# Задание на практическое занятие №3

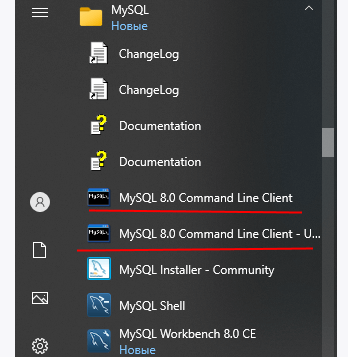
# Основы SQL. Создание базы данных

# Часть 1 Создание БД с использованием SQL-команд

Время – 2 часа

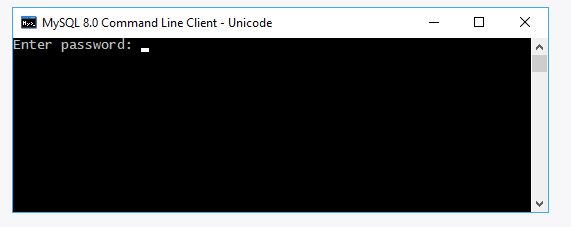
# Консольный клиент MySQL Command Line Client

При установке сервера MySQL также устанавливается консольный клиент для работы с базами данных. Например, в Windows в меню Пуск можно найти программу MySQL 8.0 Command Line Client. Это и есть собственно консольный клиент:



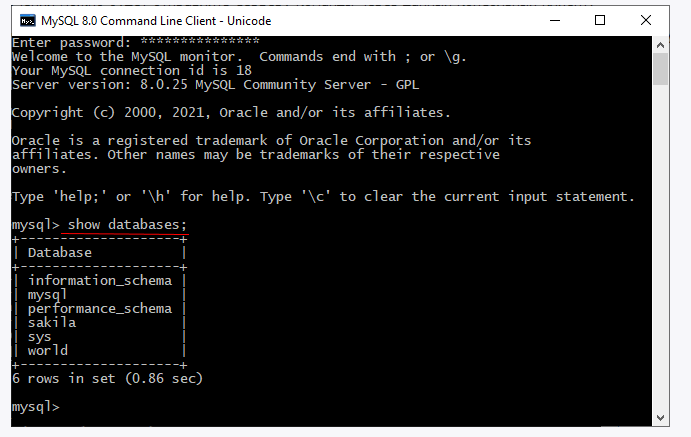
Причем клиент устанавливается сразу в двух вариантах - с поддержкой Unicode и без.

Запустим MySQL 8.0 Command Line Client - Unicode. Вначале нам отобразится предложение ввести пароль:



Здесь необходимо ввести пароль, который был установлен при установке MySQL для пользователя root. И после удачного подключения можно будет отправлять серверу команды через данный консольный клиент.

Для начала посмотрим, какие базы данные есть на сервере по умолчанию. Для этого введем команду show databases;



После выполнения этой команды мы увидим, что на сервере по умолчанию уже есть ряд баз данных, которые выполняют административные функции.

Теперь создадим базу данных с помощью следующей команды языка SQL:



Для создания базы данных применяется команда create database, после которой указывается название бд. То есть в данном случае база данных будет называться "test".

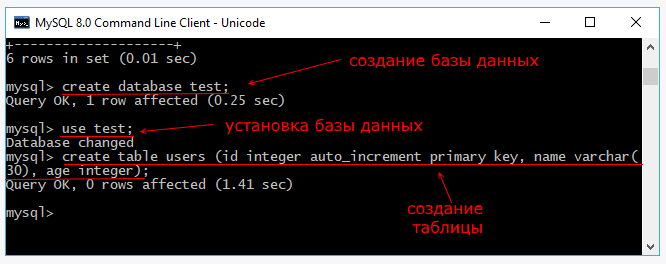
Чтобы обращаться к какой-нибудь определенной базе данных, вначале надо установить нужную базу данных в качестве текущей. Для этого нужно выполнить команду use, после которой указывается название базы данных. Например, для подключения ранее созданной базы данных test введем следующую команду:



Затем создадим в этой базе данных таблицу с помощью команды:



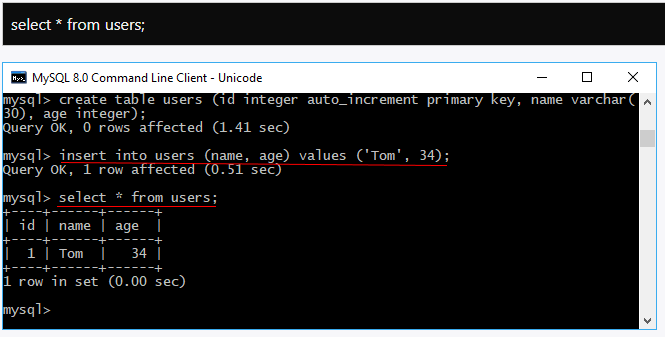
Данная команда создает таблицу users, в которой будет три столбца - id, name и age. id будет хранить уникальный числовой идентификатор пользователя и будет автоматически генерироваться базой данных, name будет хранить имя пользователя, а age - его возраст.



После этого мы можем добавлять и получать данные из выше созданной таблицы. Вначале добавим в таблицу одну строку с помощью следующей команды:



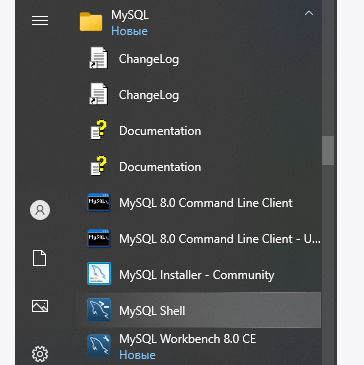
И в конце получим добавленные данные:



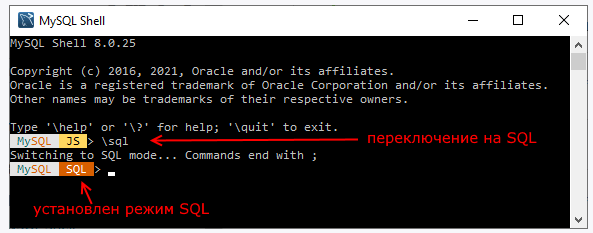
# Консольный клиент MySQL Shell

В последних версиях MySQL также был добавлен еще один консольный клиент - MySQL Shell. Это более современный консольный клиент, который представляет чуть больше функциональности, чем традиционный MySQL Command Line Client.

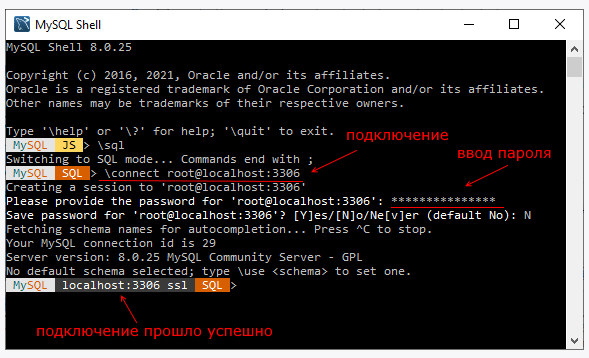
В ОС Windows в меню Пуск в секции MySQL можно найти программу MySQL Shell:



Запустим эту программу. MySQL Shell поддерживает ввод команд на трех языках: JavaScript, Python и SQL. Для установки используемого языка применяются следующие команды: \js, \py и \sql. По умолчанию применяется JavaScript. Но поскольку мы будем использовать SQL, то переключимся на этот язык, введя команду \sql



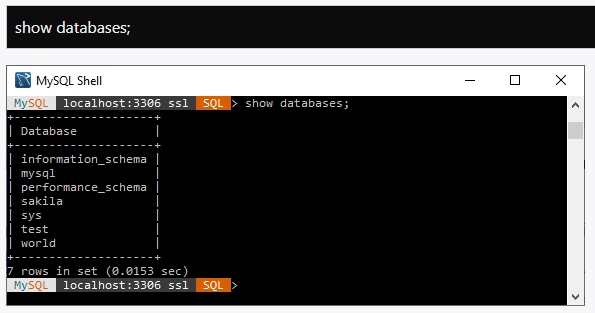
Для взаимодействия с сервером MySQL вначале необходимо подключиться к нему. Для этого применяется команда \connect, после которой указывается идентификатор (uri) в формате имя\_пользователя@хост:порт. Поскольку в большинстве случае используют локальный сервер MySQL, который запущен на порту 3306, а для сервера mySQL доступен как минимум один пользователь - root, то можно использовать для подключения следующий идентификатор: root@localhost:3306. Иначе надо поправить либо имя пользователя, либо адрес, либо порт.



После ввода этой команды программа предложит ввести пароль для пользователя root. И после удачного подключения можно будет отправлять серверу команды через MySQL Shell.

Для примера проделаем все те же вещи, что производились с MySQL Command Line Client.

Для начала посмотрим выведем список баз данных, которые есть на сервере. Для этого введем команду



Теперь создадим базу данных с помощью следующей команды языка SQL:



Для создания базы данных применяется команда create database, после которой указывается название бд. То есть в данном случае база данных будет называться "test2".

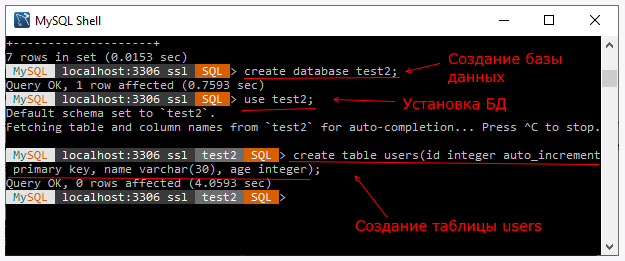
Чтобы обращаться к какой-нибудь определенной базе данных, вначале надо установить нужную базу данных в качестве текущей. Для этого нужно выполнить команду use, после которой указывается название базы данных. Например, для подключения ранее созданной базы данных test введем следующую команду:



Затем создадим в этой базе данных таблицу с помощью команды:



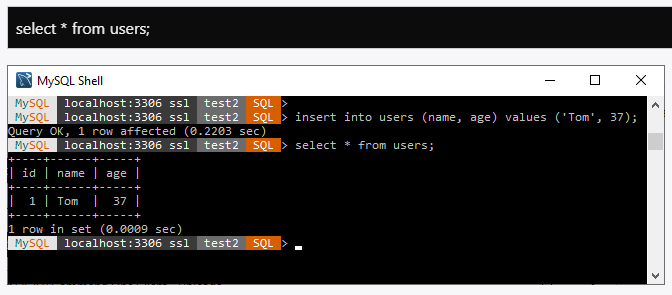
Данная команда создает таблицу users, в которой будет три столбца - id, name и age. id будет хранить уникальный числовой идентификатор пользователя и будет автоматически генерироваться базой данных, name будет хранить имя пользователя, а age - его возраст.



После этого мы можем добавлять и получать данные из выше созданной таблицы. Вначале добавим в таблицу одну строку с помощью следующей команды:



И в конце получим добавленные данные:



Графический клиент MySQL Workbench

MySQL Workbench — это кроссплатформенный инструмент проектирования реляционных баз данных с открытым исходным кодом, который добавляет функциональность и упрощает разработку MySQL и SQL. Он объединяет проектирование, разработку, создание, администрирование и обслуживание SQL, а также предлагает графический интерфейс для структурированной работы с вашими базами данных.

MySQL Workbench предоставляет возможности для управления моделями баз данных, такими как:

* Наглядное представление модели базы данных в графическом виде.
* Наглядный и функциональный механизм установки связей между таблицами, в том числе «многие ко многим» с созданием таблицы связей.
* Reverse Engineering — восстановление структуры таблиц из уже существующей на сервере БД (связи восстанавливаются в [InnoDB](https://ru.wikipedia.org/wiki/InnoDB), при использовании [MyISAM](https://ru.wikipedia.org/wiki/MyISAM) — связи необходимо устанавливать вручную).
* Удобный редактор SQL запросов, позволяющий сразу же отправлять их серверу и получать ответ в виде таблицы.
* Возможность редактирования данных в таблице в визуальном режиме.

# Создание базы данных

В программе MySQL Workbench существует 3 способа создания баз данных (БД).

1. Поэтапное создание БД и её компонентов с использованием   
   SQL-команд с «мгновенным» применением изменений к БД.
2. Создание готовой БД в результате выполнения скрипта, содержащего SQL-команды.
3. Создание БД сразу на основе модели MySQL Workbench.

# Создание БД с использованием SQL-команд

# Создание и удаление базы данных Создание базы данных

Для создания базы данных используется команда CREATE DATABASE. Она имеет следующий синтаксис:

CREATE DATABASE [IF NOT EXISTS] имя\_базы\_даных;

В конце команды указывается имя базы данных.

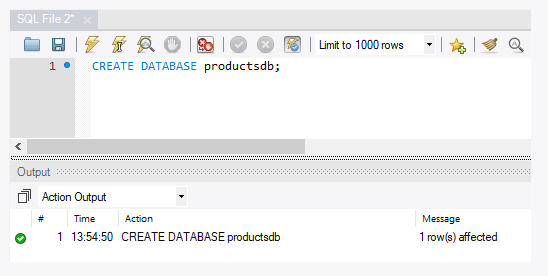
Первая форма CREATE DATABASE имя\_базы\_даных пытается создать базу данных, но если такая база данных уже существует, то операция возвратит ошибку.

Вторая форма CREATE DATABASE IF NOT EXISTS имя\_базы\_даных пытается создать базу данных, если на сервере отсутствует бд с таким именем.

Например, в MySQL Workbench CE (или в MySQL Command Line Client) выполним следующую команду:

CREATE DATABASE productsdb;

Она создаст на сервере БД productsdb.



# Установка базы данных

После создания БД с ней производятся различные операции: создание таблиц, добавление и получение данных и т.д. Но чтобы установить производить эти операции, надо установить определенную базу данных в качестве используемой. Для этого применяется оператор USE:

USE productsdb;

Удаление базы данных

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

DROP DATABASE имя\_базы\_даных;

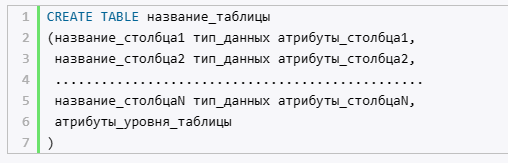
Команда пытается удалить базу данных, но если такая база данных отсутствует на сервере, то операция возвратит ошибку.

Например, удалим выше созданную базу данных productsdb:

DROP DATABASE productsdb;

# Создание таблицы

Для создания таблиц используется команда CREATE TABLE. Эта команды применяет ряд операторов, которые определяют столбцы таблицы и их атрибуты. Общий формальный синтаксис команды CREATE TABLE:



После команды CREATE TABLE идет название таблицы. Имя таблицы выполняет роль ее идентификатора в базе данных, поэтому оно должно быть уникальным. Затем в скобках перечисляются названия столбцов, их типы данных и атрибуты. В самом конце можно определить атрибуты для всей таблицы. Атрибуты столбцов, а также атрибуты таблицы указывать необязательно.

Создадим простейшую таблицу. Для этого выполним следующий скрипт:

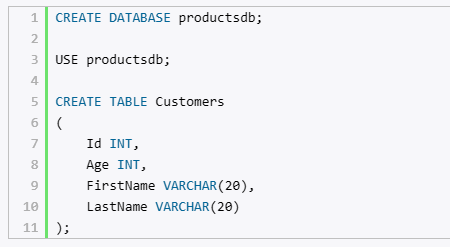
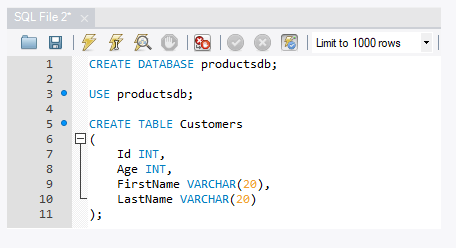


Таблица не может создаваться сама по себе. Она всегда создается в определенной базе данных. Вначале здесь создается база данных productsdb. И затем, чтобы указать, что все дальнейшие операции, в том числе создание таблицы, будут производиться с этой базой данных, применяется команда USE.

Далее собственно идет создание таблицы, которая называется Customers. Она определяет четыре столбца: Id, Age, FirstName, LastName. Первые два столбца представляют идентификатор клиента и его возраст и имеют тип INT, то есть будут хранить числовые значения. Следующие столбцы представляют имя и фамилию клиента и имеют тип VARCHAR(20), то есть представляют строку длиной не более 20 символов. В данном случае для каждого столбца определены имя и тип данных, при этом атрибуты столбцов и таблицы в целом отсутствуют.



И в результате выполнения этой команды будет создана база данных productsdb, в которой будет создана таблица Customers.

Переименование таблиц

Если после создания таблицы мы захотим ее переименовать, то для этого нужно использовать команду RENAME TABLE, которая имеет следующий синтаксис:

RENAME TABLE старое\_название TO новое\_название;

Например, переименуем таблицу Customers в Clients:

RENAME TABLE Customers TO Clients;

Полное удаление данных

Для полного удаления данных, очистки таблицы применяется команда TRUNCATE TABLE. Например, очистим таблицу Clients:

TRUNCATE TABLE Clients;

Удаление таблиц

Для удаления таблицы из БД применяется команда DROP TABLE, после которой указывается название удаляемой таблицы. Например, удалим таблицу Clients:

DROP TABLE Clients;

# Типы данных MySQL

При определении столбцов таблицы для них необходимо указать тип данных. Каждый столбец должен иметь тип данных. Тип данных определяет, какие значения могут храниться в столбце, сколько они будут занимать места в памяти.

MySQL предоставляет следующие типы данных, которые можно разбить на ряд групп.

# Символьные типы

CHAR: представляет строку фиксированной длины.

Длина хранимой строки указывается в скобках, например, CHAR(10) - строка из десяти символов. И если в таблицу в данный столбец сохраняется строка из 6 символов (то есть меньше установленной длины в 10 символов), то строка дополняется 4 пробелами и в итоге все равно будет занимать 10 символов

Тип CHAR может хранить до 255 байт.

VARCHAR: представляет строку переменной длины.

Длина хранимой строки также указыватся в скобках, например, VARCHAR(10). Однако в отличие от CHAR хранимая строка будет занимать именно столько места, сколько необходимо. Например, если определенная длина в 10 символов, но в столбец сохраняется строка в 6 символов, то хранимая строка так и будет занимать 6 символов плюс дополнительный байт, который хранит длину строки.

Всего тип VARCHAR может хранить до 65535 байт.

Начиная с MySQL 5.6 типы CHAR и VARCHAR по умолчанию используют кодировку UTF-8, которая позволяет использовать до 3 байт для хранения символа в зависимости от языка ( для многих европейских языков по 1 байту на символ, для ряда восточно-европейских и ближневосточных - 2 байта, а для китайского, японского, корейского - по 3 байта на символ).

Ряд дополнительных типов данных представляют текст неопределенной длины:

TINYTEXT: представляет текст длиной до 255 байт.

TEXT: представляет текст длиной до 65 КБ.

MEDIUMTEXT: представляет текст длиной до 16 МБ

LARGETEXT: представляет текст длиной до 4 ГБ

# Числовые типы

TINYINT: представляет целые числа от -128 до 127, занимает 1 байт

BOOL: фактически не представляет отдельный тип, а является лишь псевдонимом для типа TINYINT(1) и может хранить два значения 0 и 1. Однако данный тип может также в качестве значения принимать встроенные константы TRUE (представляет число 1) и FALSE (предоставляет число 0).

Также имеет псевдоним BOOLEAN.

TINYINT UNSIGNED: представляет целые числа от 0 до 255, занимает 1 байт

SMALLINT: представляет целые числа от -32768 до 32767, занимает 2 байтa

SMALLINT UNSIGNED: представляет целые числа от 0 до 65535, занимает 2 байтa

MEDIUMINT: представляет целые числа от -8388608 до 8388607, занимает 3 байта

MEDIUMINT UNSIGNED: представляет целые числа от 0 до 16777215, занимает 3 байта

INT: представляет целые числа от -2147483648 до 2147483647, занимает 4 байта

INT UNSIGNED: представляет целые числа от 0 до 4294967295, занимает 4 байта

BIGINT: представляет целые числа от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807, занимает 8 байт

BIGINT UNSIGNED: представляет целые числа от 0 до 18 446 744 073 709 551 615, занимает 8 байт

DECIMAL: хранит числа с фиксированной точностью. Данный тип может принимать два параметра precision и scale: DECIMAL(precision, scale).

Параметр precision представляет максимальное количество цифр, которые может хранить число. Это значение должно находиться в диапазоне от 1 до 65.

Параметр scale представляет максимальное количество цифр, которые может содержать число после запятой. Это значение должно находиться в диапазоне от 0 до значения параметра precision. По умолчанию оно равно 0.

Например, в определении следующего столбца:

salary DECIMAL(5,2)

Число 5 - precision, а число 2 - scale, поэтому данный столбец может хранить значения из диапазона от -999.99 до 999.99.

Размер данных в байтах для DECIMAL зависит от хранимого значения.

Данный тип также имеет псевдонимы NUMERIC, DEC, FIXED.

FLOAT: хранит дробные числа с плавающей точкой одинарной точности от -3.4028 \* 1038 до 3.4028 \* 1038, занимает 4 байта

Может принимать форму FLOAT(M,D), где M - общее количество цифр, а D - количество цифр после запятой

DOUBLE: хранит дробные числа с плавающей точкой двойной точности от -1.7976 \* 10308 до 1.7976 \* 10308, занимает 8 байт. Также может принимать форму DOUBLE(M,D), где M - общее количество цифр, а D - количество цифр после запятой.

Данный тип также имеет псевдонимы REAL и DOUBLE PRECISION, которые можно использовать вместо DOUBLE.

# Типы для работы с датой и временем

DATE: хранит даты с 1 января 1000 года до 31 деабря 9999 года (c "1000-01-01" до "9999-12-31"). По умолчанию для хранения используется формат yyyy-mm-dd. Занимает 3 байта.

TIME: хранит время от -838:59:59 до 838:59:59. По умолчанию для хранения времени применяется формат "hh:mm:ss". Занимает 3 байта.

DATETIME: объединяет время и дату, диапазон дат и времени - с 1 января 1000 года по 31 декабря 9999 года (с "1000-01-01 00:00:00" до "9999-12-31 23:59:59"). Для хранения по умолчанию используется формат "yyyy-mm-dd hh:mm:ss". Занимает 8 байт

TIMESTAMP: также хранит дату и время, но в другом диапазоне: от "1970-01-01 00:00:01" UTC до "2038-01-19 03:14:07" UTC. Занимает 4 байта

YEAR: хранит год в виде 4 цифр. Диапазон доступных значений от 1901 до 2155. Занимает 1 байт.

Тип Date может принимать даты в различных форматах, однако непосредственно для хранения в самой бд даты приводятся к формату "yyyy-mm-dd". Некоторые из принимаемых форматов:

yyyy-mm-dd - 2018-05-25

yyyy-m-dd - 2018-5-25

yy-m-dd - 18-05-25

В таком формате двузначные числа от 00 до 69 воспринимаются как даты в диапазоне 2000-2069. А числа от 70 до 99 как диапазон чисел 1970 - 1999.

yyyymmdd - 20180525

yyyy.mm.dd - 2018.05.25

Для времени тип Time использует 24-часовой формат. Он может принимать время в различных форматах:

hh:mi - 3:21 (хранимое значение 03:21:00)

hh:mi:ss - 19:21:34

hhmiss - 192134

Примеры значений для типов DATETIME и TIMESTAMP:

2018-05-25 19:21:34

2018-05-25 (хранимое значение 2018-05-25 00:00:00)

# Составные типы

ENUM: хранит одно значение из списка допустимых значений. Занимает 1-2 байта

SET: может хранить несколько значений (до 64 значений) из некоторого списка допустимых значений. Занимает 1-8 байт.

# Бинарные типы

TINYBLOB: хранит бинарные данные в виде строки длиной до 255 байт.

BLOB: хранит бинарные данные в виде строки длиной до 65 КБ.

MEDIUMBLOB: хранит бинарные данные в виде строки длиной до 16 МБ

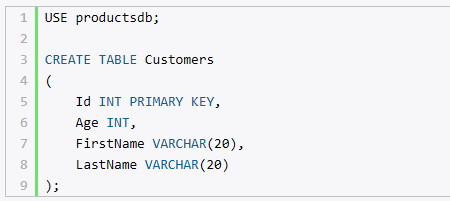
LARGEBLOB: хранит бинарные данные в виде строки длиной до 4 ГБ

# Атрибуты столбцов и таблиц

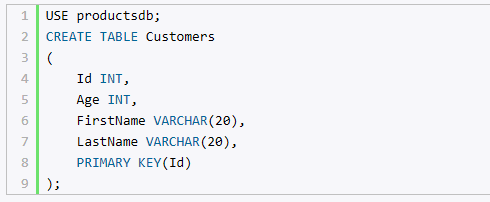
С помощью атрибутов можно настроить поведение столбцов. Рассмотрим, какие атрибуты мы можем использовать.

**PRIMARY KEY**

Атрибут PRIMARY KEY задает первичный ключ таблицы.

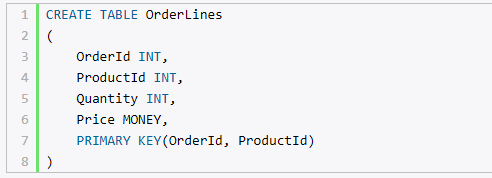


Установка первичного ключа на уровне таблицы:



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

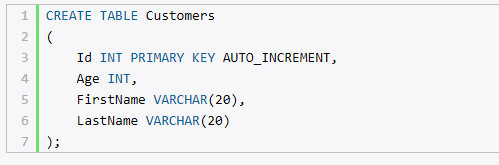
Первичный ключ может быть составным. Такой ключ использовать сразу несколько столбцов, чтобы уникально идентифицировать строку в таблице. Например:



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.

**AUTO\_INCREMENT**

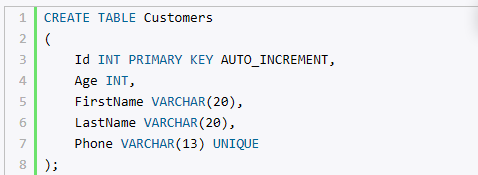
Атрибут AUTO\_INCREMENT позволяет указать, что значение столбца будет автоматически увеличиваться при добавлении новой строки. Данный атрибут работает для столбцов, которые представляют целочисленный тип или числа с плавающей точкой.



В данном случае значение столбца Id каждой новой добавленной строки будет увеличиваться на единицу.

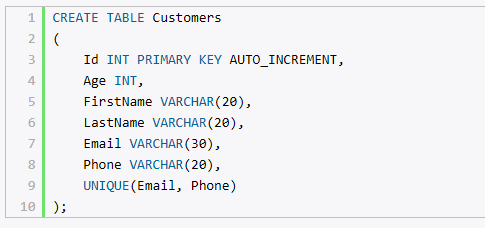
**UNIQUE**

Атрибут UNIQUE указывает, что столбец может хранить только уникальные значения.



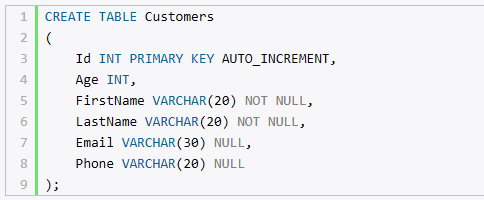
В данном случае столбец Phone, который представляет телефон клиента, может хранить только уникальные значения. И мы не сможем добавить в таблицу две строки, у которых значения для этого столбца будет совпадать.

Также мы можем определить этот атрибут на уровне таблицы:



**NULL и NOT NULL**

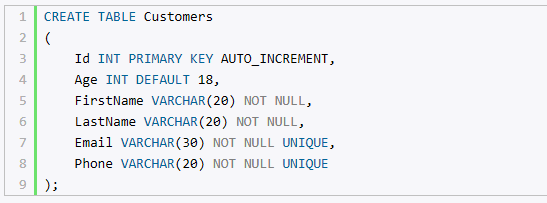
Чтобы указать, может ли столбец принимать значение NULL, при определении столбца ему можно задать атрибут NULL или NOT NULL. Если этот атрибут явным образом не будет использован, то по умолчанию столбец будет допускать значение NULL. Исключением является тот случай, когда столбец выступает в роли первичного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



В данном случае столбец Age по умолчанию будет иметь атрибут NULL.

**DEFAULT**

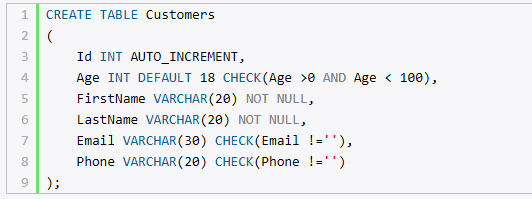
Атрибут DEFAULT определяет значение по умолчанию для столбца. Если при добавлении данных для столбца не будет предусмотрено значение, то для него будет использоваться значение по умолчанию.



Здесь столбец Age в качестве значения по умолчанию имеет число 18.

**CHECK**

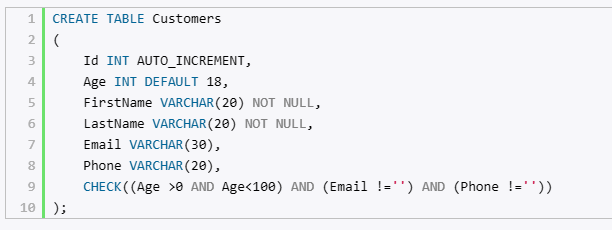
Атрибут CHECK задает ограничение для диапазона значений, которые могут храниться в столбце. Для этого после CHECK указывается в скобках условие, которому должен соответствовать столбец или несколько столбцов. Например, возраст клиентов не может быть меньше 0 или больше 100:



Кроме проверки возраста здесь также проверяется, что столбцы Email и Phone не могут иметь пустую строку в качестве значения (пустая строка не эквивалентна значению NULL).

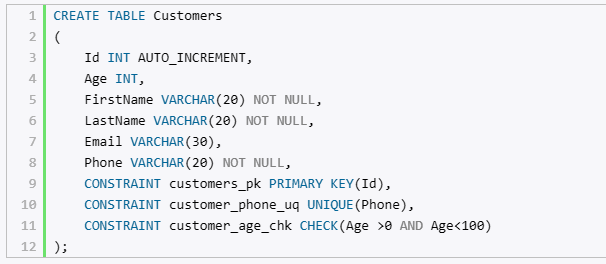
Для соединения условий используется ключевое слово AND. Условия можно задать в виде операций сравнения больше (>), меньше (<), не равно (!=).

Также CHECK можно использовать на уровне таблицы:



**Оператор CONSTRAINT.** Установка имени ограничений

С помощью ключевого слова CONSTRAINT можно задать имя для ограничений. Они указываются после ключевого слова CONSTRAINT перед атрибутами на уровне таблицы:



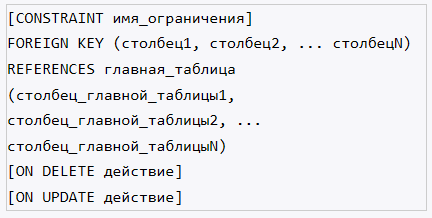
В данном случае ограничение для PRIMARY KEY называется customers\_pk, для UNIQUE - customer\_phone\_uq, а для CHECK - customer\_age\_chk. Смысл установки имен ограничений заключается в том, что впоследствии через эти имена мы сможем управлять ограничениями - удалять или изменять их.

Установить имя можно для ограничений PRIMARY KEY, CHECK, UNIQUE, а также FOREIGN KEY, который рассматриватся далее.

# Внешние ключи FOREIGN KEY

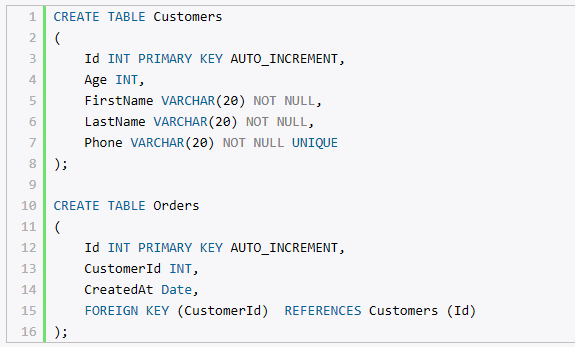
Внешние ключи позволяют установить связи между таблицами. Внешний ключ устанавливается для столбцов из зависимой, подчиненной таблицы, и указывает на один из столбцов из главной таблицы. Как правило, внешний ключ указывает на первичный ключ из связанной главной таблицы.

Общий синтаксис установки внешнего ключа на уровне таблицы:



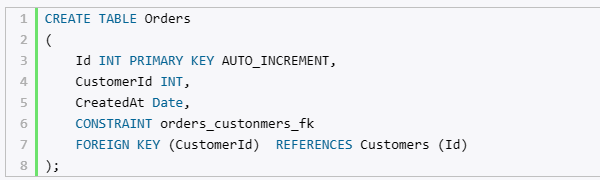
Для создания ограничения внешнего ключа после FOREIGN KEY указывается столбец таблицы, который будет представлять внешний ключ. А после ключевого слова REFERENCES указывается имя связанной таблицы, а затем в скобках имя связанного столбца, на который будет указывать внешний ключ. После выражения REFERENCES идут выражения ON DELETE и ON UPDATE, которые задают действие при удалении и обновлении строки из главной таблицы соответственно.

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



В данном случае определены таблицы Customers и Orders. Customers является главной и представляет клиента. Orders является зависимой и представляет заказ, сделанный клиентом. Таблица Orders 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.

С помощью оператора CONSTRAINT можно задать имя для ограничения внешнего ключа:



**ON DELETE и ON UPDATE**

С помощью выражений ON DELETE и ON UPDATE можно установить действия, которые выполняются соответственно при удалении и изменении связанной строки из главной таблицы. В качестве действия могут использоваться следующие опции:

CASCADE: автоматически удаляет или изменяет строки из зависимой таблицы при удалении или изменении связанных строк в главной таблице.

SET NULL: при удалении или обновлении связанной строки из главной таблицы устанавливает для столбца внешнего ключа значение NULL. (В этом случае столбец внешнего ключа должен поддерживать установку NULL)

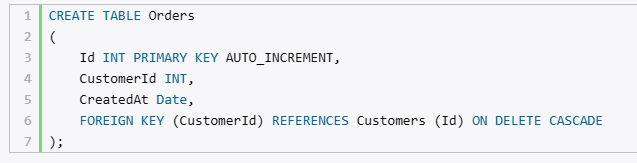
RESTRICT: отклоняет удаление или изменение строк в главной таблице при наличии связанных строк в зависимой таблице.

NO ACTION: то же самое, что и RESTRICT.

SET DEFAULT: при удалении связанной строки из главной таблицы устанавливает для столбца внешнего ключа значение по умолчанию, которое задается с помощью атрибуты DEFAULT. Несмотря на то, что данная опция в принципе доступна, однако движок InnoDB не поддерживает данное выражение.

# Каскадное удаление

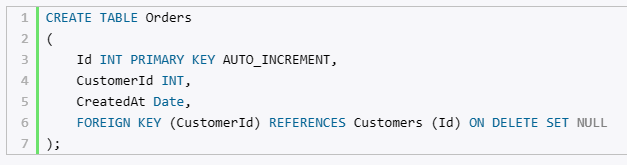
Каскадное удаление позволяет при удалении строки из главной таблицы автоматически удалить все связанные строки из зависимой таблицы. Для этого применяется опция CASCADE:



Подобным образом работает и выражение ON UPDATE CASCADE. При изменении значения первичного ключа автоматически изменится значение связанного с ним внешнего ключа. Однако поскольку первичные ключи 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.

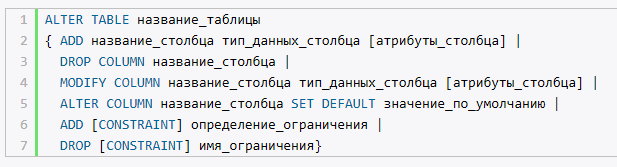
# Установка NULL

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



# Изменение таблиц и столбцов

Если таблица уже была ранее создана, и ее необходимо изменить, то для этого применяется команда ALTER TABLE. Ее сокращенный формальный синтаксис:



Вообще данная команда поддерживает гораздо больше опций и возможностей. Все их можно посмотреть в документации. Рассмотрим лишь основные сценарии, с которыми мы можем столкнуться.

# Добавление нового столбца

Добавим в таблицу Customers новый столбец Address:



В данном случае столбец Address имеет тип VARCHAR и для него определен атрибут NULL.

# Удаление столбца

Удалим столбец Address из таблицы Customers:



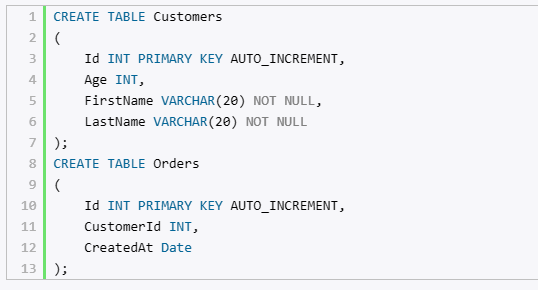
# Изменение типа столбца

Изменим в таблице Customers тип данных у столбца FirstName на CHAR(100) и установим для него атрибут NULL:



# Добавление и удаление внешнего ключа

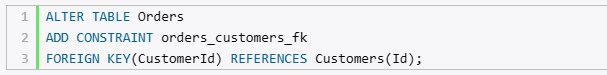
Пусть изначально в базе данных будут добавлены две таблицы, никак не связанные:



Добавим ограничение внешнего ключа к столбцу CustomerId таблицы Orders:



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



В данном случае ограничение внешнего ключа называется orders\_customers\_fk. Затем по этому имени мы можем удалить ограничение:



# Добавление и удаление первичного ключа

Добавим в таблицу Products первичный ключ:



Теперь удалим первичный ключ:



**Требования к оформлению отчета**

1. Отчет по данной работе является составной часть практического занятия «Создание и наполнение базы данных. Основы SQL
2. Отчет должен содержать:
3. Название работы
4. Тему, цель и задание к работе
5. Снимки экрана (скриншоты) процесса создания и модификации БД.
6. Выводы (что узнали, где можно применить полученные знания)