

top

КОМПЬЮТЕРНАЯ
АКАДЕМИЯ

Теория Баз Данных

Урок № 2

Таблицы,
типы данных,
запросы, основы
взаимодействия
с MS SQL Server

Содержание

1.	Таблицы	4
	Первичный ключ	6
	Значение по умолчанию	10
	Уникальность	11
2.	Типы данных	18
	Целочисленные типы данных	18
	Типы данных для хранения текста	18
	Вещественные типы данных	20
	Типы данных для хранения даты и времени	21
	Типы данных с фиксированной точкой	22
	Другие типы данных	22
3.	Индекс	24
	Что такое индекс?	24

Цели и задачи индекса	24
Внутреннее устройство индекса	25
4. Системные базы данных и таблицы	28
5. Запросы	30
Введение в язык структурированных запросов SQL.	30
Язык SQL	30
Стандарты языка SQL.	32
Диалекты языка SQL.	32
Диалект Transact-SQL	33
Понятия DDL, DML, DCL	34
6. Домашнее задание	35

1. Таблицы

На прошлом занятии вы изучили порядок создания базы данных и выполнение различных действий по ее настройке. Однако сама по себе база данных особой пользы не принесет, для того чтобы ее можно было использовать в полном объеме необходимо наличие в ней ряда таблиц.

Таблицы в базе данных представляют собой множество данных, объединенных по определенному принципу. Каждая таблица описывает некую сущность при этом в столбцах хранится конкретное свойство сущности, а в строках — совокупность свойств для конкретной сущности.

Что бы было более понятно, разберем это на примере. Допустим, у нас есть таблица **Students**, которая хранит информацию о студентах некоего учебного заведения. В этой таблице столбцы **Last Name**, **First Name**, **Birth Date**, **Grants**, **Email** необходимы для хранения информации о фамилии, имени, дате рождения, стипендии и адресе электронной почты соответственно, а каждая строка в этой таблице содержит информацию о конкретном студенте. Например, студент Joshua Johnson родился 23 мая 1997 года, адрес его электронной почты jo@net.eu, и данные о его стипендии отсутствуют (Рисунок 1.1).

Id	Last Name	First Name	Birth Date	Grants	Email
1	Doe	John	1998-02-11	NULL	jh@net.eu
2	Jones	Jack	1997-11-05	1256.00	jj@net.eu
4	Johnson	Joshua	1997-05-23	NULL	jo@net.eu

Рисунок 1.1. Таблица с информацией о студентах

Теперь, когда предназначение таблиц стало понятным, мы можем приступить к процессу создания собственных таблиц. Для того чтобы создать таблицу в базе данных необходимо раскрыть пункт необходимой БД (в нашем случае **University**) в окне Object Explorer, на папке Tables нажать правой клавишей мыши и в контекстном меню выбрать пункт Table... (Рисунок 1.2).

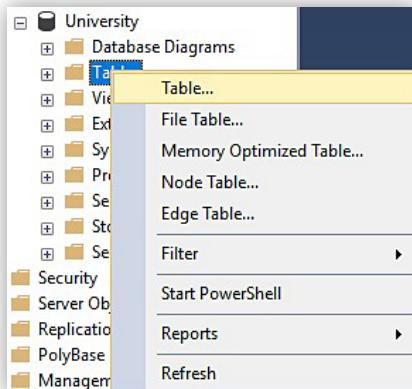


Рисунок 1.2. Создание новой таблицы (начало)

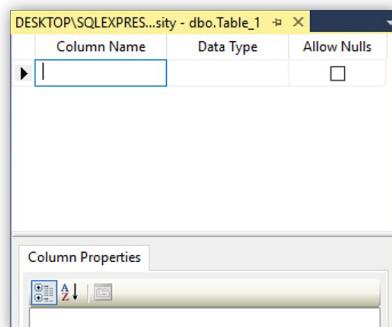


Рисунок 1.3. Создание новой таблицы (продолжение)

В открывшейся вкладке находится таблица, состоящая из трех столбцов **Column Name**, **Data Type** и **Allow Nulls**.

Эта таблица позволяет сформировать новую таблицу в текущей базе данных, путем создания необходимых полей, задав им значения в строках, соответственно, название (**Column Name**), тип данных (**Data Type**) и возможность использования неопределенного (**NULL**) значения (**Allow Nulls**) (Рисунок 1.3).

Первичный ключ

В примере прошлого раздела (Рисунок 1.1) мы нарочно проигнорировали столбец **Id** дело в том, что этот столбец не совсем обычный. В каждой таблице базы данных должно быть поле или совокупность полей, которое имеет общее название — первичный ключ. Значения первичного ключа никогда не должны повторяться, тем самым обеспечивая уникальность любой записи в таблице. В примере на рисунке 1.1 поле **Id** является первичным ключом таблицы **Students**.

UniversityName	FacultyName
Harvard	Medicine
MIT	Architecture
Stanford	Law
Stanford	Medicine
UT Austin	Architecture
UT Austin	Law

Рисунок 1.4. Пример составного первичного ключа

Существует также понятие составного первичного ключа, в этом случае первичный ключ состоит из двух и более полей, сочетание значений в которых должно быть уникальным. Например, в качестве составного первичного ключа можно использовать поля **UniversityName**

и FacultyName (Рисунок 1.4), что естественно, так как названия факультетов в одном университете не могут повторяться, а в различных университетах могут.

Из вышесказанного следует, что в любой таблице необходимо создать поле, которое будет являться первичным ключом. Обычно такое поле называют **Id** или в его названии присутствует слово **Id**, например, **StudId** и также обычно, но не обязательно, тип этого поля должен быть целочисленным (о типах данных будет рассказано далее).

Для того чтобы создать в нашей таблице поле с названием **Id** типа **int**, необходимо в новой строке таблицы в поле **Column Name** ввести соответствующее название, а в поле **Data Type** из выпадающего списка выбрать название требуемого типа данных (Рисунок 1.5).

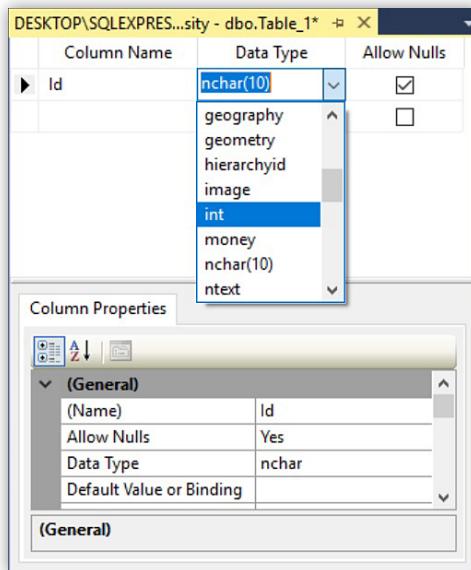


Рисунок 1.5. Выбор требуемого типа данных

Для того чтобы сделать это поле первичным ключом необходимо нажать на него правой кнопкой мыши и в появившемся контекстном меню выбрать пункт **Set Primary Key** (Рисунок 1.6).

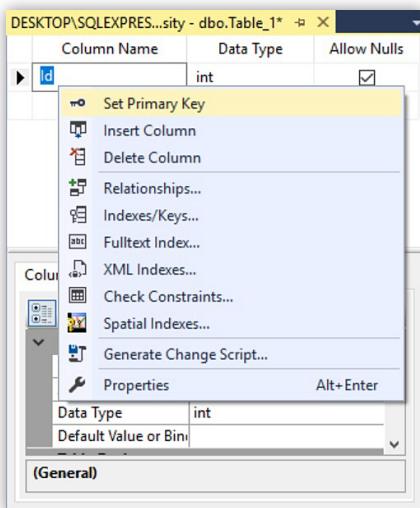


Рисунок 1.6. Создание первичного ключа

В качестве визуального результата вы заметите изображение ключа рядом с выбранным полем, также вы заметите невозможность установки первичному ключу неопределенного значения (сброшено значение в поле **Allow Nulls**).

Существует возможность настроить получение уникального значения первичного ключа путем задания автоматического инкрементирования значений в требуемом поле. Для этого необходимо в таблице **Column Properties** раскрыть свойство **Identity Specification** и в выпадающем списке поля (**Is Identity**) выбрать значение **Yes** (Рисунок 1.7).

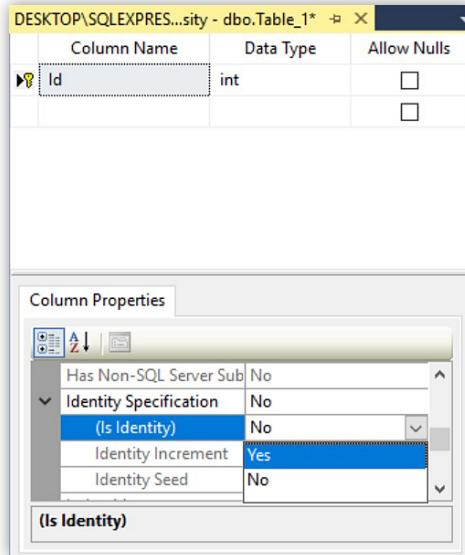


Рисунок 1.7. Настройка уникальности первичного ключа

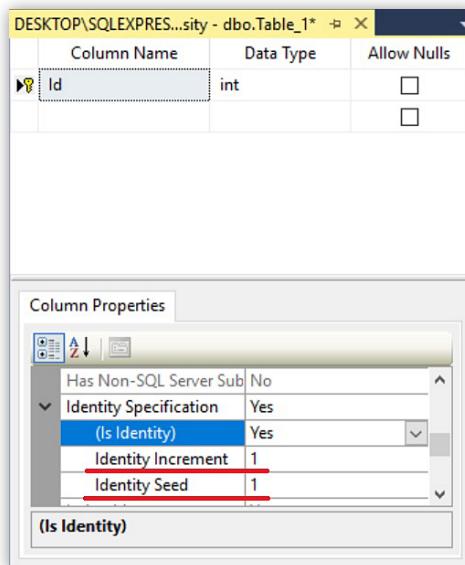


Рисунок 1.8. Настройка автоматического инкрементирования значений

После этого станут доступны поля **Identity Increment** и **Identity Seed**. Изменив значение в поле **Identity Increment**, вы можете задать величину приращения первичного ключа, при помощи поля **Identity Seed** можно задать начальное значение соответствующего поля таблицы, по умолчанию значения в этих полях равны 1 (Рисунок 1.8).

Первичные ключи также используются для создания связей между таблицами, но об этом вы более подробно узнаете в четвертом уроке.

Значение по умолчанию

При создании таблицы существует возможность задать значения по умолчанию для конкретных полей, то есть в том случае если при добавлении новой записи в таблицу полю не будет указано конкретное значение, то будет записано указанное значение по умолчанию.

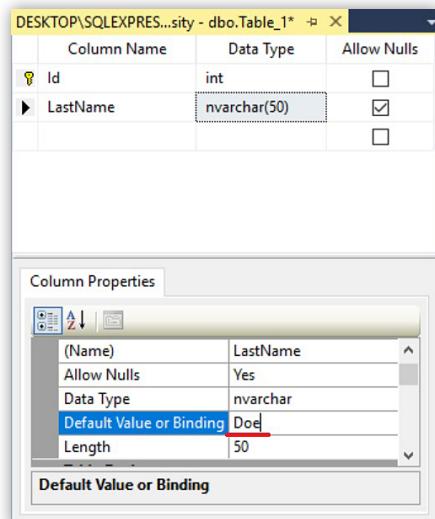


Рисунок 1.9. Настройка значения по умолчанию

Продолжим формирование полей нашей таблицы. Следующее поле имеет название **Last Name**, оно предназначено для хранения фамилии студента и имеет тип данных **nvarchar(50)**.

Для того чтобы задать полю значение по умолчанию необходимо в таблице **Column Properties** задать требуемое значение свойству **Default Value or Binding**, в нашем случае — **Doe** (Рисунок 1.9).

Уникальность

При создании таблицы может возникнуть необходимость в использовании полей с уникальными значениями для каждой записи и **Management Studio** предоставляет такую возможность.

В нашем случае таким полем является поле для хранения электронного адреса студента (**Email**). Для того чтобы объявить это поле уникальным, необходимо нажать по нему правой клавишей мыши и в контекстном меню выбрать пункт **Indexes/Keys...** (Рисунок 1.10).

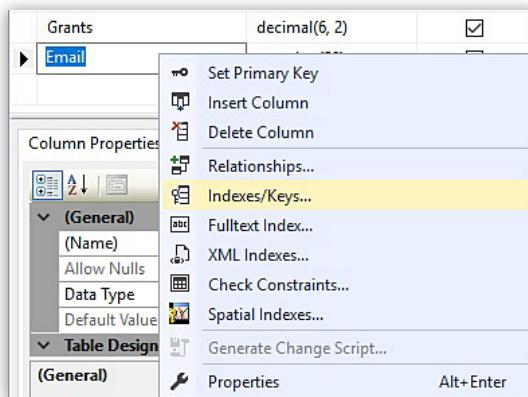


Рисунок 1.10. Создание уникального ключа (начало)

В открывшемся окне **Indexes/Keys** нужно добавить настраиваемый ключ, нажав на кнопку **Add**, при этом система сгенерирует произвольное имя этому ключу, изменить которое можно в поле (**Name**) раздела **Identity** (Рисунок 1.11).

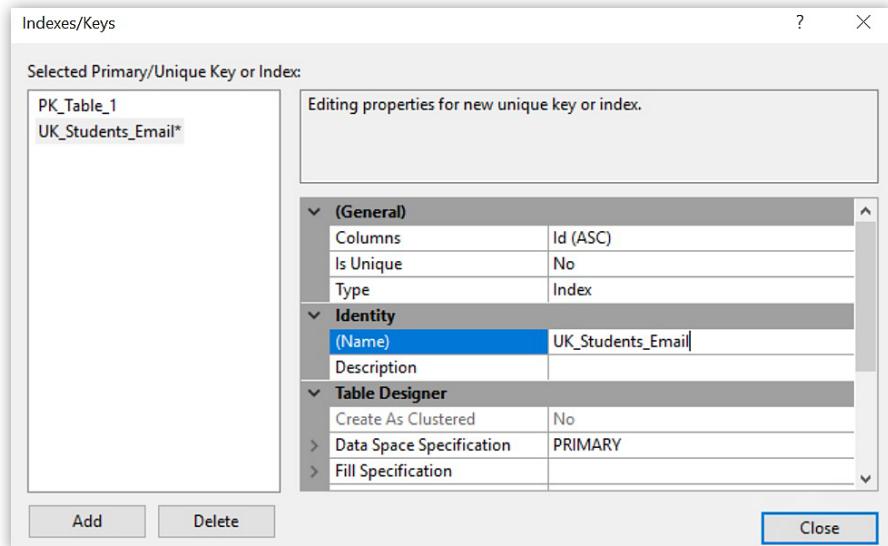


Рисунок 1.11. Изменение имени ключа

Следующее что необходимо сделать — это указать имя настраиваемого столбца в поле **Columns**, для чего нужно в этом поле нажать кнопку с тремя точками и в выпадающем списке появившегося окна **Index Columns** выбрать требуемое название, после чего нажать кнопку **OK** (Рисунок 1.12).

Последним шагом в настройке уникального поля является указание типа ключа в поле **Type** раздела (**General**), в выпадающем списке которого необходимо выбрать значение **Unique Key** (Рисунок 1.13).

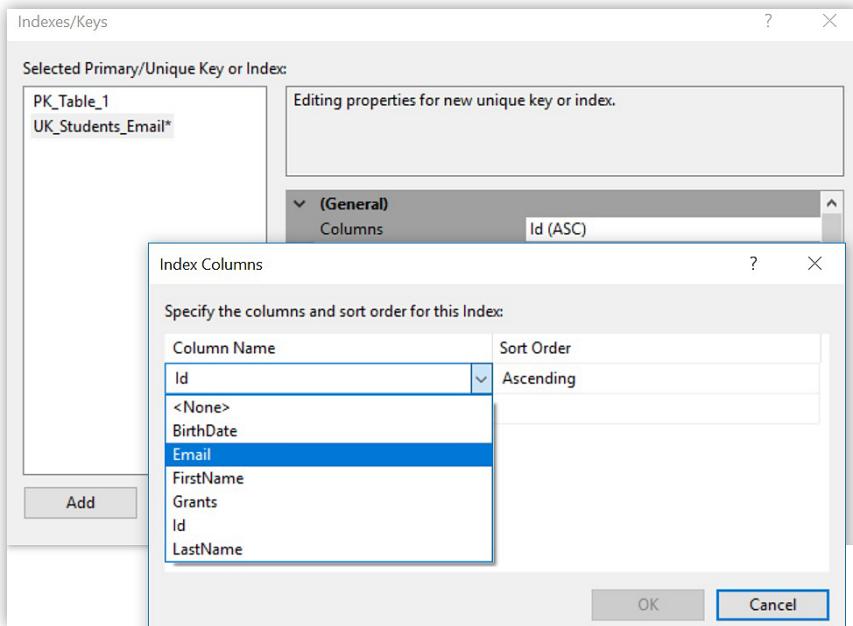


Рисунок 1.12. Выбор настраиваемого столбца

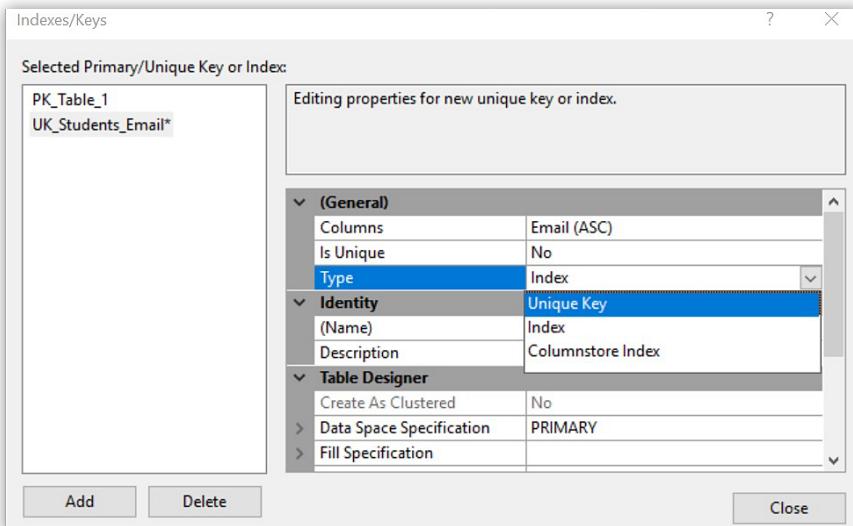


Рисунок 1.13. Указание типа ключа

Для того чтобы настройки уникальности поля вступили в силу нужно закрыть окно **Indexes/Keys**, нажав кнопку **Close**.

После того как вы создали и настроили все поля таблицы имеет смысл изменить имя таблицы, заданное по умолчанию. Это можно сделать в окне свойств текущей таблицы, указав в поле (**Name**) раздела **Identity** требуемое имя таблицы (Рисунок 1.14).

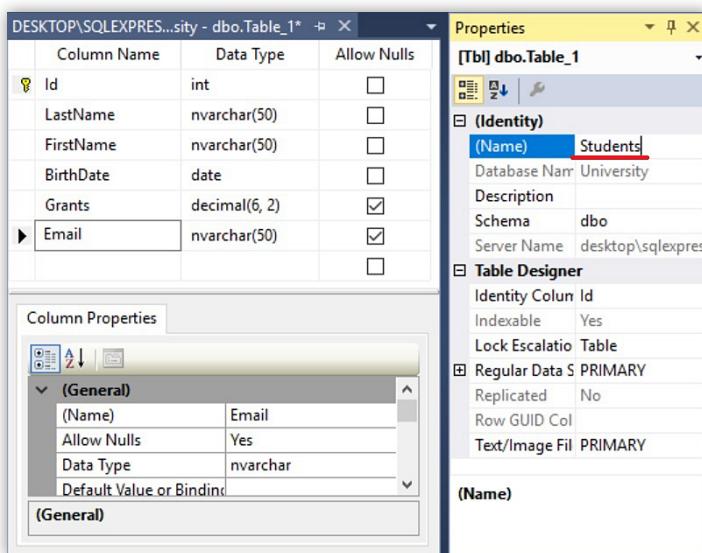


Рисунок 1.14. Переименование таблицы

Сохранение созданной нами таблицы происходит стандартным образом — нажатием кнопки с изображенной дискетой на панели инструментов или сочетанием клавиш **Ctrl+S**. После этого таблица **Students** появится в списке таблиц базы данных **University**.

Для того чтобы убедиться в правильности выполненных нами настроек, заполним таблицу **Students** значениями,

для этого необходимо нажать правой клавишей мыши на имени таблицы и в контекстном меню выбрать пункт **Edit Top 200 Rows** (Рисунок 1.15).

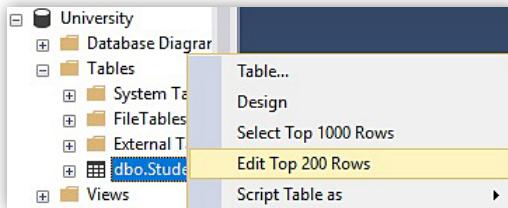


Рисунок 1.15. Заполнение таблицы значениями (начало)

В появившейся вкладке вы увидите таблицу с **NULL**-значениями в каждом поле, заполним их определенными значениями.

Вы можете заметить, что при добавлении в таблицу очередной записи значение в поле **Id** генерируется автоматически, что происходит благодаря настройке автокрементирования первичного ключа.

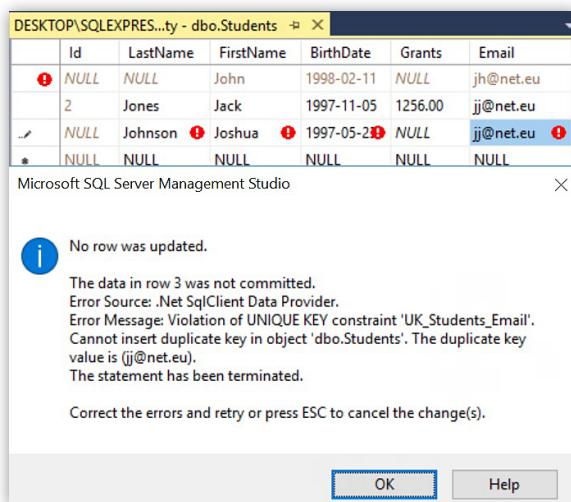


Рисунок 1.16. Ошибка: нарушение уникальности записи

При попытке указать в поле **Email** повторяющееся значение, вы увидите ошибку, в которой сообщается о нарушении уникальности и невозможности дублирования конкретной записи (Рисунок 1.16).

В первой записи мы не указали значение для поля **Last Name** (проверка на значение по умолчанию) и **Management Studio** реагирует на это как на ошибку, правда в описании указывается, что ошибке нет, и эта проблема исчезнет после сохранения таблицы (Рисунок 1.17).

	Id	Last Name	First Name	Birth Date	Grants	Email
1	NULL	NULL	John	1998-02-11	NULL	ih@net.eu

Рисунок 1.17. Ошибка: незаполненное поле

Для того чтобы проверить правильность заполнения таблицы **Students** значениями необходимо нажать правой клавишей мыши на имя таблицы и выбрать пункт **Select Top 1000 Rows** (Рисунок 1.18).

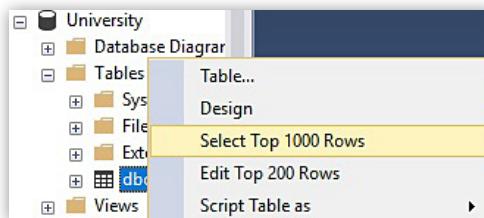


Рисунок 1.18. Выбор записей в таблице (начало)

В открывшейся вкладке вы увидите текст запроса, а в результатах — полученную таблицу с названиями полей и данными в них (Рисунок 1.19).

```
SQLQuery1.sql - DE...ESKTOP\Yuriy (53)  X
/***** Script for SelectTopNRows command from SSMS *****/
SELECT TOP (1000) [Id]
    ,[LastName]
    ,[FirstName]
    ,[BirthDate]
    ,[Grants]
    ,[Email]
FROM [University].[dbo].[Students]
100 % < >
Results Messages


|   | Id | LastName | FirstName | BirthDate  | Grants  | Email     |
|---|----|----------|-----------|------------|---------|-----------|
| 1 | 1  | Doe      | John      | 1998-02-11 | NULL    | jh@net.eu |
| 2 | 2  | Jones    | Jack      | 1997-11-05 | 1256.00 | jj@net.eu |
| 3 | 4  | Johnson  | Joshua    | 1997-05-23 | NULL    | jo@net.eu |


```

Рисунок 1.19. Результат выбора записей таблицы

И хотя текст запроса вам пока непонятен, вы можете убедиться, что все записи заполнились корректно.

2. Типы данных

Как вы уже заметили, для хранения информации в таблицах используются различные типы данных, рассмотрим основные из них.

Целочисленные типы данных

Целочисленные типы данных используются для хранения точных числовых данных, например, возраста человека или количества единиц определенного товара. Ниже представлена таблица целочисленных типов данных с указанием названия, диапазона допустимых значений и количества занимаемой памяти (Таблица 2.1).

Таблица 2.1. Целочисленные типы данных

Название	Диапазон	Память
<code>bigint</code>	от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807	8 байт
<code>int</code>	от -2 147 483 648 до 2 147 483 647	4 байта
<code>smallint</code>	от -32 768 до 32 767	2 байта
<code>tinyint</code>	от 0 до 255	1 байт

Еще одним целочисленным типом данных является `bit`, он занимает один байт памяти и может хранить значения 0, 1 или `NULL`, фактически этот тип данных является аналогом логического типа данных в таких языках программирования как C++ или C#.

Типы данных для хранения текста

Для хранения текстовой информации в базе данных используются четыре основных типа данных:

- **char(n)** — предназначен для хранения строк фиксированной длины не в кодировке Unicode, где вместо **n** указывается возможное количество символов в строке от 1 до 8000, память при этом выделяется по одному байту на символ;
- **varchar(n | max)** — также предназначен для хранения строк не в кодировке Unicode, но переменной длины, допустимое количество символов в строке можно задать либо указав значение от 1 до 8000, либо использовать для этого слово **max** и в этом случае для хранения строки выделится память до 2 ГБ;
- **nchar(n)** — предназначен для хранения строк фиксированной длины в кодировке Unicode, при помощи **n** задается максимально возможное количество символов в строке от 1 до 4000, размер выделенной памяти по два байта на символ;
- **nvarchar(n | max)** — тип данных, который предназначен для хранения строк переменной длины в кодировке Unicode, максимальное количество символов в строке можно задать от 1 до 4000 или выделить память для хранения строки до 2 ГБ, если использовать ключевое слово **max**.

В чем же состоит разница между строкой переменной длины и фиксированной строкой?

Если, например, для хранения строки указан тип данных **char** или **nchar** с размером 10 символов (**char(10)** или **nchar(10)**), а сохраняется строка из шести символов, то количество выделенной памяти будет соответствовать заявленному размеру — 10 или 20 байт, соответственно.

Если для хранения строки указан тип данных `varchar` или `nvarchar` с размером 10 символов (`varchar(10)` или `nvarchar(10)`), а сохраняется строка из шести символов, то размер выделенной памяти будет соответствовать реальному количеству символов — 6 или 12 байт, соответственно.

Вы также можете встретить типы данных `text` и `ntext`, но эти типы данных являются устаревшими и их использование не рекомендовано.

Вещественные типы данных

Для хранения числовых типов данных с плавающей точкой используются вещественные типы данных `float` и `real`.

При помощи типа данных `float` можно хранить числа в диапазоне от $-1,79\text{E}+308$ до $-2,23\text{E}-308$, 0 и от $2,23\text{E}-308$ до $1,79\text{E}+308$.

При использовании типа данных `float(n)`, указывая значение `n`, вы можете устанавливать точность числа и естественно изменять размер выделяемой памяти. Если значение `n` с 1 по 24, то SQL Server трактует его как 24 и выделяет память в 4 байта для хранения числа с точность до 7 знаков. Если в `n` указывается значение с 25 по 53, то SQL Server трактует его как 53 и выделяет 8 байт памяти для хранения числа с точность до 15 знаков. По умолчанию значение `n` равно 53.

Типа данных `real` позволяет хранить числа в диапазоне от $-3,40\text{E}+38$ до $-1,18\text{E}-38$, 0 и от $1,18\text{E}-38$ до $3,40\text{E}+38$.

Синонимом для типа данных `real` является `float(24)`.

Типы данных для хранения даты и времени

Для хранения в таблицах даты и времени SQL Server предоставляет ряд типов данных, которыми вы можете воспользоваться в зависимости от поставленной задачи.

Тип данных **datetime** позволяет хранить дату и время в 24-часовом формате с указанием долей секунды, в диапазоне с 1 января 1753 года по 31 декабря 9999 года, например, [2017-12-28 15:20:35.693](#).

Тип данных **datetime2** позволяет хранить дату и время в 24-часовом формате, но имеет большую точность долей секунды по сравнению с типом **datetime** и диапазон дат с 1 января 0001 года по 31 декабря 9999 года, например, [2017-12-28 15:20:35.6930000](#).

Тип данных **datetimeoffset** позволяет хранить дату и время в 24-часовом формате с указанием долей секунды с учетом часового пояса в диапазоне с 1 января 0001 года по 31 декабря 9999 года, например, [2017-12-28 15:20:35.6930000 +00:00](#).

Предыдущие типы данных выводят довольно точные значения времени, но, если вам достаточно получить только дату и время, например, время покупки товара, тогда вы можете использовать тип данных **smalldatetime**. Этот тип данных позволяет хранить дату и время в 24-часовом формате с секундами, всегда равными нулю, без долей секунды в диапазоне с 1 января 1900 года по 6 июня 2079 года, например, [2017-12-28 15:20:00](#).

Если поле вашей таблицы предназначено для хранения только даты, например, даты рождения человека, то вам больше подойдет тип данных **date**, который позволяет

хранить дату в диапазоне с 1 января 0001 года по 31 декабря 9999 года, например, 2017-12-28.

Тип данных **time** позволяет хранить только время в 24-часовом формате с указанием долей секунды без учета часового пояса в диапазоне от 00:00:00.0000000 до 23:59:59.9999999, например, 15:20:35.6930000.

Типы данных с фиксированной точкой

Для хранения вещественных значений в более точном формате используются типы данных **decimal(p, s)** и **numeric(p, s)**. Эти типы данных являются взаимозаменяемыми и позволяют хранить данные в диапазоне от $-10^{38}+1$ до $10^{38}-1$.

При помощи параметра **p** задается общее количество цифр в числе, как целой, так и дробной части. Диапазон значений от 1 до 38 (по умолчанию 18), чем большее значение вы укажете, тем больше байт памяти будет выделено для хранения информации. Значение в параметре **s** определяет количество цифр дробной части числа в диапазоне от 0 до **p** (по умолчанию 0).

Тип **decimal(6,2)** мы использовали в качестве типа данных для поля **Grants** таблицы **Students** из раздела № 1. В этом случае общее количество цифр в числе равно 6, а цифр после запятой 2, то есть целая часть должна состоять из 4 цифр.

Другие типы данных

Для хранения денежных значений используются типы данных **money** и **smallmoney**, которые представляют собой вещественные числа, дробная часть которых

предназначена для хранения мелочи (копейки, центы и т.д.) (Таблица 2.2).

Таблица 2.2 Денежные типы данных

Название	Диапазон	Память
money	от -922 337 203 685 477,5808 до 922 337 203 685 477,5807	8 байт
smallmoney	от -214 748,3648 до 214 748,3647	4 байта

Для хранения различной информации в бинарном виде используются типы данных **binary(n)** и **varbinary(n | max)**. Параметр **n** может принимать значения от 1 до 8000, если указать ключевое слово **max**, то выделится память для хранения 2 ГБ информации.

Если у вас есть необходимость хранить данные в двумерной системе координат, тогда вы можете использовать тип данных **geometry**.

Для хранения координат широты и долготы объекта предназначен тип данных **geography**.

Хранение XML-документов обеспечивает тип данных **xml**.

Временный результирующий набор данных, полученных в результате запроса, сохраняется в типе данных **table**.

Обычно при создании таблицы полям указываются такие типы данных, максимальный размер которых гарантировано, позволит сохранить требуемый диапазон значений, такой подход используется в целях уменьшения размера базы данных.

3. Индекс

Что такое индекс?

Индекс — это физическая структура данных в БД, при помощи которой осуществляется ускоренный доступ к необходимой информации, использование подходящего индекса значительно улучшает производительность запроса.

Для того чтобы лучше понять, что такое индекс, представим ситуацию, когда вам необходимо найти определенную главу в любой книге. Существуют два способа как это сделать — последовательный перебор всех страниц либо получение номера нужной страницы из оглавления книги. Конечно же, используя второй способ, вы быстрее достигните требуемого результата. Так вот индексы, также как оглавление в книге, позволяют ускорить доступ к требуемой информации в базе данных, если подходящий индекс отсутствует, то поиск осуществляется последовательным перебором всех записей таблицы.

Цели и задачи индекса

К этому моменту вы можете сказать: «Давайте назначим индексы для всех полей таблицы, что может быть проще?» Однако такой подход принесет больше вреда, чем пользы, ведь индекс по сути представляет собой копию данных того поля, для которого он создан, следовательно, при создании индекса увеличивается размер базы данных. Также наличие большого количества индексов

увеличивает время выполнения операций по изменению данных (**INSERT, UPDATE, DELETE**), потому что изменение полей автоматически приводит к изменению индексов, в результате чего происходит фрагментация индекса. Поэтому создавать индексы необходимо с осторожностью и помнить о некоторых особенностях:

- индексы автоматически создаются для уникальных полей таблицы и полей, которые указаны в качестве первичных ключей;
- индексы следует создавать для полей таблицы, по которым часто производится поиск (можно также создавать один индекс для набора столбцов);
- для создания индекса наиболее подходят те поля таблицы, у которых количество повторяющихся значений минимально;
- индексы также можно создавать и для представлений (**VIEWS**), которые будут рассмотрены в курсе MS SQL Server.

Внутреннее устройство индекса

Прежде чем мы опишем внутреннее устройство индекса необходимо разобраться в способе хранения данных в самой базе данных.

Минимальной единицей распределения памяти в базе данных является страницы, размер каждой страницы 8 КБ. Существует два вида страниц — страницы данных и страницы индексов. Восемь страниц образуют экстент, который используется для эффективного управления страницами, размер экстента, соответственно, 64 КБ.

Для хранения индексов используется структура данных в виде сбалансированного дерева В-дерево (*Balanced Tree — B-Tree*). Такая структура представляется в виде дерева, ветви которого направлены вниз, и обеспечивает автоматическую сбалансированность узлов, то есть количество ветвей справа от корневого узла приблизительно равна количеству ветвей слева. Такой способ хранения обеспечивает простой и быстрый способ выборки необходимой информации (Рисунок 3.1).

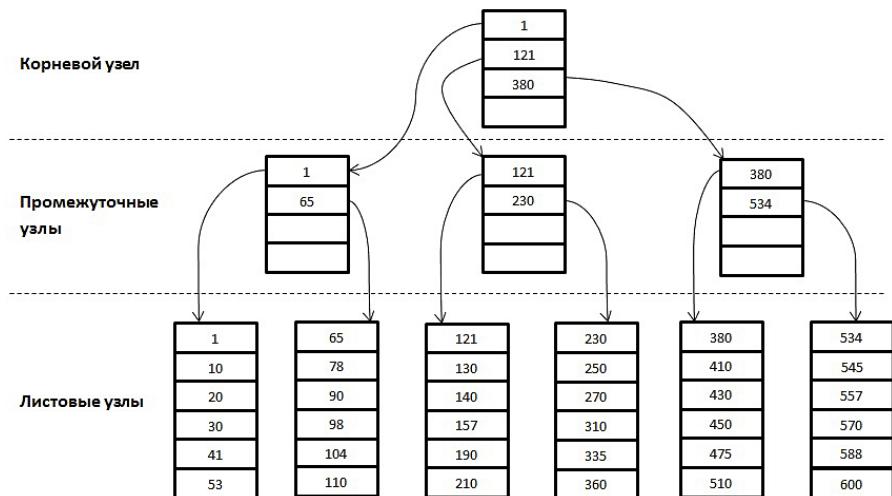


Рисунок 3.1. Пример В-дерева

Индексы бывают кластерные и некластерные.

Кластерный индекс отличается тем, что на его листовом уровне содержатся действительные данные, то есть после достижения листового уровня в таком индексе система получает требуемые данные. В таблице может быть только один кластерный индекс и создается автоматически на поле, которое указано в качестве первичного ключа.

На листовом уровне некластерного индекса находится либо так называемый идентификатор строки (*Row ID* — RID), в котором содержится информация о требуемой записи в виде номера: экстента, страницы и смещении строки на странице, либо кластеризованный ключ, который содержит информацию о кластеризованном индексе. После получения этой информации осуществляется дальнейший поиск данных. Количество некластерных индексов для одной таблицы неограничено.

Использование индексов будет подробно рассмотрено вами в курсе MS SQL Server.

4. Системные базы данных и таблицы

При установке SQL Server 2016 автоматически создаются четыре системных базы данных, наличие которых необходимо для корректного функционирования SQL Server. Вы можете их увидеть, если в окне Object Explorer Management Studio развернете папку Databases, а затем подпапку System Databases (Рисунок 4.1).

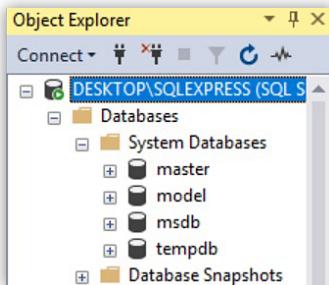


Рисунок 4.1. Системные базы данных

В этой папке находятся следующие системные базы данных:

- **master** — содержит всю информацию о самом SQL Server, а также обо всех базах данных, без этой базы данных запуск SQL Server не возможен;
- **model** — служит шаблоном при создании всех пользовательских баз данных, наличие этой базы данных обязательно;
- **msdb** — эта база данных предназначена для создания расписания заданий, например, создания резервных копий и восстановления любой базы данных;

- **tempdb** — используется для хранения различных временных объектов.

Помимо системных БД также создаются и системные таблицы, которые являются общими для всех баз данных. Системных таблиц довольно много поэтому мы рассмотрим только некоторые из них, за более подробной информацией рекомендуем обратиться к справочной документации:

- **backupfile** — содержит информацию обо всех файлах баз данных на момент создания резервных копий, находится в БД msdb;
- **restorefile** — содержит информацию по каждому восстановленному файлу БД, расположена в базе данных msdb;
- **log_shipping_primary_secondaries** — содержит информацию, которая связывает БД-источник с БД-получателем, находится в базе данных msdb;
- **cdc.lsn_time_mapping** — в этой таблице хранятся данные о всех транзакциях из таблицы изменений;
- **MSdbms** — в этой таблице находится полный список СУБД, отличных от MS SQL Server, которые поддерживают совместимую с MS SQL Server репликацию баз данных, находится в БД msdb;
- **MSreplication_options** — содержит данные, использующиеся при репликациях, расположена в базе данных master;
- **sys.sysoledbusers** — содержит данные по всем пользователям текущего сервера, находится в базе данных master.

5. Запросы

Введение в язык структурированных запросов SQL

Основным предназначением любой базы данных является накопление требуемой информации и представление ее при необходимости. Необходимые данные мы можем получить, запросив их у БД, то есть, написав некоторый код — запрос.

Для того чтобы написать соответствующий запрос нужен специализированный язык программирования. Традиционные языки программирования, существовавшие на момент появления реляционной модели данных, такие как COBOL или Fortran, на эту роль не подошли. Поэтому был разработан новый язык программирования — **SQL** (*Structured Query Language* — язык структурированных запросов).

Язык SQL

Первая версия языка реляционных баз данных появилась в начале 70-х годов прошлого столетия и называлась SEQUEL (*Structured English Query Language* — язык структурированных запросов на основе английского). Этот язык был разработан компанией IBM как часть проекта System/R, направленного на реализацию реляционной СУБД. Через несколько лет была выпущена вторая версия — SEQUEL/2, позже язык был переименован в SQL.

Язык SQL является декларативным языком, при помощи которого вы обращаетесь к базе данных, запрашивая требуемую информацию, то есть вы указываете что вам необходимо, а задачу как это получить решает сама СУБД.

Таким образом, основное предназначение языка SQL — обеспечение взаимодействия с базами данных, путем формирования различного рода запросов, которые состоят из специальных операторов. Операторы — это инструкции, при помощи которых вы запрашиваете информацию из БД, указывая какие данные вас интересуют, откуда их нужно получить и, при необходимости, ограничивая полученный результат. Например, запрос может звучать следующим образом: «Покажите мне фамилии только тех студентов, которые родились в ноябре». В данном случае часть запроса «Покажите мне» указывает, что требуется предоставить, следующая часть запроса «студентов» — информирует откуда взять данные, условие «родились в ноябре» ограничивает количество записей.

В виде SQL-запроса это выглядит следующим образом:

```
SELECT LastName  
FROM Students  
WHERE MONTH(BirthDate) = 11;
```

Не вдаваясь в подробности выполнения запроса, о которых вы узнаете из следующего урока, можно внести некоторую ясность. При помощи оператора **SELECT** вы прописываете **что** вам нужно, в операторе **FROM** вы указываете **откуда** необходимо взять требуемую информацию, а в операторе **WHERE** находится **условие**, которое конкретизирует конечный результат.

Стандарты языка SQL

Несмотря на то, что язык SQL был принят в качестве основного языка по работе с базами данных, существование нескольких производителей СУБД привело к появлению нескольких реализаций (диалектов) языка SQL от разных производителей. Такое положение вещей исключало возможность переноса программного обеспечения с одной СУБД на другую, поэтому стал вопрос о стандартизации языка SQL.

В 1986 году *Американский национальный институт стандартов (ANSI)* разработал первый стандарт языка SQL, который был утвержден *Международной организацией стандартов (ISO)* в 1987 году, получивший название SQL-86. После внесения изменения в оригинальный стандарт в 1989 году вышел следующий стандарт языка — SQL-89.

Последующие расширения языка SQL привели к появлению нескольких стандартов в 1992, 1999, 2003, 2006, 2008, 2011 и 2016 годах. В настоящее время действует, соответственно стандарт SQL-2016.

Диалекты языка SQL

Несмотря на наличие большого количества стандартов в настоящее время не существует единого диалекта языка SQL. Введение производителями СУБД новых функциональных средств вносит в существующие диалекты новые изменения, тем самым диалекты все больше отличаются друг от друга.

Приведем список диалектов языка SQL в СУБД наиболее известных производителей:

- **T-SQL** (*Transact-SQL*) — этот диалект используется в СУБД Microsoft SQL Server и Sybase ASE;
- **PL/SQL** (*Procedural Language/SQL*) — данный диалект используется в СУБД Oracle;
- **SQL/PSM** (*SQL/Persistent Stored Module*) — этот диалект используется в СУБД MySQL;
- **PLpg/SQL** (*Procedural Language/postgreSQL*) — диалект реализован в СУБД PostgreSQL;
- **SQLPL** (*SQL Procedural Language*) — этот диалект реализован в СУБД DB2;
- **Jet SQL** — данный диалект реализован в СУБД Microsoft Access.

Диалект Transact-SQL

Transact-SQL — реализация языка SQL, разработанная корпорацией Microsoft для СУБД Microsoft SQL Server. Знакомиться с этим диалектом вы будете на протяжении текущего и последующего курсов, а пока приведем его краткое описание.

T-SQL позволяет использовать различные операторы: арифметические (+, -, *, /, %), логические (AND, OR, NOT), сравнения (=, >, <, >=, <=, <>) и операторы для работы со множествами (IN). При использовании T-SQL существует возможность создавать переменные при помощи команды **DECLARE**, используя для этого специальные символы — идентификаторы (@). В T-SQL содержится условный оператор (IF) и цикл (WHILE). При написании запросов существует возможность вызова встроенных функций (COUNT, SUM, MIN, MAX, DATEDIFF, ABS).

и т.д.). Для комментирования кода в T-SQL используется либо строчный (`--`), либо блочный (`/**/`) комментарий.

Понятия DDL, DML, DCL

SQL-операторы делятся на три категории: операторы **DDL** (*Data Definition Language* — язык описания данных), **DML** (*Data Manipulation Language* — язык управления данными) и **DCL** (*Data Control Language* — язык управления доступом к данным). Рассмотрим эти операторы более подробно.

Операторы DDL позволяют работать со структурой данных в БД:

- создание объекта (**CREATE**);
- изменение объекта (**ALTER**);
- удаление объекта (**DROP**).

Операторы DML используются при работе с данными в БД:

- запрос определенной информации (**SELECT**);
- вставка необходимых данных в таблицу (**INSERT**);
- обновление существующих данных в таблицы (**UPDATE**);
- удаление данных из таблицы (**DELETE**).

Операторы DCL позволяют управлять доступом к базе данных:

- предоставление доступа пользователя для работы с объектом (**GRANT**);
- запрет на доступ пользователя к объекту (**DENY**);
- отмена привилегий доступа пользователя к объекту (**REVOKE**).

6. Домашнее задание

1. Создайте базу данных, которая содержит таблицу с информацией о книгах. Названия необходимых полей продумайте самостоятельно. Используйте все знания, полученные вами к этому моменту.