

3.2.2. Индексы.

Индексы являются одним из важнейших механизмов повышения производительности запросов. Они создаются на основе определенных столбцов таблицы и позволяют получить быстрый доступ к строкам данных на основе значений в этих столбах, без перебора всех записей, поэтому так важно проиндексировать таблицы хранилища для оптимизации запросов.

Все данные в SQL Server хранятся на страницах по 8 Кб. Если таблица не имеет индексов, то её записи не упорядочены и хранятся в виде кучи (рис. 31). В этом случае для нахождения нужной записи будет выполнен полный просмотр (сканирование) всех строк таблицы. Если таких строк сотни тысяч или даже миллионы, то это может оказать значительное влияние на производительность.

6	...
1	...
44	...
17	...
8	...
13	...

Страница 10

10	...
2	...
23	...
15	...
3	...
7	...

Страница 11

14	...
32	...
5	...
4	...
11	...
9	...

Страница 12

12	...
19	...
22	...
37	...
41	...
35	...

Страница 13

...

Рис. 31. Хранение данных таблицы в виде кучи

Для быстрого доступа к данным без полного сканирования таблицы в SQL Server используются индексы, которые представляют собой структуру в виде В – дерева (B-tree), хранящуюся на страницах по 8 Кб, называемых узлами дерева. Используется два типа индексов: *кластеризованный* и *некластеризованный*. Отличие между ними проявляется в листовых узлах дерева.

Кластеризованный индекс позволяет хранить данные таблицы на диске в отсортированном виде. Это не отдельная структура, а сама таблица, хранящаяся в виде упорядоченного дерева, листья которого содержат непосредственно данные таблицы, по этой причине кластеризованный индекс для таблицы может быть только один. По определению этот индекс является уникальным. На рис. 32 показан пример кластеризованного индекса.

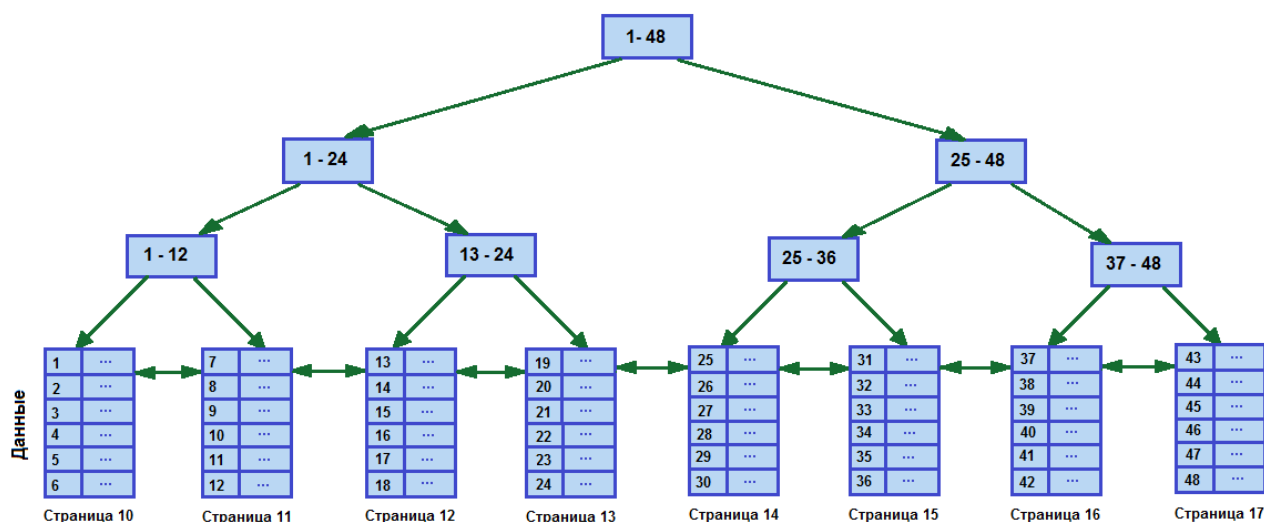


Рис. 32. Кластеризованный индекс

Как правило, все таблицы базы данных должны иметь кластеризованный индекс, поскольку это дает ряд преимуществ, таких как возможность управления фрагментацией таблиц с помощью инструкции `ALTER INDEX`, используя параметры `REBUILD` или `REORGANIZE`, или возможность перемещения таблиц в другую файловую группу, путем создания кластеризованного индекса в другой группе.

Для ускорения поиска строк по значениям в столбцах, отличных от кластеризованного индекса, используют некластеризованные индексы. Это отдельная структура, связанная с таблицей и содержащая данные из индексируемых столбцов. Листовые узлы такого индекса содержат только данные индексируемых столбцов и указатель на строки с реальными данными на соответствующей странице (рис. 33).

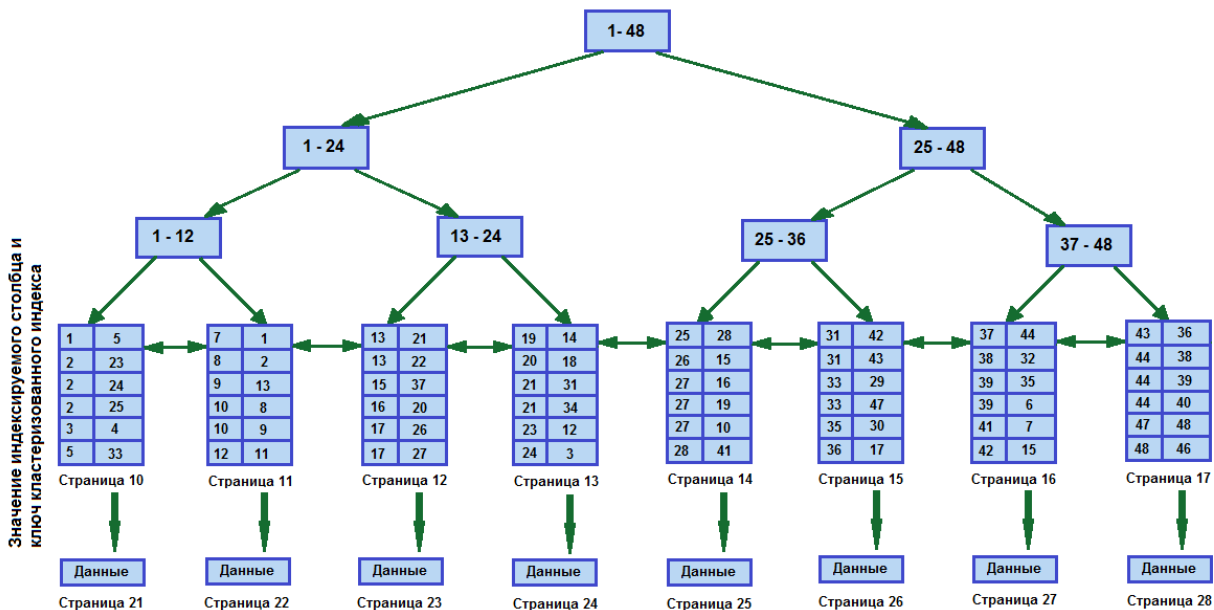


Рис. 33. Некластеризованный индекс с указателем на кластеризованный

В этом случае требуется дополнительная операция для обнаружения и получения данных, не входящих в состав индекса. Указатель содержит либо ключ кластеризованного индекса, если он имеется для таблицы, либо непосредственно идентификатор строки (row ID) с данными, если они содержатся в куче. В индекс можно включить до 16 ключевых полей с ограничением до 900 байт, а для сокращения числа операций поиска данных и обхода ограничений на длину индекса можно дополнительно включить неключевые столбцы в листовые элементы. Такой прием называется индексом с включенными столбцами (included column).

Поскольку некластеризованные индексы хранят в качестве указателей на строки данных ключи кластеризованного индекса, важно, чтобы эти ключи были как можно меньше.

Ниже демонстрируются примеры влияния различных индексов на производительность запросов.

Для начала посмотрим, как происходит поиск данных в таблице, не имеющей индексов, т.е. данные которой находятся в куче.

Таблица «dimDate» не имеет индексов и содержит 219146 строки.

Выполним следующий запрос

```
SELECT *  
FROM [dbo].[dimDate]  
WHERE [KeyDate] = 18610219
```

и посмотрим фактический план выполнения запроса (Actual Execution Plan). В нем описана последовательность физических и логических операций, которые оптимизатор запросов SQL Server использует для выполнения запроса.

В Management Studio планы выполнения запросов (предполагаемый и фактический) можно посмотреть в удобной графической форме, для этого нужно нажать соответствующую кнопку на панели инструментов при открытом окне запросов (рис. 34). При этом для предполагаемого плана не нужно запускать запрос на выполнение.

В плане с помощью древовидной структуры показан поток данных (в виде стрелок) и последовательность преобразующих его операторов (в виде значков). Толщина стрелок показывает объем обрабатываемых данных. Читать его нужно справа налево и сверху вниз. Если навести курсор на значок оператора, то появится список сведений о нем, также можно посмотреть подробности, щелкнув на значке правой кнопкой мыши и выбрав **Свойства** из контекстного меню.

Справку по всему списку доступных сведений об операторах можно посмотреть в электронной документации по ссылке: [https://technet.microsoft.com/ru-ru/library/ms178071\(v=sql.105\).aspx](https://technet.microsoft.com/ru-ru/library/ms178071(v=sql.105).aspx). Список всех операторов: [https://msdn.microsoft.com/ru-ru/library/ms175913\(v=sql.105\).aspx](https://msdn.microsoft.com/ru-ru/library/ms175913(v=sql.105).aspx).

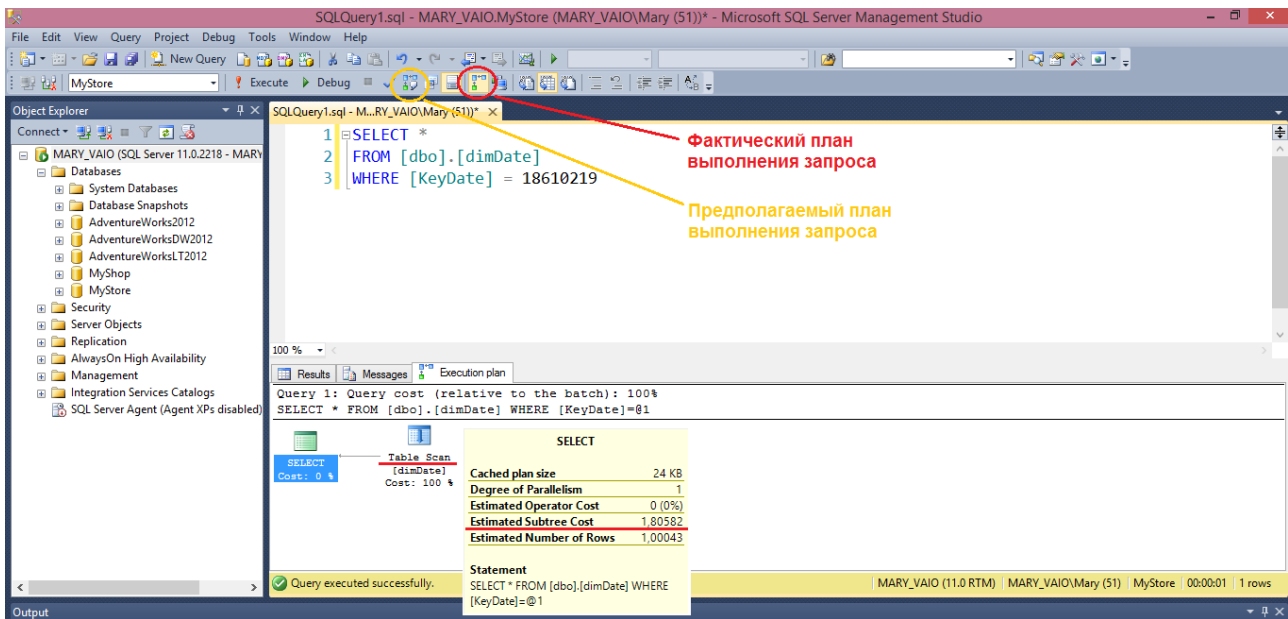


Рис. 34. План выполнения запроса к таблице, не имеющей индексов

Под каждой пиктограммой оператора показана его стоимость в процентах от общей стоимости запроса. Это число помогает понять, какая операция использует больше всего ресурсов.

Оценить примерные затраты на выполнение данной операции и всех предшествующих операций в поддереве можно с помощью показателя **Предполагаемая стоимость поддерева (Estimated Subtree Cost)**.

Для определения примерной общей стоимости всего запроса нужно выбрать корневую операцию (слева в графическом плане), будет показана суммарная стоимость всех операций.

Из рис. 34 видно, что производится сканирование всей таблицы (операция scan) и стоимость запроса ESC = 1,80582.

Эта стоимость является весьма относительным показателем производительности. Она измеряется в условных единицах и показывает предполагаемое время выполнения запроса на некой «эталонной аппаратной конфигурации». Опираясь на стоимость, оптимизатор запросов старается построить такой план, который потребует от сервера наименьшего числа ресурсов. Реальные аппаратные конфигурации могут значительно отличаться, кроме того, следует учитывать возможную неактуальность статистики или

ограничения модели оценки и т.д. Поэтому при оценке производительности запросов на реальном хранилище данных следует пользоваться более информативными метриками, предоставляемыми SQL Server.

Основные показатели, на которых следует сфокусировать внимание – это количество процессорного времени (CPU) для обработки кода запроса и операций чтения/записи (IO) из КЭШа данных (логических чтений) и, если данных нет в КЭШе, то физических чтений с диска. Предполагаемую стоимость следует учитывать для предварительной оценки запросов [5].

Таким образом, задача увеличения производительности сводится к задаче уменьшения операций чтения с диска и использования ресурсов центрального процессора.

Для замера реального времени выполнения запроса можно применить следующие подходы.

- ✓ Используя функции SQL Server работы со временем, вычислять разницу между системным временем до и после выполнения запроса.
- ✓ Использовать инструкции Transact-SQL для исследования статистики, включив их перед выполнением запроса. Установки повлияют только на текущий сеанс и позволят SQL Server выводить информацию о производительности на вкладку **Сообщения** (Messages).

SET STATISTICS IO ON – отображает сведения о взаимодействии SQL Server с диском во время выполнения запроса. Наиболее полезными сведениями здесь являются: число страниц, считанных из КЭШа данных, число страниц, считанных с диска и число страниц, помещенных в кэш для запроса (упреждающее чтение).

SET STATISTICS TIME ON – отображает процессорное время в миллисекундах, которое было использовано для синтаксического анализа, компиляции и выполнения каждой инструкции. Этот хороший способ измерения, так как это серверная метрика. На рис. 35 показан пример использования этих инструкций.

- ✓ Использовать специальную утилиту SQL Profiler, которая представляет собой развитый интерфейс, предназначенный для создания, анализа и управления трассировками. Все события сохраняются в файле трассировки, который затем может быть проанализирован (см. [5]).

Для запуска приложения из меню среды SQL Server Management Studio выберите **Инструменты (Tools)** -> **SQL Server Profiler** или в редакторе запросов щелкните правой кнопкой мыши, а затем выберите пункт **Трассировка запроса в приложении SQL Server Profiler (Trace Query in SQL Server Profiler)**.



Рис. 35. Исследование статистики запроса с помощью инструкций Transact-SQL

Однако следует помнить, что все эти способы измерения времени выполнения запроса сильно зависят от окружающей среды проведения теста.

Вообще тема исследования производительности запросов довольно обширна и не укладывается в рамки данного учебного пособия. Ей посвящены отдельные книги, некоторые из которых приведены в списке использованных источников.

Для демонстрации преимуществ использования механизма индексирования достаточно будет оценить стоимость запросов.

Проверить занимаемое таблицей место на диске можно с помощью

стандартного отчета **Использование диска таблицей** (Disk Usage by Table) (рис. 36). Для его создания щелкните правой кнопкой мыши на названии базы данных и выберите из контекстного меню **Reports -> Standard Reports -> Disk Usage by Table**.

Table Name	# Records	Reserved (KB)	Data (KB)	Indexes (KB)	Unused (KB)
dbo.dimDate	219 146	16 904	16 872	8	24

Рис. 36. Количество дискового пространства, занимаемого таблицей «dimDate»

Из рисунка видно, что под индекс стандартно выделена одна страница данных, но сам индекс не занимает место на диске.

Теперь создадим кластеризованный индекс и посмотрим, как он изменит стоимость предыдущего запроса и занимаемое таблицей место на диске. Для этого выполним инструкцию T-SQL и проверим, как изменился объем занимаемого таблицей места на диске (рис. 37).

```
CREATE CLUSTERED INDEX CL_IndexDate
ON [dbo].[dimDate]([KeyDate])
```

Table Name	# Records	Reserved (KB)	Data (KB)	Indexes (KB)	Unused (KB)
dbo.dimDate	219 146	16 792	16 544	96	152

Рис. 37. Количество дискового пространства, занимаемого таблицей «dimDate»

после создания кластеризованного индекса

Из рисунка видно, что созданный индекс не занимает много места на диске. Выполним тот же запрос, но уже на индексированной таблице и посмотрим, как изменилась его стоимость (рис. 38). Так как условие отбора задано на столбце, являющимся ключом кластеризованного индекса, то время поиска строки заметно сократилось, ESC = 0,0032831. Операция **scan** сменилась на операцию **seek**. Теперь не нужно просматривать все строки таблицы, а достаточно спуститься по нужной ветке дерева до соответствующего листа, где хранятся данные.

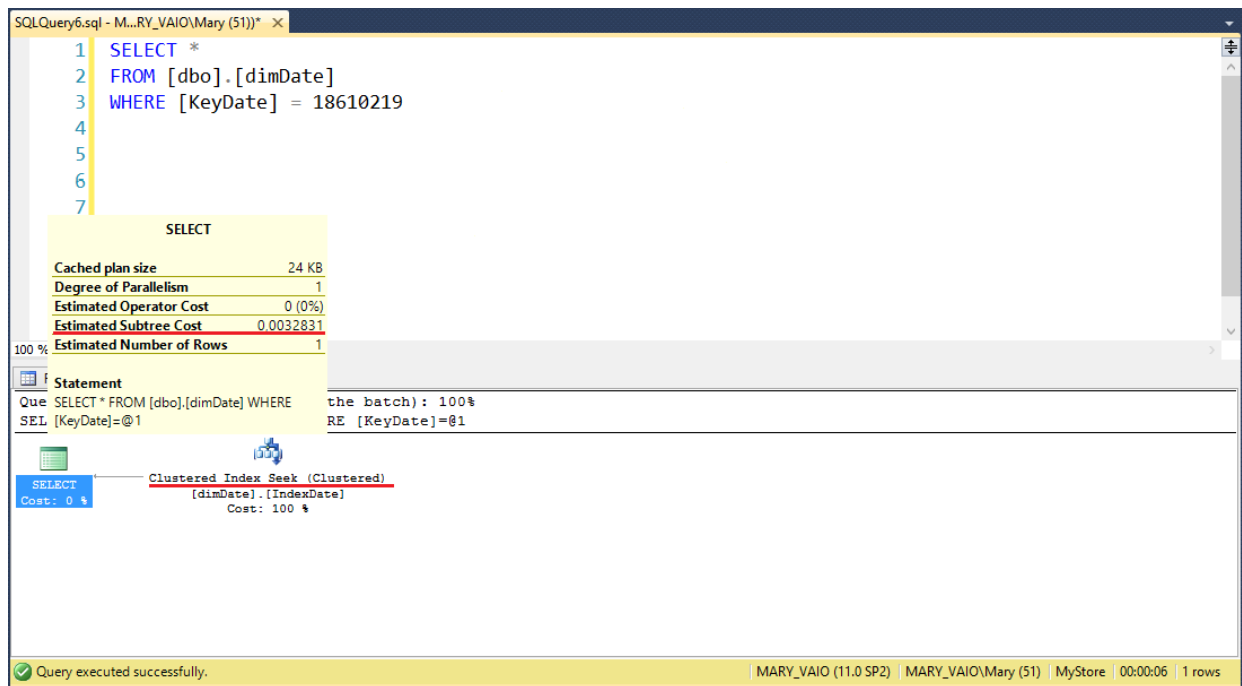


Рис. 38. Поиск данных по ключу кластеризованного индекса таблицы «dimDate»

Теперь выполним поиск строк не по ключевому полю индекса, например:

```

SELECT *
FROM [dbo].[dimDate]
WHERE [Year]=2015

```

В результате увидим, что сервер снова выполняет сканирование всей таблицы (рис. 39) и стоимость запроса вновь возрастет.

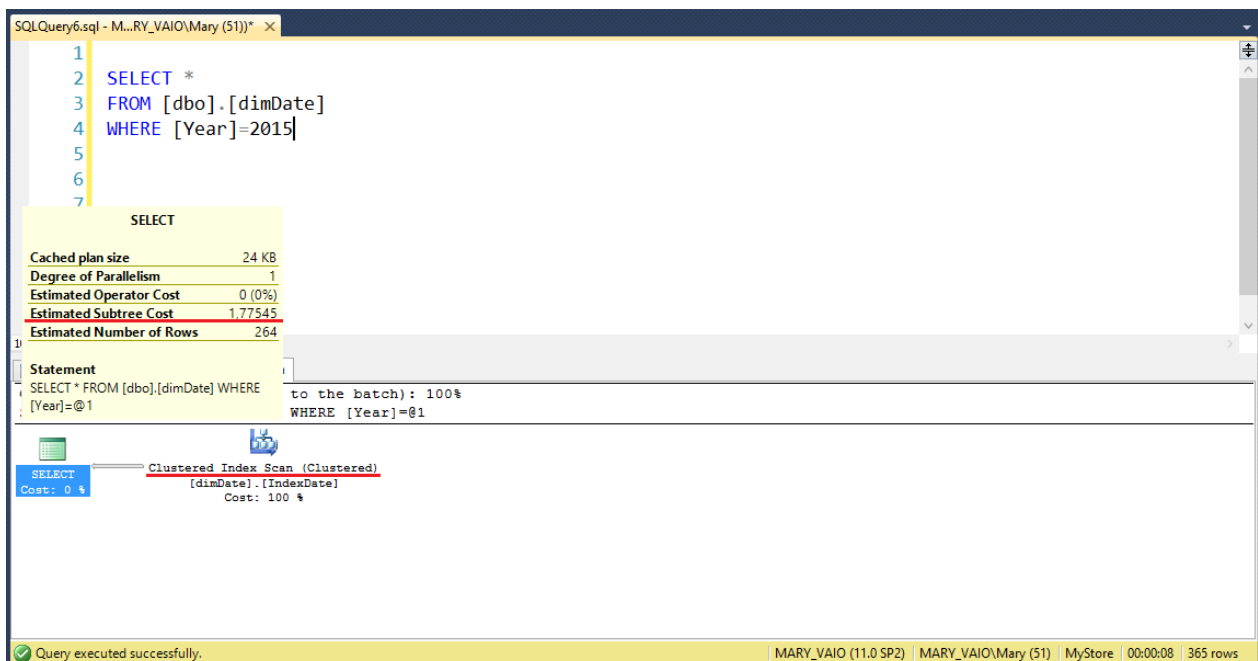


Рис. 39. Поиск данных по полю, не входящему в ключевые поля кластеризованного индекса таблицы «dimDate»

Для ускорения поиска по полям, не входящим в кластеризованный индекс, можно применить механизм некластеризованных индексов.

Создадим некластеризованный индекс по полю «Year» для таблицы «dimDate» и посмотрим, сколько теперь будет занимать места эта таблица на диске (рис. 40).

```
CREATE INDEX NonCL_Index_Year
on [dbo].[dimDate]([Year])
```

Table Name	# Records	Reserved (KB)	Data (KB)	Indexes (KB)	Unused (KB)
dbo.dimDate	219 146	20 920	16 544	4 056	320

Рис. 40. Количество дискового пространства, занимаемого таблицей «dimDate» после создания некластеризованного индекса

Как видно из рисунка, данный индекс занимает уже ощутимое место на диске – 4056Кб, это почти 25% от всего объема занимаемого таблицей.

Некластеризованные индексы повышают производительность запросов, возвращающих значения из ключевых полей этого индекса.

Например, такой запрос:

```
SELECT [KeyDate], [Year]
FROM [dbo].[dimDate]
WHERE [Year]=2015
```

будет иметь стоимость ESC = 0,0036835, но какова будет стоимость запроса, если необходимо выбрать данные, не входящие в некластеризованный индекс.

Снова выполним запрос:

```
SELECT *
FROM [dbo].[dimDate]
WHERE [Year]=2015
```

и посмотрим на его стоимость и план выполнения (рис. 41).

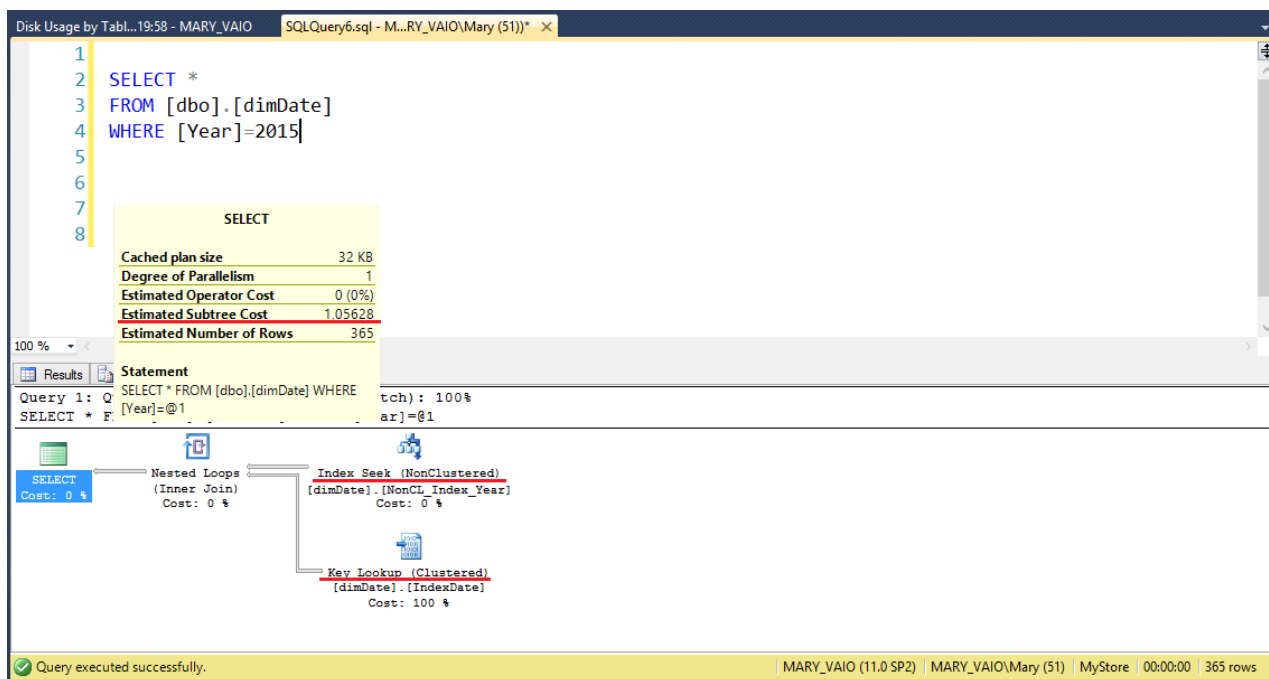


Рис. 41. Поиск по низкоселективному столбцу «Year»
некластеризованного индекса таблицы «dimDate»

Из рисунка видно, что в данном случае преимущество использования некластеризованного индекса невелико, ESC = 1,05628. Стоимость запроса ненамного уменьшилась при использовании некластеризованного индекса, а он, в свою очередь, занимает много места. Связано это с тем, что столбец, на котором задан некластеризованный индекс, содержит множество повторяющихся значений (*низкоселективных данных*) и требует многократного поиска данных по кластеризованному индексу. Отсюда вывод: некластеризованные индексы, заданные на низкоселективных столбцах часто не эффективны и стоимость запросов с их использованием может превысить стоимость простого сканирования таблицы. В таком случае сервер будет выполнять именно его.

Например, если задать некластеризованный индекс для низкоселективного столбца «Week» таблицы «dimDate», и выполнить поиск по нему, с выводом столбцов, не входящих в индекс, то сервер будет выполнять сканирование таблицы, игнорируя этот индекс (рис. 42).

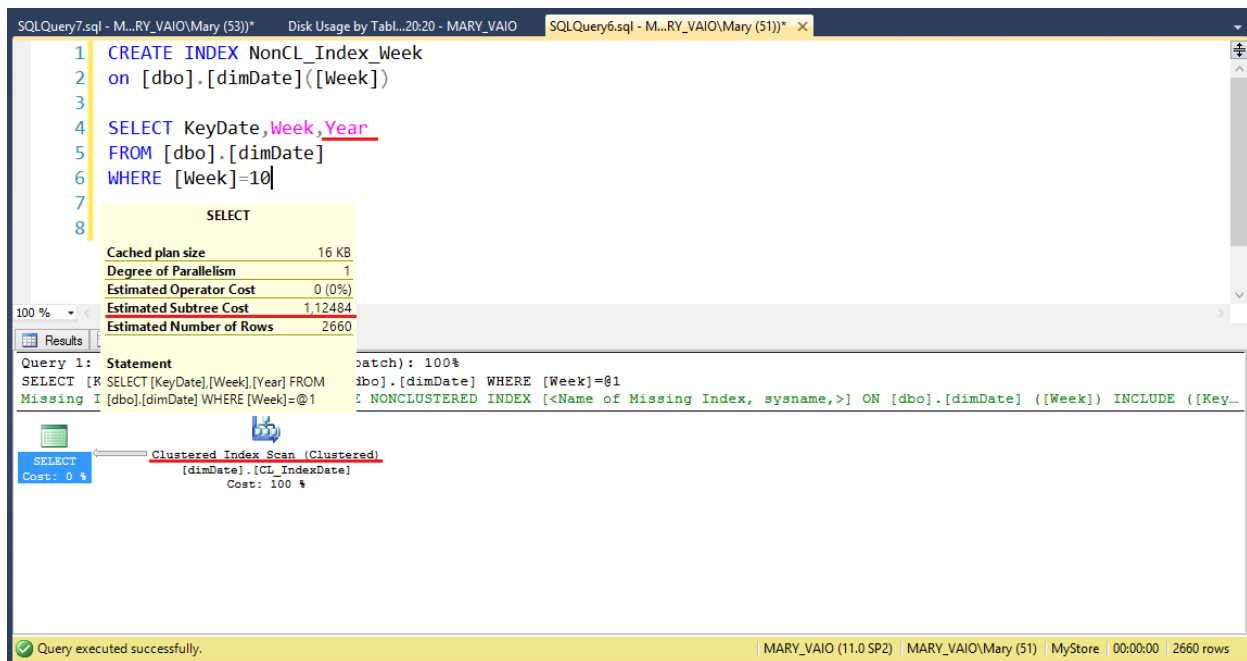


Рис. 42. Операция сканирования таблицы без использования индекса
при поиске по низкоселективному столбцу «Week»

Алгоритм принятия решения основан на статистике о селективности столбцов, которую ведет сервер для каждой таблицы. Объект статистики для таблицы создается по индексу или списку столбцов таблицы. Для просмотра свойств этого объекта из среды SQL Server Management Studio в **Обозревателе объектов** нужно выбрать таблицу и в её раскрывающемся списке папок найти папку **Статистика** (Statistics). Затем, щелкнув правой кнопкой мыши на объекте статистики, выбрать команду **Свойства** (Properties). Свойства включают заголовок, содержащий метаданные о статистике, гистограмму, содержащую распределение значений в первом ключевом столбце объекта статистики, и вектор плотностей для измерения корреляции с охватом нескольких столбцов. На рис. 43 показаны свойства для статистики по некластеризованному индексу, заданному по полю «Year».

Следующие данные описывают столбцы, возвращенные в результирующем наборе для гистограммы.

RANGE_HI_KEY – верхнее граничное значение столбца для шага гистограммы. Это значение столбца называется также ключевым значением.

RANGE_ROWS – предполагаемое количество строк, значение столбцов

которых находится в пределах шага гистограммы, исключая верхнюю границу.

EQ_ROWS – предполагаемое количество строк, значение столбцов которых равно верхней границе шага гистограммы.

DISTINCT_RANGE_ROWS – предполагаемое количество строк с различающимся значением столбца в пределах шага гистограммы, исключая верхнюю границу.

AVG_RANGE_ROWS – среднее количество строк с повторяющимися значениями столбцов в пределах шага гистограммы, исключая верхнюю границу ($\text{RANGE_ROWS} / \text{DISTINCT_RANGE_ROWS}$ для $\text{DISTINCT_RANGE_ROWS} > 0$).

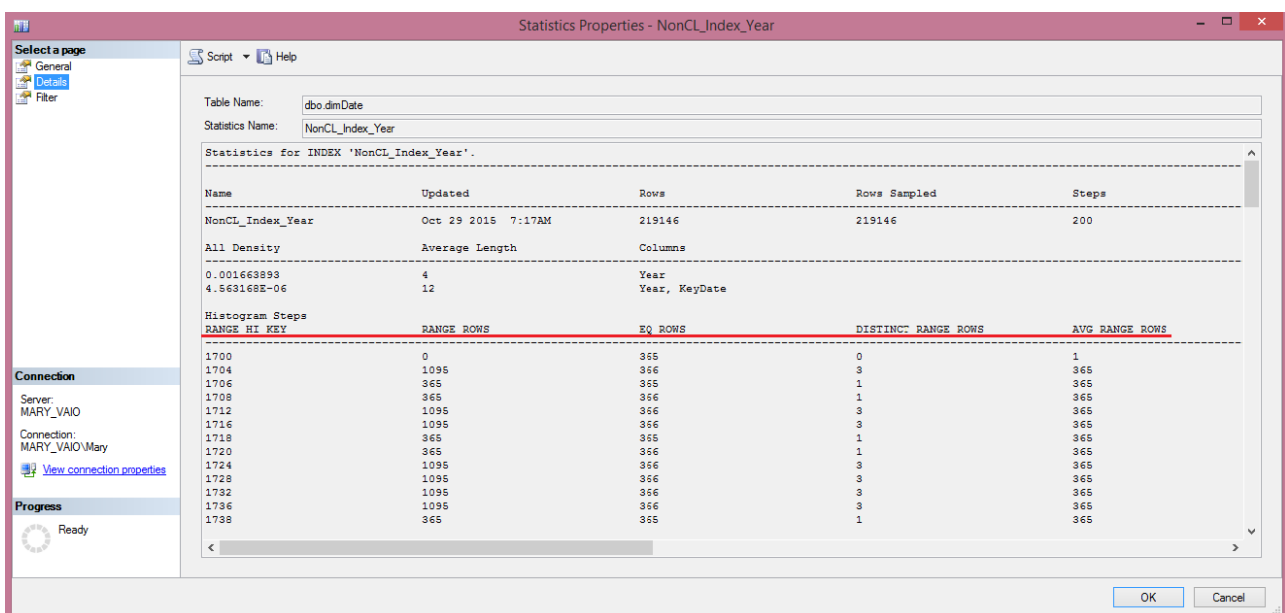


Рис. 43. Свойства статистики для некластеризованного индекса по полю «Year»

Более подробно о свойствах статистики можно узнать в электронной документации по ссылке [https://technet.microsoft.com/ru-ru/library/hh510179\(v=sql.110\).aspx](https://technet.microsoft.com/ru-ru/library/hh510179(v=sql.110).aspx).

Перед тем, как построить некластеризованный индекс, разработчик должен подумать о селективности столбцов, на которых он будет построен.

Если индексировать высокоселективные столбцы (мало повторяющихся значений), то выигрыш по стоимости запросов, содержащих условие отбора по этому столбцу, очевиден.

Для того чтобы повысить производительность низкоселективных

запросов можно использовать механизм расширения функциональности некластеризованных индексов – *включаемые столбцы* (Included Columns).

Например, если включить столбец «Year» таблицы «dimDate» в некластеризованный индекс по полю «Week» и выполнить запрос из предыдущего примера, то вместо операции полного сканирования (scan) будет выполнен поиск по индексу (seek), и стоимость запроса уменьшится примерно в сто раз (рис. 44.)

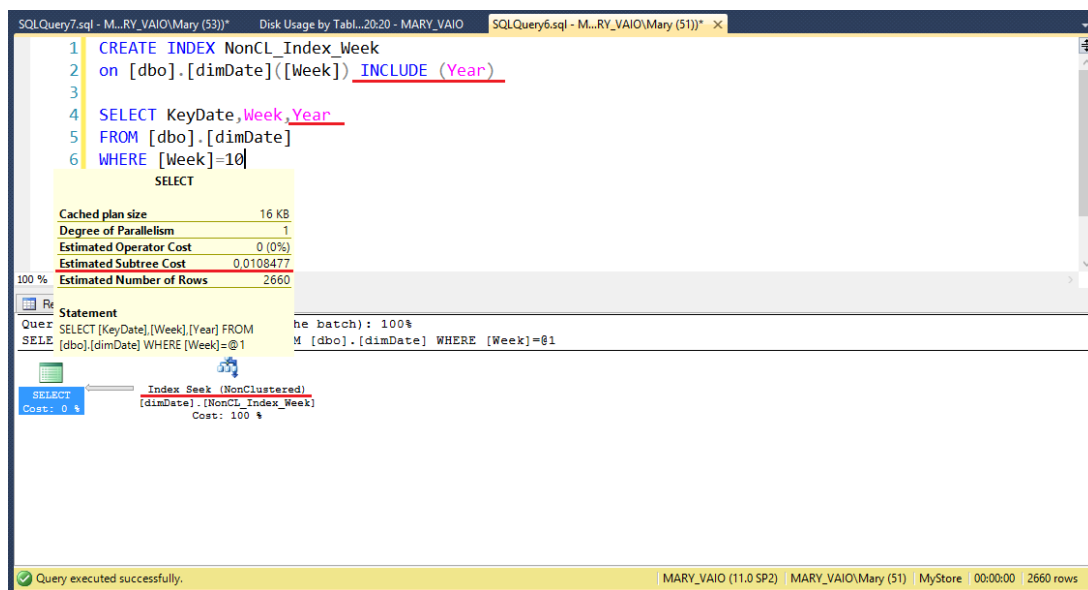


Рис. 44. Пример некластеризованного индекса с включаемым столбцом

Помимо преимуществ, использование индексов вызывает и ряд негативных моментов. Прежде всего, это дополнительный объем памяти для хранения индексов, а также замедление операций вставки, обновления и удаления записей, требующих перестроения дерева индекса.

Страницы индекса на конечном уровне представляют собой двусвязный список и выделение пространства на странице для новых записей или обновления существующих приводит к необходимости добавления новой страницы, перестройки указателей и переносу части имеющихся данных на новые 8-килобайтные страницы. Такая операция называется *разбиением страниц* (page split). Она является весьма ресурсоемкой и может занять довольно длительное время. Существует ряд приемов для уменьшения этого негативного эффекта.

Одним из них является выбор в качестве кластеризованного индекса автоинкрементных столбцов. Это могут быть глобальные идентификаторы, генерируемые с помощью функции NEWSEQUENTIALID(), гарантирующей возрастающие значения, или поле со свойством IDENTITY. Тогда все новые записи, добавляемые в хранилище, будут иметь возрастающие значения и помещаться в дерево индекса справа, без его перестройки.

Также можно установить значение *коэффициента заполнения страниц индекса* (fill factor) для резервирования свободного места на странице. Это значение в процентах от 1 до 100, которое показывает процент заполнения страниц индекса на конечном уровне. Например, если коэффициент равен 70, то на каждой странице конечного уровня будет зарезервировано 30 процентов от занимаемого ею дискового пространства. По умолчанию значение равно нулю, что означает полное заполнение страниц конечного уровня. При этом нет конкретных рекомендаций для значения коэффициента заполнения, но следует помнить, что низкое значение хотя и снизит количество операций по разделению страниц, но потребует больший объем памяти и приведет к снижению производительности операций чтения.

Значение можно подобрать эмпирически, например, отслеживая количество расщеплений страниц в секунду, происходящих в системе, с помощью счетчика Page Splits/Sec, объекта SQL Server:Access Methods. Значение счетчика можно посмотреть, выполнив запрос, показанный на рис. 45.

Подробные сведения можно посмотреть в электронной документации по ссылке: [https://technet.microsoft.com/ru-ru/library/ms177426\(v=sql.110\).aspx](https://technet.microsoft.com/ru-ru/library/ms177426(v=sql.110).aspx).

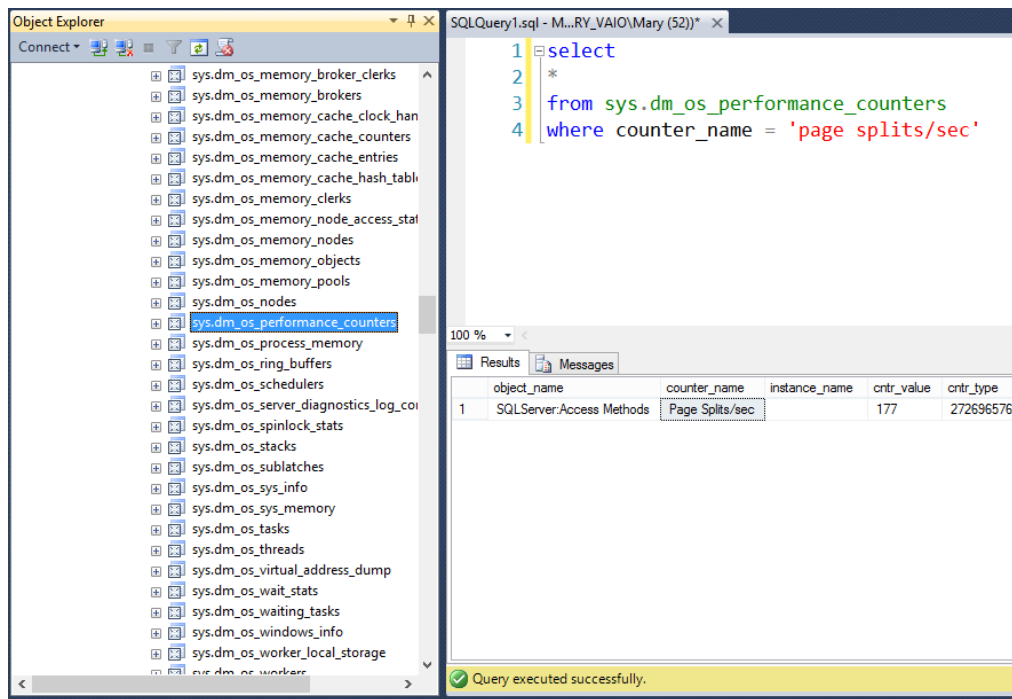



Рис. 45. Отслеживание количества расщеплений страниц в секунду

Значение коэффициента заполнения страниц индекса можно задать несколькими способами. Например, открыть таблицу в **Конструкторе таблиц** (Design), затем открыть диалоговое окно **Индексы и ключи** (Indexes/Keys), нажав на панели инструментов кнопку  и выбрав нужный индекс в списке слева диалогового окна, ввести в поле **Коэффициент заполнения** число от 1 до 100.

Можно в обозревателе объектов (Object Explorer) выбрать нужный индекс в списке индексов таблицы, и в окне свойств этого индекса, выбрав страницу **Параметры** (Options) в списке слева, ввести в поле **Коэффициент заполнения** число от 1 до 100.

Также можно воспользоваться инструкциями T-SQL, например:

```
USE [MyStore];
ALTER INDEX NonCL_Index_Week ON [dbo].[dimDate]
REBUILD WITH (FILLFACTOR = 80);
```

Для того чтобы уменьшить объем памяти для хранения индексов в SQL Server можно воспользоваться механизмом *отфильтрованных индексов*. В этом случае некластеризованный индекс будет построен только для определенного диапазона значений. Такой индекс эффективен, когда известно, что данные в

определенном диапазоне значений запрашиваются чаще других.

Например, если известно, что информация по товарам в определенной ценовой категории запрашивается чаще остальных, то на таблице фактов можно задать следующий отфильтрованный индекс:

```
CREATE INDEX NonCL_Index_Price
ON [dbo].[FactSales] ([UnitPrice]) INCLUDE ([ProductKey],[Quantity])
WHERE UnitPrice<5000
```

Столбец в выражении отфильтрованного индекса необязательно должен быть ключевым или включенным столбцом в определении отфильтрованного индекса, если выражение отфильтрованного индекса эквивалентно предикату запроса, а запрос не возвращает столбец с результатами запроса в выражение отфильтрованного индекса.

На рис. 46 показаны объемы занимаемой памяти для отфильтрованного индекса и индекса, заданного на тех же полях, но без фильтра.

Table Name	# Record s	Reserved (KB)	Data (KB)	Indexes (KB)	Unused (KB)
dbo.FactSales	42 307	7 128	4 992	1 768	368
Индекс без фильтра					
Table Name	# Record s	Reserved (KB)	Data (KB)	Indexes (KB)	Unused (KB)
dbo.FactSales	42 307	5 384	4 992	288	104
Отфильтрованный индекс					

Рис. 46. Отчеты, показывающие объем памяти, занимаемой индексом без фильтра и отфильтрованным индексом

Выигрыш по скорости выполнения будут иметь запросы, осуществляющие выборку из подмножества данных, входящих в отфильтрованный индекс.

Например:

```
SELECT [ProductKey],[Quantity],[UnitPrice]
FROM [dbo].[FactSales]
WHERE UnitPrice between 1 and 1000
```

Еще одним механизмом, ускоряющим построение агрегаций по фактическим таблицам (средняя стоимость заказанных товаров, количество проданных товаров и т.д.), являются *колоночные индексы* (Columnstore).

Этот новый метод хранения некластеризованных индексов появился в SQL Server 2012 и доступен только в редакциях Enterprise Edition и Developer Edition. Вместо обычного построчного хранения индексируемые данные хранятся в столбчатом виде (рис. 47), что позволяет запросам, обрабатывающим отдельные столбцы, не считывать строки целиком, а считывать с диска только нужный столбец, сокращая при этом количество дисковых операций ввода/вывода и использование КЭШа. Такой способ хранения может существенно ускорить выполнение запросов к ХД в 10-100 раз (см.[4]).

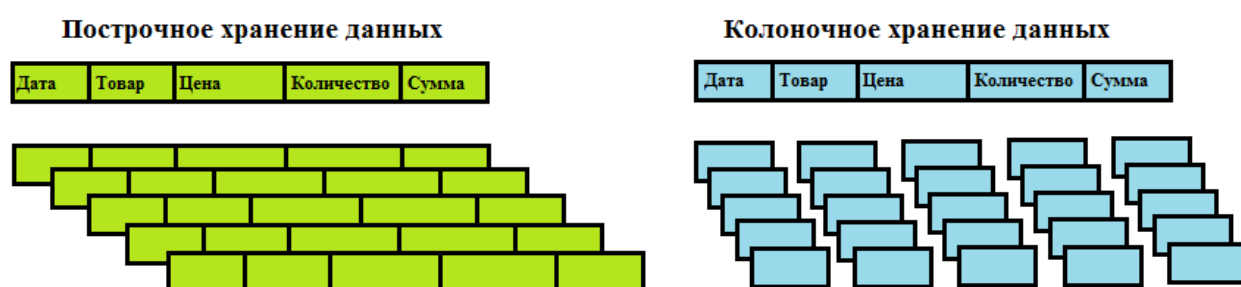


Рис. 47. Различные варианты хранения данных на диске

Колоночный индекс не может быть использован в качестве первичного (primary key) или внешнего ключа (foreign key), не может быть задан на вычисляемом поле и не может быть отфильтрованным или содержать включенные столбцы. На одной таблице может быть создан только один колоночный индекс.

В колоночный индекс могут быть включены столбцы следующих типов: int, bigint, smallint, tinyint, money, smallmoney, bit, float, real, char(n), varchar(n), nchar(n), nvarchar(n), date, datetime, datetime2, smalldatetime, time, datetimeoffset с точностью ≤ 2 , decimal или numeric с точностью ≤ 18 . Максимальное количество столбцов, которые могут быть включены в колоночный индекс — 1024.

Колоночные индексы требуют реконструкции строк, поэтому таблицы, содержащие этот индекс, превращаются в доступные **только для чтения!** Операторы INSERT, UPDATE, DELETE и MERGE не поддерживаются.

При необходимости обновления таблицы с колоночным индексом необходимо проделать следующие шаги.

- ✓ Удалить или отключить этот индекс с помощью следующих команд:
`DROP INDEX {Имя индекса} ON {Имя таблицы}` – удаление индекса;
`ALTER INDEX {Имя индекса} ON {Имя таблицы} DISABLE` – отключение индекса.
- ✓ Изменить данные.
- ✓ Перестроить колоночный индекс с помощью команды:
`ALTER INDEX {Имя индекса} ON {Имя таблицы} REBUILD`.

Если колоночный индекс задается на секционированной таблице (об этом механизме речь пойдет ниже), то процедура обновления может быть организована более эффективно (см. п. 3.2.3).

Ниже приведен пример использования колоночного индекса для таблицы «FactSales» и сравнение производительности запроса к этой таблице без его использования и с его использованием. Также сравнивается количество операций чтения с диска или КЭШ.

На рис. 48 показана стоимость запроса, вычисляющего среднюю стоимость товара по годам за последние десять лет. Функции CAST и CONVERT используются совместно т.к. SQL Server не позволяет напрямую преобразовывать данные типа bigint в тип данных date.

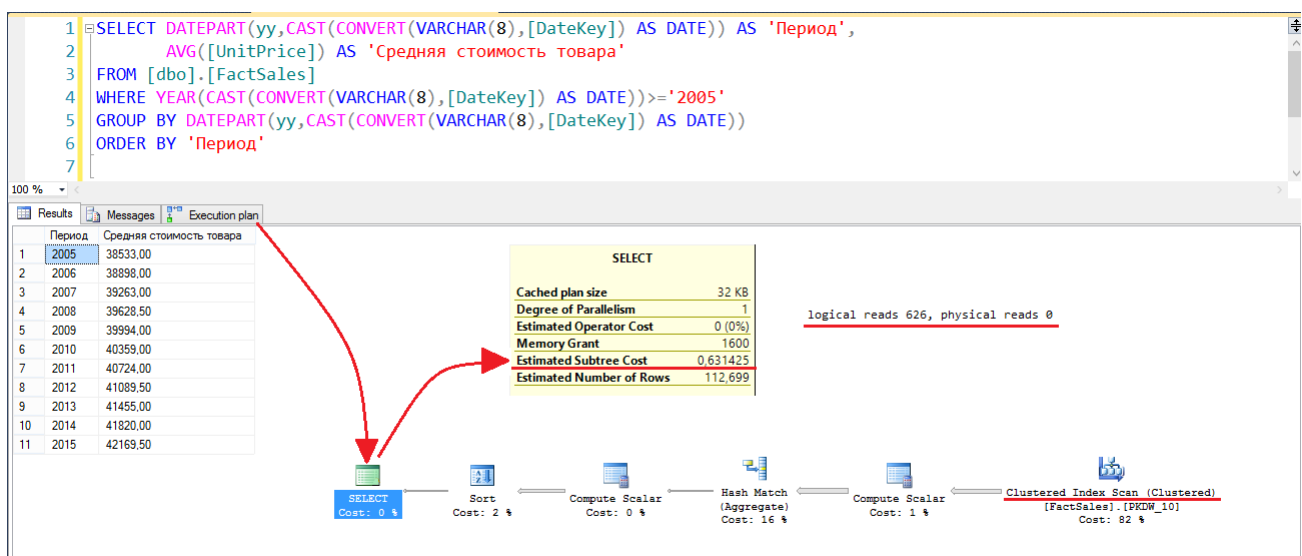


Рис. 48. Агрегирующий запрос к таблице без колоночного индекса

Для того чтобы правильно построить колоночный индекс, прежде всего необходимо продумать его структуру и, наряду с мерами, по которым будет часто проводится агрегация, необходимо включить в него все ключевые столбцы, связанные с этими мерами, по которым будут отбираться данные (foreign key).

Для построения индекса можно воспользоваться средствами графического интерфейса среды SSMS, выбрав в поддереве для конкретной таблицы папку **Индексы (Indexes)** и в её контекстном меню пункт **Новый индекс (New Index)** - > **Некластеризованный колоночный индекс (Non-Clustered Columnstore Index)**. Далее, в открывшемся диалоговом окне нажать кнопку **Добавить (Add)** и выбрать поля, которые необходимо включить в индекс.

Также можно воспользоваться инструкцией T-SQL.

```
CREATE NONCLUSTERED COLUMNSTORE INDEX MyColumnStoreIndex
ON [dbo].[FactSales] ([DateKey],[UnitPrice],[Quantity]);
```

Теперь посмотрим, как изменится стоимость запроса с использованием колоночного индекса (рис. 49).

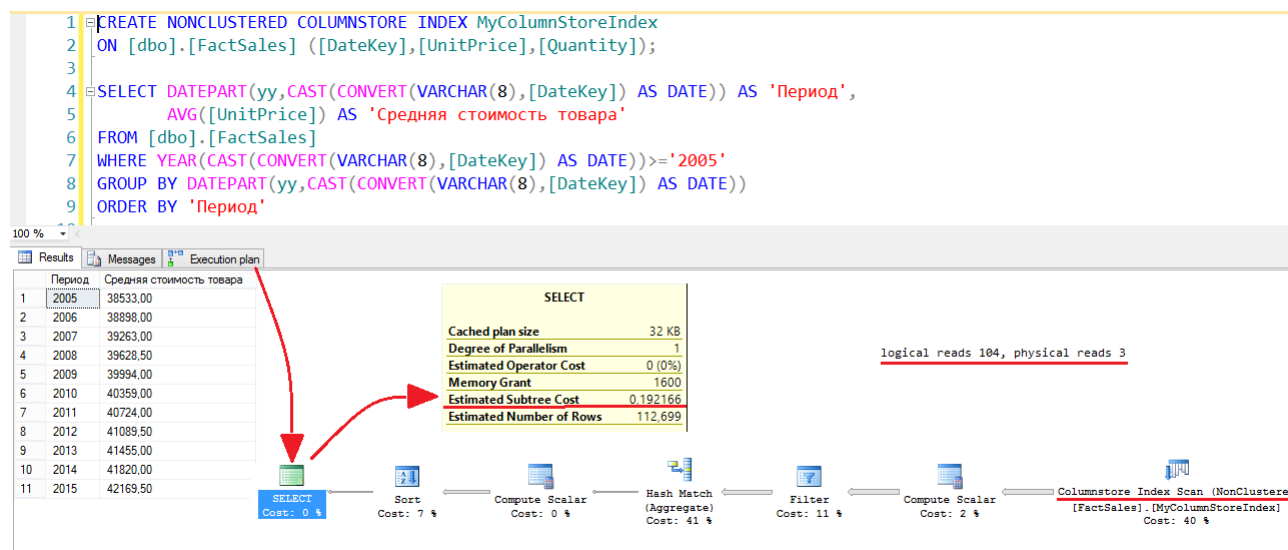


Рис. 49. Агрегирующий запрос к таблице с колоночным индексом

Стоимость заметно уменьшилась, так же, как и количество операций чтения, это связано с тем что колоночные индексы всегда архивируются и, следовательно, уменьшается количество операций чтения.

В SQL Server 2012 есть два системных представления для работы с

колоночными индексами: `sys.column_store_segments` и `sys.column_store_dictionaries`.

Можно дать несколько общих рекомендаций по стратегии создания индексов для ХД.

Для таблиц измерений необходимо создать кластеризованные индексы. При этом индекс по бизнес-ключу позволит быстрее выполнять процесс загрузки хранилища, а индекс по суррогатному ключу позволит быстрее выполнять запросы за счет ускорения операций соединения между таблицами. Если ключевое поле одно, то альтернативы нет, в противном случае нужно принимать решение, исходя из потребностей.

Далее следует создать некластеризованные индексы по всем полям, по которым впоследствии будут выполняться группировки и которые будут часто включаться в фильтр запросов. Причем, если между атрибутами существует иерархическая связь, то следует сделать этот индекс составным или с включаемыми столбцами.

Для таблиц фактов кластеризованный индекс лучше создавать по столбцу календаря (по дате), так как новые данные будут добавляться в таблицу по возрастанию даты, и, что особенно важно, по этому столбцу удобно проводить секционирование.

Некластеризованные индексы нужно создавать по внешним ключам, ссылающимся на часто используемые измерения. Возможно, следует создать эти индексы составными или с включаемыми столбцами.

Также следует рассмотреть возможность создания некластеризованных индексов по наиболее часто используемым мерам, так как по ним может осуществляться фильтрация данных.

Ниже перечислены все индексы, используемые в хранилище «MyStore» для каждой таблицы.

Таблицы измерений

Таблица «dimDate».

Название индекса	Тип индекса	Столбы, по которым построен индекс	Описание
PKDW_1	Кластеризованный индекс	KeyDate	Первичный ключ, являющийся производным от значения даты

Таблица «dimGeography».

Название индекса	Тип индекса	Столбы, по которым построен индекс	Описание
PKDW_2	Кластеризованный индекс	keyGeography	Суррогатный первичный ключ со свойством IDENTITY(1,1)
NonCL_dimG eography_Cou ntry	Некластеризованный индекс с включенными столбцами. Коэффициент заполнения равен 70.	NameCountry	Ключевое поле индекса. Повышает производительность параметризованных отчетов, основанных на выборе определенной страны
		NameRegion	Включенный столбец
		NameCity	Включенный столбец
NonCL_dimG eography_City	Некластеризованный индекс. Коэффициент заполнения равен 70.	NameCity	Ключевое поле индекса. Повышает производительность параметризованных отчетов, основанных на выборе определенного города

Таблица «MiniDimCustomerDemography».

Название индекса	Тип индекса	Столбы, по которым построен индекс	Описание
PKDW_3	Кластеризованный индекс	keyCustomerDemograp hy	Суррогатный первичный ключ со свойством IDENTITY(1,1)

Таблица «dimCustomer».

Название индекса	Тип индекса	Столбы, по которым построен индекс	Описание
PKDW_4	Кластеризованный индекс	keyCustomer	Суррогатный первичный ключ со свойством IDENTITY(1,1)
NonCL_dimCustomer_AlternateKey	Некластеризованный индекс	CustomerAlternateKey	Ключевое поле индекса. Поддерживает поиск по суррогатному ключу во время загрузки данных для медленно меняющегося измерения
NonCL_dimCustomer_GeographyKey	Некластеризованный индекс	GeographyKey	Ключевое поле индекса. Ускоряет операцию соединения с таблицей «dimGeography»

Таблица «dimPromotion».

Название индекса	Тип индекса	Столбы, по которым построен индекс	Описание
PKDW_5	Кластеризованный индекс	keyPromotion	Суррогатный первичный ключ со свойством IDENTITY(1,1)
NonCL_dimPromotion_AlternateKey	Уникальный некластеризованный индекс	PromotionAlternateKey	Ключевое поле индекса. Для ускорения процесса загрузки данных в хранилище

Таблица «dimProduct».

Название индекса	Тип индекса	Столбы, по которым построен индекс	Описание
PKDW_6	Кластеризованный индекс	keyProduct	Суррогатный первичный ключ со свойством IDENTITY(1,1)
NonCL_dimProduct_AlternateKey	Уникальный некластеризованный индекс	ProductAlternateKey	Ключевое поле индекса. Для ускорения процесса загрузки данных в хранилище
NonCL_dimProduct_CategoryProduct	Некластеризованный индекс	CategoryProduct	Ключевое поле индекса. Повышает производительность параметризованных отчетов, основанных на выборе определенных категорий товаров

Таблица «dimKindOfPay».

Название индекса	Тип индекса	Столбы, по которым построен индекс	Описание
PKDW_7	Кластеризованный индекс	keyKindOfPay	Суррогатный первичный ключ со свойством IDENTITY(1,1)

Таблица «dimOffices».

Название индекса	Тип индекса	Столбы, по которым построен индекс	Описание
PKDW_8	Кластеризованный индекс	keyOffices	Суррогатный первичный ключ со свойством IDENTITY(1,1)
NonCL_dimO ffices_Alternat eKey	Уникальный некластеризованный индекс	OfficesAlternateKey	Ключевое поле индекса. Для ускорения процесса загрузки данных в хранилище

Таблица «dimManagers».

Название индекса	Тип индекса	Столбы, по которым построен индекс	Описание
PKDW_9	Кластеризованный индекс	keyManager	Суррогатный первичный ключ со свойством IDENTITY(1,1)
NonCL_dimM anagers_Alter nateKey	Уникальный некластеризованный индекс	ManagerAlternateKey	Ключевое поле индекса. Для ускорения процесса загрузки данных в хранилище
NonCL_dimM anagers_Office Key	Некластеризованный индекс с включенными столбцами	OfficeKey	Ключевое поле индекса. Ускоряет операцию соединения с таблицей «dimOffices» и повышает производительность параметризованных отчетов, основанных на выборе менеджеров из определенных офисов
		BirthDate	Включенный столбец
		FIO	Включенный столбец

Таблица «dimMethodOfDelivery».

Название индекса	Тип индекса	Столбы, по которым построен индекс	Описание
PKDW_10	Кластеризованный индекс	keyMethodOfDelivery	Суррогатный первичный ключ со свойством IDENTITY(1,1)

Таблицы фактов

Таблица «FactSales».

Название индекса	Тип индекса	Столбы, по которым построен индекс	Описание
PKDW_11	Кластеризованный индекс	NumberSale	Ключевое поле первичного ключа
		NumberLineInSale	Ключевое поле первичного ключа
		DateKey	Ключевое поле первичного ключа. По данному полю выполняется секционирование таблицы
NonCL_FactSales_CustomerKey	Некластеризованный индекс	CustomerKey	Ключевое поле индекса на основе внешнего ключа. Помогает строить отчеты, извлекающие строки на основе избирательного предиката измерения «dimCustomer»
NonCL_FactSales_CustomerDemographyKey	Некластеризованный индекс	CustomerDemographyKey	Ключевое поле индекса на основе внешнего ключа. Помогает строить отчеты, извлекающие строки на основе избирательного предиката измерения «MiniDimCustomerDemography»
NonCL_FactSales_ProductKey	Некластеризованный индекс	ProductKey	Ключевое поле индекса на основе внешнего ключа. Помогает строить отчеты, извлекающие строки на основе избирательного предиката измерения «dimProduct»
NonCL_FactSales_ManagerKey	Некластеризованный индекс.	ManagerKey	Ключевое поле индекса на основе внешнего ключа. Помогает строить отчеты, извлекающие строки на основе избирательного предиката измерения «dimManagers»
NonCL_FactSales_KindOfPayKey	Некластеризованный индекс	KindOfPayKey	Ключевое поле индекса на основе внешнего ключа. Помогает строить отчеты, извлекающие строки на основе избирательного предиката измерения «dimKindOfPay»
ClSt_FactSales	Колоночный	DateKey	Ключевое поле, связанное с

	индекс. Повышает производительность запросов , которые просматривают и агрегируют большие объемы данных		мерами, по которому отбираются данные
		CustomerKey	Ключевое поле, связанное с мерами, по которому отбираются данные
		CustomerDemographyKey	Ключевое поле, связанное с мерами, по которому отбираются данные
		ProductKey	Ключевое поле, связанное с мерами, по которому отбираются данные
		ManagerKey	Ключевое поле, связанное с мерами, по которому отбираются данные
		KindOfPayKey	Ключевое поле, связанное с мерами, по которому отбираются данные
		PromotionKey	Ключевое поле, связанное с мерами, по которому отбираются данные
		Quantity	Столбец (мера), по которому проводится агрегация
		UnitPrice	Столбец (мера), по которому проводится агрегация
		Subtotal	Столбец (мера), по которому проводится агрегация
		DiscountCustAmt	Столбец (мера), по которому проводится агрегация
		PromoAmt	Столбец (мера), по которому проводится агрегация
		TaxAmt	Столбец (мера), по которому проводится агрегация
		Freight	Столбец (мера), по которому проводится агрегация
		Total	Столбец (мера), по которому проводится агрегация

Таблица «FactDelivery».

Название индекса	Тип индекса	Столбы, по которым построен индекс	Описание
PKDW_12	Кластеризован	keyDelivery	Поле первичного ключа.

	ный индекс.		Идентификационный номер доставки
NonCL_FactDelivery_CustomerKey	Некластеризованный индекс	CustomerKey	Ключевое поле индекса на основе внешнего ключа. Помогает строить отчеты, извлекающие строки на основе избирательного предиката измерения «dimCustomer»
NonCL_FactDelivery_GeographyKey	Некластеризованный индекс	GeographyKey	Ключевое поле индекса на основе внешнего ключа. Помогает строить отчеты, извлекающие строки на основе избирательного предиката измерения «dimGeography»
NonCL_FactDelivery_MethodOfDeliveryKey	Некластеризованный индекс	MethodOfDeliveryKey	Ключевое поле индекса на основе внешнего ключа. Помогает строить отчеты, извлекающие строки на основе избирательного предиката измерения «dimMethodOfDelivery»
NonCL_FactDelivery_Filter	Некластеризованный отфильтрованный индекс с включенными столбцами. Позволяет сэкономить место на диске, индексируя данные только за последние три года	FactualDateOfDelivery	Ключевое поле индекса, по которому задан фильтр индекса: [FactualDateOfDelivery] >='20130101' AND [FactualDateOfDelivery] <='20160101'
		CustomerKey	Включенный столбец
		GeographyKey	Включенный столбец
		MethodOfDeliveryKey	Включенный столбец
		Cost	Включенный столбец, по которому проводится агрегация

Следует помнить, что поскольку индексы увеличивают производительность запросов к хранилищу, но замедляют загрузку данных в него, то важно соблюдать баланс между оптимизацией процесса загрузки хранилища и производительностью запросов.

Одним из способов оптимизации процесса загрузки данных, повышения производительности запросов и управляемости данными в ХД является секционирование.