

C. Grant Anderson

5/10/2018

## Document Summary:

This document contains the formal documentation for the SQL Logger log stored procedure and its use for instrumenting ETL procs. This implementation of SQL Logger is based upon previous ETL implementations that I have done at previous companies on other cube based ETL system.

Contents

[Document Summary: 1](#_Toc513727575)

[1.0 The Need…And the Advantages 3](#_Toc513727576)

[1.1 Cube ETL Logging 4](#_Toc513727577)

[2.0 Overview 5](#_Toc513727578)

[2.1 The SQL\_Logger\_Run\_Number Sequence 5](#_Toc513727579)

[2.2 The SQL\_Logger Stored Procedure 6](#_Toc513727580)

[2.2.1 Parameters 8](#_Toc513727581)

[2.2.2 Insert 10](#_Toc513727582)

[2.2.3 Select – Manual Entries 10](#_Toc513727583)

[2.2.4 Select – Automatic Entries 10](#_Toc513727584)

[2.3 The SQL\_Logger\_Log Table 10](#_Toc513727585)

[2.3.1 Primary Key Columns 11](#_Toc513727586)

[2.3.2 Step Columns 12](#_Toc513727587)

[2.3.3 Primary Info Columns 12](#_Toc513727588)

[2.3.4 Date and Time Columns 12](#_Toc513727589)

[2.3.5 Log Messages 12](#_Toc513727590)

[2.3.6 Automatically Logged Info Columns 12](#_Toc513727591)

[2.4 The SQL\_Logger Reporting Views 13](#_Toc513727592)

[3.0 Instrumenting Procs and SQL 14](#_Toc513727593)

[3.4 Simple Asynchronous Logging 15](#_Toc513727594)

[3.5 Synchronous Run Number Logging 15](#_Toc513727595)

[3.6 Simple Example – Simple Asynchronous Logging 16](#_Toc513727596)

[3.7 Simple Example – Synchronous Run Number Logging 16](#_Toc513727597)

[4.0 Instrumenting Details with Code 20](#_Toc513727598)

[4.1 Starting Logging Code Block: 20](#_Toc513727599)

[4.2 Steps 1 to N Code Blocks: 21](#_Toc513727600)

[4.3 End Logging Code Block: 22](#_Toc513727601)

[5.0 Reading the Log 23](#_Toc513727602)

[5.1 Direct Reading of the Log File 23](#_Toc513727603)

[5.2 Using the SQL Logger Views 24](#_Toc513727604)

[5.2.1 SQL Logger Summary View 24](#_Toc513727605)

[5.2.2 SQL Logger Info View 24](#_Toc513727606)

[5.2.3 SQL Logger Steps View 24](#_Toc513727607)

[5.2.4 SQL Logger Error View 25](#_Toc513727608)

[5.3 Using a Log Reading Program Tool 25](#_Toc513727609)

[6.0 Part of the Larger Infrastructure 26](#_Toc513727610)

[6.1 Log Viewer 27](#_Toc513727611)

[6.1 Log Exceptions Alerter 28](#_Toc513727612)

[6.1 Web Log Dashboard 28](#_Toc513727613)

## 1.0 The Need…And the Advantages

Why have log files?

Simply put…It’s because you need to.

You need to:

1. Be able to see what happened and is happening.
2. Have a record of process errors.
3. And…You really need a record of the processes, when they ran, how they did, and how long they took.

Without this ability you are going to be massively hobbled in determining:

1. How well your system is running,
2. What errors and problems are encountered in your system, and
3. How well your system is performing.

Unfortunately, the vast majority of ETL processes for OLAP cubes are not logged or monitored.

This results in a “Hope and Pray” type of existence where problems when they occur…And they will occur…Are not proactively detected through active monitoring but a left for someone to notice and then alert the people responsible for supporting the system.

***“Hey, the numbers in the XYZ cube aren’t right!”***

***“Ah…Which numbers, specifically?”***

***“ALL of the numbers for the past eight days of off! We just noticed it….”***

Leaving problems until they are brought to your attention is wasteful, inefficient, and really bad practice. OLAP cubes are especially susceptible to this. Cube data and reports that are presented to senior management MUST be up to date and accurate. If the ETL processes are not running at 100% data errors and problems WILL slip into the cube most often with nobody being aware of them…Until something embarrassing or even disastrous occurs!

I like to have active monitoring in place on all of my cube projects. And it starts with cube ETL logging.

## 1.1 Cube ETL Logging

What is cube ETL logging?

Cube ETL logging is simply putting logging statements into the cube ETL so that what happens is recorded in a log file.

It’s simple and easy to do yet few cube installations…And other SQL processes and system…Ever use them.

This document describes how to easily do cube ETL logging with a simple single SQL stored procedure (proc). And this logging can be easily applied to SQL processes and systems beyond cube ETL.

An example of logging for the MCR Journal Write Cube from its documentation:

**SQL Server – ReportData Data Warehouse**

Log Viewer/Monitor

Logging Table

1. ETLLog\_Summary

Future

Source Tables

Cube

Staging Tables

ETL Proc

1. pLoadStageMCRJournalEntryPhoneCalls
2. ReportData.stage.MCRJCSJournalEntry
3. crm\_server.CERRSNGPROD\_MSCRM.dbo.PhoneCall
4. crm\_server.CERRSNGPROD\_MSCRM.dbo.cog\_Case
5. ERR\_1095A.dbo.HICSCases

## 2.0 Overview

The SQL Logger logging system consists of:

1. The SQL\_Logger\_Run\_Number Sequence.
2. The SQL\_Logger Stored Procedure.
3. The SQL\_Logger\_Log Table.
4. The SQL\_Logger Reporting Views:
   1. SQL\_Logger\_Summary
   2. SQL\_Logger\_Steps
   3. SQL\_Logger\_Info
   4. SQL\_Logger\_Error

SQL\_Logger Stored Procedure

SQL\_Logger\_Log Table

SQL\_Logger Views

### 2.1 The SQL\_Logger\_Run\_Number Sequence

A run is an innovation/execution of a particular SQL script, stored procedure, or other object that has been instrumented with SQL Logger.

A run is represented by a Run\_Number which is a unique SQL Server sequence number from the SQL\_Logger\_Run\_Number sequence. This number groups connected and sequential log entries together.

In practice, the SQL Logger Run Number provides a unique integer number that can be used to represent groupings of log entries for a particular SQL object such as a stored procedure (proc) or even an entire ETL process. This is very important in using and analyzing logs as it makes seeing which log entries apply to a proc or ETL process fast, easy, and convenient. The alternative is to try to group related log entries from a large number of disparate log entries into a coherent grouping. Which is no easy task when many processes are writing many different log file entries for each activation and step of their SQL operations.

The use of SQL Server sequences makes creating and using a Run Number easy.

Create Sequence SQL:

CREATE SEQUENCE [dbo].[SQL\_Logger\_Run\_Number]

AS [bigint]

START WITH 1

INCREMENT BY 1

MINVALUE 1

MAXVALUE 9223372036854775807

CYCLE

CACHE

GO

A SQL Server sequence is used by simply selecting its next value:

SELECT NEXT VALUE FOR SQL\_Logger\_Run\_Number;

It will always return the next value up to the maximum value set for the sequence. Recycling the sequence back to zero when it reaches its maximum value of 9,223,372,036,854,775,807 is automatic if the Recycle option is used.

The previous approach that I used before SQL Server sequences were available was simply to create a single Run Number tables with a column for the current Run\_Number and then use a stored procedure to lock it with a table lock and simply get the value, increment it by one, and then update the table. It then passes on the “next value” to the calling process/program and automatically puts the next value back into the table ready for the next use of it. The table lock prevents all other processes from accessing it during the very short period of time that the calling proc accesses it. Works just like a sequence.

### 2.2 The SQL\_Logger Stored Procedure

The SQL Logger stored procedure a single proc that both manually and automatically logs information to the Log table.

It is fairly straightforward in implementation.

--===========================================================

-- SQL\_Logger Proc.sql

-- Grant Anderson

-- 5/7/2018

--===========================================================

USE [ReportData]

GO

SET ANSI\_NULLS OFF

GO

SET QUOTED\_IDENTIFIER OFF

GO

CREATE PROCEDURE dbo.SQL\_Logger

@RunNumber int = 0,

@LogLevel varchar(250),

@Title varchar(250) = NULL,

@StepNumber int = 0,

@StepName varchar(250) = NULL,

@StartTime DateTime = NULL,

@StopTime DateTime = NULL,

@ElapsedTimeSeconds Decimal(12,3) = NULL,

@Message varchar(200) = NULL

AS

IF @StartTime IS NULL

Set @StartTime = SYSDATETIME();

INSERT INTO dbo.SQL\_Logger\_Log

(

Run\_Number,

Step\_Number,

Step\_Name,

Log\_Level,

Log\_Title,

Log\_DateTime,

Start\_Time,

Stop\_Time,

Elapsed\_Time\_Seconds,

Log\_Message,

Row\_Count,

[App\_Name],

[Error\_Number],

SPID,

Row\_Count\_Big,

Transaction\_Count,

Procedure\_ID,

Server\_Name,

[Database\_Name],

[Schema\_Name],

[User\_Name],

System\_User\_Name,

Last\_Identity\_Value,

Catch\_Error\_Number,

Catch\_Error\_Severity,

Catch\_Error\_State,

Catch\_Error\_Procedure,

Catch\_Error\_Line,

Catch\_Error\_Message

)

SELECT

@RunNumber, -- Run Number.

@StepNumber, -- Run\_Step Number.

@StepName, -- Run\_Step Name.

@LogLevel, -- Log\_Level.

@Title, -- Log\_Title.

SYSDATETIME(), -- Log\_DateTime.

@StartTime, -- Start\_Time.

@StopTime, -- Stop\_Time.

@ElapsedTimeSeconds, -- Elapsed\_Time\_Seconds.

@Message, -- Log\_Message.

-- Main Detail Data:

@@ROWCOUNT, -- Row\_Count.

App\_Name(), -- App\_Name.

@@ERROR, -- Error\_Number.

-- Transaction Data:

@@SPID, -- SPID.

ROWCOUNT\_BIG(), -- Row\_Count2.

@@TRANCOUNT, -- Transaction\_Count.

@@PROCID, -- Procedure\_ID.

-- Server and Such Data:

HOST\_NAME(), -- Server\_Name.

DB\_NAME(), -- Database\_Name.

SCHEMA\_NAME(), -- App\_Schema.

USER, -- Database\_User.

SYSTEM\_USER, -- App\_User.

SCOPE\_IDENTITY(), -- Last\_Identity\_Value.

-- Try Catch Block Data:

ISNULL(ERROR\_Number(), 0), -- Catch\_Error\_Number.

ISNULL(ERROR\_SEVERITY(), 0), -- Catch\_Error\_Severity.

ISNULL(ERROR\_STATE(), 0), -- Catch\_Error\_State.

ISNULL(Error\_Procedure(), 'No Procedure'), -- Catch\_Error\_Procedure.

ISNULL(ERROR\_LINE(), 0), -- Catch\_Error\_Line.

ISNULL(ERROR\_MESSAGE(), 'No Message') -- Catch\_Error\_Message

;

GO

### 2.2.1 Parameters

The main idea is to have a fairly low number of parameters that have to be populated and use defaults for them so that they are optional.

The parameters are:

1. @RunNumber – The Run Number for this group of log messages. 0 = No Run Number.
2. @LogLevel – Info, Warning, Error, Debug, or other level of the log message/record.
3. @Title – A short description for the log message, most often the name of the proc.
4. @StepNumber – An integer sequential number to denote the step of the logging and operation.
5. @StepName – A string (varchar) label for the step. Allows hierarchal tree numbering such as 1.1, 1.2, 1.3, 2.0, etc that cannot be handled with a simple integer column.
6. @StartTime – The start time, preferably SYSDATETIME() for higher accuracy.
7. @StopTime - The stop time, preferably SYSDATETIME() for higher accuracy.
8. @ElapsedTimeSeconds – The elapsed time in seconds from Start Time to Stop Time.
9. @Message – A programmer supplied message indicating where the log message comes from within the SQL object or proc and what was or should be being done at that area or an error message.

These parameters are default so that the developer/programmer does not have to populate them always:

1. @RunNumber – Defaults to 0, which is essentially a generic, no specific Run Number.
2. @Title – Defaults to null as some programmers will not, but should, use it with @LogMessage.
3. @StepNumber – Defaults to 0.
4. @StepName – Defaults to null if not specified or used.
5. @StartTime – Defaults to null. Note that the default automatic LogDateTime column date time value can always be assumed to be start time for any specific log entry but not for sub-groupings of course.
6. @StopTime – Defaults to null. This will be null when one is just logging a Start Time.
7. @ElapsedTimeSeconds – Defaults to null as not always used.
8. @Message – Defaults to null if not used.

So the only required parameters are:

1. @LogLevel

Although either @Title or @Message should of course be used with it for minimal logging.

Note that a enumerated value is not enforced for @LogLevel because there is no way to allow for easy semi-automatic selection of what is a valid Log Level in SQL Server. Sure, a foreign key and lookup table can be used to enforce relational integrity but if/when a develop does not test the logging then they may not have a valid Log Level in their code and not know it. Is it “Info” or “INFO” or “INF”? What if a person wants to put something special in this parameter?

All good questions. So as a default there is no relational integrity enforcement on this parameter. There probably should be for these values:

INFO

WARNING

ERROR

MINOR ERROR

MAJOR ERROR

CRITICAL ERROR

DEBUG

TEST

Once you determine what should be allowable then it is a simple matter to create a lookup table and a foreign key for relational integrity on the Log Level column. Then each develop will need to adhere to these standard Log Levels. Additional information can always be put into the Message parameter and column.

### 2.2.2 Insert

The insert section of the proc covers the columns of the SQL Logger Log table.

Note that Log\_ID is an identity column and does not need to be inserted.

### 2.2.3 Select – Manual Entries

All the columns associated with the proc parameters are in this category.

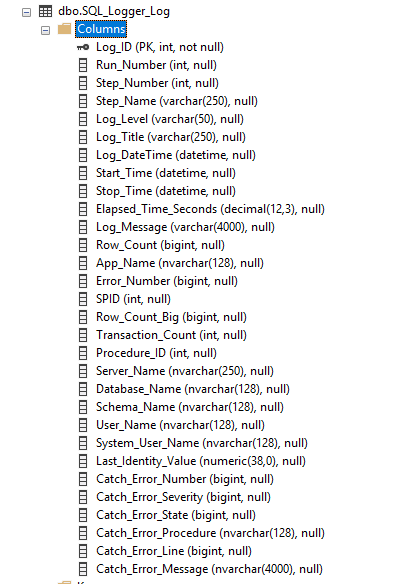
### 2.2.4 Select – Automatic Entries

And these entries are automatically determined from SQL Server global variables and functions.

Note that the Catch\_\* entries are only populated when used in a TRY CATCH block and an error occurs (except for the automatic default values).

### 2.3 The SQL\_Logger\_Log Table

The SQL Logger Log table is a rather large table that holds quite a bit of log information.



Automatically Logged Info Columns

Log Message

Date and Time Columns

Primary Info Columns

Step Columns

Primary Key Columns

### 2.3.1 Primary Key Columns

The Log\_ID column is a unique key for each log entry.

The Run\_Number column is used to group related log entries together in a virtual group.

### 2.3.2 Step Columns

The two Step Columns allow for a SQL process or proc to be organized by putting them into a series of steps and allow for sequential integer numbering and flexible naming including the ability to number using hierarchal tree numbering, i.e. 1.0, 1.2, 1.3, 2.0, 2.1, 2.1.1, etc.

### 2.3.3 Primary Info Columns

The Primary Info columns include:

1. Level
2. Title
3. DateTime

Which are the info columns that will be used virtually all of the time.

### 2.3.4 Date and Time Columns

The Date and Time columns are used to log:

1. Start Time (and date)
2. Stop Time (and date)
3. Elapsed Time in Seconds

These are all optional info parameters.

Note that ElapsedTimeSeconds is automatically calculated in the Summary log reporting view if Start Time and Stop Time are populated but Elapsed Time Seconds is null.

### 2.3.5 Log Messages

This is a programmer created message that explains the step or the error.

### 2.3.6 Automatically Logged Info Columns

The data for these columns are automatically logged in the log table by the SQL Logger proc. Some of these data columns are not regularly populated, i.e. the columns starting with the prefix “Catch\_\*” are columns that are only populated with error information when logging occurs with a TRY CATCH SQL block and an error occurs.

When this happens, this information is quite useful for debugging purposes.

### 2.4 The SQL\_Logger Reporting Views

Views have been created for use with the SQL Logger Log table so that the information in it is presented in a more comprehensible way.

The reporting views are:

1. SQL\_Logger\_Summary View – Provides a high-level summary.
2. SQL\_Logger\_Steps – Provides details of the steps.
3. SQL\_Logger\_Info – Provides information on the environment of the log entries such as server and database and other information.
4. SQL\_Logger\_Error – Provides error information.

All of these views can be filtered by Run\_Number, Log\_DateTime, and other columns so as to narrow down the display data to specific criteria.

Note that the SQL\_Logger\_Summary view automatically calculates the stop time and the elapsed time of the particular run and displays a nice high-level summary of the run status. The other 3 views provide specific information slices from the SQL\_Logger\_Log table.

## 3.0 Instrumenting Procs and SQL

Instrumenting procs, and SQL, simply means inserting SQL logging statements into the procs and SQL to accomplish logging. This is called “instrumenting” as this was done back in the old days for hardware development and debugging.

#### 3.1 Two ways to use logging

Two ways to use logging:

1. Simple Asynchronous Logging, and
2. Synchronous Run Number Logging.

Simple asynchronous logging is minimal logging where a person logs just the minimum information and most often does not provide (or loses) a lot of useful contextual information as a result. Minimal logging is useful in situations where there are not many processes, users, or things to log. Logging can be set up on an individual database basis. And it also can be customized for specific processes or scenarios. So these are primary candidates for this method of logging since there would not necessarily be a large number of processes generating log messages.

Synchronous Run Number Logging is detailed logging appropriate for large departmental and enterprise SQL systems. It is here where Run Numbers and Steps and other SQL Logger functionalities are most important as well as developer discipline in using them properly.

#### 3.2 Defaults in the SQL\_Logger proc:

Defaults in the SQL\_Logger proc:

@RunNumber int = 0,

@LogLevel varchar(250),

@Title varchar(250) = NULL,

@StepNumber int = 0,

@StepName varchar(250) = NULL,

@StartTime DateTime = NULL,

@StopTime DateTime = NULL,

@ElapsedTimeSeconds Decimal(12,3) = NULL,

@Message varchar(200) = NULL

Allow for maximum flexibility in logging. One does not have to remember or populate all the parameters. The defaults provide for optional parameters as the specific situation warrants.

### 3.4 Simple Asynchronous Logging

Simple Asynchronous Logging is minimal logging.

Let’s take a look at minimal logging:

--===============================================================

-- SQL\_Logger Proc - Test 2 - Minimum and Default Parameters.sql

-- Grant Anderson

-- 5/7/2018

--===============================================================

USE [SQL\_Logger]

GO

EXEC dbo.SQL\_Logger @LogLevel = N'Info', @Title = N'Minimal Log Message', @Message = N'This is a minimal log message!'

This creates a very simple and minimal log entry consisting of just the LogLevel, the Title, and the Message. And of course the automated entries which are automatically populated.

Useful in minimal use situations and scenarios.

### 3.5 Synchronous Run Number Logging

Synchronous Run Number Logging uses Run Numbers to denote specific instances of the execution, the running, of specific SQL procs, scripts, and objects.

-- Start Logging

DECLARE @RunNumber int;

SELECT @RunNumber = NEXT VALUE FOR SQL\_Logger\_Run\_Number;

PRINT @RunNumber;

DECLARE @StartTime DATETIME;

SET @StartTime = SYSDATETIME();

EXEC @return\_value = dbo.SQL\_Logger @RunNumber = @RunNumber, @StepNumber = 1, @StepName = 'Start', @LogLevel = N'Info', @Title = N'Start SQL', @StartTime = @StartTime, @StopTime = NULL, @Message = N'This is a test!'

Here in this example we see the following parameters being used:

1. RunNumber
2. StepNumber
3. StepName
4. LogLevel
5. Title
6. StartTime
7. StopTime
8. Message

Note that the @RunNumber variable was previously populated with the next value of the Run Number sequence. And the @StartTime variable was also populated.

We can simplify both in this manner:

DECLARE @RunNumber int = NEXT VALUE FOR SQL\_Logger\_Run\_Number;

DECLARE @StartTime DATETIME = SYSDATETIME();

Note specifically that SQL Server proc parameters cannot use functions, subselects, or other high level constructs directly in the parameter entry. Thus we cannot simply substitute SYSDATETIME() directly in the logging procedure call parameters as convenient as that would be (as with C# and other languages). Thus we must first declare and assign a variable and use that in the parameters.

### 3.6 Simple Example – Simple Asynchronous Logging

Here is a simple example without the use of Run Number, Steps, Start Times, etc.

EXEC dbo.SQL\_Logger @LogLevel = N'Info', @Title = N'Truncate Table', @Message = N'Table = DimProduct'

TRUNCATE TABLE DimProduct;

EXEC dbo.SQL\_Logger @LogLevel = N'Info', @Title = N'Truncate Table', @Message = N'Table = DimProduct truncated.'

### 3.7 Simple Example – Synchronous Run Number Logging

And here is a much more capable and better example:

--------------------------------------------------------------------------------------

-- Logging Start.

--------------------------------------------------------------------------------------

DECLARE @RunNumber int;

SELECT @RunNumber = NEXT VALUE FOR SQL\_Logger\_Run\_Number;

DECLARE @Log\_Title varchar(250) = 'pLoadDimProduct';

DECLARE @StartTime DATETIME;

SET @StartTime = SYSDATETIME();

EXEC dbo.SQL\_Logger @RunNumber = @RunNumber, @StepNumber = 0, @StepName = 'Start', @LogLevel = N'Info', @Title = @Log\_Title, @StartTime = @StartTime, @StopTime = NULL, @Message = N'Start pLoadDimProduct'

--------------------------------------------------------------------------------------

-- Step 1

--------------------------------------------------------------------------------------

DECLARE @StartTime\_Step1 DATETIME = SYSDATETIME();

BEGIN TRY

---------------------------------------------

-- INSERT YOUR SQL HERE!!!

---------------------------------------------

TRUNCATE TABLE DimProduct;

DECLARE @StopTime\_Step1 DATETIME = SYSDATETIME();

EXEC dbo.SQL\_Logger @RunNumber = @RunNumber, @StepNumber = 1, @StepName = 'Step 1', @LogLevel = N'Info', @Title = @Log\_Title, @StartTime = @StartTime\_Step1, @StopTime = @StopTime\_Step1, @Message = N'Step 1 Ok.'

END TRY

BEGIN CATCH

DECLARE @StopTime\_Step1\_Catch DATETIME = SYSDATETIME();

EXEC dbo.SQL\_Logger @RunNumber = @RunNumber, @StepNumber = 1, @StepName = 'Step 1', @LogLevel = N'ERROR', @Title = @Log\_Title, @StartTime = @StartTime\_Step1, @StopTime = @StopTime\_Step1\_Catch, @Message = N'Error! Step 1 Catch Block! Truncation Error!'

END CATCH

--------------------------------------------------------------------------------------

-- Step 2

--------------------------------------------------------------------------------------

DECLARE @StartTime\_Step2 DATETIME = SYSDATETIME();

BEGIN TRY

---------------------------------------------

-- INSERT YOUR SQL HERE!!!

---------------------------------------------

INSERT INTO dbo.DimProduct

(

ProductID,

ProductName,

ProductDescription

)

VALUES

(

1,

'First Product',

'This is the first product...'

)

***-- Obviously, this will really be a complex SQL statement here…***

DECLARE @StopTime\_Step2 DATETIME = SYSDATETIME();

EXEC dbo.SQL\_Logger @RunNumber = @RunNumber, @StepNumber = 2, @StepName = 'Step 2', @LogLevel = N'Info', @Title = @Log\_Title, @StartTime = @StartTime\_Step2, @StopTime = @StopTime\_Step2, @Message = N'Step 2 Ok.'

END TRY

BEGIN CATCH

DECLARE @StopTime\_Step2\_Catch DATETIME = SYSDATETIME();

EXEC dbo.SQL\_Logger @RunNumber = @RunNumber, @StepNumber = 2, @StepName = 'Step 2', @LogLevel = N'ERROR', @Title = @Log\_Title, @StartTime = @StartTime\_Step2, @StopTime = @StopTime\_Step2\_Catch, @Message = N'Error! Step 2 Catch Block! Insert Error!'

END CATCH

--------------------------------------------------------------------------------------

-- Logging Stop.

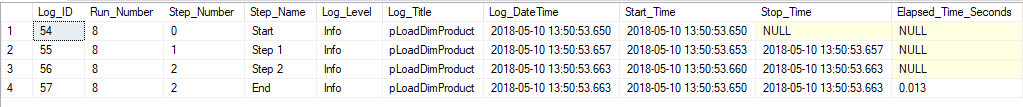
--------------------------------------------------------------------------------------

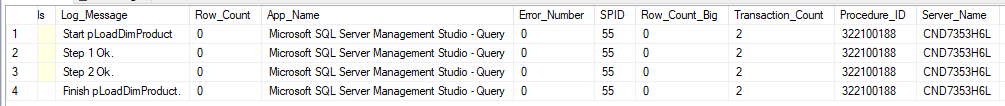
DECLARE @StopTime DATETIME = SYSDATETIME();

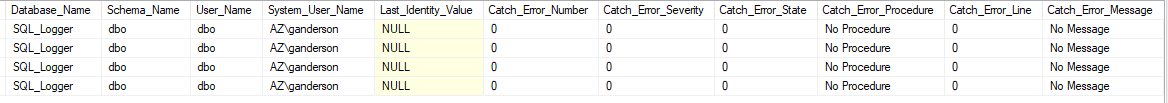
DECLARE @ElapsedTime Decimal(12,3) = CONVERT(DECIMAL, DATEDIFF(MS, @StartTime, @StopTime)) / 1000.0;

EXEC dbo.SQL\_Logger @RunNumber = @RunNumber, @StepNumber = 2, @StepName = 'End', @LogLevel = N'Info', @Title = @Log\_Title, @StartTime = @StartTime, @StopTime = @StopTime, @ElapsedTimeSeconds = @ElapsedTime, @Message = N'Finish pLoadDimProduct.'

This creates a really nice and detail log entry:







Obviously, this is what you should be using whenever and wherever possible to get the full functionality and advantages of the SQL Logger system.

## 4.0 Instrumenting Details with Code

This section details the logging code sections that can be quickly and easily applied to any SQL code or stored procedure to implement logging.

The sequence is:

1. Start Logging Code Block
2. Step 1 to N Code Blocks.
3. End Logging Code Block.

Simply install the Run Number sequence and the Logging proc in the database then just copy and paste these code blocks into your proc and SQL code. Then edit them as needed.

It’s that simple!

### 4.1 Starting Logging Code Block:

--------------------------------------------------------------------------------------

-- Logging Start.

--------------------------------------------------------------------------------------

DECLARE @RunNumber int;

SELECT @RunNumber = NEXT VALUE FOR SQL\_Logger\_Run\_Number;

DECLARE @Log\_Title varchar(250) = 'pLoadDimProduct';

DECLARE @StartTime DATETIME;

SET @StartTime = SYSDATETIME();

EXEC dbo.SQL\_Logger @RunNumber = @RunNumber, @StepNumber = 0, @StepName = 'Start', @LogLevel = N'Info', @Title = @Log\_Title, @StartTime = @StartTime, @StopTime = NULL, @Message = N'Start pLoadDimProduct

### 4.2 Steps 1 to N Code Blocks:

Add as many steps as you need to with cutting and pasting and then editing this code section:

--------------------------------------------------------------------------------------

-- Step 1

--------------------------------------------------------------------------------------

DECLARE @StartTime\_Step1 DATETIME = SYSDATETIME();

BEGIN TRY

---------------------------------------------

-- INSERT YOUR SQL HERE!!!

---------------------------------------------

DECLARE @StopTime\_Step1 DATETIME = SYSDATETIME();

EXEC dbo.SQL\_Logger @RunNumber = @RunNumber, @StepNumber = 1, @StepName = 'Step 1', @LogLevel = N'Info', @Title = @Log\_Title, @StartTime = @StartTime\_Step1, @StopTime = @StopTime\_Step1, @Message = N'Step 1 Ok.'

END TRY

BEGIN CATCH

DECLARE @StopTime\_Step1\_Catch DATETIME = SYSDATETIME();

EXEC dbo.SQL\_Logger @RunNumber = @RunNumber, @StepNumber = 1, @StepName = 'Step 1', @LogLevel = N'ERROR', @Title = @Log\_Title, @StartTime = @StartTime\_Step1, @StopTime = @StopTime\_Step1\_Catch, @Message = N'Error! Step 1 Catch Block! SQL Error!'

END CATCH

### 4.3 End Logging Code Block:

And then finalize with this code block.

--------------------------------------------------------------------------------------

-- Logging Stop.

--------------------------------------------------------------------------------------

DECLARE @StopTime DATETIME = SYSDATETIME();

DECLARE @ElapsedTime Decimal(12,3) = CONVERT(DECIMAL, DATEDIFF(MS, @StartTime, @StopTime)) / 1000.0;

EXEC dbo.SQL\_Logger @RunNumber = @RunNumber, @StepNumber = 2, @StepName = 'End', @LogLevel = N'Info', @Title = @Log\_Title, @StartTime = @StartTime, @StopTime = @StopTime, @ElapsedTimeSeconds = @ElapsedTime, @Message = N'Finish pLoadDimProduct.'

Obviously, there’s different ways to use logging. This will get you started.

## 5.0 Reading the Log

Reading the log is simply reading the SQL\_Logger\_Log table entries and filtering to get the ones that you want and need.

### 5.1 Direct Reading of the Log File

Simply use a SQL query to read the desired columns from the log table and filter on Run\_Number and/or Log\_DateTime so as to see the entries that you want.

SELECT

Log\_ID,

Run\_Number,

Step\_Number,

Step\_Name,

Log\_Level,

Log\_Title,

Log\_DateTime,

Start\_Time,

Stop\_Time,

Elapsed\_Time\_Seconds,

Log\_Message,

Row\_Count,

App\_Name,

Error\_Number,

SPID,

Row\_Count\_Big,

Transaction\_Count,

Procedure\_ID,

Server\_Name,

Database\_Name,

Schema\_Name,

User\_Name,

System\_User\_Name,

Last\_Identity\_Value,

Catch\_Error\_Number,

Catch\_Error\_Severity,

Catch\_Error\_State,

Catch\_Error\_Procedure,

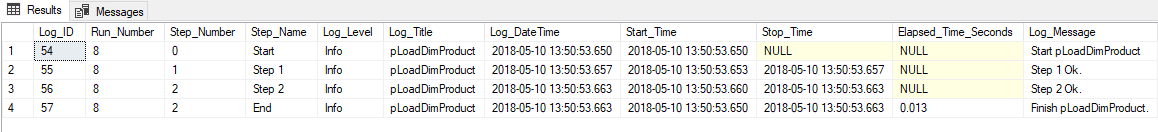
Catch\_Error\_Line,

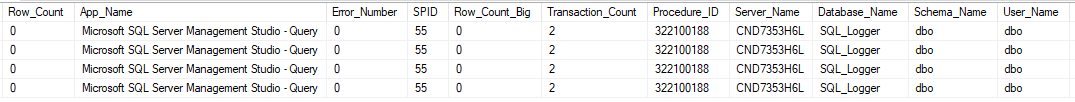
Catch\_Error\_Message

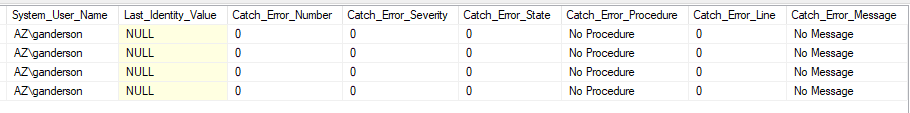
FROM SQL\_Logger.dbo.SQL\_Logger\_Log

WHERE Run\_Number = 8

Query Results:







### 5.2 Using the SQL Logger Views

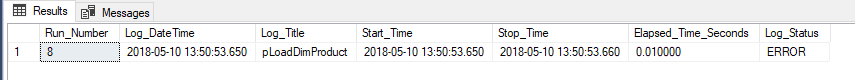
Using the SQL Logger reporting views simplifies the display of the log records.

### 5.2.1 SQL Logger Summary View

SELECT \*

FROM SQL\_Logger.dbo.SQL\_Logger\_Summary

WHERE Run\_Number = 8

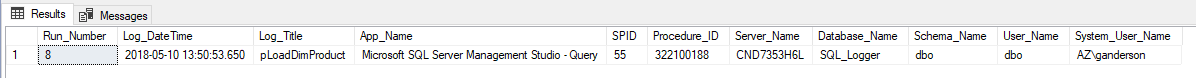


### 5.2.2 SQL Logger Info View

SELECT \*

FROM SQL\_Logger.dbo.SQL\_Logger\_Info

WHERE Run\_Number = 8

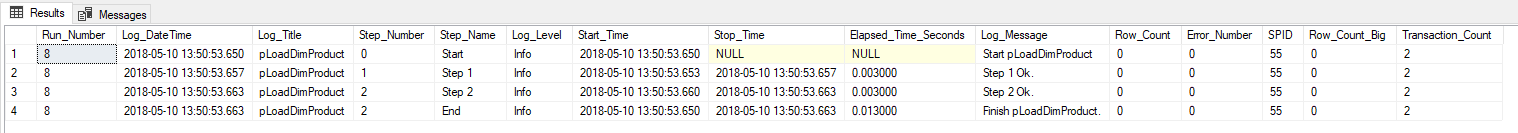


### 5.2.3 SQL Logger Steps View

SELECT \*

FROM SQL\_Logger.dbo.SQL\_Logger\_Steps

WHERE Run\_Number = 8

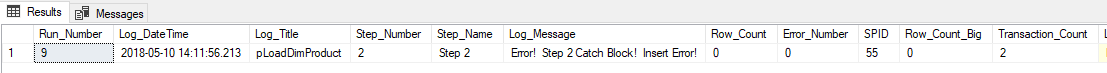


### 5.2.4 SQL Logger Error View

SELECT \*

FROM SQL\_Logger.dbo.SQL\_Logger\_Error

WHERE Run\_Number = 9



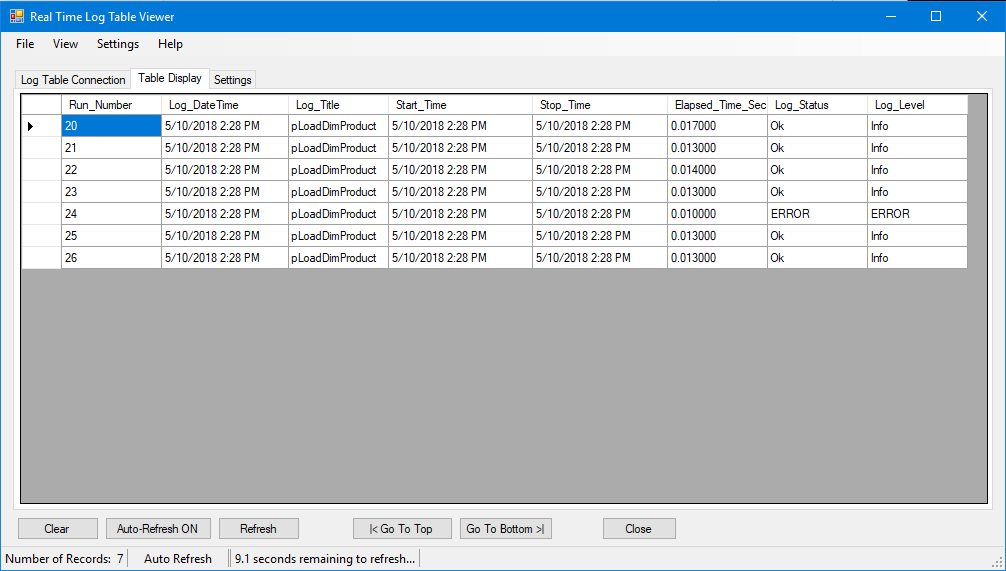


Note: This is a simulated error using a “SELECT 1/0;” statement. Nice error explanation in the Catch\_Error\_Message column!

### 5.3 Using a Log Reading Program Tool

A log reader tool, such as the one shown below, can be used to show the real time status of the logging. Here, this tool automatically updates the display every 10 seconds and thus displays a real time view of this particular SQL proc, pLoadDimProduct, every 10 seconds.

Very useful for monitoring SQL processes and procs.



## 6.0 Part of the Larger Infrastructure

The SQL Logger system can be easily used with and integrated with logging of other database systems including SSIS and C# and other programs including Windows services.

**Architecture Diagram:**

**Web Log Dashboard and Viewer**



**Master Log Database** with:

1. High Level Logs
2. Detailed Logs

**Individual Ops DBs**

**With Individual Logs**

**1. View Level**

**C# Processes**

**SSIS Processes**

**SQL Processes**

**2. Central Storage Level**

**3. Collection Level**

**Key Features:**

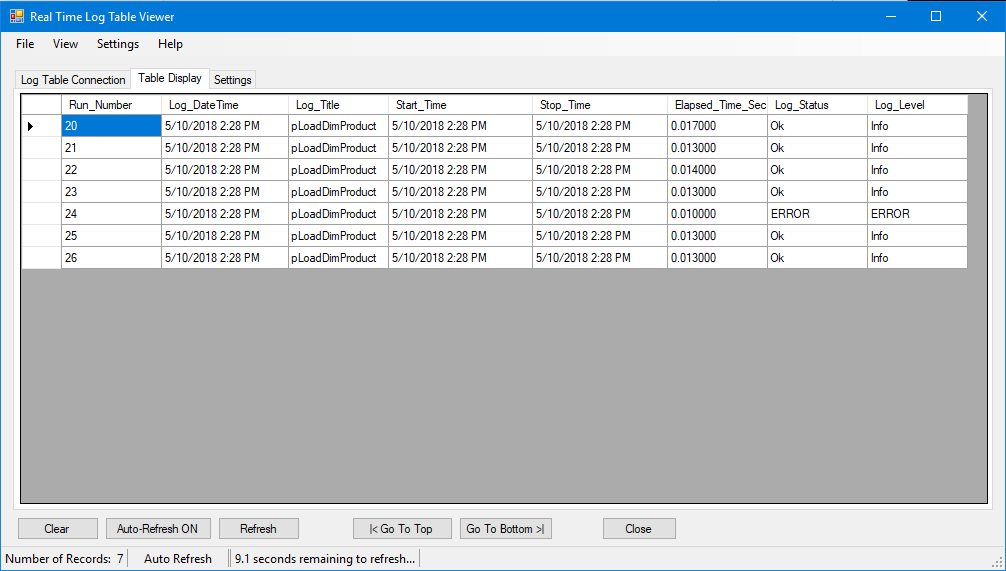
1. Provides complete enterprise level logging for a multitude of logging scenarios.
2. Standardized logging schema and table structure.
3. Flexible logging functions.
4. Common tool base available for all personnel.

**Major Components:**

1. View Layer System Components:
   1. Log Viewer
   2. Log Exceptions Alerter
   3. Web Log Dashboard
2. Central Storage Layer System Components:
   1. Central Master Log Database
   2. Individual Process Databases
   3. Standardized Log Tables and Schemas
3. Collection Layer System Components:
   1. SQL Logging Component – SQL Statements
   2. SSIS Logging Component – SQL Statements (Future Native SSIS Component)
   3. C# Logging Component – Library
      1. Database
      2. Web Services

### 6.1 Log Viewer

A log viewing tool, such as the one in the below screenshot, can be used to easily and quickly view log table records without custom SQL query coding.



Additional features, such as Auto Update every N seconds, allows the tool to be used for real time monitoring purposes. Very useful to be able to see what’s going on in the system.

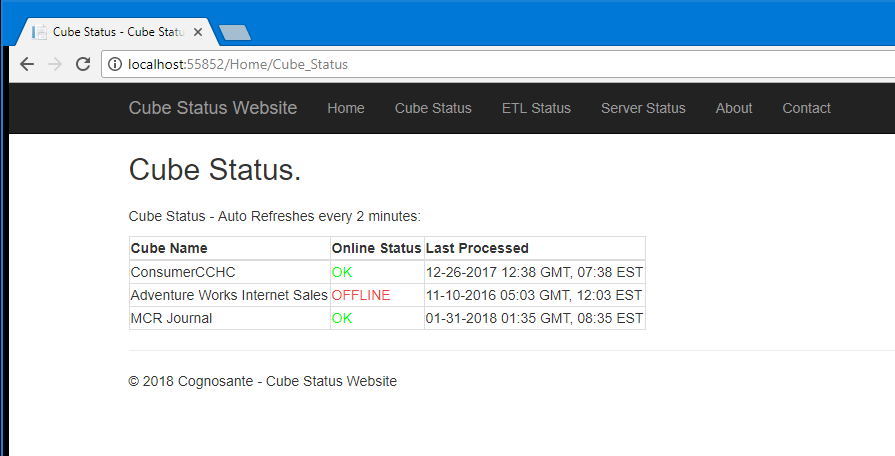
### 6.1 Log Exceptions Alerter

A Log Exceptions Alerter would be a Windows Service that would regularly scan the Log table(s) and find and extract warning and error conditions and send email, SMS text alerts, and desktop alerts to support personnel. This is a very proactive support tool.

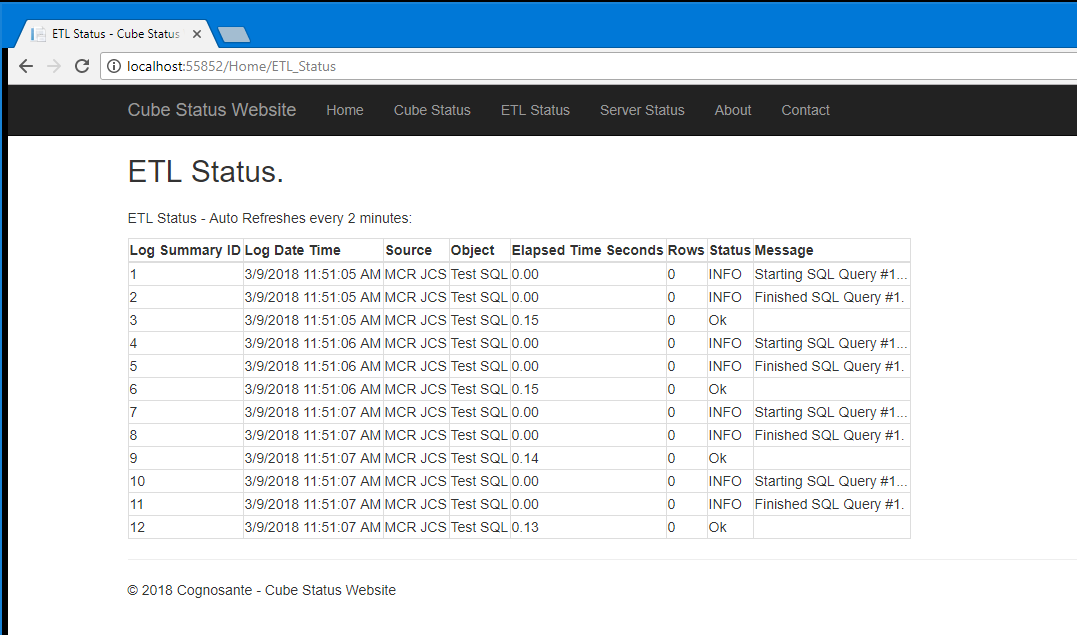
### 6.1 Web Log Dashboard

A Web Log Dashboard provides a real time web page intranet accessible view of the database and SQL Systems.

Here a Cube Status Website provides a real time display of OLAP cube status including the online/offline status and the time the cubes were last processed (which is extremely useful for reports and report users to know the “data freshness” date).



And here is a screenshot of the ETL Logging display of the Cube Status Website portal:



Once integrated with SQL Logger this page can be expanded to show summary, info, step, and error details and displays. A one stop location for support people to see system status and investigate problems rather than spending time digging around tables with SQL queries trying to figure out what when wrong.