MS SQL SERVER

СОЗДАНИЕ БД

СОЗДАНИЕ ТАБЛИЦЫ БД

ТИПЫ ДАННЫХ

ТИПЫ ДАННЫХ

Современные СУБД поддерживают определенный набор встроенных типов данных, используемых для определения полей таблицы.

Для всех типов данных допускается особое NULL-значение, которое по своей сути означает отсутствие значения.

Для каждого типа данных допускается свое NULL-значение, которое отличается от любого не NULL-значения этого же типа. В языке нет возможности представить NULL-значение, но в некоторых случаях для его обозначения используется ключевое слово NULL.

ТИПЫ ДАННЫХ

Базовые встроенные типы данных можно разделить на три группы:

- ***числовые*** типы данных,
- ***строковые*** типы данных,
- типы данных, используемые для ***представления даты и времени.***

ТИПЫ ДАННЫХ. ЧИСЛОВЫЕ ТИПЫ ДАННЫХ

Числовые типы данных используются для представления *точных* и *приближенных* чисел.
Для представления *точных чисел* используются следующие типы данных:

Тип данных	Описание
INT (или INTEGER)	Представляет целочисленные значения длиной в 4 байта в диапазоне от –2 147 483 648 до 2 147 483 647
SMALLINT	Представляет целочисленные значения длиной в 2 байта в диапазоне от -32 768 до 32 767
TINYINT	Представляет целочисленные значения длиной в 1 байт в диапазоне от 0 до 255
BIGINT	Представляет целочисленные значения длиной в 8 байт в диапазоне от -2^{63} до $2^{63} - 1$
DEC(p,[s]) (или DECIMAL(p,[s]))	Представляет значения с фиксированной точкой. Аргумент **p** (*precision* - точность) указывает общее количество разрядов, а аргумент **s** (*scale* - степень) - количество разрядов справа от полагаемой десятичной точки. В зависимости от значения аргумента p, значения **DECIMAL** сохраняются в 5 до 17 байтах.
NUMERIC(p,[s])	Синоним **DECIMAL**.
MONEY	Используется для представления денежных значений, где в записи числа младшие 4 цифры определяют дробную часть денежной единицы (центы, копейки, ... — в зависимости от используемой денежной единицы). Значения типа **MONEY** соответствуют 8-байтовым значениям типа **DECIMAL**, округленным до четырех разрядов после десятичной точки. Эквивалентен типу **DECIMAL(19,4)**.
SMALLMONEY	Представляет такие же значения, что и тип **MONEY**, но длиной в 4 байта
BIT	Целочисленный тип данных, который может принимать значения 1, 0 или NULL. Строковые значения *TRUE* и *FALSE* можно преобразовать в значения типа **bit**: TRUE преобразуется в 1, а FALSE — в 0.

ТИПЫ ДАННЫХ. ЧИСЛОВЫЕ ТИПЫ ДАННЫХ

Для представления *приближенных чисел* используются следующие типы данных.

Тип данных	Описание
FLOAT[(p)]	Представляет значения с плавающей точкой [(p)]. Аргумент **p** определяет точность. При значении *p < 25* представляемые значения имеют одинарную точность (требуют 4 байта для хранения), а при значении *p >= 25* - двойную точность (требуют 8 байтов для хранения). Если значение p не указано, по умолчанию принимается значение 53 (вещественное число двойной точности).
REAL	Применяется для представления значений с плавающей точкой. Диапазон положительных значений простирается приблизительно от 2,23E -308 до -1,18E -38. Также может быть представлено и нулевое значение. Эквивалентно заданию FLOAT(24)

ТИПЫ ДАННЫХ. СИМВОЛЬНЫЕ ТИПЫ ДАННЫХ

Существует два общих вида символьных типов данных.

Строки могут представляться однобайтовыми символами или же символами в кодировке Unicode. (В кодировке Unicode для представления одного символа применяется несколько байтов.) Кроме этого, строки могут быть разной длины.

Тип данных	Описание
CHAR[(n)]	Применяется для представления строк фиксированной длины, состоящих из n однобайтовых символов. Максимальное значение n равно 8000. Если n явно не указано, то его значение полагается равным 1. Количество символов, которое может хранить столбец, передается в скобках. Например, для столбца с типом CHAR(10)будет выделено 10 байт. И если мы сохраним в столбце строку менее 10 символов, то она будет дополнена пробелами.
VARCHAR[(n)]	Используется для представления строки однобайтовых символов переменной длины (0 < n < 8000). В отличие от типа данных CHAR, количество байтов для хранения значений типа данных VARCHAR равно их действительной длине, т.е. если в столбец с типом VARCHAR(10) будет сохранена строка в 5 символов, то в столбце будет сохранено именно пять символов.

ТИПЫ ДАННЫХ. СИМВОЛЬНЫЕ ТИПЫ ДАННЫХ

Тип данных	Описание
NCHAR[(n)]	Используется для хранения строк фиксированной длины, состоящих из символов в кодировке Unicode. Основная разница между типами данных **CHAR** и **NCHAR** состоит в том, что для хранения каждого символа строки типа **NCHAR** требуется 2 байта, а строки типа **CHAR** - 1 байт. Поэтому строка типа данных NCHAR может содержать самое большее 4000 символов. Тип **NCHAR** можно использовать для хранения, например, символов русского алфавита, т.к. однобайтовые кодировки не позволяют делать этого.
NVARCHAR[(n)]	Используется для хранения строк переменной длины, состоящих из символов в кодировке Unicode. Для хранения каждого символа строки типа **NVARCHAR** требуется 2 байта, поэтому строка типа данных **NVARCHAR** может содержать самое большее 4000 символов.

Если нужно представить Unicode-символьную строковую константу, в которой каждый символ отображается в двух байтах, записи константы должен предшествовать символ **N** (в верхнем регистре).

ТИПЫ ДАННЫХ. ПРЕДСТАВЛЕНИЕ ДАТЫ И ВРЕМЕНИ

Тип данных	Описание
DATETIME	Применяется для хранения даты и времени в диапазоне от 01/01/1753 до 31/12/9999. Занимает 8 байт.
SMALLDATETIME	Применяется для хранения даты и времени в диапазоне от 01/01/1900 до 06/06/2079, то есть ближайшие даты. Занимает от 4 байта.
DATE	Применяется для хранения даты. Значения типа **DATE** занимают 3 байта, представляя диапазон дат от 01/01/0001 до 31/12/9999.
TIME	Применяется для хранения времени в диапазоне от 00:00:00.0000000 до 23:59:59.9999999. Занимает от 3 до 5 байт.
DATETIME2	Используется для представления значений дат и времени с высокой точностью. В зависимости от требований, значения этого типа можно определять разной длины, и занимают они от 6 до 8 байтов. Составляющая времени представляет время с точностью до 100 нс. Этот тип данных не поддерживает переход на летнее время.
DATETIMEOFFSET	Определяет дату, объединенную со временем дня, с учетом часового пояса в 24-часовом формате. Диапазон даты с 1 января 1 года нашей эры до 31 декабря 9999 года нашей эры, диапазон времени от 00:00:00 до 23:59:59.9999999

ТИПЫ ДАННЫХ. ПРИМЕР

```
CREATE TABLE Employees
(  ID int, NameEmploy nvarchar(30),
    Birthday datetime,
    Email nvarchar(30),
    Position nvarchar(30),
    Department nvarchar(30) )

INSERT Employees(ID, Position, Department, Birthday)
    VALUES (1000,N'Директор',N'Администрация', '1976-06-18T10:34:09'),
       (1001,N'Программист',N'ИТ', '1992-11-12T18:09:31')
```

СОЗДАНИЕ БАЗЫ ДАННЫХ

Для создания базы данных используется команда

CREATE DATABASE Имя_БД

```
--Создание БД TestDB
CREATE DATABASE TestDB
 ON  PRIMARY --Первичный файл
 (
     NAME = N'TestDB', --Логическое имя файла БД
     FILENAME = N'D:\DataBases\TestDB.mdf' --Имя и местоположение файла БД
 )
 LOG ON --Явно указываем файлы журналов
 (
    NAME = N'TestDB_log', --Логическое имя файла журнала
    FILENAME = N'D:\DataBases\TestDB_log.ldf' --Имя и местоположение файла журнала
 )
 GO
```

УДАЛЕНИЕ БАЗЫ ДАННЫХ

Для удаления базы данных используется команда

```
DROP DATABASE Имя_БД
```

Примечание: командой **DROP** на SQL сервере удаляются все объекты.

СОЗДАНИЕ ТАБЛИЦЫ БАЗЫ ДАННЫХ

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:

```
CREATE TABLE название_таблицы
    (название_столбца1 тип_данных атрибуты_столбца1,
    название_столбца2 тип_данных атрибуты_столбца2,
    ..........................................
    название_столбцаN тип_данных атрибуты_столбцаN,
    атрибуты_таблицы
    )
```

СОЗДАНИЕ ТАБЛИЦЫ БАЗЫ ДАННЫХ.
АВТОИНКРЕМЕНТНЫЕ ПОЛЯ

Identity позволяет указать, что значение данного поля будет сформировано автоматически. Каждая новая строка будет получать число в поле с этим параметром, которое больше любого существующего в уже сформированных строках ранее.

В таблице может быть только одно поле с автоматически увеличиваемым значением. Чтобы задать данный тип поля, необходимо среди параметров указать ключевое слово **IDENTITY** в следующем виде:

```
IDENTITY [(seed, increment) [NOT FOR REPLICATION]]
```

Необязательный параметр `NOT FOR REPLICATION` задается в случае если используются два сервера и объединяются две базы данных одинаковой структуры с них. В этом случае, при вставке новой строки не будет нарушена целостность данных.

```
Id_1 int Identity (100,2) --
Id_2 int Identity -- такое объявление указывает, что поле Id автоинкрементное, начальное
                     значение и увеличение равно 1
```

начальное значения выбрано число 100, последующие: 102,104…

СОЗДАНИЕ ТАБЛИЦЫ БАЗЫ ДАННЫХ. ЗНАЧЕНИЯ ПО УМОЛЧАНИЮ

Бывают случаи, когда необходимо упростить ввод данных со стороны пользователя или просто забивать в какое-либо поле значение, без вмешательства пользователя.
Для задания значения по умолчанию, используется ключевое слово **DEFAULT**, после которого идет нужное значение в том же формате, что и поле. Это значит, что если поле числовое, то значение по умолчанию должно быть числом. Если поле строковое, то значение должно быть строкой заключенной в одинарные кавычки.

```
CREATE TABLE TestDB
   (
         id int DEFAULT 1,
         dDate datetime DEFAULT (getdate()),
         vcName varchar(50) DEFAULT 'M'
   )
```

СОЗДАНИЕ ТАБЛИЦЫ БАЗЫ ДАННЫХ. ОГРАНИЧЕНИЯ ЗНАЧЕНИЙ ПОЛЕЙ

Задача любого администратора и программиста, совместно обеспечить целостность и корректность данных. Для этого используют ограничения – универсальное средство, с помощью которого можно задать правила, которым должны удовлетворять данные, для возможности записи в поле. Если записываемое значение не удовлетворяет ограничениям, назначенным полю, то запись завершается ошибкой. Таким образом, сервер сам будет контролировать целостность данных, вводимых пользователем.

1. Разрешение или запрещение введения нулевых значений (NULL).

```
CREATE TABLE TestDB
   (
         id int DEFAULT 1 NOT NULL,
         dDate datetime DEFAULT (getdate()) NULL,
         vcName varchar(50) NOT NULL
   )
```

СОЗДАНИЕ ТАБЛИЦЫ БАЗЫ ДАННЫХ. ОГРАНИЧЕНИЯ ЗНАЧЕНИЙ ПОЛЕЙ

Ограничения NULL и NOT NULL являются не жесткими и некоторые специалисты даже не относят их к ограничениям, хотя, по своей сути они такими являются.

2. Более жесткие ограничения задаются оператором CHECK.

```
CREATE TABLE TestDB
    (
        id int DEFAULT 1 NOT NULL,
        dDate datetime DEFAULT (getdate()) NULL,
        vcName varchar(50) NOT NULL,
        iApartment int CHECK (iApartment>0 and iApartment<1000)
    )
```

В данном случае задано ограничение целостности данных на уровне поля. Более корректным является задание ограничений на уровне таблицы через оператор **Constraint**:

```
CONSTRAINT имя CHECK (ограничения)
```

СОЗДАНИЕ ТАБЛИЦЫ БАЗЫ ДАННЫХ. ОГРАНИЧЕНИЯ ЗНАЧЕНИЙ ПОЛЕЙ

При создании ограничения, можно использовать многие операторы сравнения языка SQL.

```
CREATE TABLE TestDB
    (
            id int DEFAULT 1 NOT NULL,
            dDate datetime,
            iApartment int,
            cPol nchar(1),
            vc_Phone varchar(20) NOT NULL
            CONSTRAINT ch_iApartment CHECK (iApartment>0 and iApartment<1000),
            CONSTRAINT ch_dDate CHECK (dDate<getdate()),
            CONSTRAINT ch_cPol CHECK (cPol IN ('M', 'Ж'))
            CONSTRAINT ch_vcPhone CHECK (vc_Phone LIKE ' ([0-9][0-9][0-9])
            [0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]')
    )
```

Имена ограничений внутри базы данных должны быть уникальными. Это значит, что нельзя создать два ограничения с одним и тем же именем не только для одной и той же таблицы, но и для разных таблиц одной базы данных.

СОЗДАНИЕ ТАБЛИЦЫ БАЗЫ ДАННЫХ. ОГРАНИЧЕНИЯ ЗНАЧЕНИЙ ПОЛЕЙ

3. Бывает необходимость, чтобы определенное поле или сочетание полей были в базе данных уникальными. Для создания ограничения уникальности используется ограничение **UNIQUE**, которое выглядит следующим образом:

```
CONSTRAINT имя UNIQUE (поле или список полей)
```

```
CREATE TABLE TestDB
    (
            idName int,
            vcName nvarchar(20),
            CONSTRAINT un_id UNIQIE (idName)
    )
```

СОЗДАНИЕ ТАБЛИЦЫ БАЗЫ ДАННЫХ. ПЕРВИЧНЫЙ КЛЮЧ

Только один первичный ключ может быть создан для таблицы.
Для создания первичного ключа используется оператор **PRIMARY KEY**

```
CREATE TABLE TestDB
    (
            idName int IDENTITY(1,1) PRIMARY KEY,
            vcName nvarchar(20)
    )
```

```
CREATE TABLE TestDB
    (
            idName int IDENTITY(1,1),
            vcName nvarchar(20),
            CONSTRAINT pk_ID PRIMARY KEY (idName,vcName)
        )
```

СОЗДАНИЕ ТАБЛИЦЫ БАЗЫ ДАННЫХ. ВНЕШНИЙ КЛЮЧ

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. Внешний ключ также может указывать на какой-то другой столбец, который имеет уникальное значение.
Общий синтаксис установки внешнего ключа на уровне столбца:

```
[FOREIGN KEY] REFERENCES главная_таблица (столбец_главной_таблицы)
  [ON DELETE {CASCADE|NO ACTION}]
  [ON UPDATE {CASCADE|NO ACTION}]
```

Общий синтаксис установки внешнего ключа на уровне таблицы:

```
FOREIGN KEY (столбец1, столбец2, …, столбецN)
 REFERENCES главная_таблица (столбец_гл_таб1,…, столбец_гл_табN)
  [ON DELETE {CASCADE|NO ACTION}]
  [ON UPDATE {CASCADE|NO ACTION}]
```

СОЗДАНИЕ ТАБЛИЦЫ БАЗЫ ДАННЫХ. ПРИМЕР

```
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
)
```

СОЗДАНИЕ ТАБЛИЦЫ БАЗЫ ДАННЫХ. ВНЕШНИЙ КЛЮЧ

С помощью оператора **CONSTRAINT** можно задать имя для ограничения внешнего ключа. Обычно это имя начинается с префикса "FK_":
Общий синтаксис установки внешнего ключа на уровне столбца:

```
CREATE TABLE Orders
(
    Id INT PRIMARY KEY IDENTITY,
    CustomerId INT,
    CreatedAt Date,
    CONSTRAINT FK_Orders_To_Customers FOREIGN KEY
(CustomerId)  REFERENCES Customers (Id)
)
```

СОЗДАНИЕ ТАБЛИЦЫ БАЗЫ ДАННЫХ. 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 можно 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.

СОЗДАНИЕ ТАБЛИЦЫ БАЗЫ ДАННЫХ. ON DELETE И ON UPDATE

По умолчанию, если на строку из главной таблицы по внешнему ключу ссылается какая-либо строка из зависимой таблицы, то нельзя удалить эту строку из главной таблицы. Вначале 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**:

```
CREATE TABLE Orders
(
    Id INT PRIMARY KEY IDENTITY,
    CustomerId INT,
    CreatedAt Date,
    CONSTRAINT FK_Orders_To_Customers FOREIGN KEY (CustomerId)
REFERENCES Customers (Id) ON DELETE CASCADE
)
```

СОЗДАНИЕ ТАБЛИЦЫ БАЗЫ ДАННЫХ. ИЗМЕНЕНИЕ ТАБЛИЦЫ

Для уже существующей таблицы поля можно изменить определение таблицы.
Для изменения таблиц используется команда **ALTER TABLE**.
Общий формальный синтаксис команды выглядит следующим образом:

```
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}
```

СОЗДАНИЕ ТАБЛИЦЫ БАЗЫ ДАННЫХ. ИЗМЕНЕНИЕ ТАБЛИЦЫ. ПРИМЕРЫ

```
ALTER TABLE Customers
    ADD Address NVARCHAR(50) NOT NULL DEFAULT 'Неизвестно'
```

```
ALTER TABLE Customers
    DROP COLUMN Address
```

```
ALTER TABLE Customers
    ALTER COLUMN FirstName NVARCHAR(200)
```

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

```
ALTER TABLE Customers WITH NOCHECK
ADD CHECK (Age > 21);
```

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

ЗАДАНИЕ

1. Создать базу данных.
2. Создать таблицы. В каждой таблице предусмотреть задание ограничений (CHECK, DEFAULT, UNIQUE).
3. Для зависимых таблиц создать ссылочную целостность.
4. С помощью команды INSERT добавить не менее 5 записей в каждую таблицу.
5. Знать команду модификации таблиц.
6. Сохранить скрипт.