Практическая работа 16
Создание, перестройка и удаление индекса

Цель занятия: Получить практический опыт построения запросов на языке SQL

Перечень оборудования и программного обеспечения

Персональный компьютер
Microsoft Office (Word, Visio, Access)
Microsoft SQL Server Management Studio

Краткие теоретические сведения

Что такое индексы в базе данных?

Индекс — это объект базы данных, который представляет собой структуру данных, состоящую из ключей, построенных на основе одного или нескольких столбцов таблицы или представления, и указателей, которые сопоставляются с местом хранения заданных данных. Индексы предназначены для более быстрого получения строк из таблицы, другими словами, индексы обеспечивают быстрый поиск данных в таблице, что значительно повышает производительность запросов и приложений. Индексы также могут быть использованы и для обеспечения уникальности строк таблицы, гарантируя тем самым целостность данных.

Типы индексов в Microsoft SQL Server

В Microsoft SQL Server существуют следующие типы индексов:

- **Кластеризованный** (*Clustered*) – это индекс, который хранит данные таблицы в отсортированном, по значению ключа индекса, виде. У таблицы может быть только один кластеризованный индекс, так как данные могут быть отсортированы только в одном порядке. По возможности каждая таблица должна иметь кластеризованный индекс, если у таблицы нет кластеризованного индекса, такая таблица называется «*кучей*». Кластеризованный индекс создается автоматически при создании ограничений PRIMARY KEY (*первичный ключ*) и UNIQUE, если до этого кластеризованный индекс для таблицы еще не был определен. В случае создания кластеризованного индекса для таблицы (*кучи*), в которой есть некластеризованные индексы, то после создания все их необходимо перестроить.
- **Некластеризованный** (*Nonclustered*) – это индекс, который содержит значение ключа и указатель на строку данных, содержащую значение этого ключа. У таблицы может быть несколько некластеризованных индексов. Создаваться некластеризованные индексы могут как на таблицах с

кластеризованным индексом, так и без него. Именно этот тип индекса используется для повышения производительности часто используемых запросов, так как некластеризованные индексы обеспечивают быстрый поиск и доступ к данным по значениям ключа;

- **Фильтруемый** (*Filtered*) – это оптимизированный некластеризованный индекс, который использует предикат фильтра для индексирования части строк в таблице. Если хорошо спроектировать такой тип индекса, то он может повысить производительность запросов, а также снизить затраты на обслуживание и хранение индексов по сравнению с полнотабличными индексами;
- **Уникальный** (*Unique*) – это индекс, который обеспечивает отсутствие повторяющихся (*одинаковых*) значений ключа индекса, гарантируя тем самым уникальность строк по данному ключу. Уникальными могут быть как кластеризованные, так и некластеризованные индексы. Если создавать уникальный индекс по нескольким столбцам, индекс гарантирует уникальность каждой комбинации значений в ключе. При создании ограничений PRIMARY KEY или UNIQUE SQL сервер автоматически создает уникальный индекс для ключевых столбцов. Уникальный индекс может быть создан только в том случае, если у таблицы на текущий момент отсутствуют дублирующие значения по ключевым столбцам;
- **Колоночный** (*Columnstore*) – это индекс, основанный на технологии хранения данных в виде столбцов. Данный тип индекса эффективно использовать для больших хранилищ данных, поскольку он может увеличить производительность запросов к хранилищу до 10 раз и также до 10 раз уменьшить размер данных, так как данные в Columnstore индексе сжимаются. Существуют как кластеризованные колоночные индексы, так и некластеризованные;
- **Полнотекстовый** (*Full-text*) – это специальный тип индекса, который обеспечивает эффективную поддержку сложных операций поиска слов в символьных строковых данных. Процесс создания и обслуживания полнотекстового индекса называется «*заполнением*». Существует такие типы заполнения как: полное заполнение и заполнение на основе отслеживания изменений. По умолчанию SQL сервер полностью заполняет новый полнотекстовый индекс сразу после его создания, но на это может потребоваться значительный объем ресурсов, в зависимости от размеров таблицы, поэтому есть возможность откладывать полное заполнение. Заполнение на основе отслеживания изменений используется для обслуживания полнотекстового индекса после его первоначального полного заполнения;
- **Пространственный** (*Spatial*) – это индекс, который обеспечивает возможность более эффективного использования конкретных операций на пространственных объектах в столбцах с типом данных geometry или geography. Данный тип индекса может быть создан только для пространственного столбца, также таблица, для которой определяется

пространственный индекс, должна содержать первичный ключ (*PRIMARY KEY*);

– **XML** – это еще один специальный тип индекса, который предназначен для столбцов с типом данных XML. Благодаря XML-индексу повышается эффективность обработки запросов к XML столбцам. Существует два вида XML-индекса: первичные и вторичные. Первичный XML-индекс индексирует все теги, значения и пути, хранимые в XML столбце. Он может быть создан, только если у таблицы есть кластеризованный индекс по первичному ключу. Вторичный XML-индекс может быть создан, только если у таблицы есть первичный XML-индекс и используется он для повышения производительности запросов по определенному типу обращения к XML-столбцу, в связи с этим существует несколько типов вторичных индексов: PATH, VALUE и PROPERTY;

– Также существуют специальные индексы для таблиц, оптимизированных для памяти (*In-Memory OLTP*) такие как: Хэш (*Hash*) индексы и некластеризованные индексы, оптимизированные для памяти, которые создаются для сканирования диапазона и упорядоченного сканирования.

При создании индексов нужно учитывать 2 фактора два фактора, для гарантирования, что индексы будут более эффективны, чем сканирование таблицы: природа данных и природа запросов к таблице. Сервер SQL использует индексы для указания на расположение строки в странице данных вместо просматривания всех страниц таблицы. При создании индексов нужно учитывать достоинства и недостатки индексов.

Достоинства индексов:

- Индексы обычно увеличивают скорость выполнения запросов связанных таблиц и выполнение сортировки и группировки. Индексы принуждают делать строки уникальными, если включена уникальность. Индексы создаются в порядке возрастания или уменьшения.
- Индексы достаточно полезны, но они занимают место на диске и берут на себя дополнительные накладные расходы и расходы на эксплуатацию.

Недостатки индексов:

- Когда вы изменяете данные в индексной колонке, сервер SQL обновляет связанные индексы.
- Накладные расходы на поддержку индексов требуют времени и ресурсов. Поэтому не создавайте индексы, которые не будете часто использовать.

Индексы на колонки, содержащие большое количество дублирующих данных могут иметь несколько преимуществ. Вы должны создавать только самые необходимые индексы, потому что каждый лишний индекс может серьезно ударить по производительности во время добавления новых записей. Это особенно становится заметным, при массовой загрузке данных.

Создание и удаление индексов в Microsoft SQL Server

Перед тем как приступать к созданию индекса его необходимо хорошо спроектировать, для того чтобы эффективно использовать этот индекс, так как плохо спроектированные индексы могут не увеличить производительность, а наоборот снизить ее. Например, большое количество индексов в таблице снижает производительность инструкций INSERT, UPDATE, DELETE и MERGE, потому что при изменении данных в таблице все индексы должны быть изменены соответствующим образом.

Синтаксис для SQL Server

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name
   ON <object> ( column [ ASC | DESC ] [ ,...n ] )
   [ INCLUDE ( column_name [ ,...n ] ) ]
   [ WHERE <filter_predicate> ]
   [ WITH ( <relational_index_option> [ ,...n ] ) ]
   [ ON { partition_scheme_name ( column_name )
       | filegroup_name
       | default
       }
   ]
  [ FILESTREAM_ON { filestream_filegroup_name | partition_scheme_name | "NULL" } ]

[ ; ]

<object> ::=
{ database_name.schema_name.table_or_view_name | schema_name.table_or_view_name | table_or_view_name }

<relational_index_option> ::=
{
   PAD_INDEX = { ON | OFF }
  | FILLFACTOR = fillfactor
  | SORT_IN_TEMPDB = { ON | OFF }
  | IGNORE_DUP_KEY = { ON | OFF }
  | STATISTICS_NORECOMPUTE = { ON | OFF }
  | STATISTICS_INCREMENTAL = { ON | OFF }
  | DROP_EXISTING = { ON | OFF }
  | ONLINE = { ON | OFF }
  | RESUMABLE = { ON | OFF }
  | MAX_DURATION = <time> [MINUTES]
```

```
  | ALLOW_ROW_LOCKS = { ON | OFF }
  | ALLOW_PAGE_LOCKS = { ON | OFF }
  | OPTIMIZE_FOR_SEQUENTIAL_KEY = { ON | OFF }
  | MAXDOP = max_degree_of_parallelism
  | DATA_COMPRESSION = { NONE | ROW | PAGE }
    [ ON PARTITIONS ( { <partition_number_expression> | <range> }
    [ , ...n ] ) ]
}

<filter_predicate> ::=
   <conjunct> [ AND <conjunct> ]

<conjunct> ::=
   <disjunct> | <comparison>

<disjunct> ::=
      column_name IN (constant ,...n)

<comparison> ::=
      column_name <comparison_op> constant

<comparison_op> ::=
   { IS | IS NOT | = | <> | != | > | >= | !> | < | <= | !< }

<range> ::=
<partition_number_expression> TO <partition_number_expression>
```

Аргументы

UNIQUE

Создает уникальный индекс для таблицы или представления. Уникальным является индекс, в котором не может быть двух строк с одним и тем же значением ключа индекса. Кластеризованный индекс представления должен быть уникальным.

CLUSTERED

Создает индекс, в котором логический порядок значений ключа определяет физический порядок соответствующих строк в таблице. На нижнем (конечном) уровне кластеризованного индекса хранятся действительные строки данных таблицы. Для таблицы или представления в каждый момент времени может существовать только один кластеризованный индекс. Представление с уникальным кластеризованным индексом называется индексированным. Создание уникального кластеризованного индекса физически материализует представление. Уникальный кластеризованный индекс для представления должен быть создан до того, как для этого же представления будут определены какие-либо другие индексы.

Создавайте кластеризованные индексы до создания любых некластеризованных. При создании кластеризованного индекса все существующие некластеризованные индексы таблицы перестраиваются.

Если аргумент CLUSTERED не указан, создается некластеризованный индекс.

NONCLUSTERED

Создает индекс, задающий логическое упорядочение для таблицы. Логический порядок строк в некластеризованном индексе не влияет на их физический порядок. Каждая таблица может содержать до 999 некластеризованных индексов независимо от способа их создания: неявно с помощью ограничений PRIMARY KEY и UNIQUE или явно с помощью инструкции CREATE INDEX.

Для индексированных представлений некластеризованные индексы могут создаваться только в случае, если уже определен уникальный кластеризованный индекс.

Если не указано иное, типом индекса по умолчанию является NONCLUSTERED.

index_name

Имя индекса. Имена индексов должны быть уникальными в пределах таблицы или представления, но необязательно должны быть уникальными в пределах базы данных. Имена индексов должны удовлетворять правилам для идентификаторов.

column

Столбец или столбцы, на которых основан индекс. Имена одного или нескольких столбцов для создания комбинированного индекса. Столбцы, которые должны быть включены в составной индекс, указываются в скобках за аргументом table_or_view_name в порядке сортировки.

В один ключ составного индекса могут входить до 32 столбцов. Все столбцы ключа составного индекса должны находиться в одной таблице или одном и том же представлении. Максимально допустимый размер значений составного индекса составляет 900 байтов для кластеризованного индекса или 1700 для некластеризованного индекса. Ограничениями являются 16 столбцов и 900 байт для версий до База данных SQL и SQL Server 2016 (13.x). Столбцы с типами данных LOB ntext , text , varchar(max) , nvarchar(max) , varbinary(max) , xml или image нельзя указать в качестве столбцов для индекса. Кроме того, определение представления не может включать столбцы типов ntext , text или image , даже если они не указаны в инструкции CREATE INDEX.

Можно создавать индексы на столбцах с определяемым пользователем типом данных CLR, если этот тип поддерживает двоичное упорядочение. Можно также создавать индексы на вычисляемых столбцах, определенных как вызовы методов для столбцов с определяемыми пользователем типами данных, если эти методы помечены как детерминированные и не выполняют операции доступа к данным.

[ASC | DESC]

Определяет сортировку значений заданного столбца индекса: по возрастанию или по убыванию. Значение по умолчанию — ASC.

INCLUDE (column [, ... n])

Указывает неключевые столбцы, добавляемые на конечный уровень некластеризованного индекса. Некластеризованный индекс может быть уникальным или неуникальным.

Имена столбцов в списке INCLUDE не могут повторяться и не могут использоваться одновременно как ключевые и неключевые. Некластеризованные индексы всегда содержат столбцы кластеризованного индекса, если для таблицы определен кластеризованный индекс. Допускаются данные всех типов, за исключением text, ntext и image . Индекс должен создаваться или перестраиваться в режиме "вне сети" (ONLINE = OFF), если любой из заданных неключевых столбцов имеет тип данных varchar(max) , nvarchar(max) или varbinary(max) .

Вычисляемые столбцы, являющиеся детерминированными и точными или неточными, могут быть включенными столбцами. Вычисляемые столбцы, производные от типов данных image, ntext, text, varchar(max), nvarchar(max), varbinary(max) и xml , могут быть включены в неключевые столбцы, если типы данных вычисляемого столбца допускаются в качестве столбца для включения.

WHERE <filter_predicate>

Создает отфильтрованный индекс путем указания строк для включения в индекс. Отфильтрованный индекс должен быть некластеризованным индексом для таблицы. Создается отфильтрованная статистика для строк данных отфильтрованного индекса. Предикат фильтра использует простую логику сравнения и не может ссылаться на вычисляемый столбец, столбец определяемого пользователем типа, столбец типа пространственных данных или столбец типа hierarchyID. Сравнения с помощью литералов NULL

Создание индексов

Для создания индексов в Microsoft SQL Server существует два способа: первый – это с помощью графического интерфейса среды SQL Server Management Studio (SSMS), и второй – это с помощью языка Transact-SQL.

Исходные данные для примеров

Давайте представим, что у нас есть таблица с товарами под названием TestTable, в которой есть три столбца:

- ProductId – идентификатор товара;
- ProductName – наименование товара;
- CategoryID – категория товара.

```
CREATE TABLE TestTable(
    ProductId INT IDENTITY(1,1) NOT NULL,
    ProductName VARCHAR(50) NOT NULL,
    CategoryID INT NULL,
```

) ON [PRIMARY]

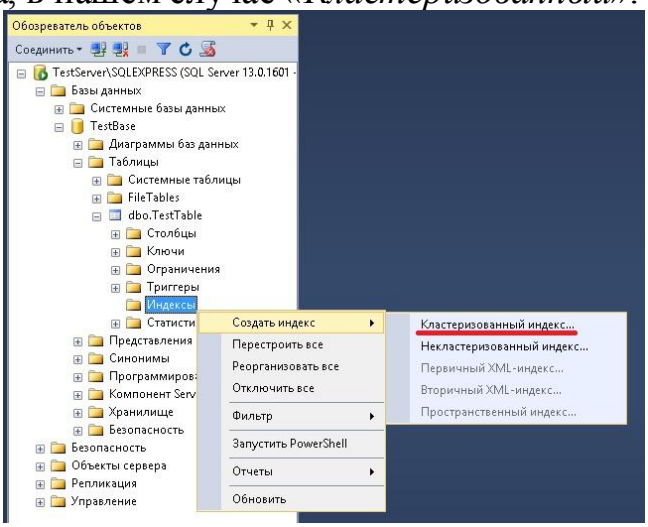
Пример создания кластеризованного индекса

Как известно, кластеризованный индекс создается автоматически, если мы, например, при создании таблицы указываем конкретный столбец в качестве первичного ключа (*PRIMARY KEY*), но так как мы этого не сделали, давайте рассмотрим пример самостоятельного создания кластеризованного индекса.

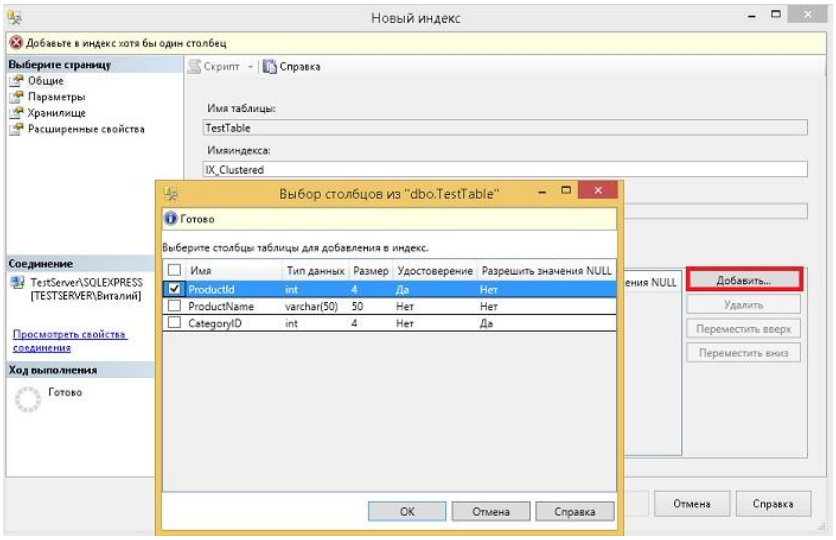
Для создания кластеризованного индекса мы можем у таблицы указать первичный ключ, и тем самым кластеризованный индекс будет создан автоматически или мы можем создать кластеризованный индекс отдельно.

Для примера давайте просто создадим кластеризованный индекс, без создания первичного ключа. Сначала сделаем это с помощью Management Studio.

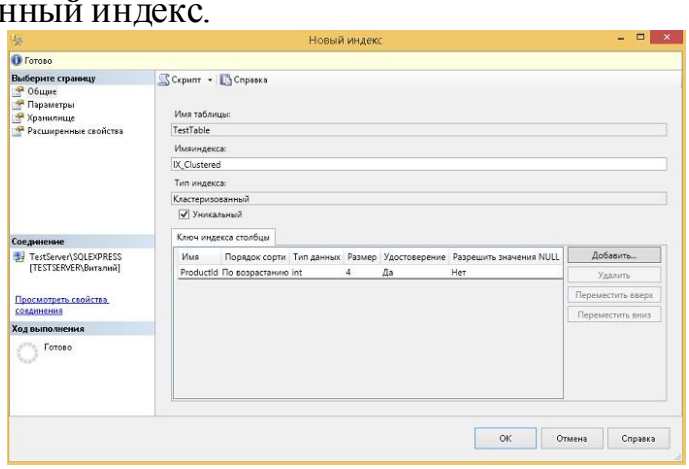
Открываем SSMS и в обозревателе объектов находим нужную таблицу и щелкаем правой кнопкой мыши по пункту «*Индексы*», выбираем «*Создать индекс*» и тип индекса, в нашем случае «*Кластеризованный*».



Откроется форма «*Новый индекс*», где нам необходимо указать имя нового индекса (*оно должно быть уникальным в пределах таблицы*), также указываем, будет ли этот индекс уникальным, если мы говорим об идентификаторе товара в таблице товаров, то, конечно же, он должен быть уникальным. Потом выбираем столбец (*ключ индекса*), на основе которого у нас будет создан кластеризованный индекс, т.е. будут отсортированы строки данных в таблице, с помощью кнопки «*Добавить*».



После ввода всех необходимых параметров жмем «*ОК*», в итоге будет создан кластеризованный индекс.



Точно также можно было бы создать кластеризованный индекс, используя инструкцию T-SQL **CREATRE INDEX**, например, вот так

```
CREATE UNIQUE CLUSTERED INDEX IX_Clustered ON TestTable
(
    ProductId ASC
)
GO
```

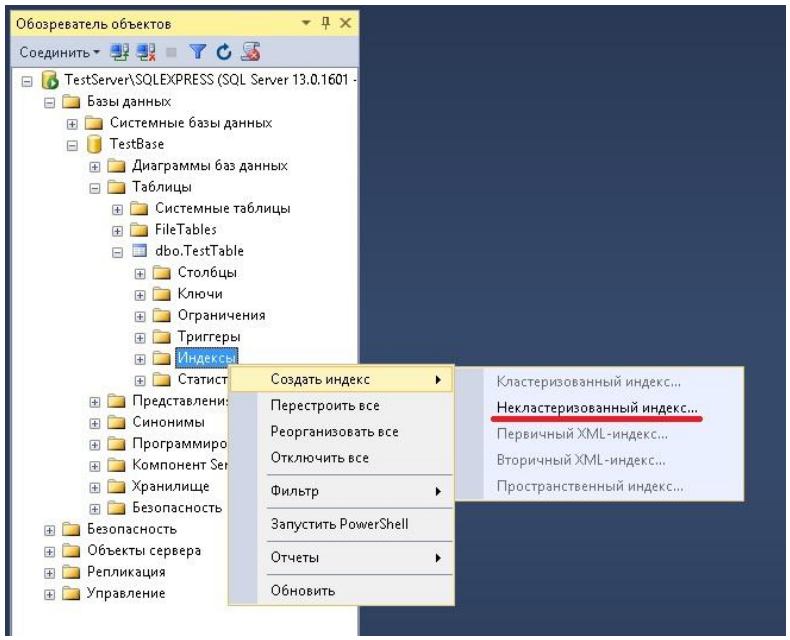
Или можно было бы использовать и инструкцию создания первичного ключа, например

```
ALTER TABLE TestTable ADD CONSTRAINT PK_TestTable PRIMARY KEY
CLUSTERED
(
    ProductId ASC
)
GO
```

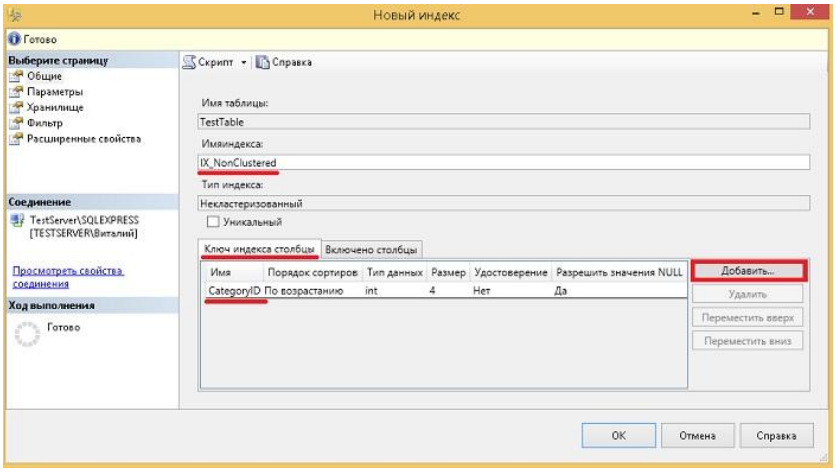
Пример создания некластеризованного индекса с включенными столбцами

Сейчас давайте рассмотрим пример создания некластеризованного индекса, при этом мы укажем столбцы, которые не будет являться ключевыми, но будут включаться в индекс. Это полезно в тех случаях, когда Вы создаете индекс для конкретного запроса, например, для того чтобы индекс полностью покрывал запрос, т.е. содержал все столбцы (*это называется «Покрытием запроса»*). Благодаря покрытию запроса повышается производительность, так как оптимизатор запросов может найти все значения столбцов в индексе, при этом не обращаясь к данным таблиц, что приводит к меньшему числу дисковых операций ввода-вывода. Но помните, что включение в индекс неключевых столбцов влечет за собой увеличение размера индекса, т.е. для хранения индекса потребуется больше места на диске, а также может повлечь и снижение производительности операций INSERT, UPDATE, DELETE и MERGE на базовой таблице.

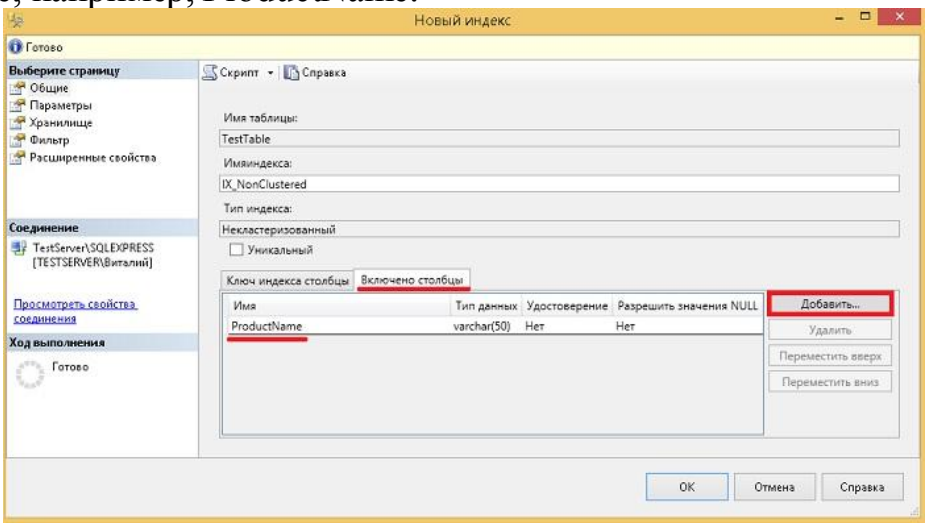
Для того чтобы создать некластеризованный индекс с помощью графического интерфейса Management Studio, мы также находим нужную таблицу и пункт индексы, только в данном случае мы выбираем «*Создать -> Некластеризованный индекс*».



После открытия формы «*Новый индекс*» мы указываем название индекса, добавляем ключевой столбец или столбцы с помощью кнопки «*Добавить*», например, для нашего тестового случая давайте укажем CategoryID.



Далее переходим на вкладку «*Включено столбцы*» и с помощью кнопки «*Добавить*» добавляем столбцы, которые мы хотим включить в индекс, в нашем случае, например, ProductName.

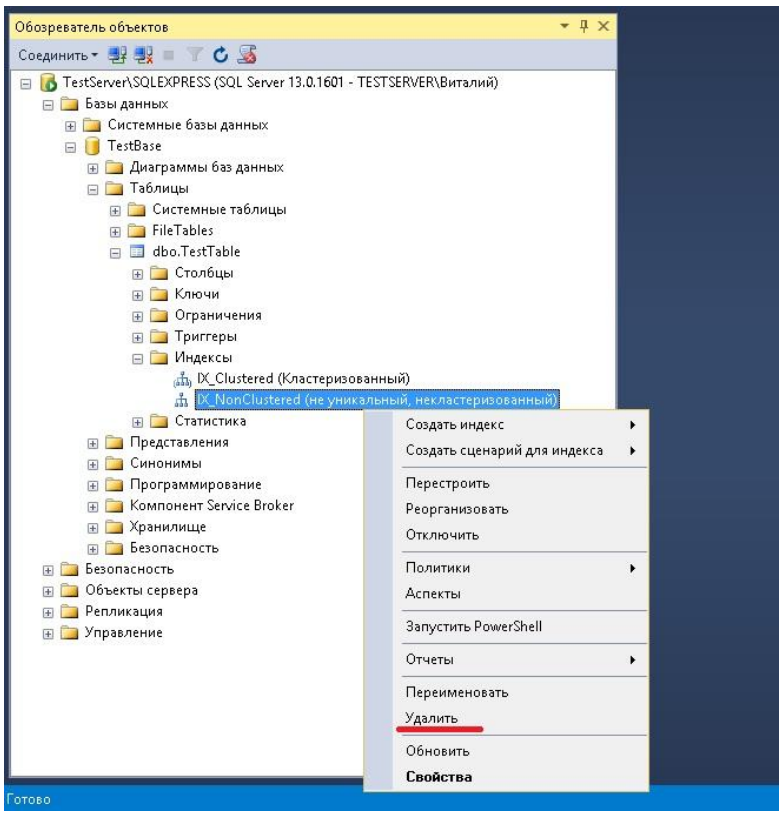


На Transact-SQL это будет выглядеть следующим образом.

```
CREATE NONCLUSTERED INDEX IX_NonClustered ON TestTable
(
	CategoryID ASC
)
INCLUDE (ProductName)
GO
```

Пример удаления индекса в Microsoft SQL Server

Для того чтобы удалить индекс можно щелкнуть правой кнопкой по нужному индексу и нажать «*Удалить*», затем подтвердить свое действия нажав «*ОК*».



или также можно использовать инструкцию **DROP INDEX**, например

```
DROP INDEX IX_NonClustered ON TestTable
```

Следует отметить, что инструкция DROP INDEX неприменима к индексам, которые были созданы путем создания ограничений PRIMARY KEY и UNIQUE. В данном случае для удаления индекса нужно использовать инструкцию ALTER TABLE с предложением DROP CONSTRAINT.

Оптимизация индексов в Microsoft SQL Server

В результате выполнения операций обновления, добавления или удаления данных в таблицах SQL сервер автоматически вносит соответствующие изменения в индексы, но со временем все эти изменения могут вызвать фрагментацию данных в индексе, т.е. они окажутся разбросанными по базе данных. Фрагментация индексов влечет за собой снижение производительности запросов, поэтому периодически необходимо выполнять операции обслуживания индексов, а именно дефрагментацию, к таким можно отнести операции реорганизации и перестроения индексов.

В каких случаях использовать реорганизацию индекса, а в каких перестроение?

Чтобы ответить на этот вопрос сначала необходимо определить степень фрагментации индекса, так как в зависимости от фрагментации индекса тот или иной метод дефрагментации будет предпочтительней и эффективней. Для определения степени фрагментации индекса можно использовать системную табличную функцию sys.dm_db_index_physical_stats, которая возвращает подробные сведения о размере и фрагментации индексов.

Например, используя следующий запрос, Вы можете узнать степень фрагментации индексов у всех таблиц в текущей базе данных.

```
SELECT OBJECT_NAME(T1.object_id) AS NameTable,
       T1.index_id AS IndexId,
       T2.name AS IndexName,
       T1.avg_fragmentation_in_percent AS Fragmentation
FROM sys.dm_db_index_physical_stats (DB_ID(), NULL, NULL, NULL, NULL) AS T1
LEFT JOIN sys.indexes AS T2 ON T1.object_id = T2.object_id AND T1.index_id = T2.index_id
```

В данном случае нас интересует столбец avg_fragmentation_in_percent, т.е. процентная доля логической фрагментации.

Так вот, Microsoft рекомендует:

Если степень фрагментации менее 5%, то реорганизацию или перестроение индекса вообще не стоит запускать;

Если степень фрагментации от 5 до 30%, то имеет смысл запустить реорганизацию индекса, так как данная операция использует минимальные системные ресурсы и не требует долговременных блокировок;

Если степень фрагментации более 30%, то необходимо выполнять перестроение индекса, так как данная операция, при значительной фрагментации, дает больший эффект чем операция реорганизации индекса.

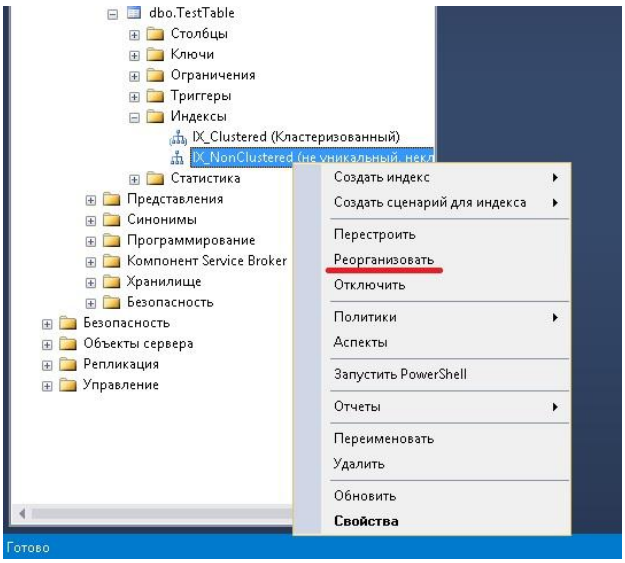
Если у Вас небольшая компания и база данных не требует максимальной отдачи в режиме 24 часа в сутки, т.е. она не суперактивная БД, то Вы можете смело периодически выполнять операцию перестроения индексов, при этом даже не определяя степень фрагментации.

Реорганизация индексов

Реорганизация индекса – это процесс дефрагментации индекса, который дефрагментирует конечный уровень кластеризованных и некластеризованных индексов по таблицам и представлениям, физически переупорядочивая страницы концевого уровня в соответствии с логическим порядком (слева направо) конечных узлов.

Для реорганизации индекса можно использовать как графический инструмент SSMS, так и инструкцию Transact-SQL.

Реорганизация индекса с помощью Management Studio



Реорганизация индекса с помощью Transact-SQL

```
ALTER INDEX IX_NonClustered ON TestTable
REORGANIZE
GO
```

Перестроение индексов

Перестроение индекса – это процесс, при котором происходит удаление старого индекса и создание нового, в результате чего фрагментация устраняется.

Для перестроения индексов можно использовать два способа.

Первый. Используя инструкцию ALTER INDEX с предложением REBUILD. Эта инструкция заменяет инструкцию DBCC DBREINDEX. Обычно для массового перестроения индексов используется именно этот способ.

Пример

```
ALTER INDEX IX_NonClustered ON TestTable
REBUILD
GO
```

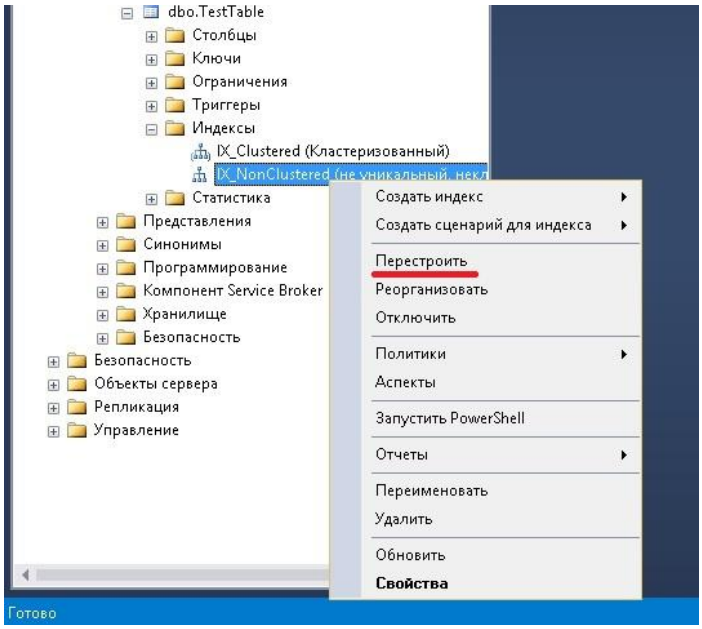
И второй, используя инструкцию CREATE INDEX с предложением DROP_EXISTING. Можно использовать, например, для перестроения индекса с изменением его определения, т.е. добавления или удаления ключевых столбцов.

Пример

```
CREATE NONCLUSTERED INDEX IX_NonClustered ON TestTable
(
    CategoryID ASC
)
```

```
WITH(DROP_EXISTING = ON)
GO
```

В Management Studio функционал для перестроения также доступен. Правой кнопкой по нужному индексу «*Перестроить*».

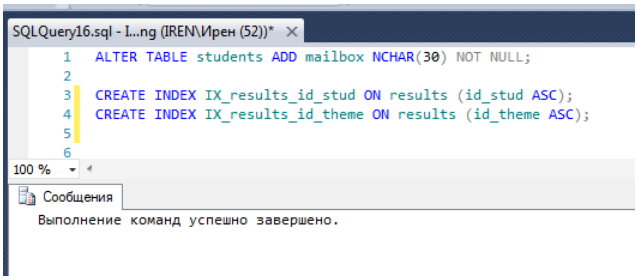


Задания

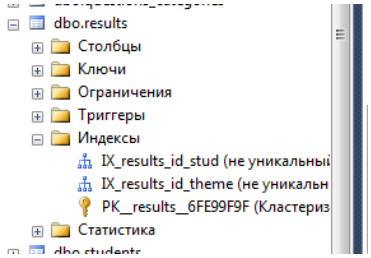
1 Изучить теоретические сведения.
2 В соответствии с вариантом задания создать индексы в базе данных.

Порядок выполнения работы

Т.к. первичные ключи в каждой таблице созданы, то по умолчанию они являются кластеризованными индексами. Создадим некластеризованные. Например, в базе данных тестирования есть таблица результатов тестирования по каждому студенту и каждой теме и группе. В ней одним их полей является код студента. Для анализа прохождения тестов необходимо выполнять выборку по теме, по коду студента, по коду группы. А для оптимизации выборки создадим индексы в перечисленных полях.

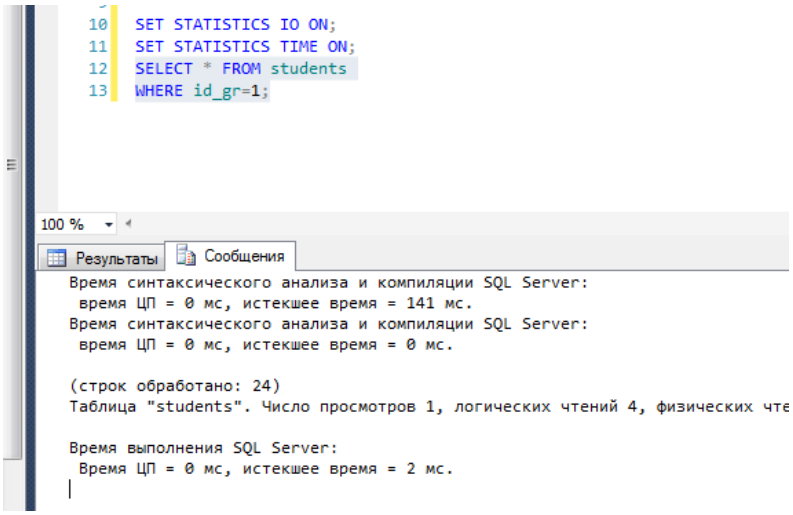


После обновление таблицы:

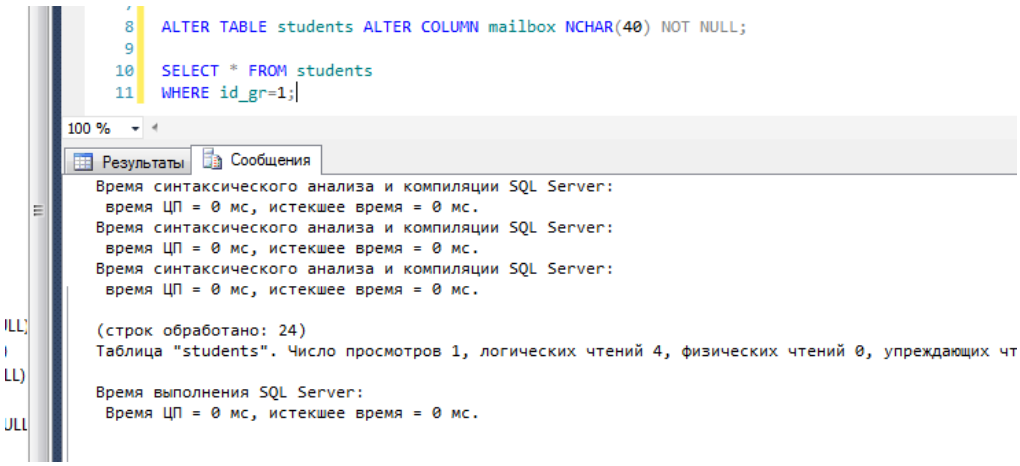


Попробуем сравнить выполнение запроса на выборку. Для этого воспользуемся командами, которые позволяют SQL Server отображать сведения об активности диска и время в миллисекундах, связанные с выполнением инструкций Transact-SQL:

SET STATISTICS IO ON;
SET STATISTICS TIME ON;



В таблице студентов добавим индекс в поле код группы.



Значение истекшего времени мало, но количество обработанных записей в таблице 50, а если их будет в разы больше, и запросов одновременно будет обрабатываться несколько, то оптимизация выборки будет заметнее.

Из вышесказанного можно сделать вывод, что …

Содержание отчета

1 Название работы
2 Цель работы
3 Перечень технических средств обучения
4 Порядок выполнения работы
5 Вывод

Варианты заданий

Варианты заданий представлены в практической работе № 13.

Используемая литература

– Г.Н.Федорова Основы проектирования баз данных. М.: Академия, 2020
– Г.Н.Федорова Разработка, администрирование и защита баз данных. М.: Академия, 2018
– https://info-comp.ru/programmirovanie/575-index-basics-in-ms-sql-server.html
– https://www.bibliofond.ru/view.aspx?id=724782