http://www.sqlservercentral.com/articles/Administration/3243/
Printed 2007/10/10 07:50AM

Deadlock Notifications in SQL Server 2005

By Patrick LeBlanc, 2007/09/04

Overview

A client, running SQL Server 2005, recently contacted me because they were experiencing deadlocks. This client wanted to be notified, via email, whenever a deadlock occurred.To get this, normally, I would run the following:

```
sp_altermessage @message_id = 1205,
                @parameter = 'write_to_log',
                @parameter_value = 'true'
```

However, in SQL Server 2005 you receive the following error:

```
Msg 15178, Level 16, State 1, Procedure sp_altermessage, Line 20
Cannot drop or alter a message with an ID less than 50,000.
```

In both SQL Server 2000 and 2005 you can create alerts, but in SQL Server 2000 you were able to run sp_altermessage against any message in the sysmessages table, including a message with an ID less than 50,000. The effect of sp_altermessage with the WITH_LOG option will cause the message to be written to the Windows NT application log. This will also cause any SQL Alert that is associated with that message to fire.

However, in SQL Server 2005 you can only run sp_altermessage against messages with IDs greater than or equal to 50,000. Therefore, associating a system message with an Alert that does not have WITH_LOG specified will not fire the Alert. As a result, I have written a workaround.

Prerequisites : Please note that you must have Database Mail configured, SQL Server Agent must have the correct Alert System Configurations set to utilize this method and Traceflag 1222 must be on (DBCC TRACEON(1222, -1).

To make this work, first you will need to create the stored procedure that performs several steps, which are outlined below. Run this script to create the stored procedure:

```
IF OBJECT_ID( 'dbo.usp_DeadlockNotification') IS NOT NULL
   DROP PROC dbo.usp_DeadlockNotification
--GO
CREATE PROC dbo.usp_DeadlockNotification
   @FilterBIT = 0,
   @Minutes INT = 30
AS
DECLARE @ErrorLog TABLE
(
  LogDate DATETIME NOT NULL,
 ProcessInfo  VARCHAR(75),
 LogInfo VARCHAR(MAX)
```

```
)

DECLARE @Count INT,
        @StartDate DATETIME,
        @EndDate DATETIME

SET @Count = 0

SET NOCOUNT ON

-- Step I: Import Errorlog
INSERT INTO @Errorlog
  EXEC xp_readerrorlog

---- Step II: How to search Errorlog
IF (@Filter <> 0)
BEGIN
   SELECT @EndDate = GETDATE()
   SELECT @StartDate = DATEADD(mi, -@Minutes, @EndDate)

    SELECT @Count = COUNT(*)
     FROM @Errorlog
     WHERE LogDate BETWEEN @StartDate AND @EndDate
     AND LogInfo LIKE '%Deadlock%'
END
ELSE
BEGIN
    SELECT @Count = COUNT(*)
FROM @Errorlog
      WHERE LogInfo LIKE '%Deadlock%'
END

---- Step III: Send Email
IF (@Count > 0)
BEGIN
   EXEC msdb.dbo.sp_send_dbmail
 @profile_name = 'SQLTestCluster',
 @recipients   = 'pleblanc@lamar.com',
 @subject      = 'Deadlocks',
  @body = 'Please check errorlog for Deadlocks'
END
```

The stored procedure can be divided into four basic steps, and they are as follows:

Step I: Import Errorlog

The SQL Server Error Log is imported into variable table using the xp_readerrorlog undocumented stored procedure. I will not go into much detail about the system stored procedure, but you can get more information from the following blog:http://blogs.sqlservercentral.com/blogs/robert_davis/articles/SQL_Server_Error_Logs.aspx

Step II: How to Search Errorlog Variable Table

At this point the current errorlog has been imported into the variable table. The stored procedure has two optional parameters, @filter and @minutes. If @filter is set to anything other than zero a restricted search is performed on the Errorlog variable table. The filter is based on how many minutes have been specified for the @minutes parameter. If you set @minutes to 60, all the rows that were logged to the

errorlog within the last hour will be searched. If the @filter parameter is not set the entire errorlog is searched for deadlocks.

Step III: Send Email

Finally, if the local @Count variable is greater than zero an email is sent. Note in this part of the stored procedure you will have to change the @profile_name parameter and the @recipients parameter.

Finally, create a SQL Job that runs at intervals during the day, specifically when you think the deadlocks occurring. Add one step to the job that runs the stored procedure. An example of the stored procedure execution is as follows:

```
EXEC dbo.usp_DeadlockNotification 1, 60
```

Then schedule the job to run as needed.

Further research, revealed that another possible method of detecting deadlocks programmatically would be to use the sys.dm_os_performance_counters dynamic management view. Replace the above dbo.usp_DeadlockNotification stored with the following:

```
CREATE PROC dbo.usp_DeadlockNotification
AS
DECLARE @NumOfDeadLocks int

SELECT @NumOfDeadLocks = SUM(cntr_value)
FROM sys.dm_os_performance_counters
WHERE counter_name like '%dead%'

SET NOCOUNT ON

IF(@NumOfDeadLocks > 0)
BEGIN
  EXEC msdb.dbo.sp_send_dbmail
 @profile_name = 'SQLTestCluster',
 @recipients = 'pleblanc@tsqlscripts.com',
 @subject = 'Deadlocks',
 @body = 'Please check errorlog for Deadlocks'

END
```

I did not use this stored procedure in my example because I am still researching the behavior of the sys.dm_os_performance_counters dynamic management view. I am unsure of how often the data is reset or refreshed. Using this method could yield unwanted Alerts.

Conclusion

The client was not sure when the deadlocks were occurring therefore we searched the errorlog every 30 minutes using the following stored procedure call:

```
EXEC dbo.usp_DeadlockNotification 1, 30
```

We were notified after the first run of the Job. I added an index and modified the two stored procedures to ensure that the operations (UPDATE, INSERTS, DELETES) on the objects were executing in the

same order.

If you have any questions, concerns, comments or suggestions regarding this article please email me at pleblanc@tsqlscripts.com.
