

*Программирование
и администрирование
СУБД*

Microsoft®
SQL Server

Урок №4

Индексы

Содержание

1. Индексы в SQL Server	3
2. Индексированные представления.	18
3. Полнотекстовое индексирование и поиск.	21

1. Индексы в SQL Server

Для повышения скорости выполнения запросов и доступа к данным базы данных рекомендуется использовать индексы. Без индексов СУБД читает данные, просматривая каждую страницу таблиц, указанных в операторе SQL, проводит так называемое **сканирование таблицы**.

Понятие индекс встречается не только в теории баз данных и скорее всего вы с ним уже сталкивались в повседневной жизни. Индексы баз данных можно сравнить с библиотечным каталогом, в котором все книги записаны в карточки и упорядоченные определенным образом: по алфавиту или по темам, а в каждой карточке написано, где именно в архивах размещается данная книга. Другим примером может служить предметный указатель (индекс) в конце книги, который помогает читателю легко найти нужное место. Стоит напомнить, что в него входит только небольшое количество терминов, отсортированных по алфавиту, что позволяет быстро найти необходимую информацию. Похожим образом организованы и индексы в MS SQL Server.

Сам **индекс** представляет собой упорядоченный логический указатель на записи в таблице. **Указатель** означает, что индекс содержит значение одного или нескольких полей в таблице и адреса страниц данных, на которых размещаются эти значения. Иными словами, индекс состоит из пар "значение поля – физическое размещение этого поля". Таким образом, по значению поля (или полей),

входящих в индекс, можно быстро найти то место в таблице, где расположена запись с искомым значением. **Упорядоченный** означает, что значение полей, хранящихся в индексе, упорядочены.

Индексы в SQL Server организованы в так называемое **сбалансированное дерево (balanced tree)** или **В-дерево**. Его ориентировочный вид следующий:

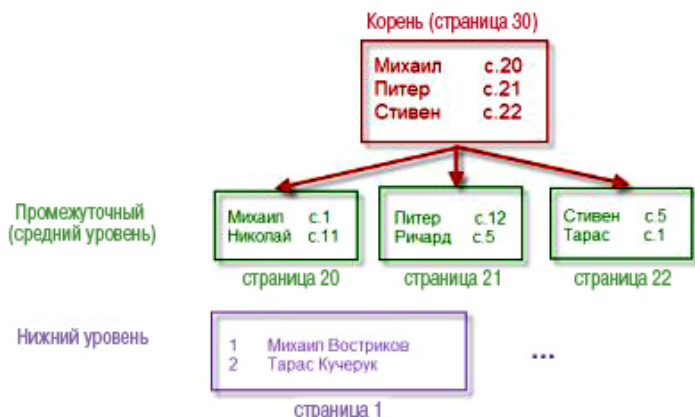


Каждый уровень индекса состоит из нескольких страниц данных, за исключением корня, который содержит только одну страницу. На этих страницах размещаются данные, на которые указывает индекс. Все страницы имеют одинаковую структуру и могут содержать 8192 байта данных, включая заголовок размером 96 байт. Исходя из этого, число промежуточных уровней индекса, а также количество страниц в каждом из них определяется путем несложных расчетов.

Поиск данных начинается с корневой страницы, указатель на которую находится в системной таблице **sysindexes** и системном представлении **sys.indexes**. Каждый уровень индекса представляет собой список с двойной связью (на уровень ниже и выше), а каждое значение корневого и промежуточного уровня – это первый элемент страницы

уровня ниже по порядку. Нижний уровень индекса содержит, как правило, записи таблицы базы данных.

Приведем пример индекса для поля с именем автора.



Обратите внимание, что данные на всех уровнях индекса, включая нижний, упорядочены по ключу индекса (по имени автора). При этом упорядочение не отражает физическую последовательность размещения данных.

Индексы в SQL Server, аналогично тому, как и в других СУБД, следует использовать с умом, ведь каждый создаваемый индекс имеет объем, равный объему данных в индексной поле плюс объем дополнительной информации о размещении записей. Итак, если создать индексы на каждое поле в таблице, то их суммарный объем будет больше, чем объем данных в таблице. Кроме того, существует так называемое "**правило 20%**": если запрос на выборку возвращает более 20% записей из таблицы, то использование индекса может замедлить выборку данных. Конечно, ситуация зависит от конкретной записи и условий, наложенных на выборку, но нужно помнить, что 20%

являются пределом, когда эффективность использования индексов ставится под вопрос.

Учитывая вышесказанное, определим **основные случаи**, когда поле необходимо проиндексировать:

- если это поле очень часто используется в условиях поиска в запросах;
- когда это поле используется при связывании таблиц (JOIN), поскольку индексы также обеспечивают уникальность записей таблицы, гарантируя тем самым целостность данных;
- когда это поле используется при сортировке (ORDER BY).

В MS SQL Server различают следующие **типы индексов**:

1. В зависимости от физического размещения индекса:
 - **кластеризованные** – это индексы, которые сортируют записи в таблицах или представлениях на основе их ключевых значений (но не физически на диске (!)). Этими значениями являются поля, включенные в индекс. Промежуточный уровень такого индекса по сути отсутствует, а нижний уровень содержит данные таблицы.

Значения кластеризованного индекса всегда уникальны и поэтому в таблице может быть только один такой индекс. Кроме того, при добавлении кластеризованного индекса, доступ к данным почти всегда осуществляется быстрее, чем в случае некластеризованного индекса, поскольку в первом случае исключается дополнительный поиск записи данных.

На практике для каждой таблицы необходимо создать кластеризованный индекс, который будет первичным ключом (но это не обязательно должно быть именно поле

первичного ключа). После этого, такая таблица будет называться **кластеризованной**. Таблицы, которые не содержат кластеризованных индексов, называют **кучами**.

- **некластеризованные** – это индексы, которые задают логическое упорядочение для таблицы. Структура такого индекса имеет структуру классического сбалансированного дерева. Нижний уровень такого индекса содержит значение ключа, вместе с идентификатором записи. Для каждой таблицы можно создать до 999 некластеризованных индексов.
По умолчанию все индексы некластеризованные.

2. В зависимости от типа значения, которое в них хранится:

- **уникальные (UNIQUE KEY)** – индексы, которые образуются при указании характеристики UNIQUE при создании ключа. Уникальность индекса означает, что его значения не повторяются;
- **неуникальные** – создаются по умолчанию и указывают на то, что значение их ключей могут повторяться.

3. В зависимости от количества полей, которые в них входят:

- **простые** – индексируется только одно поле;
- **составные** – это индексы, указывающие на несколько полей. Такой индекс может содержать от 2 до 16 полей. Максимальный общий размер значений составляющих индекса – 900 байт. Например, если таблица содержит поля Фамилия и Имя определенного клиента, для повышения производительности запросов, которые постоянно доступны до их полного имени, следует создать индекс на эти поля.

4. В зависимости от упорядоченности данных в индексе:
 - **растущие;**
 - **спадающие.**
5. В зависимости от степени охвата данных:
 - **полнотабличные** – это индексы, которые охватывают все записи индексируемой таблицы;
 - **фильтрованные** – это некластеризованные индексы, которые охватывают только часть записей в индексируемых таблицах. Они появились в версии SQL Server 2008 и имеют преимущества в случае, если типичные запросы к таблице в результате возвращают небольшое количество записей. Например, когда в запросах участвуют поля, которые в основном содержат NULL-значения.

Такой тип индекса рекомендуется использовать в случае наличия:

- разреженных полей, которые содержат небольшое количество отличных от NULL значений;
- разреженных полей, которые по-разному заполнены для различных категорий записей;
- полей, содержащих диапазоны значений (деньги, время, дату и т.д.).

Кроме того, существуют еще такие **виды индексов**, как:

- **полнотекстовые индексы** – индексы, которые используются в полнотекстовом поиске (см. раздел "Полнотекстовое индексирование и поиск").
- **пространственные индексы** – это индексы, которые позволяют более эффективно использовать операции с пространственными данными в полях типа *geometry*, *geography*. Для их создания используется оператор

CREATE SPATIAL INDEX. Более подробно об этом типе индексов см. в разделе "Обзор пространственного индексирования" электронной документации по SQL Server 2008.

- **XML-индексы** – это индексы для полей типа данных xml. Для их создания используется оператор CREATE XML INDEX. Более подробно об этом типе индексов см. в разделе "Индексы для полей типа данных xml" электронной документации по SQL Server 2008.

Не следует забывать также о том, что индекс не является частью таблицы – это отдельный объект, который связан с таблицей и другими объектами БД. Синтаксис создания индекса имеет следующий вид:

```
CREATE [ UNIQUE ]                               /* уникальность */
    [ CLUSTERED | NONCLUSTERED ]               /* кластеризованный
или нет */
INDEX имя_индекса
ON [БД.][схема.][таблица | представление] (поле ASC |
DESC [,... n])
[ INCLUDE ( поле [ ,...n ] ) ]                 /*дополнительные поля
на нижний уровень */
[ WHERE предикатное_выражение ]               /*для фильтрованного
индекса */
/* дополнительные параметры */
[ WITH ( [ PAD_INDEX = { ON | OFF } ]
    [, FILLFACTOR = x ]
    [, SORT_IN_TEMPDB = { ON | OFF } ]
    [, IGNORE_DUP_KEY = { ON | OFF } ]
    [, STATISTICS_NORECOMPUTE = { ON | OFF } ]
    [, DROP_EXISTING = { ON | OFF } ]
    [, ONLINE = { ON | OFF } ]
    [, ALLOW_ROW_LOCKS = { ON | OFF } ]
    [, ALLOW_PAGE_LOCKS = { ON | OFF } ]
    [, MAXDOP = максимальная_ступень_параллелизма ]
    [, DATA_COMPRESSION = { NONE | ROW | PAGE }
```

```

[ ON PARTITIONS ( { выражение_номера_секции | диапазон }
[ , ...n ] ) ]

]

) ]
/* где создать индекс */
[ ON { имя_схемы_секционирования ( название_поля )
| файловая_группа
| default /* в файловой группе по умолчанию */
} ]
[ FILESTREAM_ON { имя_файловой_группы_filestream | схема_
секционирования | "NULL" } ]

```

Параметр **INCLUDE** указывает на неключевые поля, которые добавляются на нижний уровень некластеризованного индекса. Поля типов text, ntext и image не допускаются.

Предикатное выражение в параметре **WHERE** используется для фильтрованных индексов и определяет те записи таблицы, которые должны быть проиндексированы. Стоит отметить, что фильтруемыми индексами не могут быть полнотекстовые и XML-индексы. Для уникальных индексов (UNIQUE) лишь избранные записи должны иметь уникальное значение. Кроме того, фильтровые индексы не поддерживают параметр IGNORE_DUP_KEY.

Параметр **ON** позволяет указать место сохранения индекса. Если его не указать, а таблица или представление несекционированные, SQL Server размещает индекс в той же файловой группе, где находится таблица или представление, для которых он создан.

Параметр **FILESTREAM_ON** позволяет перемещать данные FILESTREAM в другую файловую группу FILESTREAM или схему секционирования.

Расшифруем дополнительные **параметры создания индекса**:

- **PAD_INDEX** – используется, как правило, с фактором заполнения и определяет степень заполнения страниц на разных уровнях индекса (разреженность индекса). Параметр ON указывает на то, что степень заполнения страниц промежуточного уровня приравнивается к процентному отношению свободного пространства на диске, указанном в параметре FILLFACTOR. По умолчанию (OFF) страницы промежуточного уровня заполняются почти полностью – свободного места хватает не более, чем на одну запись от максимального значения размера индекса.
- **FILLFACTOR** – это коэффициент (фактор) заполнения, который определяет полноту заполнения каждой страницы нижнего уровня индекса. По умолчанию он равен 0. Значение фактора заполнения 0 или 100 говорит о том, что страницы нижнего уровня при создании заполнены полностью (на 100%).
- **SORT_IN_TEMPDB** – сохранять ли промежуточные результаты сортировки, которые образуются при создании индекса, в базе данных tempdb. По умолчанию промежуточные результаты сортировки хранятся в той же базе, что и индекс (OFF).
- **IGNORE_DUP_KEY** – включает игнорирование значения ключа, которое повторяется, для уникальных индексов. По умолчанию при повторе значений генерируется сообщение об ошибке (OFF).
- **STATISTICS_NORECOMPUTE** – указывает на необходимость сохранения статистических данных о составе ин-

декса. Значение по умолчанию OFF включает автоматическое обновление статистики.

- **DROP_EXISTING** – указывает на то, что индекс должен уничтожаться и создаваться заново, то есть на пересоздаваемость индекса. По умолчанию (OFF) при существовании одноименных индексов генерируется сообщение об ошибке, иначе (ON) существующий индекс удаляется и перестраивается.
- **ONLINE** – запрещает запросы и изменения данных в индексированных таблицах и их индексах во время операций над индексами по сети (OFF; по умолчанию). Данная функция доступна только для выпусков SQL Server Enterprise, Developer и Evaluation.
- **ALLOW_ROW_LOCKS** – разрешена или нет блокировка записей. По умолчанию блокировка записей разрешена при доступе к индексам (ON).
- **ALLOW_PAGE_LOCKS** – разрешена или нет блокировка страниц. По умолчанию блокировка страниц возможна при доступе к индексу (ON).
- **MAXDOP** – устанавливает параметр конфигурации "максимальная степень параллелизма", который задает время операции индекса. MAXDOP ограничивает число процессоров, которые принимают участие в параллельном выполнении плана. Максимальное количество процессоров – 64. Данная функция доступна только для выпуска SQL Server Enterprise.
- **DATA_COMPRESSION** – задает режим компрессии данных для указанного индекса или указанных секций. Он может принимать одно из следующих значений:

- NONE – компрессия данных;
- ROW – компрессия записей;
- PAGE – компрессия страниц.
- **ON PARTITIONS** – указывает секции, в которых применяется параметр DATA_COMPRESSION. Если опция ON PARTITIONS не указана, то параметр DATA_COMPRESSION применяется ко всем секциям секционированного индекса.

В выражении номера секции можно указывать:

- номер секции, например, ON PARTITIONS (2)
- перечень секций, разделенный запятыми, например, ON PARTITIONS (1, 3, 7);
- диапазон секций вместе с отдельными секциями, например, ON PARTITIONS (2, 4, 6 TO 8).

При определении диапазона, номера секций разделяются ключевым словом TO, например, ON PARTITIONS (2 TO 5).

Для примера создадим несколько индексов:

```
-- simple clustered index on a book title stored in the
Books table
create index i_book on Books (NameBook);

-- composite clustered index that contains the name of
book's author
-- of thr Authors table
create index i_author_name on Authors (FirstName, LastName);

-- simple clustered index for the DateOfSale field of the
Sales table
create clustered index ci_salesdate on sales (DateOfSale);
```

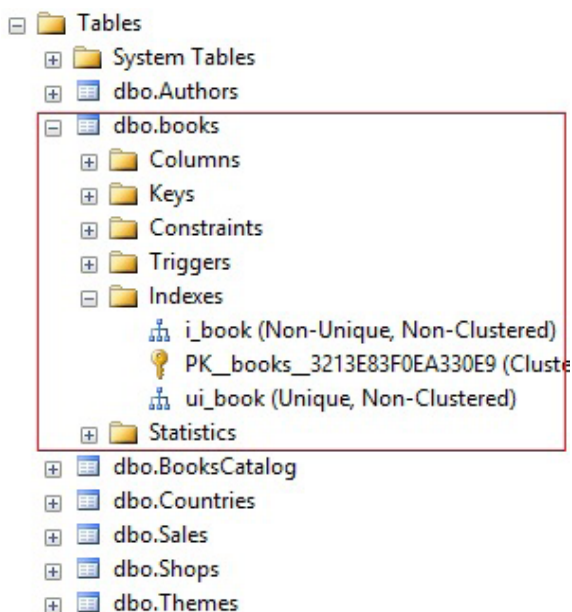
```

-- simple nonclustered index for the id field of the
Countries table, that can be rebuilt.
-- we also define the filling factor of the leaf level of the
index
create nonclustered index i_country on Countries(id)
with (fillfactor = 80, pad_index = on, drop_existing = on);

-- simple clustered with a filler by country name (USA)
create index i_country_USA on Countries(Country)
where country = 'USA';

```

Если индекс создан, то он будет размещен в папке **Indexes** необходимой таблицы БД:



Чтобы определить **используется ли индекс при выполнении SQL запроса** используются следующие операторы T-SQL:

- **SET SHOWPLAN_ALL ON / OFF** – показать весь план выполнения запроса;

- **SET SHOWPLAN_TEXT ON / OFF** – показать текст плана выполнения запроса.

Например:

```
use Press
go
set showplan_all on
go
select NameBook, DateOfPublish
from book.Books
```

Результатом будет вывод информации о том, какие данные и откуда берутся для формирования результирующего набора, которые индексы при этом используются и откуда они берутся и тому подобное.

Индекс можно временно отключить или снова включить с помощью инструкции **ALTER INDEX**:

```
-- включить индекс
ALTER INDEX { имя_индекса | ALL } ON объект DISABLE
-- включить индекс; при этом он перестраивается
ALTER INDEX { имя_индекса | ALL } ON объект REBUILD
```

Стоит отметить, что при отключении кластеризованного индекса, вся таблица становится недоступна.

Например:

```
alter index ci_salesdate on sale.Sales disable
go
alter index ci_salesdate on sale.Sales rebuild
go
```

Полный синтаксис оператора **ALTER INDEX** позволяет изменить и его первичные настройки:

```

ALTER INDEX { имя_индекса | ALL }
ON [БД.][схема.][таблица | представление]
{ REBUILD /* перестроить индекс (по
желанию со следующими условиями) */
  [ [PARTITION = ALL] /* перестраиваются все секции
индекса */
    [ WITH ( добавочные_опции [ ,...n ] ) ] /* согласно
таких акций */
    | [ PARTITION = номер_секции
/* перестраивается только одна секция со следующими
параметрами */
      [ WITH ( параметры_перестройки_единичной_секции [
,...n ] ) ]
    ]
  | DISABLE /* отключить индекс */
  | REORGANIZE /* реорганизовать конечный уровень
индекса */
    [ PARTITION = номер_секции ] /* для конкретной
секции */
    [ WITH ( LOB_COMPACTION = { ON | OFF } ) ]
/* все страницы с данными типа LOB (image, text, ntext,
varchar(max), xml nvarchar(max), varbinary(max))
сжимаются*/
    | SET ( параметры_индекса [ ,...n ] ) /*
параметры без перестройки или реорганизации индекса */
}
-- параметры перестройки единичной секции
{
  SORT_IN_TEMPDB = { ON | OFF }
  | MAXDOP = максимальная_ступень_параллелизма
  | DATA_COMPRESSION = { NONE | ROW | PAGE } }
}
-- параметры индекса без перестройки или реорганизации
индекса
{
  ALLOW_ROW_LOCKS = { ON | OFF }
  | ALLOW_PAGE_LOCKS = { ON | OFF }
  | IGNORE_DUP_KEY = { ON | OFF }
  | STATISTICS_NORECOMPUTE = { ON | OFF }
}

```


Удаление реляционного индекса осуществляется инструкцией **DROP INDEX**.

```
DROP INDEX имя_индекса
ON [БД.][схема.][таблица | представление]
[ WITH ( добавочные_опции_индекса [ ,...n ] ) ]
[ ,...n ]
-- дополнительные опции индекса
{
    MAXDOP = максимальная_ступень_параллелизма
    | ONLINE = { ON | OFF }
    /* куда будут перемещены после удаления строки данных
    конечного уровня индекса */
    | MOVE TO { имя_схемы_секционирования ( название_поля
    )
        | файловая_группа
        | default
    }
    /* в какую папку будет перемещена таблица filestream
    конечного уровня индекса */
    [ FILESTREAM_ON { имя_файловой_группы_filestream |
    схема_секционирования | "NULL" } ]
}
```

Например:

```
drop index ci_salesdate on sale.Sales,
iBook on book.Books
```

Для получения статистических данных по индексам используются следующие инструкции T-SQL и системные хранимые процедуры:

1. Создание статистики – **CREATE STATISTICS**;
2. Обновление статистики – **sp_updatestats**;
3. Автостатистика – **AUTO_UPDATE_STATISTICS**;
4. Просмотр статистики – **sp_helpstats**.

2. Индексированные представления

Индексированные представления (indexed view) позволяют заранее рассчитать результирующий набор, который будет возвращаться представлением. Например, провести все необходимые объединения таблиц, рассчитать функции агрегирования и тому подобное. При этом отпадает необходимость вводить условие в каждый запрос. Таким образом, производительность приложений, использующих индексированные представления вместо базовых таблиц, возрастает в десятки раз. Это очень существенно, когда работа стандартного представления связана со сложной обработкой большого количества запросов, например, при ведении и расчета статистики.

Теоретически, создание таких представлений сводится к созданию представления и построения для него кластеризованного индекса. Но это не совсем так, поскольку таблицы, которые будут принимать участие в индексированных представлениях, индексы для них, а также сами представления должны соответствовать ряду критериев. Эти критерии введены для того, чтобы избежать неоднозначных расчетов. В связи с этим в индексированных представлениях разрешается использование только детерминированные функции. Более подробно об ограничении читайте в разделе "Создание индексированных представлений" электронной документации по SQL Server 2008.

Кстати, в редакцию SQL Server Enterprise и Developer встроенная интересная возможность оптимизатора запросов. Если оптимизатор определит, что для выполнения запроса эффективнее использовать индексированные представления, а не базовую таблицу, он перепишет запрос. При этом в запросе даже не нужно указывать индексированное представление – достаточно указать таблицу, для которой оно определено.

Проведем небольшой пример построения индексированного представления.

```
-- defining parameters for supporting of indexed views (if
they are not defined)
set NUMERIC_ROUNDABORT off;
set ANSI_PADDIND, ANSI_WARNINGS, CONCAT_NULL_YIELDS_NULL,
ARITHABORT,
QUOTED_IDENTIFIER, ANSI_NULLS on;
go

-- creating a view
create view iv_Orders
with schemabinding
as
select sh.Shop as 'Shop',
s.DateOfSale as 'Sale Date',
sum(s.price*s.quantity) as 'Cost',
COUNT_BIG(*) AS COUNT
from dbo.Sales s, dbo.Shops sh
where s.id_shop = sh.id
group by sh.Shop, s.DateOfSale
go

-- creating index for the view
create unique clustered index uci_ivorders on iv_
Orders(Shop)
go
```

После этого, в последующих запросах будет использоваться созданное нами индексируемое представление, хотя на него и нет явной ссылки.

```
select sh.Shop as 'Shop',
s.DateOfSale as 'Sale Date',
sum(s.price*s.quantity) as 'Cost'
from dbo.Sales s, dbo.Shops sh
where s.id_shop = sh.id and
      s.DateOfSale between '01/01/2006' and '01/01/2010'
group by sh.Shop, s.DateOfSale
go
select sh.Shop as 'Shop',
sum(s.price*s.quantity) as 'Cost'
from dbo.Sales s, dbo.Shops sh
where s.id_shop = sh.id and
      datepart(mm, s.DateOfSale) = 1
group by sh.Shop
```

Ряд СУБД поддерживают аналоги индексируемых представлений SQL Server. Например, в СУБД Oracle это материализованные представления.

Подытоживая, можно выделить разницу между обычным представлением и индексируемым. **Стандартное представление** – это SELECT запрос, на который ссылаются по имени и который сохраняется в SQL Server. Само по себе оно не содержит данных.

Индексируемое представление – это представления с кластеризованным индексом. Так SQL Server материализует и сохраняет на диске результаты запроса, заданного в представлении.

3. Полнотекстовое индексирование и поиск

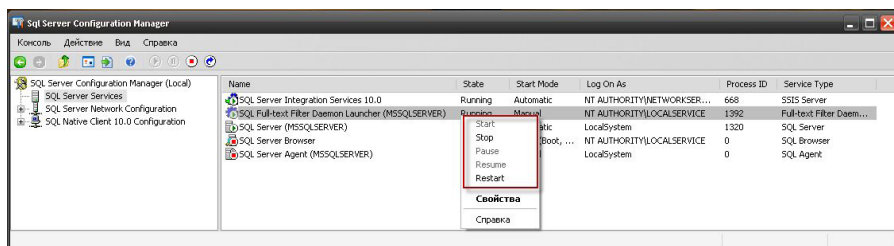
Для повышения эффективности выборки при поиске текстовых данных вместо операторов LIKE или оператора равенства (=), а также поиска в неструктурированных данных, SQL Server и ряд других СУБД рекомендуют использовать полнотекстовый поиск. В SQL Server полнотекстовый поиск обеспечивает отдельный компонент **Full-Text Search**.

Для того, чтобы организовать полнотекстовый поиск необходимо:

1. Создать полнотекстовый каталог.
2. Определить таблицы и поля, для которых необходимо использовать полнотекстовый поиск.
3. Создать полнотекстовые индексы на указанные поля.
4. Задать метод заполнения полнотекстового индекса, который будет обеспечивать согласованность полнотекстового индекса с данными. То есть метод заполнения отвечает за то, чтобы все изменения в отобранных полях таблиц были отражены в соответствующих полнотекстовых индексах.
5. Создать запрос, который будет с помощью специальных функций осуществлять полнотекстовый поиск.

Полнотекстовое индексирование имеет свою собственную службу – **Средство полнотекстового поиска диспетчера**

конфигурации SQL Server (SQL Full-Text Filter Daemon Launcher, MSSQLFDLauncher) – предназначенную для управления полнотекстовыми каталогами.



Итак, процесс организации полнотекстового индексирования начинается с создания полнотекстового каталога, поскольку именно он обеспечивает поддержку полнотекстовых индексов в SQL Server. Фактически, **полнотекстовый каталог** – это набор полнотекстовых индексов базы данных. Начиная с версии SQL Server 2008, в базе данных полнотекстовый каталог является виртуальным объектом и не входит в файловую группу. В предыдущих версиях, его рекомендовалось размещать в отдельной файловой группе базы данных.

Для создания полнотекстового каталога используется инструкция T-SQL **CREATE FULLTEXT CATALOG**:

```
CREATE FULLTEXT CATALOG название_каталога
    [ ON FILEGROUP файловая_группа ] -- для
совместимости с предыдущими версиями
    [ IN PATH 'корневой_путь' ] -- для совместимости
с предыдущими версиями, в следующей версии планируется ее
исключение
    [ WITH ACCENT_SENSITIVITY = { ON | OFF } ]
    [ AS DEFAULT ] -- данный каталог
будет каталогом по умолчанию
    [ AUTHORIZATION имя_собственника ]
```

Параметр **WITH** указывает на то, будет ли каталог учитывать диакритические знаки для полнотекстового индексирования. Важно знать, что при изменении данного параметра, индекс автоматически перестраивается, в отличие от предыдущих версий SQL Server, где необходимо было вручную перестроить все полнотекстовые индексы каталога.

После создания полнотекстового каталога необходимо создать полнотекстовые индексы. Одна таблица или индексированное представление может иметь только один полнотекстовый индекс, хотя в ней может быть проиндексировано несколько полей. Для реализации полнотекстового индексирования таблица должна иметь одно уникальное поле с не NULL значением, а также иметь поля типа char, varchar, varchar (max), nchar, nvarchar, text, ntext, image, xml, varbinary или varbinary (max).

Чтобы создать полнотекстовые индексы для двоичных типов данных (image, varbinary, varbinary (max)) используются **специальные обработчики протокола и фильтры**. Они позволяют получать текст из файлов Word, Excel, PowerPoint, PDF и т.д., хранящиеся в базе данных. Для работы этих сервисов необходимо в таблицу добавить поле, которое содержит информацию о типе документа, который хранится в поле двоичного типа. После этого фильтр загружает двоичный поток, который хранится в поле, удаляет всю информацию о форматировании и возвращает текст документа в средство разбиения на слова. После этого механизм полнотекстового поиска:

- 1) вычисляет лексемы, которые являются сжатыми формами самих слов;

- 2) строит все лексемы в поле в специальную сжатую структуру файла, который используется для дальнейшего поиска. В связи с этим, размеры самого полнотекстового индекса ограничены доступными ресурсами памяти компьютера.

Начиная с версии SQL Server 2008 полнотекстовые индексы встроены в компонент Database Engine, а не размещены в файловой системе, как в предыдущих версиях SQL Server.

Для создания полнотекстовых индексов используется инструкция **CREATE FULLTEXT INDEX**.

```
CREATE FULLTEXT INDEX ON таблица
    [ ( название_поля [ TYPE COLUMN поле_типа ] [
LANGUAGE язык_данных ] [ ,...n ] ) ]
    KEY INDEX название_уникального_индекса
    [ ON полнотекстовый_каталог ]
    [ WITH [ ( ) опции [ ,...n ] [ ) ] ]
-- опции
{ CHANGE_TRACKING [ = ] { MANUAL | AUTO | OFF [, NO
POPULATION ] }
  | STOPLIST [ = ] { OFF | SYSTEM | список_стоп-слов }
}
```

Параметр **TYPE COLUMN** содержит название поля таблицы, в котором хранится тип документа (расширение файла, которое указывается пользователем (.DOC, .PDF, .XLS т.д.)) для документов varbinary, varbinary (max) или image. Если индекс создается не для двоичных данных, тогда параметр TYPE COLUMN не указывается. Это поле называется полем типа и должно иметь тип char, nchar, varchar или nvarchar.

Параметр **LANGUAGE** указывает на язык данных, который хранится в индексируемом поле. Данный параметр

рекомендуется использовать, если в индексируемых таблицах содержатся поля на разных языках. Например, если данные поля переведены на несколько языков. Получить список доступных кодов для языков сохраняется в системном представлении **sys.fulltext_languages**.

Опция **CHANGE_TRACKING** указывает на то, как будут отслеживаться изменения индексируемых данных (при update, insert, delete):

- **MANUAL** – синхронизацию данных следует осуществлять вручную, путем вызова инструкций **ALTER FULLTEXT INDEX ... START UPDATE POPULATION** (заполнение вручную) или с помощью установки расписания синхронизации изменений в SQL Server Agent.
- **AUTO** (по умолчанию) – изменения распространяются автоматически (с помощью фонового процесса) в ходе модификации данных в базовой таблице.
- **OFF [, NO POPULATION]** – изменения данных не отслеживаются. Аргумент **NO POPULATION** указывает на то, что полнотекстовый индекс будет создан без заполнения. Дальнейшее заполнение индекса осуществляется с помощью оператора **ALTER FULLTEXT INDEX**.

Стоит отметить, что **процесс заполнения полнотекстового индекса очень ресурсоемкий**, поэтому создание такого индекса должно быть осуществлено, когда активность базы данных невелика. В связи с этим, в большинстве случаев полнотекстовые индексы создают с опцией **OFF** и **NO POPULATION**, а когда активность базы данных минимальная создают задачи для заполнения всех полнотекстовых

индексов. После этого, для индексируемых полей, которые редко меняются, заполнение индексов выставляется в режим AUTO.

Параметр **STOPLIST** связывает полнотекстовый список стоп-слов с индексом (до версии SQL Server 2008 – пропускаемых слов). Индекс не заполняется значениями, которые являются частью данного списка. Если данный параметр не указан, тогда с индексом связывается системный полнотекстовый список стоп-слов.

- OFF – указывает на то, что с полнотекстовым индексом не связано ни одно значение из списка стоп-слов. То есть список не используется.
- SYSTEM – для полнотекстового индекса будет использоваться системный список стоп-слов.
- Также можно указать конкретное имя списка стоп-слов, который будет связываться с полнотекстовым индексом.

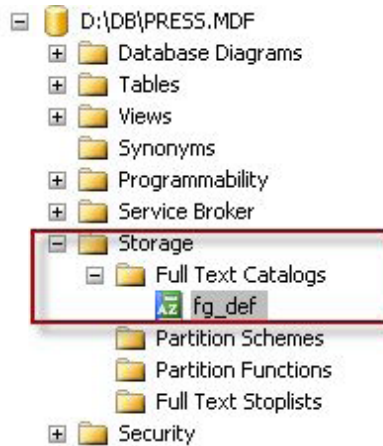
Посмотреть список стоп-слов можно из системного представления **sys.fulltext_stopwords**. Управлять списком стоп-слов можно с помощью операторов **CREATE/ALTER/DROP FULLTEXT STOPLIST**, но только при уровне совместимости 100.

Для примера возьмем таблицу, которая содержит информацию об авторах. Чтобы организовать полнотекстовый поиск по полям FirstName и LastName необходимо создать для них полнотекстовые индексы.

Итак, сначала создаем полнотекстовый каталог, в котором будут определяться поля для индексирования:

```
-- создаем полнотекстовый каталог по умолчанию  
create fulltext catalog fg_def as default;
```

Новосозданный полнотекстовый каталог в базе данных размещается в папке **Storage-> Full Text Catalogs**.



Следующим шагом является создание полнотекстового индекса на необходимые поля. Но, если в таблице отсутствует уникальный индекс (обычно им является первичный ключ), тогда следует его создать.

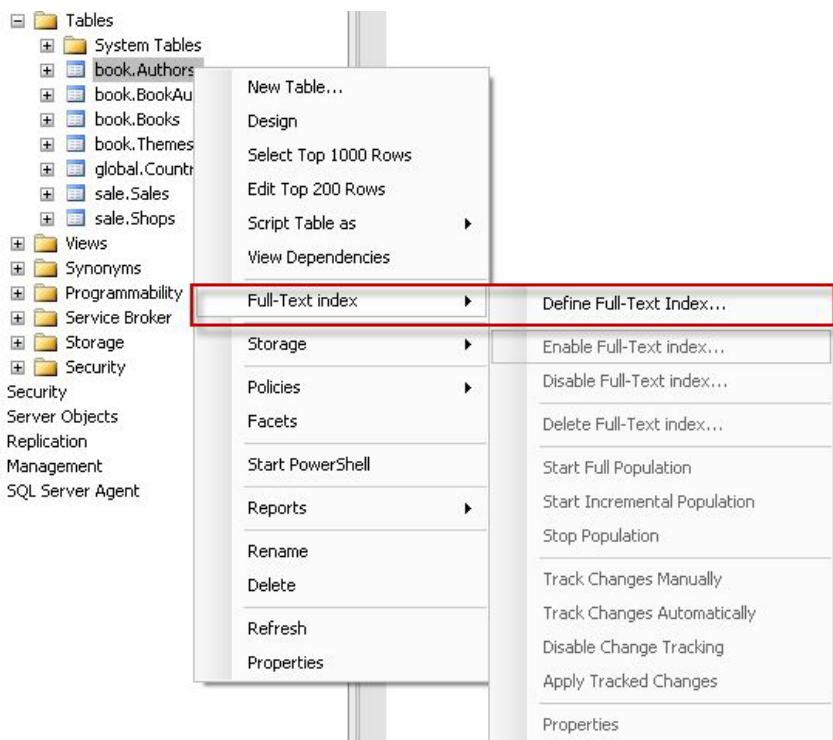
```
-- если в таблице нет уникального индекса, тогда создадим его
create unique index ui_AuthorId on book.Authors (ID_AUTHOR);
go

-- создаем полнотекстовый индекс для полей FirstName
и LastName с использованием полнотекстового каталога fg_def
и системного списка стоп-слов
create fulltext index on book.Authors (FirstName,
LastName)
    key index ui_AuthorId
    on fg_def;      -- параметр необязательный
```

Можно также указать код языка, который будет являться языком данных в полях. Код (LCID) английского языка – 1033, русского – 1049, украинского – 1058:

```
create fulltext index on book.Authors
(FirstName language 1049,
 LastName language 1049)
key index ui_AuthorId
```

Чтобы создать полнотекстовый индекс средствами Management Studio, следует выбрать пункт контекстного меню **Full-Text index-> Define Full-Text Index ...** таблицы, в которой предполагается его наличие. После этого запустится визард создания полнотекстового индекса, в котором необходимо указать уникальный индекс таблицы, полнотекстовый каталог, в котором будет размещаться индекс, необходимые поля и тому подобное.



Для изменения установленных параметров полнотекстового индекса следует воспользоваться инструкцией **ALTER FULLTEXT INDEX:**

```
ALTER FULLTEXT INDEX ON таблица
{
    -- активизировать или отключить полнотекстовый
    индекс
    ENABLE | DISABLE
    -- отражать изменения данных поля в полнотекстовом
    индексе, который ему соответствует
    | SET CHANGE_TRACKING { MANUAL | AUTO | OFF }
    -- добавить поле для индекса
    | ADD ( название_поля [ TYPE COLUMN поле_типа ] [
    LANGUAGE язык_данных ] [ ,...n ] )
    [ WITH NO POPULATION ]
    -- удалить поле из индекса | DROP (название_поля
    [,...n] ) [ WITH NO POPULATION ]
    -- начать заполнение полнотекстового индекса
    | START { FULL | INCREMENTAL | UPDATE } POPULATION
    -- остановить, приостановить или возобновить процесс
    заполнения индекса
    | {STOP | PAUSE | RESUME } POPULATION
    -- как завязывать полнотекстовый список стоп-слов
    с индексом
    | SET STOPLIST { OFF | SYSTEM | список_стоп-слов } [
    WITH NO POPULATION ]
}
```

Опция **WITH NO POPULATION** указывает на то, что полнотекстовый индекс не будет заполнен после добавления (ADD) или удаление (DROP) поля полнотекстового индекса, а также после операции SET STOPLIST. Индекс будет заполняться только после выполнения команды START ... POPULATION.

Параметр **START ... POPULATION** указывает SQL Server, что следует начать заполнение полнотекстового индекса одним из следующих способов:

- **FULL** – каждая строка таблицы должна принимать участие в полнотекстовом индексировании, даже если ее записи уже были проиндексированы.
- **INCREMENTAL** – указывает, что для полнотекстового индексирования используются только те записи, которые были изменены после последнего индексирования. Данный аргумент можно использовать только, если в таблице существует поле типа `timestamp`. Если такого поля нет, тогда выполняется заполнение **FULL**.
- **UPDATE** – обрабатываются все операции **INSERT**, **DELETE** и **UPDATE** после последнего обновления. При этом, таблица должна поддерживать отслеживание изменений (включена опция), но индекс фоновое обновления или автоматическое отслеживание изменений выключены.

Параметр **{STOP | PAUSE | RESUME} POPULATION** позволяет остановить, приостановить или возобновить приостановленный процесс заполнения индекса. Но в их работе существуют небольшие ограничения:

- **STOP** – несмотря на то, что процесс заполнения индекса останавливается, автоматическое отслеживание изменений или фоновое обновление индекса продолжается. Чтобы их остановить, следует использовать команду `SET CHANGE_TRACKING OFF`.
- **PAUSE** и **RESUME** могут использоваться только для полных (**FULL**) заполнений. Для других типов

заполнений они несущественны, поскольку сканирование возобновляется с момента последней проверки.

Например:

```
-- активизировать полнотекстовый индекс в таблице book.
Authors
alter fulltext index on book.Authors enable
go
-- заполнить полнотекстовый индекс в таблице book.Authors
alter fulltext index on book.Authors start update
population
```

Для удаления полнотекстового индекса используется инструкция **DROP FULLTEXT INDEX**:

```
-- синтаксис
DROP FULLTEXT INDEX ON таблица
-- например
drop fulltext index on book.Authors
```

Теперь можно создавать и выполнять полнотекстовые запросы. Для полнотекстового поиска используются свои функции, которые значительно эффективнее чем оператор LIKE и (=). В SQL Server это предикаты:

- **FREETEXT** – используется для поиска нечеткого совпадения. Она определяет список различных форм искомого слова, фактически осуществляя поиск по его сути.

Синтаксис:

```
FREETEXT ( { название_поля | ( список_полей ) | * },
           'текст для поиска'
           [ , LANGUAGE язык ]
        )
```

- **CONTAINS** – ищет точные совпадения слов и префиксов слов. Сокращенный синтаксис данного предиката аналогичный FREETEXT.

Примечание! Тип второго параметра вышеописанных предикатов **nvarchar**, поэтому для корректной работы запросов необходимо, чтобы и поля для поиска имели аналогичный тип. Кроме того, чтобы поиск фраз или слов на русском, украинском и других европейских языках работал корректно, не забывайте при создании полнотекстового индекса также указывать параметр **LANGUAGE**.

Например, необходимо найти всех авторов, в фамилии которых встречается Herbert:

```
select FirstName, LastName
from Authors
where freetext(FirstName,N'Herbert');
select FirstName, LastName
from Authors
where contains(FirstName,N'Herbert');
```

Результат:

	FirstName	LastName
1	Clifford	Simak
2	Dino	Esposito
3	Donald	Knuth
4	Herbert	Schildt
5	Herbert	Wells
6	Jeffrey	Richter
7	Matthew	MacDonald
8	Richard	Waymire

freetext

	FirstName	LastName
1	Herbert	Schildt
2	Herbert	Wells

	FirstName	LastName
1	Herbert	Schildt
2	Herbert	Wells

contain

Чтобы функция contains вернула результат, аналогичный freetext, следует воспользоваться **префиксами**. Для

идентификации префикса необходимо использовать двойные кавычки. При их отсутствии запрос будет осуществлять поиск слова.

```
select FirstName, LastName
from Authors
where contains(FirstName, N'H');
```

Теперь будут найдены все фамилии авторов, начинающихся с 'H'.

Обе функции позволяют использовать в шаблонах поиска фразы. Например, среди книг необходимо найти такие, которые содержат фразу "NET MVC":

```
-- this query will find all the books that contain "NET" and "MVC"
select b.NameBook, t.NameTheme
from Books b, Themes t
where b.id_theme = t.id and
      contains(b.NameBook, 'NET MVC')

-- with the use of 'and' (&), 'or' (|), and 'not' (!)
select b.NameBook, t.NameTheme
from Books b, Themes t
where b.id_theme = t.id and
      contains(b.NameBook, 'NET' or 'MVC')

-- this query will find all the books that contain "NET" and "MVC"
select b.NameBook, t.NameTheme
from Books b, Themes t
where b.id_theme = t.id and
      freetext(b.NameBook, 'NET MVC');
```

Оператор **AND NOT** возвращает true, если первое условие истинно, а вторая ошибочно.

Результаты:

Results Messages	
NameBook	NameTheme
1 Applied Microsoft .NET Framework Programming	Computer Science
2 Programming ASP.NET Core (Developer Reference)	Web Technologies
3 Programming Microsoft ASP.NET MVC (3rd Edition)	Web Technologies
4 ASP.Net: The Complete Reference	Computer Science
5 Pro WPF in C# 2010: Windows Presentation Foundation in .Net 4	Computer Science
6 Pro Wpf 4.5 in C#: Windows Presentation Foundation in .Net 4.5	Computer Science

freetext

Results Messages	
NameBook	NameTheme
1 Programming Microsoft ASP.NET MVC (3rd Edition)	Web Technologies

contains

Примечание! Поиск символов в слове или фразе регистронезависимый. Такие слова как "a", "and", "is", "the" (в зависимости от языка) и т.д., а также знаки препинания при поиске пропускаются, поскольку они считаются ненужными и включены в список стоп-слов.

Кстати, в предикате contains префиксы используются и для фраз. В таком случае, каждое слово из фразы считается отдельным префиксом. Например, при поиске фразы "local wine *" будут отобраны строки с текстом "local winery", "locally wined and dined" и другие.

Функция contains может принимать различные параметры, что позволяет настроить ее использование. Два варианта использования, кроме "классического" поиска слов, мы уже рассмотрели – это использование префиксов и фраз в шаблонах. Рассмотрим ее возможности шире:

1. Для поиска вариантов фраз используют функцию **FORMSOF**, первый параметр которой указывает на принцип генерации словоформ:

- **INFLECTIONAL** – поиск по основе слова, например, слово "погода" выдаст соответствие для "погода", "погоды", "согласовывать" и др;
- **THESAURUS** – поиск синонимов, определенных в XML-файле тезауруса. Например, слово "металл" может иметь синоним "золото", "алюминий", "железо" и другие. По умолчанию файл тезауруса пустой и размещается по пути место_установления _SqlServer \ Microsoft SQL Server \ MSSQL10_50.MSSQLSERVER \ MSSQL \ FTDATA \.

```
create fulltext index on Books (NameBook language 1049)
key index ui_book_id
go
select b.NameBook, t.NameTheme
from Books b, Themes t
where b.id_theme = t.id and
      contains(b.NameBook, 'formsof (inflectional, Java)')
```

Результат:

Results		Messages
	NameBook	NameTheme
1	Java: A Beginner Guide	Computer Science
2	Java: The Complete Reference	Computer Science

- Поиск слов или фраз по близости их месторасположения. Для этого используется ключевое слово NEAR (~). Расстояние между заданными словами измеряется также в словах, с учетом принадлежности к предложению или абзаца. Предикат CONTAINS редко используется с этим ключевым словом, поскольку

ранг результирующих слов не может быть определен напрямую.

```
select b.NameBook, t.NameTheme
from Books b, Themes t
where b.id_theme = t.id and
      contains(b.NameBook, 'Complete NEAR Reference')
-- ИЛИ
select b.NameBook, t.NameTheme
from Books b, Themes t
where b.id_theme = t.id and
      contains(b.NameBook, 'Complete ~ Reference')
```

Результатом этих запросов будут названия книг, которые содержат слово "Complete" рядом со словом "Reference".

Results Messages	
NameBook	NameTheme
1 Java: The Complete Reference	Computer Science
2 C++: The Complete Reference, 4th Edition	Computer Science
3 ASP.Net: The Complete Reference	Computer Science

Results Messages	
NameBook	NameTheme
1 Java: The Complete Reference	Computer Science
2 C++: The Complete Reference, 4th Edition	Computer Science
3 ASP.Net: The Complete Reference	Computer Science

3. Функция **ISABOUT** позволяет задавать относительный вес определенным критериям поиска (в диапазоне от 0.0 до 1.0).

```
select b.NameBook, t.NameTheme
from Books b, Themes t
where b.id_theme = t.id and
      contains(b.NameBook, 'isabout(Windows weight(.8), C#
weight (.3))')
```

В данном запросе нас больше интересует вхождение слова "Windows", поэтому ему мы установили больший весовой коэффициент.

Стоит отметить, что ключевое слово **WEIGHT** не влияет на запросы с **CONTAINS**, но влияет на значение **RANK**, которое возвращает **CONTAINSTABLE**.

Функции **CONTAINS** и **FREETEXT** имеют свои аналоги для работы с таблицами – это **CONTAINSTABLE** и **FREETEXTTABLE**. Отличие последних заключается лишь в том, что они возвращают набор записей, который должен быть соединен с другой таблицей на основе значения ключа. Вместе с результатами поиска они возвращают **дополнительные поля**:

- **RANK** – определяет ранг соответствия каждой записи шаблона: чем выше ранг, тем точнее совпадение (число от 0 до 1000);
- **KEY** – ключевое поле входной таблицы, соответствующее найденным записям.

Синтаксис их следующий:

```
CONTAINSTABLE (  таблица, { поле | (список_полей) | * },
                  'текст для поиска'
                  [ , LANGUAGE язык ]
                  [ , первые_n_по_рангу ]
                )

FREETEXTTABLE (  таблица, { поле | (список_полей) | * },
                  'текст для поиска'
                  [ , LANGUAGE язык ]
                  [ , первые_n_по_рангу ]
                )
```

Например, напишите запрос, который ищет сборники в таблице **book.Books**. Поиск осуществляется по всем полям.

```
select *
from containstable(book.Books, *, N'сборник')
```

Результат:

	KEY	RANK
1	27	80
2	35	80
3	36	80
4	37	80

Вышерассмотренный пример можно переписать следующим образом:

```
select cb.Rank as 'Rank', b.id as 'Code',
       b.NameBook as 'Book Title',
       a.FirstName + ' ' + a.LastName as 'Author'
from containstable(Books, *, N'Complete') as cb,
     Books as b, Authors as a
where cb.[KEY] = b.id and
       b.id_author = a.id
```

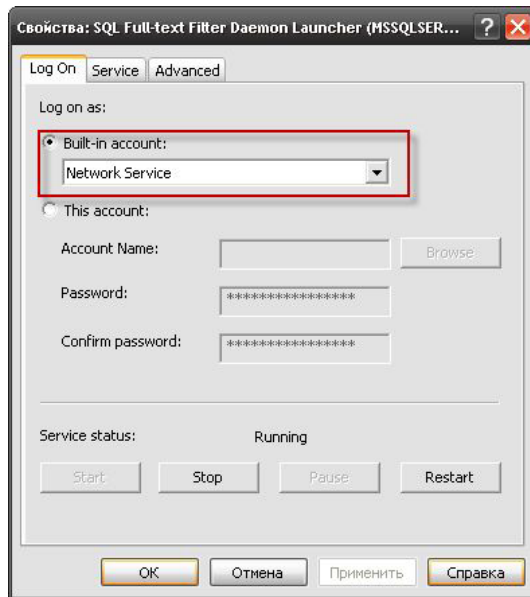
Как видно из примера, поле KEY должно быть указано в квадратных скобках, поскольку оно является ключевым словом T-SQL.

Результат:

	Rank	Code	Book Title	Author
1	80	12	Java: The Complete Reference	Herbert Schildt
2	80	13	C++: The Complete Reference, 4th Edition	Herbert Schildt
3	80	17	ASP.Net: The Complete Reference	Matthew MacDonald
4	80	24	Microsoft SQL Server 2005 Complete Handbook	Richard Waymire

Примечание! Иногда, SQL Server при выполнении запросов с полнотекстовым поиском может выдавать ошибку "SQL Server encountered error 0x80070218 while communicating with full-text filter daemon host (FDHost) proces". Для того, чтобы избежать данной ошибки необходимо:

1. Зайти в Диспетчер конфигурации SQL Server.
2. Выбрать свойства службы SQL Full-text Filter Deamon Launcher.
3. Для входа (вкладка Log On) выбрать Network Service.



4. Перезапустить службу и экземпляр сервера.



Урок №4 Индексы

© Компьютерная Академия ШАГ
www.itstep.org

Все права на охраняемые авторским правом фото-, аудио- и видеопроизведения, фрагменты которых использованы в материале, принадлежат их законным владельцам. Фрагменты произведений используются в иллюстративных целях в объеме, оправданном поставленной задачей, в рамках учебного процесса и в учебных целях, в соответствии со ст. 1274 ч. 4 ГК РФ и ст. 21 и 23 Закона Украины «Про авторське право і суміжні права». Объем и способ цитируемых произведений соответствует принятым нормам, не наносит ущерба нормальному использованию объектов авторского права и не ущемляет законные интересы автора и правообладателей. Цитируемые фрагменты произведений на момент использования не могут быть заменены альтернативными, не охраняемыми авторским правом аналогами, и как таковые соответствуют критериям добросовестного использования и честного использования.

Все права защищены. Полное или частичное копирование материалов запрещено. Согласование использования произведений или их фрагментов производится с авторами и правообладателями. Согласованное использование материалов возможно только при указании источника.

Ответственность за несанкционированное копирование и коммерческое использование материалов определяется действующим законодательством Украины.