

Re: Please advise on approach

Source:

<http://www.tech-archive.net/Archive/SQL-Server/microsoft.public.sqlserver.server/2004-09/3227.html>

From: Scott Morris (*bogus_at_bogus.com*)

Date: 09/22/04

Date: Wed, 22 Sep 2004 14:17:18 -0400

First, timestamp has no relationship to date or time; the synonym is rowversion. It is unique for a db and appears to be currently be implemented in such a way as to be predictable. Based on the current implementation, one can make comparisons (involving greater than and less than) that give correct results. However, I don't think the current implementation can be safely relied on and, IIRC, MS only supports equal / not equal comparisons. Note – since you are posting in a db-related NG, I assume that your reference to timestamp is actually a reference to the sql server datatype. If this assumption is incorrect – well that's your fault for using an ambiguous term outside of the implied context.

Second, datetime values are accurate to 3 milliseconds. If your term "timestamp" refers to a datatype of this nature, is this accuracy sufficient? Note that there are two aspects to accuracy. First is the relative difference in accuracy between your datasource timestamp values and this datatype. The second aspect is relative to a difference in accuracy. Your timestamp data might be more accurate (e.g., 1 ms) but occurs at intervals that are significantly less accurate (e.g., > 3 ms). Of course, you could use a different datatype, losing the ability to use the builtin datetime functions and creating potential ordering problems.

Third, a table is, by definition, unordered. Your narrative assumes and implies otherwise. Clustering does affect physical ordering. However, there is nothing that can guarantee you can access rows in physical insertion order unless that order can be determined by the data itself. In other words, you must provide a way to include the appropriate information in an order by clause. Even with a heap and a cursor, I'm not certain that there is any way to guarantee the desired order.

Given this information, how do you intend to store the data (regardless of number of tables) in such a way that one can reconstruct the events in chrono. order? There are two implications in your narrative that might be problematic. First, is that "reconstruction" implies a single thread of execution. Is this valid? If not, then you might also need to record additional information to differentiate the multiple streams of data (which just might be adding data at the same instant in time). The other potential pitfall is that reconstruction often implies "re-running" (e.g.,

reconstruction of the data in the same order/timeframe). Fear this!

So now we're down to identifying chronological order. Assume that source A has data with timestamp 10:22:01 (ignoring the date portion for simplicity) and source B has data with the same timestamp (you said this was possible). They both get inserted into the DB (regardless of table) at roughly the same time. Based on data alone, it is not possible to determine which is first. What does the DB have to assist? You could use a timestamp for the table (or tables) and rely on the current implementation to derive order. Someone recently posted that this type of solution was working. You could use a datetime column to mark each row with the datetime of insert. This approach may suffer due to accuracy limitations. Lastly, you could use a single table with an identity. This should logically work (and it is what you considered) but may be problematic due to contention / locking. Hotspots will definitely be something to avoid. Note that this approach is also predicated on the immediate insertion of data, implying that every data "event" is recorded on a one-by-one basis. This is not the best approach in terms of network usage (and does not scale well).

You might want to investigate any potential client-side approaches as well. In a single-threaded application, you could also impose your own serial numbering on the data before insertion.

Lastly, you might want to indicate what your ultimate goal is. Relational databases are often slower than file-based approaches for data of this type (there is a lot of overhead that you may not need for recording data). There are other ways to record "data" in fifo order that might later be imported into a real database for analysis or reporting. If there are no real relationships in your data, then perhaps you don't need a rdbms. Without knowing what you intend to do with the information, this is the approach I would investigate first.

"Paul" <paulsmith5@hotmail.com> wrote in message
news:ca236fb1.0409220850.2058abc9@posting.google.com...

> Hi,

>

> *Apologies for the vague subject title but I couldn't think of a snappy
> title to describe what I am doing. Please bear with me as I explain...*

>

> *I am developing an application that records data from several
> different sources operating at relatively high rates (e.g. >5 times
> per sec). The data from the various sources is diverse, in terms of
> the type (e.g. data from source A may comprise of 5 parameters in
> binary format, whereas data from source B may comprise of a single
> parameter in ascii format). The one thing they have in common is a
> timestamp. The timestamp although unique for each source is not unique
> amongst all sources. For example although there will be only ever be a
> single row in TableA with a particular timestamp representing a
> particular point in time, there may also be a row in TableB with the
> same timestamp. In an attempt to have the writes as efficient as
> possible and to prevent the tables growing too big I record data from*

microsoft.public.sqlserver.server: Re: Please advise on approach

> each source to different tables. These tables are created at runtime
> before I begin to record the data. Following a period of recording
> some tables may still have in excess of 500,000 rows. I now wish to
> reconstruct the sequence of events in chronological order, i.e.
> retrieve the earliest data recorded from all the sources, followed by
> the next and so on. I'm wondering how to approach this. I was thinking
> of creating a single large table into which I would insert the
> timestamp and ID of every row from each source as well as the table
> name, then ceating a clustered index on the timestamp field, so that
> the rows would be physically sorted in chronological order. This table
> may look like the following when complete
>
> 1 09/22/2004 16:00:00 Table1
> 2 09/22/2004 16:00:01 Table1
> 1 09/22/2004 16:00:02 Table2
> 3 09/22/2004 16:00:03 Table1
> 2 09/22/2004 16:00:03 Table2
> etc.
>
> This would only ever have to happen once (I appreciate it may take
> some time to construct) – then each time I need to I could simply move
> through the table row by row retrieving the ID and table name which I
> could then use as parameters in a query that would retrieve the data.
> I'd appreciate any feedback on this approach i.e. is it madness! I was
> wondering about views etc. but I don't know if these could help at all
>
> Thanks,
>
> Paul

Re: Please advise on approach