Surviving SQL

When you have a table with existing data, some SQL servers are better than others at changing the structure of that table while preserving the data. But, in general, the task is a pain. Third-party products may be available that will manage the task of issuing all the proper SQL statements to create the table with the new structure, copy the data from the old table to the new one, delete the original table, and rename the new table to the original name. If these products work for you, fine. But you may find them awkward or inadequate and be tempted to write your own. This is exactly what the developers at Ultimate Software Group did. With a project containing 200 plus tables and over 5,000 fields, you can see that the need to modify a table's structure may come up from time to time: a tool tailor-made for this job would be useful.

Originally, `TBatchMove` was used to copy the data from the old table to the new table. `TBatchMove` had an interesting side effect when a default was added to a column. When `TBatchMove` copies null data into a column with a default constraint, the null is recoded to the default value. Normally, SQL does not inherently convert null values to defaults. A column default is applied only when the column is omitted completely from an `INSERT` statement. If the column is present in the `INSERT` and a null value is assigned to it, then it gets a null value. So how is `TBatchMove` performing this magic?

Well, we've sort of answered our own question in the previous paragraph. `TBatchMove` generates a separate `INSERT` statement for each row in the table. When it detects a null value in a column, it omits that column from the `INSERT` statement. Figure 1 shows some sample data copied with `TBatchMove` and the generated `INSERT` statements. The result of this is that the row in the new table gets the column's default value, if any. If there is no default on the column, then it gets a null anyway.

Now the problem with `TBatchMove` is that it's slow. We can do the same thing with a single SQL statement that runs considerably faster. We can write an `INSERT` statement that accepts its values from a `SELECT` statement as shown below. This reads the rows from one table and inserts them into the new table all in a single operation.

```
INSERT INTO NewPeople (Name, Weight, Age)
  SELECT Name, Weight, Age FROM People
```

On a table of 86 columns and 6,700 rows, using `TBatchMove` to copy data took 4 minutes and 49 seconds. Copying the same data with `INSERT/SELECT` took... 7 seconds. The series of individual `INSERT` statements sent by `TBatchMove` was over 40 times slower. For this test the application was on the same machine as the SQL server, so we can't even blame network overhead for the `TBatchMove` to fetch each row from the server and send it back in an `INSERT` statement.

But how can we mimic `TBatchMove`'s ability to recode null values to a column default? Since there is only one SQL statement handling all the rows, we certainly can't selectively omit columns when the row happens to contain a null value. Fortunately, most dialects of SQL include a function called `Coalesce` which accepts any number of parameters and returns the first one it finds which is not null. So if we use `Coalesce` in the select list and pass in the column and the column's default value, what we get back is the column value if it is not null and the column default value if it is. Perfect. Figure 2 shows the `INSERT` statement and the resulting data.

`TBatchMove` still has some advantages. You can collect key violation and problem records in a separate file and continue processing, whereas the `INSERT/SELECT` will abort (and rollback everything) on the first problem. Also, `TBatchMove` can commit its work at intervals throughout the process. `INSERT/SELECT` does the entire job with a single transaction. On some systems this may mean a requirement for transaction log space on the disk at least as big as the table itself. Our practice has been to use the `INSERT/SELECT` method by default and switch to the slower `TBatchMove` if there are problems to troubleshoot.

Steve Troxell is a software engineer with Ultimate Software Group in the USA. He can be contacted via email at Steve_Troxell@USGroup.com

➤ *Figure 1*

```
Name                   Weight       Age
====================   ==========   ======
John                   165          27
Jane                   (null)       25
George                 180          (null)

INSERT INTO NewPeople (Name ,Weight ,Age )
  VALUES ('John', 165, 27)
INSERT INTO NewPeople (Name ,Age )
   VALUES ('Jane', 25)
INSERT INTO NewPeople (Name ,Weight )
  VALUES ('George', 180)
```

➤ *Figure 2*

```
INSERT INTO NewPeople (Name, Weight, Age)
  SELECT Name, Coalesce(Weight, 0), Coalesce(Age, 0)
    FROM People

Name                   Weight       Age
====================   ==========   ==========
John                   165          27
Jane                   0            25
George                 180          0
```