База данных

Создание базы данных

В различных СУБД процедура создания *баз данных* обычно закрепляется только за администратором *баз данных*. В однопользовательских системах принимаемая по умолчанию *база данных* может быть сформирована непосредственно в процессе установки и настройки самой СУБД. Стандарт SQL не определяет, как должны создаваться *базы данных*, поэтому в каждом из диалектов языка SQL обычно используется свой подход. В соответствии со стандартом SQL, *таблицы* и другие объекты *базы данных* существуют в некоторой среде. Помимо всего прочего, каждая среда состоит из одного или более *каталогов*, а каждый *каталог* – из набора *схем*. *Схема* представляет собой поименованную коллекцию объектов *базы данных*, некоторым образом связанных друг с другом (все объекты в *базе данных* должны быть описаны в той или иной *схеме*). Объектами *схемы* могут быть *таблицы*, представления, домены, утверждения, сопоставления, толкования и наборы символов. Все они имеют одного и того же владельца и множество общих значений, принимаемых по умолчанию.

Стандарт SQL оставляет за разработчиками СУБД право выбора конкретного механизма создания и уничтожения *каталогов*, однако механизм создания и удаления *схем* регламентируется посредством операторов `CREATE SCHEMA` и `DROP SCHEMA` . В стандарте также указано, что в рамках оператора создания *схемы* должна существовать возможность определения диапазона привилегий, доступных пользователям создаваемой *схемы*. Однако конкретные способы определения подобных привилегий в разных СУБД различаются.

В настоящее время операторы `CREATE SCHEMA` и `DROP SCHEMA` реализованы в очень немногих СУБД. В других реализациях, например, в СУБД MS SQL Server, используется оператор `CREATE DATABASE` .

Создание базы данных в среде MS SQL Server

Процесс создания *базы данных* в системе SQL-сервера состоит из двух этапов: сначала организуется сама *база данных*, а затем принадлежащий ей *журнал транзакций*. Информация размещается в соответствующих файлах, имеющих расширения `*.mdf` (для *базы данных*) и `*.ldf` . (для *журнала транзакций*). В файле *базы данных* записываются сведения об основных объектах (*таблицах*, *индексах*, представлениях и т.д.), а в файле *журнала транзакций* – о процессе работы с транзакциями (контроль целостности данных, состояния *базы данных* до и после выполнения транзакций).

Создание *базы данных* в системе SQL-сервер осуществляется командой `CREATE DATABASE` . Следует отметить, что процедура создания *базы данных* в SQL-сервере требует наличия прав администратора сервера.

```
<определение_базы_данных> ::=
   CREATE DATABASE имя_базы_данных
   [ON [PRIMARY]
   [ <определение_файла>  [,...n] ]
   [,<определение_группы> [,...n] ] ]
   [ LOG ON {<определение_файла>[,...n] } ]
   [ FOR LOAD | FOR ATTACH ]
```

Рассмотрим основные параметры представленного оператора.

При выборе имени *базы данных* следует руководствоваться общими правилами именования объектов. Если имя *базы данных* содержит пробелы или любые другие недопустимые символы, оно заключается в ограничители (двойные кавычки или квадратные скобки). Имя *базы данных* должно быть уникальным в пределах сервера и не может превышать 128 символов.

При создании и изменении *базы данных* можно указать имя файла, который будет для нее создан, изменить имя, путь и исходный размер этого файла. Если в процессе использования *базы данных* планируется ее размещение на нескольких дисках, то можно создать так называемые *вторичные файлы базы данных* с расширением `*.ndf` . В этом случае основная информация о *базе данных* располагается в *первичном* (`PRIMARY`) файле, а при нехватке для него свободного места добавляемая информация будет размещаться во *вторичном файле*. Подход, используемый в SQL-сервере, позволяет распределять содержимое *базы данных* по нескольким дисковым томам.

Параметр `ON` определяет список файлов на диске для размещения информации, хранящейся в *базе данных*.

Параметр `PRIMARY` определяет *первичный файл*. Если он опущен, то *первичным* является первый файл в списке.

Параметр `LOG ON` определяет список файлов на диске для размещения *журнала транзакций*. Имя файла для *журнала транзакций* генерируется на основе имени *базы данных*, и в конце к нему добавляются символы `_log` .

При создании *базы данных* можно определить набор файлов, из которых она будет состоять. Файл определяется с помощью следующей конструкции:

```
<определение_файла>::=
   ([ NAME=логическое_имя_файла,]
   FILENAME='физическое_имя_файла'
   [,SIZE=размер_файла ]
   [,MAXSIZE={max_размер_файла |UNLIMITED } ]
   [, FILEGROWTH=величина_прироста ] )[,...n]
```

Здесь ***логическое имя файла*** – это имя файла, под которым он будет опознаваться при выполнении различных SQL-команд.

Физическое имя файла предназначено для указания полного пути и названия соответствующего физического файла, который будет создан на жестком диске. Это имя останется за файлом на уровне операционной системы.

Параметр `SIZE` определяет первоначальный размер файла; минимальный размер параметра – 512 Кб, если он не указан, по умолчанию принимается 1 Мб.

Параметр `MAXSIZE` определяет максимальный размер файла *базы данных*. При значении параметра `UNLIMITED` максимальный размер *базы данных* ограничивается свободным местом на диске.

При создании *базы данных* можно разрешить или запретить автоматический рост ее размера (это определяется параметром `FILEGROWTH`) и указать приращение с помощью абсолютной величины в Мб или процентным соотношением. Значение может быть указано в килобайтах, мегабайтах, гигабайтах, терабайтах или процентах (%). Если указано число без суффикса МБ, КБ или %, то по умолчанию используется значение МБ. Если *размер шага* роста указан в процентах (%), размер увеличивается на заданную часть в процентах от размера файла. Указанный размер округляется до ближайших 64 КБ.

Дополнительные файлы могут быть включены в группу:

```
<определение_группы>::=FILEGROUP имя_группы_файлов
    <определение_файла>[,...n]
```

Пример 3.1. Создать *базу данных*, причем для данных определить три файла на диске С, для *журнала транзакций* – два файла на диске С.

```
CREATE DATABASE Archive
ON PRIMARY ( NAME=Arch1,
  FILENAME=’c:\user\data\archdat1.mdf’,
SIZE=100MB, MAXSIZE=200, FILEGROWTH=20),
(NAME=Arch2,
  FILENAME=’c:\user\data\archdat2.mdf’,
  SIZE=100MB, MAXSIZE=200, FILEGROWTH=20),
(NAME=Arch3,
  FILENAME=’c:\user\data\archdat3.mdf’,
  SIZE=100MB, MAXSIZE=200, FILEGROWTH=20)
LOG ON
(NAME=Archlog1,
  FILENAME=’c:\user\data\archlog1.ldf’,
    SIZE=100MB, MAXSIZE=200, FILEGROWTH=20),
(NAME=Archlog2,
```

```
  FILENAME='c:\user\data\archlog2.ldf',
    SIZE=100MB, MAXSIZE=200, FILEGROWTH=20)
```

Пример 3.1. Создание базы данных.

Изменение базы данных

Большинство действий по изменению конфигурации *базы данных* выполняется с помощью следующей конструкции:

```
<изменение_базы_данных> ::=
  ALTER DATABASE имя_базы_данных
  { ADD FILE <определение_файла>[,...n]
    [TO FILEGROUP имя_группы_файлов ]
  | ADD LOG FILE <определение_файла>[,...n]
  | REMOVE FILE логическое_имя_файла
  | ADD FILEGROUP имя_группы_файлов
  | REMOVE FILEGROUP имя_группы_файлов
  | MODIFY FILE <определение_файла>
  | MODIFY FILEGROUP имя_группы_файлов
  <свойства_группы_файлов>}
```

Как видно из синтаксиса, за один вызов команды может быть изменено не более одного параметра конфигурации *базы данных*. Если необходимо выполнить несколько изменений, придется разбить процесс на ряд отдельных шагов.

В *базу данных* можно добавить (`ADD`) новые файлы данных (в указанную группу файлов или в группу, принятую по умолчанию) или файлы *журнала транзакций*.

Параметры файлов и групп файлов можно изменять (`MODIFY`).

Для удаления из *базы данных* файлов или групп файлов используется параметр `REMOVE` . Однако удаление файла возможно лишь при условии его освобождения от данных. В противном случае сервер не разрешит удаление.

В качестве свойств группы файлов используются следующие:

`READONLY` – группа файлов используется только для чтения; `READWRITE` – в группе файлов разрешаются изменения; `DEFAULT` – указанная группа файлов принимается по умолчанию.

Удаление базы данных

Удаление *базы данных* осуществляется командой:

```
DROP DATABASE имя_базы_данных [,...n]
```

Удаляются все содержащиеся в *базе данных* объекты, а также файлы, в которых она размещается. Для исполнения операции удаления *базы данных* пользователь должен обладать соответствующими правами.

Таблица

Создание таблицы

После создания общей структуры *базы данных* можно приступить к *созданию таблиц*, которые представляют собой отношения, входящие в состав проекта *базы данных*.

Таблица – основной объект для хранения информации в реляционной *базе данных*. Она состоит из содержащих данные *строк* и *столбцов*, занимает в *базе данных* физическое пространство и может быть постоянной или временной.

Поле, также называемое в реляционной *базе данных столбцом*, является частью *таблицы*, за которой закреплен определенный тип данных. Каждая *таблица базы данных* должна содержать хотя бы один *столбец*. *Строка* данных – это запись в *таблице базы данных*, она включает поля, содержащие данные из одной записи *таблицы*.

Приступая к *созданию таблицы*, необходимо иметь ответы на ряд вопросов:

- Как будет называться *таблица*?
- Как будут называться *столбцы* (поля) *таблицы*?
- Какие типы данных будут закреплены за каждым *столбцом*?

Какой размер памяти должен быть выделен для хранения каждого *столбца*?

Какие *столбцы таблицы* требуют обязательного ввода?

Из каких *столбцов* будет состоять первичный ключ?

Базовый синтаксис оператора *создания таблицы* имеет следующий вид:

```
<определение_таблицы> ::=
   CREATE TABLE имя_таблицы
   (имя_столбца тип_данных
     [NULL | NOT NULL ] [,...n])
```

Приведенный стандарт совпадает с реализацией оператора *создания таблицы* в среде MS SQL Server.

Главное в команде *создания таблицы* – определение *имени таблицы* и описание набора имен полей, которые указываются в соответствующем порядке. Кроме того, этой командой оговариваются типы данных и размеры полей *таблицы*.

Ключевое слово `NULL` используется для указания того, что в данном *столбце* могут содержаться значения `NULL` . Значение `NULL` отличается от пробела или нуля – к нему прибегают, когда необходимо указать, что данные недоступны, опущены или недопустимы. Если указано ключевое слово `NOT NULL` , то будут отклонены любые попытки поместить значение `NULL` в данный *столбец*. Если указан параметр `NULL` , помещение значений `NULL` в *столбец* разрешено. По умолчанию стандарт SQL предполагает наличие ключевого слова `NULL` .

Мы использовали упрощенную версию оператора `CREATE TABLE` стандарта SQL. Его полная версия приводится при обсуждении вопросов обеспечения целостности данных.

Пример 3.2. Создать *таблицу* для хранения данных о товарах, поступающих в продажу в некоторой торговой фирме. Необходимо учесть такие сведения, как название и тип товара, его цена, сорт и город, где товар производится.

```
CREATE TABLE Товар
(Название       VARCHAR(50) NOT NULL,
    Цена        MONEY NOT     NULL,
    Тип         VARCHAR(50) NOT NULL,
    Сорт        VARCHAR(50),
    ГородТовара VARCHAR(50))
```

```
Пример 3.2. Создание таблицы для хранения данных о товарах, поступающих в продажу в
некоторой торговой фирме.
```

Пример 3.3.Создать *таблицу* для сохранения сведений о постоянных клиентах с указанием названий города и фирмы, фамилии, имени и отчества клиента, номера его телефона.

```
CREATE TABLE  Клиент
    (Фирма       VARCHAR(50) NOT NULL,
    Фамилия      VARCHAR(50) NOT NULL,
    Имя          VARCHAR(50) NOT NULL,
    Отчество     VARCHAR(50),
    ГородКлиента VARCHAR(50),
    Телефон      CHAR(10) NOT NULL)
```

```
Пример 3.3. Создание таблицы для сохранения сведений о постоянных клиентах.
```

Изменение таблицы

Структура существующей *таблицы* может быть модифицирована с помощью команды `ALTER TABLE` , упрощенный синтаксис которой представлен ниже:

```
ALTER TABLE имя_таблицы
  {[ADD [COLUMN] имя_столбца тип_данных [
    NULL | NOT NULL ]]
  | [DROP [COLUMN] имя_столбца]}
```

В среде MS SQL Server упрощенный синтаксис команды модификации *таблицы* имеет вид:

```
ALTER TABLE имя_таблицы
  {[ALTER COLUMN имя_столбца
  {новый_тип_данных [(точность[,масштаб])]
    [ NULL | NOT NULL ]}]
  | ADD { [имя_столбца тип_данных]
  | имя_столбца AS выражение } [,...n]
  | DROP {COLUMN имя_столбца}[,...n]
  }
```

Команда позволяет добавлять и удалять *столбцы*, изменять их определения.

Одно из основных правил при добавлении *столбцов* в существующую *таблицу* гласит: когда в *таблице* уже содержатся данные, добавляемый *столбец* не может быть определен с атрибутом `NOT NULL` . Этот атрибут означает, что для каждой *строки* данных соответствующий *столбец* должен содержать некоторое значение, поэтому добавление *столбца* с атрибутом `NOT NULL` приводит к появлению противоречия – уже существующие *строки* данных *таблицы* не будут иметь в новом *столбце* ненулевых значений.

Тем не менее существует способ добавления обязательных полей в существующую *таблицу*. Для этого необходимо:

добавить в *таблицу* новый *столбец*, определив его с атрибутом `NULL` (т.е. *столбец* не обязан содержать каких-либо значений);

ввести в новый *столбец* какие-либо значения для каждой *строки* данных *таблицы* ;

убедившись, что новый *столбец* содержит ненулевые значения для каждой *строки данных*, изменить структуру *таблицы*, заменив атрибут этого *столбца* на `NOT NULL` .

При *изменении* определений *столбцов* следует принимать во внимание некоторые общепринятые правила:

размер *столбца* может быть увеличен до максимального значения, допускаемого соответствующим типом данных;

размер *столбца* может быть уменьшен только в том случае, если содержащееся в нем наибольшее значение не будет превосходить его нового размера;

количество разрядов числового типа данных всегда может быть увеличено;

количество разрядов числового типа данных может быть уменьшено только в том случае, если количество разрядов наибольшего значения в соответствующем *столбце* не будет превосходить нового числа разрядов, определенного для этого *столбца* ;

количество десятичных знаков числового типа данных может быть уменьшено или увеличено;

тип данных столбца, как правило, может быть изменен.

Некоторые реализации фактически могут ограничить разработчика в использовании некоторых опций команды `ALTER TABLE` . Например, может оказаться недопустимым удаление *столбцов* из существующей *таблицы*. Чтобы добиться этого, сначала потребуется удалить саму *таблицу* и только потом заново ее построить с нужными *столбцами*. Причем уже внесенные в *таблицу* данные будут потеряны.

Возможны трудности, связанные с удалением из *таблицы столбца*, который зависит от некоторого *столбца* другой *таблицы*. В таком случае сначала придется удалить ограничение *столбца*, а затем сам *столбец*.

Пример 3.4. Добавить в *таблицу* `Клиент` поле для номера расчетного счета.

```
ALTER TABLE Клиент ADD Рас_счет CHAR(20)
```

```
Пример 3.4. Добавление в таблицу Клиент поля для номера расчетного счета.
```

Удаление таблицы

С течением времени структура *базы данных* меняется: создаются новые *таблицы*, а прежние становятся ненужными и удаляются из *базы данных* с помощью оператора:

```
DROP TABLE имя_таблицы [RESTRICT | CASCADE]
```

Следует отметить, что эта команда удалит не только указанную *таблицу*, но и все входящие в нее *строки* данных. Если требуется удалить из *таблицы* лишь данные, сохранив структуру *таблицы*, следует воспользоваться командой `DELETE` .

Оператор `DROP TABLE` дополнительно позволяет указывать, следует ли операцию *удаления* выполнять каскадно. Если в операторе указано ключевое слово `RESTRICT` , то при наличии в *базе данных* хотя бы одного объекта, существование которого зависит от удаляемой *таблицы*, выполнение оператора `DROP`

TABLE будет отменено. Если указано ключевое слово CASCADE , автоматически удаляются и все прочие объекты *базы данных*, чье существование зависит от удаляемой *таблицы*, а также другие объекты, зависящие от удаляемых объектов. Общий эффект от выполнения оператора DROP TABLE с ключевым словом CASCADE может оказаться весьма ощутимым, поэтому подобные операторы следует использовать с максимальной осторожностью.

Чаще всего оператор DROP TABLE используется для исправления ошибок, допущенных при *создании таблицы*. Если *таблица* была создана с некорректной структурой, можно воспользоваться оператором DROP TABLE для ее *удаления*, после чего создать *таблицу* заново.

Индексы

Индексы в стандарте языка

Индексы представляют собой структуру, позволяющую выполнять ускоренный доступ к *строкам таблицы* на основе значений одного или более ее *столбцов* . Наличие *индекса* может существенно повысить скорость выполнения некоторых запросов и сократить время поиска необходимых данных за счет физического или логического их упорядочивания. ***Индекс*** – это набор ссылок, упорядоченных по определенному *столбцу таблицы*, который в данном случае будет называться индексированным *столбцом* . Хотя *индекс* и связан с конкретным *столбцом* (или *столбцами*) *таблицы*, все же он является самостоятельным объектом *базы данных*.

Физически *индекс* – всего лишь упорядоченный набор значений из индексированного *столбца* с указателями на места физического размещения исходных *строк* в структуре *базы данных*. Когда пользователь выполняет обращающийся к индексированному *столбцу* запрос, СУБД автоматически анализирует *индекс* для поиска требуемых значений.

Однако, поскольку *индексы* должны обновляться системой при каждом внесении *изменений* в их базовую *таблицу*, они создают дополнительную нагрузку на систему.

Индексы обычно создаются с целью удовлетворения определенных критериев поиска после того, как *таблица* уже находилась некоторое время в работе и увеличилась в размерах. *Создание индексов* не предусмотрено стандартом SQL, однако большинство диалектов поддерживают как минимум следующий оператор:

```
CREATE [ UNIQUE ] INDEX имя_индекса
  ON имя_таблицы(имя_столбца[ASC|DESC][,...n])
```

Указанные в операторе *столбцы* составляют *ключ индекса*. *Индексы* могут создаваться только для базовых *таблиц*, но не для представлений. Если в операторе указано ключевое слово UNIQUE , уникальность значений *ключа индекса* будет автоматически поддерживаться системой. Требование уникальности значений обязательно для первичных ключей, а также возможно и для других *столбцов таблицы* (например, для альтернативных ключей). Хотя *создание индекса* допускается в любой момент, при его построении для уже заполненной данными *таблицы* могут возникнуть проблемы, связанные с дублированием данных в различных *строках*. Следовательно, *уникальные индексы* (по крайней мере, для первичного ключа) имеет смысл создавать непосредственно при формировании *таблицы*. В результате система сразу возьмет на себя контроль за уникальностью значений данных в соответствующих *столбцах*.

Если созданный *индекс* впоследствии окажется ненужным, его можно удалить с помощью оператора

```
DROP INDEX имя_индекса
```

Индексы в среде MS SQL Server

Индекс представляет собой средство, помогающее ускорить поиск необходимых данных за счет физического или логического их упорядочивания. Индекс представляет собой набор ссылок, упорядоченных по определенному столбцу таблицы, который в данном случае будет называться индексированным столбцом. Индексы - это наборы уникальных значений для некоторой *таблицы* с соответствующими ссылками на данные. Они расположены в самой *таблице* и являются удобным внутренним механизмом системы SQL-сервера, с помощью которого осуществляется доступ к данным оптимальным способом. В среде SQL Server реализованы эффективные алгоритмы поиска нужного значения в строго определенной последовательности данных. Ускорение поиска достигается именно за счет того, что данные представляются упорядоченными (хотя физически, в зависимости от типа *индекса*, они могут храниться в соответствии с очередностью их добавления в *таблицу*). К настоящему времени разработаны эффективные математические алгоритмы поиска данных в упорядоченной последовательности. Наиболее эффективной структурой для поиска данных в машинном представлении являются B-деревья – многоуровневая иерархическая структура с переменным количеством элементов в каждом узле.

Создание индекса

Если выборка данных из *таблицы* требует значительного времени, это означает, что для нее необходимо создать *индекс*. *Индексы* могут существенно повысить производительность выполнения операций поиска и выборки данных. При выборе *столбца* для *индекса* следует проанализировать, какие типы запросов чаще всего выполняются пользователями и какие *столбцы* являются ключевыми, т.е. задающими критерии выборки данных, например, порядок сортировки.

В среде SQL Server реализовано несколько типов *индексов*:

кластерные индексы ;

некластерные индексы ;

уникальные индексы.

Некластерный индекс

Некластерные индексы – наиболее типичные представители семейства *индексов*. В отличие от *кластерных*, они не перестраивают физическую структуру *таблицы*, а лишь организуют ссылки на соответствующие *строки* .

Для идентификации нужной *строки* в *таблице некластерный индекс* организует специальные указатели, включающие в себя:

информацию об идентификационном номере файла, в котором хранится *строка* ;

идентификационный номер страницы соответствующих данных;

номер искомой *строки* на соответствующей странице;

содержимое *столбца*.

В большинстве случаев следует ограничиваться 4-5 *индексами*.

Кластерный индекс

Принципиальным отличием ***кластерного индекса*** от *индексов* других типов является то, что при его определении в *таблице* физическое расположение данных перестраивается в соответствии со структурой *индекса* . Логическая структура *таблицы* в этом случае представляет собой скорее словарь, чем *индекс*. Данные в словаре физически упорядочены, например по алфавиту.

Кластерные индексы могут дать существенное увеличение производительности поиска данных даже по сравнению с обычными *индексами*. Увеличение производительности особенно заметно при работе с последовательными данными. Если в *таблице* определен *некластерный индекс*, то сервер должен сначала обратиться к *индексу*, а затем найти нужную *строку* в *таблице*. При использовании *кластерных индексов* следующая порция данных располагается сразу после найденных ранее данных. Благодаря этому отпадают лишние операции, связанные с обращением к *индексу* и новым поиском нужной *строки* в *таблице*.

Естественно, в *таблице* может быть определен только один *кластерный индекс*. В качестве такового следует выбирать наиболее часто используемые *столбцы*. При этом стоит следовать общим рекомендациям *создания индексов* и не индексировать слишком длинные *столбцы*.

Кластерный индекс может включать несколько *столбцов*. Однако количество таких *столбцов* рекомендуется по возможности свести к минимуму.

Необходимо избегать *создания кластерного индекса* для часто изменяемых *столбцов*, поскольку сервер должен будет выполнять физическое перемещение всех данных в *таблице*, чтобы они находились в упорядоченном состоянии, как того требует *кластерный индекс*. Для интенсивно изменяемых *столбцов* лучше подходит *некластерный индекс*.

При создании в *таблице* первичного ключа (`PRIMARY KEY`) сервер автоматически создает для него *кластерный индекс*, если его не существовало ранее или если при определении ключа не был явно указан другой тип *индекса*.

Когда же в *таблице* определен еще и *некластерный индекс*, то его указатель ссылается не на физическое положение *строки* в *базе данных*, а на соответствующий элемент *кластерного индекса*, описывающего эту *строку*, что позволяет не перестраивать структуру *некластерных индексов* всякий раз, когда *кластерный индекс* меняет физический порядок *строк* в *таблице*.

Уникальный индекс

Уникальность значений в индексируемом *столбце* гарантируют *уникальные индексы*. При их наличии сервер не разрешит вставить новое или изменить существующее значение таким образом, чтобы в результате этой операции в *столбце* появились два одинаковых значения.

Уникальный индекс является своеобразной надстройкой и может быть реализован как для *кластерного*, так и для *некластерного индекса* . В одной *таблице* может существовать один уникальный *кластерный* и множество уникальных *некластерных индексов*.

Уникальные индексы следует определять только тогда, когда это действительно необходимо. Для обеспечения целостности данных в *столбце* можно определить ограничение целостности `UNIQUE` или `PRIMARY KEY`, а не прибегать к *уникальным индексам*. Их использование только для обеспечения целостности данных является неоправданной тратой пространства в *базе данных*. Кроме того, на их поддержание тратится и процессорное время.

Средства языка SQL предлагают несколько способов определения *индекса*:

- автоматическое *создание индекса* при создании первичного ключа;
- автоматическое *создание индекса* при определении ограничения целостности `UNIQUE`;
- *создание индекса* с помощью команды `CREATE INDEX`.

Последняя команда имеет следующий формат:

```
<создание_индекса>::=
  CREATE [ UNIQUE ]
    [ CLUSTERED | NONCLUSTERED ]
  INDEX имя_индекса ON имя_таблицы(имя_столбца
    [ASC|DESC][,...n])
  [WITH [PAD_INDEX]
    [[,] FILLFACTOR=фактор_заполнения]
    [[,] IGNORE_DUP_KEY]
    [[,] DROP_EXISTING]
    [[,] STATISTICS_NORECOMPUTE] ]
    [ON имя_группы_файлов ]
```

Рассмотрим некоторые параметры приведенной команды.

Имя *индекса* должно быть уникальным в пределах *таблицы*, а сам *индекс* создается исключительно для *таблицы* текущей *базы данных*.

Параметр `UNIQUE` используется при необходимости ввода в определенное поле только уникальных значений. При указании этого ключевого слова будет создан *уникальный индекс*. В индексируемом *столбце* желательно запретить хранение значений `NULL`, чтобы избежать проблем, связанных с уникальностью значений. После того как для *столбца* появится *уникальный индекс*, сервер не разрешит выполнение команд `INSERT` и `UPDATE`, которые приведут к появлению дублирующих значений.

Параметр `CLUSTERED` использует возможность физического индексирования данных и позволяет произвести так называемое кластерное индексирование, в результате чего будут отсортированы данные в самой *таблице* согласно порядку этого *индекса*, а вся добавляемая информация станет приводить к изменению физического порядка данных. *Кластерным* может быть только один *индекс* в *таблице*.

Параметр `NONCLUSTERED` позволяет создавать *некластерные индексы*.

Параметр `FILLFACTOR` осуществляет настройку разбиения *индекса* на страницы и заметно оптимизирует работу SQL-сервера. Коэффициент `FILLFACTOR` определяет в процентном соотношении размер создаваемых индексных страниц. При этом имеется обратно пропорциональная зависимость частоты работы с *таблицей* и коэффициента `FILLFACTOR`.

Параметр `PAD_INDEX` определяет заполнение внутреннего пространства *индекса* и применяется совместно с `FILLFACTOR`.

Параметр `DROP_EXISTING` при использовании *кластерного индекса* определяет его повторное создание, что позволяет предотвратить нежелательное обновление *кластерных индексов*.

Параметр `STATISTICS_NORECOMPUTE` определяет функции автоматического обновления статистики для *таблицы*.

Параметр имя_группы_файлов позволяет осуществить выбор файловой группы, в которой будет находиться создаваемый *индекс*. Использование *индекса* из другой файловой группы повышает производительность *некластерных индексов* в связи с параллельностью выполнения процессов ввода/вывода и работы с самим *индексом*.

Удаление индекса

Удаление индекса выполняется командой

```
DROP INDEX 'имя_индекса'[,...n]
```

Пример 3.5. Создать уникальный *кластерный индекс* для *таблицы* `Клиент` по *столбцу* `Фамилия` в первичной группе файлов.

```
CREATE UNIQUE CLUSTERED INDEX index_klient1
  ON Клиент (Фамилия)
  WITH DROP_EXISTING
  ON PRIMARY
```

Пример 3.5. Создание уникального кластерного индекса.

Пример 3.6.Создать уникальный *некластерный индекс* для *таблицы* `Клиент` по *столбцам* `Фамилия` и `Имя` в первичной группе файлов. Кроме того, элементы *индекса* будут упорядочены по убыванию. Также запретим автоматическое обновление статистики при изменении данных в *таблице* и установим фактор заполнения индексных страниц на уровне 30%.

```
CREATE UNIQUE NONCLUSTERED INDEX index_klient2
  ON Клиент (Фамилия DESC,Имя DESC)
  WITH FILLFACTOR=30,
  STATISTICS_NORECOMPUTE
  ON PRIMARY
```

Пример 3.6. Создание уникального некластерного индекса.