SQL Excluding Record sets

Posted At : July 14, 2011 9:45 AM | Posted By : Steve
Related Categories: SQL

I was eating with another speaker at **cf.Objective()** (I am resolved to find a way to work my having spoken at a conference into all future anecdotes) and an interesting SQL question came up: How to delete everything *except* the first 1000 records. To my mind, this brought up a general class of problems in SQL. Which is, returning results that exclude the result of a query.

With that in mind, let's look at a few of those.

Not IN

This is the approach most likely to work for an UPDATE or DELETE query. Let's take a look at an example of getting all of the records in a table *except* the first 1000 records.

```
SELECT   LogID, LogDate
FROM   logs
WHERE   1 = 1
   AND   LogID NOT IN (
         SELECT      TOP 1000 LogID
         FROM      logs
         ORDER BY   LogDate
      )
```

This is pretty simple and effective. The nice thing to note here is that this is *not* a **correlated subquery**, so it should execute pretty fast.

This can easily be applied to a DELETE query:

```
DELETE
FROM   logs
WHERE   1 = 1
   AND   LogID NOT IN (
         SELECT      TOP 1000 LogID
         FROM      logs
         ORDER BY   LogDate
      )
```

EXCEPT Queries

While this one isn't really useful for UPDATE and DELETE queries, it is still a handy tool to have in your toolbox. I have found that most developers are familiar with UNION queries, but unfamiliar with its close cousins EXCEPT and INTERSECT.

Here is an example of a UNION query.

```
SELECT      LogID, LogDate
FROM      logs
WHERE      LogName = 'Alpha'
UNION
SELECT      LogID, LogDate
FROM      logs
WHERE      LogName = 'Bravo'
```

```
ORDER BY   LogID
```

A UNION query returns all rows that match either the query above the UNION statement of the query below it, so this would return a combination of results where LogName is "Alpha" and where it is "Bravo". In this case, it would clearly be better to use a single query with LogName IN ('Alpha','Bravo'), but the point is just to examine how these things work.

It is important to note that with UNION (and EXCEPT and INTERSECT) queries, the ORDER BY statement has to be at the end and applies to the entire result set. You can't apply it to just one side of the UNION operator. Also note that the column names and data types must match in both queries (though aliases can be used to achieve the column name match).

An EXCEPT query work much like a UNION query except that it returns all rows in the first excluding those that exist in the second query.

```
SELECT      LogID, LogDate
FROM      logs
WHERE      LogName IN ('Alpha','Bravo','Charlie')
EXCEPT
SELECT      LogID, LogDate
FROM      logs
WHERE      LogName = 'Bravo'
ORDER BY   LogID
```

This would return records only where LogName is "Alpha" or "Charlie". Again, a simple LogName IN ('Alpha','Charlie') would be superior for this particular example, but sometimes very complicated SQL (especially hitting different tables) can provide scenarios where this is advantageous.

Not EXISTS

Even though they often involve correlated subqueries which can potentially drag down performance a bit, I really love NOT EXISTS queries. They so often express just exactly what you mean and I love it when my SQL is accurately expressive of my intent. Fortunately, I have found that for most situations NOT EXISTS performs very well (your mileage may very, of course).

In this case, let's delete any users that haven't made any sales calls.

```
DELETE
FROM   users
WHERE   NOT EXISTS (
         SELECT   1
         FROM   salescalls
         WHERE   UserID = users.UserID
      )
```

Note that the "WHERE UserID = users.UserID" means that this is a correlated subquery which means that the subquery is getting executed for every single record returned by the outer query (every row in the users table, in this example). If that outer query will return a very high number of records (in this example, if the users table has a lot of records) then this could be slow. If the outer query is returning a relatively modest number of records, however, then the performance should be decent and the expressiveness of this syntax is very convenient. Someone readying the query later should know just want you intended - even if you forget to comment it.

This certainly isn't an exhaustive coverage of the topic, but I wanted to get these things down while I was thinking of them.