**Лабораторная работа № 9. Создание и применение индексов**

**КУРНОСЕНКО СОФЬЯ**

**Индекс** – это объект базы данных, позволяющий **ускорить поиск** в определенной таблице, так как при этом данные организуются в виде сбалансированного бинарного дерева поиска.

Как и любой другой объект базы данных, индекс может быть создан с помощью оператора CREATE, модифицирован с помощью ALTER и удален с помощью оператора DROP. Для одной таблицы возможно построение нескольких индексов.

Индексы бывают кластеризованные, некластеризованные, уникальные, неуникальные и др.

1.С помощью SSMS определить все индексы, которые имеются в БД **UNIVER**. Определить, какие из них являются кластеризованными, а какие некластеризованными.

Создать временную локальную таблицу. Заполнить ее данными (не менее 1000 строк).

Разработать SELECT-запрос. Получить план запроса и определить его стоимость.

Создать кластеризованный индекс, уменьшающий стоимость SELECT-запроса.

***→ Теория (методичка)***

Обычно *кластеризованные* индексы создаются автоматически при создании таблицы если в ней присутствует первичный ключ (ограничение PRIMARY KEY).

Кластеризованные индексы физически упорядочены в соответствии со значениями индексируемых столбцов. В таблице может быть только один кластеризованный индекс.

С помощью системной процедуры SP\_HELPINDEX можно получить перечень индексов, связанных с заданной таблицей:



Пусть сформирована временная локальная таблица:

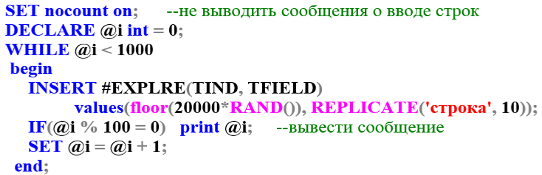
**CREATE table #EXPLRE**

**( TIND int,**

**TFIELD varchar(100)**

**);**

Заполнить эту таблицу можно с помощью следующего сценария:



План запроса можно посмотреть, выполнив в контекстном меню запроса команду **Display Estimated Execution Plan** (**Показать предполагаемый план выполнения**)

или нажав соответствующую кнопку на панели инструментов.

**SELECT \* FROM #EXPLRE where TIND between 1500 and 2500 order by TIND**

Общая стоимость запроса (**Estimated Subtree Cost**) появляется во всплывающем окне, если подвести курсор к компоненту **Table Scan** (она равна 0,011). Чтобы объективно оценить время выполнения следующего запроса, надо очистить буферный кэш:

**checkpoint;** --фиксация БД

**DBCC DROPCLEANBUFFERS;** --очистить буферный кэш

Если создать кластеризованный индекс:

**CREATE clustered index #EXPLRE\_CL on #EXPLRE(TIND asc)**

и вновь выполнить запрос, то его стоимость станет равна 0,0033, т. е. уменьшится.

***→ Выполнение заданий***

-- 1.1

use UNIVERSITY;

exec sp\_helpindex 'AUDITORIUM';

exec sp\_helpindex 'AUDITORIUM\_TYPE';

exec sp\_helpindex 'FACULTY';

exec sp\_helpindex 'GROUPS';

exec sp\_helpindex 'PROFESSION';

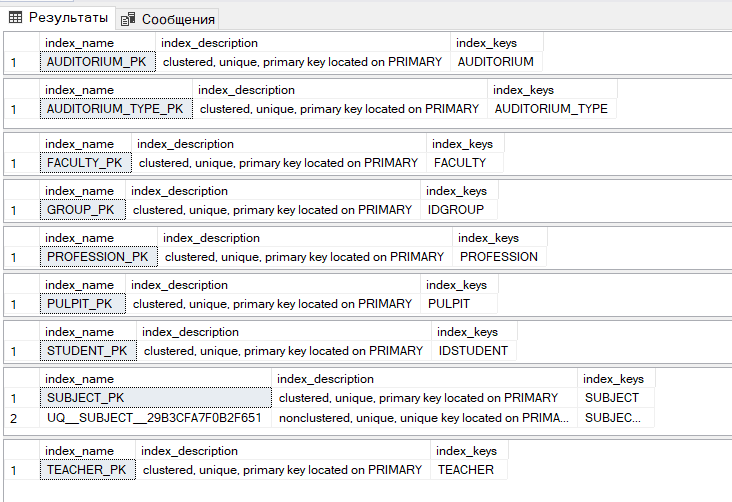
exec sp\_helpindex 'PROGRESS';

exec sp\_helpindex 'PULPIT';

exec sp\_helpindex 'STUDENT';

exec sp\_helpindex 'SUBJECT';

exec sp\_helpindex 'TEACHER';



-- 1.2

-- создание временной таблицы

create table #tempTable(

num int,

string char(2)

);

-- заполнение таблицы

set nocount on; -- не выводить сообщения о затронутых строках

declare @i int = 0;

while @i < 1000

begin

insert #tempTable

values(floor(rand()\*20000), 'po');

set @i += 1;

end;

-- select-запрос

select \* from #tempTable

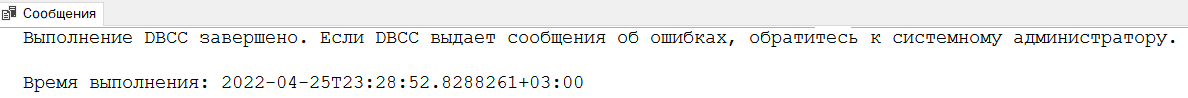
where num between 1500 and 2500

order by num;

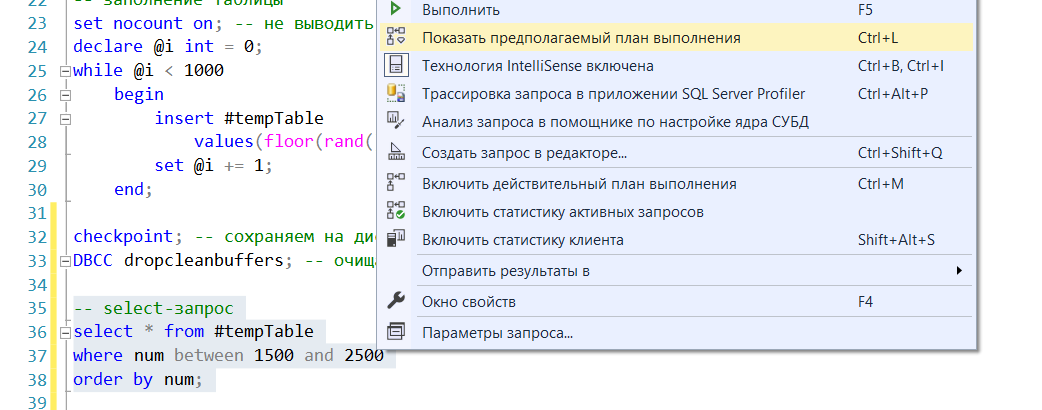
Очистка кэша:

checkpoint; -- сохраняем на диск все данные предыдущих инструкций из буффера

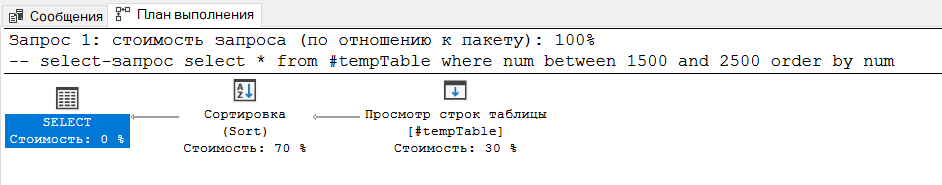
DBCC dropcleanbuffers; -- очищаем буферный кэш



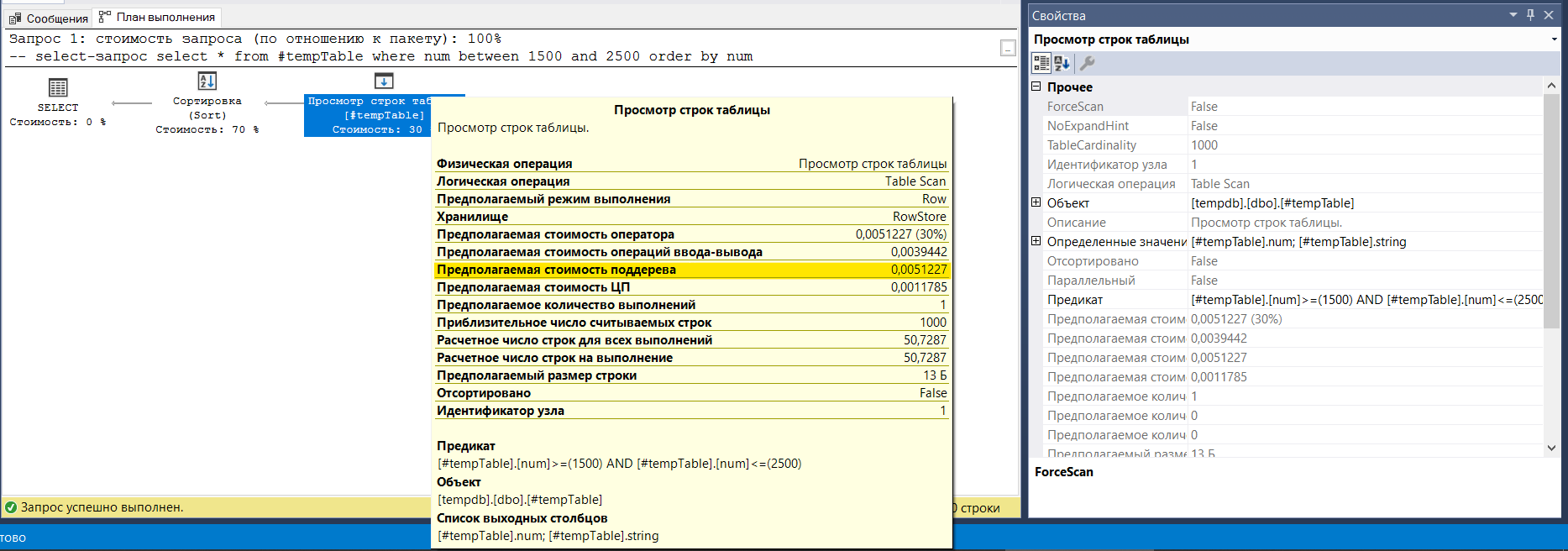
Выделяю строку select-запроса к временной таблице и выбираю в контекстном меню «Показать предполагаемый план выполнения»:



Получаю такой план:



При наведении курсора на значок:



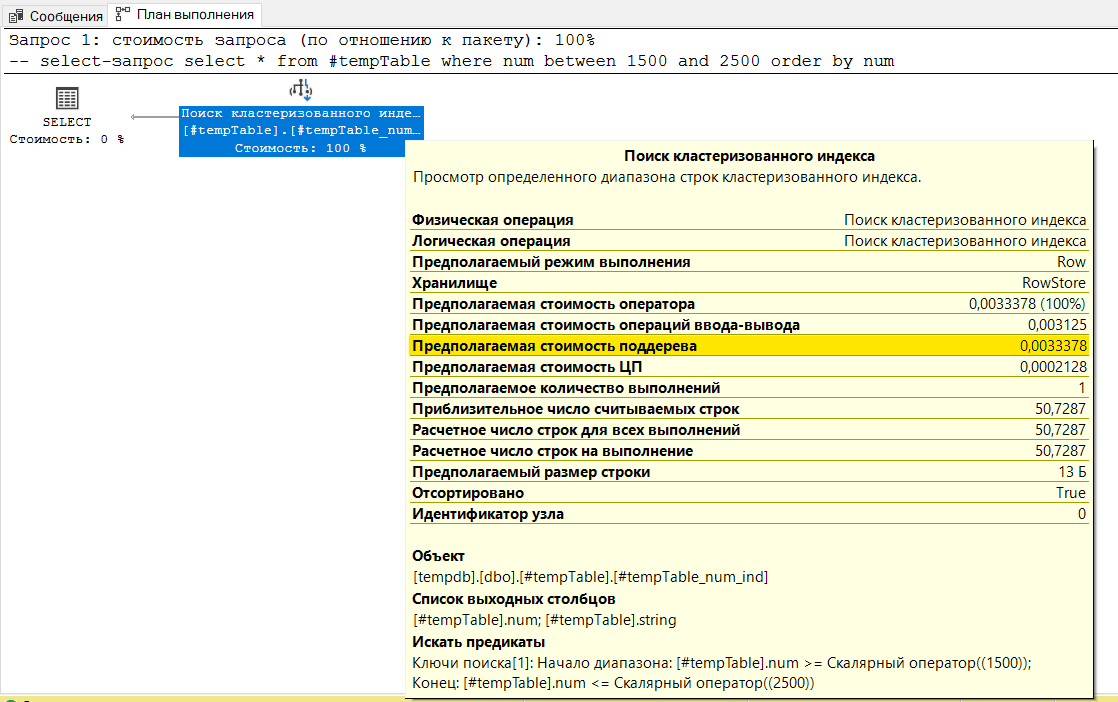
Создала кластеризованный индекс:

-- создание кластеризованного индекса на первый столбец таблицы #tempTable

create clustered index #tempTable\_num\_ind on #tempTable(num);

Снова очистка кэша.

Теперь план выполнения:



2.Создать временную локальную таблицу. Заполнить ее данными (10000 строк или больше).

Разработать SELECT-запрос. Получить план запроса и определить его стоимость.

Создать *некластеризованный* неуникальный *составной* индекс.

Оценить процедуры поиска информации.

***→ Теория (методичка)***

*Некластеризованные* индексы не влияют на физический порядок строк в таблице. Пусть сформирована и заполнена временная локальная таблица **#EX**:

**CREATE table #EX**

**( TKEY int,**

**CC int identity(1, 1),**

**TF varchar(100));**

**);**

**set nocount on;**

**declare @i int = 0;**

**while @i < 20000 --** добавление в таблицу 20000 строк

**begin**

**INSERT #EX(TKEY, TF) values(floor(30000\*RAND()), replicate('строка ', 10));**

**set @i = @i + 1;**

**end;**

**SELECT count(\*)[количество строк] from #EX;**

**SELECT \* from #EX**

MSS допускает создавать индексы по нескольким столбцам – такие индексы называются *составными*. Можно создать составной неуникальный, некластеризованный индекс **#EX\_NONCLU** по двум столбцам **TKEY** и **CC** таблицы **#EX** с помощью оператора:

**CREATE index #EX\_NONCLU on #EX(TKEY, CC)**

Этот индекс не применяется оптимизатором ни при фильтрации, ни при сортировке строк таблицы #EX, в чем можно убедиться, посмотрев планы следующих запросов:

**SELECT \* from #EX where TKEY > 1500 and CC < 4500;**

**SELECT \* from #EX order by TKEY, CC**

Но, если хотя бы одно из индексируемых значений зафиксировать (задать одно значение), то оптимизатор применит индекс. Это можно проверить, выполнив запрос:

**SELECT \* from #EX where TKEY = 556 and CC > 3**

***→ Выполнение заданий***

-- создание таблицы

create table #tempTable\_2(

id int identity(1,1),

num int,

string char(2),

);

-- заполнение таблицы; при floor(rand()\*300) индекс не применялся, кстати

set nocount on;

declare @i\_2 int = 0;

while @i\_2 < 10000

begin

insert #tempTable\_2

values(@i\_2, floor(rand()\*3000), 'po');

set @i\_2 += 1;

end;

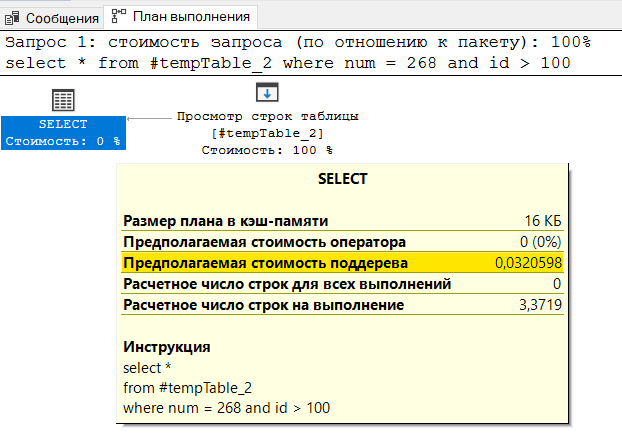
План выполнения запроса:

-- запрос к временной таблице

select \*

from #tempTable\_2

where num = 268 and id > 100;

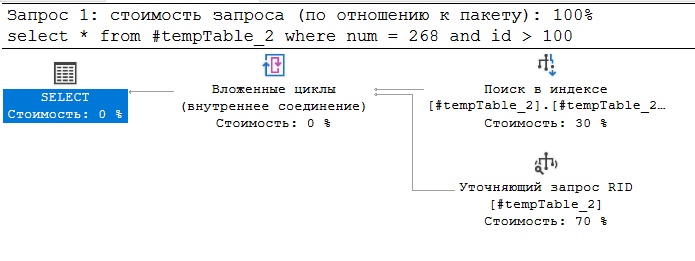
******

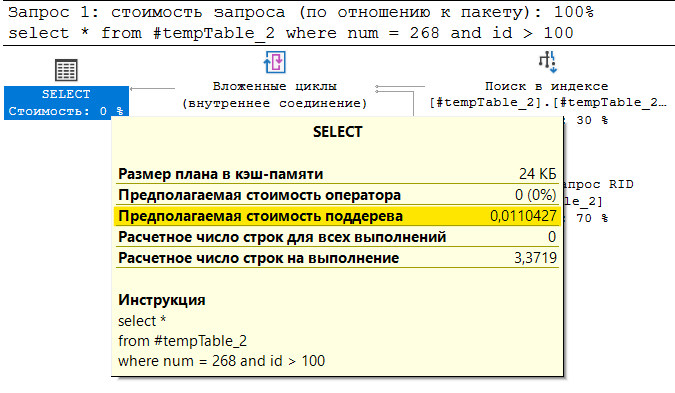
Создаю индекс

-- некластеризованный неуникальный составной индекс

create index #tempTable\_2\_id\_num\_ind on #tempTable\_2(id, num);

План выполнения запроса после создания индекса:





3.Создать временную локальную таблицу. Заполнить ее данными (не менее 10000 строк).

Разработать SELECT-запрос. Получить план запроса и определить его стоимость.

Создать *некластеризованный индекс покрытия*, уменьшающий стоимость SELECT-запроса.

***→ Теория (методичка)***

*Некластеризованный индекс покрытия* запроса (индекс с включенными столбцами) позволяет включить в состав индексной строки значения одного или нескольких неиндексируемых столбцов. Например, индекс покрытия **#EX\_TKEY\_X** включаетзначения столбца **CC** (ключевое слово INCLUDE):

**CREATE index #EX\_TKEY\_X on #EX(TKEY) INCLUDE (CC)**

Чтобы оценить процедуры поиска можно посмотреть планы выполнения запроса без применения индексов и с использованием индекса покрытия.

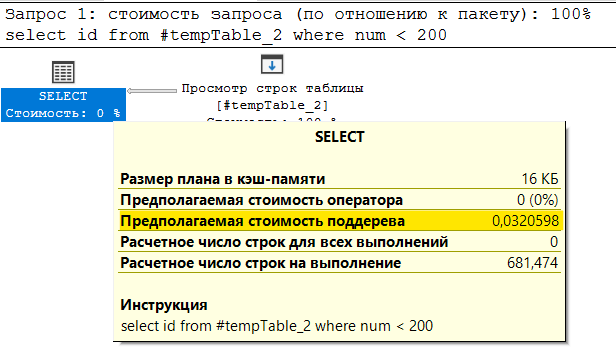
**SELECT CC from #EX where TKEY>15000**

***→ Выполнение заданий***

Запрос:

select id from #tempTable\_2 where num < 200;

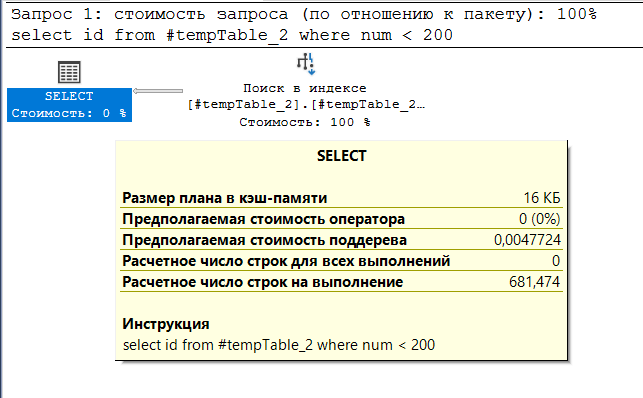
План выполнения до применения индекса:

******

Индекс:

create index #tempTable\_2\_num\_include on #tempTable\_2(num) include (id);

План после создания индекса:



4.Создать и заполнить временную локальную таблицу.

Разработать SELECT-запрос, получить план запроса и определить его стоимость.

Создать *некластеризованный фильтруемый индекс*, уменьшающий стоимость SELECT-запроса.

***→ Теория (методичка)***

Если запросы основаны на WHERE-фильтрации строк, то может быть эффективным применение *фильтруемых некластеризованных* индексов.

Пусть имеется три запроса. Надо оценить планы их выполнения.

**SELECT TKEY from #EX where TKEY between 5000 and 19999;**

**SELECT TKEY from #EX where TKEY>15000 and TKEY < 20000**

**SELECT TKEY from #EX where TKEY=17000**

Затем можно создать фильтрующий индекс с именем, например **#EX\_WHERE**:

**CREATE index #EX\_WHERE on #EX(TKEY) where (TKEY>=15000 and**

**TKEY < 20000);**

Здесь фильтруемый индекс создается только для строк таблицы **#EX,** которыеудовлетворяют логическому условию. Стоимость запросов уменьшится.

***→ Выполнение заданий***

Создала 3 запроса:

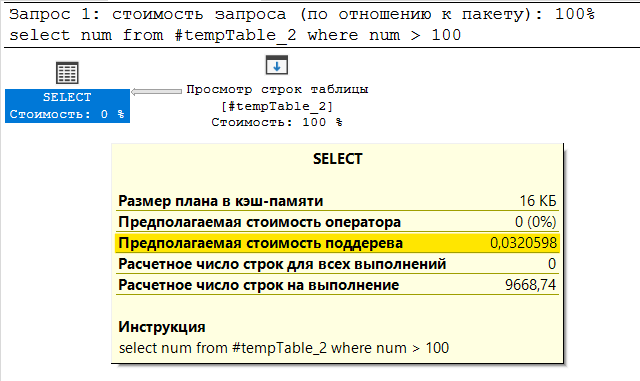
-- запросы

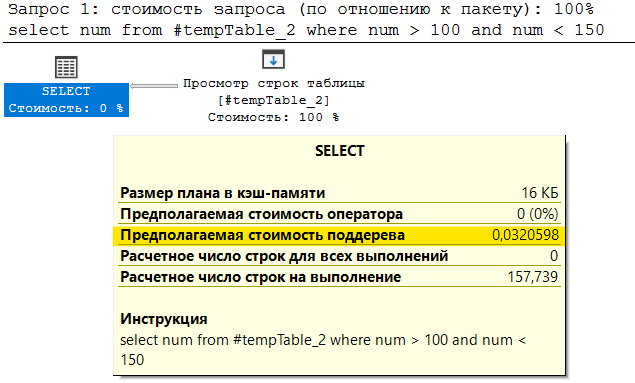
select num from #tempTable\_2 where num > 100

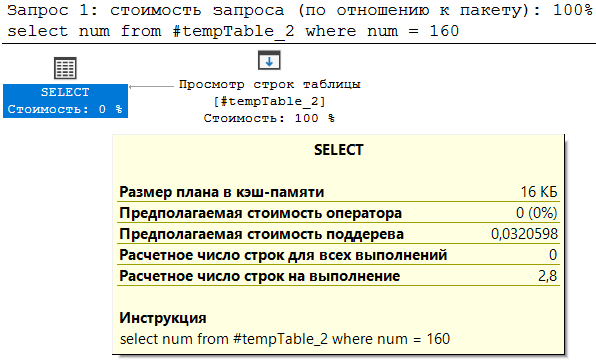
select num from #tempTable\_2 where num > 100 and num < 150

select num from #tempTable\_2 where num = 160

Планы выполнения до создания индекса:



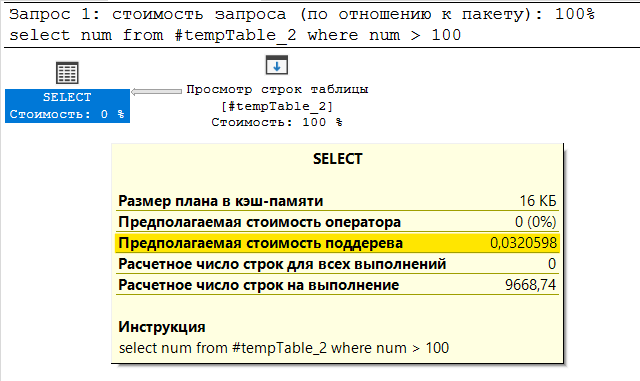


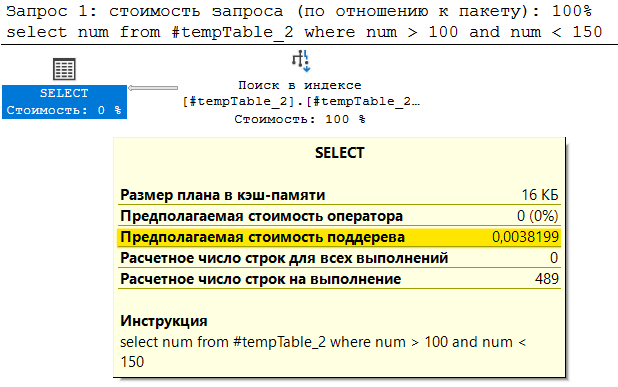


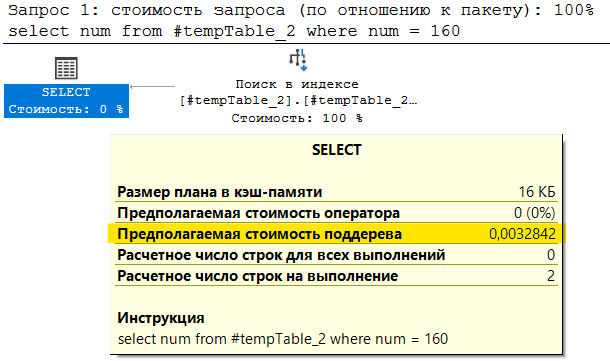
Индекс:

create index #tempTable\_2\_whereNum on #tempTable\_2(num) where (num > 100 and num < 200);

Планы выполнения после создания индекса:







Видно, что индекс применяется, если множество искомых значений в запросе входит в множество значений, на основе которых создан индекс.

Но, если запрос охватывает значения вне диапазона индекса, то индекс применятся не будет.

5.Заполнить временную локальную таблицу.

Создать некластеризованный индекс. Оценить уровень *фрагментации индекса*.

Разработать сценарий на T-SQL, выполнение которого приводит к уровню фрагментации индекса выше 90%. Оценить уровень фрагментации индекса.

Выполнить процедуру *реорганизации* индекса, оценить уровень фрагментации.

Выполнить процедуру *перестройки* индекса и оценить уровень фрагментации индекса.

***→ Теория (методичка)***

Операции добавления и изменения строк базы данных могут повлечь образование неиспользуемых фрагментов в области памяти индекса. Процесс образования неиспользуемых фрагментов памяти называется *фрагментацией*.

Фрагментация индексов снижает эффект от их применения.

Пусть создан индекс:

**CREATE index #EX\_TKEY ON #EX(TKEY);**

Получить информацию о степени фрагментации индекса можно с помощью операторов:

**SELECT name [Индекс], avg\_fragmentation\_in\_percent [Фрагментация (%)]**

**FROM sys.dm\_db\_index\_physical\_stats(DB\_ID(N'TEMPDB'),**

**OBJECT\_ID(N'#EX'), NULL, NULL, NULL) ss**

**JOIN sys.indexes ii on ss.object\_id = ii.object\_id and ss.index\_id = ii.index\_id**

**WHERE name is not null;**

Если вставить 10000 строк с помощью

**INSERT top(10000) #EX(TKEY, TF) select TKEY, TF from #EX;**

и получить данные о фрагментации с помощью операторов, приведенных выше,то можно увидеть, что уровень фрагментации превысит 99%.

Для избавления от фрагментации индекса предусмотрены две специальные операции: реорганизация и перестройка индекса.

*Реорганизация* (REORGANIZE) выполняется быстро, но после нее фрагментация будет убрана только на самом нижнем уровне.

Пусть выполнена реорганизация с помощью оператора ALTER для индекса **#EX\_TKEY.**

**ALTER index #EX\_TKEY on #EX reorganize;**

Тогда выполнение соответствующего запроса покажет, что уровень фрагментации значительно снизился, но не до конца.

Операция *перестройки* (REBUILD) затрагивает все узлы дерева, поэтому после ее выполнения степень фрагментации равна нулю.

Пусть выполнена перестройка с помощью оператора ALTER для индекса **#EX\_TKEY** в режиме **OFFLINE**.

**ALTER index #EX\_TKEY on #EX rebuild with (online = off);**

Выполнением запроса о фрагментации можно оценить ее уровень.

***→ Выполнение заданий***

Удаляю ранее созданные индексы, чтобы сконцентрироваться на одном, который мы должны создать по условию данного задания:

-- некластеризованный неуникальный составной индекс

create index #tempTable\_2\_id\_num\_ind on #tempTable\_2(num, id);

-- drop index #tempTable\_2.#tempTable\_2\_id\_num\_ind

create index #tempTable\_2\_num\_include on #tempTable\_2(num) include (id);

-- drop index #tempTable\_2.#tempTable\_2\_num\_include

-- некластеризованный фильтруемый индекс

create index #tempTable\_2\_whereNum on #tempTable\_2(num) where (num > 100 and num < 200);

-- drop index #tempTable\_2.#tempTable\_2\_whereNum

Создаю индекс:

-- создание некластеризованого индекса

create index #tempTable\_2\_num on #tempTable\_2(num);

Информация о фрагментации всех индексов временной таблицы #tempTable\_2:

-- переключение контекста на БД tempdb с таблицей #tempTable\_2

**use tempdb;**

-- получение информации о индексах таблицы

select indInfo.name as [Индекс],

fragInfo.avg\_fragmentation\_in\_percent as [Фрагментация, %]

from sys.dm\_db\_index\_physical\_stats(db\_id('tempdb'),

object\_id('#tempTable\_2'),

null,

null,

null) as fragInfo

join sys.indexes as indInfo

on fragInfo.object\_id = indInfo.object\_id

and fragInfo.index\_id = indInfo.index\_id

where indInfo.name is not null



Увеличиваем фрагментацию (повторяем операцию около 3 раз для повышения фрагментации индекса свыше 90% или в top(10000000), вместо 10000000 пишем число больше):

-- увеличиваем фрагментацию индекса

insert top(10000000) #tempTable\_2(num, string)

select num, string from #tempTable\_2



Реорганизация:

alter index #tempTable\_2\_num on #tempTable\_2 reorganize



Перестройка:

alter index #tempTable\_2\_num on #tempTable\_2 rebuild with (online = off)



6.Разработать пример, демонстрирующий применение параметра FILLFACTOR при создании некластеризованного индекса.

***→ Теория (методичка)***

Уровнем фрагментации можно в некоторой степени управлять, если при создании или изменении индекса использовать параметры FILLFACTOR и PAD\_INDEX.

Параметр FILLFACTOR указывает процент заполнения индексных страниц нижнего уровня.

Пусть индекс пересоздан со значением параметра FILLFACTOR равным 65:

**DROP index #EX\_TKEY on #EX;**

**CREATE index #EX\_TKEY on #EX(TKEY)**

**with (fillfactor = 65);**

После добавления строк в таблицу **#EX** можно оценить уровеньфрагментации:

**INSERT top(50)percent INTO #EX(TKEY, TF)**

**SELECT TKEY, TF FROM #EX;**

**SELECT name [Индекс], avg\_fragmentation\_in\_percent [Фрагментация (%)]**

**FROM sys.dm\_db\_index\_physical\_stats(DB\_ID(N'TEMPDB'),**

**OBJECT\_ID(N'#EX'), NULL, NULL, NULL) ss JOIN sys.indexes ii**

**ON ss.object\_id = ii.object\_id and ss.index\_id = ii.index\_id**

**WHERE name is not null;**

***→ Выполнение заданий***

-- drop index #tempTable\_2.#tempTable\_2\_num;

create index #tempTable\_2\_num on #tempTable\_2(num)

with(fillfactor = 65)

Фрагментация:

******

7. Создать необходимые индексы и проанализировать планы запросов с использованием этих индексов для таблицы базы данных **X\_MyBASE**.

1.

-- 1.1

-- определить все индексы, которые имеются в БД Ku\_MyBase

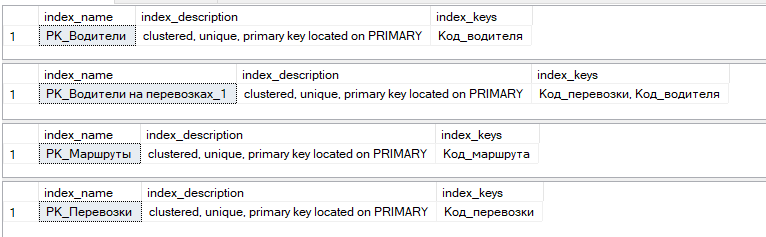
use Ku\_MyBase;

exec sp\_helpindex 'Водители';

exec sp\_helpindex 'Водители на перевозках';

exec sp\_helpindex 'Маршруты';

exec sp\_helpindex 'Перевозки';



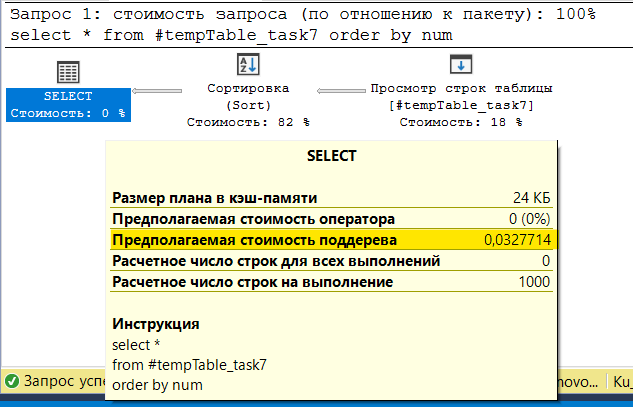
Запрос:

select \*

from #tempTable\_task7

order by num;

План выполнения запроса и стоимость запроса6

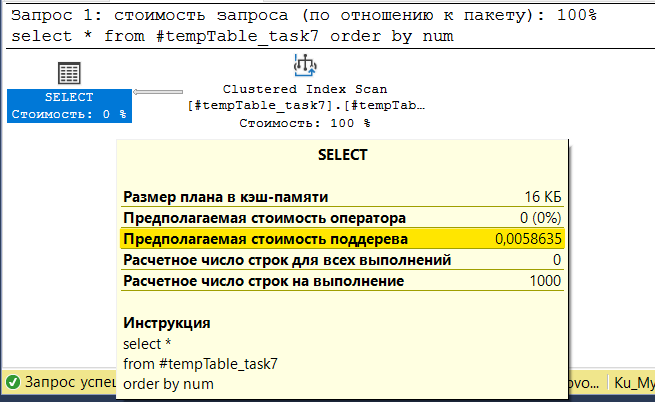


Создаю индекс:

-- кластеризованный индекс

create clustered index #tempTable\_task7\_num\_ind on #tempTable\_task7(num);

Стоимость запроса теперь:



2.

-- заполняю таблицу 10000 строк

declare @ind\_2 int = 0;

while(@ind\_2 < 10000)

begin

insert #tempTable\_task7 values(floor(rand()\*6000), 'UH OH!');

set @ind\_2 += 1;

end;

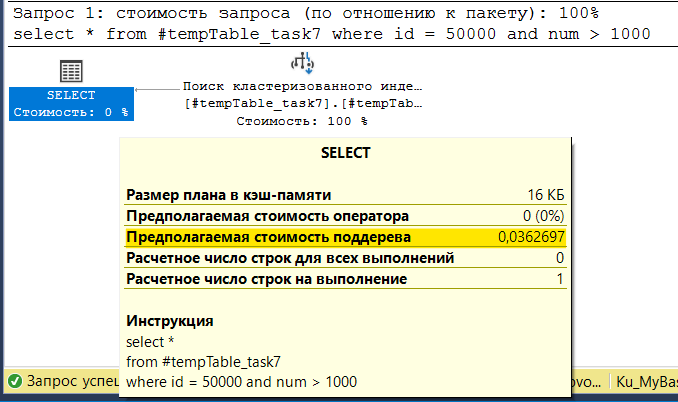
Запрос:

select \*

from #tempTable\_task7

where id = 50000 and num > 1000

Стоимость запроса (сервер использовал индекс, созданный в предыдущем задании):

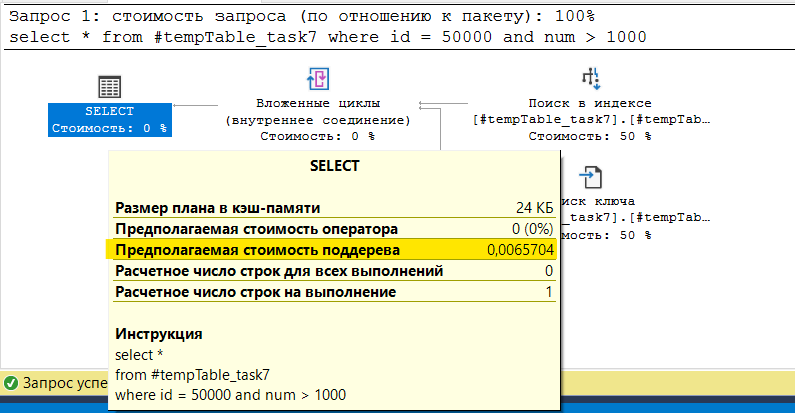


Индекс:

-- некластеризованный неуникальный составной индекс

create index #tempTable\_task7\_composite\_ind on #tempTable\_task7(id, num);

Стоимость:



3.

Удаляю индексы, созданные в предыдущих пунктах, чтобы сервер не задействовал их, нам ведь нужно рассмотреть индекс созданный именно в этом задании:

-- кластеризованный индекс

create clustered index #tempTable\_task7\_num\_ind on #tempTable\_task7(num);

-- drop index #tempTable\_task7.#tempTable\_task7\_num\_ind;

-- некластеризованный неуникальный составной индекс

create index #tempTable\_task7\_composite\_ind on #tempTable\_task7(id, num);

-- drop index #tempTable\_task7.#tempTable\_task7\_composite\_ind;

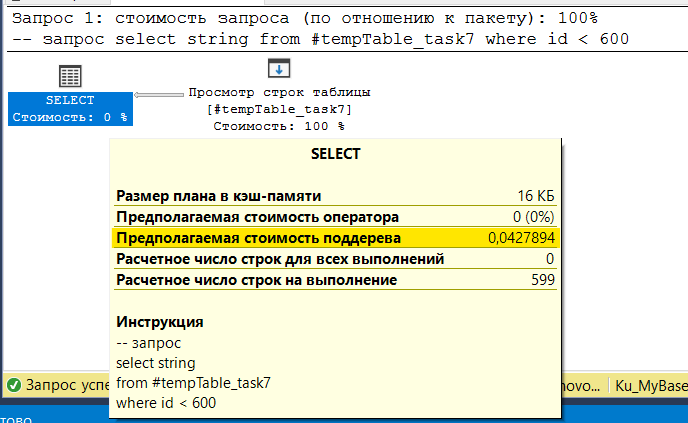
Запрос:

select string

from #tempTable\_task7

where id < 600;

Стоимость запроса:

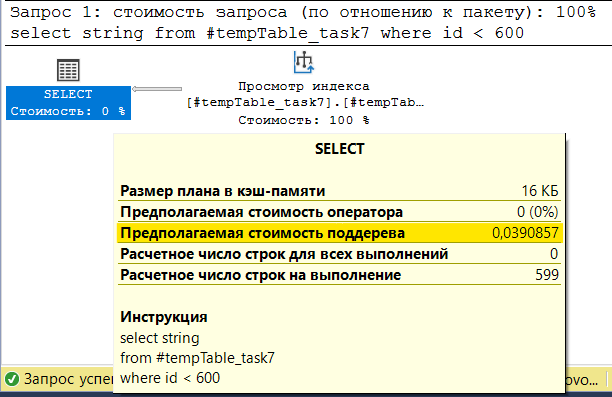


Индекс:

-- некластеризованный индекс покрытия

create index #tempTable\_task7\_id\_inc on #tempTable\_task7(string) include (id);

Стоимость запроса после создания индекса:



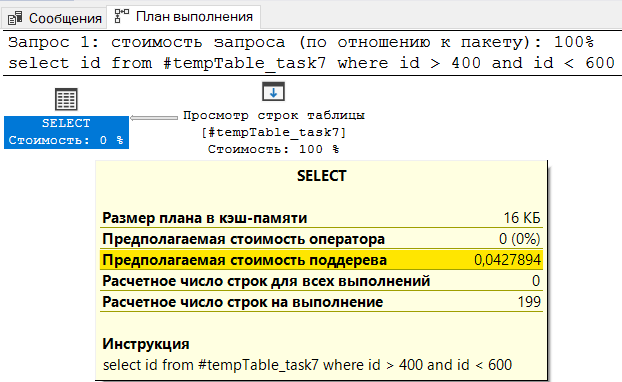
4.

Опять же удаляю индексы из предыдущих пунктов.

Запрос:

select id from #tempTable\_task7 where id > 400 and id < 600;

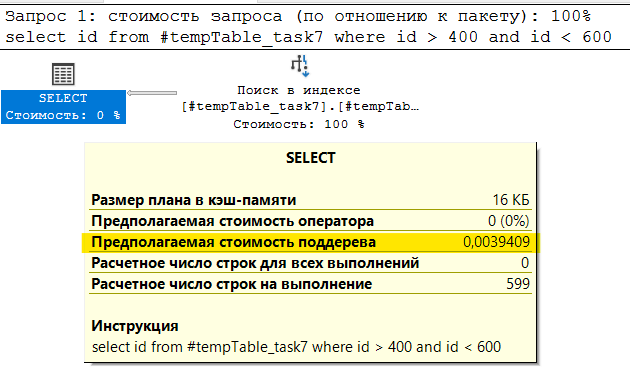
Стоимость:



Индекс:

create index #tempTable\_task7\_where on #tempTable\_task7(id) where (id > 400 and id < 700);

Стоимость после создания индекса:



5.

Индекс:

-- некластеризованный индекс

create index #tempTable\_task7\_string on #tempTable\_task7(string);

Фрагментация:

-- информация о % фрагментации

use tempdb;

select indInfo.name as [Индекс],

fragInfo.avg\_fragmentation\_in\_percent as [Фрагментация, %]

from sys.dm\_db\_index\_physical\_stats(db\_id('tempdb'),

object\_id('#tempTable\_task7'),

null,

null,

null) as fragInfo

join sys.indexes as indInfo

on fragInfo.object\_id = indInfo.object\_id

and fragInfo.index\_id = indInfo.index\_id

where indInfo.name is not null;



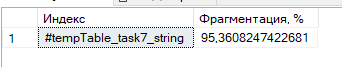
Заполняем таблицу данными, увеличивая фрагментацию (8 раз выполнила запрос):

-- увеличиваем фрагментацию

insert top(99999999999999999) #tempTable\_task7(num, string)

select num, string from #tempTable\_task7;

Фрагментация:



Реорганизация:

-- реогранизация

alter index #tempTable\_task7\_string on #tempTable\_task7 reorganize;

Перестройка:

-- перестройка

alter index #tempTable\_task7\_string on #tempTable\_task7 rebuild with (online = off)

6.

create index #tempTable\_task7\_id on #tempTable\_task7(id)

with (fillfactor = 20);

