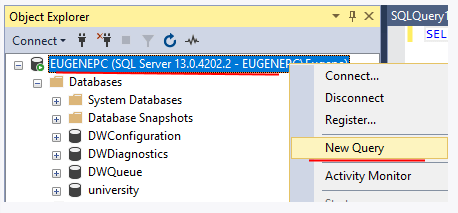
**Основы T-SQL**

**Создание и удаление базы данных**

**Создание базы данных**

Для создания базы данных используется команда **CREATE DATABASE**.

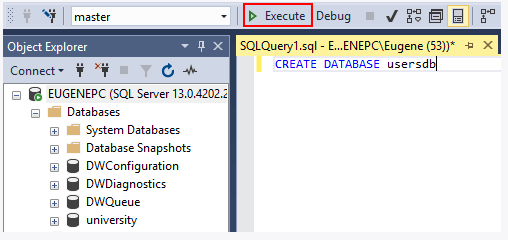
Чтобы создать новую базу данных откроем SQL Server Management Studio. Нажмем на назначение сервера в окне Object Explorer и в появившемся меню выберем пункт **New Query**.



В центральное поле для ввода выражений sql введем следующий код:

|  |  |
| --- | --- |
| 1 | CREATE DATABASE usersdb |

Тем самым мы создаем базу данных, которая будет называться "usersdb":



Для выполнения команды нажмем на панели инструментов на кнопку Execute или на клавишу F5. И на сервере появится новая база данных.

После создания базы даных, мы можем установить ее в качестве текущей с помощью команды **USE**:

|  |  |
| --- | --- |
| 1 | USE usersdb; |

**Прикрепление базы данных**

Возможна ситуация, что у нас уже есть файл базы данных, который, к примеру, создан на другом компьютере. Файл базы данных представляет файл с расширением mdf, и этот файл в принципе мы можем переносить. Однако даже если мы скопируем его компьютер с установленным MS SQL Server, просто так скопированная база данных на сервере не появится. Для этого необходимо выполнить прикрепление базы данных к серверу. В этом случае применяется выражение:

|  |  |
| --- | --- |
| 1  2  3 | CREATE DATABASE название\_базы\_данных  ON PRIMARY(FILENAME='путь\_к\_файлу\_mdf\_на\_локальном\_компьютере')  FOR ATTACH; |

В качестве каталога для базы данных лучше использовать каталог, где хранятся остальные базы данных сервера. На Windows 10 по умолчанию это каталог C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\MSSQL\DATA. Например, пусть в моем случае файл с данными называется userstoredb.mdf. И я хочу этот файл добавить на сервер как базу данных. Вначале его надо скопировать в выше указанный каталог. Затем для прикрепления базы к серверу надо использовать следующую команду:

|  |  |
| --- | --- |
| 1  2  3 | CREATE DATABASE contactsdb  ON PRIMARY(FILENAME='C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\MSSQL\DATA\userstoredb.mdf')  FOR ATTACH; |

После выполнения команды на сервере появится база данных contactsdb.

**Удаление базы данных**

Для удаления базы данных применяется команда **DROP DATABASE**, которая имеет следующий синтаксис:

|  |  |
| --- | --- |
| 1 | DROP DATABASE database\_name1 [, database\_name2]... |

После команды через запятую мы можем перечислить все удаляемые базы данных. Например, удаление базы данных contactsdb:

|  |  |
| --- | --- |
| 1 | DROP DATABASE contactsdb |

Стоит учитывать, что даже если удаляемая база данных была прикреплена, то все равно будут удалены все файлы базы данных.

## Создание и удаление таблиц

Для создания таблиц применяется команда CREATE TABLE. С этой командой можно использовать ряд операторов, которые определяют столбцы таблицы и их атрибуты. И кроме того, можно использовать ряд операторов, которые определяют свойства таблицы в целом. Одна база данных может содержать до 2 миллиардов таблиц.

Общий синтаксис создания таблицы выглядит следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | CREATE TABLE название\_таблицы  (название\_столбца1 тип\_данных атрибуты\_столбца1,   название\_столбца2 тип\_данных атрибуты\_столбца2,   ................................................   название\_столбцаN тип\_данных атрибуты\_столбцаN,   атрибуты\_таблицы  ) |

После команды CREATE TABLE идет название создаваемой таблицы. Имя таблицы выполняет роль ее идентификатора в базе данных, поэтому оно должно быть уникальным. Имя должно иметь длину не больше 128 символов. Имя может состоять из алфавитно-цифровых символов, а также символов $ и знака подчеркивания. Причем первым символом должна быть буква или знак подчеркивания.

Имя объекта не может включать пробелы и не может представлять одно из ключевых слов языка Transact-SQL. Если идентификатор все же содержит пробельные символы, то его следует заключать в кавычки. Если необходимо в качестве имени использовать ключевые слова, то эти слова помещаются в квадратные скобки.

Примеры корректных идентификаторов:

|  |  |
| --- | --- |
| 1  2  3  4  5 | Users  tags$345  users\_accounts  "users accounts"  [Table] |

После имени таблицы в скобках указываются параметры всех столбцов и в самом конце атрибуты, которые относятся ко всей таблице. Атрибуты столбцов и атрибуты таблицы являются необязательными компонентами, и их можно не указывать.

В самом просто виде команда **CREATE TABLE** должна содержать как минимум имя таблицы, имена и типы столбцов.

Таблица может содержать от 1 до 1024 столбцов. Каждый столбец должен иметь уникальное в рамках текущей таблицы имя, и ему должен быть назначен тип данных.

Например, определение простейшей таблицы Customers:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | CREATE TABLE Customers  (      Id INT,      Age INT,      FirstName NVARCHAR(20),      LastName NVARCHAR(20),      Email VARCHAR(30),      Phone VARCHAR(20)  ) |

В данном случае в таблице Customers определяются шесть столбцов: Id, FirstName, LastName, Age, Email, Phone. Первые два столбца представляют идентификатор клиента и его возраст и имеют тип **INT**, то есть будут хранить числовые значения. Следующие два столбца представляют имя и фамилию клиента и имеют тип **NVARCHAR(20)**, то есть представляют строку UNICODE длиной не более 20 символов. Последние два столбца Email и Phone представляют адрес электронной почты и телефон клиента и имеют тип **VARCHAR(30/20)** - они также хранят строку, но не в кодировке UNICODE.

**Создание таблицы в SQL Management Studio**

Создадим простую таблицу на сервере. Для этого откроем SQL Server Management Studio и нажмем правой кнопкой мыши на название сервера. В появившемся контекстном меню выберем пункт **New Query**.

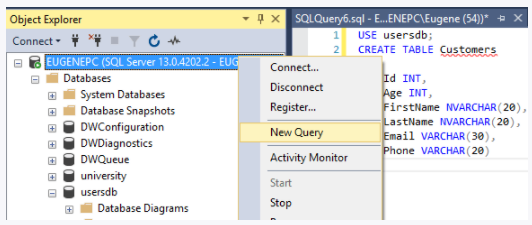
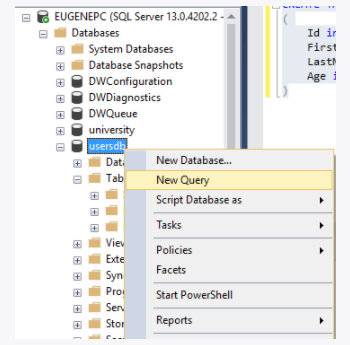


Таблица создается в рамках текущей базы данных. Если мы запускаем окно редактора SQL как это сделано выше – из-под названия сервера, то база данных по умолчанию не установлена. И для ее установки необходимо применить команду **USE**, после которой указывается имя базы данных. Поэтому введем в поле редактора SQL-команд следующие выражения:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11** | **USE usersdb;**    **CREATE TABLE Customers**  **(**  **Id INT,**  **Age INT,**  **FirstName NVARCHAR(20),**  **LastName NVARCHAR(20),**  **Email VARCHAR(30),**  **Phone VARCHAR(20)**  **);** |

То есть в базу данных добавляется таблица Customers, которая была рассмотрена ранее.

Также можно открыть редактор из-под базы данных, также нажав на нее правой кнопкой мыши и выбрав New Query:



В этом случае в качестве текущей будет рассматриваться та база данных, из-под которой был открыт редактор, и дополнительно ее устанавливать с помощью команды USE не потребуется.

**Удаление таблиц**

Для удаления таблиц используется команда **DROP TABLE**, которая имеет следующий синтаксис:

|  |  |
| --- | --- |
| 1 | DROP TABLE table1 [, table2, ...] |

Например, удаление таблицы Customers:

|  |  |
| --- | --- |
| 1 | DROP TABLE Customers |

Переименование таблицы

Для переименования таблиц применяется системная хранимая процедура "sp\_rename". Например, переименование таблицы Users в UserAccounts в базе данных usersdb:

|  |  |
| --- | --- |
| 1  2 | USE usersdb;  EXEC sp\_rename 'Users', 'UserAccounts'; |

**Атрибуты и ограничения столбцов и таблиц**

При создании столбцов в T-SQL мы можем использовать ряд атрибутов, ряд которых являются ограничениями. Рассмотрим эти атрибуты.

PRIMARY KEY

С помощью выражения **PRIMARY KEY** столбец можно сделать первичным ключом.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | CREATE TABLE Customers  (      Id INT PRIMARY KEY,      Age INT,      FirstName NVARCHAR(20),      LastName NVARCHAR(20),      Email VARCHAR(30),      Phone VARCHAR(20)  ) |

Первичный ключ уникально идентифицирует строку в таблице. В качестве первичного ключа необязательно должны выступать столбцы с типом int, они могут представлять любой другой тип.

Установка первичного ключа на уровне таблицы:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | CREATE TABLE Customers  (      Id INT,      Age INT,      FirstName NVARCHAR(20),      LastName NVARCHAR(20),      Email VARCHAR(30),      Phone VARCHAR(20),      PRIMARY KEY(Id)  ) |

Первичный ключ может быть составным (compound key). Такой ключ может потребоваться, если у нас сразу два столбца должны уникально идентифицировать строку в таблице. Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | CREATE TABLE OrderLines  (      OrderId INT,      ProductId INT,      Quantity INT,      Price MONEY,      PRIMARY KEY(OrderId, ProductId)  ) |

Здесь поля OrderId и ProductId вместе выступают как составной первичный ключ. То есть в таблице OrderLines не может быть двух строк, где для обоих из этих полей одновременно были бы одни и те же значения.

IDENTITY

Атрибут **IDENTITY** позволяет сделать столбец идентификатором. Этот атрибут может назначаться для столбцов числовых типов INT, SMALLINT, BIGINT, TYNIINT, DECIMAL и NUMERIC. При добавлении новых данных в таблицу SQL Server будет инкрементировать на единицу значение этого столбца у последней записи. Как правило, в роли идентификатора выступает тот же столбец, который является первичным ключом, хотя в принципе это необязательно.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | CREATE TABLE Customers  (      Id INT PRIMARY KEY IDENTITY,      Age INT,      FirstName NVARCHAR(20),      LastName NVARCHAR(20),      Email VARCHAR(30),      Phone VARCHAR(20)  ) |

Также можно использовать полную форму атрибута:

|  |  |
| --- | --- |
| 1 | IDENTITY(seed, increment) |

Здесь параметр seed указывает на начальное значение, с которого будет начинаться отсчет. А параметр increment определяет, насколько будет увеличиваться следующее значение. По умолчанию атрибут использует следующие значения:

|  |  |
| --- | --- |
| 1 | IDENTITY(1, 1) |

То есть отсчет начинается с 1. А последующие значения увеличиваются на единицу. Но мы можем это поведение переопределить. Например:

|  |  |
| --- | --- |
| 1 | Id INT IDENTITY (2, 3) |

В данном случае отсчет начнется с 2, а значение каждой последующей записи будет увеличиваться на 3. То есть первая строка будет иметь значение 2, вторая - 5, третья - 8 и т.д.

Также следует учитывать, что в таблице только один столбец должен иметь такой атрибут.

**UNIQUE**

Если мы хотим, чтобы столбец имел только уникальные значения, то для него можно определить атрибут **UNIQUE**.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | CREATE TABLE Customers  (      Id INT PRIMARY KEY IDENTITY,      Age INT,      FirstName NVARCHAR(20),      LastName NVARCHAR(20),      Email VARCHAR(30) UNIQUE,      Phone VARCHAR(20) UNIQUE  ) |

В данном случае столбцы, которые представляют электронный адрес и телефон, будут иметь уникальные значения. И мы не сможем добавить в таблицу две строки, у которых значения для этих столбцов будет совпадать.

Также мы можем определить этот атрибут на уровне таблицы:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | CREATE TABLE Customers  (      Id INT PRIMARY KEY IDENTITY,      Age INT,      FirstName NVARCHAR(20),      LastName NVARCHAR(20),      Email VARCHAR(30),      Phone VARCHAR(20),      UNIQUE(Email, Phone)  ) |

**NULL и NOT NULL**

Чтобы указать, может ли столбец принимать значение NULL, при определении столбца ему можно задать атрибут **NULL** или **NOT NULL**. Если этот атрибут явным образом не будет использован, то по умолчанию столбец будет допускать значение NULL. Исключением является тот случай, когда столбец выступает в роли первичного ключа - в этом случае по умолчанию столбец имеет значение NOT NULL.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | CREATE TABLE Customers  (      Id INT PRIMARY KEY IDENTITY,      Age INT,      FirstName NVARCHAR(20) NOT NULL,      LastName NVARCHAR(20) NOT NULL,      Email VARCHAR(30) UNIQUE,      Phone VARCHAR(20) UNIQUE  ) |

**DEFAULT**

Атрибут **DEFAULT** определяет значение по умолчанию для столбца. Если при добавлении данных для столбца не будет предусмотрено значение, то для него будет использоваться значение по умолчанию.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | CREATE TABLE Customers  (      Id INT PRIMARY KEY IDENTITY,      Age INT DEFAULT 18,      FirstName NVARCHAR(20) NOT NULL,      LastName NVARCHAR(20) NOT NULL,      Email VARCHAR(30) UNIQUE,      Phone VARCHAR(20) UNIQUE  ); |

Здесь для столбца Age предусмотрено значение по умолчанию 18.

**CHECK**

Ключевое слово **CHECK** задает ограничение для диапазона значений, которые могут храниться в столбце. Для этого после слова CHECK указывается в скобках условие, которому должен соответствовать столбец или несколько столбцов. Например, возраст клиентов не может быть меньше 0 или больше 100:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | CREATE TABLE Customers  (      Id INT PRIMARY KEY IDENTITY,      Age INT DEFAULT 18 CHECK(Age >0 AND Age < 100),      FirstName NVARCHAR(20) NOT NULL,      LastName NVARCHAR(20) NOT NULL,      Email VARCHAR(30) UNIQUE CHECK(Email !=''),      Phone VARCHAR(20) UNIQUE CHECK(Phone !='')  ); |

Здесь также указывается, что столбцы Email и Phone не могут иметь пустую строку в качестве значения (пустая строка **не** эквивалентна значению NULL).

Для соединения условий используется ключевое слово **AND**. Условия можно задать в виде операций сравнения больше (>), меньше (<), не равно (!=).

Также с помощью CHECK можно создать ограничение в целом для таблицы:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | CREATE TABLE Customers  (      Id INT PRIMARY KEY IDENTITY,      Age INT DEFAULT 18,      FirstName NVARCHAR(20) NOT NULL,      LastName NVARCHAR(20) NOT NULL,      Email VARCHAR(30) UNIQUE,      Phone VARCHAR(20) UNIQUE,      CHECK((Age >0 AND Age<100) AND (Email !='') AND (Phone !=''))  ) |

**Оператор CONSTRAINT. Установка имени ограничений.**

С помощью ключевого слова **CONSTRAINT** можно задать имя для ограничений. В качестве ограничений могут использоваться PRIMARY KEY, UNIQUE, DEFAULT, CHECK.

Имена ограничений можно задать на уровне столбцов. Они указываются после CONSTRAINT перед атрибутами:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | CREATE TABLE Customers  (      Id INT CONSTRAINT PK\_Customer\_Id PRIMARY KEY IDENTITY,      Age INT          CONSTRAINT DF\_Customer\_Age DEFAULT 18          CONSTRAINT CK\_Customer\_Age CHECK(Age >0 AND Age < 100),      FirstName NVARCHAR(20) NOT NULL,      LastName NVARCHAR(20) NOT NULL,      Email VARCHAR(30) CONSTRAINT UQ\_Customer\_Email UNIQUE,      Phone VARCHAR(20) CONSTRAINT UQ\_Customer\_Phone UNIQUE  ) |

Ограничения могут носить произвольные названия, но, как правило, для применяются следующие префиксы:

* "PK\_" - для PRIMARY KEY
* "FK\_" - для FOREIGN KEY
* "CK\_" - для CHECK
* "UQ\_" - для UNIQUE
* "DF\_" - для DEFAULT

В принципе необязательно задавать имена ограничений, при установке соответствующих атрибутов SQL Server автоматически определяет их имена. Но, зная имя ограничения, мы можем к нему обращаться, например, для его удаления.

И также можно задать все имена ограничений через атрибуты таблицы:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | CREATE TABLE Customers  (      Id INT IDENTITY,      Age INT CONSTRAINT DF\_Customer\_Age DEFAULT 18,      FirstName NVARCHAR(20) NOT NULL,      LastName NVARCHAR(20) NOT NULL,      Email VARCHAR(30),      Phone VARCHAR(20),      CONSTRAINT PK\_Customer\_Id PRIMARY KEY (Id),      CONSTRAINT CK\_Customer\_Age CHECK(Age >0 AND Age < 100),      CONSTRAINT UQ\_Customer\_Email UNIQUE (Email),      CONSTRAINT UQ\_Customer\_Phone UNIQUE (Phone)  ) |

**Внешние ключи**

Внешние ключи применяются для установки связи между таблицами. Внешний ключ устанавливается для столбцов из зависимой, подчиненной таблицы, и указывает на один из столбцов из главной таблицы. Хотя, как правило, внешний ключ указывает на первичный ключ из связанной главной таблицы, но это необязательно должно быть непременным условием. Внешний ключ также может указывать на какой-то другой столбец, который имеет уникальное значение.

Общий синтаксис установки внешнего ключа на уровне столбца:

|  |  |
| --- | --- |
| 1  2  3 | [FOREIGN KEY] REFERENCES главная\_таблица (столбец\_главной\_таблицы)      [ON DELETE {CASCADE|NO ACTION}]      [ON UPDATE {CASCADE|NO ACTION}] |

Для создания ограничения внешнего ключа на уровне столбца после ключевого слова **REFERENCES** указывается имя связанной таблицы и в круглых скобках имя связанного столбца, на который будет указывать внешний ключ. Также обычно добавляются ключевые слова **FOREIGN KEY**, но в принципе их необязательно указывать. После выражения REFERENCES идет выражение **ON DELETE** и **ON UPDATE**.

Общий синтаксис установки внешнего ключа на уровне таблицы:

|  |  |
| --- | --- |
| 1  2  3  4 | FOREIGN KEY (стобец1, столбец2, ... столбецN)      REFERENCES главная\_таблица (столбец\_главной\_таблицы1, столбец\_главной\_таблицы2, ... столбец\_главной\_таблицыN)      [ON DELETE {CASCADE|NO ACTION}]      [ON UPDATE {CASCADE|NO ACTION}] |

Например, определим две таблицы и свяжем их посредством внешнего ключа:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | CREATE TABLE Customers  (      Id INT PRIMARY KEY IDENTITY,      Age INT DEFAULT 18,      FirstName NVARCHAR(20) NOT NULL,      LastName NVARCHAR(20) NOT NULL,      Email VARCHAR(30) UNIQUE,      Phone VARCHAR(20) UNIQUE  );    CREATE TABLE Orders  (      Id INT PRIMARY KEY IDENTITY,      CustomerId INT REFERENCES Customers (Id),      CreatedAt Date  ); |

Здесь определены таблицы Customers и Orders. Customers является главной и представляет клиента. Orders является зависимой и представляет заказ, сделанный клиентом. Эта таблица через столбец CustomerId связана с таблицей Customers и ее столбцом Id. То есть столбец CustomerId является внешним ключом, который указывает на столбец Id из таблицы Customers.

Определение внешнего ключа на уровне таблицы выглядело бы следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | CREATE TABLE Orders  (      Id INT PRIMARY KEY IDENTITY,      CustomerId INT,      CreatedAt Date,      FOREIGN KEY (CustomerId)  REFERENCES Customers (Id)  ); |

С помощью оператора **CONSTRAINT** можно задать имя для ограничения внешнего ключа. Обычно это имя начинается с префикса "FK\_":

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | CREATE TABLE Orders  (      Id INT PRIMARY KEY IDENTITY,      CustomerId INT,      CreatedAt Date,      CONSTRAINT FK\_Orders\_To\_Customers FOREIGN KEY (CustomerId)  REFERENCES Customers (Id)  ); |

В данном случае ограничение внешнего ключа CustomerId называется "FK\_Orders\_To\_Customers".

**ON DELETE и ON UPDATE**

С помощью выражений **ON DELETE** и **ON UPDATE** можно установить действия, которые выполняться соответственно при удалении и изменении связанной строки из главной таблицы. И для определения действия мы можем использовать следующие опции:

* **CASCADE**: автоматически удаляет или изменяет строки из зависимой таблицы при удалении или изменении связанных строк в главной таблице.
* **NO ACTION**: предотвращает какие-либо действия в зависимой таблице при удалении или изменении связанных строк в главной таблице. То есть фактически какие-либо действия отсутствуют.
* **SET NULL**: при удалении связанной строки из главной таблицы устанавливает для столбца внешнего ключа значение NULL.
* **SET DEFAULT**: при удалении связанной строки из главной таблицы устанавливает для столбца внешнего ключа значение по умолчанию, которое задается с помощью атрибуты DEFAULT. Если для столбца не задано значение по умолчанию, то в качестве него применяется значение NULL.

**Каскадное удаление**

По умолчанию, если на строку из главной таблицы по внешнему ключу ссылается какая-либо строка из зависимой таблицы, то мы не сможем удалить эту строку из главной таблицы. Вначале нам необходимо будет удалить все связанные строки из зависимой таблицы. И если при удалении строки из главной таблицы необходимо, чтобы были удалены все связанные строки из зависимой таблицы, то применяется каскадное удаление, то есть опция **CASCADE**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | CREATE TABLE Orders  (      Id INT PRIMARY KEY IDENTITY,      CustomerId INT,      CreatedAt Date,      FOREIGN KEY (CustomerId) REFERENCES Customers (Id) ON DELETE CASCADE  ) |

Аналогично работает выражение **ON UPDATE CASCADE**. При изменении значения первичного ключа автоматически изменится значение связанного с ним внешнего ключа. Но так как первичные ключи, как правило, изменяются очень редко, да и с принципе не рекомендуется использовать в качестве первичных ключей столбцы с изменяемыми значениями, то на практике выражение **ON UPDATE** используется редко.

**Установка NULL**

При установки для внешнего ключа опции **SET NULL** необходимо, чтобы столбец внешнего ключа допускал значение NULL:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | CREATE TABLE Orders  (      Id INT PRIMARY KEY IDENTITY,      CustomerId INT,      CreatedAt Date,      FOREIGN KEY (CustomerId) REFERENCES Customers (Id) ON DELETE SET NULL  ); |

Установка значения по умолчанию

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | CREATE TABLE Orders  (      Id INT PRIMARY KEY IDENTITY,      CustomerId INT,      CreatedAt Date,      FOREIGN KEY (CustomerId) REFERENCES Customers (Id) ON DELETE SET DEFAULT  ) |

**Изменение таблицы**

Возможно, в какой-то момент мы захотим изменить уже имеющуюся таблицу. Например, добавить или удалить столбцы, изменить тип столбцов, добавить или удалить ограничения. То есть потребуется изменить определение таблицы. Для изменения таблиц используется выражение **ALTER TABLE**.

Общий формальный синтаксис команды выглядит следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | ALTER TABLE название\_таблицы [WITH CHECK | WITH NOCHECK]  { ADD название\_столбца тип\_данных\_столбца [атрибуты\_столбца] |    DROP COLUMN название\_столбца |    ALTER COLUMN название\_столбца тип\_данных\_столбца [NULL|NOT NULL] |    ADD [CONSTRAINT] определение\_ограничения |    DROP [CONSTRAINT] имя\_ограничения} |

Таким образом, с помощью ALTER TABLE мы можем провернуть самые различные сценарии изменения таблицы. Рассмотрим некоторые из них.

**Добавление нового столбца**

Добавим в таблицу Customers новый столбец Address:

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE Customers  ADD Address NVARCHAR(50) NULL; |

В данном случае столбец Address имеет тип NVARCHAR и для него определен атрибут NULL. Но что если нам надо добавить столбец, который не должен принимать значения NULL? Если в таблице есть данные, то следующая команда не будет выполнена:

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE Customers  ADD Address NVARCHAR(50) NOT NULL; |

Поэтому в данном случае решение состоит в установке значения по умолчанию через атрибут DEFAULT:

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE Customers  ADD Address NVARCHAR(50) NOT NULL DEFAULT 'Неизвестно'; |

В этом случае, если в таблице уже есть данные, то для них для столбца Address будет добавлено значение "Неизвестно".

**Удаление столбца**

Удалим столбец Address из таблицы Customers:

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE Customers  DROP COLUMN Address; |

**Изменение типа столбца**

Изменим в таблице Customers тип данных у столбца FirstName на NVARCHAR(200):

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE Customers  ALTER COLUMN FirstName NVARCHAR(200); |

**Добавление ограничения CHECK**

При добавлении ограничений SQL Server автоматически проверяет имеющиеся данные на соответствие добавляемым ограничениям. Если данные не соответствуют ограничениям, то такие ограничения не будут добавлены. Например, установим для столбца Age в таблице Customers ограничение Age > 21.

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE Customers  ADD CHECK (Age > 21); |

Если в таблице есть строки, в которых в столбце Age есть значения, несоответствующие этому ограничению, то sql-команда завершится с ошибкой. Чтобы избежать подобной проверки на соответствие и все таки добавить ограничение, несмотря на наличие несоответствующих ему данных, используется выражение **WITH NOCHECK**:

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE Customers WITH NOCHECK  ADD CHECK (Age > 21); |

По умолчанию используется значение **WITH CHECK**, которое проверяет на соответствие ограничениям.

**Добавление внешнего ключа**

Пусть изначально в базе данных будут добавлены две таблицы, никак не связанные:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | CREATE TABLE Customers  (      Id INT PRIMARY KEY IDENTITY,      Age INT DEFAULT 18,      FirstName NVARCHAR(20) NOT NULL,      LastName NVARCHAR(20) NOT NULL,      Email VARCHAR(30) UNIQUE,      Phone VARCHAR(20) UNIQUE  );  CREATE TABLE Orders  (      Id INT IDENTITY,      CustomerId INT,      CreatedAt Date  ); |

Добавим ограничение внешнего ключа к столбцу CustomerId таблицы Orders:

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE Orders  ADD FOREIGN KEY(CustomerId) REFERENCES Customers(Id); |

**Добавление первичного ключа**

Используя выше определенную таблицу Orders, добавим к ней первичный ключ для столбца Id:

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE Orders  ADD PRIMARY KEY (Id); |

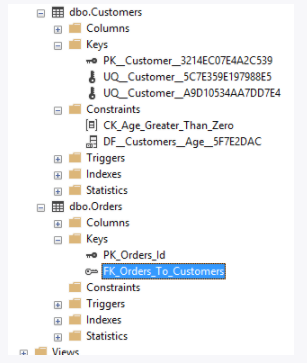
**Добавление ограничений с именами**

При добавлении ограничений мы можем указать для них имя, используя оператор CONSTRAINT, после которого указывается имя ограничения:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | ALTER TABLE Orders  ADD CONSTRAINT PK\_Orders\_Id PRIMARY KEY (Id),      CONSTRAINT FK\_Orders\_To\_Customers FOREIGN KEY(CustomerId) REFERENCES Customers(Id);    ALTER TABLE Customers  ADD CONSTRAINT CK\_Age\_Greater\_Than\_Zero CHECK (Age > 0); |

**Удаление ограничений**

Для удаления ограничений необходимо знать их имя. Если мы точно не знаем имя ограничения, то его можно узнать через SQL Server Management Studio:



Раскрыв узел таблиц в подузле Keys можно увидеть названия ограничений первичного и внешних ключей. Названия ограничений внешних ключей начинаются с "FK". А в подузле Constraints можно найти все ограничения CHECK и DEFAULT. Названия ограничений CHECK начинаются с "CK", а ограничений DEFAULT - с "DF".

Например, как видно на скриншоте в моем случае имя ограничения внешнего ключа в таблице Orders называется "FK\_Orders\_To\_Customers". Поэтому для удаления внешнего ключа я могу использовать следующее выражение:

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE Orders  DROP FK\_Orders\_To\_Customers; |

**Пакеты. Команда GO**

В предыдущих случаях сначала создавалась база данных, а затем в эту БД добавлялась таблица с помощью отдельных команд SQL. Но можно сразу совместить в одном скрипте несколько команд. В этом случае отдельные наборы команд называются **пакетами** (batch).

Каждый пакет состоит из одного или нескольких SQL-выражений, которые выполняются как оно целое. В качестве сигнала завершения пакета и выполнения его выражений служит команда GO.

Смысл разделения SQL-выражений на пакеты состоит в том, что одни выражения должны успешно выполниться до запуска других выражений. Например, при добавлении таблиц мы должны бы уверены, что была создана база данных, в которой мы собираемся создать таблицы.

Например, определим следующий скрипт:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | CREATE DATABASE internetstore;  GO    USE internetstore;    CREATE TABLE Customers  (      Id INT PRIMARY KEY IDENTITY,      Age INT DEFAULT 18,      FirstName NVARCHAR(20) NOT NULL,      LastName NVARCHAR(20) NOT NULL,      Email VARCHAR(30) UNIQUE,      Phone VARCHAR(20) UNIQUE  );    CREATE TABLE Orders  (      Id INT PRIMARY KEY IDENTITY,      CustomerId INT,      CreatedAt DATE,      FOREIGN KEY (CustomerId) REFERENCES Customers (Id) ON DELETE CASCADE  ); |

Вначале создается бд internetstore. Затем идет команда GO, которая сигнализирует, что можно выполнять следующий пакет выражений. И далее выполняется второй пакет, который добавляет в нее две таблицы - Customers и Orders.