

使用ROW_NUMBER 函数分页

```
SELECT *
FROM ( SELECT ROW_NUMBER() OVER ( ORDER BY dbo.Products.ProductID DESC )
AS rownum ,      FROM      dbo.Products ) AS temp
WHERE temp.rownum BETWEEN 1 AND 10
```

<http://www.cnblogs.com/vangecnu/p/3702975.html>

<http://jingyan.baidu.com/article/9989c74604a644f648ecfef3.html>

<http://www.cnblogs.com/qqlin/archive/2012/11/01/2745161.html>

数据库分页是老生常谈的问题了。如果使用ORM框架，再使用LINQ的话，一个Skip和Take就可以搞定。但是有时由于限制，需要使用存储过程来实现。在SQLServer中使用存储过程实现分页的已经有很多方法了。之前在面试中遇到过这一问题，问如何高效实现数据库分页。刚好上周在业务中也遇到了这个需求，所以在这里简单记录和分享一下。

一 需求

这里以SQLServer的示例数据库NorthWind为例，里面有一张Product表，现在假设我们的需求是要以UnitPrice降序排列，并且分页，每一页10条记录。要求服务端分页。参数为每页记录数和页码。

二 实现

Top分页

当时采用的最直接做法就是使用两个Top来实现，最后返回的结果是升序的，在C#代码里再处理一下就可以了。这里作为演示，语句中使用*为了方便，实际开发中要替换为具体的列名。下面的方法简单吧。

```
SELECT TOP (@pageSize)
*
FROM ( SELECT TOP ( @pageSize * @pageIndex )
*
FROM [Northwind].[dbo].[Products]
ORDER BY UnitPrice DESC
) AS product
ORDER BY product.UnitPrice
```

但是这个代码是有问题的，不知道各位发现了没有。当符合条件的纪录集小于每页记录数时，没有问题，但是当大于就有问题了，比如，在实例数据库中Products中有77条记录，当每页10条记录，第8页只应该返回7条记录，第9页应该返回空，但是使用如上的方法，每次都会返回10条记录。

沿用上面的思路，把代码修改为了如下采用三层Select，最内一层查询所有记录之前的数据，然后第二层选择Top PageSize个所有NOT IN 第一层数据中的数据即可，因为使用了NOT IN所以不存在第一种方法中的bug

```
SELECT *
FROM dbo.Products
WHERE ProductID IN (
    SELECT TOP ( @pageSize )
        ProductID
    FROM dbo.Products
    WHERE ProductID NOT IN ( SELECT TOP ( @pageSize *
(@pageIndex-1) )
        ProductID
    FROM dbo.Products
    ORDER BY UnitPrice DESC )
    ORDER BY dbo.Products.UnitPrice DESC )
ORDER BY dbo.Products.UnitPrice ASC
```

使用ROW_NUMBER 函数分页

其实还有一种最简单最直接的思路，那就是采用临时表，即在内存中创建一个表变量，该变量中包含一个自增列，表关键字列，然后将待排序的表按照排序条件和规则插入到这张表中，然后就可以将自增列作为行号使用了，在比较早的如SQLServer 2000中，只能这样做，但是对于大数据量的记录集，需要创建的临时表也比较大，效率比较低，这里就不介绍了。

在SQLServer2005中引入了[ROW_NUMBER\(\)](#) 函数,通过这个函数，可以根据给定好的排序字段规则，生成记录序号，其基本用法为：

```
SELECT ROW_NUMBER() OVER ( ORDER BY dbo.Products.ProductID DESC ) AS
rownum ,
*
FROM dbo.Products
```

这样，结果集中第一列就为 rownum，从1开始按步长为1递增，这有点类似从1开始步长为1的自增字段。这里需要提一下的是，这个语句中赋值的rownum列不能使用在当前的where语句中，也不可以把整个ROW_NUMBER()语句放到where中作为条件，下面两种使用方式都是错误的。

```
SELECT ROW_NUMBER() OVER ( ORDER BY dbo.Products.ProductID DESC ) AS
rownum ,
*
FROM dbo.Products
WHERE rownum BETWEEN 1 AND 10
```

会提示错误：

```
Invalid column name 'rownum'.
```

```
SELECT ROW_NUMBER() OVER ( ORDER BY dbo.Products.ProductID DESC ) AS
rownum ,
*
FROM dbo.Products
WHERE ( ROW_NUMBER() OVER (ORDER BY City) AS rown ) BETWEEN 1 AND 10
```

会提示错误：

```
Incorrect syntax near the keyword 'AS'.
```

正确的做法是，把查询的结果作为一个内查询，再在外面套上一个外查询语句：

```
SELECT *
FROM ( SELECT ROW_NUMBER() OVER ( ORDER BY dbo.Products.ProductID
DESC ) AS rownum ,
*
FROM dbo.Products
) AS temp
WHERE temp.rownum BETWEEN 1 AND 10
```

有了以上基础之后，我们就可以利用ROW_NUMBER这个特性来进行排序了。

```
SELECT *
FROM ( SELECT TOP ( @pageSize * @pageIndex )
ROW_NUMBER() OVER ( ORDER BY dbo.Products.UnitPrice
DESC ) AS rownum ,
*
FROM dbo.Products
) AS temp
WHERE temp.rownum > ( @pageSize * ( @pageIndex - 1 ) )
ORDER BY temp.UnitPrice
```

策略很简单，首先我们选取包含要查页的数据，然后使用ROW_NUMBER函数进行编号，然后在外查询中指定rownum大于页起始记录即可。这种方式简单快捷。

这里还有一种[使用CTE的方式](#) (common_table_expression，公用表表达式，不是CTE四六级哦，我第一次接触到这个是在面试的时候被问到如何使用SQL编写递归，呵呵)，使用很简单，就是把内查询放在CTE 里面，如下：

```
WITH ProductEntity
AS ( SELECT TOP ( @pageSize * @pageIndex )
ROW_NUMBER() OVER ( ORDER BY
dbo.Products.UnitPrice DESC ) AS rownum ,
*
FROM dbo.Products
)
SELECT *
FROM ProductEntity
WHERE ProductEntity.rownum > ( @pageSize * ( @pageIndex - 1 ) )
ORDER BY ProductEntity.UnitPrice
```

这种性能和上面的类似。但是在某些情况下，使用CTE会比直接采用外接查询具有更好的效率。例如，我们可以仅使用CTE来存储行号，关键字以及排序字段，然后用来和原表做join查询，如下：

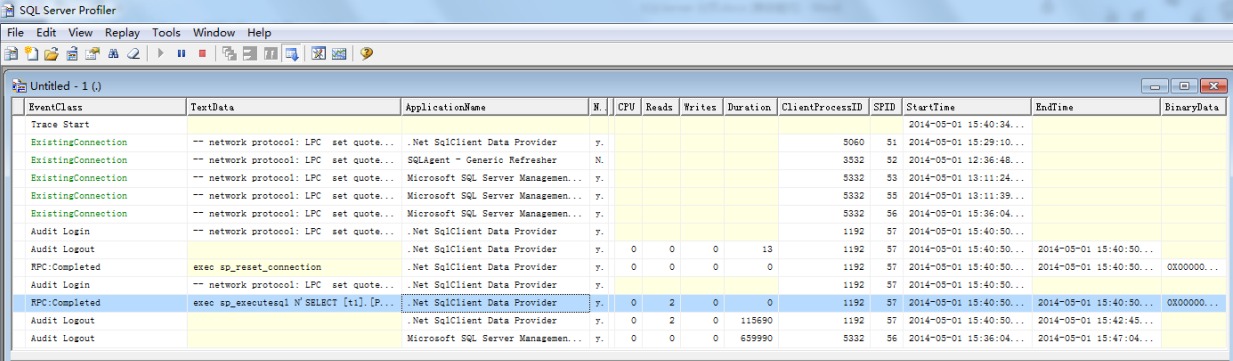
```
WITH ProductEntity
AS ( SELECT TOP ( @pageSize * @pageIndex )
ROW_NUMBER() OVER ( ORDER BY
dbo.Products.UnitPrice DESC ) AS rownum ,
ProductID , --主键，
UnitPrice--待排序字段
FROM dbo.Products
)
SELECT *
FROM ProductEntity
INNER JOIN dbo.Products ON dbo.Products.ProductID =
ProductEntity.ProductID
WHERE ProductEntity.rownum > ( @pageSize * ( @pageIndex - 1 ) )
ORDER BY ProductEntity.UnitPrice
```

使用ROW_NUMBER来进行分页是一种使用很广的分页方式，在本文开头讲到在LINQ中可以采用的TAKE 和 SKIP语句，但是与数据库交互只能使用SQL语句，LINQ在内部会帮我们转化为合适的SQL语句，语句里面其实也是采用ROW_NUMBER这一函数，为了演示，我们新建一个Console程序，然后在里面添加一个LINQ To SQL的类，使用方法非常简单，如下：

```
List<Product> product;
int pageSize = 10;
int pageIndex = 8;
using (ProductsDataContext context = new ProductsDataContext())
{
    product = context.Products.OrderByDescending(x => x.UnitPrice) //排序
                                .Skip(pageSize * (pageIndex-1)) //跳过前面的记录
                                .Take(pageSize) //选取每一页个数
                                .ToList();
}
```

寥寥几句就实现了分页。

我们知道LINQ其实是将C#表达式树转换成了SQL语言，通过SQLServer Profile 工具，我们可以看到程序发送给SQLServer的请求，如下：



EventClass	TextData	ApplicationName	SPID	StartTime	EndTime	BinaryData
Trace Start				2014-05-01 15:40:34...		
ExistingConnection	-- network protocol: LPC set quote...	.Net SqlClient Data Provider	5060	2014-05-01 15:29:10...		
ExistingConnection	-- network protocol: LPC set quote...	SQLAgent - Generic Refresher	3532	2014-05-01 12:36:48...		
ExistingConnection	-- network protocol: LPC set quote...	Microsoft SQL Server Managemen...	5332	2014-05-01 13:11:24...		
ExistingConnection	-- network protocol: LPC set quote...	Microsoft SQL Server Managemen...	5332	2014-05-01 13:11:39...		
ExistingConnection	-- network protocol: LPC set quote...	Microsoft SQL Server Managemen...	5332	2014-05-01 15:36:04...		
Audit Login	-- network protocol: LPC set quote...	.Net SqlClient Data Provider	1192	2014-05-01 15:40:50...		
Audit Logout		.Net SqlClient Data Provider	1192	2014-05-01 15:40:50...	2014-05-01 15:40:50...	
RPC:Completed	exec sp_reset_connection	.Net SqlClient Data Provider	1192	2014-05-01 15:40:50...	2014-05-01 15:40:50...	0X000000...
Audit Login	-- network protocol: LPC set quote...	.Net SqlClient Data Provider	1192	2014-05-01 15:40:50...		
RPC:Completed	exec sp_executesql N'SELECT [t1].[P...	.Net SqlClient Data Provider	1192	2014-05-01 15:40:50...	2014-05-01 15:40:50...	0X000000...
Audit Logout		.Net SqlClient Data Provider	1192	2014-05-01 15:40:50...	2014-05-01 15:42:45...	
Audit Logout		Microsoft SQL Server Managemen...	5332	2014-05-01 15:36:04...	2014-05-01 15:47:04...	

```
exec sp_executesql N'SELECT [t1].[ProductID], [t1].[ProductName], [t1].[SupplierID], [t1].[CategoryID], [t1].[QuantityPerUnit], [t1].[UnitPrice], [t1].[UnitsInStock], [t1].[UnitsOnOrder], [t1].[ReorderLevel], [t1].[Discontinued]
FROM
    SELECT ROW_NUMBER() OVER (ORDER BY [t0].[UnitPrice] DESC) AS [ROW_NUMBER], [t0].[ProductID], [t0].[ProductName], [t0].[SupplierID], [t0].[CategoryID], [t0].[QuantityPerUnit], [t0].[UnitPrice], [t0].[UnitsInStock], [t0].[UnitsOnOrder], [t0].[ReorderLevel], [t0].[Discontinued]
FROM [dbo].[Products] AS [t0]
    AS [t1]
WHERE [t1].[ROW_NUMBER] BETWEEN @p0 + 1 AND @p0 + @p1
ORDER BY [t1].[ROW_NUMBER]', N'@p0 int, @p1 int', @p0 = 70, @p1 = 10
```

Translate LINQ to SQL

我把下面的语句拷贝出来，可以看到

```
EXEC sp_executesql N'SELECT [t1].[ProductID], [t1].[ProductName], [t1].[SupplierID], [t1].[CategoryID], [t1].[QuantityPerUnit], [t1].[UnitPrice], [t1].[UnitsInStock], [t1].[UnitsOnOrder], [t1].[ReorderLevel], [t1].[Discontinued]
FROM (
    SELECT ROW_NUMBER() OVER (ORDER BY [t0].[UnitPrice] DESC) AS [ROW_NUMBER], [t0].[ProductID], [t0].[ProductName], [t0].[SupplierID], [t0].[CategoryID], [t0].[QuantityPerUnit], [t0].[UnitPrice], [t0].[UnitsInStock], [t0].[UnitsOnOrder], [t0].[ReorderLevel], [t0].[Discontinued]
FROM [dbo].[Products] AS [t0]
    ) AS [t1]
WHERE [t1].[ROW_NUMBER] BETWEEN @p0 + 1 AND @p0 + @p1
ORDER BY [t1].[ROW_NUMBER]', N'@p0 int, @p1 int', @p0 = 70, @p1 = 10
```

这正是我们之前手写的采用ROW_NUMBER的分页程序。可见，简简单单的一句SKIP和TAKE，LINQ在后面帮我们做了很多工作。

使用OFFSET FETCH子句分页

既然LINQ这么简单的搞定了分页，那么SQL Server中有没有类似的简单的语句就能搞定分页了，答案是有的，那就是SQL Server Compact 4.0中引入的[OFFSET FETCH](#)子句。

```
SELECT *  
FROM    dbo.Products  
ORDER   BY UnitPrice DESC  
OFFSET  ( @pageSize * ( @pageIndex - 1 ) ) ROWS  
FETCH NEXT @pageSize ROWS ONLY;
```

是不是和LINQ很像，OFFSET相当于SKIP，FETCH NEXT相当于TAKE。

可以在官网下载[SQL Server CE 4.0](#)，目前仅支持SQL Server 2012及SQL Server 2014，不过可以使用[Microsoft Webmatrix](#)这个[工具](#)来用这一新功能。

比较

在讨论性能之前，首先需要明确的是，我们在编写SQL语句的时候，尽量要减少不必要字段的输出，文中出于演示，所以都用的*，在实际中不要这样。还有就是要根据业务逻辑，比如查询条件，建立合适的聚合索引和非聚合索引，索引对于查找的效率影响非常大，SQL中的索引其实就是建立某种平衡查找树，如B树来进行，这方面的知识可以看我之前写的算法中的文章，再有就是了解一下SQL Server的一些特性比如CTE，IN 和Exist的区别等等，有些小的地方对性能可能有一定的影响。

在上面这些处理好了之后，我们现在来讨论那种分页方案更好。

- 采用Top - Not In - Top方案比较复杂，里面包含了in语句，效率不高，但是兼容个版本的SQL Server。
- 采用ROW_Number方法实现分页难易适中，效率较高。LINQ中的SKIP和TAKE也是采用这种方式来进行分页的，应该是目前采用的比较广泛的分页方式。
- OFFSET FETCH 方法是[SQL Server CE 4.0](#) 中才引入的，由于本文没有SQL Server 2012 以及测试数据，从 [comparing-performance-for-different-sql-server-paging-methods](#)这篇文章来以及园子里的[Sql Server 2012 分页方法分析\(offset and fetch\)](#)，性能应该比较好的。

以上是对SQL Server数据库SQL分页的一点总结，希望对您有所帮助。