Лекция 11. Индексы.

Оглавление

Общие сведения

Системы баз данных обычно используют индексы для обеспечения быстрого доступа к реляционным данным. Индекс представляет собой отдельную физическую структуру данных, которая позволяет получать быстрый доступ к одной или нескольким строкам данных. Таким образом, правильная настройка индексов является ключевым аспектом улучшения производительности запросов.

Индекс базы данных во многом сходен с индексом (алфавитным указателем) книги. Когда нам нужно быстро найти какую-либо тему в книге, мы сначала смотрим в индексе, на каких страницах книги эта тема рассматривается, а потом сразу же открываем нужную страницу. Подобным образом, при поиске определенной строки таблицы компонент Database Engine обращается к индексу, чтобы узнать ее физическое местонахождение.

Но между индексом книги и индексом базы данных есть две существенные разницы.

- Читатель книги имеет возможность самому решать, использовать ли индекс в каждом конкретном случае или нет. Пользователь базы данных такой возможности не имеет, и за него это решение принимает компонент системы, называемый оптимизатором запросов.
- Индекс для определенной книги создается вместе с книгой, после чего он больше не изменяется. Это означает, что индекс для определенной темы всегда будет указывать на один и тот же номер страницы. В противоположность, индекс базы данных может меняться при каждом изменении соответствующих данных.

Если для таблицы отсутствует подходящий индекс, для выборки строк система использует метод сканирования таблицы. Выражение сканирование таблицы означает, что система последовательно извлекает и исследует каждую строку таблицы (от первой до последней), и помещает строку в результирующий набор, если для нее удовлетворяется условие поиска в предложении **`WHERE`**. Таким образом, все строки извлекаются в соответствии с их физическим расположением в памяти. Этот метод менее эффективен, чем доступ с использованием индексов.

Индексы сохраняются в дополнительных структурах базы данных, называющихся страницами индексов.

Для каждой индексируемой строки имеется элемент индекса (index entry), который сохраняется на странице индексов. Каждый элемент индекса состоит из ключа индекса и указателя. Вот поэтому элемент индекса значительно короче, чем строка таблицы, на которую он указывает. По этой причине количество элементов индекса на каждой странице индексов намного больше, чем количество строк в странице данных. Это свойство индексов играет очень важную роль, поскольку количество операций ввода/вывода, требуемых для прохода по страницам индексов, значительно меньше, чем количество операций ввода/вывода, требуемых для прохода по соответствующим страницам данных. Иными словами, для сканирования таблицы, скорей всего, потребовалось бы намного больше операций ввода/вывода, чем для сканирования индекса этой таблицы.

Индексы компонента Database Engine создаются, используя структуру данных сбалансированного дерева B^+. B^+-дерево имеет древовидную структуру, в которой все самые нижние узлы (листья) находятся на расстоянии одинакового количества уровней от вершины (корневого узла) дерева. Это свойство поддерживается даже тогда, когда в индексированный столбец добавляются или удаляются данные.

На рис. 1 показана структура B^+-дерева для таблицы **`employee`** и прямой доступ к строке в этой таблице со значением **`25348`** для столбца **`emp_no`**. (Предполагается, что таблица **`employee`** проиндексирована по столбцу **`emp_no`**.) На этом рисунке можно также видеть, что B^+-дерево состоит из корневого узла, листьев дерева (узлов листьев) и промежуточных узлов, количество которых может быть от нуля и больше.

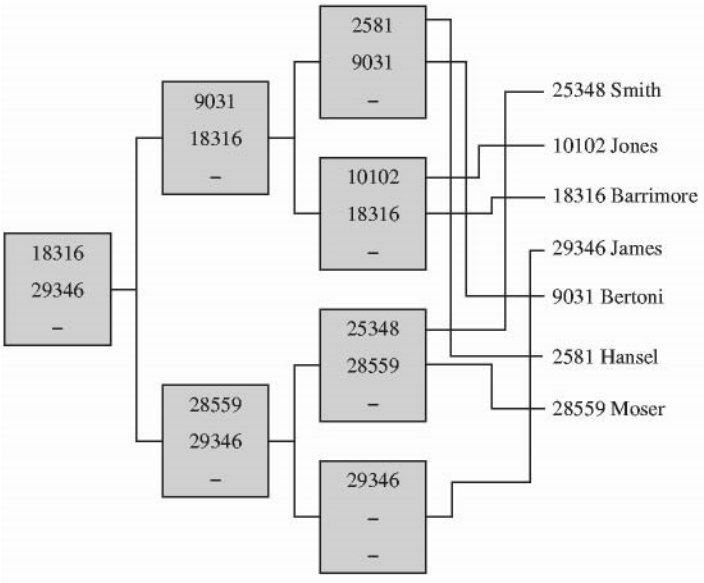


Рис. 1. B^+-дерево для столбца emp_no таблицы employee

Поиск в этом дереве значения **25348** можно выполнить следующим образом. Начиная с корня дерева, выполняется поиск наименьшего значения ключа, большего или равного требуемому значению. Таким образом, в корневом узле таким значением будет **29346**, поэтому делается переход на промежуточный узел, связанный с этим значением. В этом узле заданным требованиям отвечает значение **28559**, вследствие чего выполняется переход на листья дерева, связанный с этим значением. Этот узел и содержит искомое значение **25348**. Определив требуемый индекс, мы можем извлечь его строку из таблицы данных с помощью соответствующих указателей. (Альтернативным эквивалентным подходом будет поиск меньшего или равного значения индекса.)

Индексированный поиск обычно является предпочтительным методом поиска в таблицах с большим количеством строк по причине его очевидного преимущества. Используя индексированный поиск, мы можем найти любую строку в таблице за очень короткое время, применив лишь несколько операций ввода/вывода. А последовательный поиск (т. е. сканирование таблицы от первой строки до последней) требует тем больше времени, чем дальше находится требуемая строка.

Рассмотрим два существующих типа индексов, кластеризованные и некластеризованные.

Кластеризованные индексы

Кластеризованный индекс определяет физический порядок данных в таблице. Компонент Database Engine позволяет создавать для таблицы лишь один кластеризованный индекс, т. к. строки таблицы нельзя упорядочить физически более чем одним способом. Поиск с использованием кластеризованного индекса выполняется от корневого узла B^+-дерева по направлению к листьям дерева, которые связаны между собой в двунаправленный связанный список (doubly linked list), называющийся цепочкой страниц (page chain). Важным свойством кластеризованного индекса является та особенность, что его листья дерева (узлы-листья) содержат страницы данных. (Узлы кластеризованного индекса всех других уровней содержат страницы индекса.) Таблица, для которой определен кластеризованный индекс (явно или неявно), называется кластеризованной таблицей. Структура B^+-дерева кластеризованного индекса показана на рис. 2.

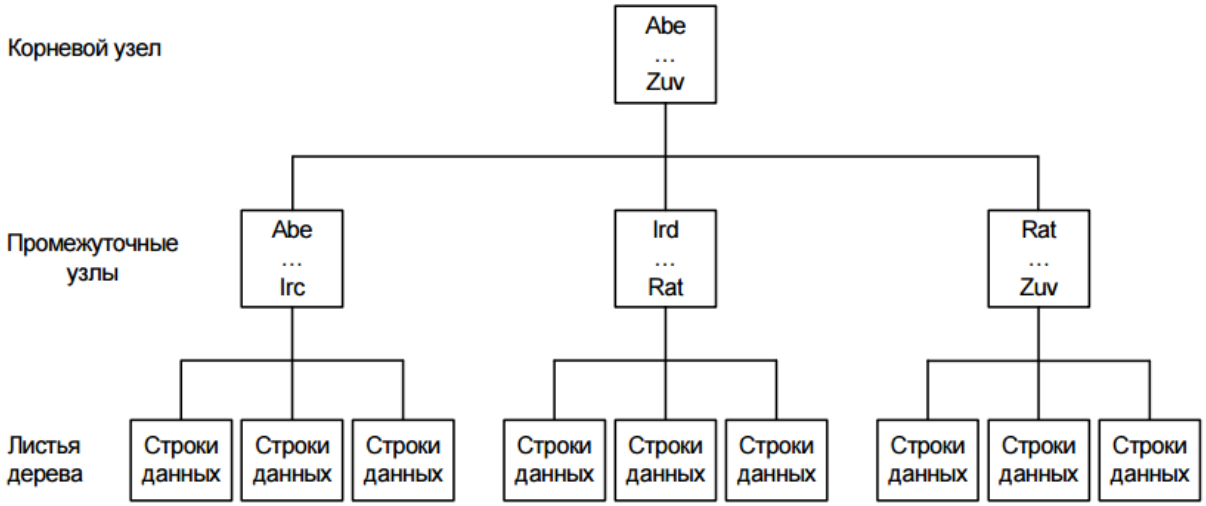


Рис. 2. Физическая структура кластеризованных индексов

Кластеризованный индекс создается по умолчанию для каждой таблицы, для которой с помощью ограничения первичного ключа определен первичный ключ. Кроме этого, каждый кластеризованный индекс однозначен по умолчанию, т. е. в столбце, для которого определен кластеризованный индекс, каждое значение данных может встречаться только один раз. Если кластеризованный индекс создается для столбца, содержащего повторяющиеся значения, система баз данных принудительно обеспечивает однозначность, добавляя четырехбайтовый идентификатор к строкам, содержащим дубликаты значений.

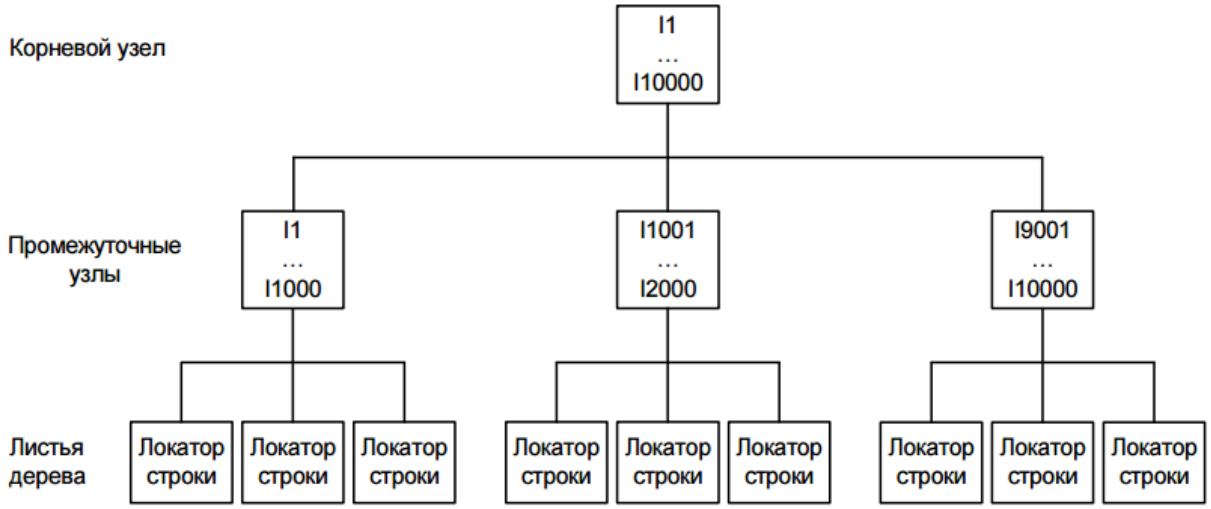
Кластеризованные индексы обеспечивают очень быстрый доступ к данным, когда запрос осуществляет поиск в диапазоне значений

Некластеризованные индексы

Структура некластеризованного индекса точно такая же, как и кластеризованного, но с двумя важными отличиями:

1. некластеризованный индекс не изменяет физическое упорядочивание строк таблицы;
2. страницы листьев некластеризованного индекса состоят из ключей индекса и закладок.

Если для таблицы определить один или более некластеризованных индексов, физический порядок строк этой таблицы не будет изменен. Для каждого некластеризованного индекса компонент Database Engine создает дополнительную индексную структуру, которая сохраняется в индексных страницах. Структура B^+- дерева некластеризованного индекса показана на рис. 3.



Локатор строки = Идентификатор строки (RID) или указатель на журнал кластеризованного индекса

Рис. 3. Структура некластеризованного индекса

Закладка в некластеризованном индексе указывает, где находится строка, соответствующая ключу индекса. Составляющая закладки ключа индекса может быть двух видов, в зависимости от того, является ли таблица кластеризованной таблицей или кучей (heap). (Согласно терминологии SQL Server, кучей называется таблица без кластеризованного индекса.) Если существует кластеризованный индекс, то закладка некластеризованного индекса показывает B^+-дерево кластеризованного индекса таблицы. Если таблица не имеет кластеризованного индекса, закладка идентична идентификатору строки (RID — Row Identifier), состоящего из трех частей: адреса файла, в котором хранится таблица, адреса физического блока (страницы), в котором хранится строка, и смещения строки в странице.

Как уже упоминалось ранее, поиск данных с использованием некластеризованного индекса можно осуществлять двумя разными способами, в зависимости от типа таблицы:

- куча — прохождение при поиске по структуре некластеризованного индекса, после чего строка извлекается, используя идентификатор строки;
- кластеризованная таблица — прохождение при поиске по структуре некластеризованного индекса, после чего следует прохождение по соответствующему кластеризованному индексу.

В обоих случаях количество операций ввода/вывода довольно велико, поэтому следует подходить к проектированию некластеризованного индекса с осторожностью, и применять его только в том случае, если есть уверенность, что его использование существенно повысит производительность.

Язык Transact-SQL и индексы

Теперь, когда мы познакомились с физической структурой индексов, рассмотрим, как их создавать, изменять и удалять, а также как получать информацию о фрагментации индексов и редактировать информацию об индексах. Все это подготовит нас к последующему обсуждению использования индексов для улучшения производительности системы.

Создание индексов

Индекс для таблицы создается с помощью инструкции **CREATE INDEX**. Эта инструкция имеет следующий синтаксис:

```
CREATE [UNIQUE] [CLUSTERED |NONCLUSTERED INDEX index_name
      ON table_name (column1 [ASC DESC],...)
  [INCLUDE (colunm_name [,... ])]
      [WITH
         [FILLFACTOR=n]
         [[,] PAD_INDEX = {ON | OFF}]
         [[,] DROP_EXISTING = {ON | OFF}]
         [[,] SORT_IN_TEMPDB = {ON | OFF}]
         [[,] IGNORE_DUP_KEY = {ON | OFF}]
         [[,] ALLOW_ROW_LOCKS = {ON | OFF}]
         [[,] ALLOW_PAGE_LOCKS = {ON | OFF}]
         [[,] STATISTICS_NORECOMPUTE = {ON | OFF}]
         [[,] ONLINE = {ON | OFF}]]
         [ON file_group | "default"]
```

Параметр **index_name** задает имя создаваемого индекса. Индекс можно создать для одного или больше столбцов одной таблицы, обозначаемой параметром **table_name**. Столбец, для которого создается индекс, указывается параметром **column1**. Числовой суффикс этого параметра указывает на то, что индекс можно создать для нескольких столбцов таблицы. Компонент Database Engine также поддерживает создание индексов для представлений.

Можно проиндексировать любой столбец таблицы. Это означает, что столбцы, содержащие значения типа данных VARBINARY(max), BIGINT и SQL_VARIANT, также могут быть индексированы.

Индекс может быть простым или составным. Простой индекс создается по одному столбцу, а составной индекс — по нескольким столбцам. Для составного индекса существуют определенные ограничения, связанные с его размером и количеством столбцов. Индекс может иметь максимум 900 байтов и не более 16 столбцов.

Параметр **UNIQUE** указывает, что проиндексированный столбец может содержать только однозначные (т. е. неповторяющиеся) значения. В однозначном составном индексе однозначной должна быть комбинация значений всех столбцов каждой строки. Если ключевое слово **UNIQUE** не указывается, то повторяющиеся значения в проиндексированном столбце (столбцах) разрешаются.

Параметр **CLUSTERED** задает кластеризованный индекс, а параметр **NONCLUSTED** (применяется по умолчанию) указывает, что индекс не изменяет порядок строк в таблице. Компонент Database Engine разрешает для таблицы максимум 249 некластеризованных индексов.

Возможности компонента Database Engine были расширены, позволяя поддержку индексов с убывающим порядком значений столбцов. Параметр **ASC** после имени столбца указывает, что индекс создается с возрастающим порядком значений столбца, а параметр **DESC** означает убывающий порядок значений столбца индекса. Таким образом, в использовании индекса предоставляется большая гибкость. С убывающим порядком следует создавать составные индексы на столбцах, значения которых упорядочены в противоположных направлениях.

Параметр **INCLUDE** позволяет указать неключевые столбцы, которые добавляются к страницам листьев некластеризованного индекса. Имена столбцов в списке **INLCUDE** не

должны повторяться, и столбец нельзя использовать одновременно как ключевой и неключевой. Чтобы по-настоящему понять полезность параметра **INCLUDE**, нужно понимать, что собой представляет покрывающий индекс (covering index). Если все столбцы запроса включены в индекс, то можно получить значительное повышение производительности, т. к. оптимизатор запросов может определить местонахождение всех значений столбцов по страницам индекса, не обращаясь к данным в таблице. Такая возможность называется покрывающим индексом или покрывающим запросом. Поэтому включение в страницы листьев некластеризованного индекса дополнительных неключевых столбцов позволит получить больше покрывающих запросов, при этом их производительность будет значительно повышена.

Параметр **FILEFACTOR=n** задает заполнение в процентах каждой страницы индекса во время его создания. Значение параметра **FILEFACTOR** можно установить в диапазоне от 1 до 100. При значении **n=100** каждая страница индекса заполняется на 100%, т. е. существующая страница листа так же, как страница, не относящаяся к листу, не будет иметь свободного места для вставки новых строк. Поэтому это значение рекомендуется применять только для статических таблиц. (Значение по умолчанию, **n=0**, означает, что страницы листьев индекса заполняются полностью, а каждая из промежуточных страниц содержит свободное место для одной записи.) При значении параметра **FILEFACTOR** между **1** и **99** страницы листьев создаваемой структуры индекса будут содержать свободное место. Чем больше значение **n**, тем меньше свободного места в страницах листьев индекса. Например, при значении **n=60** каждая страница листьев индекса будет иметь 40% свободного места для вставки строк индекса в дальнейшем. (Строки индекса вставляются посредством инструкции **INSERT** или **UPDATE**.) Таким образом, значение **n=60** будет разумным для таблиц, данные которых подвергаются довольно частым изменениям. При значениях параметра **FILEFACTOR** между **1** и **99** промежуточные страницы индекса содержат свободное место для одной записи каждая.

После создания индекса в процессе его использования значение FILEFACTOR не поддерживается. Иными словами, оно только указывает объем зарезервированного места с имеющимися данными при задании процентного соотношения для свободного места. Для восстановления исходного значения параметра FILEFACTOR применяется инструкция ALTER INDEX

Параметр **PAD_INDEX** тесно связан с параметром **FILEFACTOR**. Параметр **FILEFACTOR** в основном задает объем свободного пространства в процентах от общего объема страниц листьев индекса. А параметр **PAD_INDEX** указывает, что значение параметра **FILEFACTOR** применяется как к страницам индекса, так и к страницам данных в индексе.

Параметр **DROP_EXISTING** позволяет повысить производительность при воспроизведении кластеризованного индекса для таблицы, которая также имеет некластеризованный индекс.

Параметр **SORT_IN_TEMPDB** применяется для помещения в системную базу данных **tempdb** данных промежуточных операций сортировки, применяющихся при создании индекса. Это может повысить производительность, если база данных **tempdb** размещена на другом диске, чем данные.

Параметр **IGNORE_DUP_KEY** разрешает системе игнорировать попытку вставки повторяющихся значений в индексированные столбцы. Этот параметр следует применять только для того, чтобы избежать прекращения выполнения длительной транзакции, когда инструкции **INSERT** вставляет дубликат данных в индексированный столбец. Если этот параметр активирован, то при попытке инструкции **INSERT** вставить в таблицу строки, нарушающие однозначность индекса, система базы данных вместо аварийного завершения

выполнения всей инструкции просто выдает предупреждение. При этом компонент Database Engine не вставляет строки с дубликатами значений ключа, а просто игнорирует их и добавляет правильные строки. Если же этот параметр не установлен, то выполнение всей инструкции будет аварийно завершено.

Когда параметр **ALLOW_ROW_LOCKS** активирован (имеет значение **ON**), система применяет блокировку строк. Подобным образом, когда активирован параметр **ALLOW_PAGE_LOCKS**, система применяет блокировку страниц.

Параметр **STATISTICS_NORECOMPUTE** определяет состояние автоматического перерасчета статистики указанного индекса.

Активированный параметр **ONLINE** позволяет создавать, пересоздавать (rebuild) и удалять индекс в диалоговом режиме. Данный параметр позволяет в процессе изменения индекса одновременно изменять данные основной таблицы или кластеризованного индекса и любых связанных индексов. Например, в процессе пересоздания кластеризованного индекса можно продолжать обновлять его данные и выполнять запросы по этим данным.

Параметр **ON** создает указанный индекс или на файловой группе по умолчанию (значение **default**), или на указанной файловой группе (значение **file_group**).

В примере 1 показано создание некластеризованного индекса.

```
USE library;
CREATE INDEX i_title ON books (title);
```

Создание однозначного составного индекса показано в примере 2.

```
USE lirary;
CREATE UNIQUE INDEX i_title_author
 ON books (title, author)
WITH FILLFACTOR= 80;
```

В примере 2 значения в каждом столбце должны быть однозначными. При создании индекса заполняется 80% пространства каждой страницы листьев индекса.

Создание однозначного индекса для столбца невозможно, если этот столбец содержит повторяющиеся значения. Такой индекс можно создать лишь в том случае, если каждое значение (включая значение **NULL**) встречается в столбце только один раз. Кроме этого, любая попытка вставить или изменить существующее значение данных в столбец, включенный в существующий уникальный индекс, будет отвергнута системой в случае дублирования значения.

Получение информации о фрагментации индекса

В течение жизненного цикла индекса он может подвергнуться фрагментации, вследствие чего процесс хранения данных в страницах индекса станет неэффективным. Существует два типа фрагментации индекса: внутренняя фрагментация и внешняя фрагментация. Внутренняя фрагментация определяет объем данных, хранящихся в каждой странице, а внешняя фрагментация возникает при нарушении логического порядка страниц.

Для получения информации о внутренней фрагментации индекса применяется динамическое административное представление (ДАП), называемое **sys.dm_db_index_physical_stats**. Это ДАП (DMV) возвращает информацию об объеме и фрагментации данных и индексов указанной страницы. Для каждой страницы возвращается одна строка для каждого уровня B^+-дерева. С помощью этого ДАП можно получить информацию о

степени фрагментации строк в страницах данных, на основе которой можно принять решение о необходимости реорганизации данных.

Использование представления **sys.dm_db_index_physical_stats** показано в примере 3.

```
DECLARE @db_id INT;
DECLARE @tab_id INT;
DECLARE @ind_id INT;
SET @db_id = DB_ID('library');
SET @tab_id = OBJECT_ID('books');
SELECT avg_fragmentation_in_percent, avg_page_space_used_in_percent
   FROM sys.dm_db_index_physical_stats
(@db_id, @tab_id, NULL, NULL, NULL)
```

Как видно из примера 3, представление **sys.dm_db_index_physical_stats** имеет пять параметров. Первые три параметра определяют идентификаторы текущей базы данных, таблицы и индекса соответственно. Четвертый параметр задает идентификатор раздела, а последний определяет уровень сканирования, применяемый для получения статистической информации. (Значение по умолчанию для определенного параметра можно указать посредством значения **NULL**.)

Наиболее важными из столбцов этого представления являются столбцы **avg_fragmentation_in_percent** и **avg_ page_space_used_in_percent**. В первом указывается средний уровень фрагментации в процентах, а во втором определяется объем занятого пространства в процентах.

Редактирование информации индекса

После ознакомления с информацией о фрагментации индекса эту и другую информацию индекса можно редактировать с помощью следующих системных средств:

- представления каталога **sys.indexes**;
- представления каталога **sys.index_columns**;
- системной процедуры **sp_helpindex**;
- функции свойств **OBJECTPROPERTY**;
- среды управления Management Studio сервера SQL Server;
- динамического административного представления (ДАП) **sys.dm_db_index_usage_stats**;
- динамического административного представления (ДАП) **sys.dm_db_missing_index_details**.

Представление каталога **sys.indexes** содержит строку для каждого индекса и строку для каждой таблицы без кластеризованного индекса. Наиболее важными столбцами этого представления каталога являются столбцы **object_id**, **name** и **index_id**. Столбец **object_id** содержит имя объекта базы данных, которой принадлежит индекс, а столбцы **name** и **index_id** содержат имя и идентификатор этого индекса соответственно.

Представление каталога **sys.index_columns** содержит строку для каждого столбца, являющегося частью индекса или кучи. Эту информацию можно использовать совместно с информацией, полученной посредством представления каталога **sys.indexes**, для получения дополнительных сведений о свойствах указанного индекса.

Системная процедура **sp_helpindex** возвращает данные об индексах таблицы, а также статистическую информацию для столбцов. Эта процедура имеет следующий синтаксис: **sp_helpindex [@db_object =] 'name'**

Здесь переменная **@db_object** представляет имя таблицы.

Применительно к индексам, функция свойств **OBJECTPROPERTY** имеет два свойства: **IsIndexed** и **IsIndexable**. Первое свойство предоставляет сведения о наличии индекса у таблицы или представления, а второе указывает, поддается ли таблица или представление индексированию.

Для редактирования информации существующего индекса с помощью среды SQL Server Management Studio выберите требуемую базу данных в папке Databases (базы данных), разверните узел Tables (таблицы), в этом узле разверните требуемую таблицу и ее папку Indexes (индексы). В папке таблицы отобразится список всех существующих индексов для данной таблицы. Двойной щелчок мышью по индексу откроет диалоговое окно Index Properties со свойствами этого индекса. (Создать новый индекс или удалить существующий можно также с помощью среды Management Studio.)

Представление **sys.dm_db_index_usage_stats** возвращает подсчет разных типов операций с индексами и время последнего выполнения каждого типа операции. Каждая отдельная операция поиска, просмотра или обновления по указанному индексу при исполнении одного запроса считается использованием индекса и увеличивает на единицу значение соответствующего счетчика в этом ДАП (DMV). Таким образом можно получить общую информацию о частоте использования индекса, чтобы на ее основе определить, какие индексы используются больше, а какие меньше.

Представление **sys.dm_db_missing_index_details** возвращает подробную информацию о столбцах таблицы, для которых отсутствуют индексы. Наиболее важными столбцами этого ДАП являются столбцы **index_handle** и **object_id**. Значение в первом столбце определяет конкретный отсутствующий индекс, а во втором — таблицу, в которой отсутствует индекс.

Изменение индексов

Компонент Database Engine является одной из немногих систем баз данных, которые поддерживают инструкцию **ALTER INDEX**. Эту инструкцию можно использовать для выполнения операций по обслуживанию индекса. Синтаксис инструкции **ALTER INDEX** очень сходен с синтаксисом инструкции **CREATE INDEX**. Иными словами, эта инструкция позволяет изменять значения параметров **ALLOW_ROW_LOCKS**, **ALLOW_PAGE_LOCKS**, **IGNORE_DUP_KEY** и **STATISTICS_NORECOMPUTE**, которые были описаны ранее при рассмотрении инструкции **CREATE INDEX**.

Кроме вышеперечисленных параметров, инструкция **ALTER INDEX** поддерживает три другие параметра:

- параметр **REBUILD**, используемый для пересоздания индекса;
- параметр **REORGANIZE**, используемый для реорганизации страниц листьев индекса;
- параметр **DISABLE**, используемый для отключения индекса.

Пересоздание индекса

При любом изменении данных, используя инструкции **INSERT**, **UPDATE** или **DELETE**, возможна фрагментация данных. Если эти данные проиндексированы, то также возможна фрагментация индекса, когда информация индекса оказывается разбросанной по разным физическим страницам. В результате фрагментации данных индекса компонент Database Engine может быть вынужден выполнять дополнительные операции чтения данных, что понижает общую производительность системы.

В таком случае требуется пересоздать (rebuild) все фрагментированные индексы.

Это можно сделать двумя способами:

1. посредством параметра **REBUILD** инструкции **ALTER INDEX**;
2. посредством параметра **DROP_EXISTING** инструкции **CREATE INDEX**.

Параметр **REBUILD** применяется для пересоздания индексов. Если для этого параметра вместо имени индекса указать **ALL**, будут вновь созданы все индексы таблицы. (Разрешив динамическое пересоздание индексов, не нужно будет удалять и создавать их заново.)

Параметр **DROP_EXISTING** инструкции **CREATE INDEX** позволяет повысить производительность при пересоздании кластеризованного индекса таблицы, которая также имеет некластеризованные индексы. Он указывает, что существующий кластеризованный или некластеризованный индекс нужно удалить и создать заново указанный индекс. Как упоминалось ранее, каждый некластеризованный индекс в кластеризованной таблице содержит в своих листьях дерева соответствующие значения кластеризованного индекса таблицы. По этой причине при удалении кластеризованного индекса таблицы требуется создать вновь все ее некластеризованные индексы. Использование параметра **DROP_EXISTING** позволяет избежать повторного пересоздания некластеризованных индексов.

Параметр DROP_EXISTING более мощный, чем параметр REBUILD, поскольку он более гибкий и предоставляет несколько опций, таких как изменение столбцов, составляющих индекс, и изменение некластеризованного индекса в кластеризованный.

Реорганизация страниц листьев индекса

Параметр **REORGANIZE** инструкции **ALTER INDEX** задает реорганизацию страниц листьев указанного индекса, чтобы физический порядок страниц совпадал с их логическим порядком — слева направо. Это удаляет определенный объем фрагментации индекса, повышая его производительность.

Отключение индекса

Параметр **DISABLE** отключает указанный индекс. Отключенный индекс недоступен для применения, пока он не будет снова включен. Обратите внимание, что отключенный индекс не изменяется при внесении изменений в соответствующие данные. По этой причине, чтобы снова использовать отключенный индекс, его нужно полностью создать вновь. Для включения отключенного индекса применяется параметр **REBUILD** инструкции **ALTER TABLE**.

При отключенном кластеризованном индексе таблицы данные этой таблицы будут недоступны, так как все страницы данных таблицы с кластеризованным индексом хранятся в его листьях дерева.

Удаление и переименование индексов

Для удаления индексов в текущей базе данных применяется инструкция **DROP INDEX**. Обратите внимание, что удаление кластеризованного индекса таблицы может быть очень ресурсоемкой операцией, т. к. потребуется пересоздать все некластеризованные индексы. (Все некластеризованные индексы используют ключ индекса кластеризованного индекса, как указатель в своих страницах листьев.) Использование инструкции **DROP INDEX** для удаления индекса показано в примере 4.

```
USE library;
DROP INDEX i_title ON books;
```

Инструкция **DROP INDEX** имеет дополнительный параметр, **MOVE TO**, значение которого аналогично параметру **ON** инструкции **CREATE INDEX**. Иными словами, с помощью этого параметра можно указать, куда переместить строки данных, находящиеся в страницах листьев кластеризованного индекса. Данные перемещаются в новое место в виде кучи. Для нового места хранения данных можно указать или файловую группу по умолчанию, или именованную файловую группу.

Инструкцию DROP INDEX нельзя использовать для удаления индексов, которые создаются неявно системой для ограничений целостности, таких индексов, как PRIMARY KEY и UNIQUE. Чтобы удалить такие индексы, нужно удалить соответствующее ограничение.

Индексы можно переименовывать с помощью системной процедуры **sp_rename**.

Рекомендации по созданию и использованию индексов

Хотя компонент Database Engine не накладывает никаких практических ограничений на количество индексов, по паре причин это количество следует ограничивать. Во-первых, каждый индекс занимает определенный объем дискового пространства, следовательно, существует вероятность того, что общее количество страниц индекса базы данных может превысить количество страниц данных в базе. Во-вторых, в отличие от получения выгоды при использовании индекса для выборки данных, вставка и удаление данных такой выгоды не предоставляют по причине необходимости обслуживания индекса. Чем больше индексов имеет таблица, тем больший требуется объем работы по их реорганизации. Общим правилом будет разумно выбирать индексы для частых запросов, а затем оценивать их использование.

Некоторые рекомендации по созданию и использованию индексов предоставляются в этом разделе.

Последующие рекомендации являются всего лишь общими правилами. В конечном итоге их эффективность будет зависеть от способа использования базы данных на практике и типа наиболее часто выполняемых запросов. Индексирование столбца, который никогда не будет использоваться, не принесет никакой пользы.

Индексы и условия предложения WHERE

Если предложение **WHERE** инструкции **SELECT** содержит условие поиска с одним столбцом, то для этого столбца следует создать индекс. Это особенно рекомендуется при высокой селективности условия. Под селективностью (selectivity) условия имеется в виду соотношение количества строк, удовлетворяющих условию, к общему количеству строк в таблице. Высокой селективности соответствует меньшему значению этого соотношения. Обработка поиска с использованием индексированного столбца будет наиболее успешной при селективности условия, не превышающей 5%.

Столбец не следует индексировать при постоянном уровне селективности условия 80% или более. В таком случае для страниц индекса потребуются дополнительные операции ввода/вывода, которые уменьшат любую экономию времени, достигаемую за счет использования индексов. В этом случае быстрее выполнять поиск сканированием таблицы, что и будет обычно выбрано оптимизатором запросов, делая индекс бесполезным.

Если условие поиска часто используемого запроса содержит операторы **AND**, лучше всего будет создать составной индекс по всем столбцам таблицы, указанным в предложении **WHERE** инструкции **SELECT**. Создание такого составного индекса показано в примере 5.

```
USE library;
CREATE INDEX i_exp ON exemplar(isbn, date_in);
```

```
SELECT isbn, in_date, present
   FROM exemplar
   WHERE isbn = 29346 AND date_in='1.4.2016';
```

В этом запросе оператором **AND** соединены два условия, поэтому для обоих столбцов в этих условиях следует создать составной некластеризованный индекс.

Индексы и оператор соединения

В случае операции соединения рекомендуется создавать индекс для каждого соединяемого столбца. Соединяемые столбцы часто представляют первичный ключ одной из таблицы и соответствующий внешний ключ другой таблицы. Если указываются ограничения для обеспечения целостности **PRIMARY KEY** и **FOREIGN KEY** для соответствующих соединяемых столбцов, следует создать только некластеризованный индекс для столбца внешнего ключа, т. к. система неявно создаст кластеризованный индекс для столбца первичного ключа.

В примере 6 показано создание индексов, которые будут использованы, если у вас есть запрос с операцией соединения и дополнительным фильтром.

```
USE library;
SELECT isbn, readers.name
   FROM exemplar, readers
   WHERE exemplar.num_reader = readers.num_reader
   AND date_in = '10.5.2016';
```

Для запроса в примере 6 рекомендуется создать два отдельных индекса для столбца **num_reader** как в таблице **exemplar**, так и в таблице **readers**. Кроме этого, следует создать дополнительный индекс для столбца **date_in**.

Покрывающий индекс

Как уже упоминалось ранее, включение всех столбцов запроса в индекс может значительно повысить производительность запроса. Создание такого индекса, называемого покрывающим (covering), показано в примере 7.

```
CREATE INDEX i_address_zip
   ON Person.Address (PostalCode)
   INCLUDE (City, Street);
GO
SELECT City, Street FROM Person.Address
   WHERE PostalCode = 83000;
```

В примере 7 создается новый индекс, который помимо столбца **PostalCode** включает два дополнительных столбца. Наконец, инструкция **SELECT** в конце примера показывает запрос, покрываемый индексом. Для этого запроса системе нет необходимости выполнять поиск данных в страницах данных, поскольку оптимизатор запросов может найти все значения столбцов в страницах листьев некластеризованного индекса.

Покрывающие индексы рекомендуется применять по той причине, что страницы индексов обычно содержат намного больше записей, чем соответствующие страницы данных. Кроме этого, для того чтобы использовать этот метод, фильтруемые столбцы должны быть первыми ключевыми столбцами в индексе.

Специальные типы индексов

Компонент Database Engine позволяет создавать следующие специальные типы индексов:

- индексированные представления;
- фильтруемые индексы;
- индексы для вычисляемых столбцов;
- секционированные индексы;
- индексы сохранения столбца;
- XML-индексы;
- полнотекстовые индексы.

В этом разделе рассматриваются вычисляемые столбцы и связанные с ними индексы.

Вычисляемым столбцом называется столбец таблицы, в котором сохраняются результаты вычислений данных таблицы. Такой столбец может быть виртуальным или постоянным.

Виртуальные вычисляемые столбцы

Вычисляемый столбец, который не имеет соответствующего кластеризованного индекса, является логическим, т. е. он физически на жестком диске не хранится. Таким образом, он вычисляется при каждом обращении к строке. Использование виртуальных вычисляемых столбцов показано в примере 8.

```
CREATE TABLE orders
 (orderid INT NOT NULL,
  price MONEY NOT NULL,
  quantity INT NOT NULL,
  orderdate DATETIME NOT NULL,
  total AS price * quantity,
  shippeddate AS DATEADD (DAY, 7, orderdate)
 );
```

Таблица **orders** в примере 8 имеет два виртуальных вычисляемых столбца: **total** и **shippeddate**. Столбец **total** вычисляется с использованием двух других столбцов, **price** и **quantity**, а столбец **shippeddate** вычисляется при использовании функции **DATEADD** и столбца **orderdate**.

Постоянные вычисляемые столбцы

Компонент Database Engine позволяет создавать индексы для детерминированных вычисляемых столбцов, где базовые столбцы имеют точные типы данных. (Вычисляемый столбец называется детерминированным, если всегда возвращаются одни и те же значения для одних и тех же данных таблицы.)

Индексированный вычисляемый столбец может быть создан только в том случае, если следующим параметрам инструкции **SET** присвоено значение **ON** (эти параметры обеспечивают детерминированность столбца):

```
QUOTED_IDENTIFIER;
CONCAT_NULL_YIELDS_NULL;
ANSI_NULLS;
ANSI_PADDING;
ANSI_WARNINGS.
```

Кроме этого, параметру **NUMERIC_ROUNDABORT** нужно присвоить значение **OFF**.

Если для вычисляемого столбца создать кластеризованный индекс, то значения столбца будут существовать физически в соответствующих строках таблицы, поскольку страницы листьев кластеризованного индекса содержат строки данных.

В примере 9 показано создание кластеризованного индекса для вычисляемого столбца **total** из примера 8.

```
CREATE CLUSTERED INDEX i1 ON orders (total);
```

После выполнения инструкции **CREATE INDEX** вычисляемый столбец **total** будет присутствовать в таблице физически. Это означает, что все обновления базовых столбцов вычисляемого столбца будут вызывать его обновление.

Столбец можно сделать постоянным и другим способом, используя параметр PERSISTED. Этот параметр позволяет задать физическое наличие вычисляемого столбца, даже не создавая соответствующего кластеризованного индекса. Эта возможность требуется для создания физических вычисляемых столбцов, которые создаются на столбцах с приблизительным типом данных (FLOAT или REAL). (Как упоминалось ранее, индекс для вычисляемого столбца можно создать только в том случае, если его базовые столбцы имеют точный тип данных.)

Выводы

Индексы применяются для того, чтобы обеспечить более эффективный доступ к данным. Они могут оказывать влияние не только на производительность инструкции **SELECT**, но также на производительность инструкций **INSERT**, **UPDATE** и **DELETE**. Существуют кластеризованные и некластеризованные, однозначные и неоднозначные, а также простые и составные индексы. Кластеризованный индекс физически сортирует строки таблицы в порядке указанного столбца (или столбцов). Однозначный индекс указывает, что каждое значение может встречаться в таблице только один раз. Составной индекс создается по нескольким столбцам.

Прекрасным средством, относящимся к индексам, является Database Engine Tuning Advisor (DTA), который, среди прочего, анализирует образчик действительной загруженности (предоставляемой либо пользователем с помощью файла сценария, либо приложением SQL Server Profiler посредством записанного файла трассировки) и рекомендует, какие индексы следует добавить или удалить на основе данной загруженности.