

Физическое проектирование хранилища данных

Так как хранилище – это реляционная база данных, то основные приемы создания базы, таблиц, связей и т.п. для него в SQL Server те же, что и для OLTP-баз. Однако существует ряд особенностей, на которые следует обратить внимание при реализации хранилища на SQL Server.

Во-первых, данные в хранилище загружаются периодически по расписанию, а не в режиме реального времени, а значит, можно использовать простую модель восстановления (simple recovery model), в которой место в журнале зафиксированных транзакций освобождается автоматически и которая требует минимального места на диске для его хранения. Если же применяется модель полного восстановления (full recovery model) или модель с неполным протоколированием (bulk Logged recovery model), то необходимо регулярно создавать резервные копии журнала транзакций, так как он постоянно растет с каждой загрузкой данных. Подробные сведения о различных моделях восстановления базы данных можно узнать в электронной документации по ссылке: [https://msdn.microsoft.com/ru-ru/library/ms189275\(v=sql.110\).aspx](https://msdn.microsoft.com/ru-ru/library/ms189275(v=sql.110).aspx)

Во-вторых, для облегчения процесса загрузки данных в хранилище, возможно, потребуется создать промежуточные таблицы (staging tables). Они используются для временного хранения данных перед этапом очистки и объединения с данными из других источников, а также могут служить так называемым «буфером» между хранилищем и транзакционными базами данных. Например, если в исходной базе данных произошли изменения, требующие модификации способов загрузки данных в хранилище, то достаточно лишь изменить запрос, считывающий эти данные в промежуточные таблицы, а сам ETL-процесс будет выполняться обычным образом.

Промежуточные таблицы являются внутренним инструментом организации ETL-процесса и никогда не показываются конечным пользователям. Для удобства их можно хранить в отдельной схеме базы данных (Schema), что упростит администрирование. В типовом хранилище данных достаточно двух схем: одна с обычными таблицами ХД и другая с промежуточными [4].

Еще один момент, который следует обязательно учитывать при создании хранилища данных – это механизм управления дисковым пространством. По умолчанию в SQL Server применяется динамическое управление, т.е. база данных может автоматически увеличиваться или сжиматься по мере необходимости, но операции эти происходят в момент загрузки хранилища и замедляют её. Также многочисленные операции по незначительному автоматическому увеличению или сжатию базы могут вызвать фрагментирование данных, а это влияет на производительность запросов, считывающих большой объем данных. Поэтому режим **автоматического сжатия** (Auto Shrink) для хранилищ должен быть отключен, а ростом всех файлов данных и журналов следует управлять вручную, заранее предусмотрев достаточный объем диска для хранения данных и регистрационных журналов.

Ниже приведен скрипт создания базы данных для хранилища «MyStore».

```
USE Master;
IF DB_ID(N'MyStore') IS NOT NULL
DROP DATABASE [MyStore];
GO
CREATE DATABASE [MyStore]
ON PRIMARY
( NAME = N'MyStore', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\MyStore.mdf' , SIZE = 51200KB ,
FILEGROWTH = 10240KB )
LOG ON
( NAME = N'MyStore_log', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\MyStore_log.ldf' , SIZE = 10240KB ,
FILEGROWTH = 10%)
COLLATE Cyrillic_General_100_CI_AI
GO
ALTER DATABASE [MyStore] SET RECOVERY SIMPLE WITH NO_WAIT;
GO
ALTER DATABASE [MyStore] SET AUTO_SHRINK OFF
GO
```

Изначально для хранилища создается один файл данных MyStore.mdf с начальным размером 50 Мбайт и приращением в 10 Мбайт и файл журнала транзакций MyStore_log.ldf с начальным объемом в 10 Мбайт и приращением на 10%. После создания базы данных модель восстановления меняется на простую.

Существенное влияние на производительность, масштабируемость и управляемость хранилища данных оказывает выбор того или иного способа

хранения и индексирования данных, поэтому, уделив должное внимание проектированию физической модели хранилища, можно значительно улучшить эти аспекты.

В этом разделе будут описаны основные механизмы SQL Server, используемые при физическом проектировании хранилищ данных.

3.2.1. Файловые группы

В первую очередь следует продумать стратегию хранения данных на дисках, и хотя конкретные детали того, как данные размещаются на физических носителях, могут варьироваться от одного аппаратного решения к другому, существуют некоторые общие принципы, которые необходимо учитывать.

Прежде всего, это распределение данных в соответствии с их свойствами между физическими носителями. Компонент Microsoft SQL Server Database Engine может параллельно обрабатывать потоки при чтении данных с нескольких физических устройств, что может значительно сократить время на извлечение данных для запроса. В большинстве хранилищ распределение данных между физическими устройствами осуществляется через избыточный массив независимых дисков – RAID, как правило, в сети хранения данных (анг. Storage Area Network – SAN) или выделенном сервере хранения. В отсутствие решения RAID, можно предусмотреть распределение таблиц хранилища по нескольким физическим дискам, в зависимости от свойств данных этих таблиц. Например, если объем данных в таблице быстро растет, то возможно, для неё следует предусмотреть диск большего объема, или поместить таблицу на более быстрый диск, если её данные часто запрашиваются.

Для распределения данных по различным физическим дискам в SQL Server существует механизм файловых групп.

По умолчанию при создании базы данных создается два системных файла: файл данных и файл журнала транзакций. Файлы данных содержат данные и объекты, такие как таблицы, индексы, хранимые процедуры и представления. Файлы журнала содержат сведения, необходимые для восстановления всех транзакций в базе данных.

Если база содержит несколько файлов данных, то нельзя напрямую указать в каких именно файлах следует хранить данные из таблиц, но для таблиц можно указать файловые группы, к которым они будут относиться, а каждая файловая группа, в свою очередь, является своеобразным контейнером для файлов, хранящихся на различных дисках (рис. 27).

По умолчанию существует одна файловая группа PRIMARY, и все создаваемые файлы относятся к ней. Помимо самих данных эта файловая группа содержит еще и метаданные, то есть всю информацию о структуре базы данных. Удалить эту файловую группу нельзя. В случае если файловая группа PRIMARY единственная, сервер автоматически распределяет данные по файлам базы данных.

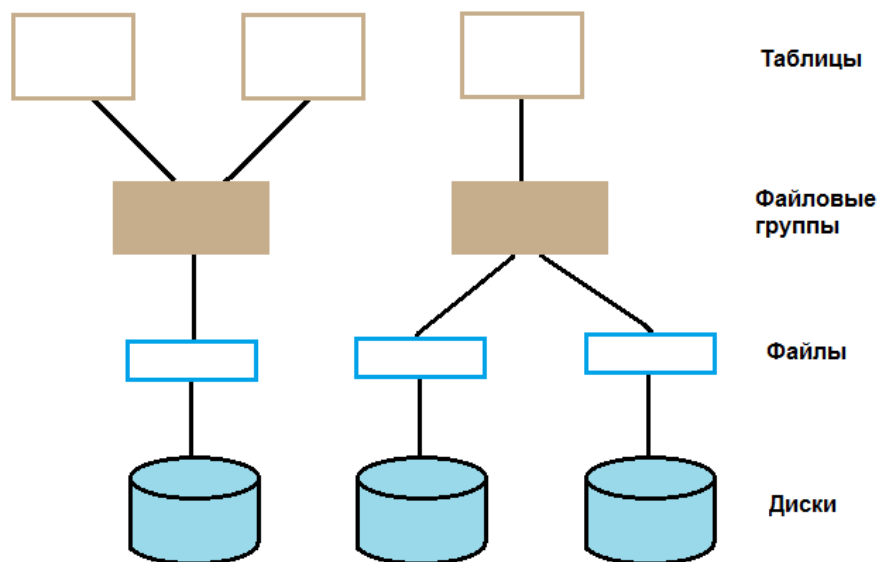


Рис. 27. Распределение данных по физическим носителям

Разработчик хранилища данных не может знать заранее, сколько будет дисков, но он может заранее предусмотреть инфраструктуру файловых групп, исходя из свойств данных, хранящихся в таблицах. Нет общих правил о распределении таблиц по различным файловым группам и, соответственно, количестве этих групп, но можно руководствоваться следующими рекомендациями:

- ✓ создать отдельные файловые группы для быстрорастущих таблиц;
- ✓ таблиц с часто запрашиваемыми данными;

- ✓ таблиц с редко запрашиваемыми данными;
- ✓ таблиц, данные в которых не изменяются со временем (для таких таблиц не нужно каждый раз создавать резервную копию);
- ✓ создать отдельную файловую группу для индексов;
- ✓ предусмотреть файловую группу по умолчанию, в которую будут попадать таблицы, для которых не указана явно файловая группа;
- ✓ создать файловую группу для промежуточных и временных таблиц.

Желательно предусмотреть как можно больше файловых групп, это не скажется на производительности ХД, но существенно повысит управляемость его данными.

Для управления файловыми группами нужно зайти в свойства базы данных и слева в списке страниц выбрать страницу **Файловые группы** (Filegroups). Здесь можно создать файловые группы и указать их свойства (рис. 28).

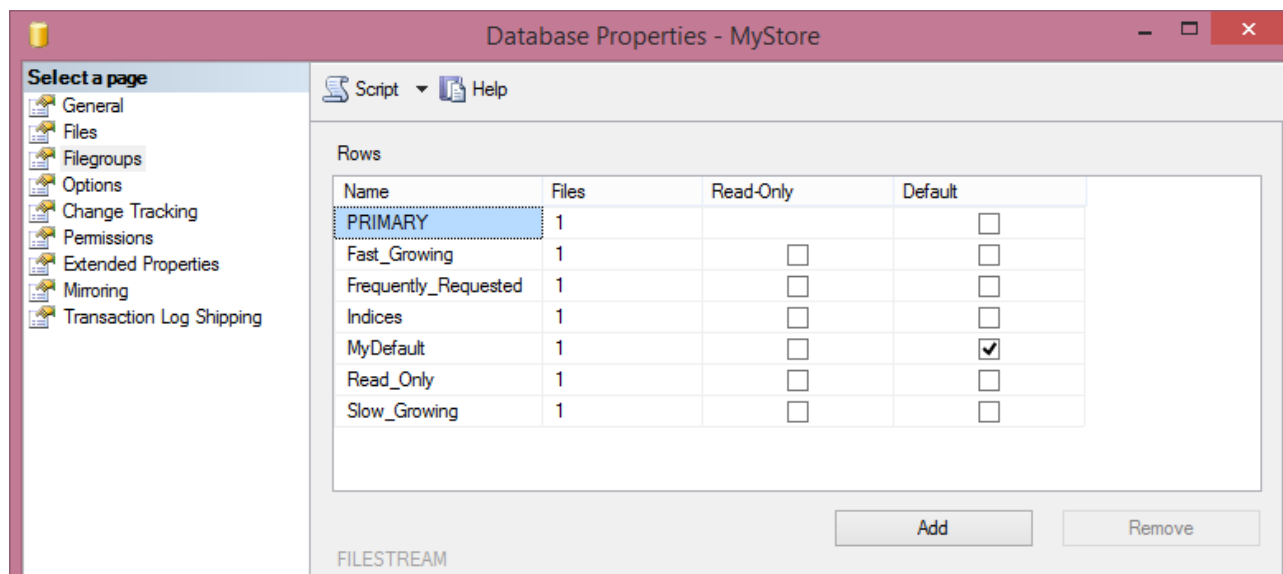


Рис. 28. Управление файловыми группами

Если для таблицы явно не указана файловая группа, то будет использоваться файловая группа, заданная по умолчанию, а это группа PRIMARY. Поскольку в этой группе хранится файл .mdf со всей важной информацией о хранилище, то возможно, следует переопределить файловую группу по умолчанию на другую.

Таблицы-справочники, данные в которых не изменяются, лучше поместить в отдельную файловую группу и затем после начальной загрузки данных

установить для этой файловой группы свойство **Только для чтения** (ReadOnly). Это позволит в дальнейшем не включать её каждый раз в резервные копии ХД.

Для каждой новой файловой группы необходимо указать хотя бы один файл. Создать файл, указать файловую группу, к которой он будет относиться и все необходимые свойства можно, зайдя на страницу **Файлы** (Files) (рис. 29).

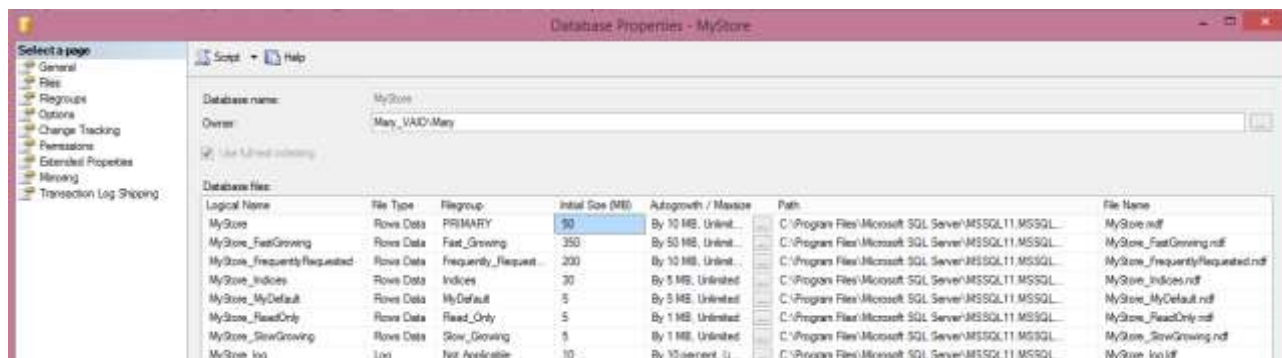


Рис. 29. Создание файлов для базы данных

Можно установить следующие свойства.

- ✓ **Начальный размер файла** (Initial Size).
- ✓ **Автоувеличение** (Autogrowth), если существующий размер файла недостаточен для хранения добавляемых данных, сервер автоматически запросит систему выделить файлу дополнительное дисковое пространство. Объем дополнительного дискового пространства (в процентах или мегабайтах) указывается в поле **Увеличение размера файла** (File Growth), а в разделе **Максимальный размер файла** (Maximum File Size) можно ограничить максимальный размер файла, установив переключатель **Ограничен** (Limited to). Для автоматического увеличения размера базы данных нужно установить флажок **Разрешить авторасширение** (Enable Autogrowth).

Управлять файловыми группами и файлами можно с помощью кода T-SQL. В следующем примере создается файловая группа «Fast_Growing» для базы данных «MyStore» и к ней добавляется файл с логическим именем «MyStore_FastGrowing».

```
USE [master]
GO
ALTER DATABASE [MyStore] ADD FILEGROUP [Fast_Growing]
```

```
GO
ALTER DATABASE [MyStore] ADD FILE
( NAME = N'MyStore_FastGrowing', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\MyStore_FastGrowing.ndf', SIZE
= 358400KB, FILEGROWTH = 51200KB)
TO FILEGROUP [Fast_Growing]
GO
```

Подробный синтаксис инструкции можно посмотреть в электронной документации по SQL Server 2012 в статье «Параметры инструкции ALTER DATABASE для файлов и файловых групп (Transact-SQL)» по адресу [https://msdn.microsoft.com/ru-ru/library/bb522469\(v=sql.110\).aspx](https://msdn.microsoft.com/ru-ru/library/bb522469(v=sql.110).aspx).

Также создавать и управлять файловыми группами можно на странице свойств **Общие** (General) при добавлении нового файла к базе данных, выбрав для него уже существующую файловую группу или создав новую. Для этого в столбце **Файловые группы** (Filegroup) из выпадающего списка нужно выбрать опцию **<Новая файловая группа>** (new filegroup) и задать необходимые параметры.

Следует также помнить, что распространенной практикой является размещение файлов данных и журналов транзакций на различных дисках.

После создания хранилища данных с набором необходимых файловых групп, можно создавать таблицы с привязкой к ним.

Таблицы создаются обычным образом. При использовании конструктора таблиц в Management Studio задать файловую группу можно в окне **Свойства** (Properties), которое вызывается по кнопке F4. В пункте **Спецификация регулярного пространства данных** (Regular Data Space Specification) нужно выбрать из списка необходимую файловую группу (рис. 30). При этом BLOB (анг. Binary Large Object — двоичный большой объект) поля таблиц в SQL Server можно выносить в отдельные файловые группы, указав их в пункте Text/Image Filegroup.

Если таблицы создаются с помощью кода, то в конце дописывается инструкция *ON {Имя файловой группы}*.

Далее приведен пример кода создания таблицы измерения «dimDate» с привязкой к файловой группе «Frequently_Requested».

```
CREATE TABLE dimDate
```

```

(KeyDate BIGINT NOT NULL,
[Date] DATE NOT NULL,
[Year] INT NOT NULL,
[Quarter] INT NOT NULL,
[Month] int NOT NULL,
[Week] int NOT NULL,
[Day] int NOT NULL,
[MonthName] NVARCHAR(20) NOT NULL,
[DayName] NVARCHAR(20) NOT NULL,
CONSTRAINT PKDW_1 PRIMARY KEY (KeyDate)
) ON [Frequently_Requested]

```

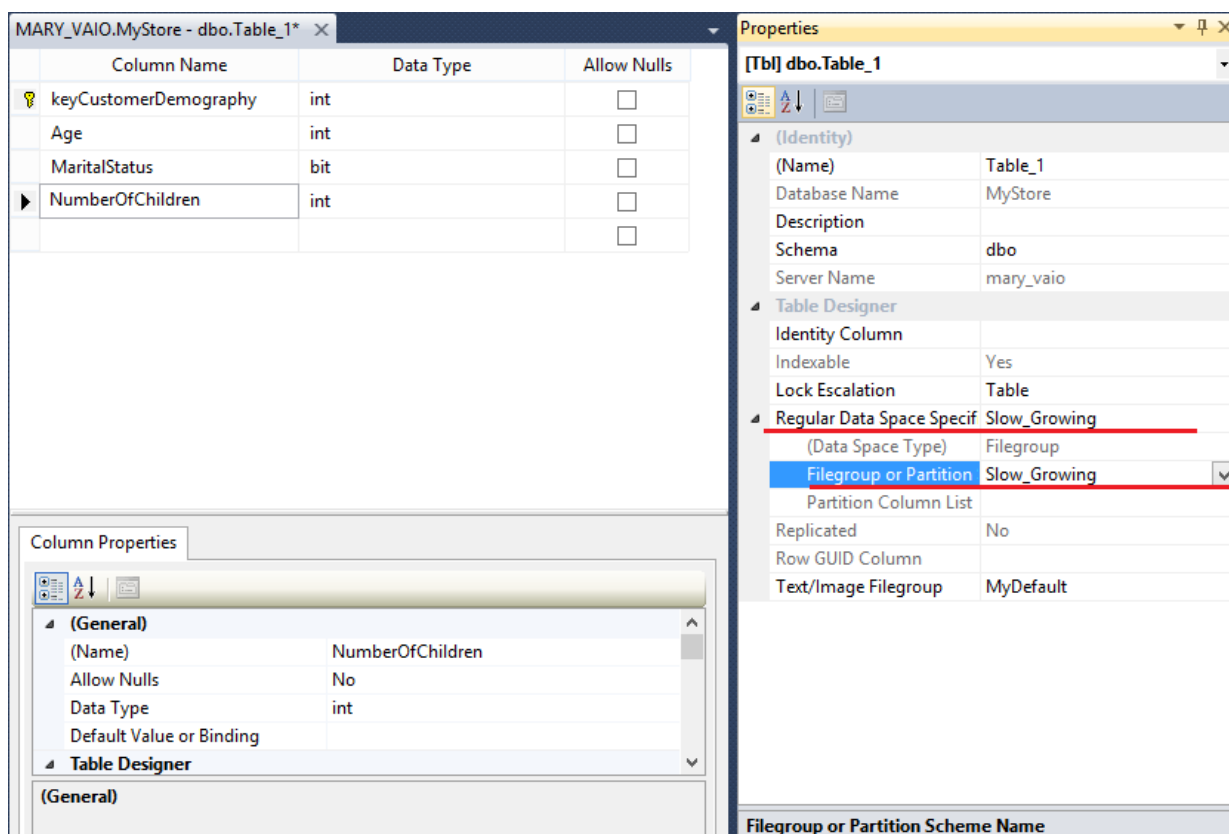


Рис. 30. Создание таблицы с привязкой к файловой группе

В прил. 1 целиком приведен скрипт создания хранилища данных «MyStore», с набором необходимых файловых групп и файлов.

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 столбцах, 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	...	10	...	14	...	12
1	...	2	...	32	...	19	...	
44	...	23	...	5	...	22	...	
17	...	15	...	4	...	37	...	
8	...	3	...	11	...	41	...	
13	...	7	...	9	...	35	...	
Страница 10		Страница 11		Страница 12		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.

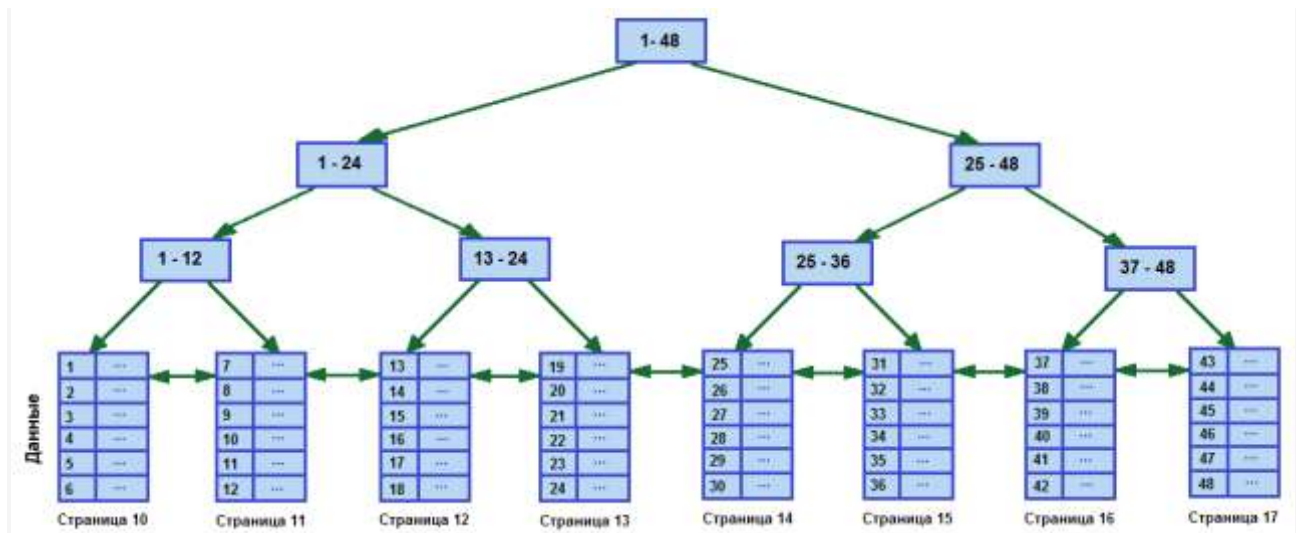


Рис. 32. Кластеризованный индекс

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).

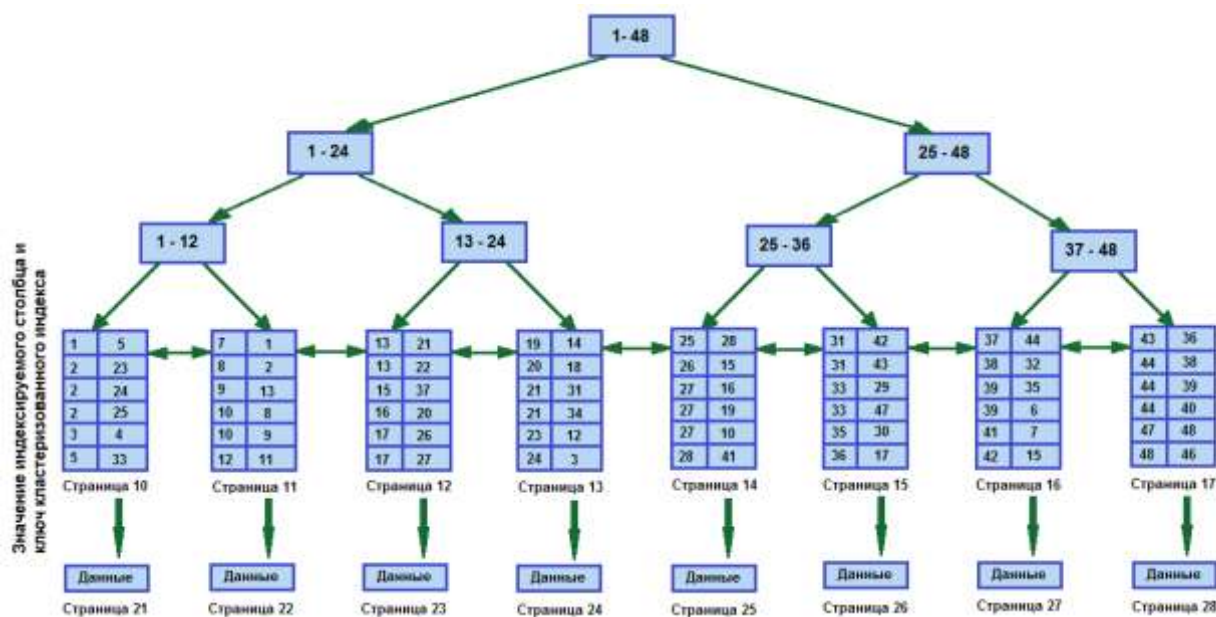


Рис. 33. Некластеризованный индекс с указателем на кластеризованный

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).

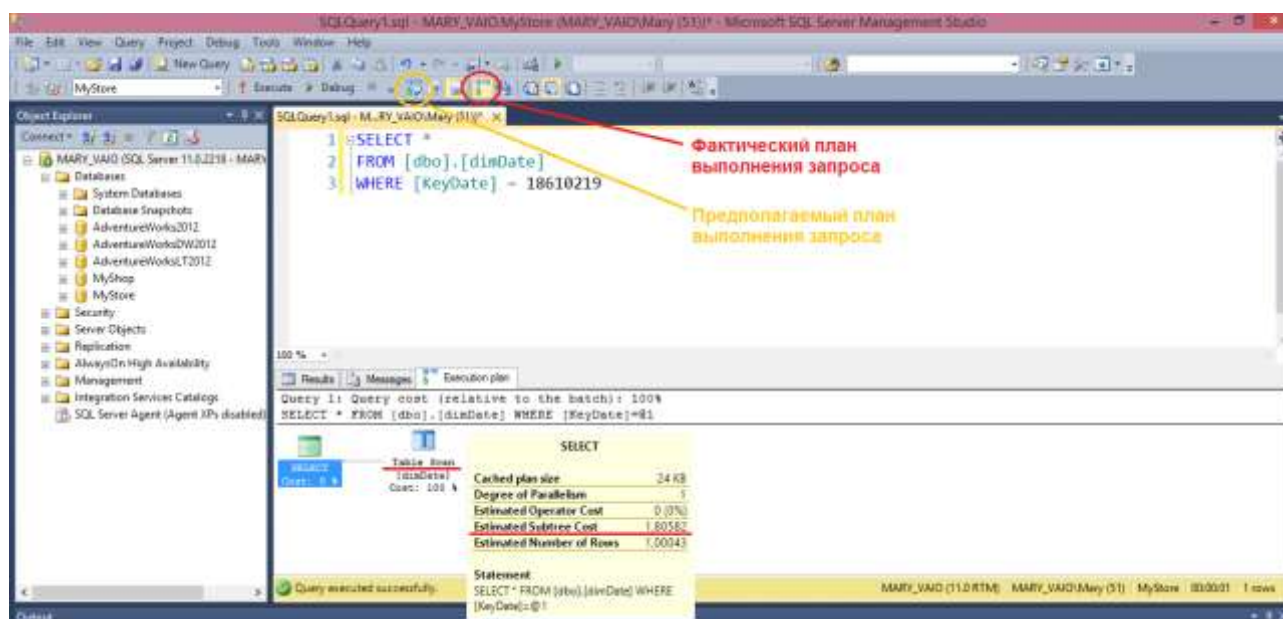


Рис. 34. План выполнения запроса к таблице, не имеющей индексов

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.

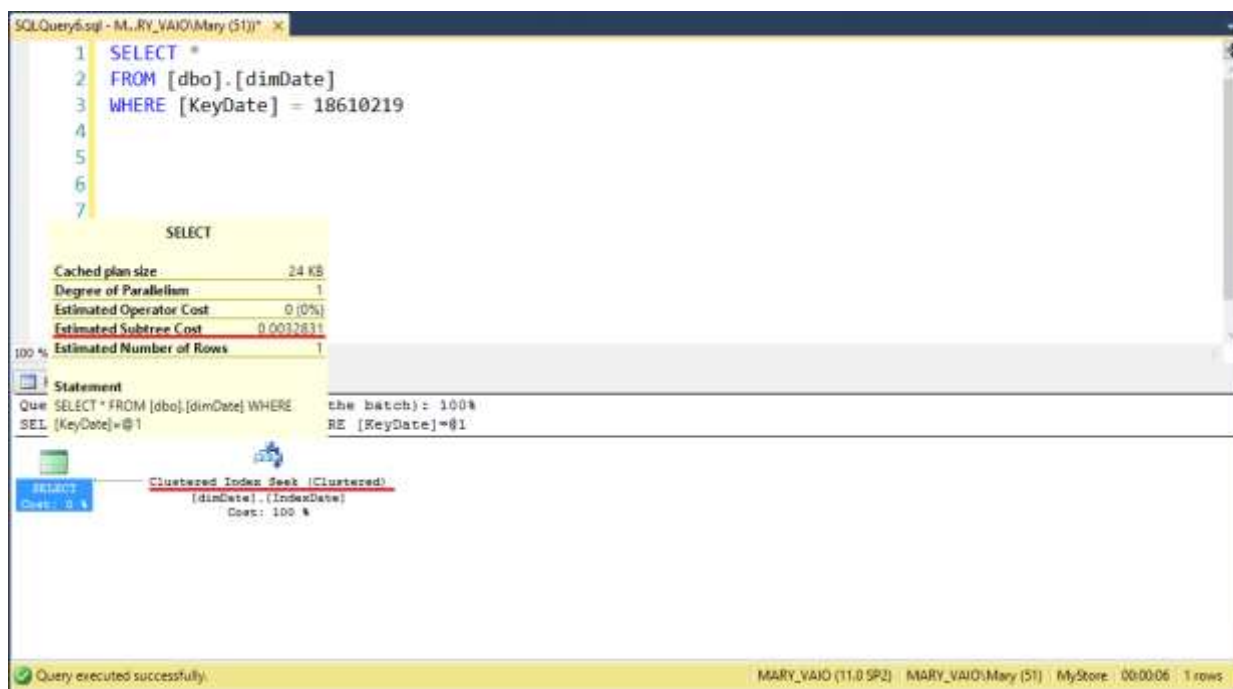


Рис. 38. Поиск данных по ключу кластеризованного индекса таблицы «dimDate»

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).

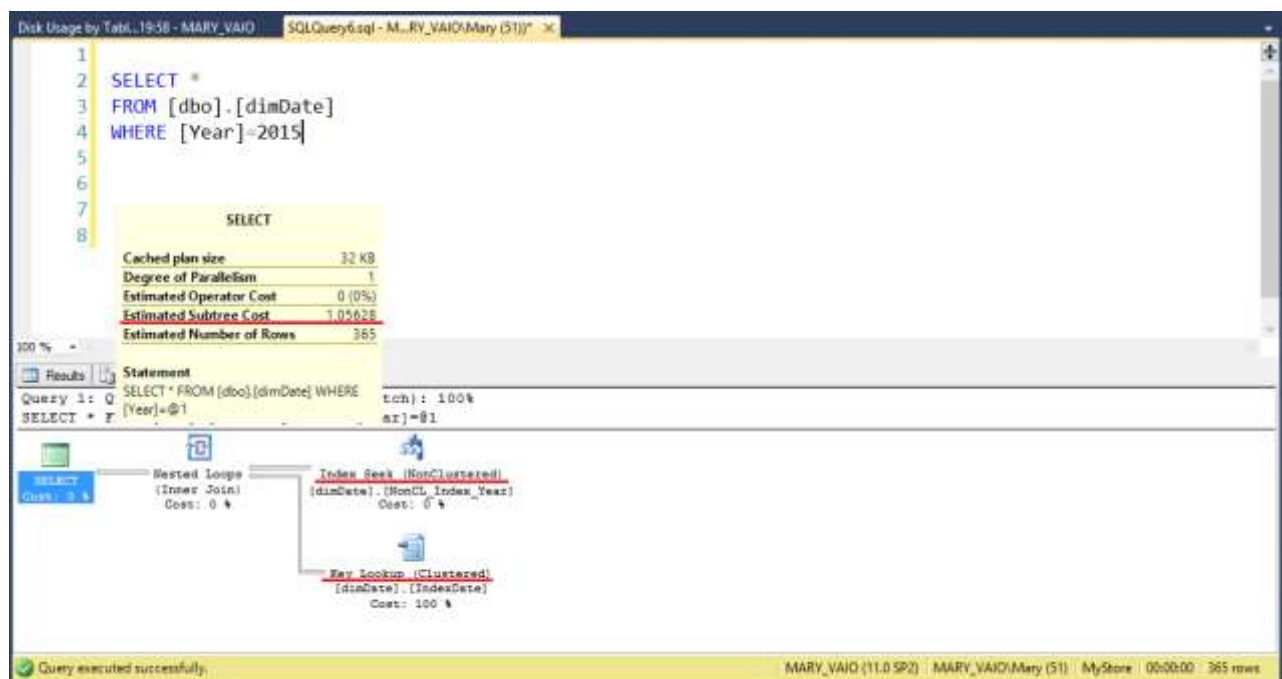


Рис. 41. Поиск по низкоселективному столбцу «Year»
некластеризованного индекса таблицы «dimDate»

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 ($RANGE_ROWS / DISTINCT_RANGE_ROWS$ для $DISTINCT_RANGE_ROWS > 0$).

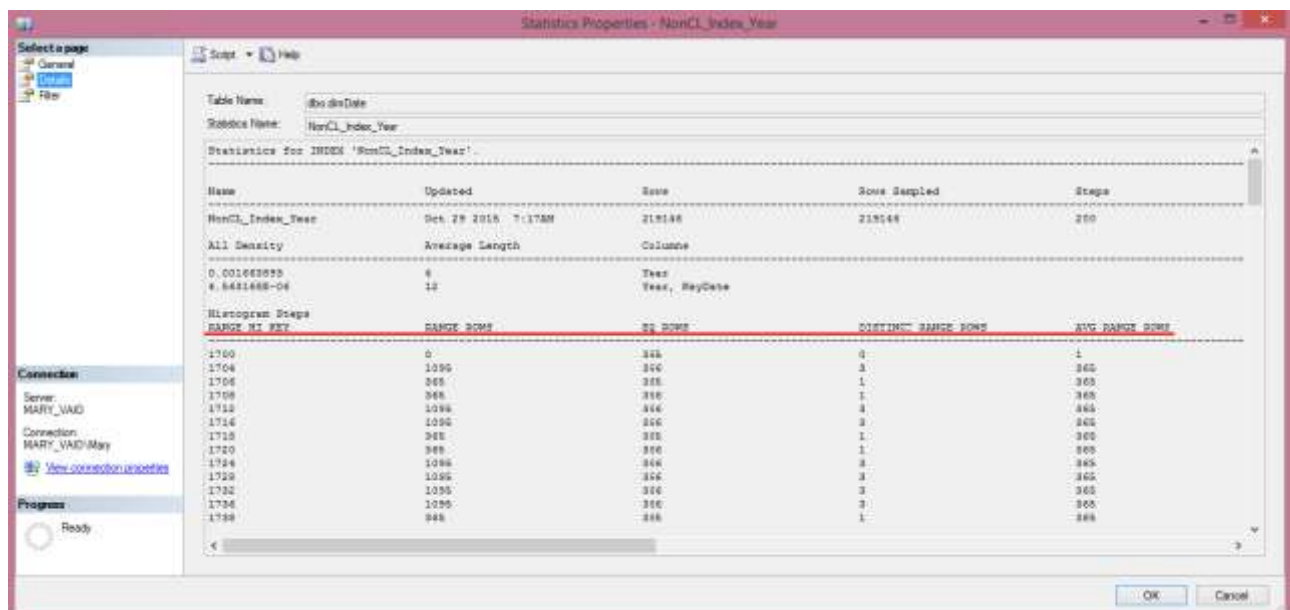


Рис. 43. Свойства статистики для некластеризованного индекса по полю «Year»

Более подробно о свойствах статистики можно узнать в электронной

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.)

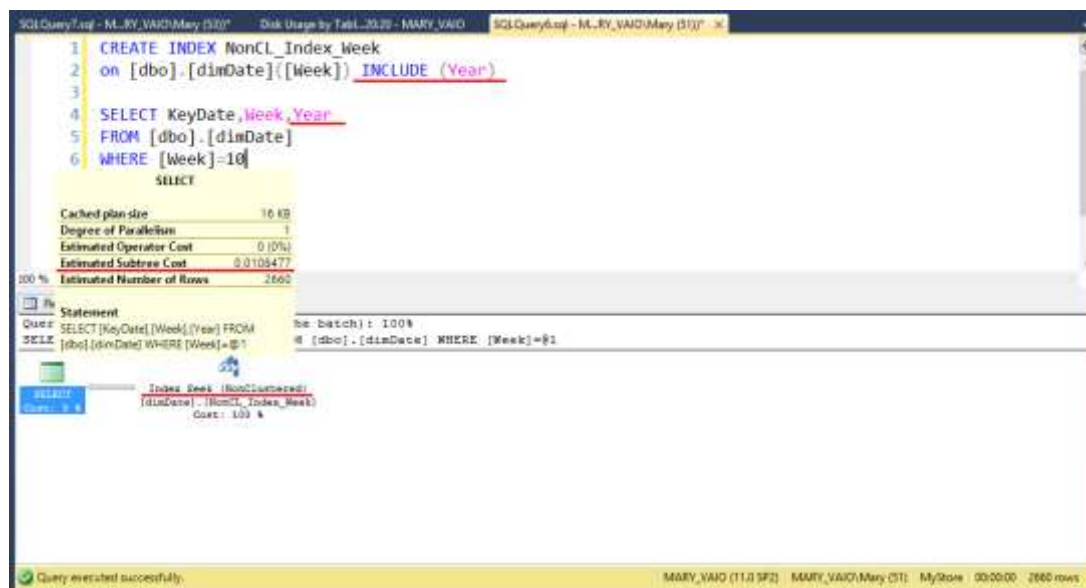


Рис. 44. Пример некластеризованного индекса с включаемым столбцом

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).

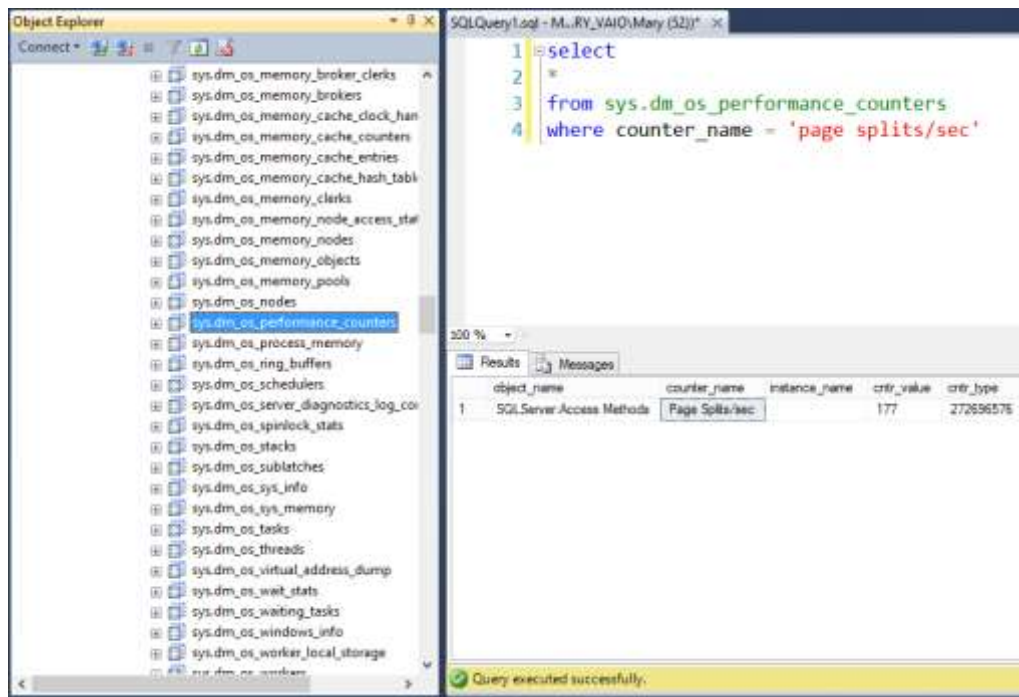



Рис. 45. Отслеживание количества расщеплений страниц в секунду

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	# Records	Reserved (KB)	Data (KB)	Indexes (KB)	Unused (KB)
dbo.FactSales	42 307	7 128	4 992	1 768	368
Индекс без фильтра					
Table Name	# Records	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]).

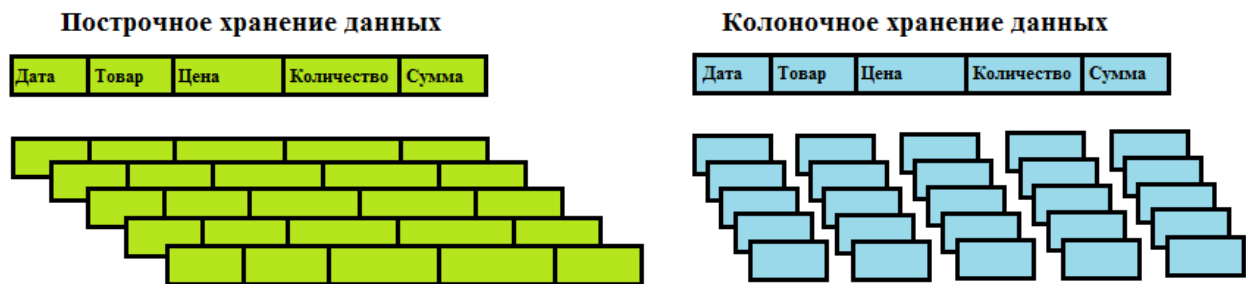


Рис. 47. Различные варианты хранения данных на диске

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.

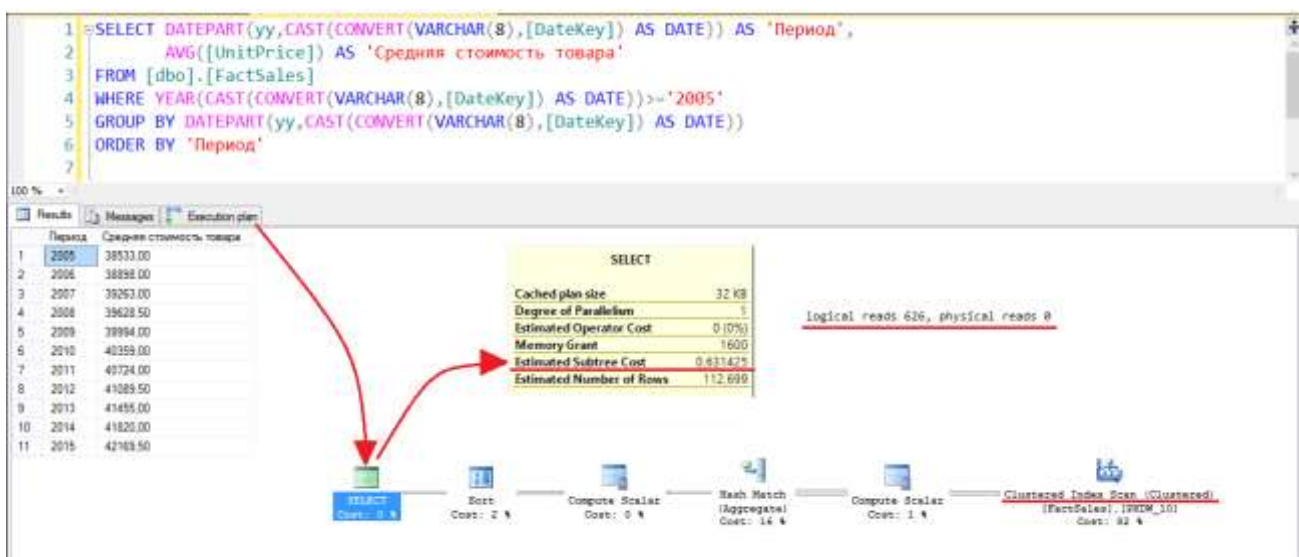


Рис. 48. Агрегирующий запрос к таблице без колоночного индекса

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).

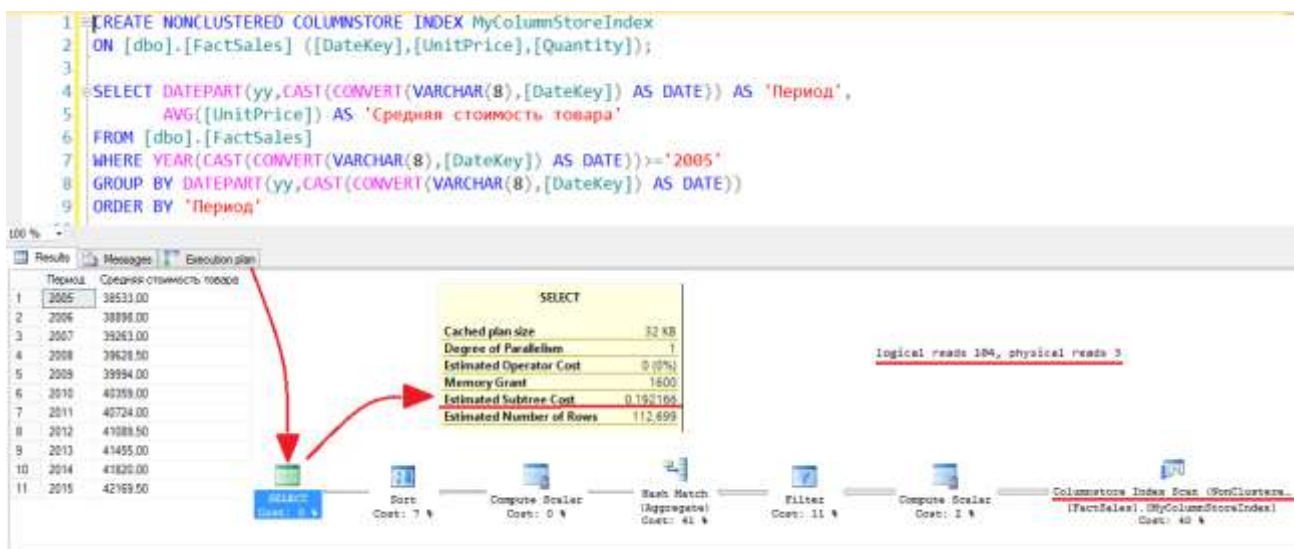


Рис. 49. Агрегирующий запрос к таблице с колоночным индексом

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_dimC	Некластеризованный	CustomerAlternateKey	Ключевое поле индекса.

ustomer_AlternateKey	индекс		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	Кластеризованн ый 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 индекс. Повышает производительность запросов, которые просматривают и агрегируют большие объемы данных	DateKey	Ключевое поле, связанное с 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	Кластеризованный индекс.	keyDelivery	Поле первичного ключа. 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.

3.2.3. Секционирование

Таблицы фактов обычно содержат очень большой объем данных, поэтому управление ими может вызывать затруднения. Вставка больших объемов данных и возможное перестроение индексов может занимать значительное время, к тому же может потребоваться перезагрузить ранее загруженные данные, а для этого нужно удалить часть данных из хранилища, что тоже требует временных затрат.

Секционирование позволяет упростить управление большой таблицей за счет горизонтального разбиения её на секции, каждая из которых хранится в

отдельной файловой группе (рис. 50).

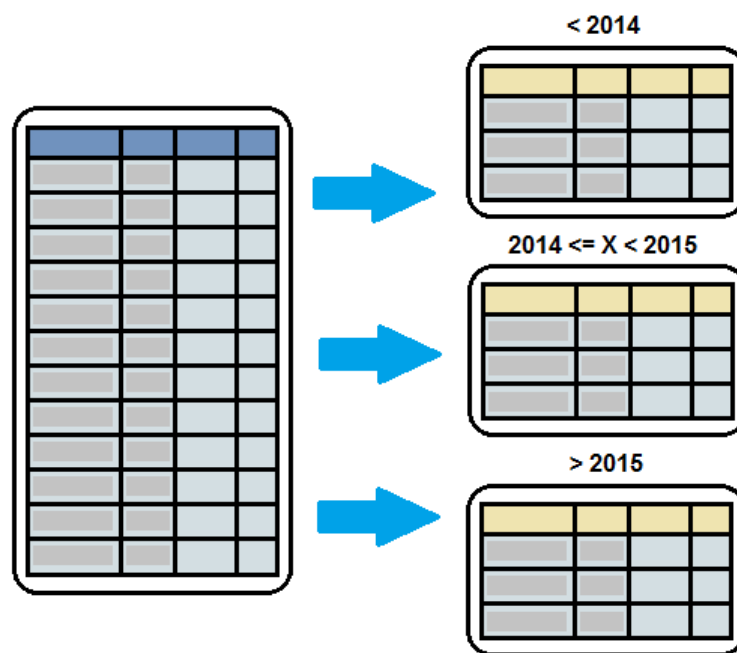


Рис. 50. Секционирование таблицы на основе столбца дат

Секционирование имеет следующие преимущества:

- ✓ увеличение производительности запросов за счет распараллеливания операций считывания с разных дисков, на которых хранятся секции;
- ✓ ускорение загрузки и удаления данных в хранилище, при использовании метода скользящего окна;
- ✓ использование более гибких вариантов резервного копирования и восстановления по секциям;
- ✓ уменьшение диапазона блокировок;
- ✓ возможность управлять индексами на уровне секции, например, восстанавливать индекс для одной секции, где данные были изменены.

Этот механизм доступен только в выпусках SQL Server Enterprise Edition, Developer Edition и Evaluation Edition.

Начиная с 2005 версии, SQL Server поддерживает автоматическое секционирование. Для пользователей таблица фактов представляется единой и неделимой, а сервер сам поддерживает заданную схему секционирования и распределяет данные по секциям, определяя необходимость объединения секций для обслуживания пользовательских запросов.

Для реализации секционирования нужно выполнить следующие шаги.

- ✓ Выбрать ключ секционирования, т.е. поле, по которому будут определяться границы секций.
- ✓ Создать функцию секционирования (Partition function), указывающую границы секций.
- ✓ Затем создать схему секционирования, управляющую размещением секций в файловой системе с использованием файловых групп.

В качестве ключа секционирования можно выбрать одно любое поле таблицы, в том числе и вычисляемое, за исключением полей следующих типов: timestamp, ntext, text, image, xml, varchar(max), nvarchar(max) и varbinary(max).

Если секционирование задается на кластеризованной таблице, то ключ секционирования должен быть частью первичного ключа или кластеризованного индекса.

Как правило, секционирование проводят на основе ключевого поля даты. Интервал (день, месяц, год и т.д.) выбирается исходя из потребностей пользователей. В SQL Server 2012 таблица может иметь до 15 000 секций.

Ниже приведен пример разделения таблицы фактов «FactSales» на три секции: продажи за текущий год, продажи за предыдущий год и продажи за все остальные годы.

Сначала создается функция секционирования:

```
CREATE PARTITION FUNCTION PartFunctionFactSales_Date (bigint)  
AS RANGE RIGHT FOR VALUES (20140101, 20150101)
```

, где PartFunctionFactSales_Date – название функции секционирования, bigint — тип данных ключа секционирования, в скобках указаны границы для секций. RANGE LEFT | RIGHT – необязательный параметр, который указывает какая граница интервала будет открытой, а какая замкнутой. LEFT – открытая граница слева, RIGHT – открытая граница справа. Если значение не задано, то по умолчанию используется значение LEFT.

Область действия функции секционирования ограничена базой данных, в которой она была создана. Посмотреть список всех функций секционирования для

конкретной базы данных можно в обозревателе объектов, пройдя по узлам дерева: **База данных -> Хранилище (Storage)- > Функции секционирования (Partition Functions)**).

Приведенная выше функция распределит строки данных таблицы фактов по секциям следующим образом:

Секция	1	2	3
Интервал значений	Ключ даты <20140101	20140101<= Ключ даты <20150101	20150101<= Ключ даты

Если бы не был указан параметр RIGHT, то интервалы значений выглядели бы так:

Секция	1	2	3
Интервал значений	Ключ даты <=20140101	20140101< Ключ даты <=20150101	20150101< Ключ даты

Затем необходимо задать схему секционирования. Это позволит привязать каждую секцию к определенной файловой группе и разместить их на разных дисках.

```
CREATE PARTITION SCHEME PartSchFactSales_Date
AS PARTITION PartFunctionFactSales_Date TO
([Fast_Growing],
[Frequently_Requested],
[Frequently_Requested])
```

Схема секционирования обязательно опирается на определенную функцию секционирования и в скобках перечисляются те файловые группы, к которым следует привязать каждую секцию. В данном примере продажи за текущий и предыдущий годы помещаются в файловую группу для наиболее часто запрашиваемых данных, а продажи за все остальные годы в файловую группу для быстро растущих таблиц.

Посмотреть список всех схем секционирования можно в обозревателе объектов, пройдя по узлам дерева: **База данных -> Хранилище (Storage)- > Схема секционирования (Partition Schemes)**.

Очевидно, что файловые группы, функция и схема секционирования должны существовать в БД до того, как будет создана таблица.

Ниже приведен пример кода создания секционированной таблицы фактов «FactSales», которая логически представляется в базе данных как одна таблица.

При создании такой таблицы вместо указания файловой группы указывается схема секционирования и столбец, по которому будет проведено разбиение. Если в таблице задан первичный ключ, то в его составе должно быть поле ключа секционирования.

```
CREATE TABLE FactSales
(NumberSale INT NOT NULL,
NumberLineInSale INT NOT NULL,
CustomerKey INT NOT NULL,
CustomerDemographyKey INT,
ProductKey INT NOT NULL,
DateKey BIGINT NOT NULL,
ManagerKey INT,
KindOfPayKey INT,
PromotionKey INT,
Quantity INT NOT NULL,
UnitPrice DECIMAL(15,2) NOT NULL,
Subtotal DECIMAL(15,2) NOT NULL,
DiscountCustAmt DECIMAL(15,2),
PromoAmt DECIMAL(15,2),
TaxAmt DECIMAL(15,2) NOT NULL,
Freight DECIMAL(15,2),
Total DECIMAL(15,2) NOT NULL,
CONSTRAINT PKDW_10 PRIMARY KEY (NumberSale,NumberLineInSale,DateKey)
) ON PartSchFactSales_Date(DateKey)
```

Информация о секциях хранится в специальной системной таблице Sys.Partitions, посмотреть которую можно выполнив запрос, представленный на рис. 51.

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Object Explorer' pane displays the database structure for 'MARY_VAIO (SQL Server 11.0.5058 - 11.0.5058)'. The 'Tables' folder is expanded, showing 'dbo.FactSales'. The main window displays a query executed against 'PartitionFactTable_MARY_VAIO/Mary (530)'. The query is:
 1 SELECT * FROM Sys.Partitions
 2 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'FactSales')
 3
 4
 5
 The 'Results' pane shows the following data:

partition_id	object_id	index_id	partition_number	hobt_id	rows	filestream_filegroup_id	data_compression	data_compression_desc
1	72057594042515456	226099846	1	72057594042515456	1096	0	0	NONE
2	72057594042580992	226099846	2	72057594042580992	365	0	0	NONE
3	72057594042548528	226099846	3	72057594042548528	181	0	0	NONE

The status bar at the bottom indicates 'Query executed successfully.' and 'MARY_VAIO (11.0 SP2) MARY_VAIO/Mary (53)'.

Рис. 51. Просмотр информации о секциях таблицы «FactSales»

Секции нумеруются автоматически, начиная с единицы (столбец `partition_number`). Первая секция содержит самые ранние данные. В столбце `rows` показано количество строк данных в каждой секции, а столбец `index_id` показывает, задан ли кластеризованный индекс на таблице, 0 – нет, 1 – да.

Для каждой секции строится отдельное индексное дерево и, следовательно, ускорение запросов на секционированной таблице происходит, если указать поле ключа секционирования в условии `WHERE`. В этом случае просматривается только определенная секция, а не вся таблица целиком.

На рис. 52 показан пример выполнения запроса, затрагивающего только одну секцию таблицы (`Actual partition count = 1`), хранящую данные за 2011 год.

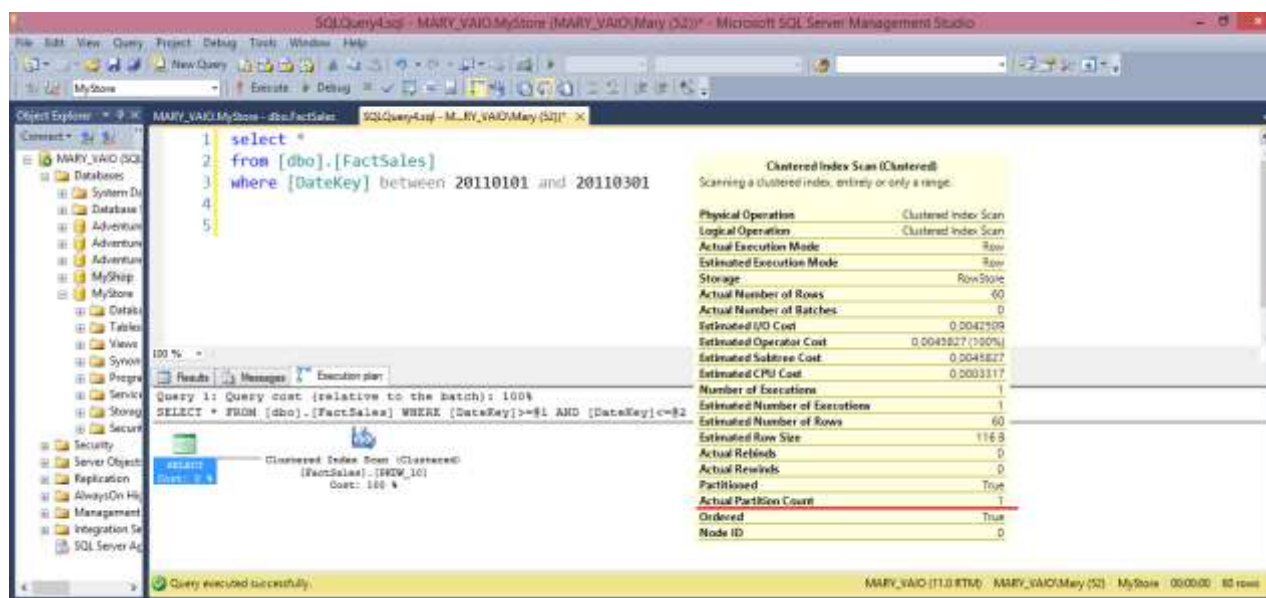


Рис. 52. Запрос, затрагивающий только одну секцию таблицы «FactSales»

Помимо секционирования таблиц (кучи или кластеризованного индекса) можно также секционировать индексы, которые в этом случае могут быть двух видов.

Выровненный индекс – индекс, созданный на основе той же схемы секционирования, что и соответствующая таблица. Если таблица и ее индексы выровнены, SQL Server может быстро переключаться с секции на секцию, сохраняя при этом структуру секций как таблицы, так и ее индексов. Для выравнивания с базовой таблицей индексу необязательно использовать функцию

секционирования с тем же именем. Однако функции секционирования индекса и базовой таблицы не должны существенно различаться, то есть:

- ✓ аргументы функции секционирования должны иметь один и тот же тип данных;
- ✓ функции должны определять одинаковое количество секций;
- ✓ функции должны определять для секций одинаковые граничные значения.

Невыровненный индекс – индекс, секционированный независимо от соответствующей таблицы, т. е. имеющий другую схему секционирования или находящийся в другой файловой группе, нежели базовая таблица.

В любом случае, при секционировании индексов следует придерживаться следующих правил:

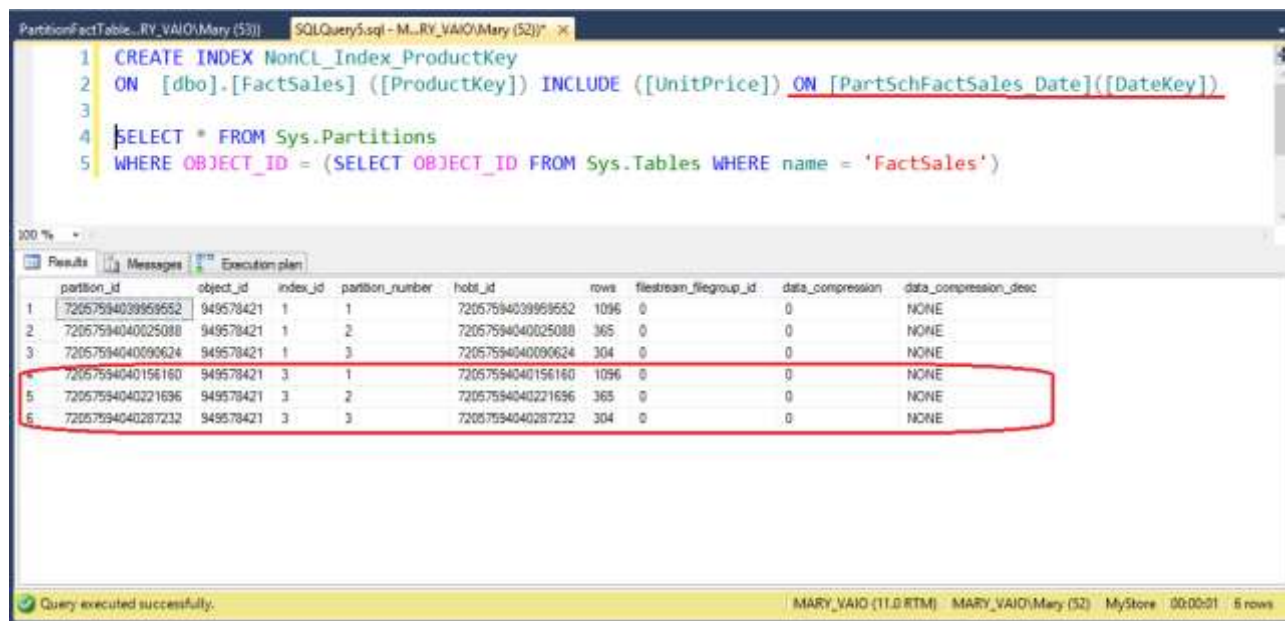
- ✓ для секционирования уникальных индексов (кластеризованного или некластеризованного) столбец секционирования должен содержаться в ключе индекса;
- ✓ для секционирования неуникального кластеризованного индекса, если столбец секционирования не указан явно в ключе кластеризации, SQL Server по умолчанию добавляет столбец секционирования в список ключей кластеризованного индекса;
- ✓ для секционирования неуникального некластеризованного индекса SQL Server по умолчанию добавляет столбец секционирования как неключевой (включенный) столбец индекса, чтобы обеспечить выравнивание индекса с базовой таблицей. Если столбец секционирования уже присутствует в индексе, SQL Server его не добавляет.

Ниже приведен пример создания секционированного выровненного неуникального некластеризованного индекса для таблицы «FactSales» при этом столбец секционирования «DateKey» явно в индекс не входит, а добавляется SQL Server автоматически.

```
CREATE INDEX NonCL_Index_ProductKey  
ON [dbo].[FactSales] ([ProductKey]) INCLUDE ([UnitPrice]) ON  
[PartSchFactSales_Date]([DateKey])
```

Теперь посмотрим на вновь образованные секции для таблицы «FactSales» и

её индексов (рис. 53). В столбце «Index_id» значение 1 – задан кластеризованный индекс, 3 – некластеризованный индекс.



```
1 CREATE INDEX NonCL_Index_ProductKey
2 ON [dbo].[FactSales] ([ProductKey]) INCLUDE ([UnitPrice]) ON [PartSchFactSales_Date]([DateKey])
3
4 SELECT * FROM Sys.Partitions
5 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'FactSales')
```

partition_id	object_id	index_id	partition_number	hobit_id	rows	filestream_filegroup_id	data_compression	data_compression_desc
1	72057594039959552	949578421	1	72057594039959552	1096	0	0	NONE
2	72057594040025088	949578421	1	72057594040025088	365	0	0	NONE
3	72057594040090624	949578421	1	72057594040090624	304	0	0	NONE
4	72057594040156160	949578421	3	72057594040156160	1096	0	0	NONE
5	72057594040221696	949578421	3	72057594040221696	365	0	0	NONE
6	72057594040287232	949578421	3	72057594040287232	304	0	0	NONE

Рис. 53. Секции таблицы «FactSales»

При работе с секциями нужно уметь выполнять следующие операции.

- ✓ Переключение секции (switch partition), т.е. переприсваивание блоков данных из одной таблицы или секции другой таблице или секции. Эта процедура выполняется с помощью инструкции T-SQL - ALTER TABLE.

Можно переключать:

- все данные из несекционированной таблицы на имеющуюся пустую секцию секционированной таблицы;
- секции одной секционированной таблицы на секции другой секционированной таблицы;
- все данные из секции секционированной таблицы на существующую пустую несекционированную таблицу.

При этом все таблицы, секции, соответствующие индексы или секции индексов должны быть расположены в одной файловой группе, для того чтобы избежать физического перемещения данных между дисками. В этом случае это операция только над метаданными.

Операцию переключения между секциями часто применяют при загрузке хранилища, используя две вспомогательные неиндексированные

таблицы с той же структурой, что и секционированная таблица для вставки или удаления больших порций данных с минимальным протоколированием. При вставке данные добавляются во вспомогательную таблицу, а затем переключаются на пустую секцию основной таблицы. Это позволяет не блокировать основную таблицу на время загрузки данных, переключение между секциями занимает минимальное время. При удалении секция переключается на пустую таблицу, а затем данные из неё удаляются.

- ✓ Разделение секции (split partition). Позволяет разделить существующую секцию на две. Эта процедура выполняется с помощью инструкции T-SQL - `ALTER PARTITION FUNCTION {SPLIT RANGE (boundary_value)}`. Аргумент `boundary_value` определяет диапазон новой секции, который должен отличаться от существующих пограничных значений функции секционирования. Для того чтобы назначить файловую группу для вновь создаваемых секций, нужно изменить схему секционирования, установив для файловой группы свойство `NEXT USED` с помощью инструкции `ALTER PARTITION SCHEME partition_scheme_name NEXT USED [filegroup_name]`.
- ✓ Слияние двух секций (merge partition). Удаляет секцию и объединяет все значения, существующие в секции, в одну из оставшихся. Эта процедура также выполняется с помощью инструкции T-SQL - `ALTER PARTITION FUNCTION {MERGE RANGE (boundary_value)}`. Аргумент `boundary_value` должен быть существующим пограничным значением для двух объединяемых секций.

Хорошей практикой является резервирование пустых секций в начале и в конце диапазона секций, для того чтобы избежать перемещения данных при разбиении секций (до загрузки новых данных) и их объединении (после выгрузки старых данных).

Это особенно важно при реализации метода *скользящего окна* (sliding window), позволяющего динамически секционировать таблицы.

С помощью этого метода можно не только физически, но и логически (явно

для пользователя) разделить данные из таблицы фактов, перенося устаревшие в архивную таблицу или удаляя, в зависимости от потребностей бизнеса.

Дело в том, что при секционировании логически таблица представляется единой и это накладывает свои ограничения, такие как необходимость поддерживать одинаковые индексы на всех секциях или наложение блокировок на всю таблицу при определенных действиях, поэтому иногда бывает полезным явно разделить данные.

Для реализации метода необходимо создать архивную таблицу с такой же структурой, как и секционированная таблица и выполнить следующие шаги.

- ✓ При создании новой секции, куда будут загружаться свежие данные, выполнить операцию SPLIT на последней пустой секции таблицы фактов. Выполнить ту же операцию для подготовки новой пустой секции в архивной таблице.
- ✓ Для выгрузки устаревших данных нужно переключить секцию, их содержащую, в пустую секцию архивной таблицы, и выполнить операцию MERGE на этой секции, ту же операцию выполнить в архивной таблице, объединив пустую и только что загруженную секции. При этом физического перемещения данных не происходит, если секции расположены в одной файловой группе.
- ✓ создать вспомогательную таблицу без индексов для загрузки новых данных, затем загрузить в неё данные и построить индексы для подготовки к переключению в новую секцию основной таблицы.

На рис. 54 подробно показан процесс перестройки секций при реализации «скользящего окна».

Таблица фактов

пустая секция < 2013	2013	2014	2015	пустая секция 2016
-------------------------	------	------	------	-----------------------

Архивная таблица

пустая секция < 2013	пустая секция 2013
-------------------------	-----------------------

1) Добавление новой секции (SPLIT)

Таблица фактов

пустая секция < 2013	2013	2014	2015	пустая секция 2016	пустая секция 2017
-------------------------	------	------	------	-----------------------	-----------------------

Архивная таблица

пустая секция < 2013	пустая секция 2013	пустая секция 2014
-------------------------	-----------------------	-----------------------

2) Перенос данных за 2013 год в архивную таблицу (SWITCH)

Таблица фактов

пустая секция < 2013	2013	2014	2015	пустая секция 2016	пустая секция 2017
-------------------------	------	------	------	-----------------------	-----------------------



Архивная таблица

пустая секция < 2013	пустая секция 2013	пустая секция 2014
-------------------------	-----------------------	-----------------------

3) Слияние первых двух секций (MERGE)

Таблица фактов

пустая секция 2013	2014	2015	пустая секция 2016	пустая секция 2017
-----------------------	------	------	-----------------------	-----------------------

Архивная таблица

2013	пустая секция 2014
------	-----------------------

Рис. 54. Реализация метода «скользящего окна»

Можно создать хранимую процедуру без параметров, которая будет запускаться по расписанию через SQL Agent и позволит автоматически переключать секции.

Ниже приведен пример процедуры для создания «скользящего окна» с шагом в один год и пятью секциями для таблицы «FactSales». В таблице должны храниться данные только за текущий и два предыдущих года, а данные за более ранние года перемещаются в архивную таблицу «ArchivalFactSales», имеющую ту

же структуру, что и таблица «FactSales». В начале и в конце диапазона зарезервированы пустые секции, для быстрого разделения и объединения секций.

1. Создание функций и схем секционирования для таблиц «FactSales» и «ArchivalFactSales».

```
CREATE PARTITION FUNCTION PartFunctionFactSales_Date (bigint)
AS RANGE RIGHT FOR VALUES (20130101,20140101,20150101,20160101)
```

```
CREATE PARTITION SCHEME PartSchFactSales_Date
AS PARTITION PartFunctionFactSales_Date TO
([Fast_Growing],
[Fast_Growing],
[Frequently_Requested],
[Frequently_Requested],
[Frequently_Requested])
```

```
CREATE PARTITION FUNCTION PartFunctionForArchivalTable (bigint)
AS RANGE RIGHT FOR VALUES (20130101)
```

```
CREATE PARTITION SCHEME PartSchForArchivalTable
AS PARTITION PartFunctionForArchivalTable TO
([Fast_Growing],
[Fast_Growing])
```

2. Создание таблиц «FactSales» и «ArchivalFactSales».

```
CREATE TABLE FactSales
(NumberSale INT NOT NULL,
NumberLineInSale INT NOT NULL,
CustomerKey INT NOT NULL,
CustomerDemographyKey INT,
ProductKey INT NOT NULL,
DateKey BIGINT NOT NULL,
ManagerKey INT,
KindOfPayKey INT,
PromotionKey INT,
Quantity INT NOT NULL,
UnitPrice DECIMAL(15,2) NOT NULL,
Subtotal DECIMAL(15,2) NOT NULL,
DiscountCustAmt DECIMAL(15,2),
PromoAmt DECIMAL(15,2),
TaxAmt DECIMAL(15,2) NOT NULL,
Freight DECIMAL(15,2),
Total DECIMAL(15,2) NOT NULL,
CONSTRAINT PKDW_10 PRIMARY KEY (NumberSale,NumberLineInSale,DateKey)
) ON PartSchFactSales_Date(DateKey)
GO
CREATE TABLE ArchivalFactSales
(NumberSale INT NOT NULL,
NumberLineInSale INT NOT NULL,
```

```

CustomerKey INT NOT NULL,
CustomerDemographyKey INT,
ProductKey INT NOT NULL,
DateKey BIGINT NOT NULL,
KindOfPayKey INT,
PromotionKey INT,
Quantity INT NOT NULL,
UnitPrice DECIMAL(15,2) NOT NULL,
Subtotal DECIMAL(15,2) NOT NULL,
DiscountCustAmt DECIMAL(15,2),
PromoAmt DECIMAL(15,2),
TaxAmt DECIMAL(15,2) NOT NULL,
Freight DECIMAL(15,2),
Total DECIMAL(15,2) NOT NULL,
CONSTRAINT PK_Arch_1 PRIMARY KEY (NumberSale, NumberLineInSale, DateKey)
) ON PartSchForArchivalTable(DateKey)

```

3. После заполнения таблицы «FactSales» данными выполним запросы к системной таблице Sys.Partitions и посмотрим на содержимое секций для таблиц «FactSales» и «ArchivalFactSales» (рис. 55).

```

66 SELECT partition_number, rows FROM Sys.Partitions
67 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'FactSales')
68
69 SELECT partition_number, rows FROM Sys.Partitions
70 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'ArchivalFactSales')
71 GO

```

partition_number	rows
1	0
2	365
3	365
4	334
5	0

partition_number	rows
1	0
2	0

Рис. 55. Секции, изначально созданные для таблиц «FactSales» и «ArchivalFactSales»

Для таблицы «FactSales» образовано 5 секций. Данные содержатся в секциях за 2013, 2014 и 2015 год. Секция для более ранних данных и за 2016 год данных не содержат. Для таблицы «ArchivalFactSales» образованы две пустые секции.

4. Процедура реализации «скользящего окна».

```

CREATE PROCEDURE Pr_SlidingWindow
AS
DECLARE @DayForPartFactSales VARCHAR(8)
DECLARE @DayForPartArchival VARCHAR(8)

```

Находим текущую верхнюю границу секций.

```

SET @DayForPartFactSales = CAST((SELECT TOP 1 [value] FROM
sys.partition_range_values
    WHERE function_id = (SELECT function_id
        FROM sys.partition_functions
        WHERE name = 'PartFunctionFactSales_Date')
    ORDER BY boundary_id DESC) AS VARCHAR(8))

```

Находим текущую нижнюю границу секций.

```

SET @DayForPartArchival = CAST((SELECT TOP 1 [value] FROM
sys.partition_range_values
    WHERE function_id = (SELECT function_id
        FROM sys.partition_functions
        WHERE name = 'PartFunctionFactSales_Date')
    ORDER BY boundary_id ASC) AS VARCHAR(8))

```

Определение граничных значений для новых секций. Значение года увеличивается на единицу.

```

DECLARE @Day_DT DATE
SET @Day_DT = DATEADD(YEAR, 1, CAST(@DayForPartFactSales AS DATE))

DECLARE @DayArchival_DT DATE
SET @DayArchival_DT = DATEADD(YEAR, 1, CAST(@DayForPartArchival AS DATE))

```

Назначение файловой группы для вновь создаваемых секций.

```

ALTER PARTITION SCHEME PartSchFactSales_Date
NEXT USED [Frequently_Requested];

ALTER PARTITION SCHEME PartSchForArchivalTable
NEXT USED [Fast_Growing]

```

Операция SPLIT на последней пустой секции таблицы фактов и архивной таблицы.

```

ALTER PARTITION FUNCTION PartFunctionFactSales_Date()
SPLIT RANGE (CAST(CONVERT (VARCHAR(8),@Day_DT, 112) AS BIGINT))

ALTER PARTITION FUNCTION PartFunctionForArchivalTable()
SPLIT RANGE (CAST(CONVERT (VARCHAR(8),@DayArchival_DT, 112) AS BIGINT))

```

После выполнения этих операций для таблицы «FactSales» будет добавлена еще одна секция для данных за 2017 год, а для таблицы «ArchivalFactSales» секция для данных за 2014 год (рис. 56).


```

130 SELECT partition_number, rows FROM Sys.Partitions
131 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'FactSales')
132
133 SELECT partition_number, rows FROM Sys.Partitions
134 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'ArchivalFactSales')

```

	partition_number	rows
1	1	0
2	2	365
3	3	365
4	4	334
5	5	0
6	6	0

	partition_number	rows
1	1	0
2	2	0
3	3	0

Рис. 56. Секции для таблиц «FactSales» и «ArchivalFactSales», образованные после выполнения операции SPLIT

Перемещение данных из секции таблицы «FactSales» в секцию таблицы «ArchivalFactSales» (рис. 57).

```

ALTER TABLE FactSales
SWITCH PARTITION 2
TO ArchivalFactSales PARTITION 2

```

```

153 SELECT partition_number, rows FROM Sys.Partitions
154 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'FactSales') ORDER BY partition_number
155
156 SELECT partition_number, rows FROM Sys.Partitions
157 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'ArchivalFactSales') ORDER BY
partition_number

```

	partition_number	rows
1	1	0
2	2	0
3	3	365
4	4	334
5	5	0
6	6	0

	partition_number	rows
1	1	0
2	2	365
3	3	0

Рис. 57. Перемещение данных между секциями таблиц «FactSales» и «ArchivalFactSales»

Объединение секции, из которой данные были перенесены в архивную таблицу, с ближайшей справа секцией таблицы «FactSales» и объединение пустой и только что загруженной секции архивной таблицы.

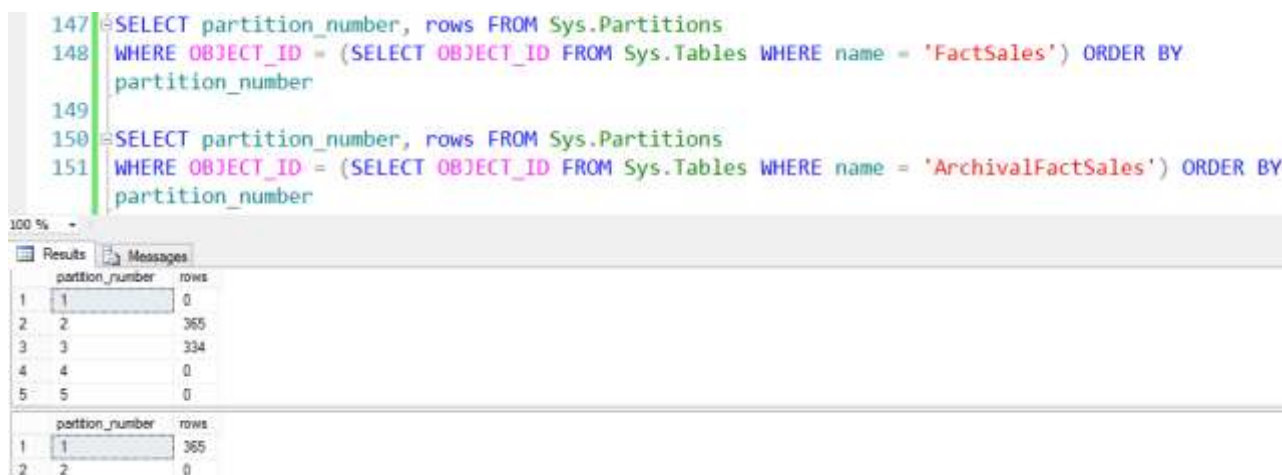
```

ALTER PARTITION FUNCTION PartFunctionFactSales_Date()
MERGE RANGE (CAST(@DayForPartArchival AS BIGINT));

ALTER PARTITION FUNCTION PartFunctionForArchivalTable()
MERGE RANGE (CAST(@DayForPartArchival AS BIGINT));

```

Содержимое секций для таблиц «FactSales» и «ArchivalFactSales» после выполнения процедуры показано на рис. 58.



```
147 SELECT partition_number, rows FROM Sys.Partitions
148 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'FactSales') ORDER BY
partition_number
149
150 SELECT partition_number, rows FROM Sys.Partitions
151 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'ArchivalFactSales') ORDER BY
partition_number
```

partition_number	rows
1	0
2	365
3	334
4	0
5	0

partition_number	rows
1	365
2	0

Рис. 58. Результат работы процедуры «Pr_SlidingWindow»

На секционированных таблицах также может быть задан колоночный индекс, если столбец секционирования является одним из столбцов этого индекса, но поскольку в таблицу, где задан колоночный индекс, нельзя загружать данные, то алгоритм загрузки данных в секционированную таблицу с колоночным индексом будет выглядеть следующим образом:

- ✓ Создать временную таблицу с той же структурой, что и таблица в которую необходимо загрузить данные.
- ✓ Загрузить данные во временную таблицу.
- ✓ Построить на временной таблице колоночный индекс.
- ✓ Переключить временную таблицу (SWITCH) на секцию основной таблицы.

3.2.4. Сжатие таблиц

Начиная с 2008 версии, SQL Server позволяет выполнять компрессию строк и страниц таблиц и индексов для экономии дискового пространства, но только в редакции Enterprise. Однако для сжатия и распаковки данных при обмене данными с приложениями требуются дополнительные ресурсы ЦП на сервере баз данных (см. [2]). Для пользователя работа с такой таблицей ничем не отличается от работы с обычной несжатой таблицей.

Сжатие можно применять к следующим объектам базы данных: таблице, хранящейся в виде кучи или кластеризованного индекса; некластеризованному

индексу; некластеризованному представлению; секционированной таблице и индексу, причем параметры сжатия можно задать для каждой секции, и разные секции объекта могут иметь различные параметры.

Параметры сжатия таблицы не применяются автоматически к ее некластеризованным индексам. Для системных таблиц сжатие недоступно.

Возможно два типа сжатия.

ROW-сжатие строк является имитацией компрессии за счет уменьшения объема дополнительных метаданных и сохранения столбцов с фиксированным типом данных в формате переменной длины (строки и числовые данные). Например, символы Unicode занимают два байта. Сжатие Unicode (Unicode compression) отводит один байт на хранение тех символов, которым на самом деле не нужны два байта.

Алгоритм сжатия строк немного уменьшает объем данных и слабо влияет на загрузку процессора.

PAGE-сжатие страниц включает сжатие строк, но добавляет сжатие префиксов и словарей. При сжатии префикса (prefix compression) повторяющиеся префиксы значений из одного столбца сохраняются в специальной структуре сжатой информации (compression information, CI), которая располагается сразу за заголовком страницы и повторяющиеся префиксные значения заменяются ссылкой на соответствующий префикс. При сжатии словаря (dictionary compression) повторяющиеся значения, встретившиеся в любом месте страницы, сохраняются в структуре CI. Сжатие словаря не ограничивается значениями в одном столбце (см.[4]).

Этот способ сжатия более эффективен, чем сжатие строк, но вызывает большие нагрузки на ЦП, поэтому не всегда может быть применен.

Применение механизмов сжатия данных в хранилище является полезной практикой, так как данные в денормализованных таблицах очень часто повторяются, в том числе и внешние ключи в таблице фактов. Таблицы измерения со строками Unicode могут выиграть от сжатия Unicode.

Помимо экономии места, сжатие данных позволяет повысить

производительность при рабочих нагрузках с интенсивным вводом-выводом, поскольку данные хранятся в меньшем количестве страниц и в запросах требуется считывать меньше страниц с диска.

Ниже приведен пример применения механизма сжатия к секционированной таблице фактов «FactSales».

Сравним показатели загруженности ЦП и операций чтения с диска и из КЭШ на запросе к несжатой и сжатой таблице. Для просмотра статистики нужно включить соответствующие опции. На рис. 59 показаны значения для несжатой таблицы. Если параметр `physical reads = 0`, значит все данные таблицы загружены в КЭШ.



```
1 SET STATISTICS TIME ON
2 SET STATISTICS IO ON
3
4 SELECT * FROM [dbo].[FactSales]
```

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 5 ms.

(42387 row(s) affected)
Table 'FactSales', Scan count 5, logical reads 633, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

SQL Server Execution Times:
CPU time = 31 ms, elapsed time = 1891 ms.

Рис. 59. Статистика запроса к несжатой таблице «FactSales»

Сжатие таблицы можно задать с помощью встроенного мастера SQL Server Management Studio, выбрав из контекстного меню таблицы пункт **Хранилище** (Storage) -> **Управление сжатием** (Manage Compression). Так как таблица «FactSales» секционирована, то можно задать режим сжатия для каждой отдельной секции (рис. 60).

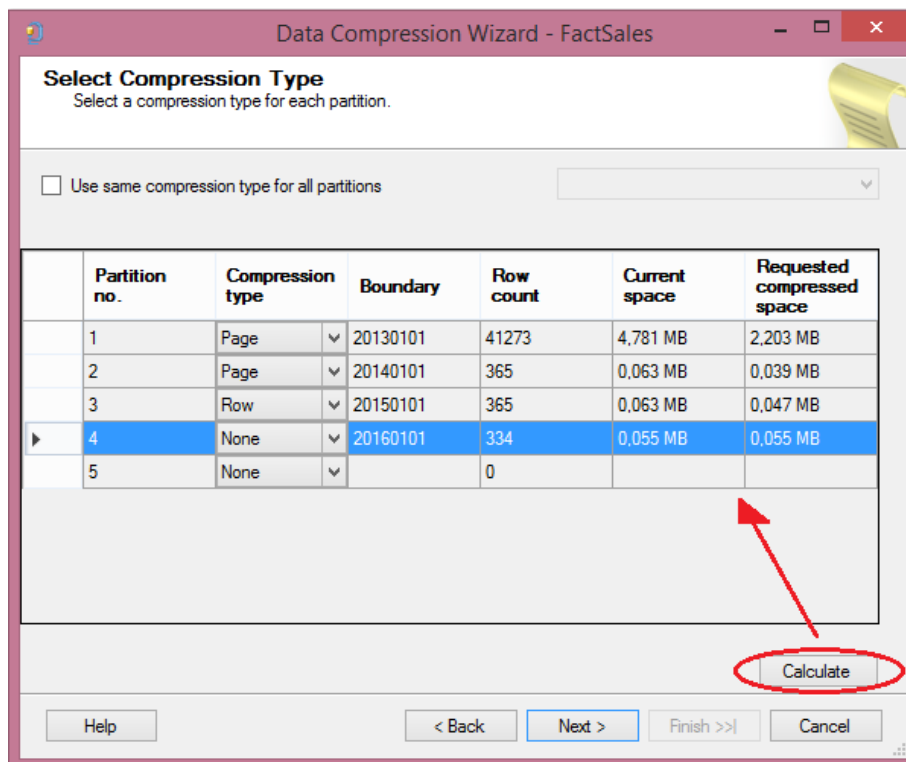


Рис 60. Мастер сжатия данных для таблицы «FactSales»

Для первых двух секций, где хранятся данные за прошедшие года, применяется режим сжатия PAGE, для секции данных за текущий год режим ROW. При нажатии на кнопку **Рассчитать** (Calculate) в последних двух столбцах показываются текущий объем дискового пространства, занимаемый секциями таблицы, и предполагаемый объем после сжатия. Как видно из рисунка, сжатие PAGE существенно уменьшает объем данных.

Выполним тот же запрос на сжатой таблице и посмотрим статистику (рис. 61).

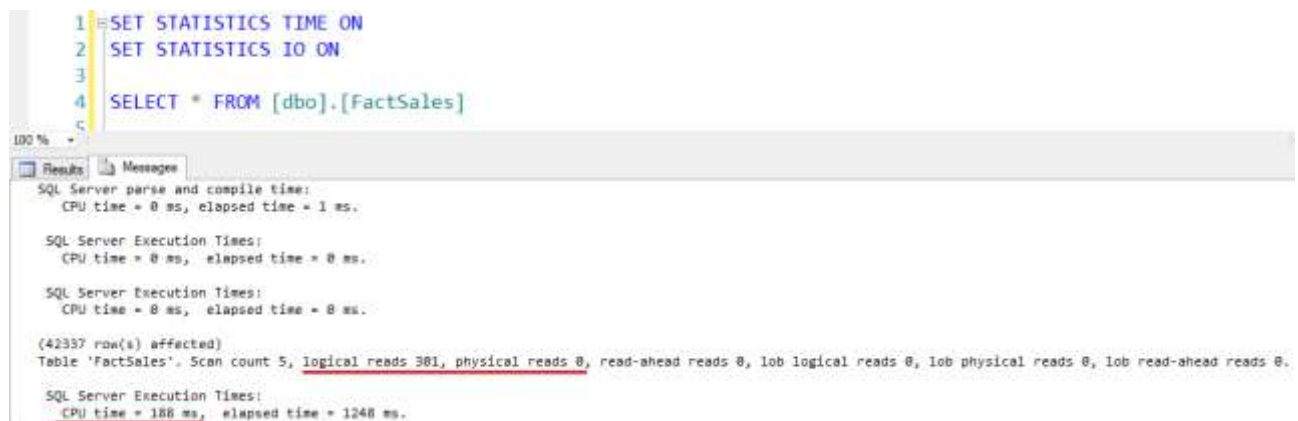


Рис. 61. Статистика запроса к сжатой таблице «FactSales»

Как видно из рисунка, количество операций чтения сократилось вдвое, но и

загруженность ЦП увеличилась.

Также режимы сжатия таблицы можно задать с помощью инструкций T-SQL. USE [MyStore]

```
ALTER TABLE [dbo].[FactSales] REBUILD PARTITION = 1 WITH(DATA_COMPRESSION = PAGE )
```

```
ALTER TABLE [dbo].[FactSales] REBUILD PARTITION = 2 WITH(DATA_COMPRESSION = PAGE )
```

```
ALTER TABLE [dbo].[FactSales] REBUILD PARTITION = 3 WITH(DATA_COMPRESSION = ROW )
```

```
ALTER TABLE [dbo].[FactSales] REBUILD PARTITION = 4 WITH(DATA_COMPRESSION = NONE )
```

```
ALTER TABLE [dbo].[FactSales] REBUILD PARTITION = 5 WITH(DATA_COMPRESSION = NONE )
```

Отключить сжатие можно с помощью следующей команды:

```
USE [MyStore]
```

```
ALTER TABLE [dbo].[FactSales] REBUILD PARTITION = 3 WITH(DATA_COMPRESSION = NONE )
```

Следует отметить, что колоночные индексы поддерживают собственный патентованный алгоритм сжатия Microsoft **VertiPaq**, для них нельзя использовать сжатие строк или страниц. Это сжатие является более эффективным, чем сжатие типа PAGE и ROW. Колоночный индекс автоматически сжимается при создании и пользователи не могут влиять или контролировать сжатие колоночного индекса.

Более подробную информацию о механизмах сжатия можно найти в электронной документации по ссылке [https://msdn.microsoft.com/ru-ru/library/cc280449\(v=sql.110\).aspx](https://msdn.microsoft.com/ru-ru/library/cc280449(v=sql.110).aspx).