

## Re: Using transactions to insert into to a table while allowing read access to existing data

**Source:**

<http://www.tech-archive.net/Archive/SQL-Server/microsoft.public.sqlserver.programming/2004-11/2604.html>

---

**From:** Sean Aitken ([sean.aitken\\_at\\_tekelec.spamtrap.com](mailto:sean.aitken_at_tekelec.spamtrap.com))

**Date:** 11/10/04

Date: Wed, 10 Nov 2004 13:41:57 -0500

Thanks for the thorough reply.

It seems like there would be a good enhancement to add to an RDBMS that would allow for "transparent" transactions, allowing for this kind of thing.. perhaps through the use of temporary tables behind the scenes.

I think I am just too stubborn to accept the fact that the design of a transaction in general always updates actual data, and in all cases relying on the transaction log for 'rollbacks'.

Thanks for spelling it out. I think I'm going to use a session temporary table and simply "select into ..." at the end of the inserts.

I must admit, that this kind of update is one that wouldn't happen normally. We are taking a snapshot of an LDAP directory are pumping it over a WAN to a remote SQL Server.

Many thanks!

-Sean

Steve Kass wrote:

> Sean,  
>  
> *I don't think what you want is possible, at least not in any simple  
> way. If an update has been made but is uncommitted, "data as it existed  
> prior to the transaction" is simply not available, at least not for any  
> purpose but to roll back the transaction.*  
>  
> *Suppose you are halfway through the update transaction and have  
> completed but not committed 500 updates. Now someone attempts to read  
> one of these 500 rows. The "data as it existed prior to the  
> transaction" is not in the table. The rows have been updated, and the  
> rows with uncommitted updates are marked with locks. The updates are  
> uncommitted, so a SELECT will be blocked from reading them if the*

> isolation level is *READ COMMITTED* (the default), or a *SELECT* will be  
> allowed to read them if the isolation level is *READ UNCOMMITTED* (or if  
> there is a *NOLOCK* hint on the *SELECT*).  
>  
> Either the *SELECT* is blocked or it reads the uncommitted updated data.  
> Those are the only choices unless you introduce a staging table for the  
> updates, row versioning, or some other sort of logic to handle the data  
> while it is being updated.  
>  
> Steve Kass  
> Drew University  
>  
> Sean Aitken wrote:  
>  
>> Hello,  
>>  
>> Sorry for the long subject, but this is a very interesting problem I am  
>> having. I am faced with the following situation:  
>>  
>> – Single table used by various applications for read-only lookups  
>> – Updates to that table are slow due to network latency (~5 minutes)  
>>  
>> I am updating the table by beginning a transaction, performing many  
>> inserts, then finally committing the transaction. The problem I am  
>> encountering is that during the entire transaction, this table is  
>> locked for the duration of the transaction. This is preventing the  
>> consumer apps (there are quite a few of them, over 1,000 users) from  
>> *SELECT*'ing any data, as there is a block on the table.  
>>  
>> I have tried using batch inserts (I am using a *JDBC* interface),  
>> setting various transaction isolation levels, and am about to resort  
>> to inserting into a temporary table and bulk-copying the data after  
>> the insert.  
>>  
>> What I would like to achieve is the 5 minute update, and allow users  
>> to see the data as it existed prior to the beginning of my  
>> transaction. Once I commit, suddenly, they would see new data. I can  
>> understand the situation if two transactions begun and overlapped, one  
>> would have to lose, but in this case, the table is only read only to  
>> the consumer apps.  
>>  
>> It seems that this is a feasible thing to expect, but I can't figure  
>> out if it is a limitation of the technology or my own ignorance.  
>>  
>> Any help, pointers, ideas *GREATLY* appreciated!  
>>  
>> Thank you,  
>> –Sean