MS SQL 2011 (Denali) – Offset

Offset и Fetch First\Next выражения – расширения команды Order By (NEW)

В новом SQL Server 2011 (Denali) расширяются возможности команды Order By с помощью двух долгожданных дополнительных команд:

- Offset (смещение)
- Fetch First или Fetch Next (**взять первые…** или **взять следующие…**)

Offset

Использование данной команды позволяет пропустить указанное количество строк перед тем как выводить результаты запроса. Что под этим подразумевается: Допустим, у нас есть 100 записей в таблице и нужно пропустить первые 10 строк и вывести строки с 11 по 100. Теперь это легко решается следующим запросом:

```
Select *
From  <SomeTable>
Order by  <SomeColumn>
Offset 10 Rows
```

Для тех товарищей, которые практикуют .Net должен быть знаком метод расширения для коллекций **Skip**, который пропускает указанное количество строк. Так вот выражение **Offset** работает точно так же. После того как данные упорядочены каким-либо образом, можно применять выражение Offset.

Ситуации, в которых может быть использовано выражение Offset

Во всех последующих примерах на Offset будет использовать набор данных построенных в результате данного скрипта:

```
-- Declare a table variable
Declare @tblSample Table
(
        [Person Name] Varchar(50)
        ,Age int
        ,Address Varchar(100)
)

-- Populate some data to the table
Insert into @tblSample
Select
                'Person Name' + CAST(Number AS VARCHAR)
                , Number
                , 'Address' + CAST(Number AS VARCHAR)
From master..spt_values
Where Type = 'p'
And Number Between 1 and 50
```

Задача 1. Пропустить первые 10 записей и показать остальные.

Скрипт будет простой.

```
Select *
```

```
From @tblSample
Order by Age
Offset 10 Row
```

Или

```
Select *
From @tblSample
Order by Age
Offset (10) Rows
```

Вывод результатов будет таким:

```
Person Name      Age      Address
Person Name11    11       Address11
Person Name12    12       Address12
 . . . . .  . . . . . . . . . .
 . . . . . . . . . . . . . . . .
Person Name49    49       Address49
Person Name50    50       Address50
```

Неважно, какое слово использовать после указания количества строк: **Row** или **Rows** – они синонимы в данном случае.

Задача 2. Передать количество строк для пропуска в виде переменной

```
-- Объявляем переменную в которой будет содержаться кол-во строк для пропуска
Declare @RowSkip As int

-- Выставляем количество строк для пропуска
 Set @RowSkip = 10

-- получаем результат
Select *
From @tblSample
Order by Age
Offset @RowSkip Row
```

Задача 3. Задать количество строк для пропуска в виде выражения

```
-- получить строки с 14 по 50
Select *
From @tblSample
Order by Age
Offset (select MAX(number)/99999999 from  master..spt_values) Rows
```

Выражение *select MAX(number)/99999999 from master..spt_values* вернет число 14.

Задача 4. Задать количество строк для пропуска в виде пользовательской функции

```
Select *
From @tblSample
Order by Age
Offset (select dbo.fn_test()) Rows
```

Код для скалярной пользовательской функции

```
CREATE FUNCTION fn_test()
```

```
RETURNS int
AS
BEGIN
        -- Declare the return variable
        Declare @ResultVar as int
        -- Enter some value to the return variable
        Select @ResultVar = 10
        -- Return the result of the function
        RETURN @ResultVar
END
GO
```

Задача 5. Использование Offset с Order by внутри представлений (view), функций, подзапросах, вложенных таблицах, общих выражениях для таблиц (Common Table Expressions - CTE).

Например, использование в общих выражениях.

```
;With Cte As
(
        Select *
        From @tblSample
        Order By Age
        Offset 10 Rows)

Select *
From Cte
```

Пример ниже показывает использование Offset и Order by внутри вложенной таблицы.

```
Select *
From
        (Select *
        From @tblSample
        Where Age >10
        Order By Age
        Offset 10 Rows) As PersonDerivedTable
```

И еще пример на работу Offset и Order с представлениями.

- *Создание view*

```
Create View vwPersonRecord AS
Select * FROM tblSample
GO

-- выборка данных из view
Select *
From vwPersonRecord
Where Age > 10
Order By Age
Offset 10 Rows
```

Когда Offset не будет работать

1. Так как это «метод расширения», то без выражения order by ничего работать не будет.

```
Select *
From @tblSample
Offset (10) Rows
```

Сообщит об ошибке

Msg 102, Level 15, State 1, Line 21 Incorrect syntax near '10'.

2. Нельзя задавать отрицательное значение для Offset.

```
Select *
From @tblSample
Order by Age
Offset (-10) Rows
```

Движок SQL сервера выдаст

Msg 10742, Level 15, State 1, Line 22 The offset specified in a OFFSET clause may not be negative.

3. Нельзя задавать значения отличные от целочисленного типа .

```
Select *
From @tblSample
Order by Age
Offset 10.5 Rows
```

или

```
Select *
From @tblSample
Order by Age
Offset Null Rows
```

Выдаст нам

Msg 10743, Level 15, State 1, Line 24 The number of rows provided for a OFFSET clause must be an integer.

4. Не может быть использован внутри выражения **Over()**.

```
;With Cte As
(
        Select
                        *,
                        Rn = Row_Number() Over(Order by Age Offset 10 Rows)
        From @tblSample
)

Select * from Cte
```

Во время выполнения запроса получим сообщение

Msg 102, Level 15, State 1, Line 22 Incorrect syntax near 'Offset'.

Использование Fetch First / Fetch Next

Эти ключевые слова используются для уточнения количества возвращаемых строк после пропуска массива строк по выражению Offset. Представьте, что у нас есть 100 строк и нам надо пропустить первые 10 и получить следующие 5 строк. Т.е. надо получить строки с 11 по 15.

```
Select *
From  <SomeTable>
Order by  <SomeColumn>
Offset 10 Rows
Fetch Next 5 Rows Only; -- или Fetch First 5 Rows Only
```

Такой запрос вернет ожидаемое кол-во строк. Программисты на .Net тут же припомнят метод расширения Take.

Далее рассмотрим ситуации, где можно применить эти ключевые слова.

Задача 1. Пропустить первые 10 записей и получить следующие 5

```
Select *
From @tblSample
Order by Age
Offset 10 Row
Fetch First 5 Rows Only
```

Результат будет таким:

```
Person Name      Age      Address
Person Name11    11       Address11
Person Name12    12       Address12
Person Name13    13       Address13
Person Name14    14       Address14
Person Name15    15       Address15
```

Задача 2. Задать количество строк для вывода с помощью переменной

```
-- переменная для указания смещения
Declare @RowSkip As int
-- переменная для указания кол-ва возвращаемых строк
Declare @RowFetch As int

-- кол-во строк для пропуска
 Set @RowSkip = 10
-- кол-во строк для возврата
 Set @RowFetch = 5

-- вывод строк с 11 по 15
Select *
From @tblSample
Order by Age
Offset @RowSkip Row
Fetch  Next @RowFetch Rows Only;
```

В целом и общем, с этими ключевыми словами можно делать все то же самое, что и с Offset. Подзапросы, представления, функции и т.д.

Когда Fetch First / Fetch Next не будут работать

Ограничения на эти ключевые слова полностью совпадают с ограничениями на Offset.

Симуляция Offset и Fetch Next для Sql Server 2005/2008

В предыдущих версиях SQL сервера можно было получить тот же функционал путем применения функции ранжирования Row_Number(). Конечно код получался не такой изящный и лаконичный, например:

```
-- Переменная для указания строк смещения
Declare @RowSkip As int
-- Переменная для указания кол-ва строк для возврата
Declare @RowFetch As int

-- Задание переменных
 Set @RowSkip = 10
```

```
Set @RowFetch = 5

 ;With Cte As
 (
        Select
                rn=ROW_NUMBER()
                Over(Order by (Select 1) /* генерируем служебную колонку */ )
                ,*
        From @tblSample
 )
-- забираем записи с 11 по 15
Select
        [Person Name]
        ,Age
        ,Address
From Cte
-- симуляция поведения Offset и Fetch First/Fetch Next

Where rn Between  (@RowSkip+1) -- симуляция Offset
                And  (@RowSkip+ @RowFetch) -- симуляция Fetch First/Fetch Next Clause
```

Внутри CTE идет генерация служебной колонки которая просто нумерует строки, после чего строки фильтруются по этому полю. Способ не самый быстрый как вы понимаете.

Симуляция Offset и Fetch Next для Sql Server 2000

Для этих древних серверов не было функций ранжирования, но и тогда можно было повторить обсуждаемый функционал. Тогда в ход шли временные таблицы с авто инкрементальным полем. Пример скрипта:

```
Declare @RowSkip As int
Declare @RowFetch As int

Set @RowSkip = 10
Set @RowFetch = 5

--если временная таблица существует, то удалить ее
IF OBJECT_ID('tempdb..#Temp') IS NOT NULL
BEGIN
Drop Table #Temp
END

--создание временной таблицы
Create Table #Temp
(
        Rn int Identity
        ,[Person Name] Varchar(50)
        ,Age int
        ,Address Varchar(100)
)

-- заполнение временной таблицы
Insert Into #Temp([Person Name],Age,Address)
Select  [Person Name],Age,Address
From @tblSample

-- получение строк с 11 по 15
Select
        [Person Name]
        ,Age
        ,Address
From #Temp

-- симуляция поведения Offset и Fetch First/Fetch Next
Where Rn Between  (@RowSkip+1) -- симуляция Offset
                And  (@RowSkip+ @RowFetch) -- симуляция Fetch First/Fetch Next
```

В этом скрипте сначала создается временная таблица, куда перезаписываются данные из целевой таблицы. Причем во временной таблице есть автоинкрементальное поле, по которому потом и осуществляется запрос нужных строк.

Практическое применение Offset и Fetch с замерами времени и ресурсов.

Я уверен, что всё предыдущее объяснение об использовании и назначении Offset и Fetch подвело вас к ясному пониманию, зачем они нужны и где их можно использовать. Родились идеи по оптимизации существующего кода. Далее мы рассмотрим пример из реальной практики, когда может пригодиться Offset. Так же будут приведены результаты замеров производительности на разных SQL серверах. Тесты будут прогоняться на выборке из 1 миллиона строк.

Для начала создадим счет-таблицу по следующему скрипту.

```
-- удалить таблицу tblSample, если она существует
IF OBJECT_ID('tblSample','U') IS NOT NULL BEGIN
        DROP TABLE tblSample
END
GO

-- создать таблицу
Create Table tblSample (
        [Person ID]      Int Identity
        ,[Person Name] Varchar(100)
        ,Age Int
        ,DOB Datetime
        ,Address Varchar(100)
)
GO

-- заполнить таблицу миллионом записей
Insert into tblSample
Select
                'Person Name' + CAST(N AS VARCHAR)
                , N
                ,DATEADD(D,N, '1900-01-01')
                ,'Address' + CAST(N AS VARCHAR)
From dbo.tsqlc_Tally
Where  N Between 1 and 1000000

-- вывести данные
Select *
From tblSample
```

Постраничный просмотр данных на стороне сервера

Постраничный просмотр является наиболее часто встречающейся функцией в системах просмотра записей из каких-либо баз. Теперь это возможно проделывать как на стороне клиента, так и на стороне сервера. Пэйджинг на стороне клиента подразумевает загрузку всей таблицы или же очень большой ее части в память, с тем, чтобы программными средствами делать постраничный просмотр. С другой стороны это может быть произведено на стороне сервера, тогда приложе ние получит только те данные, которые оно запросило для отображения нужной страницы. При таком подходе сокращается время на пересылку данных, постобработку и хранение их в памяти. Т.е. происходит существенное ускорение производительности приложения.

В целях эксперимента мы пропустим первые 20 000 записей и возьмем следующие 50 000.

Подход для SQL Server 2000

```
-- сброс буфера и кэша статистики
```

```
DBCC DROPCLEANBUFFERS
DBCC FREEPROCCACHE

USE TSQLDB;
GO
SET STATISTICS IO ON;
SET STATISTICS TIME ON;
GO

-- Переменные для пэйджинга
Declare @RowSkip As int
Declare @RowFetch As int

-- Установка значений постраничного просмотра
 Set @RowSkip = 20000
Set @RowFetch = 50000

--Удаление временной таблицы, если она есть
IF OBJECT_ID('tempdb..#Temp') IS NOT NULL
BEGIN
Drop Table #Temp
END

-- создание временной таблицы
Create Table #Temp
(
        Rn int Identity
        ,[Person ID] int
        ,[Person Name] Varchar(50)
        ,Age int
        ,DOB datetime
        ,Address Varchar(100)
)

-- Занесение данных во временную таблицу
Insert Into #Temp([Person ID],[Person Name],Age,DOB,Address)
Select  [Person ID],[Person Name],Age,DOB,Address
From dbo.tblSample

-- выборка данных с 20 000 по 70 000
Select
        [Person ID]
        ,[Person Name]
        ,Age
        ,DOB
        ,Address
From #Temp

-- симуляция поведения Offset и Fetch First/Fetch Next
Where Rn Between  (@RowSkip+1) -- симуляция Offset
                And  (@RowSkip+ @RowFetch) -- симуляция Fetch First/Fetch Next

GO
SET STATISTICS IO OFF;
SET STATISTICS TIME OFF;
GO
```

Я думаю что предыдущих примеров и комментариев хватает, чтобы понять работу скрипта.

Время выполнения:

```
SQL Server Execution Times:
 CPU time = 110 ms,  elapsed time = 839 ms.
```

Статистика ввода\вывода:

```
Scan count 1,
logical reads 8037,
physical reads 0,
read-ahead reads 0,
lob logical reads 0,
lob physical reads 0,
lob read-ahead reads 0.
```

Подход для SQL Server 2005/2008

```
DBCC DROPCLEANBUFFERS
DBCC FREEPROCCACHE

USE TSQLDB;
GO
SET STATISTICS IO ON;
SET STATISTICS TIME ON;
GO

Declare @RowSkip As int
Declare @RowFetch As int

Set @RowSkip = 20000
Set @RowFetch = 50000

;With Cte As
 (
        Select
                rn=ROW_NUMBER()
                Over(Order by (Select 1))
                ,*
        From dbo.tblSample
 )

Select
        [Person ID]
        ,[Person Name]
        ,Age
        ,DOB
        ,Address
From Cte

Where rn Between  (@RowSkip+1)
                And  (@RowSkip+ @RowFetch)

GO
SET STATISTICS IO OFF;
SET STATISTICS TIME OFF;
GO
```

Время выполнения:

```
SQL Server Execution Times:
   CPU time = 78 ms,  elapsed time = 631 ms.
```

Статистика ввода\вывода:

```
Scan count 1,
logical reads 530,
physical reads 0,
read-ahead reads 1549,
lob logical reads 0,
lob physical reads 0,
lob read-ahead reads 0.
```

Подход для SQL Server 2011

```
DBCC DROPCLEANBUFFERS
DBCC FREEPROCCACHE

USE TSQLDB;
GO
SET STATISTICS IO ON;
SET STATISTICS TIME ON;
GO

Declare @RowSkip As int
Declare @RowFetch As int

Set @RowSkip = 20000
Set @RowFetch = 50000

Select *
From dbo.tblSample
Order by (Select 1)
Offset @RowSkip Row
Fetch  Next @RowFetch Rows Only;

GO
SET STATISTICS IO OFF;
SET STATISTICS TIME OFF;
GO
```

Время выполнения:

```
SQL Server Execution Times:
   CPU time = 47 ms,  elapsed time = 626 ms.
```

Статистика ввода\вывода:

```
Scan count 1,
logical reads 530,
physical reads 0,
read-ahead reads 1549,
lob logical reads 0,
lob physical reads 0,
lob read-ahead reads 0.
```

Наиболее интересен результат по использованию процессорного времени (CPU Time) и время выполнения (Elapsed Time - время потребовавшееся запросу на выполнение). Сравнение замеров представлено ниже:

Sql Server Version	CPU Time	Elapsed Time
2000	110ms	839 ms
2005/2008	78ms	631 ms
2011	46ms	626 ms

В таблице наглядно представлено, что новый SQL Server работает заметно быстрее по сравнению с предыдущими версиями. Естественно, что для вашей машины замеры времени могут отличаться, но производительность нового сервера будет всегда выше.

Альтернатива выражению TOP.

Новые возможности Denali в некоторых ситуациях могут стать заменой выражению TOP. Для примера возьмем ситуацию, когда необходимо получить первые 10 записей отсортированные по убыванию какого-либо параметра.

Подходы на предыдущих версиях

```
Select Top(10)
         [Person ID]
         ,[Person Name]
         ,Age
         ,DOB
         ,Address
From dbo.tblSample
Order By Age Desc
```

Подход возможный в SQL Server Denali

```
Select
         [Person ID]
         ,[Person Name]
         ,Age
         ,DOB
         ,Address
From dbo.tblSample
Order By Age Desc
Offset 10 Rows
```

Дальше будет интереснее, так что оставайтесь на связи. Перевод.

Hard'n'heavy!

Violet Tape