

## **СОЗДАНИЕ И НАСТРОЙКА ИНДЕКСОВ**

### **Содержание:**

[Урок 1: Планирование индексов](#)

[Урок 2: Создание индексов](#)

[Урок 3: Оптимизация индексов](#)

[Урок 4: Создание индексов XML](#)

## СОЗДАНИЕ И НАСТРОЙКА ИНДЕКСОВ



### Цели

После завершения этой темы, студенты смогут:

- Планировать индексы
- Создавать индексы
- Оптимизировать индексы
- Создавать индексы XML

### Введение

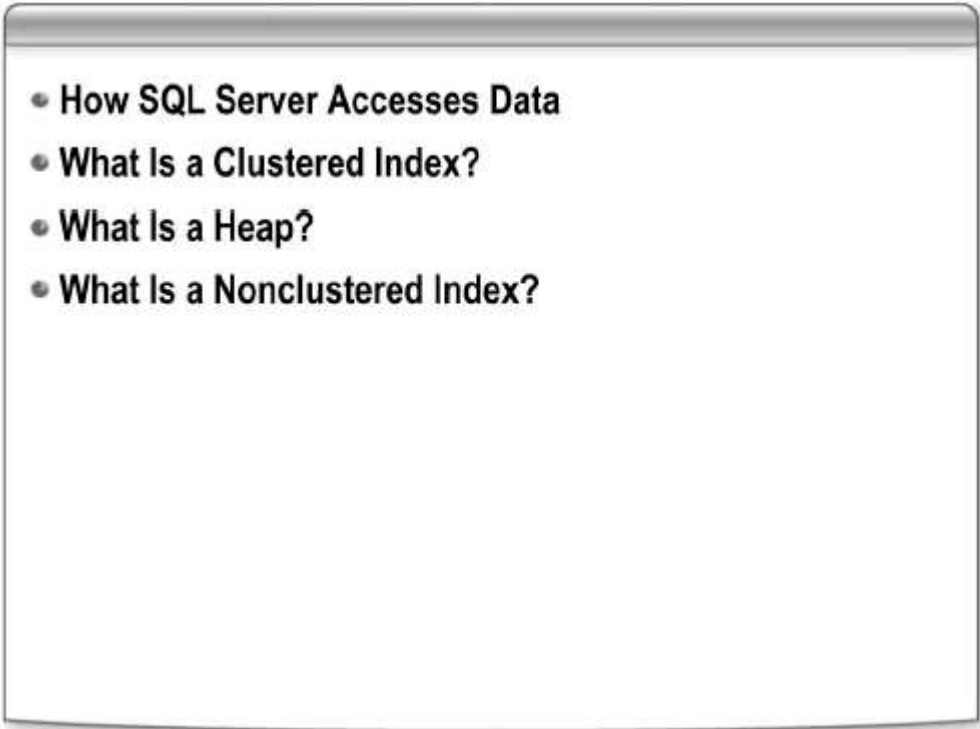
Индекс - коллекция страниц, связанных с таблицей (или представлением), предназначенная для повышения скорости поиска строк таблицы или для обеспечения уникальности. Например, без индекса необходимо было бы пролистать весь учебник страница за страницей, чтобы найти информацию о нужной теме. Microsoft® SQL Server™ 2005 использует индексы, чтобы указать местоположение строки на странице данных вместо того, чтобы просматривать все страницы данных таблицы.

Индекс содержит ключи, построенные из одного или более столбцов в таблице. Эти ключи хранятся таким образом, что SQL Server 2005 ищет строки по этому ключу быстро и эффективно.

Этот модуль содержит краткий обзор планирования, создания и оптимизации индексов. Здесь рассматриваются различия между кучами, кластеризованными индексами и некластеризованными индексами, а также способы использования каждого из этих типов индексов. Кроме того описывается, как создавать индексы различных типов и как формировать и поддерживать их для достижения оптимальной производительности.

**Замечание.** Информация об индексах таблиц, представленная в этом модуле, также относится к индексам представлений. В данном модуле рассматриваются индексы таблиц. Однако данный материал вполне применим также к представлениям.

## Урок 1: Планирование индексов

- 
- How SQL Server Accesses Data
  - What Is a Clustered Index?
  - What Is a Heap?
  - What Is a Nonclustered Index?

### Цели урока

После завершения этого урока, студенты смогут:

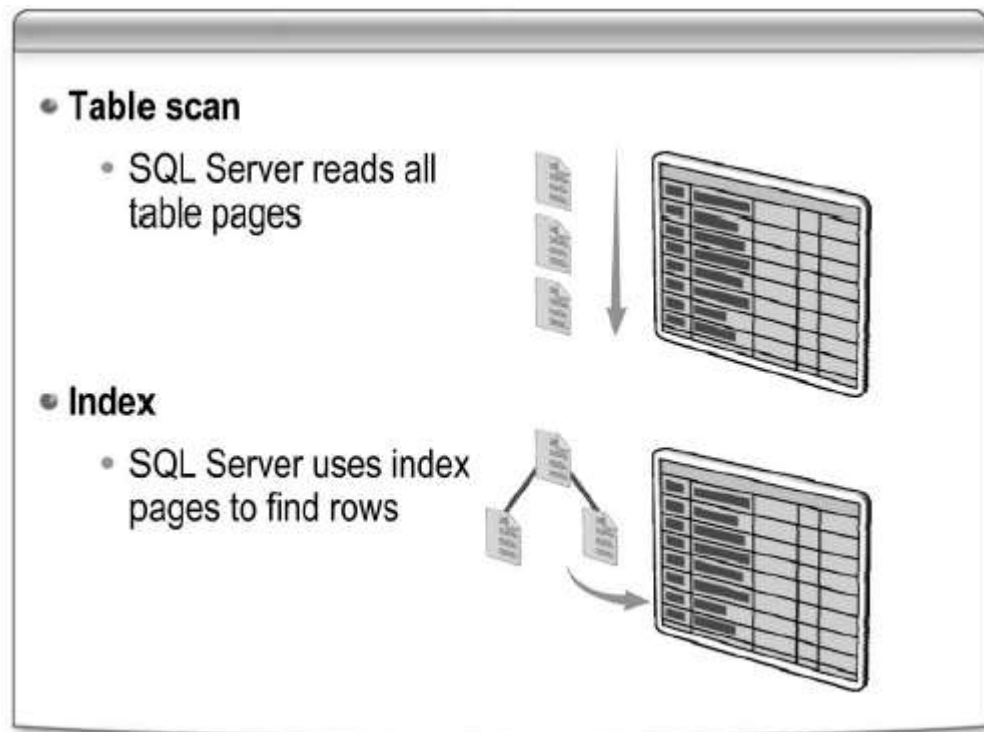
- Описывать, как SQL Server получает доступ к данным.
- Создавать кластеризованный индекс и определять, когда использовать кластеризованный индекс.
- Создавать кучу и определять, когда использовать кучи.
- Создавать некластеризованный индекс и определять, когда использовать некластеризованный индекс.

### Введение

В этом уроке, Вы изучите три фундаментальных варианта индексации, предлагаемые SQL Server 2005: кластеризованные индексы, кучи и некластеризованные индексы. Вы узнаете преимущества и недостатки каждого типа индекса и сможете определять, когда использовать тот или иной тип индекса.

Планирование индексов является одним из самых важных аспектов повышения производительности. Это требует понимания как структуры индекса так и того, как используются данные. Освоение основ того, как хранятся данные и как осуществляется доступ к ним, является первым шагом для понимания того, как работают индексы, почему нужно использовать их и в каких случаях нужно использовать определенный тип индекса, поддерживаемый SQL Server 2005.

## Как осуществляется доступ к данным в SQL Server 2005



## Как осуществляется доступ к данным в SQL Server 2005

SQL Server получает доступ к данным одним из двух способов:

■ **сканирование таблицы** (*полный просмотр таблицы*). Когда SQL Server выполняет просмотр таблицы, он:

1. Начинает просмотр с начала таблицы.
2. Сканирует страницу за страницей, чтобы просмотреть все строки таблицы.
3. Извлекает строки, которые отвечают критериям запроса.

■ **индексное сканирование**. Когда SQL Server использует индекс, он:

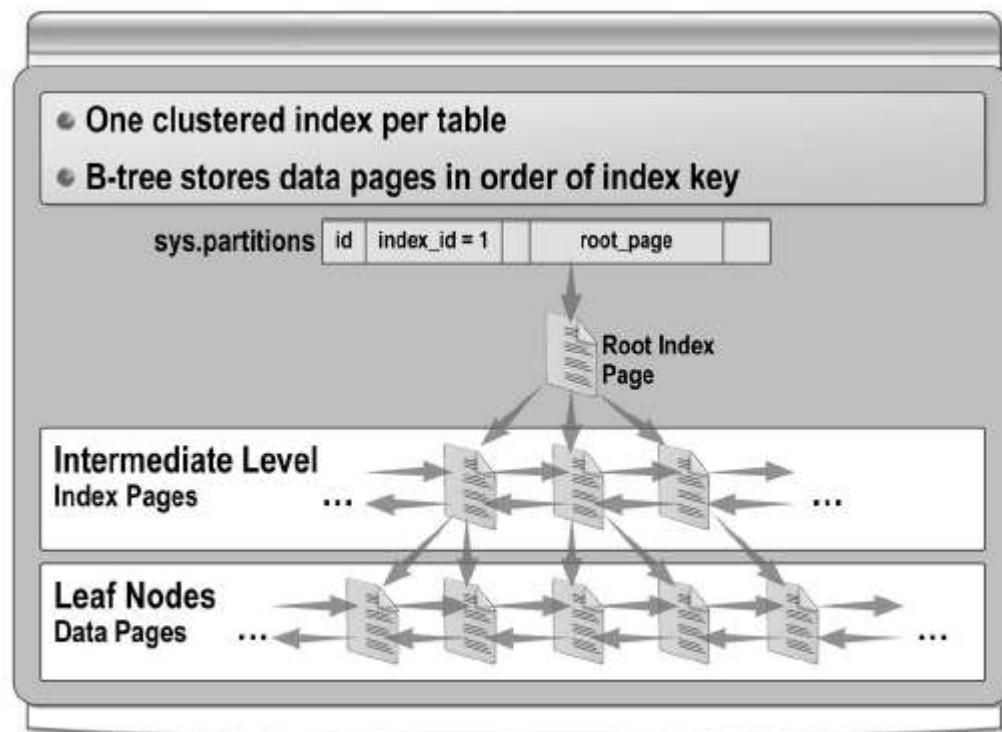
1. Проходит структуру дерева индекса, чтобы найти строки, соответствующие запросу.
2. Извлекает строки, которые отвечают критериям запроса.

SQL Server сначала определяет, существует ли индекс. Затем оптимизатор запросов – компонент, ответственный за построение оптимального плана выполнения – планирует

запрос, т.е. определяет, что будет эффективнее, просмотреть всю таблицу или использовать индекс для того, чтобы найти нужные данные.

**Дополнительная информация.** Для получения дополнительной информации о том, как SQL Server получает доступ к данным см. "Архитектура обработки запросов" в SQL Server Books Online.

## Что такое кластеризованный индекс?



## Что такое кластеризованный индекс?

Кластеризованный индекс сортирует и хранит строки данных таблицы в соответствии с ключом кластеризованного индекса. Кластеризованный индекс реализуется как В-дерево. Каждая страница в В-дереве называется узлом индекса. Верхний узел В-дерева называют корневым узлом. Нижний уровень в индексе называют листовым уровнем. Все уровни индекса между корневым и листовыми узлами называются промежуточными узлами. Каждая страница в промежуточном или листовом уровне содержит указатель на предшествующую и последующую страницы, т.е. каждый из этих уровней представляет собой двунаправленный связанный список. Эта структура обеспечивает высокоэффективный механизм ускорения процесса поиска данных.

В кластеризованном индексе корневой и промежуточные узлы содержат страницы индекса с индексными строками. Каждая строка индекса содержит ключ и указатель или на промежуточную страницу в В-дереве или на строку данных в листовом уровне индекса. Страницы в каждом уровне индекса связаны в двунаправленный связанный список.

Поскольку кластеризованный индекс определяет порядок, в котором строки таблицы хранятся фактически, то у каждой таблицы может быть только один кластеризованный индекс



**Важно!** Столбцы со следующими типами данных не могут использоваться как ключи кластеризованного индекса: **ntext**, **text**, **varchar (Max)**, **nvarchar (Max)**, **varbinary (Max)**, **xml**, или **image**.

### **Когда использовать кластеризованный индекс**

Поскольку может быть только один кластеризованный индекс на таблицу, необходимо гарантировать, что он создается для достижения максимально возможной выгоды. Прежде, чем создавать кластеризованный индекс, необходимо понять, как осуществляется доступ к данным. Поскольку кластерный индекс определяет порядок, в котором SQL Server 2005 хранит строки данных таблицы, существуют определенные типы данных и варианты использования кластеризованных индексов.

Кластеризованные индексы являются эффективными, когда используются для поддержки следующих запросов:

- Запросы возвращают диапазон значений при использовании таких операторов как BETWEEN, >, >=, < и <=.

Поскольку данные таблицы физически хранятся в порядке индекса, после найденной строки с первым значением при использовании кластеризованного индекса, строки с последующим значением индекса будут физически смежны.

- Запросы возвращают отсортированные данные при использовании ORDER BY или GROUP BY. Индексы на столбцах, определенных в ORDER BY или GROUP BY, могли бы удалить потребность в сортировке для двигателя базы данных, потому что строки уже сортированы индексом. Это улучшает производительность запроса.

- Запросы возвращают соединенные данные при использовании JOIN; типично кластеризованный индекс создается по внешнему ключу.

- Запросы возвращают большие наборы данных.

### **Соглашения по использованию кластеризованных индексов**

При создании кластеризованного индекса, необходимо определить ключ индекса с наименьшим количеством столбцов, насколько это возможно. Хранение маленького ключевого значения увеличит число строк индекса, которые могут быть помещены на

страницу индекса и сократит число уровней, которое должно быть пройдено для поиска данных. Это минимизирует время ввода / вывода. Кроме того, рассмотрите следующие столбцы в качестве кандидатов на кластеризованный ключ:

- столбцы уникальны или содержат много различных значений, включая определенные как `IDENTITY`, потому что такой столбец гарантированно будет уникален в пределах таблицы. Уникальность значения ключа поддерживается явно при использовании ключевого слова `UNIQUE`, или неявно, при использовании внутреннего уникального идентификатора. Если кластеризованный индекс содержит значения-дубликаты, SQL Server должен различить строки, которые содержат идентичные значения в ключевом столбце или столбцах. Это достигается при использовании 4-байтового целого числа (`uniquifier value`) в дополнительном системном столбце (`uniquifier column`). Этот уникальный идентификатор является внутренним для SQL Server и недоступен для пользователя.

- столбцы часто используются для сортировки данных таблицы, потому что это экономит стоимость запроса на операции сортировки всякий раз, когда данные сортируются по этим столбцам.

- столбцы часто используются для выборки последовательных значений.

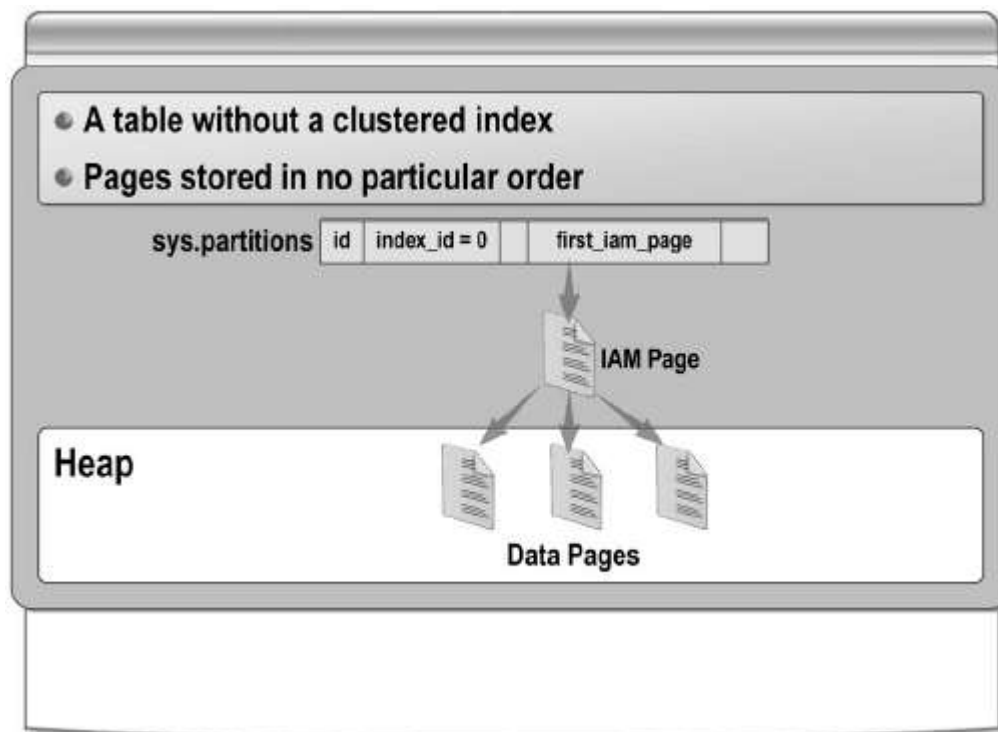
### **Когда не следует использовать кластеризованные индексы**

Кластеризованные индексы не следует создавать, когда:

- данные ключа индекса будут изменяться часто. Изменения в кластерном индексе приводят к тому, что вся строка данных должна быть перемещена, потому что движатель базы данных должен хранить значения данных строки в физическом порядке в соответствии с индексом. Это – важное соглашение в системах обработки транзакций, в которых данные типично изменчивы.

- ключи индекса *широкие*. Широкие ключи – это сложные ключи на нескольких столбцах или несколько крупных столбцов. Все некластеризованные индексы используют ключевые значения от кластеризованного индекса как ключи поиска. Любые некластеризованные индексы, определенные на той же самой таблице, будут значительно большими, потому что некластеризованные записи индекса содержат кластеризованный ключ, а также ключевые столбцы, определенные для этого некластеризованного индекса.

## Что такое Куча?



## Что такое куча?

Куча – таблица без кластеризованного индекса. В этом случае строки данных не хранятся в каком либо специфическом порядке, и нет никакого специфического порядка для последовательности страниц данных. Страницы данных не связаны в список. SQL Server всегда поддерживает страницы данных в куче, если только для таблицы не определен кластеризованный индекс. Чтобы поддерживать кучи, SQL Server использует Карту Распределения Индекса (Index Allocation Map, IAM) страницы.

Страницы IAM:

- Содержат информацию о том, где SQL Server хранит экстенды кучи. Системная таблица **sys.partitions** хранит указатель на первую страницу IAM, связанную с кучей. Это будет запись с `index_id = 0`.
- Позволяют осуществлять навигацию через кучу, чтобы найти доступное место для вставки новых строк таблицы.
- Связывают страницы данных с таблицей. Страницы данных и строки в ней них не хранятся в каком-либо определенном порядке и не связаны между собой.

Единственная логическая связь между страницами данных – это та, в которой они записываются в страницах IAM.

### **Когда использовать кучу**

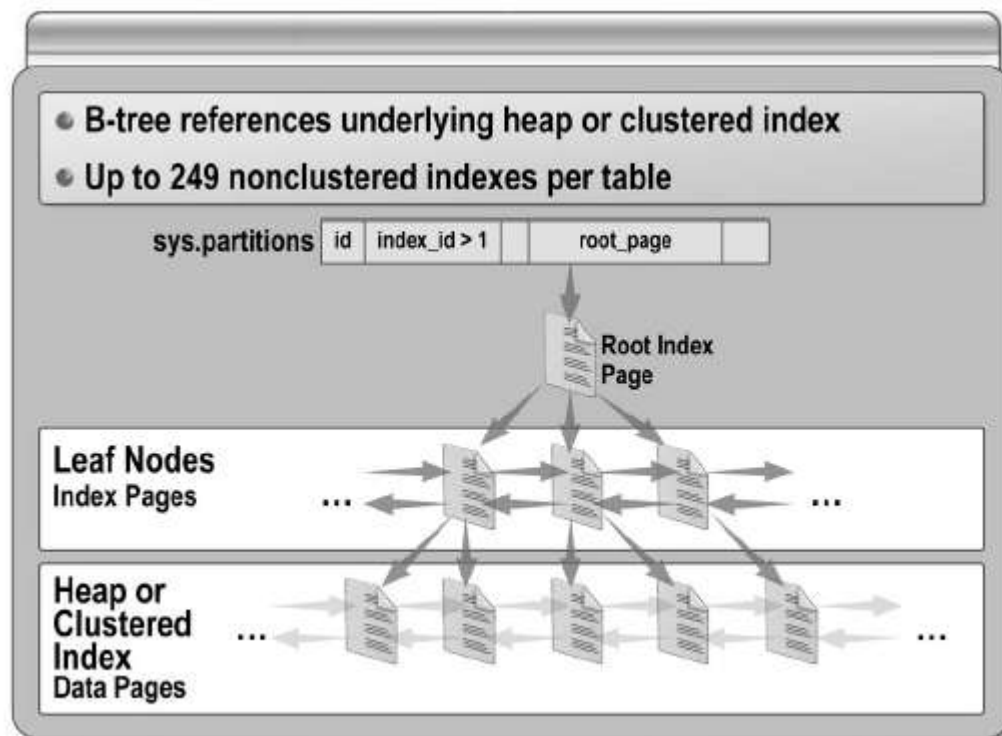
Куча используется по умолчанию всякий раз, когда для таблицы не определен кластеризованный индекс. Куча используется, если для данной таблицы нецелесообразно использовать кластеризованный индекс. В этом случае можно определить любое количество некластеризованных индексов.

Для таблиц используются кучи, в случае если они:

- Содержат изменчивые данные, где строки часто добавляются, удаляются и обновляются. Стоимость обслуживания индекса может превышать выгоды.
- Содержат маленькое количество данных. Полное сканирование таблицы для поиска данных может быть более быстрым чем поддержание и использование индекса.
- Содержат преимущественно повторяющиеся строки данных. Сканирование таблицы может быть более быстрым чем индексный поиск.
- Содержат данные, которые записываются и редко читаются, как журнал аудита. Индекс может вызвать ненужное хранение и обслуживание.

**Важно!** Когда Вы создаете ограничение первичного ключа (PRIMARY KEY) для таблицы, автоматически создается уникальный индекс на столбец (или столбцы). По умолчанию, этот индекс кластеризованный, т.е. таблица больше не хранится как куча. Вы можете определить некластеризованный индекс при создании ограничения с опцией NONCLUSTERED. Для большей информации о первичных ключах, см. “Ограничения PRIMARY KEY” в SQL Server Books Online.

## Что такое некластеризованный индекс?



## Что такое некластеризованный индекс?

Некластеризованные индексы имеют ту же самую структуру В-дерева как кластеризованные индексы за исключением того, что строки данных таблицы не сортируются и не хранятся в соответствии с некластеризованными ключами. В некластеризованном индексе данные и индекс хранятся отдельно, а лиственный уровень индекса состоит из страниц индекса вместо страниц данных.

Строки в некластеризованном индексе хранятся в порядке значений ключа индекса, но для строк данных нет гарантии какого-либо специфического порядка, если только для таблицы не создан кластеризованный индекс. Каждая строка индекса в некластеризованном индексе содержит некластеризованное значение ключа и локатор строки. Этот локатор указывает на данные строки, если таблица является кучей, и содержит кластеризованный ключ индекса в случае, если таблица имеет кластеризованный индекс.

Если индексируемая таблица имеет кластеризованный индекс, столбец или столбцы, определенные в кластеризованном индексе, автоматически добавляются в конец каждого некластеризованного индекса таблицы. Это делает возможным не указывать кластеризованный индекс столбца в определении некластеризованного индекса. Например,

если у таблицы есть кластеризованный индекс по столбцу **C**, некластеризованный индекс по столбцам **B** и **A** будет иметь в качестве ключа значения столбцов **B**, **A**, и **C**.

У таблицы может быть до 249 некластеризованных индексов. Некластеризованные индексы по таблице могут быть определены независимо от того, использует ли таблица кластеризованный индекс или кучу.

**Важно!** Колонки со следующими типами данных не могут использоваться как ключ в некластеризованном индексе: **ntext**, **text**, **varchar (Max)**, **nvarchar (Max)**, **varbinary (Max)**, **xml**, или **image**.

### Когда использовать некластеризованный индекс

Некластеризованные индексы полезны, когда пользователи запрашивают различные данные. Например, пользователь может часто выполнять поиск по базе данных сада, запрашивая общие и научные названия растений. Можно создать некластеризованный индекс для поиска научных названий и кластеризованный индекс для поиска общих названий. Некластеризованные индексы предназначены для улучшения работы часто используемых запросов, которые не покрываются кластеризованным индексом. Если таблица уже имеет кластеризованный индекс и необходимо проиндексировать еще один столбец, то нет иного пути, кроме как использовать некластеризованный индекс. Прежде, чем создавать некластеризованные индексы, необходимо понять, как осуществляется доступ к данным.

Вы достигаете максимального увеличения производительности запросов, когда индекс содержит все столбцы запроса. Термин *охват* (*coverage*) относится к столбцам запроса, которые поддерживаются индексом. С полным охватом оптимизатор запросов может определить местонахождение всех значений столбца в индексе, т.е. данные кучи или кластеризованного индекса не требуются и это приводит к меньшему количеству дисковых операций ввода / вывода. Однако широкие ключи и слишком большое количество индексов могут также быть вредными для общей производительности базы данных.

Использование некластеризованного индекса целесообразно, когда:

- Необходимо улучшить производительность запросов, которые используют JOIN или GROUP BY. Вы должны создать множественные некластеризованные индексы по столбцам соединения или группировки, а также кластеризованный индекс по внешнему ключу.

■ Таблица имеет низкую частоту обновления, но содержит большие объемы данных – например, приложение для поддержки принятия решения. Такие приложения содержат большое количество данных преимущественно только для чтения, и использование при этом множества некластеризованных индексов может принести значительный выигрыш в производительности. У оптимизатора запросов, таким образом, есть больше индексов для выбора и возможность определить самый быстрый метод доступа, а низкая частота обновления базы данных означает, что обслуживание индекса вряд ли будет снижать производительность.

■ Запросы не возвращают большие наборы данных.

■ Необходимо индексировать те столбцы, которые часто вовлекаются в условия поиска запроса (WHERE) и возвращают точные соответствия.

■ Необходимо индексировать те столбцы, которые содержат много различных значений, таких как комбинация фамилии и имени. Некластеризованные индексы работают лучше всего по столбцам, в которых селективность данных располагается от очень селективного до уникального.

### **Соглашения по использованию некластеризованных индексов**

При создании некластеризованных индексов следует руководствоваться следующими принципами:

■ Создавать сначала кластеризованные индексы, а затем – некластеризованные индексы. SQL Server автоматически перестраивает существующие некластеризованные индексы когда:

- существующий кластеризованный индекс удаляется.
- кластеризованный индекс создается.
- столбцы кластеризованного индекса изменяются.

■ Избегайте большого количества некластеризованных индексов, когда таблица используется в приложениях онлайн обработки транзакций (OLTP) или часто обновляется. Большое число индексов по таблице ухудшают работу команд вставки, обновления и удаления, потому что все индексы должны быть обновлены одновременно с данными в таблице.

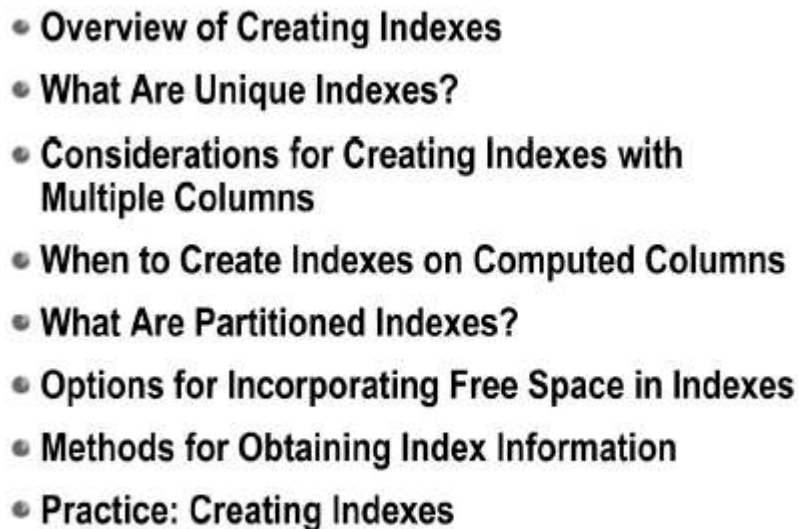
## **Факты о некластеризованных индексах**

При использовании некластеризованных индексов следует помнить, что:

- движатель базы данных создаст некластеризованный индекс, если Вы не определите явно то, что индекс должен быть кластеризованным.
- оптимизатор запросов не будет использовать индекс, если таблица содержит очень мало отличных значений в индексном столбце, потому что полное сканирование таблицы в этом случае будет эффективнее.
- порядок страниц листового уровня в некластеризованном индексе отличается от физического порядка страниц в таблице.



## Урок 2: Создание индексов

- 
- Overview of Creating Indexes
  - What Are Unique Indexes?
  - Considerations for Creating Indexes with Multiple Columns
  - When to Create Indexes on Computed Columns
  - What Are Partitioned Indexes?
  - Options for Incorporating Free Space in Indexes
  - Methods for Obtaining Index Information
  - Practice: Creating Indexes

### Цели урока

После завершения этого урока, студенты смогут:

- Описывать синтаксис и опции команд создания и удаления индексов.
- Описывать уникальные индексы, а также соглашения и синтаксис создания уникальных индексов.
- Описывать различия между сложными индексами и индексами с включенными столбцами.
- Описывать требования и синтаксис создания индексов по вычисляемым столбцам.
- Описывать секционированные индексы и соглашения по их созданию.
- Описывать соглашения для включения в индексы свободного пространства.
- Описывать способы получения информации об индексах.

### Введение

Приняв решение о том, что именно необходимо индексировать и какой индекс использовать – кластеризованный или некластеризованный, можно приступить к созданию

индекса. При этом доступно много опций, которые оказывают большое влияние на производительность и обслуживаемость (maintainability) индекса.

Из этого урока Вы узнаете, как создавать индексы и как использовать некоторые ключевые опции для конфигурации для улучшения полезности, производительности и обслуживаемости создаваемых индексов.

**Предупреждение.** Не индексируйте столбцы, которые определены как **ntext**, **text**, **varchar (Max)**, **nvarchar (Max)**, **varbinary (Max)**, **xml**, или **image**. Столбцы с этими типами данных не могут быть индексированы.

## Краткий обзор создания индексов

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ]  
INDEX index_name ON { table | view } ( column [ ASC | DESC  
] [ ,...n ] )  
INCLUDE ( column [ ,...n ] )  
[WITH option [ ,...n ] ]  
[ON {partition_scheme (column) | filegroup | "default" } ]
```

WITH option	Purpose
ALLOW_ROW_LOCKS	Enables/disables row-level locks on index
ALLOW_PAGE_LOCKS	Enables/disables page-level locks on index
ONLINE	Enables/disables access to index during creation
FILLFACTOR	Controls free space on leaf-level pages
PAD_INDEX	Controls free space on non-leaf-level pages

### Введение

Вы можете создавать индексы либо с помощью SQL Server Management Studio либо Transact-SQL.

### Создание индексов

Можно создать индекс с помощью обозревателя объектов (Object Explorer) в SQL Server Management Studio или команды Transact-SQL CREATE INDEX.

Чтобы создать индекс с помощью Transact-SQL, используют команду CREATE INDEX. Команда CREATE INDEX имеет следующий синтаксис.

```

CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ]
INDEX index_name ON { table | view } ( column [ ASC | DESC ] [ ,...n ] )
INCLUDE ( column [ ,...n ] )
[WITH
    [PAD_INDEX = { ON | OFF }]
    [[,] FILLFACTOR = fillfactor ]
    [[,] IGNORE_DUP_KEY = { ON | OFF }]
    [[,] ONLINE = { ON | OFF }]
    [[,] ALLOW_ROW_LOCKS = { ON | OFF }]
    [[,] ALLOW_PAGE_LOCKS = { ON | OFF }]]
[ON {partition_scheme (column) | filegroup | "default" } ]

```

Фраза WITH в синтаксисе команды создания индекса рассматривается далее в этом уроке.

**Дополнительная информация.** Команда CREATE INDEX предлагает дополнительные опции, которые здесь не рассматриваются. Для получения деталей относительно создания индексов при использовании Transact-SQL, см. “CREATE INDEX (Transact-SQL)” в SQL Server Books Online.

Следующий пример кода создает некластеризованный возрастающий индекс с именем **AK\_Employee\_LoginID** по столбцу **LoginID** для таблицы **HumanResources.Employee** базы данных **AdventureWorks**.

```

CREATE NONCLUSTERED INDEX [AK_Employee_LoginID]
ON [HumanResources].[Employee] ( [LoginID] ASC)

```

### Опции блокировки

Опции ALLOW\_ROW\_LOCKS и ALLOW\_PAGE\_LOCKS являются новыми в SQL Server 2005. Следующая таблица описывает функцию этих опций.

Опция	Описание
ALLOW_ROW_LOCKS	Определяет, разрешены ли замки уровня строк при доступе к индексу. Если замки уровня строк не разрешены, то вместо этого используются замки уровня страницы то вместо этого используются и замки уровня таблицы.
ALLOW_PAGE_LOCKS	Определяет, разрешены ли замки уровня страницы, при доступе к индексу. Если замки уровня страницы не

	разрешены, то вместо этого используются замки уровня строки и замки уровня таблицы.
--	---

Двигатель базы данных выберет соответствующий уровень замка в пределах ограничений, формируемых при использовании опций ALLOW\_ROW\_LOCKS и ALLOW\_PAGE\_LOCKS. Если и ALLOW\_ROW\_LOCKS и ALLOW\_PAGE\_LOCKS будут выключены (OFF), то будут использоваться только замки уровня таблицы.

Вот предыдущий пример с выключенными замками уровня строк.

```
CREATE NONCLUSTERED INDEX [AK_Employee_LoginID]
ON [HumanResources].[Employee] ( [LoginID] ASC)
WITH ALLOW_ROW_LOCKS = OFF
```

**Дополнительная информация.** Для получения дополнительной информации об опциях блокировки ALLOW\_ROW\_LOCKS и ALLOW\_PAGE\_LOCKS, см. "Настройка блокировки для индекса" в SQL Server Books Online.

### Онлайн vs офлайн

Опция ONLINE также является новой в SQL Server 2005. Эта опция определяет, доступны ли указываемые в команде CREATE INDEX таблицы и связанные с ними индексы для модификации данных для запросов во время выполнения операции индексирования.

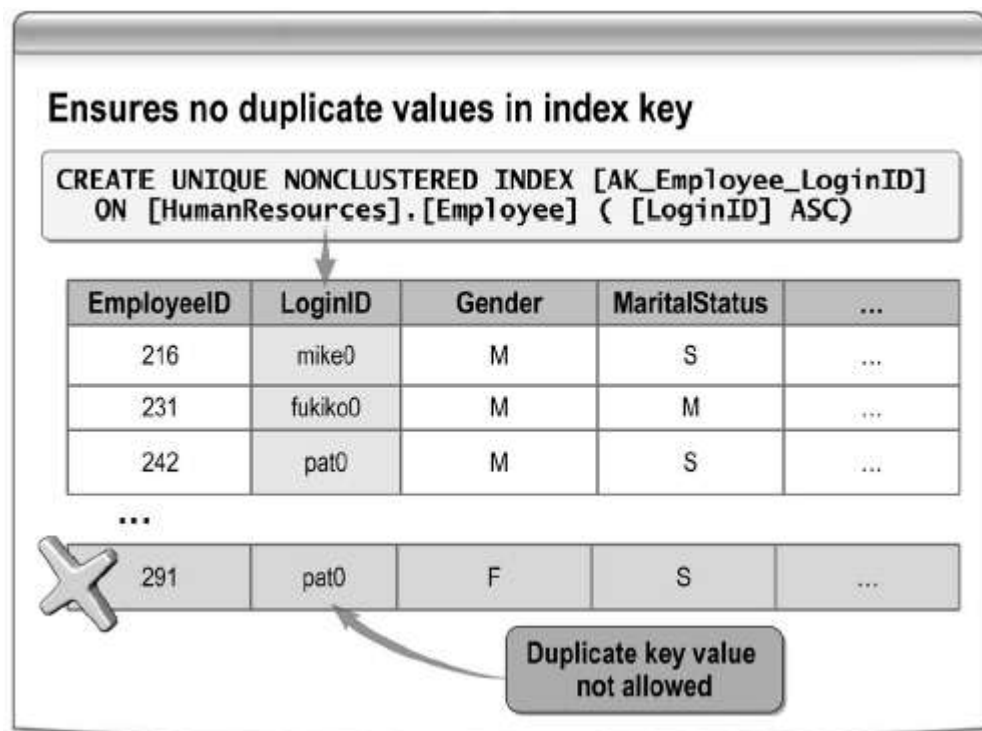
Вот предыдущий пример с включенной опцией онлайн.

```
CREATE NONCLUSTERED INDEX [AK_Employee_LoginID]
ON [HumanResources].[Employee] ( [LoginID] ASC)
WITH ONLINE = ON
```

**Замечание.** Операции индексирования с опцией ONLINE доступны только в SQL Server 2005 Enterprise Edition.

**Дополнительная информация.** Для получения дополнительной информации об опции ONLINE, см. "Как работают онлайн операции индексирования" в SQL Server Books Online.

## Что такое уникальные индексы?



## Что такое уникальный индекс?

Уникальный индекс - индекс, который гарантирует, что все данные в индексном столбце уникальны и не содержат одинаковых значений.

Когда существует уникальный индекс, при выполнении операции вставки движатель базы данных проверяет индексный столбец на значения дубликаты. Операции вставки, которые могут генерировать двойные ключевые значения, откатываются, и движатель базы данных показывает сообщение об ошибке. Это происходит, даже если операция вставки изменяет много строк, но есть попытка добавить только один дубликат. Однако если при попытке добавить данные, для которых существует уникальный индекс и в команде CREATE INDEX опция IGNORE\_DUP\_KEY была установлена в ON, будет отменена вставка только тех строк, которые нарушают уникальный индекс.

При создании индекса на таблицу, которая уже содержит данные, движатель базы данных гарантирует, что в индексном столбце нет повторяющихся значений.

**Дополнительная информация.** Для получения дополнительной информации об уникальных индексах, см. “Создание уникальных индексов” в SQL Server Books Online.

## Когда создавать уникальный индекс

Вы создаете уникальный индекс для кластеризованных или некластеризованных индексов, когда сами данные собственно уникальны. Необходимо создавать уникальные индексы только по столбцам, в которых должна быть реализована целостность сущностей. Например, не нужно создавать уникальный индекс по столбцу **LastName** таблицы **Person.Contact** в базе данных **AdventureWorks**, потому что у некоторых служащих могут быть одинаковые фамилии. Если некоторый столбец должен быть уникальным, то нужно для этого столбца создать ограничение первичного ключа (PRIMARY KEY) или уникальности (UNIQUE) вместо того, чтобы создавать уникальный индекс.

**Замечание.** Если таблица имеет ограничение первичного ключа (PRIMARY KEY) или уникальности (UNIQUE), SQL Server автоматически создает уникальный индекс при выполнении команд CREATE TABLE или ALTER TABLE. По умолчанию, этот индекс будет кластеризованным, но можно сделать его некластеризованным, если использовать опцию NONCLUSTERED в команде CREATE TABLE.

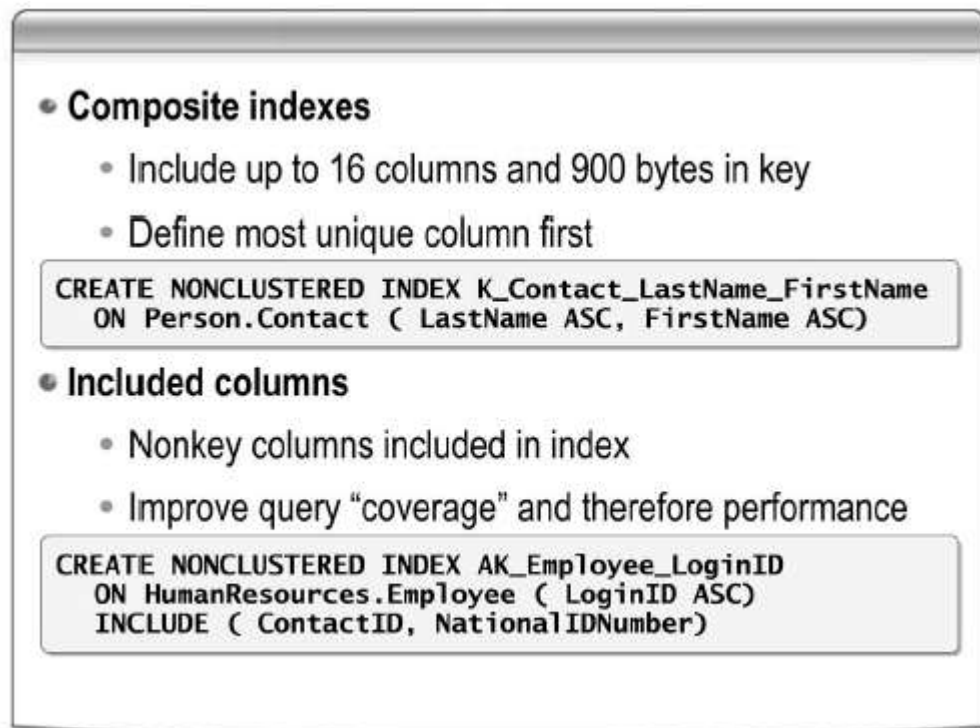
## Пример создания уникального индекса

При создании индекса в SQL Server Management Studio будет создан уникальный индекс, если установить флажок Unique в диалоговом окне New Index.

Следующий пример показывает код Transact-SQL для создания уникального возрастающего некластеризованного индекса с именем **AK\_Employee\_LoginID** по столбцу **LoginID** в таблице **HumanResources.Employee** базы данных **AdventureWorks**.

```
CREATE UNIQUE NONCLUSTERED INDEX [AK_Employee_LoginID]  
ON [HumanResources].[Employee] ( [LoginID] ASC)
```

## Соглашения по созданию сложных индексов



### Что такое композитный (составной) индекс?

Композитный или составной индекс определяет больше чем одну колонку в качестве ключа индекса. Производительность запросов улучшается при использовании составных индексов, особенно когда пользователи регулярно выполняют поиск информации больше чем одним способом. Однако широкие ключи требуют больше памяти для хранения индекса.

У композитных индексов есть следующие требования и ограничения:

- до 16 столбцов могут быть объединены для формирования одного композитного индекса.
- сумма длин столбцов, которые составляют композитный индекс, не может превышать 900 байт.
- фраза WHERE запроса должна ссылаться на первый столбец композитного индекса для того, чтобы оптимизатор запросов использовал этот индекс. Покрывающие индексы не подвергаются этому ограничению.
- столбцы в композитном индексе должны все быть из одной таблицы, кроме тех случаев, когда индекс создается для представления. В этом случае они должны все быть из одного представления.



- композитный индекс по столбцам (column1, column2) не то же самое, что индекс по столбцам (column2, column1) - у каждого из них свой порядок.

**Замечание.** Когда индекс содержит все столбцы, на которые ссылается запрос, это типично называется покрытием запроса. Покрытые запросы уменьшают дисковый ввод / вывод, и это может значительно улучшить производительность запроса, потому что оптимизатор может получить все результаты поиска со страниц индекса без дополнительного чтения со страниц данных.

### **Когда создавать композитный индекс**

Создавайте композитные индексы когда:

- Два или более столбцов используются для поиска.
- Запросы ссылаются только на столбцы индекса.

### **Принципы создания композитных индексов**

При создании композитных индексов следует руководствоваться следующими принципами:

- Самый уникальный столбец должен идти первым в индексе. Первый столбец, определенный в команде CREATE INDEX имеет наивысший порядок сортировки.
- Используйте композитный индекс для таблиц с составными ключами поиска.
- Используйте композитные индексы для увеличения производительности запросов и уменьшения числа индексов для таблиц.

### **Включенные столбцы**

SQL Server 2005 позволяет определять дополнительные столбцы таблицы, содержимое которых сохраняется с некластеризованным индексом. С помощью включения неключевых колонок можно создать некластеризованные индексы, которые покрывают больше запросов. У неключевых столбцов есть следующие дополнительные преимущества:

- Они могут быть определены с типами данных, не допустимыми для столбцов ключа индекса.

- Их не рассматривает движатель базы данных при вычислении числа столбцов в ключе индекса или размера ключа индекса.

**Дополнительная информация.** Для получения дополнительной информации о включенных столбцах, см. “Индекс с включенными столбцами” в SQL Server Books Online.

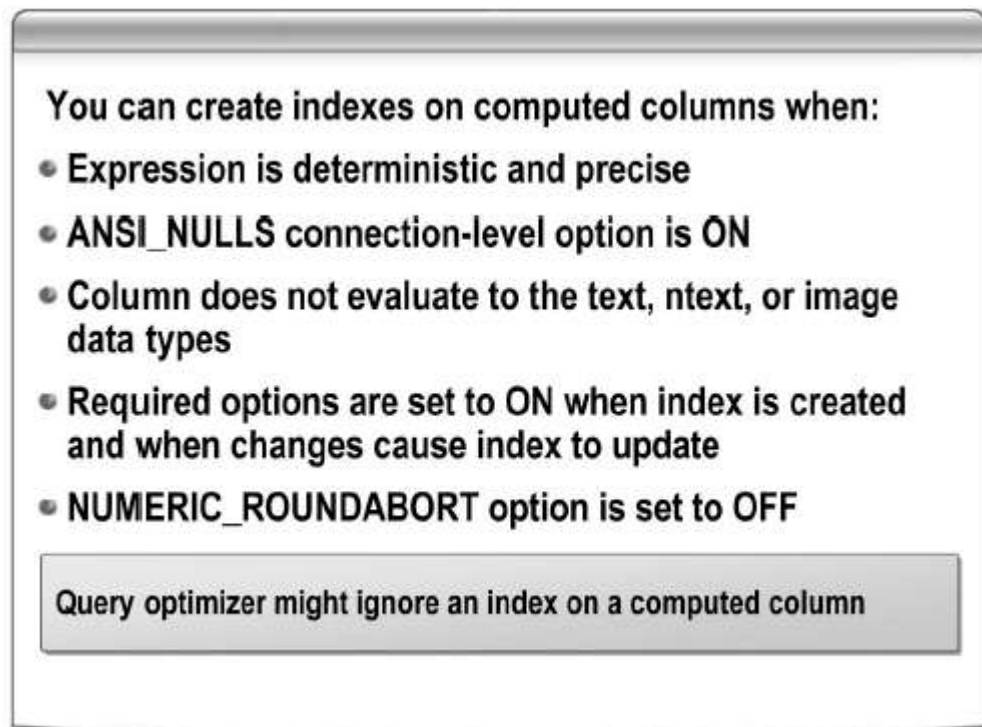
### **Пример создания сложных индексов**

Создать индекс с включенными столбцами можно с помощью SQL Server Management Studio. Для этого выберите дополнительные столбцы для включения в секции **Include Columns** диалогового окна **New Index**.

Следующий пример показывает код Transact-SQL для включения столбцов **ContactID** и **NationalIDNumber** в индекс с именем **AK\_Employee\_LoginID** по столбцу **LoginID** в таблице **HumanResources.Employee** базы данных **AdventureWorks**.

```
CREATE UNIQUE NONCLUSTERED INDEX [AK_Employee_LoginID]  
ON [HumanResources].[Employee] ( [LoginID] ASC)  
INCLUDE ( [ContactID], [NationalIDNumber])
```

## Когда создавать индексы по вычисляемым столбцам



## Когда создать индексы по вычисляемым столбцам

Можно создать индексы по вычисляемым столбцам когда:

- Вычисляемое выражение столбца должно быть *детерминированным* и *точным*. Детерминированные выражения всегда возвращают один и тот же результат для указанного набора входных данных. Точные выражения не включают типы данных float и real, а также типы данных, созданные на их основе.
- Опция уровня соединения ANSI\_NULLS установлена в ON при выполнении оператора CREATE TABLE. Функция OBJECTPROPERTY информирует, установлена ли данная опция в ON через свойство IsAnsiNullsOn.
- Вычисляемое выражение столбца не может вычислять результат типа text, ntext, или image.
- Соединение, в котором создается индекс – и все соединения, пытающиеся выполнить команды INSERT, UPDATE или DELETE, которые изменяют значения в индексе – имеют шесть опций SET, установленных в ON и одну опцию, установленную в OFF. Следующие опции должны быть установлены в ON:

- ANSI\_NULLS

- ANSI\_PADDING
- ANSI\_WARNINGS
- CONCAT\_NULL\_YIELDS\_NULL
- QUOTED\_IDENTIFIER
- ARITHABORT

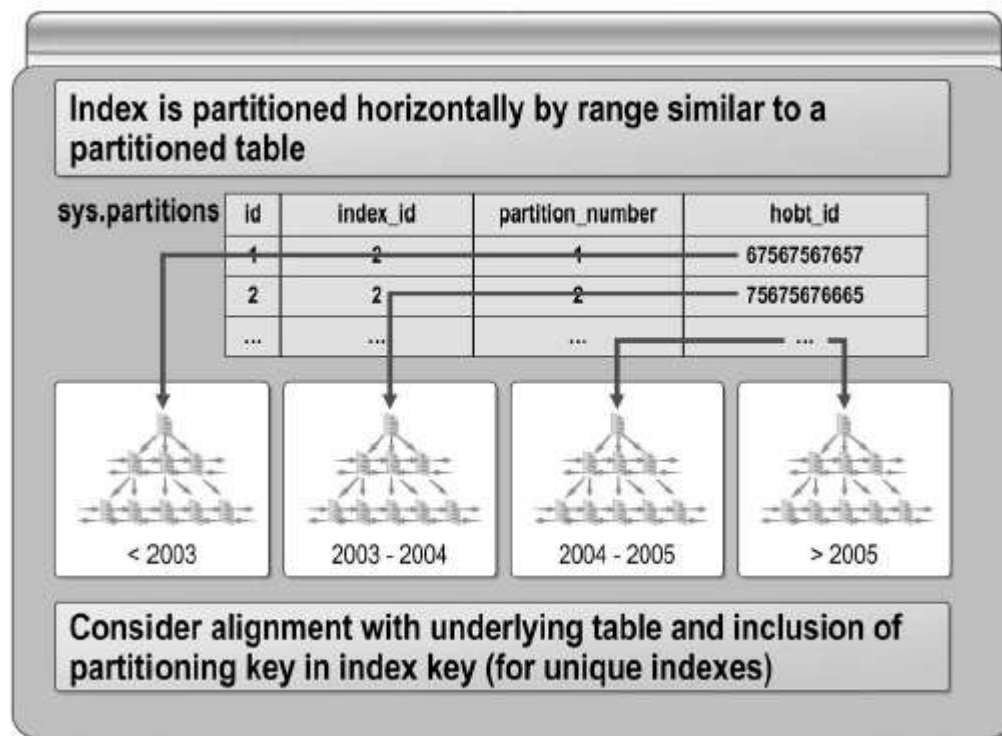
**Замечание.** В SQL Server 2005 при установке опции ANSI\_WARNINGS в ON неявно устанавливает опцию ARITHABORT в ON. В более ранних версиях SQL Server опция ARITHABORT должна быть явно установлена в ON.

■ Опция NUMERIC\_ROUNDABORT должна быть установлена в OFF.

**Замечание.** Оптимизатор запросов игнорирует индекс по вычисляемому столбцу для любой команды SELECT, которая выполняется соединением с другими настройками вышеперечисленных опций.

**Дополнительная информация.** Для получения дополнительной информации о включенных столбцах, см. “Создание индексов по вычисляемым столбцам” в SQL Server Books Online.

## Что такое секционированные индексы?



## Что такое секционированные индексы?

В SQL Server 2005, так же, как Вы можете разделить таблицы на основе диапазонов значений, Вы можете также разделить или секционировать индексы.

Подобно секционированным таблицам, секционированные индексы – это кластеризованные или некластеризованные индексы, в которых страницы индекса разделены горизонтально по различным физическим локациям на основе диапазона значений в столбце индекса. Физические локации для секций являются файловыми группами.

Вы делите индексы по той же самой причине, что Вы делите таблицы: для улучшения производительности и управляемости больших индексов, т.е. возможности выполнения задач управления в отдельной секции индекса, а не во всем индексе.

## Что такое привязка (alignment) индекса?

Индекс, который разделен таким же образом как его таблица, называется привязанным (aligned). Секционированный индекс является привязанным, если он отвечает следующим правилам:

- ключ секционирования индекса совместим с ключом секционирования таблицы.
- индекс имеет то же самое число секций, что и таблица.
- диапазон значений секций индекса соответствует диапазонам секций таблицы.

## Создание секционированных индексов

По умолчанию создание индекса по секционированной таблице создает привязанный индекс. Этот подход соответствует потребностям большинства сценариев. Однако можно также создать непривязанные секционированные индексы, которые используют их собственную логику разделения. Прежде, чем сделать это, необходимо:

1. Создать функцию секционирования для определения того, как индекс, который использует эту функцию, может быть секционирован.
2. Создать схему секционирования для определения размещения секций функции секционирования по файловым группам.

**Дополнительная информация.** Для получения дополнительной информации о создании функции и схемы секционирования, см. “Создание секционированных таблиц и индексов” и “Сценарии для выбора опций настройки” в SQL Server Books Online.

## Ключи секционирования и индексные ключи

Когда Вы создаете уникальный секционированный индекс, Вы должны включить ключ секционирования в индекс. Это может оказать влияние на Ваш проект базы данных. Например, таблица **Sales** могла бы использовать уникальное поле **OrderID**, чтобы идентифицировать коммерческие заказы, но для целей управляемости, Вы могли бы секционировать эту таблицу по времени. В этом случае, Вам нужно либо включить **OrderDate** и **OrderID** в композитный индекс, создать непривязанный индекс с использованием поля **OrderDate**, либо перепроектировать поле **OrderID** таким образом, чтобы оно включало дату.

Когда Вы создаете секционированный кластеризованный индекс, SQL Server автоматически добавляет ключ секционирования к индексу. SQL Server также добавит ключ секционирования к некластеризованным индексам, создаваемым по таблицам, которые были уже секционированы. SQL Server также автоматически секционирует некластеризованный индекс и сделает его привязанным, если опустить определенные детали конфигурации секционирования при создании индекса.

## Опции управления свободным пространством в индексах

- **Availability of free space affects performance of index updates**
- **FILLFACTOR determines the amount of free space on leaf nodes**
  - Use low FILLFACTOR for OLTP applications
  - Use high FILLFACTOR for OLAP applications
- **PAD\_FILL determines the amount of free space on non-leaf index nodes**

```
CREATE UNIQUE NONCLUSTERED INDEX [AK_Employee_LoginID]  
ON [HumanResources].[Employee] ( [LoginID] ASC)  
WITH ( FILLFACTOR = 65, PAD_INDEX = ON)
```

### Введение

Доступность свободного места на странице индекса может иметь существенное влияние на производительность операций обновления индекса. Если должна быть вставлена запись индекса и там нет свободного места, то создается новая страница индекса, а содержимое старой страницы расщепляется на две страницы. Это может ухудшить производительность, если это случается слишком часто.

SQL Server 2005 предлагает две важных опции, которые позволяют контролировать количество свободного пространства в индексе: FILLFACTOR и PAD\_INDEX.

### Опция FILLFACTOR

Опция FILLFACTOR позволяет выбирать процент (от 0 до 100) свободного места на странице индекса листового уровня, чтобы уменьшить количество расщеплений страниц. Этот процент определяет, на сколько страницы листового уровня могут быть заполнены. Например, фактор заполнения в 65% заполняет страницы листового уровня на 65%, оставляя 35% свободного места страницы для новых строк.

Следующая таблица показывает параметры настройки опции FILLFACTOR и типичное окружение, в котором эти значения фактора заполнения используются.

Процент FILLFACTOR	Страницы листового уровня	Страницы не листового уровня	Деятельность по значениям ключа	Типичное бизнес окружение
0 (по умолчанию)	Заполняются полностью	Остается пространство в верхнем уровне дерева индекса	Нет или легкая модификация	OLAP
1–99	Заполняются на указанный процент	Остается пространство для одной индексной записи	Умеренная или значительная модификация	Смешанный или OLTP
100	Заполняются полностью	Остается пространство для одной индексной записи	Нет или легкая модификация	OLAP

Системная таблица **sysindexes** наряду с другой информацией об индексах хранит значение фактора заполнения, которое последним было применено к индексу. С помощью системной хранимой процедуры **sp\_configure** можно изменить значения фактора заполнения на уровне сервера.

**Замечание.** Главная цель опции FILLFACTOR состоит в том, чтобы отложить расщепление страниц. Поэтому это не подходит для таблиц, кластеризованных по идентичному столбцу, потому что они не подвергаются расщеплениям страниц.

## Руководящие принципы для настройки опции FILLFACTOR

Значение фактора заполнения для таблицы, зависит от того, как часто данные изменяются (команды INSERT и UPDATE), а также от среды окружения. В общем случае, необходимо:

- Использовать низкое значение фактора заполнения для онлайн обработки транзакций (OLTP). Это обеспечивает максимальное пространство для роста таблиц, где часто вставляются строки или часто изменяются значения ключа индекса.



- Использовать высокое значение фактора заполнения для онлайн аналитической обработки (OLAP).

**Дополнительная информация.** Для получения дополнительной информации об опции FILLFACTOR, см. “Фактор заполнения” в SQL Server Books Online.

### Опция PAD\_INDEX

Опция PAD\_INDEX позволяет определять процент заполнения нелистовых страниц индекса. Можно использовать опцию PAD\_INDEX только, когда опция FILLFACTOR определена, т.к. значение процента PAD\_INDEX определяется процентом для FILLFACTOR.

Следующая таблица показывает воздействие параметров настройки опции FILLFACTOR при использовании опции PAD\_INDEX и типичную среду окружения, в которой используются значения PAD\_INDEX.

Процент FILLFACTOR	Страницы листового уровня	Страницы не листового уровня	Деятельность по значениям ключа	Типичное бизнес окружение
1–99	Заполняются до указанного процент	Заполняются до указанного процент	Умеренная или значительная модификация	OLTP

По умолчанию SQL Server всегда оставляет достаточно места, чтобы расположить, по крайней мере одну строку индекса максимального размера для каждой нелистовой страницы, независимо от значения фактора заполнения. Число строк на нелистовой странице индекса всегда не меньше чем две, независимо значения фактора заполнения.

**Дополнительная информация.** Для получения дополнительной информации об опции PAD\_INDEX, см. “Фактор заполнения” в SQL Server Books Online.

### Пример конфигурирования свободного места в индексе

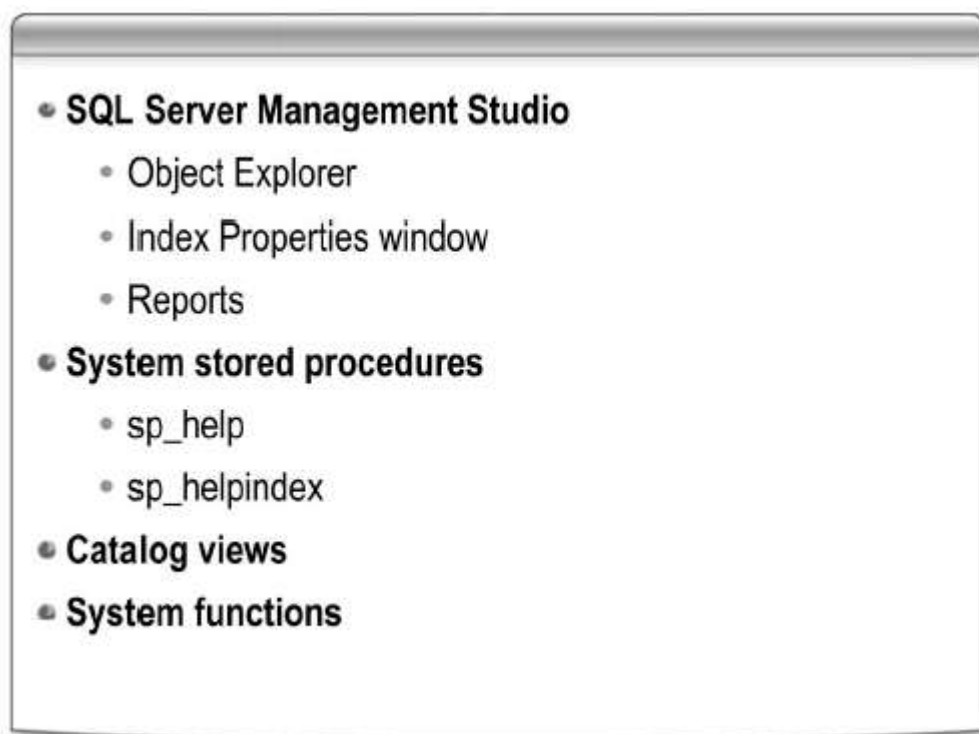
Вы можете настраивать опции FILLFACTOR и PAD\_INDEX при создании индекса в обозревателе объектов (Object Explorer) в среде SQL Server Management Studio. Для этого в диалоговом окне **New Index**, на вкладке **Options**, установите флажок **Set Fill Factor**. Это

позволит ввести значение фактора заполнения и откроет флажок **Pad Index**, который можно будет установить или снять.

Следующий пример показывает код Transact-SQL для установки значения опции FILLFACTOR в 65% и устанавливает опцию PAD\_INDEX в ON при создании индекса.

```
CREATE UNIQUE NONCLUSTERED INDEX [AK_Employee_LoginID]  
ON [HumanResources].[Employee] ( [LoginID] ASC)  
WITH ( FILLFACTOR = 65, PAD_INDEX = ON)
```

## Способы получения информации об индексах



### Введение

Прежде, чем создавать, изменять или удалять индекс, можно запросить информацию о существующих индексах. SQL Server 2005 обеспечивает много способов получения информации об индексах. Эти способы могут быть классифицированы следующим образом:

- SQL Server Management Studio
- Системные хранимые процедуры
- Представления Каталога
- Системные функции

### Использование SQL Server Management Studio для получения информации об индексах

SQL Server Management Studio обеспечивает визуальные инструменты, чтобы просматривать информацию об индексах в пределах окружающей среды управления. Следующая таблица перечисляет обычно используемые инструменты.

SQL Server Management Studio	Описание
Object Explorer	Позволяет осуществлять навигацию по иерархии базы данных, чтобы просмотреть список существующих

	индексов для определенной таблицы
Properties window	Позволяет просмотреть свойства любого индекса. Получить доступ можно правым щелчком по индексу в Object Explorer и затем выбором Properties
Reports	Позволяют генерировать отчеты уровня базы данных, показывая: <ul style="list-style-type: none"> <li>■ как пользователи и система используют индексы</li> <li>■ число операций, выполняемых на индексах</li> </ul>

### Использование системных хранимых процедур для получения информации об индексах

Системная хранимая процедура **sp\_helpindex** возвращает информацию об индексах, созданных для указанной таблицы. Данная информация включает для каждого индекса: название индекса, его описание и ключи.

Системная хранимая процедура **sp\_help** возвращает обширную информацию об указанной таблице, которая также включает ту же самую информацию об индексах, возвращаемую процедурой **sp\_helpindex**.

Следующий пример показывает, как получить информацию об индексах для таблицы **Production.Product** при использовании системной хранимой процедуры **sp\_helpindex**.

```
EXEC sp_helpindex [Production.Product]
```

Результаты показаны в следующей таблице.

index_name	index_description	index_keys
AK_Product_Name	nonclustered, unique, located on PRIMARY	Name
AK_Product_ProductNumber	nonclustered, unique, located on PRIMARY	ProductNumber
AK_Product_rowguid	nonclustered, unique, located on PRIMARY	rowguid
PK_Product_ProductID	clustered, unique, primary key located on PRIMARY	ProductID

**Дополнительная информация.** Для получения дополнительной информации об этих хранимых процедурах см. “sp\_help (Transact-SQL)” и “sp\_helpindex (Transact-SQL)” в SQL Server Books Online.

### **Использование представлений каталога для получения информации об индексах**

Следующая таблица перечисляет представления каталога, которые предоставляют информацию об индексах и описание предоставляемой информации.

<b>Представление каталога</b>	<b>Предоставляет информацию о</b>
sys.indexes	Тип индекса, файловая группа или ID схемы секционирования, и текущие установки индексных опций, которые хранятся в метаданных
sys.index_columns	ID столбца, его положение в индексе, тип (ключевой или неключевой) и порядок сортировки (ASC или DESC)
sys.stats	Статистики, связанные с индексом, включая название статистики и была ли она создана автоматически или пользователем
sys.stats_columns	ID столбца, связанного со статистикой
sys.xml_columns	XML тип индекса, первичный или вторичный, а также вторичный тип и описание

**Дополнительная информация.** Для получения дополнительной информации об этих представлениях каталога см. “Просмотр информации об индексах” в SQL Server Books Online.

### **Использование системных функций для получения информации об индексах**

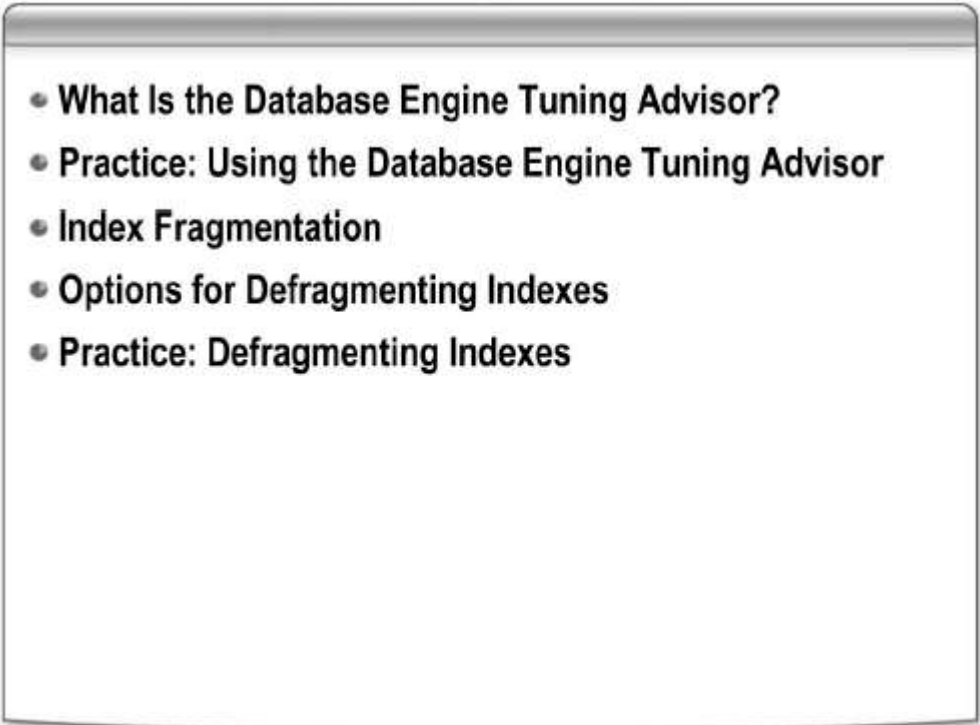
Следующая таблица перечисляет системные функции, которые предоставляют информацию об индексах и описание предоставляемой информации.

<b>Функция</b>	<b>Предоставляет информацию о</b>
sys.dm_db_index_physical_stats	Размер индекса и статистика фрагментации
sys.dm_db_index_operational_stats	Текущий индекс и таблица статистики ввода/ вывода

sys.dm_db_index_usage_stats	Статистика использования индекса по типу запроса
INDEXKEY_PROPERTY	Положение индексного столбца в индексе и порядок сортировки (ASC или DESC)
INDEXPROPERTY	Тип индекса, число уровней, а также текущие установки индексных опций, которые хранятся в метаданных
INDEX_COL	Имя ключевого столбца для указанного индекса

**Дополнительная информация.** Для получения дополнительной информации об этих функциях, см. “Просмотр информации об индексах” в SQL Server Books Online.

## Урок 3: Оптимизация индексов

- 
- **What Is the Database Engine Tuning Advisor?**
  - **Practice: Using the Database Engine Tuning Advisor**
  - **Index Fragmentation**
  - **Options for Defragmenting Indexes**
  - **Practice: Defragmenting Indexes**

### Цели урока

После завершения этого урока, студенты смогут:

- Описать Database Engine Turning Adviser
- Определить фрагментацию индекса
- Описать соглашения для дефрагментации индексов

### Введение

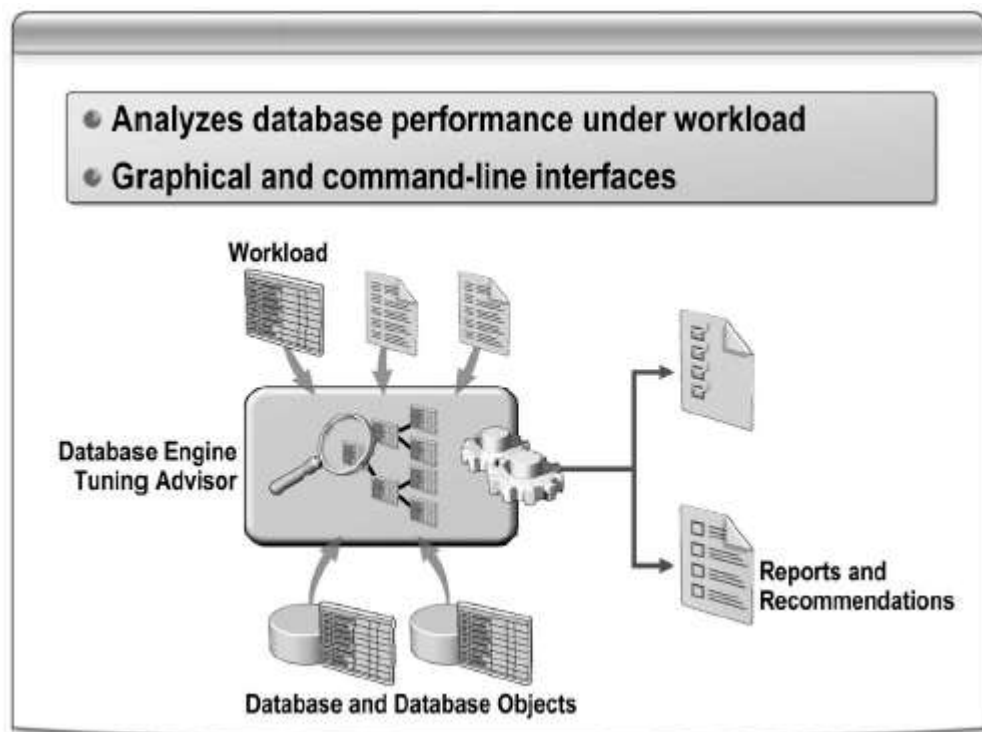
Эффективность индексов играет главную роль в общей производительности базы данных. Поэтому важно удостовериться, что индексы разработаны и реализованы наилучшим образом для поддержки приложений. После реализации индексов, необходимо поддерживать индексы, чтобы гарантировать их длительную оптимальную работу. Поскольку данные добавляются, изменяются и удаляются в базе данных, индексы становятся фрагментированными. В зависимости от среды окружения и цели приложения базы данных, фрагментация может быть или хорошей или плохой для производительности, но ею необходимо управлять соответственно потребностям.

Этот урок рассматривает главные инструменты, предоставляемые SQL Server 2005, которые помогают в оптимизации проекта индексирования и поддержке индексов, чтобы управлять фрагментацией на соответствующем уровне.

**Дополнительная информация.** Для получения дополнительной информации об оптимизации индексов см. “Оптимизация индексов” в SQL Server Books Online.



## Что такое Database Engine Turning Adviser?



### Введение

Database Engine Turning Adviser является новым инструментом в SQL Server 2005. До SQL Server 2005 некоторые функциональные возможности Database Engine Turning Adviser предоставлялись инструментом Index Turning Wizard. Database Engine Turning Adviser оценивает больше типов событий и структур и обеспечивает рекомендации более высокого качества. Database Engine Turning Adviser обеспечивает два интерфейса:

- автономный инструмент графического пользовательского интерфейса для настройки базы данных, а также просмотра рекомендаций по настройке и отчетов.
- утилиту командной строки (dta.exe), которая обеспечивает доступ к функциональным возможностям Database Engine Turning Adviser из командной строки и скриптов.

**Предостережение.** При первом использовании Database Engine Turning Adviser должен инициализироваться пользователем с разрешениями администратора системы. После инициализации любые пользователи, которые являются членами фиксированной роли базы данных **db\_owner**, могут использовать Database Engine Turning Adviser, чтобы настраивать таблицы баз данных, которыми они владеют.

## Как работает Database Engine Turning Adviser

Database Engine Turning Adviser – инструмент, который анализирует влияние рабочей нагрузки на производительность одной или более баз данных. Рабочая нагрузка – это набор команд Transact-SQL, который выполняется на базах данных, предназначенных для настройки. Источником рабочей нагрузки может быть файл, содержащий команды Transact-SQL, файл трассировки, сгенерированный инструментом SQL Profiler, или таблица информации о трассировке, также сгенерированная SQL Profiler.

Анализ может быть выполнен в любом из следующих двух режимов.

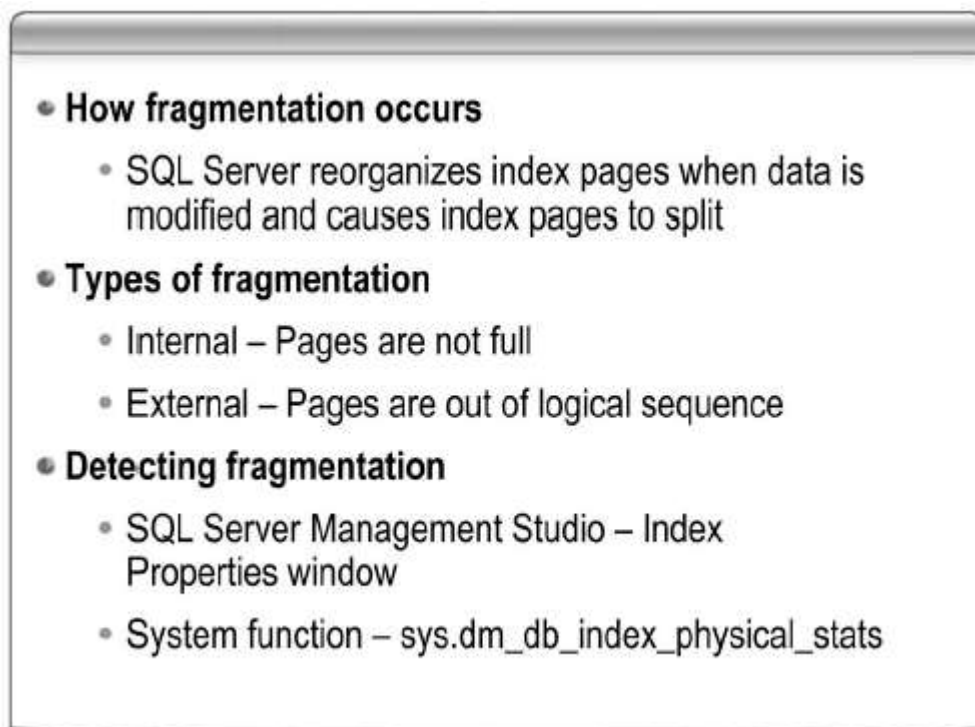
Тип анализа	Описание
Evaluate	В режиме Evaluate инструмент Database Engine Turning Adviser сравнивает стоимость текущей конфигурации (C) со стоимостью пользовательской конфигурации (U) для одной и той же рабочей нагрузки. C – всегда реальная конфигурация, потому что она состоит из физических структур проекта, которые в настоящее время существует в базе данных. U – конфигурация, которая состоит из реальных и гипотетических физических структур проекта. Если Database Engine Turning Adviser сообщает, что стоимость U ниже, чем стоимость C, вероятно, что физический проект U будет лучше, чем C.
Tune	В режиме Tune администратор базы данных уже знает, что часть физического проекта базы данных уже фиксирована, но хочет получить рекомендации от Database Engine Turning Adviser по улучшению физических структур проекта остальной части базы данных.

После анализа эффектов рабочей нагрузки на базах данных, Database Engine Turning Adviser выдает рекомендации. Эти рекомендации включают предложения по изменению в базе данных, такие как создание новых индексов, удаление существующих индексов и в зависимости от опций настройки рекомендации по секционированию.

Рекомендации предоставляются в виде набора команд Transact-SQL, которые произвели бы предложенные изменения. Можно просмотреть код Transact-SQL и сохранить его для более позднего обзора и применения, или можно осуществить рекомендуемые изменения немедленно.

**Дополнительная информация.** Для получения дополнительной информации о Database Engine Turning Adviser, см. “Настройку физического проекта базы данных” в SQL Server Books Online.

## Фрагментация индекса



### Что такое фрагментация?

Фрагментация индекса – неэффективное использование страниц в индексе. Фрагментация происходит в течение длительного времени, вследствие изменений данных. Например, когда строки данных добавляются или удаляются из таблицы, или когда значения индексных столбцов изменяются, SQL Server вносит соответствующие изменения в страницы индекса, чтобы поддержать хранение индексных данных. Регулирование страниц индекса известно как *расщепление страницы*. Процесс расщепления увеличивает размер таблицы и время на обработку запросов.

### Типы фрагментации

Существует два типа фрагментации индекса, внешняя и внутренняя, как описано в следующей таблице.

Тип фрагментации	Описания
Внутренняя	Неэффективное использование страниц в индексе, потому что количество данных, хранимых в пределах каждой страницы, меньше чем страница данных может содержать.

	Внутренняя фрагментация приводит к увеличенному логическому и физическому вводу / выводу, и требуется больше памяти для хранения строк в кэше. Эти дополнительные чтения могут привести к деградации в производительности запросов. Однако таблицы с интенсивной вставкой могут извлечь выгоду из внутренней фрагментации, потому что тогда меньше вероятность, что вставка вызовет расщепление страниц.
Внешняя	Неэффективное использование страниц в индексе, потому что логическая последовательность страниц является неправильной. Внешняя фрагментация замедляет дисковый доступ к строкам в результате поиска головкой диска, и такая фрагментация никогда не выгодна.

### Обнаружение фрагментации

Можно использовать SQL Server Management Studio или динамическую функцию управления **sys.dm\_db\_index\_physical\_stats**, чтобы определить степени фрагментации индексов.

Чтобы просмотреть детали фрагментации индекса в SQL Server Management Studio, откройте окно Properties для индекса, которым Вы интересуетесь, и затем щелкните страницу **Fragmentation**. В дополнение к некоторым общим свойствам о страницах в индексе, окно Properties показывает среднее наполнение страницы и общее количество фрагментации индекса в процентах. Чем выше значение, тем больше индекс фрагментирован.

Можно использовать **sys.dm\_db\_index\_physical\_stats**, чтобы просмотреть фрагментацию в определенном индексе, все индексы по таблице или представлению, все индексы в базе данных, или все индексы во всех базах данных. Аргументы функции **sys.dm\_db\_index\_physical\_stats** включают ID базы данных, таблицы, индекса и секции, которую Вы хотите оценить.

Результирующий набор функции **avg\_fragmentation\_in\_percent** включает столбец, который показывает среднюю фрагментацию индекса в процентах (то же самое число, показанное в окне свойств индекса в SQL Server Management Studio).

**Дополнительная информация.** Для получения дополнительной информации о функции **sys.dm\_db\_index\_physical\_stats** см. “sys.dm\_db\_index\_physical\_stats” в SQL Server Books Online.

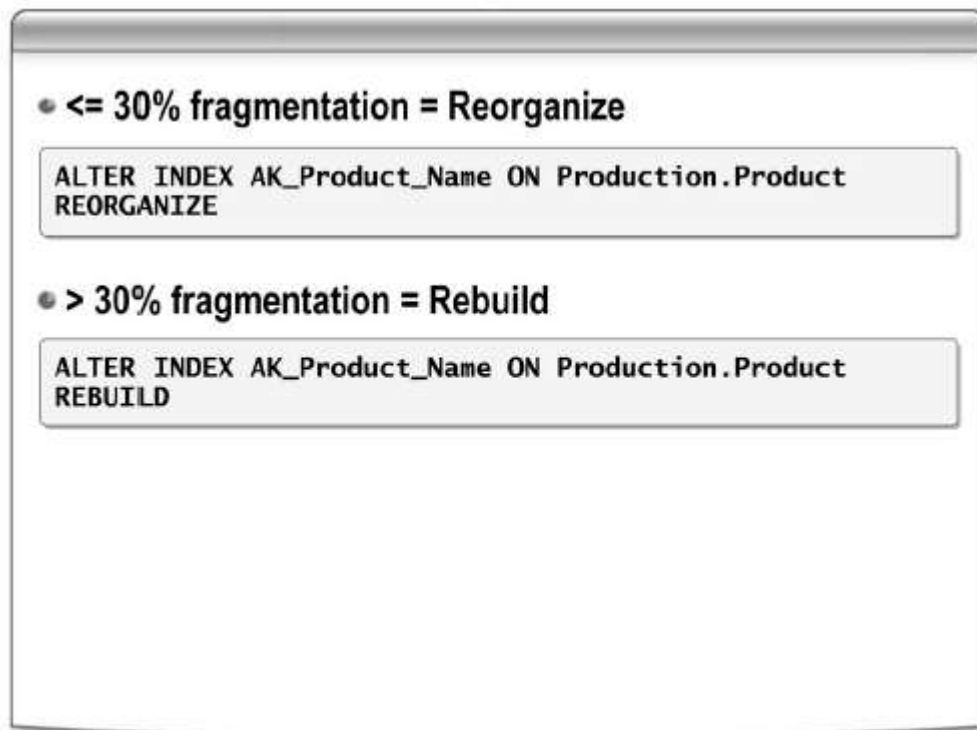
Следующий пример кода показывает, как получить среднюю фрагментацию по всем индексам на таблицу **Production.Product** при использовании **sys.dm\_db\_index\_physical\_stats**.

```
SELECT a.index_id, name, avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats (DB_ID(N'AdventureWorks'),
    OBJECT_ID(N'Production.Product'), NULL, NULL, NULL) AS a
JOIN sys.indexes AS b ON a.object_id = b.object_id AND a.index_id =
b.index_id;
```

Результаты будут выглядеть примерно следующими:

index_id	name	avg_fragmentation_in_percent
1	PK_Product_ProductID	23.0769230769231
2	AK_Product_ProductNumber	50
3	AK_Product_Name	66.6666666666667
4	AK_Product_rowguid	50

## Опции для дефрагментации индексов



## Опции для дефрагментации индексов

Есть два варианта для дефрагментации индекса: *реорганизация* и *перестроение*.

Реорганизация индекса дефрагментирует листовой уровень кластеризованных и некластеризованных индексов в таблицах, физически реорганизуя нелистовые страницы таким образом, чтобы они соответствовали логическому порядку (слева направо) листовых узлов. Хранение страниц в правильном порядке улучшает производительность индексного сканирования. Индекс реорганизуется только для существующих страниц, выделенных для этого; никаких новых страниц не выделяется. Если индекс занимает больше чем один файл, то все его файлы реорганизуются за один раз. Страницы не мигрируют между файлами.

Реорганизация индекса также уплотняет страницы индекса. Любые пустые страницы, созданные этим уплотнением, удаляются, обеспечивая дополнительное доступное дисковое пространство. Уплотнение выполняется основе значения фактора заполнения в представлении каталога **sys.indexes**.

Перестроение индекса удаляет индекс и создает новый. В процессе этого, фрагментация удаляется, дисковое пространство восстанавливается вследствие уплотнения страниц с использованием специфицированного в команде или существующего значения фактора

заполнения, а строки индекса переупорядочиваются в смежные страницы (выделяются новые страницы при необходимости). Это может улучшить производительность дисковых операций, вследствие уменьшения числа страниц, требуемых для получения запрошенных данных.

**Дополнительная информация.** Для получения дополнительной информации о дефрагментации индексов, см. “Реорганизация и перестроение индексов” в SQL Server Books Online.

### Реорганизация vs перестроение индекса

Решение относительно того, реорганизовывать или перестраивать индекс, чтобы удалить фрагментация, принимается на основе уровня фрагментации индекса, значение которого можно получить с помощью SQL Server Management Studio или динамической функции управления `sys.dm_db_index_physical_stats`.

Следующая таблица дает представление о лучшем подходе для удаления различных степеней фрагментации.

<b>avg_fragmentation_in_percent</b>	<b>Действие</b>
<= 30 %	Реорганизовать
> 30 %	Перестроить

**Замечание.** Если фрагментация составляет меньше чем 30%, и реорганизация индекса обеспечивает только небольшое улучшение, то следует попытаться перестроить индекс.

### Реорганизация индекса

Для дефрагментации индексов можно использовать опцию REORGANIZE команды ALTER INDEX. Команда ALTER INDEX с опцией REORGANIZE заменяет команду DBCC INDEXDEFRAG более ранних версий SQL Server.

Следующий пример кода показывает синтаксис команды ALTER INDEX для реорганизации индекса **AK\_Product\_Name**.

```
ALTER INDEX AK_Product_Name ON Production.Product  
REORGANIZE
```

Вы можете также реорганизовать *все* индексы таблицы, как показано в следующем примере.

```
ALTER INDEX ALL ON Production.Product  
REORGANIZE
```

### **Перестроение индекса**

Для дефрагментации индексов можно использовать опцию **REBUILD** команды **ALTER INDEX**. Команда **ALTER INDEX** с опцией **REBUILD** заменяет команду **DBCC DBREINDEX** более ранних версий SQL Server.

Следующий пример показывает синтаксис команды **ALTER INDEX**, для перестроения индекса **AK\_Product\_Name** таблицы **Production.Product**.

```
ALTER INDEX AK_Product_Name ON Production.Product  
REBUILD
```



## Урок 4: Создание индексов XML

- 
- What Are XML Indexes?
  - Types of XML Index
  - Practice: Creating XML Indexes

### Цели урока

После завершения этого урока, студенты смогут:

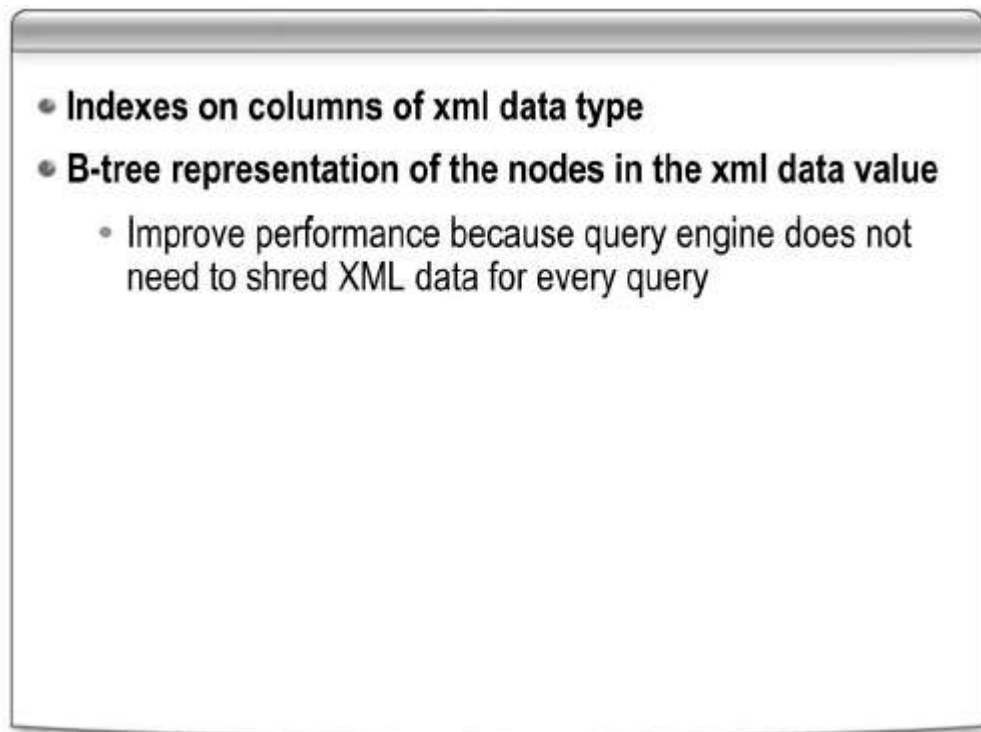
- Описывать индексы XML.
- Определять типы индексов XML.

### Введение

SQL Server 2005 оказывает обширную нативную поддержку для данных расширяемого языка разметки (XML). Экземпляры XML хранятся в столбцах типа **xml** как большие объекты (LOB). Эти экземпляры XML могут быть большими, хранимое бинарное представление данных типа **xml** может составить до 2 Гб (GB). Такие большие данные могут оказать существенное воздействие на производительность запросов; поэтому SQL Server 2005 поддерживает индексы XML, чтобы ускорить эти операции.

Этот урок описывает, что такое индексы XML, а также различные типы индексов XML. Кроме того, здесь рассматриваются вопросы, которые необходимо принять во внимание, создавая индексы XML, а также синтаксис создания индексов XML.

## Что такое индексы XML?



## Что такое индексы XML?

Индексы XML – индексы по столбцам таблицы, которые содержат данные типа **xml**. Когда запрос включает **xml**-столбец, процессор запроса должен разобрать XML каждый раз, когда выполняется запрос. Можно значительно улучшить производительность такого запроса, создав индекс по **xml**-столбцам.

При создании индексов XML, SQL Server создает кластеризованное представление в виде В-дерева узлов данных XML и сохраняет его во внутренней таблице. Индекс XML является кластеризованным по первичному ключу таблицы. Когда Вы запускаете запросы, включающие индексированный **xml** столбец, оптимизатор запросов будет всегда использовать этот индекс, за исключением случаев, когда нужно будет вернуть сам XML документ.

## Требования по использованию индексов XML

Прежде, чем использовать индекс XML, Вы должны знать о следующем:

- Прежде, чем создавать индекс XML, необходимо создать кластеризованный первичный ключ по таблице, содержащей **xml**-столбец.
- Нельзя изменить кластеризованный первичный ключ таблицы, если существует индекс XML по этой таблице. Прежде, чем изменить первичный ключ, необходимо будет удалить все индексы XML по таблице.
- Вы можете создать только один первичный индекс XML по **xml** столбцу.
- Нельзя создать XML-индекс и не-XML индекс по одной и той же таблице с одним и тем же именем.
- Нельзя использовать опции IGNORE\_DUP\_KEY и ONLINE при создании индекса XML.
- Нельзя создать индекс XML по **xml** столбцу в представлении, по табличной переменной с **xml** столбцами, или по **xml** переменной.
- Прежде, чем использовать команду ALTER TABLE для изменения индекса с типизированного на нетипизированный или наоборот, нужно удалить индекс XML по **xml** столбцу.
- Необходимо установить опцию ARITHABORT в ON, когда создается индекс XML или когда выполняется какая-либо операция модификации данных (INSERT, UPDATE, или DELETE) на **xml** столбце.

**Дополнительная информация.** Для получения дополнительной информации об индексах XML, см. “Индексы по столбцам типа данных xml” в SQL Server Books Online.

## Типы индексов XML

Index type		Description
Primary		Clustered B-tree representation of the nodes contained in an xml column Required to support one or more secondary xml indexes
Secondary	Path	For retrieving data by path and value
	Property	For retrieving data by specifying a path
	Value	For retrieving data by using imprecise paths

### Два типа XML индексов

Индексы XML делятся на две категории:

- Первичный индекс XML. Это – кластеризованное представление В-дерева узлов для данных XML. Первый индекс по столбцу типа xml должен быть первичным XML индексом.
- Вторичный индекс XML. Это – некластеризованный индекс первичного индекса XML. А первичный индекс XML должен существовать прежде, чем будет создан любой вторичный индекс. Существует три типа вторичных индексов XML:
  - индексы Path. Улучшают производительность запросов, которые используют пути и значения для выборки данных
  - индексы Property. Улучшают производительность запросов, которые используют пути для выборки данных
  - индексы Value. Улучшают производительность запросов, которые используют неточные пути для выборки данных

Первичный индекс XML будет всегда улучшать производительность запроса, если **xml** столбец содержит все что угодно, но не тривиальные документы XML. В зависимости от типов запросов по **xml** столбцам, вторичные индексы XML могут обеспечить дополнительную выгоду в производительности.

## Первичные индексы XML

Чтобы создать первичный индекс XML по **xml** столбцу, нужно использовать команду CREATE PRIMARY XML INDEX, как показано в следующем примере.

```
CREATE PRIMARY XML INDEX PXML_ProductModel_CatalogDescription  
ON Production.ProductModel (CatalogDescription)
```

## Индексы XML типа Path

Вы должны рассмотреть создание вторичного индекса типа Path, если намереваетесь выполнять запросы для получения данных **xml** столбцов по пути и значениям. Например, если Вы хотите выполнить запрос, который проверяет на существование выражение XQuery, такое как `/ItemList/Item [@ProductID = "1"]`.

Следующий пример кода показывает, как создать индекс типа Path, который использует первичный индекс, созданный ранее.

```
CREATE XML INDEX XMLPATH_ProductModel_CatalogDescription  
ON Production.ProductModel(CatalogDescription)  
USING XML INDEX PXML_ProductModel_CatalogDescription  
FOR PATH
```

## Индексы XML типа Property

Вы должны рассмотреть создание вторичного индекса типа Property, если Вы намереваетесь выполнить запросы для получения данных узлов **xml** столбцов по заданному пути. Например, если Вы хотите выполнить запрос, который возвращает значение узла, идентифицированного выражением XQuery, таким как `(/ItemList/Item / ProductID) [1]`.

Следующий пример кода показывает, как создать индекс типа Path, который использует первичный индекс, созданный ранее.

```
CREATE XML INDEX XMLPROPERTY_ProductModel_CatalogDescription
ON Production.ProductModel(CatalogDescription)
USING XML INDEX PXML_ProductModel_CatalogDescription
FOR PROPERTY
```

### **Индексы XML типа Value**

Вы должны рассмотреть создание вторичного индекса типа Value, если Вы намереваетесь выполнить запросы для получения данных **xml** столбцы, определяя неточный путь. Например, если Вы хотите выполнить запрос, который проверяет на существование узла, идентифицированного выражением XQuery, таким как **//Item[@ProductID = "1"]**.

Следующий пример кода показывает, как создать индекс типа Value, который использует первичный индекс, созданный ранее.

```
CREATE XML INDEX XMLVALUE_ProductModel_CatalogDescription
ON Production.ProductModel(CatalogDescription)
USING XML INDEX PXML_ProductModel_CatalogDescription
FOR VALUE
```