

Re: Transactions across batches

Source:

<http://www.tech-archive.net/Archive/SQL-Server/microsoft.public.sqlserver.server/2004-12/3388.html>

From: AnthonyThomas (*Anthony.Thomas_at_CommerceBank.com*)

Date: 12/30/04

Date: Thu, 30 Dec 2004 08:25:06 -0600

I've come across this issue several times in various ways and it is not specific to the Yukon code base.

Consider using the Server Defaults. My opinion is that anything coded should control its environment settings, transactions, isolation, and errors, which needs to be taught. However, I always set up my Server installations with the system defaults set, not to the out-of-the-box configuration, but to certain standards that deal with Ad Hoc users that don't necessarily know what they are doing. As I said, anything coded SHOULD KNOW.

So, here they are. I use the USER OPTIONS value, which is a bit mask, to set up default user connection environment settings. Now, a user at the Connection, API, Session, or Statement level has the ability to override these settings, but typically does not. Coded solutions should always do this regardless of the Server settings—those may change in the future and we do not want to have to recode the apps.

Set the ANSI compliant defaults: ANSI NULLS, ANSI WARNINGS, ANSI PADDING, QUOTED IDENTIFIER, ARITHABORT, CONCAT NULL YIELDS NULL, and NUMERIC ROUNDABORT OFF. Set CURSOR LOCAL, NOCOUNT ON, and IMPLICIT TRANSACTIONS OFF. And, here's the best one, SET XACT ABORT ON.

This is what it buys you. If you use or plan to use materialized Views, then the ANSI settings are required at creation and for every runtime query, coded or Ad Hoc. The local cursor setting helps with concurrency issues. The NOCOUNT reduces system overhead. Your issue may have to do with the default, out-of-the-box handling of Implicit Transactions: if explicit transaction are NOT used and this setting is off, then each statement, whether a batch by itself or sent with other statements commits with the successful completion of each statement (AUTOCOMMIT mode). Lastly, few users control their error handling correctly; so, the XACT ABORT setting ensures that the batch will rollback if an error is encountered, any error, even informational ones. This can cause problems, but not if you know what you are doing...and that is the point.

microsoft.public.sqlserver.server: Re: Transactions across batches

As far as the long running, idle queries, check the last_batch_time attribute in sysprocesses. You can do a DATEDIFF calculation on this and KILL the spid if it hasn't done anything in awhile. I have a SQL Agent job that runs once an hour that KILLS spid that haven't done anything for the past 6 hours, calculated on number of milliseconds (21,600,000 ms = 21,600 sec = 360 min = 6 hours) on the DATEDIFF function.

Sincerely,

Anthony Thomas

--

"Hans S. Tømmerholt" <Hans S. Tømmerholt@discussions.microsoft.com> wrote in message news:1A361BBF-848E-4F1D-8591-3BFE47E22D12@microsoft.com...

Hi!

I'm an assistant on a University course where we use the SQL Server 2005 Yukon beta. Students connect to an MS SQL database through PHP. We have a problem where our students forget to rollback/commit their transactions. As a

result, the transactions wait, and span new batches later on, keeping locks on different resources. This seriously bogs down our server.

We are wondering if there is some way of not allowing transactions to span more than one batch, or some way to force an auto-rollback of such transactions. Searches in the online documentation has not yet yielded any results.

Alternatively, is there some way of programatically interacting with running transactions? We've considered a daemon script which checks for long-running transactions/processes and kills them. We've seen C#/C++ transaction objects in the documentation, but no way of accessing the internals of the running server.

Any ideas?

Sincerely yours

Hans S. Tømmerholt

Department of Informatics, University of Oslo, Norway