Индексы и проблемы производительности

Общие сведения об индексах

Индекс – это объект базы данных, обеспечивающий дополнительные способы быстрого поиска и извлечения данных. Индекс может создаваться на одном или нескольких столбцах. Это означает, что индексы бывают *простыми* и *составными*. Если в таблице нет индекса, то поиск нужных строк выполняется простым сканированием по всей таблице. При наличии индекса время поиска нужных строк можно существенно уменьшить. К недостаткам индексов следует отнести:

a) дополнительное место на диске и в оперативной памяти,
b) замедляются операции вставки, обновления и удаления записей.

В SQL Server индексы хранятся в виде сбалансированных деревьев. Представление индекса в виде сбалансированного дерева означает, что стоимость поиска любой строки остается относительно постоянной, независимо от того, где находится эта строка. Сбалансированное дерево состоит из:

a) *корневого узла* (root node), содержащего одну страницу,
b) нескольких *промежуточных уровней* (intermediate levels), содержащих дополнительные страницы, и
c) *листового уровня* (leaf level).

На страницах листового уровня находятся отсортированные элементы, соответствующие индексируемым данным. По мере добавления данных в таблицу индекс будет разрастаться, но по-прежнему оставаться в форме сбалансированного дерева.

Рассмотрим пример. Известно, что страница данных SQL Server имеет размер 8192 байта и может содержать до 8060 байт пользовательских данных. Определим таблицу CREATE TABLE Tab (Col char(60)); и построим индекс для столбца Col. Если в таблице будет 100 строк с данными, то для их хранения потребуется 6000 байт памяти. Все эти строки разместятся на одной странице данных, поэтому в индексе будет всего одна страница – одновременно на уровне корня и на листовом уровне. На самом деле в таблицу можно занести 134 строки и при этом выделить только одну страницу для индекса.

При добавлении в таблицу 135-й строки SQL Server создаст две дополнительные страницы. Индекс будет состоять из корневой страницы и двух страниц листового уровня. На первой странице листового уровня будет размещаться первая половина таблицы, на второй странице листового уровня – вторая половина таблицы, а на корневой странице – две строки данных. Для этого индекса не нужен промежуточный уровень, потому что корневая страница будет содержать все значения, с которых начинаются страницы листового уровня. Для поиска нужной строки в таблице придется просмотреть ровно две страницы индекса.

Для таблицы, содержащей 17957 строк, индекс будет состоять из 134 страниц листового уровня и корневой страницы. На каждой странице будет размещаться по 134 строки. При добавлении в таблицу 17957-й строки SQL Server должен создать еще одну страницу индекса на листовом уровне, но на корневой странице нельзя разместить 135 строк. Поэтому SQL Server добавит промежуточный уровень, содержащий две страницы. На первой странице будут находиться начальные строки первой половины страниц листового уровня, а на второй - начальные строки второй половины страниц листового уровня. Теперь корневая страница будет содержать две строки, соответствующие начальным значениям двух страниц промежуточного уровня.

Когда в таблицу будет добавлена 2406105-я строка, SQL Server создаст еще один промежуточный уровень и т. д. Очевидно, что такой тип структуры позволяет SQL Server достаточно быстро находить строки, удовлетворяющие запросам, даже в очень больших таблицах. Так, чтобы найти строку в таблице, содержащей около 2500000 строк, SQL Server должен просмотреть всего три страницы индекса. **Конец примера.**

В SQL Server существует два типа индексов:

a) *кластерные* индексы;
b) *некластерные* индексы, которые включают:
    i. некластерные индексы на основе кучи;
    ii. некластерные индексы на основе кластерных индексов.

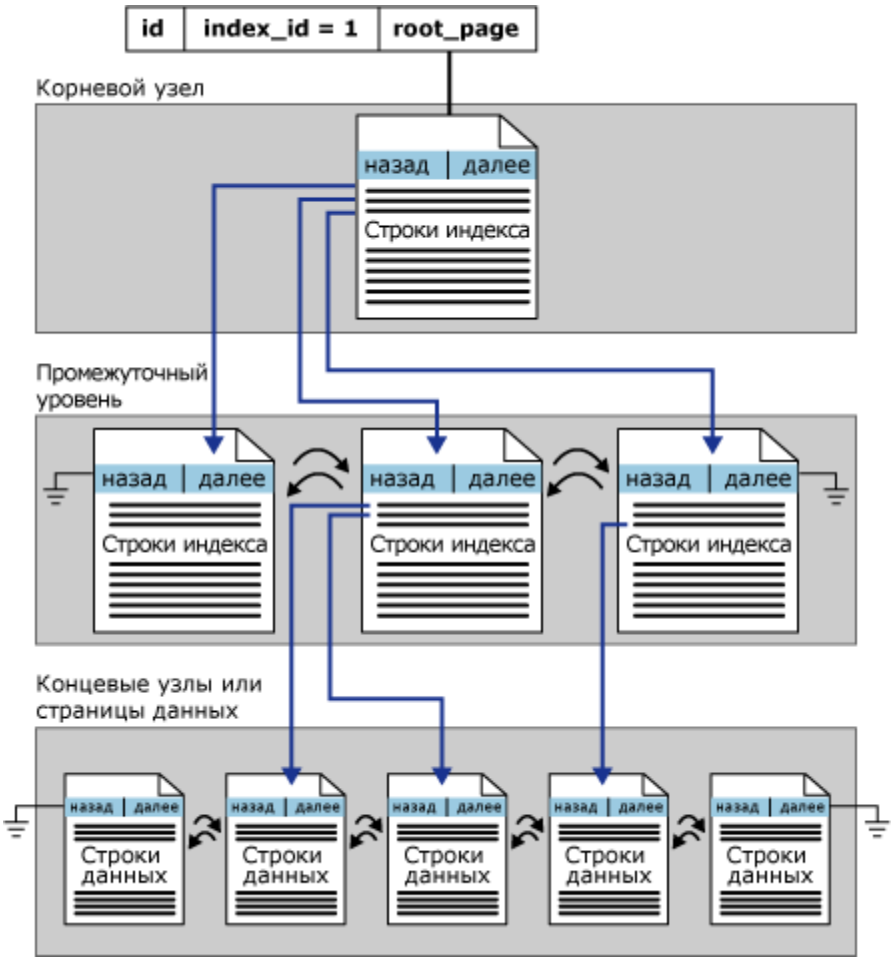
Кластерные индексы

В кластерном индексе таблица представляет собой часть индекса, или индекс представляет собой часть таблицы в зависимости от вашей точки зрения. Листовой узел кластерного индекса – это страница таблицы с данными. Поскольку сами данные таблицы являются частью индекса, то очевидно, что для таблицы может быть создан только один кластерный индекс.

В SQL Server кластерный индекс является уникальным индексом по определению. Это означает, что все ключи записей должны быть уникальные. Если существуют записи с одинаковыми значениями, SQL Server делает их уникальными, добавляя номера из внутреннего (невидимого снаружи) 4-х байтного счетчика. Кластерный индекс использовать необязательно. Тем не менее, кластерные индексы часто используют в качестве первичного ключа. На уровне листьев кластерного индекса информация упорядочивается и физически сохраняется на диске в соответствии с заданными критериями сортировки. На каждом уровне индекса страницы организованы в виде двунаправленного связного списка. Вход в корень кластерного индекса дает поле **root_page** системного представления **sys.system_internals_allocation_units.**

При добавлении каждой новой записи хранимые данные пересортировываются, чтобы сохранить физическую упорядоченность данных. Если требуется создать новую запись в середине индексной структуры, то происходит обычное разбиение страницы. Половина записей из старой страницы переносится в новую, а новая запись добавляется либо в новую страницу, либо в старую страницу (для сохранения равновесия). Если же новая запись логически попадает в конец индексной структуры, тогда в новую создаваемую страницу помещается только новая строка (информация не делится поровну между двумя рассматриваемыми страницами).

На следующем рисунке изображена структура кластерного индекса.



Некластерные индексы на основе кучи

В листьях некластерного индекса на основе кучи хранятся указатели на строки данных. Указатель строится на основе идентификатора файла (ID), номера страницы и номера строки на странице. Весь указатель целиком называется идентификатором строки (RID). С точки зрения физического размещения данных – они хранятся в полном беспорядке. С точки зрения технической реализации нужные записи приходится искать по всему файлу. Вполне может статься, что в этом случае SQL Server придется несколько раз возвращаться к одной и той же странице для чтения различных строк, поскольку определить заранее, откуда придется физически считывать следующую строку нет никакой возможности.

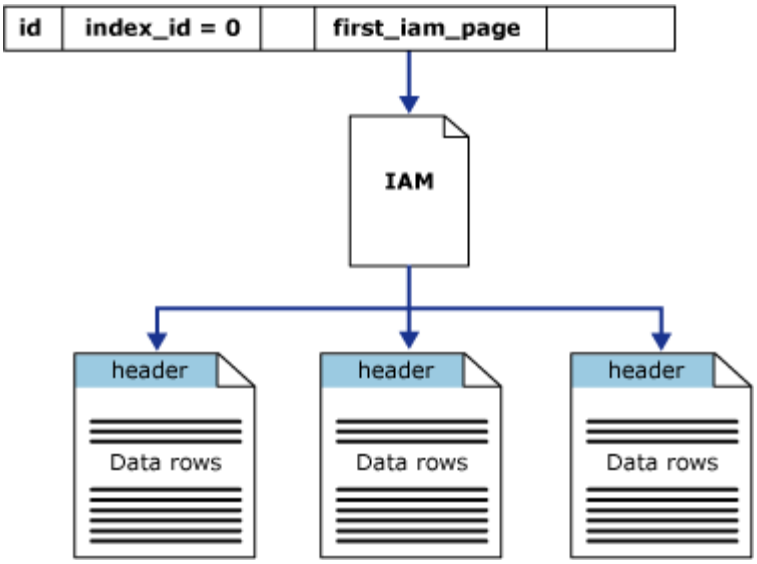
Некластерные индексы, основанные на кластерных таблицах

В листьях некластерного индекса, основанного на кластерных таблицах, хранятся указатели на корневые узлы кластерных индексов. Поиск в таком индексе состоит из двух этапов:

a) поиск в некластерном индексе и
b) поиск в кластерном индексе.

Замечание. Столбец **first_iam_page** в системном представлении **sys.system_internals_allocation_units** указывает на первую IAM-страницу в цепи IAM-страниц, управляющей выделением пространства в куче. SQL Server использует IAM-страницы для перемещения по куче. Страницы данных и строки в этих страницах не расположены в каком-либо порядке и не связаны. Единственным логическим соединением страниц данных являются данные, записанные в IAM-страницы.

На следующей иллюстрации демонстрируется, как компонент SQL Server Database Engine использует IAM-страницы для получения строк данных из кучи с одной секцией.



Индексы могут создаваться двумя способами:

a) явно при помощи инструкции CREATE INDEX;
b) неявно в качестве объекта при создании ограничения.

Создание индекса с помощью инструкции CREATE INDEX

Упрощенный синтаксис инструкции следующий:

CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX *имя-индекса*
ON *имя-таблицы-или-представления* **(** *список-столбцов* **)**
[INCLUDE (*список-столбцов*)]
[WITH *список-опций*]
[ON *файловая-группа*]

Замечания.

a) Для каждого столбца в списке столбцов могут указываться две взаимоисключающие опции ASC/DESC, которые позволяют выбрать способ сортировки индекса. По умолчанию используется вариант ASC, что означает сортировку по возрастанию.
b) Предложение INCLUDE указывает неключевые столбцы, добавляемые к ключевым столбцам некластеризованного индекса.
c) Следующие за ключевым словом WITH опции являются необязательными. Их можно использовать в любом сочетании. Одни (как например PAD_INDEX, IGNORE_DUP_KEY, DROP_EXISTING, STATISTICS_NORECOMPUTE, SORT_IN_TEMPDB) используются довольно редко, другие (как например FILLFACTOR) существенно влияют на быстродействие и алгоритм работы системы. Описания опций индекса находятся по адресу http://msdn.microsoft.com/ru-ru/library/ms188783.aspx.

d) С помощью предложения ON можно задать место хранения индекса. По умолчанию индекс помещается в ту же файловую группу, где находится таблица или представление, для которых он создан.

Пример создания индекса.

```
CREATE UNIQUE NONCLUSTERED INDEX
[IX_Address_AddressLine1_AddressLine2_City_StateProvinceID_PostalCode] ON
[Person].[Address]
([AddressLine1] ASC, [AddressLine2] ASC, [City] ASC, [StateProvinceID] ASC, [PostalCode]
ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY
= OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
```

Индексы, создаваемые вместе с ограничениям

Такой тип индексов часто называют «связанными индексами». Связанные индексы создаются при добавлении одного из следующих двух типов ограничений:
a) ограничения первичного ключа (PRIMARY KEY);
b) ограничения уникальности (UNIQUE).

Пример создания индекса.

```
CREATE TABLE [Person].[Address](
[AddressID] [int] IDENTITY(1,1) NOT FOR REPLICATION NOT NULL,
...
CONSTRAINT [PK_Address_AddressID] PRIMARY KEY CLUSTERED
([AddressID] ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
) ON [PRIMARY]
```

Анализ запросов

Как индексация влияет на производительность?

```
USE AdventureWorks;
GO
SELECT name AS index_name, STATS_DATE(object_id, index_id) AS statistics_update_date
FROM sys.indexes
WHERE object_id = OBJECT_ID('Person.Address');
GO
```

```
PK_Address_AddressID    2012-02-27 22:22:03.257
AK_Address_rowguid      2012-02-27 22:22:04.910
IX_Address_AddressLine1_AddressLine2_City_StateProvinceID_PostalCode    2012-02-27 22:22:04.987
IX_Address_StateProvinceID          2012-02-27 22:22:05.033
```

Использование статистики для повышения производительности запросов
http://msdn.microsoft.com/ru-ru/library/ms190397(v=sql.105).aspx#StatisticsOptions

Анализ запроса
http://msdn.microsoft.com/ru-ru/library/ms191227(v=sql.105).aspx

В справочнике «Значки графических планов выполнения» по адресу
http://msdn.microsoft.com/ru-ru/library/ms175913(v=sql.105).aspx
приведены изображения значков, ранжированных по категориям:

1) Основные операторы (60, цвет синий)
2) Операторы курсора (7, цвет желтый)
3) Операторы параллелизма (3, цвет синий)
4) Операторы T-SQL (8, цвет зеленый)

В справочнике «Справочник по логическим и физическим операторам» по адресу
http://msdn.microsoft.com/ru-ru/library/ms191158(v=sql.105).aspx
приведен список из 107 операторов, которые классифицируются как логические и физические.
Логические операторы описывают операции реляционной алгебры на концептуальном уровне. Физические операторы фактически реализуют операцию, определенную логическим оператором, используя конкретный метод или алгоритм. План запроса — это дерево физических операторов. Каждый физический оператор является объектом или процедурой, выполняющей операцию. Например, "Inner Join" является логическим оператором внутреннего соединения, "Nested Loops" является физическим оператором, который реализует различные логические операторы соединения (внутреннее соединение, левое внешнее соединене, левое полусоединение и антилевое полусоединение), "Table-valued Function" является одновременно логическим и физическим оператором, который вычисляет функцию, возвращающую табличное значение, а "UDX" - это группа из 8-ми логических и физическимх операторов, реализующих часть операций XQuery и XPath. Каждый физический оператор является объектом или процедурой, выполняющей операцию.

Замечание. Над операторами первой группы, выполняющимися параллельно, отображается значок параллельного процесса (черный на желтом).

Чтобы создать план выполнения запроса необходимо:
a) Иметь разрешения на выполнение запросов, для которых создается план выполнения, и разрешение SHOWPLAN для всех баз данных, на которые ссылается запрос.
b) Открыть файл с существующим запросом или создать его заново.
c) В контексном меню редактора запросов выбрать **Включить действительный план выполнения** (Include Actual Execution Plan) или нажать соответствующую кнопку на панели инструментов.
d) Выполнить запрос. План запроса отображается на вкладке **План выполнения** в области результатов.

Останавливая указатель мыши над логическими и физическими операторами, можно просмотреть описание и свойства операторов во всплывающих подсказках. Также можно просмотреть свойства оператора в окне **Свойства.**

Графически план выполнения запроса представляется в виде дерева (корень слева, листья справа), которое читается справа налево и сверху вниз. Интерпретация графического отображения плана выполнения запроса:
1) Каждый узел дерева представлен в виде значка, указывающего логический и физический оператор, используемый для выполнения этой части запроса.
2) Каждый узел связан со своим родительским узлом. Дочерние узлы одного родительского узла отображаются в одном столбце. Однако все узлы в одном столбце не обязательно имеют общий родительский узел. Правила со стрелками на конце соединяют каждый узел с его родителем.

Как оптимизатор запросов использует статистику для создания планов запросов

Статистика для оптимизации запросов — это объекты, содержащие статистические сведения о распределении значений в одном или нескольких столбцах. Оптимизатор запросов использует эти сведения для оценки числа строк в результатах запроса. Например, оптимизатор запросов, используя статистику, может выбрать оператор *index seek* вместо оператора *index scan*.
Каждый объект статистики создается по индексу или списку из одного или нескольких столбцов и содержит:
a) заголовок, содержащий метаданные о статистике,
b) гистограмму, содержащую распределение значений в первом ключевом столбце объекта статистики, и
c) вектор плотностей для измерения корреляции с охватом нескольких столбцов.

Для отображения статистики можно использовать:
a) Properties в контекстном меню Statistics в окне Object Explorer в среде SSMS, например, последовательно раскрывая узлы Databases|AdventureWorks|Tables|Person.Address|Statistics и click правой кнопкой по IX_Address_AddressLine1_AddressLine2_City_StateProvinceID_PostalCode и Properties
b) инструкцию DBCC SHOW_STATISTICS, например,

```
DBCC SHOW_STATISTICS ("Person.Address",
IX_Address_AddressLine1_AddressLine2_City_StateProvinceID_PostalCode)
WITH STAT_HEADER, HISTOGRAM
```

```
GO

AddressID   AddressLine1            AddressLine2      City        StateProvinceID
      PostalCode
29076             Attaché de Presse       NULL        Saint-Denis       179
      93400

RANGE_HI_KEY            RANGE_ROWS  EQ_ROWS     DISTINCT_RANGE_ROWS
      AVG_RANGE_ROWS
------------------ ------------ ------------ ------------ ----------- ---------------
Attaché de Presse       121         17          117                     1,034188
```

Для большинства запросов оптимизатор создает статистику самостоятельно, но в некоторых случаях для повышения производительности запросов нужно создать дополнительные статистические данные с помощью инструкции CREATE STATISTICS. В этой инструкции с помощью предиката фильтра в предложении WHERE можно создавать отфильтрованную статистику.

Описания столбцов, возвращаемых в результирующий набор, если указан параметр HISTOGRAM.

Имя столбца	Описание
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).

Когда создается и обновляется статистика

a) Явно с помощью инструкцией CREATE STATISTICS и UPDATE STATISTICS и с помощью системной хранимой процедуры sp_updatestats.
b) Неявно с помощью пераметров статистики AUTO_CREATE_STATISTICS, AUTO_UPDATE_STATISTICS и AUTO_UPDATE_STATISTICS_ASYNC, действующих на уровне базы данных и устанавливаемых в инструкции ALTER DATABASE.

AUTO_CREATE_STATISTICS { ON | OFF }
AUTO_UPDATE_STATISTICS { ON | OFF }
AUTO_UPDATE_STATISTICS_ASYNC { ON | OFF }

Для определения времени последнего обновления статистики пользуйтесь функцией STATS_DATE.

Статистика и выбор индексов

Возникает вопрос – если таблица имеет один кластерный индекс и четыре не кластерных, как SQL Server узнает какой из индексов использовать. Распределение статистики дает query optimiser'y возможность выбрать определенный индекс.

Распределение статистики

Начиная с SQL Server 2000, статистика хранится в поле данных для хранения имиджей. То есть – растет пропорционально размеру индекса, это позволяет не терять точность при увеличении размера индексов.
Чтобы понять что из себя представляет статистика, рассмотрим следующий пример. Возьмем таблицу Orders из базы данных Northwind. Если мы выполним следующий запрос:

```
SELECT TOP 24 OrderID, convert(char(11), OrderDate) as "Order Date"
FROM Orders ORDER BY OrderDate

OrderID     Order Date
----------- -----------
      10248 Jul  4 1996
      10249 Jul  5 1996
      10250 Jul  8 1996
      10251 Jul  8 1996
      10252 Jul  9 1996
      10253 Jul 10 1996
      10254 Jul 11 1996
      10255 Jul 12 1996
      10256 Jul 15 1996
      10257 Jul 16 1996
      10258 Jul 17 1996
      10259 Jul 18 1996
      10260 Jul 19 1996
      10261 Jul 19 1996
      10262 Jul 22 1996
      10263 Jul 23 1996
      10264 Jul 24 1996
      10265 Jul 25 1996
      10266 Jul 26 1996
      10267 Jul 29 1996
      10268 Jul 30 1996
      10269 Jul 31 1996
      10270 Aug  1 1996
      10271 Aug  1 1996

(24 rows affected)
```

В этом примере выбраны записи с 4 июля 1996 по 1 августа 1996. Теперь посчитаем, сколько раз встречаются те или иные значения:

```
SELECT CONVERT(char(11), A.OrderDate) as "Order Date" , count(*) as "# of OrderDate"
FROM(SELECT TOP 24 OrderID, OrderDate FROM Orders ORDER BY OrderDate) as A GROUP BY
A.OrderDate

Order Date  # of OrderDate
----------- --------------
Jul  4 1996              1
Jul  5 1996              1
Jul  8 1996              2
Jul  9 1996              1
Jul 10 1996              1
Jul 11 1996              1
Jul 12 1996              1
Jul 15 1996              1
Jul 16 1996              1
Jul 17 1996              1
Jul 18 1996              1
Jul 19 1996              2
Jul 22 1996              1
Jul 23 1996              1
Jul 24 1996              1
Jul 25 1996              1
Jul 26 1996              1
Jul 29 1996              1
Jul 30 1996              1
```

```
Jul 31 1996              1
Aug  1 1996              2

(21 rows affected)
```

Когда SQL Server считает или сортирует данные, он заранее знает как много тех или иных значений он найдет в указанном запросе. Например, выполним следующий запрос:

```
SELECT OrderID, CustomerID, EmployeeID, OrderDate
FROM Orders WHERE OrderDate BETWEEN '1996-07-15' AND '1996-07-20'

OrderID     CustomerID EmployeeID  OrderDate
----------- ---------- ----------- -----------------------
      10256 WELLI                3 1996-07-15 00:00:00.000
      10257 HILAA                4 1996-07-16 00:00:00.000
      10258 ERNSH                1 1996-07-17 00:00:00.000
      10259 CENTC                4 1996-07-18 00:00:00.000
      10260 OTTIK                4 1996-07-19 00:00:00.000
      10261 QUEDE                4 1996-07-19 00:00:00.000

(6 rows affected)
```

SQL Server знает что запрос вернет только 6 записей еще до того как будет произведен доступ к таблице. Назначение статистики базируется на простом алгоритме: для выбора стратегии доступа к данным, SQL Server должен знать, как много записей вернет запрос.

В SQL Server статистика не просто учитывает количество записей для того или иного значения. Первое, статистика более точно описывает распределение данных по значениям. Второе, зона описания статистики может не содержать всех значений индексируемого поля. Например, в таблице Orders 830 записей, но статистика хранится только для 186 значений. Как и почему именно 187 – этот алгоритм не рассматривается в данном издании. Выполним команду

```
dbcc show_statistics (Orders, OrderDate)
```

Этот результат показывает следующее

- статистика была просчитана в последний раз 23 декабря 2002
- таблица содержит 830 записей
- все записи были проанализированы для получения статистики
- в статистике информация сохранена в дискретности на 187 записей
- средняя плотность распределения примерно 0.17%

Пожалуй самым интересным здесь будет значение плотности распределения. Если каждое значение в таблице уникальное, то плотность будет 1/830, то есть 0.12%. Но в нашем примере мы имеем 0.17% показывает что некоторые значения встречаются 2 и более раз. Например мы видим что для 8 июля 1996 встречается дважды. Теория говорит, что чем меньше плотность, тем лучше – это увеличивает избирательность, а следовательно и ценность построенного индекса.

Например если колонка содержит только 3 значения, плотность распределения будет равна 33.3%, что показывает бесполезность построения индекса по данному полю. Индексы занимают место на диске и в оперативной памяти и отнимает быстродействие. В идеале самый лучший индекс имеет плотность распределения равную единице, деленной на количество записей в таблице - все записи уникальны. При построении индексов, обращайте внимание на плотность распределения – если она превышает 10%, то индекс можно считать бесполезным. Сканирование по таблице в таком случае будет более эффективным.

Второй результат, возвращаемый командой dbcc show_statistics
Это дает очень интересный результат потому что плотность для одного поля OrderDate 0.2%, а для пары OrderDate, OrderID уже 0.12. Поскольку OrderID является primary key для таблице, то понижение плотности вполне очевидно.

Последний результат может быть наиболее интересен для полного понимания важности статистики. Во-первых, заметим, что только пять значений в зоне распределения вместо 24 в самой таблице. Тем не менее учитывая значения по всем колонками, система знает сколько записей будет в выборке. Колонка RANGE_HI_KEY дает высшее значение для значения, сохраненного в статистике. Мы знаем что 1996-07-04 является первым значением и следующим за ним идет 1996-07-15. Между этими двумя значениями находится 7 записей. Колонка RANGE_ROWS дает нам эту информацию. Только три

значения есть между 1996-07-15 и 1996-07-19, и так далее. Колонка DISTINCT_RANGE_ROWS содержит информацию о том сколько определенных (неповторяющихся) значений в интервале. Например в интервале с 1996-07-04 по 1996-07-15 есть 6 определенных значений из 7 записей в интервале. Это говорит что одно значение в указанном интервале повторяется дважды. Мы не указали еще одну колонку в результате команды dbcc show_statistics - AVG_RANGE_ROWS, которое является результатом простой арифметической операции RANGE_ROWS /DISTINCT_RANGE_ROWS.

Анализируя эту информацию, можно определить, как распределены данные в индексе, а так же как много записей вернет запрос.

Выбор индексов

Как мы видели в предыдущих примерах, SQL Server достаточно точно знает, как данные распределены по таблице при наличии индекса. Каждый раз при выполнении запроса, SQL Server первым делом оценивает статистику. Давайте рассмотрим несколько примеров для более полного понимания данного процесса.

```
SELECT * FROM Orders WHERE OrderDate BETWEEN '1996-07-19' AND '1996-07-25'
```

SQL Server сначала проверяет существования индекса по полю OrderDate или составного индекса, начинающегося с поля OrderDate. В таком случае SQL Server знает, как много записей вернется в результате запроса. Query Analyser дает следующую картинку:
Для того что бы посмотреть детали каждого этапа выполнения запроса, просто выберете нужный этап и кликните правой клавишей мыши. Мы видим, что оценочное количество записей – 6, что и реально соответствует действительности.
Теперь выполним этот запрос:

```
SELECT * FROM Orders WHERE OrderDate BETWEEN '1996-07-19' AND '1996-07-25' AND CustomerID
= 'FOLKO'
```

SQL Server найдет один индекс OrderDate и один CustomerID. Он будет использовать их обоих и результатом будет пересечение по этим двум условиям. Оценка дает 6 значений по OrderDate и только одно для CustomerID. В этом конкретном случае SQL Server вернет только одно значение.

Если заменить условие операции AND на OR, система может сделать объединение индексов или может предпочесть полное сканирование таблицы, если сочтет что такой путь будет более быстрым и имеет меньшую стоимость чем работа с индексами. В нашем конкретном случае в результате запроса:

```
SELECT * FROM Orders WHERE OrderDate BETWEEN '1996-07-19' AND '1996-07-25' OR CustomerID
= 'FOLKO'
```

Было произведено полное сканирование таблицы, так как стоимость операции OR оказалась выше. Это объясняется тем, что размеры таблицы сравнительно малы.

Что же произойдет, если для таблицы нет индексов, следовательно, нет статистики? Ответ прост: SQL Server не может работать без статистики! И он создаст статистику без каких либо индексов. Если выполнить запрос:

```
SELECT * FROM Orders WHERE ShipCity='Grass'
```

SQL Server автоматически создаст статистику для этого поля, поскольку нет индекса по нему. Для того что бы просмотреть все индексы, созданные для определенной таблицы, выполним запрос:

```
SELECT name, first, root  FROM sysindexes WHERE id=OBJECT_ID('Orders')
```

Мы получили результат:

```
Name                           first    root
------------------------------ -------- --------
PK_Orders                      0xC90000 0xCC0000
CustomerID                     0x380100 0x3B0100
CustomersOrders                0x3D0100 0x400100
EmployeeID                     0x420100 0x450100
EmployeesOrders                0x460100 0x490100
```

```
OrderDate                      0x4A0100 0x4D0100
ShippedDate                    0x4E0100 0x510100
ShippersOrders                 0x520100 0x550100
ShipPostalCode                 0x560100 0x590100
_WA_Sys_0000000B_08EA5793      NULL     NULL

(10 rows affected)
```

Заметьте имя “индекса” _WA_Sys_0000000B_08EA5793 с root адресам NULL – потому что это не индекс. Это статистика для поля ShipCity. Вам не стоит беспокоится о засорении вашей базы данных ненужной информацией – автоматически созданная статистика будет уничтожена так же автоматически когда SQL Server посчитает что она более не нужна. Просто доверьтесь SQL Server. (Хорош оборот, неправда ли!?)

Вы можете так же изучить наличие статистики при помощи команды:

```
sp_helpstats 'Orders'

statistics_name                statistics_keys
-----------------------------  ---------------
_WA_Sys_0000000B_08EA5793      ShipCity
```

Автоматическое обновление статистики является огромной помощью для DBA. Порой крайне сложно определить, какая же статистика должна быть создана и эту функцию на себя взял SQL Server. Существует только одна проблема - поддерживать корректную на каждый момент времени статистику.

Обслуживание статистики

Что произойдет если статистика устареет и не будет отражать реальную картину с данными в таблице. Ответ очень прост – выбор индекса может быть неверен. Представим, что статистика была собрана, когда в таблице было только 1 000 записей, а теперь ее размер 100 000. Что бы быть реально полезной, статистика должна содержать текущие реальные данные.

SQL Server позволяет обновлять статистику автоматически без привлечения дополнительных усилий со стороны DBA. Проверить, установлен ли флаг автоматического обновления статистики, можно командой:

```
SELECT DATABASEPROPERTYEX('Northwind', 'IsAutoUpdateStatistics')

(No column name)
----------------
1
```

Если результат равен 1, это значит, что опция автоматического обновления статистики включена. Установить опцию в этот режим можно командой

```
ALTER DATABASE Northwind SET AUTO_UPDATE_STATISTICS ON
```

а выключить

```
ALTER DATABASE Northwind SET AUTO_UPDATE_STATISTICS OFF
```

Так же эту операцию можно установить используя хранимую процедуру sp_dboption, но в SQL Server 2000 она оставлена только для обратной совместимости и может исчезнуть в будущих версиях.

Рекомендуется оставлять эту функцию включенной. В этом случает SQL Server будет сам обновлять статистику когда посчитает ее устаревшей. Алгоритм обновления полностью определяется SQL Server и зависит от количества обновлений, удалений и добавлений записей в таблицу. Если таблица имеет один миллион записей и только 100 из них были изменены (0.01%), вряд ли имеет смысл обновлять статистику поскольку такие изменения в таблице вряд ли драматически поменяли общую картину данных.
Кроме того если размер таблицы более 8МБ (1 000 страниц), SQL Server не будет использовать все данные для вычисления статистики. Все эти ограничения разработаны для того что бы работа со обновлением статистики наносила как можно меньший удар на быстродействие сервера.

Так же управлять автоматическим обновлением статистики можно при помощи хранимых процедур типа sp_autostats:
Команда

```
sp_autostats 'Orders'

Global statistics settings for [Northwind]:
  Automatic update statistics: ON
  Automatic create statistics: ON

settings for table [Orders]

Index Name                     AUTOSTATS Last Updated
-----------------------------  --------- -----------------------
[PK_Orders]                    ON        2012-06-20 21:35:35.680
[CustomerID]                   ON        2012-06-20 21:35:35.693
[CustomersOrders]              ON        2012-06-20 21:35:35.700
[EmployeeID]                   ON        2012-06-20 21:35:35.700
[EmployeesOrders]              ON        2012-06-20 21:35:35.703
[OrderDate]                    ON        2012-06-20 21:35:35.707
[ShippedDate]                  ON        2012-06-20 21:35:35.710
[ShippersOrders]               ON        2012-06-20 21:35:35.717
[ShipPostalCode]               ON        2012-06-20 21:35:35.720
[_WA_Sys_0000000B_08EA5793]    ON        2013-03-21 23:28:18.767

(10 rows affected)
```

показывает включена ли опция автоматического обновления статистики для конкретных индексов и когда обновление было сделано в последний раз. Так же эта команда позволяет включить или отключить опцию обновление для всех индексов таблицы или для какого-то конкретного индекса. Итого что бы отключить автоматическое обновление статистики у вас есть несколько вариантов:

c) ALTER DATABASE dbname SET AUTO_UPDATE_STATISTICS
d) sp_autostats
e) используйте STATISTICS_NORECOMPUTE в команде CREATE INDEX
f) используйте NORECOMPUTE в STATISTICS UPDATES или CREATE STATISTICS

Как только автоматическое обновление будет отключено, вы будете вынуждены обновлять статистику вручную. Эта операция может быть выполнена при сопровождении индексов или операцией UPDATE STATISTICS.
Полное описание UPDATE STATISTICS стоит изучить по BOL.

Создание статистики

Создание индексов и статистики достаточно простой процесс и выполняется командами CREATE INDEX и CREATE STATISTICS. Давайте посмотрим как же индексы создаются.

Статистика

Мы видели что система может сама создавать статистику для полей таблицы когда индекс не существует. Это работает, но отнимает ресурсы при выполнении запроса. Более того – автоматически создаваемая статистика создается только для одного поля, а для оптимизатора может быть необходима статистика по нескольким полям сразу. Для изучения синтаксиса CREATE STATISTICS обратитесь к BOL. Помните что статистика оказывает воздействие на решения принимаемые оптимизатором при построении плана запроса, но не оказывает воздействие на скорость выполнения запроса. Эта функция лежит на индексах.