Data Paging with CTEs

The following is chapter 7 from my Common Table Expressions book in its entirety.

I can remember the first time that I worked on data paging code. I had to page through web site search results 20 at a time on a given web page. My task was to understand how it had been written and to do some bug fixing. After reviewing about 2,000 lines of code, and reviewing the seven different variables that were being used to know the current page, the next page, the previous page, the page size, the row at the top of the page, the row at the bottom of the page, and the number of pages, I finally understood what the code was intended to do. What a mess that was, but beyond the mess, the performance was horrible. The way the page worked was that based on the page you were on, all of the rows would be queried, then a loop would read through all of the rows before the current page, then loop through the rows on the current page displaying them on the page, and finally it would ignore the results after the current page. So page 1 was slow, page 2 was slower than page 1, page 3 was slower than page 2 and on and on.

Since that point I have implemented several different data paging algorithms myself, all better than the original implementation but none as elegant as the CTE way of doing data paging. I used to look at data paging as a painful task, but thanks to the SQL Server implementation of CTEs there is no more pain.

READER NOTE: *Please run the CTEBookSetup.sql script in order to follow along with the examples in this chapter. The script mentioned in this chapter is available at http://SteveStedman.com.*

Data Paging

The need for data paging comes up when working with a result set that is too large to display in the space available or in the time available. A familiar example may be an internet search engine. When a search is run on a search engine generally the first 20 results out of many more are displayed. For instance, when searching for “Steve Stedman” it displays the first 20 out of

1,120,000 results. When searching for “John” (with no last name) the search site still only displays the first 20 results, but it is out of 2.7 billion results.

Without the concept of data paging, these searches would have been limited to a fixed number of results, or the alternative of displaying all of the results. Imagine if that search engine tried to display 2.7 billion results. It may take a while for the page to load.

When using a search engine, the first 20 results are usually displayed, but if the result you want isn’t in the top 20 results, there is the ability to click through to the second or third page (or beyond). This concept of grabbing a fixed number of results and then jumping to page 2 or beyond is known as data paging.

For the example of building a data paging CTE, we will start with a query that returns more results than can be displayed on a single page as shown in the following example:

```
SELECT CustomerID,
       CustomerType,
       FirstName,
       LastName,
       CompanyName
  FROM Customer
 ORDER BY CustomerID ASC;
```

Editor | Results | Messages

	CustomerID	CustomerTy...	FirstName	LastName	CompanyName
762	762	Consumer	Paul	Hernandez	NULL
763	763	Consumer	Tessa	Anderson	NULL
764	764	Consumer	Sanjay	Collins	NULL
765	765	Consumer	Steven	Martinez	NULL
766	766	Consumer	Sandra	Lopez	NULL
767	767	Consumer	Jeff	Baker	NULL
768	768	Consumer	Michael	Anderson	NULL
769	769	Consumer	Anthony	Garcia	NULL
770	770	Consumer	Susan	Lewis	NULL
771	771	Consumer	Susan	Johnson	NULL
772	772	Consumer	Helen	Green	NULL
773	773	Consumer	Laura	Taylor	NULL
774	774	Consumer	Susan	Campbell	NULL
775	775	Consumer	Lisa	Nelson	NULL

Figure 7.1 Query with 775 rows of output.

In Figure 7.1 it may be feasible to display 775 results on a single page or it may just be a little too overwhelming. In order to make it more manageable we can use a CTE with data paging.

Data Paging CTE

The steps to create a data paging CTE are:

1. Create the original query that needs paging.
2. Using the ROW_NUMBER function with the OVER clause, add a new column into the original query to display a row number. If the original query had an ORDER BY clause specified, move the ORDER BY into the OVER clause.
3. Wrap the query from Step 2 with a CTE, and select everything from that CTE.
4. Add a WHERE clause to the outer query to filter the rows for the first page.
5. Remove the fixed values and replace them with variables to hold the page number and page size.

The reason that a CTE is needed to accomplish this is that the ROW_NUMBER function and OVER clause (known as the windowing functions) can't be used in a WHERE clause, but when they are returned from a CTE they can be used in the outer WHERE clause.

Step 1 - Create the original query that needs paging. We can just use the one from the previous example:

```
SELECT CustomerID,
       CustomerType,
       FirstName,
       LastName,
       CompanyName
  FROM Customer
 ORDER BY CustomerID;
```

Step 2 - Using the ROW_NUMBER function with the OVER clause, add a new column into the original query to display a row number. If the original query had an ORDER BY clause specified, move the ORDER BY into the OVER clause.

The OVER clause used this way is what is known as a windowing function in SQL Server.

```
SELECT CustomerID,
       CustomerType,
       FirstName,
       LastName,
       CompanyName,
       ROW_NUMBER() OVER (ORDER BY CustomerID) AS RowNum
  FROM Customer;
```

Editor | Results | Messages

	CustomerID	CustomerTy...	FirstName	LastName	CompanyName	RowNum
766	766	Consumer	Sandra	Lopez	NULL	766
767	767	Consumer	Jeff	Baker	NULL	767
768	768	Consumer	Michael	Anderson	NULL	768
769	769	Consumer	Anthony	Garcia	NULL	769
770	770	Consumer	Susan	Lewis	NULL	770
771	771	Consumer	Susan	Johnson	NULL	771
772	772	Consumer	Helen	Green	NULL	772
773	773	Consumer	Laura	Taylor	NULL	773
774	774	Consumer	Susan	Campbell	NULL	774
775	775	Consumer	Lisa	Nelson	NULL	775

Figure 7.2 Output from original query with RowNum calculated from the windowing function.

The common mistake at this point is to try and put the ROW_NUMBER with the windowing function into the WHERE clause, which isn't allowed as shown in the following code:

```
-- Don't do it this way, an error will be thrown
SELECT CustomerID,
      CustomerType,
      FirstName,
      LastName,
      CompanyName,
      ROW_NUMBER() OVER (ORDER BY CustomerID) AS RowNum
  FROM Customer
  WHERE ROW_NUMBER() OVER (ORDER BY CustomerID)
        BETWEEN 10 and 20;
```

Editor | Messages

```
Msg 4108, Level 15, State 1, Line 9
Windowed functions can only appear in the SELECT or ORDER BY clauses.
```

100 %

Figure 7.3 Error caused by attempting to use the OVER windowing function in the WHERE clause.

Instead of attempting to do the filtering, first the query needs to be wrapped in a CTE, which brings us to Step 3.

Step 3 - Wrap the query from Step 2 with a CTE, and select everything from that CTE.

```
;WITH CustomerPagingCTE AS
(
  SELECT CustomerID,
         CustomerType,
         FirstName,
         LastName,
         CompanyName,
         ROW_NUMBER() OVER (ORDER BY CustomerID) AS RowNum
    FROM Customer
)
SELECT *
  FROM CustomerPagingCTE;
```

Editor | Results | Messages

	CustomerID	CustomerTy...	FirstName	LastName	CompanyName	RowNum
766	766	Consumer	Sandra	Lopez	NULL	766
767	767	Consumer	Jeff	Baker	NULL	767
768	768	Consumer	Michael	Anderson	NULL	768
769	769	Consumer	Anthony	Garcia	NULL	769
770	770	Consumer	Susan	Lewis	NULL	770
771	771	Consumer	Susan	Johnson	NULL	771
772	772	Consumer	Helen	Green	NULL	772
773	773	Consumer	Laura	Taylor	NULL	773
774	774	Consumer	Susan	Campbell	NULL	774
775	775	Consumer	Lisa	Nelson	NULL	775

Figure 7.4 Output from the CTE version of the query.

Step 4 – Add a WHERE clause to the outer query to filter the rows for the first page.

In the following example, WHERE RowNum BETWEEN 0 and 10;has been added to filter the first page with a page size of 10.

```
;WITH CustomerPagingCTE AS
(
  SELECT CustomerID,
         CustomerType,
         FirstName,
         LastName,
         CompanyName,
         ROW_NUMBER() OVER (ORDER BY CustomerID) AS RowNum
    FROM Customer
)
SELECT *
  FROM CustomerPagingCTE
 WHERE RowNum BETWEEN 0 AND 10;
```

Editor | Results | Messages

	CustomerID	CustomerTy...	FirstName	LastName	CompanyName	RowNum
1	1	Consumer	Mark	Williams	NULL	1
2	2	Consumer	Lee	Young	NULL	2
3	3	Consumer	Patricia	Martin	NULL	3
4	4	Consumer	Mary	Lopez	NULL	4
5	5	Business	NULL	NULL	MoreTechnology.com	5
6	6	Consumer	Ruth	Clark	NULL	6
7	7	Consumer	Tessa	Wright	NULL	7
8	8	Consumer	Jennifer	Garcia	NULL	8
9	9	Consumer	Linda	Adams	NULL	9
10	10	Consumer	Robert	Wilson	NULL	10

Figure 7.5 Filter added to the query.

We can test the data paging at this point by adjusting the numbers in the WHERE clause and confirming that we get the correct results.

Step 5 - Remove the fixed values and replace them with variables to hold the page number and page size.

For this step we will be declaring 2 variables, @pageNum and @pageSize which will be used in the WHERE clause to calculate the row numbers that are to be shown on the current page. BETWEEN is used in the WHERE clause to calculate the row number of the first row on this page and to calculate the row number of the last row on this page.

```
DECLARE @pageNum AS INT;
DECLARE @pageSize AS INT;
SET @pageNum = 2;
SET @pageSize = 10;


;WITH CustomerPagingCTE AS
(
  SELECT CustomerID,
         CustomerType,
         FirstName,
         LastName,
         CompanyName,
         ROW_NUMBER() OVER (ORDER BY CustomerID) AS RowNum
    FROM Customer
)
SELECT *
  FROM CustomerPagingCTE
 WHERE RowNum BETWEEN (@pageNum - 1) * @pageSize + 1
```

```
AND @pageNum * @pageSize;
```

Editor | Results | Messages

	CustomerID	CustomerTy...	FirstName	LastName	CompanyName	RowNum
1	11	Consumer	Kimberly	Taylor	NULL	11
2	12	Consumer	Helen	Hernandez	NULL	12
3	13	Consumer	Linda	Hernandez	NULL	13
4	14	Consumer	Elizabeth	White	NULL	14
5	15	Consumer	Anthony	Scott	NULL	15
6	16	Consumer	Paul	Perez	NULL	16
7	17	Consumer	Sanjay	Mitchell	NULL	17
8	18	Consumer	Donald	Walker	NULL	18
9	19	Consumer	Patricia	Rodriguez	NULL	19
10	20	Consumer	Elizabeth	Wilson	NULL	20

Figure 7.6 Output from data paging page 2.

At this point all we need to do to get to page 3 is change the @pageNum variable from the previous code example.

```
SET @pageNum = 3;
```

Editor | Results | Messages

	CustomerID	CustomerTy...	FirstName	LastName	CompanyName	RowNum
1	21	Consumer	Carol	Thompson	NULL	21
2	22	Consumer	George	Hall	NULL	22
3	23	Consumer	Daniel	Gonzalez	NULL	23
4	24	Consumer	Paul	Edwards	NULL	24
5	25	Consumer	Lee	Hall	NULL	25
6	26	Consumer	Sandra	Williams	NULL	26
7	27	Consumer	George	Allen	NULL	27
8	28	Consumer	John	Allen	NULL	28
9	29	Consumer	Lisa	Scott	NULL	29
10	30	Consumer	Steven	Baker	NULL	30

Figure 7.7 Output from data paging page 3.

With these five easy steps a query which produces a large result set can be quickly converted into a data paging query. These steps will simplify most any other method of implementing data paging.

Offset and Fetch in SQL Server 2012

SQL Server has evolved over time and each new version adds new features designed to make queries easier and more powerful. SQL Server 2012 introduced the OFFSET and FETCH keywords which allow for data paging which is a bit easier than the CTE way of doing it, but is only available in SQL Server 2012 and later.

Offset and Fetch

One thing to consider with OFFSET and FETCH is that the query must have an ORDER BY clause. Hard coded or dynamic values can be used to specify rows to be returned. On the ROWS keyword the ‘S’ is optional and it can be entered as ROW or ROWS.

Let’s use the query from the beginning of the chapter and add a line to the end of the query specifying the OFFSET and number of rows to FETCH. The following example will return page 1 with a page size of 10.

```
SELECT CustomerID,
       CustomerType,
       FirstName,
       LastName,
       CompanyName
  FROM Customer
 ORDER BY CustomerID
OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;
```

Editor | Results | Messages

	CustomerID	CustomerTy...	FirstName	LastName	CompanyName
1	1	Consumer	Mark	Williams	NULL
2	2	Consumer	Lee	Young	NULL
3	3	Consumer	Patricia	Martin	NULL
4	4	Consumer	Mary	Lopez	NULL
5	5	Business	NULL	NULL	MoreTechnology.com
6	6	Consumer	Ruth	Clark	NULL
7	7	Consumer	Tessa	Wright	NULL
8	8	Consumer	Jennifer	Garcia	NULL
9	9	Consumer	Linda	Adams	NULL
10	10	Consumer	Robert	Wilson	NULL

Figure 7.8 Output from data paging using OFFSET and FETCH.

Next we take the OFFSET and FETCH version of our query and modify it to work similarly to the CTE version which allowed us to specify a page number and a page size.

```
DECLARE @pageNum AS INT;
DECLARE @pageSize AS INT;
SET @pageNum = 3;
SET @pageSize = 10;


SELECT CustomerID,
       CustomerType,
       FirstName,
       LastName,
       CompanyName
  FROM Customer
 ORDER BY CustomerID
OFFSET (@pageNum - 1) * @pageSize ROWS
 FETCH NEXT @pageSize ROWS ONLY;
```

Editor | Results | Messages

	CustomerID	CustomerTy...	FirstName	LastName	CompanyName
1	21	Consumer	Carol	Thompson	NULL
2	22	Consumer	George	Hall	NULL
3	23	Consumer	Daniel	Gonzalez	NULL
4	24	Consumer	Paul	Edwards	NULL
5	25	Consumer	Lee	Hall	NULL
6	26	Consumer	Sandra	Williams	NULL
7	27	Consumer	George	Allen	NULL
8	28	Consumer	John	Allen	NULL
9	29	Consumer	Lisa	Scott	NULL
10	30	Consumer	Steven	Baker	NULL

Figure 7.9 Page 3 of paging using OFFSET and FETCH.

In the previous examples we see that the OFFSET and FETCH method of doing data paging may be easier to use than the CTE method. One day this may completely replace the need to use the CTE data paging.

Summary

CTEs give a way do to data paging on queries with large result sets that would otherwise not usually be feasible in T-SQL.

With 5 steps it is possible to quickly and easily add data paging into an existing query.

1. Create the original query that needs paging.
2. Using the ROW_NUMBER function with the OVER clause, add a new column into the original query to display a row number. If the original query had an ORDER BY clause specified, move the ORDER BY into the OVER clause.
3. Wrap the query from Step 2 with a CTE, and select everything from that CTE.
4. Add a WHERE clause to the outer query to filter the rows for the first page.
5. Remove the fixed values and replace them with variables to hold the page number and page size.

The use of the windowing functions with the ROW_NUMBER function and the OVER clause allow the assignment of a row identifier that can be used to filter out only the rows for a given page.

SQL Server 2012 introduced OFFSET and FETCH to replace the need to use CTEs for data paging. The OFFSET and FETCH method may be a bit easier and cleaner to write in T-SQL, but it is only available in SQL Server 2012 and newer.

Either the CTE data paging, or the OFFSET and FETCH method of data paging are cleaner and more efficient than trying to do data paging in application level code.