

Re: Discussing 3 different strategies for deleting from multiple tables

Source: <http://www.tech-archive.net/Archive/Data/microsoft.public.data.ado/2005-10/msg00080.html>

- *From:* "Mark J. McGinty" <mmcginty@xxxxxxxxxxxxxxxxx>
 - *Date:* Mon, 17 Oct 2005 11:59:47 -0700
-

"Jiho Han" <jhan@xxxxxxxxxxxxxxxxx> wrote in message
news:a19ab9b12c7a12a17347378@xxxxxxxxxxxxxxxxxxxxxxxxx

> Thanks for your input.

>

> From what I understand SQL Server (as of 2000) did not support parameters
> in the subquery. That was the case for T-SQL as well as anything going
> through the OLE DB Provider.

I fear you have been misinformed:

```
CREATE PROCEDURE sptestsubqueryparams (@id int)
AS
SELECT * FROM (
SELECT * FROM sysobjects
WHERE (id = @id)
) so
```

Albeit a contrived example, it works, even if you nest derived rowsets 10 levels deep. (Maybe even deeper, 10 was as deep as I tested -- which promoted it from a merely contrived to a massively contrived example.) :-)

-Mark

> So yes, I will be using SQL Server but I am riding on top of a third party
> OLE DB Provider. I cannot create triggers nor stored procedures to do the
> job I'm afraid.

> I am looking for the least costly method that can work completely on the
> application side.

>

> Jiho

>

>> You're right that method #1 is costly, that approach is suitable for
>> periodic maintenance... is it really that important that orphaned
>> records get cleaned-up in real-time on a per top-level delete basis?
>>

Re: Discussing 3 different strategies for deleting from multiple tables

>> Method #2 is a little more frugal, but still pricey, and I can
>> certainly see your point about query complexity -- btw, what db engine
>> restricts use of parameters in sub-queries?
>>
>> I doubt method #3 will work reliably, if at all, but if it did, it
>> would involve many, many more server round trips under the covers.
>> (ADO is famous for generating excessive server traffic in certain
>> situations.) It would also almost certainly be many times slower than
>> using set-based ops.
>>
>> The significant question is, what db engine will you be using? If SQL
>> Server, you might want to consider using triggers to cascade these
>> deletes, that way they would always be maintained even if rows were
>> deleted outside of your app. Another advantage would be that the set
>> of IDs required by each lower-level delete would already be resolved
>> when the triggers are called, reducing complexity of each op to only
>> that required to maintain the next level down. You'd have to make
>> sure that no locking conflicts would result, and of course, nested
>> triggers would have to be enabled.
>>
>> Last note, certainly the cascading delete paradigm is not atypical,
>> but 12 relationships tends a little towards the frisky side.
>>
>> -Mark
>>
>> "Jiho Han" <jhan@xxxxxxxxxxxxxxxx> wrote in message
>> news:a19ab9b12c47fb8c79ed85940777f@xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
>>
>>> I have a database where tables are structured so that deleting a
>>> record in
>>> one of the table must also result in deletes from 8 other tables.
>>> I am guessing that this is not atypical in applications out there.
>>> This
>>> is due to how the tables are hierarchically related to one another.
>>> In
>>> some situations, it may involve more than 12 tables even.
>>> Having said that I would like to discuss and get some feedback on
>>> what the
>>> best strategy may be to accomplish this goal. Note that the solution
>>> obviously must incorporate ADO.
>>> Also note that none of the databases have "hard" foreign key
>>> relationships
>>> defined and assume that I cannot create them either.
>>> I will only use 3 tables for brevity - Account, Contact, Opportunity
>>> - and
>>> assume the following relationship. FYI, Account contains around 20K
>>> records, Contact 80K, and Opportunity 700K.
>>> Account.Accountid = Contact.Accountid
>>> Contact.Contactid = Opportunity.Contactid
>>> So a deleting an account must result in a delete of all contacts
>>> under that account and a delete of all opportunities under those

Re: Discussing 3 different strategies for deleting from multiple tables

>>> contacts as well.

>>>

>>> -----

>>> Method 1 –

>>> DELETE FROM ACCOUNT WHERE ACCOUNTID = ?

>>> DELETE FROM CONTACT WHERE ACCOUNTID NOT IN (SELECT ACCOUNTID FROM
>>> ACCOUNT)

>>> DELETE FROM OPPORTUNITY WHERE CONTACTID NOT IN (SELECT CONTACTID FROM
>>> CONTACT)

>>> I can see this method as the slowest but at the same time simple and

>>> foolproof.

>>> This results in one parameterized query followed by two more trips to

>>> the

>>> database to delete any orphaned records. This method also has the

>>> added

>>> side-effect of possibly cleaning up any "dirty" data from any other

>>> sessions.

>>> And perhaps the above queries can be rewritten using EXISTS to make

>>> it more efficient. But the idea is same. Perhaps something like

>>> this:

>>>

>>> DELETE FROM OPPORTUNITY WHERE NOT EXISTS (SELECT CONTACTID FROM
>>> CONTACT

>>> WHERE CONTACTID = OPPORTUNITY.CONTACTID)

>>> -----

>>> Method 2 –

>>> DELETE FROM OPPORTUNITY WHERE OPPORTUNITYID IN (

>>> SELECT OPPORTUNITYID FROM OPPORTUNITY A1 INNER JOIN CONTACT A2 ON
>>> (A1.CONTACTID = A2.CONTACTID) WHERE A2.ACCOUNTID = ?

>>>)

>>> DELETE FROM CONTACT WHERE ACCOUNTID = ?

>>> DELETE FROM ACCOUNT WHERE ACCOUNTID = ?

>>> This method works by deleting from the "leaf" and works up to the

>>> "root".

>>> This may be more efficient but as the number of tables involved gets

>>> larger and larger, the queries will get more complex. Also, the

>>> first

>>> delete statement in the example will not work since you cannot have a

>>> parameter in the subquery. Although it's not essential, I would

>>> rather

>>> have a solution that does not use string concatenation to generate a

>>> query.

>>> -----

>>> Method 3 –

>>> SELECT A1.ACCOUNTID

>>> FROM ACCOUNT A1 LEFT JOIN CONTACT A2 ON (A1.ACCOUNTID = A2.ACCOUNTID)

>>> LEFT JOIN OPPORTUNITY A3 ON (A2.CONTACTID = A3.CONTACTID)

>>> WHERE A1.ACCOUNTID = ?

>>> I would retrieve the result into a recordset and simply delete all

>>> rows returned by calling .Delete method on each row. And when I

>>> Update the recordset, ADO will generate all necessary delete sql

Re: Discussing 3 different strategies for deleting from multiple tables

>>> statements. I am basically benefitting from not specifying Unique
>>> Table property and thus letting ADO delete all involved rows from all
>>> tables in the query. The added benefit of this method is that there
>>> is a single trip to the database (well two, if you count the select).
>>> However, as more tables are involved, the query gets complex and the
>>> number of joins required goes up. -----
>>>
>>> I am leaning towards method 1 for the moment (at least while in
>>> initial development). Method 2 seems like it doesn't provide much
>>> more benefit to Method 3 as the level of depth grows larger, the
>>> resulting subquery will approximate the query in Method 3 anyway.
>>>
>>> I would love for some feedback on this and experiences you've had on
>>> similar situations.
>>> Thanks so much.
>>> Jiho Han
>>> Senior Software Engineer
>>> Infinity Info Systems
>>> The Sales Technology Experts
>>> Tel: 212.563.4400 x216
>>> Fax: 212.760.0540
>>> jhan@xxxxxxxxxxxxxxxxxxx
>>> www.infinityinfo.com
>
>

• **Follow-Ups:**

- ◆ **[Re: Discussing 3 different strategies for deleting from multiple tables](#)**
◇ From: Jiho Han

• **References:**

- ◆ **[Re: Discussing 3 different strategies for deleting from multiple tables](#)**
◇ From: Mark J. McGinty
- ◆ **[Re: Discussing 3 different strategies for deleting from multiple tables](#)**
◇ From: Jiho Han

- Prev by Date: **[Re: Store procedure](#)**
- Next by Date: **[Re: Discussing 3 different strategies for deleting from multiple tables](#)**
- Previous by thread: **[Re: Discussing 3 different strategies for deleting from multiple tables](#)**
- Next by thread: **[Re: Discussing 3 different strategies for deleting from multiple tables](#)**
- Index(es):
 - ◆ **[Date](#)**
 - ◆ **[Thread](#)**