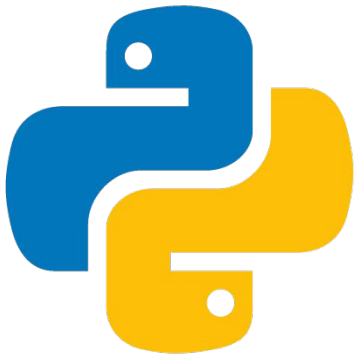


ТЕМА 5: РАБОТА С БАЗАМИ ДАННЫХ В ЯЗЫКЕ «PYTHON»

Приложение «sqlite3.exe»

Работа с базами данных предполагает использование структурированного языка запросов – **SQL (Structured Query Language)**, который ориентирован на выполнение операций над данными.
Для выполнения SQL-запросов применяется приложение **«sqlite3.exe»**, позволяющее работать с SQLite из командной строки

Указанное приложение загружается с сайта: http://www.sqlite.org/download.html, где в разделе «Precompiled Binaries for Windows» необходимо выбрать архив, соответствующий разрядности установленной на компьютер операционной системы, загрузить его и распаковать.

После чего нужно скопировать хранящийся в распакованном архиве файл **«sqlite3.exe»** в каталог, предназначенный для дальнейшей работы, в данном случае таковым является предварительно созданный каталог «C:\lesson».

Вменю **«Пуск»** в строке поиска следует ввести команду **«cmd»** и кликнуть по появившейся иконке, в результате чего откроется окно с приглашением для ввода команд.

Создание базы данных

1. Нужно перейти в каталог ***«C:\lesson»***, выполнив команду ***«cd C:\lesson»***. В командной строке появится приглашение: ***«C:\lesson>»***. По умолчанию в консоли используется кодировка ***«cp 866»***. Для изменения кодировки на ***«cp 1251»*** нужно ввести команду:

chcp 1251

2. Необходимо изменить название шрифта, поскольку точечные шрифты не поддерживают кодировку ***«Windows-1251»***. Для этого следует кликнуть правой кнопкой мыши на заголовке окна и из контекстного меню выбрать пункт ***«Свойства»***. Перейти на вкладку ***«Шрифт»*** открывшегося окна и выбрать пункт ***«Lucida Console»***, также можно изменить размер шрифта. После чего нужно нажать на кнопку ***«OK»***, для сохранения всех изменений. Проверить правильность установки кодировки можно посредством команды ***«chcp»***. Результат выполнения должен иметь следующий вид:

C:\lesson>chcp
Текущая кодовая страница: 1251

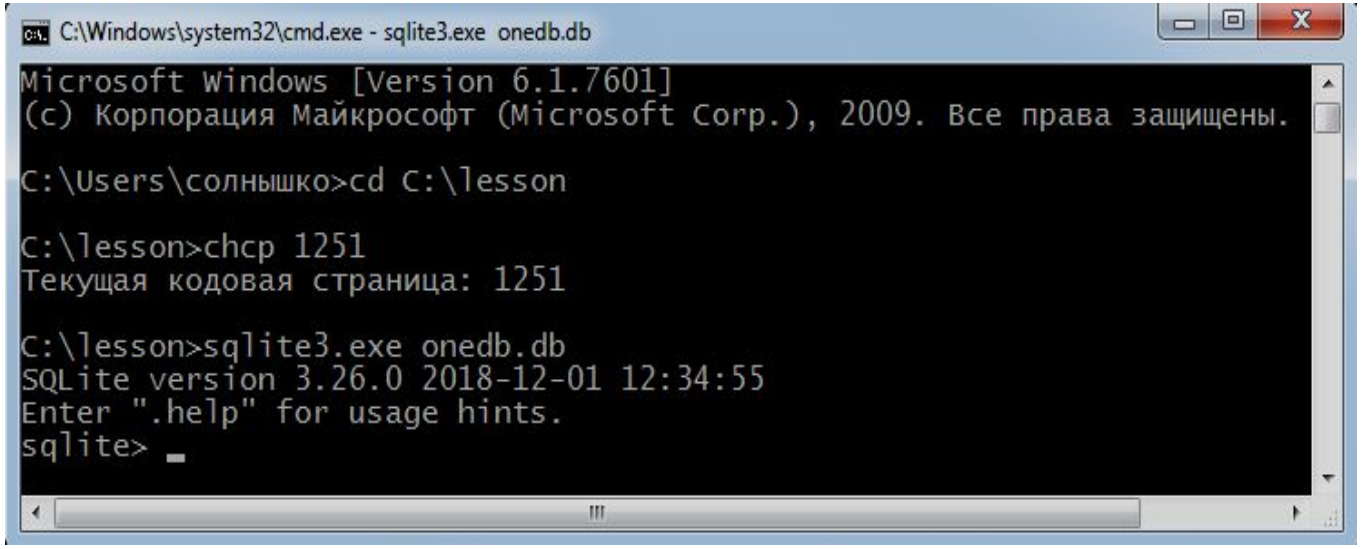
После выполнения всех указанных действий можно переходить к созданию новой базы данных, что осуществляется командой:

C:\lesson>sqlite3.exe onedb.db

Создание базы данных

Если файл ***«onedb.db»*** не существует, то будет создана и открыта для дальнейшей работы база данных с таким именем. В случае, если такая база данных уже существует, то она просто откроется без удаления содержимого.

Строка ***«sqiite>»*** здесь является приглашением для ввода команд. Каждая команда завершается точкой с запятой.



SQLite **позволяет использовать однострочные и многострочные комментарии:**

sqlite> -- Это однострочный комментарий
sqlite> /* Это многострочный комментарий */

Создание таблиц базы данных

СОЗДАТЬ ТАБЛИЦУ БАЗЫ ДАННЫХ МОЖНО С ПОМОЩЬЮ СЛЕДУЮЩЕЙ КОМАНДЫ:

CREATE [TEMP/TEMPORARY] TABLE [IF NOT EXISTS] <имя таблицы> (<имя поля 1> <тип данных поля 1> <опции поля 1>, <имя поля 2> <тип данных поля 2> <опции поля 2>, . . . , (<имя поля > <тип данных поля > <опции поля >, <дополнительные опции>);

КОМПОНЕНТЫ УКАЗАННОЙ КОМАНДЫ:

1. Если в рассмотренной команде после ключевого слова ***CREATE*** указано слово ***TEMP*** или ***TEMPORARY***, то это свидетельствует о том, что создается временная таблица. Которая после закрытия базы данных автоматически будет удалена. Пример создания временной таблицы ***«day»***, содержащей столбец ***«number»*** (в рассмотренном примере команда ***«.tables»*** выводит список всех таблиц из базы данных):

```
sqlite> CREATE TEMP TABLE day (number);
sqlite> .tables
tmp. day
```

Создание таблиц базы данных

КОМПОНЕНТЫ УКАЗАННОЙ КОМАНДЫ:

2. Применение необязательного словосочетания ***IF NOT EXISTS*** означает, что если таблица уже создана, то создавать ее вновь не нужно, в случае, если таблица уже существует, а словосочетание ***IF NOT EXISTS*** не указано, то будет выведено сообщение об ошибке:

```
sqlite> CREATE TEMP TABLE day (number);
sqlite> CREATE TEMP TABLE day (name);
Error: table day already exists
sqlite> CREATE TEMP TABLE IF NOT EXISTS day (name);
sqlite> PRAGMA table_info(day);
0| number||0||0
```

Команда ***«PRAGMA table_info (<имя таблицы>)»*** позволяет получить информацию о полях таблицы, результат работы указанной команды свидетельствует о том, что структура временной таблицы ***«day»*** не изменилась после выполнения запроса на создание таблицы с таким же названием.

Созданную таблицу базы данных можно удалить из базы, применяя при этом команду ***DROP TABLE <имя таблицы>***:

sqlite> DROP TABLE day;

Создание таблиц базы данных

КОМПОНЕНТЫ УКАЗАННОЙ КОМАНДЫ:

3. В параметрах ***<имя таблицы>*** и ***<имя поля>*** указывается идентификатор или строка.

ВАЖНО ПОМНИТЬ: имена, начинающиеся с префикса «sqlite_», зарезервированы для служебного использования.
Если в параметрах ***<имя таблицы>*** и ***<имя поля>*** указывается идентификатор, то **НАЗВАНИЕ НЕ ДОЛЖНО СОДЕРЖАТЬ ПРОБЕЛОВ**, а также **НЕ ДОЛЖНО СОВПАДАТЬ С КЛЮЧЕВЫМИ СЛОВАМИ SQL**, поскольку будет выведено сообщение об ошибке.

CREATE TEMP TABLE table (number);
Error: near "table": syntax error

Однако если вместо идентификатора указать строку, то ошибки не возникнет:

CREATE TEMP TABLE "table" (number);

Кроме того, идентификатор можно разместить в квадратных скобках:

CREATE TEMP TABLE [table] (number);

Создание таблиц базы данных

КОМПОНЕНТЫ УКАЗАННОЙ КОМАНДЫ:

4. Значение, указанное в параметре ***<тип данных>*** может быть отнесено к одному из пяти типов данных:
1) ***NULL*** – значение null;
2) ***INTEGER*** – целые числа;
3) ***REAL*** – вещественные числа;
4) ***TEXT*** – строки;
5) ***BLOB*** – бинарные данные.

При этом, если после ***INTEGER*** указаны ключевые слова ***PRIMARY KEY*** (ситуация, когда поле является первичным ключом), то в это поле можно вставить только целые числа или значение ***NULL***. При указании значения ***NULL*** будет вставлено число, на единицу большее максимального числа в столбце:

```
sqlite> CREATE TEMP TABLE day (number INTEGER PRIMARY
KEY);
sqlite> INSERT INTO day VALUES (16);
sqlite> INSERT INTO day VALUES (12.5);
Error: datatype mismatch
sqlite> INSERT INTO day VALUES ("первый");
Error: datatype mismatch
sqlite> INSERT INTO day VALUES (NULL);
sqlite> SELECT * FROM day;
16
17
```

Создание таблиц базы данных

КОМПОНЕНТЫ УКАЗАННОЙ КОМАНДЫ:

4. Значение, указанное в параметре ***<тип данных>*** может быть отнесено к одному из пяти типов данных:
1) ***NULL*** – значение null;
2) ***INTEGER*** – целые числа;
3) ***REAL*** – вещественные числа;
4) ***TEXT*** – строки;
5) ***BLOB*** – бинарные данные.

Кроме того, следует отметить, что при работе с типами данных может иметь место следующее преобразование типов данных: если строку, содержащую вещественное число, преобразовать в класс ***INTEGER***, то дробная часть будет отброшена:

```
sqlite> CREATE TEMP TABLE day (number TEXT);
sqlite> INSERT INTO day VALUES ("145.478");
sqlite> INSERT INTO day VALUES ("45.0");
sqlite> SELECT number, typeof (number) FROM day;
145.478|text
45.0|text
sqlite> SELECT CAST (number AS INTEGER) FROM day;
145
45
```

Создание таблиц базы данных

КОМПОНЕНТЫ УКАЗАННОЙ КОМАНДЫ:

5. В параметре **<опции поля>** могут быть указаны следующие конструкции:

1) ***NOT NULL [<обработка ошибок>]*** – означает, что поле обязательно должно иметь значение при вставке новой записи, если опция не указана, поле может содержать значение ***NULL***, необходимо отметить, что необязательный параметр ***[<обработка ошибок>]*** здесь, и далее задает способ разрешения конфликтных ситуаций, при этом форма конструкции имеет вид:

ON CONFLICT <алгоритм>

в параметре ***<алгоритм>*** могут быть указаны следующие значения:

– **ROLLBACK** – при ошибке транзакция завершается с откатом всех измененных ранее записей, дальнейшее выполнение прерывается, и выводится сообщение об ошибке, если активной транзакции нет, то используется алгоритм **ABORT**;

– **ABORT** – при возникновении ошибки аннулируются все изменения, произведенные текущей командой, и выводится сообщение об ошибке, все изменения, сделанные в транзакции предыдущими командами, сохраняются, алгоритм **ABORT** используется по умолчанию;

– **FAIL** – при возникновении ошибки все изменения, произведенные текущей командой, сохраняются, а не аннулируются, как в алгоритме **ABORT**, дальнейшее выполнение команды прерывается, и выводится сообщение об ошибке, а все изменения, сделанные в транзакции предыдущими командами, сохраняются;

– **IGNORE** – проигнорировать ошибку и продолжить выполнение без вывода сообщения об ошибке;

– **REPLACE** – при нарушении условия **UNIQUE** существующая запись удаляется, а новая вставляется, сообщение об ошибке не выводится, при нарушении условия **NOT NULL** значение **NULL** заменяется значением по умолчанию, если значение по умолчанию для поля не задано, то используется алгоритм **ABORT**, если нарушено условие **CHECK**, применяется алгоритм **IGNORE**:

Создание таблиц базы данных

КОМПОНЕНТЫ УКАЗАННОЙ КОМАНДЫ:

5. В параметре **<опции поля>** могут быть указаны следующие конструкции:

```
sqlite> CREATE TEMP TABLE day (
...> number UNIQUE ON CONFLICT REPLACE,
name TEXT);
sqlite> INSERT INTO day VALUES (4, "четверг");
sqlite> INSERT INTO day VALUES (5, "пятница");
sqlite> SELECT * FROM day;
4|четверг
5|пятница
```

Создание таблиц базы данных

КОМПОНЕНТЫ УКАЗАННОЙ КОМАНДЫ:

5. В параметре **<опции поля>** могут быть указаны следующие конструкции:

2) ***DEFAULT <значение>*** – задает для поля значение, которое будет использовано, если при вставке записи для этого поля не было явно указано значение:

```
sqlite> CREATE TEMP TABLE day (number, name INTEGER
DEFAULT 0);
sqlite> INSERT INTO day (number) VALUES (12);
sqlite> INSERT INTO day VALUES (15, 456);
sqlite> SELECT * FROM day;
12|0
15|456
```

в данном параметре могут быть указаны следующие специальные значения:

– ***CURRENT_TIME*** – текущее время в формате чч: мм: сс;

– ***CURRENT_DATE*** – текущая дата в формате гггг-мм-дд;

– ***CURRENT_TIMESTAMP*** – текущая дата и время в формате гггг-мм-дц чч: мм:СС:

```
sqlite> CREATE TEMP TABLE day (number
INTEGER,
...> t TEXT DEFAULT CURRENT_TIME,
...> d TEXT DEFAULT CURRENT_DATE,
...> dt TEXT DEFAULT CURRENT TIMESTAMP);
sqlite> INSERT INTO day (number) VALUES (1);
sqlite> SELECT * FROM day;
1|10:00:55|2019-02-05|2019-02-05 10:00:55
sqlite> /* Текущая дата на компьютере: 2019-02-05
10:00:55 */
```

Создание таблиц базы данных

КОМПОНЕНТЫ УКАЗАННОЙ КОМАНДЫ:

5. В параметре **<опции поля>** могут быть указаны следующие конструкции:

3) ***COLLATE <функция>*** – задает функцию сравнения для класса text, здесь могут быть указаны следующие функции:

– ***BINARY*** – обычное сравнение, значение по умолчанию);

– ***NOCASE*** – сравнение без учета регистра (не учитывает регистр только латинских букв, однако, при использовании русских букв возникают проблемы с регистром);

– ***RTRIM*** – предварительное удаление лишних пробелов справа:

```
sqlite> CREATE TEMP TABLE day (number, name TEXT
COLLATE NOCASE);
sqlite> INSERT INTO day VALUES ("abc", "abc");
sqlite> SELECT number = "ABC" FROM day; /* значение
не найдено */
0
sqlite> SELECT name = "ABC" FROM day; /* значение
найдено */
1
```

Создание таблиц базы данных

КОМПОНЕНТЫ УКАЗАННОЙ КОМАНДЫ:

5. В параметре **<опции поля>** могут быть указаны следующие конструкции:

4) ***UNIQUE [<обработка ошибок>]*** – указывает, что поле может содержать только уникальные значения;

5) ***CHECK (<условие>)*** – значение поля, должно удовлетворять указанному условию:

```
sqlite> CREATE TEMP TABLE day (
...> number INTEGER CHECK (number IN (15, 30)));
sqlite> INSERT INTO day VALUES (15); /* значение
удовлетворяет условию */
sqlite> INSERT INTO day VALUES (20); /* значение
не удовлетворяет условию */
Error: constraint failed
```

Создание таблиц базы данных

КОМПОНЕНТЫ УКАЗАННОЙ КОМАНДЫ:

5. В параметре **<опции поля>** могут быть указаны следующие конструкции:

6) ***PRIMARY KEY [<обработка ошибок>]*** – указывает, что поле является первичным ключом таблицы, если полю назначен класс ***INTEGER***, то в это поле можно вставить только целые числа или значение ***NULL***, при указании значения ***NULL*** будет вставлено число, на единицу большее максимального из хранящихся в поле чисел:

```
sqlite> CREATE TEMP TABLE day (number INTEGER
PRIMARY KEY, name TEXT);
sqlite> INSERT INTO day VALUES (NULL, "понедельник");
sqlite> INSERT INTO day VALUES (NULL, "вторник");
sqlite> SELECT * FROM day;
1|понедельник
2|вторник
sqlite> DELETE FROM day WHERE number=2;
sqlite> INSERT INTO day VALUES (NULL, "среда");
sqlite> SELECT * FROM day;
1|понедельник
2|среда
```

Создание таблиц базы данных

КОМПОНЕНТЫ УКАЗАННОЙ КОМАНДЫ:

6. В необязательном параметре ***<дополнительные опции>*** могут быть указаны следующие конструкции:

1) ***PRIMARY KEY (<список полей через запятую>) [<обработка ошибок>]*** – позволяет задать первичный ключ для нескольких полей таблицы;

2) ***UNIQUE (<список полей через запятую>) [<обработка ошибок>]*** – указывает, что заданные поля могут содержать только уникальный набор значений;

3) ***CHECK (<условие>)*** – значение должно удовлетворять указанному условию

Вставка записей в таблицу

ДОБАВИТЬ ЗАПИСЬ В ТАБЛИЦУ БАЗЫ ДАННЫХ МОЖНО С ПОМОЩЬЮ СЛЕДУЮЩЕЙ КОМАНДЫ:

INSERT [OR <алгоритм>] INTO <имя таблицы> (<имя поля 1>, <имя поля 2>, . . . , <имя поля >); VALUES (<значение поля 1>, <значение поля 2>, . . . , <значение поля >)/DEFAULT VALUES;

В рассмотренной конструкции параметр ***OR <алгоритм>*** **ЯВЛЯЕТСЯ НЕОБЯЗАТЕЛЬНЫМ**, он применяется для задания алгоритма обработки ошибок.

Если в таблице существуют поля, которым в инструкции ***INSERT*** не присваивается значение, то они получат значения по умолчанию. В случае если список полей не указан, то значения задаются в том порядке, в котором поля перечислены в инструкции ***CREATE TABLE.***

Конструкция ***VALUES (<список полей>)*** может быть заменена на ***DEFAULT VALUES***. В этом случае будет создана новая запись, все поля которой получат значения по умолчанию или ***NULL***, если таковые не были заданы при создании таблицы.

Вставка записей в таблицу

Создание таблиц: «buyer» (покупатель), «supplier» (поставщик) и «product» (товар):

```
sqlite> CREATE TABLE buyer (
...> id_ buyer INTEGER PRIMARY KEY AUTOINCREMENT,
...> name_ buyer TEXT;
sqlite> CREATE TABLE supplier (
...> id_ supplier INTEGER PRIMARY KEY AUTOINCREMENT,
...> name_ supplier TEXT);
sqlite> CREATE TABLE product (
...> id_ buyer INTEGER,
...> id_ supplier INTEGER,
...> name_ product TEXT);
```

Вставка записей в таблицу

Заполнение таблиц связанными данными:

```
sqlite> INSERT INTO buyer (name_ buyer)
...> VALUES ("Иванов");
sqlite> INSERT INTO supplier VALUES (NULL, "Петров");
sqlite> SELECT * FROM buyer;
1|Иванов
sqlite> SELECT * FROM supplier;
1|Петров
sqlite> INSERT INTO product (id_ buyer, id_ supplier, name_
product)
...> VALUES (1, 1, "телевизор");
sqlite> SELECT * FROM product;
1|1|телевизор
```

В таблице ***«buyer»*** указано только одно поле ***«name_ buyer»***, в данном случае полю ***«id_ buyer»*** присваивается значение по умолчанию.

В таблице ***«supplier»*** поле ***«id_supplier»*** объявлено как первичный ключ, поэтому туда будет вставлено значение, на единицу большее максимального значения в поле. Такого же эффекта можно достичь, если в качестве значения передать ***NULL***.

В таблице ***«product»*** значения полей ***«id_ buyer»*** и ***«id_supplier»*** должны содержать идентификаторы соответствующих записей из таблиц ***«buyer»*** и ***«supplier»***. Для этого сначала следует выполнить запрос на выборку данных из родительских таблиц ***«buyer»*** и ***«supplier»***.

Обновление и удаление записей

На практике довольно часто встречаются ситуации, когда предпринимается попытка добавления записи с уже существующим в таблице идентификатором, или при условии того, что значение индекса ***UNIQUE*** не уникально, в результате указанных действий программа выведет сообщение об ошибке. Если необходимо, чтобы имеющиеся неуникальные записи обновлялись без вывода сообщения об ошибке, можно применять следующую инструкцию:

```
INSERT OR REPLACE INTO <имя таблицы>
VALUES (<значение поля 1>, <значение поля 2>,.....
<значение поля n>);
```

Аналогичный результат может быть получен с помощью следующей инструкции:

```
REPLACE INTO <имя таблицы> VALUES (<значение поля 1>,
<значение поля 2>,..... <значение поля n>);
```

Пример замены значения поля ***«name_ buyer»***, идентификатор которого равен 1:

```
sqlite> INSERT OR REPLACE INTO buyer
...> VALUES (1, Сидоров);
sqlite> SELECT * FROM buyer;
1|Сидоров
```

```
sqlite> REPLACE INTO buyer VALUES (1, Сидоров);
sqlite> SELECT * FROM buyer;
1|Сидоров
```

Обновление и удаление записей

Выполнить обновление записи в таблице также позволяет инструкция ***UPDATE***, имеющая следующий формат:

```
UPDATE [OR <алгоритм>] <имя таблицы> SET <имя поля 1>="<значение поля 1>", <имя поля 2>="<значение поля 2>", ..... <имя поля n>="<значение поля n>" WHERE <условие>;
```

Параметр **OR <алгоритм> ЯВЛЯЕТСЯ НЕОБЯЗАТЕЛЬНЫМ**, он применяется для задания алгоритма обработки ошибок.

После ключевого слова ***SET*** указываются названия полей и их новые значения, следующие за знаком равенства.

Для ограничения набора изменяемых записей, применяется инструкция ***WHERE***. Если не указано конкретное ***<условие>***, то в таблице будут обновлены все записи:

Пример изменения значения поля ***«name_buyer»***, идентификатор которого равен 1:

```
sqlite> UPDATE buyer SET name_buyer='Васильев' WHERE id_buyer=1;
sqlite> SELECT * FROM buyer;
1|Васильев
```

Обновление и удаление записей

Удаление записи может быть достигнуто через применение инструкции ***DELETE***:

```
DELETE FROM <имя таблицы> WHERE <условие>;
```

Пример удаления значения поля ***«name_buyer»***, идентификатор которого равен 1:

```
sqlite> DELETE FROM buyer WHERE id_ buyer =1;
SELECT * FROM buyer;
```

В случае, когда не указано конкретное ***<условие>***, то из таблицы будут удалены все записи.

Преобразование структуры таблицы

Для выполнения изменений структуры таблиц баз данных в ***SQLite*** применяется инструкция ***ALTER TABLE***, которая позволяет осуществлять следующие действия:

1) переименование таблицы;
2) добавление поля.

Формат данной инструкции имеет вид:

```
ALTER TABLE <имя таблицы> <преобразование>;
```

При этом параметр ***<преобразование>***, в зависимости от предназначения, может быть представлен в одной из следующих форм:

1) ***RENAME TO <новое имя таблицы>***; – применяется для переименования таблиц, в качестве примера переименуем таблицу ***«product»*** в ***«items»***, и в качестве результата выведем названия всех таблиц базы данных:

```
sqlite> ALTER TABLE product RENAME TO items;
sqlite> .tables
buyer    items    supplier
```

Преобразование структуры таблицы

Для выполнения изменений структуры таблиц баз данных в ***SQLite*** применяется инструкция ***ALTER TABLE***, которая позволяет осуществлять следующие действия:

1) переименование таблицы;
2) добавление поля.

Формат данной инструкции имеет вид:

```
ALTER TABLE <имя таблицы> <преобразование>;
```

При этом параметр ***<преобразование>***, в зависимости от предназначения, может быть представлен в одной из следующих форм:

2) ***ADD COLUMN <имя нового поля> <тип данных нового поля > <опции нового поля >;*** – применяется для добавления нового поля, которое будет размещено после всех существующих полей таблицы.

*** В новом поле нужно задать значение по умолчанию, или же значение ***NULL*** должно быть допустимым, кроме того, вновь добавляемое поле не может быть объявлено как ***PRIMARY KEY или UNIQUE***.

ПРИМЕР добавления поля ***«quantity»*** (количество) в таблицу **«items»** и вывода информации о полях таблицы:

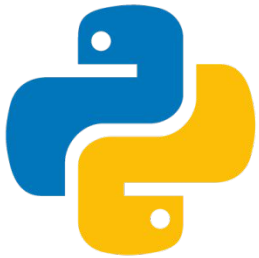
```
sqlite> ALTER TABLE items ADD COLUMN quantity INTEGER DEFAULT 0;
sqlite> PRAGMA table_info(items);
0|id_buyer|INTEGER|0||0
1|id_supplier|INTEGER|0||0
2|name_product|TEXT|0||0
3| quantity |INTEGER|0|0|0
```

Извлечение данных из таблицы

Извлекать данные из таблицы позволяет инструкция ***SELECT***, которая записывается в следующем формате:

```
SELECT ALL/DISTINCT <имя поля 1>, <имя поля 2>,.....
<имя поля n> FROM <имя таблицы> AS <псевдоним>
WHERE <условие>;
GROUP BY <имя поля> HAVING <агрегатная функция>
<условие>;
ORDER BY <имя поля> COLLATE/BINARY/NOCASE
ASC/DESC;
LIMIT <ограничение>;
```

После ключевого слова ***SELECT*** можно указать слово ***ALL*** или ***DISTINCT***. При этом, ***ALL*** является значением по умолчанию и говорит, что выводятся все записи, а применение слова ***DISTINCT*** позволяет вывести только уникальные значения.
SQL-команда ***SELECT*** находит в указанной таблице все записи, удовлетворяющие условию инструкции ***WHERE***. Если инструкция *WHERE* не указана, то из таблицы будут выведены все записи.



Извлечение данных из таблицы

ПРИМЕР получения всех записей из таблицы ***«supplier»***:

```
sqlite> SELECT id_ supplier, name_ supplier FROM supplier;
1|Петров
2| Зайцев
```

```
sqlite> SELECT * FROM supplier;
1|Петров
2| Зайцев
```

ПРИМЕР вывода записи с идентификатором, равным единице из таблицы ***«supplier»:***

```
sqlite> SELECT id_ supplier, name_ supplier FROM supplier
WHERE id_ supplier =l;
1|Петров
```

ПРИМЕР SQL-команда ***SELECT*** позволяет вместо перечисления полей указывать математическое выражение, которое будет вычислено и выведено в качестве результата:

```
sqlite> SELECT 12-4;
8
```

ПРИМЕР облегчения обращения к результату выполненного выражения через псевдоним, указать который нужно после выражения через ключевое слово ***AS***:

```
sqlite> SELECT (15+3) AS number_1, (35/5) AS number_2;
18|7
```

Извлечение данных из таблицы

ПРИМЕР замены индекса поставщика ***(id_ supplier)*** в таблице ***«items»*** на соответствующее название ***(name_ supplier)*** из таблицы ***«supplier»***:

```
sqlite> SELECT i.name_product, s.name_supplier FROM items AS
I, supplier AS s
...> WHERE i.id_supplier = s.id_ supplier;
телевизор|Петров
```

Инструкция ***GROUP BY*** позволяет осуществлять группировку нескольких записей.

ПРИМЕР вывода количества товаров, предоставленных каждым из поставщиком:

```
sqlite> SELECT id_ supplier, COUNT (id_ supplier) FROM items
...> GROUP BY id_ supplier;
1|1
2|3
```

ПРИМЕР вывода идентификаторов поставщиков, предоставивших более одного наименования товара:

```
sqlite> SELECT id_ supplier FROM items
...> GROUP BY id_ supplier HAVING COUNT (id_ supplier)>1;
2
```

При условии необходимости ограничения сгруппированного набора записей следует воспользоваться инструкцией ***HAVING***, которая выполняет те же функции, что и инструкция ***WHERE***, но только для сгруппированного набора.

Извлечение данных из таблицы

В качестве агрегатной функции могут применяться следующие:

1) COUNT (<имя поля>/*) – выводит количество записей в указанном поле;

2) MIN (<имя поля>) – выводит минимальное значение в указанном поле;

3) MAX (<имя поля>) – выводит максимальное значение в указанном поле;

4) AVG (<имя поля>) – выводит среднюю величину значений в указанном поле;

5) SUM (<имя поля>) – выводит сумму значений в указанном поле в виде целого числа;

6) TOTAL (<имя поля>) – сумму значений в указанном поле в виде в виде числа с плавающей точкой;

7) GROUP_CONCAT (<имя поля >, <разделитель>) – выводит строку, которая содержит все значения из указанного поля, разделенные указанным разделителем, при этом, если разделитель не указан, то используется запятая.

Извлечение данных из таблицы

Записи таблиц можно сортировать, применяя при этом инструкцию ***ORDER BY.*** Возможна сортировка сразу по нескольким полям.

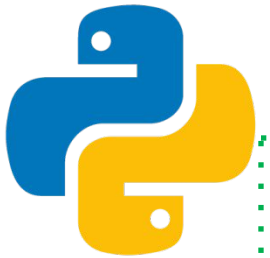
1) по возрастанию с помощью значения ***ASC***, важно отметить, что сортировка по возрастанию выполняется по умолчанию;

2) по убыванию посредством применения значения ***DESC***.

ПРИМЕР имена поставщиков по возрастанию и убыванию:

```
sqlite> SELECT * FROM supplier ORDER BY name_ supplier DESC;
1|Петров
2|Зайцев
```

```
sqlite> SELECT * FROM supplier ORDER BY name_ supplier;
2|Зайцев
1|Петров
```



Извлечение данных из таблицы

В случае, если требуется, чтобы по результатам поиска выводились не все найденные записи, а лишь их часть, то следует использовать инструкцию ***LIMIT***.

1) LIMIT <количество записей> (задает количество записей от начальной позиции, которая имеет индекс, равный нулю);

2) LIMIT <начальная позиция>, <количество записей>;

3) LIMIT <количество записей> OFFSET <начальная позиция>.

```
sqlite> CREATE TEMP TABLE day (id INTEGER);
sqlite> INSERT INTO day VALUES(1);
sqlite> INSERT INTO day VALUES(2);
sqlite> INSERT INTO day VALUES(3) ;
sqlite> INSERT INTO day VALUES(4);
sqlite> INSERT INTO day VALUES(5);
sqlite> SELECT * FROM day LIMIT 2;
1
2
sqlite> SELECT * FROM day LIMIT 3, 3;
4
5
sqlite> SELECT * FROM day LIMIT 3 OFFSET 2;
3
4
5
```

Извлечение данных из таблицы

Выборка данных из нескольких таблиц

Применение команды ***SELECT*** позволяет осуществлять выборку данных одновременно из нескольких таблиц.
Для этого достаточно перечислить нужные таблицы через запятую в инструкции ***FROM*** и указать в инструкции ***WHERE*** через запятую пары полей, являющиеся для этих таблиц связующими. При этом в условии и перечислении полей сначала указывается название таблицы или ее псевдоним, а затем через точку название поля.

ПРИМЕР вывода товары из таблицы ***«items»***, где вместо ***«id_buyer»*** указывается ***«name_buyer»***, а ***«id_supplier»*** заменено ***«name_supplier»***:

```
sqlite> SELECT items.name_product, buyer.name_buyer,
supplier.name_supplier
...> FROM items, buyer, supplier
...> WHERE items.id_buyer=buyer.id_buyer AND
...> items.id_supplier=supplier.id_supplier;
микроволновая печь|Швецов|Зайцев
пылесос|Степанов|Зайцев
ноутбук|Степанов|Зайцев
```

ПРИМЕР вывода товары из таблицы ***«items»***, где вместо ***«id_buyer»*** указывается ***«name_buyer»***, а ***«id_supplier»*** заменено ***«name_supplier»***:

```
sqlite> SELECT name_product, name_buyer, name_supplier
...> FROM items AS i, buyer AS b, supplier AS s
...> WHERE i.id_buyer =b.id_buyer AND
...> i.id_supplier =s.id_supplier;
микроволновая печь|Швецов|Зайцев
пылесос|Степанов|Зайцев
ноутбук|Степанов|Зайцев
```

Извлечение данных из таблицы

Выборка данных из нескольких таблиц

Связать таблицы возможно, применив оператор ***JOIN***.
ПРИМЕР применения данного оператора:

```
sqlite> SELECT name_product, name_buyer, name_supplier
...> FROM items JOIN buyer JOIN supplier
...> WHERE items.id_buyer=buyer.id_buyer AND
...> items.id_supplier=supplier.id_supplier;
микроволновая печь|Швецов|Зайцев
пылесос|Степанов|Зайцев
ноутбук|Степанов|Зайцев
```

Инструкция ***WHERE*** может быть заменена на инструкцию ***ON***, помимо этого в инструкции ***WHERE*** можно указывать условие.
ПРИМЕР вывода товаров, предоставленных поставщиком, идентификатор которого равен двум:

```
sqlite> SELECT name_product, name_buyer, name_supplier
...> FROM items JOIN buyer JOIN supplier
...> ON items.id_buyer=buyer.id_buyer AND
...> items.id_supplier=supplier.id_supplier;
...> WHERE items.id_supplier=2;
микроволновая печь|Швецов|Зайцев
пылесос|Степанов|Зайцев
ноутбук|Степанов|Зайцев
```

Извлечение данных из таблицы

Выборка данных из нескольких таблиц

В случае, если названия связующих полей в таблицах являются одинаковыми, то вместо инструкции ***ON*** можно применять инструкцию ***USING***:

```
sqlite> SELECT name_product, name_buyer, name_supplier
...> FROM buyer JOIN items USING (id_ buyer) JOIN supplier
USING (id_ supplier);
микроволновая печь|Швецов|Зайцев
пылесос|Степанов|Зайцев
ноутбук|Степанов|Зайцев
```

Оператор ***JOIN*** объединяет все записи, которые существуют во всех связующих полях.

ПРИМЕР если вывести количество товаров, поставленных каждым из поставщиков, то поставщики, не предоставившие товар, выведены не будут:

```
sqlite> SELECT name_ supplier, COUNT(name_product)
...> FROM supplier JOIN items USING (id_supplier)
...> GROUP BY supplier.id_supplier;
Петров|1
Зайцев|3
```

Извлечение данных из таблицы

Выборка данных из нескольких таблиц

В рассмотренном примере не выведено количество товаров, которые предоставили другие поставщики из таблицы ***«supplier»***. Чтобы получить количество товаров, полученных от каждого из поставщиков необходимо использовать левостороннее объединение, которое имеет следующий формат:

```
FROM <имя таблицы 1> LEFT JOIN <имя таблицы 2>
USING (<имя поля>);
```

Применение левостороннего объединения позволяет выводить записи, соответствующие условию, а также записи из таблицы ***<имя таблицы 1>***, которым нет соответствия в таблице ***<имя таблицы 2>*** (при этом поля из таблицы ***<имя таблицы 2>*** будут иметь нулевые значение).

ПРИМЕР применения левостороннего объединения:

```
sqlite> SELECT name_supplier, COUNT(name_product)
...> FROM supplier LEFT JOIN items USING (id_supplier)
...> GROUP BY supplier.id_supplier
...> ORDER BY supplier.name_supplier;
Петров|1
Краснов|0
Зайцев|3
```

Извлечение данных из таблицы

Работа с условиями в инструкциях *WHERE* и *HAVING*

Инструкции ***WHERE*** и ***HAVING*** позволяют ограничить набор выводимых, изменяемых или удаляемых записей с помощью некоторого условия внутри которого могут быть указаны соответствующие операторы сравнения:

1. Условие равенства, выполняемое с помощью символов «=» или «==»:

```
sqlite> SELECT * FROM supplier WHERE id_ supplier=2;
2|Зайцев

sqlite> SELECT 10 =10, 5 ==4;
1|0
```

Результат сравнения двух строк зависит от применяемой функции сравнения, которую можно задать с помощью ранее рассмотренной функции ***COLLATE <функция>***.
ПРИМЕР

```
sqlite> CREATE TEMP TABLE day (number, name TEXT
COLLATE NOCASE);
sqlite> INSERT INTO day VALUES (1, "понедельник");
sqlite> SELECT name = "ПОНЕДЕЛЬНИК" FROM day;
0
sqlite> SELECT name = "понедельник" FROM day;
1
```

```
sqlite> SELECT 'w' = 'W' COLLATE NOCASE;
1
sqlite> SELECT 'ð' = 'Д' COLLATE NOCASE;
0
```

Извлечение данных из таблицы

Работа с условиями в инструкциях *WHERE* и *HAVING*

Инструкции ***WHERE*** и ***HAVING*** позволяют ограничить набор выводимых, изменяемых или удаляемых записей с помощью некоторого условия внутри которого могут быть указаны соответствующие операторы сравнения:

2. Условие неравенства, выполняемое с помощью символов *«!=»* или *«<>»*:

```
sqlite> SELECT 10!=10, 10<>15;
1|0
```

3. Условие «меньше», выполняемое с помощью символа *«<»*;

4. Условие «больше», выполняемое с помощью символа *«>»*;

5. Условие «меньше либо равно», выполняемое с помощью символа *«<=»*;

6. Условие «больше либо равно», выполняемое с помощью символа *«>=»*;

7. Условие на наличие нулевого значения, выполняемое с помощью ***«IS NOT NULL»***, ***«NOT NULL»*** или ***«NOTNULL»***;

8. Условие на отсутствие нулевого значения, выполняемое с помощью ***«IS NULL»*** или ***«ISNULL»;***

Извлечение данных из таблицы

Работа с условиями в инструкциях *WHERE* и *HAVING*

Инструкции ***WHERE*** и ***HAVING*** позволяют ограничить набор выводимых, изменяемых или удаляемых записей с помощью некоторого условия внутри которого могут быть указаны соответствующие операторы сравнения:

9. Условие на проверку вхождения в диапазон значений, выполняемое с помощью ***«BETWEEN <начало> AND <конец>»***, при этом начало и конец указанного диапазона тоже учитывается:

```
sqlite> SELECT 15 BETWEEN 12 AND 38;
1
```

10. Условие на проверку наличия значения в определенном наборе, выполняемое с помощью ***«IN (<список значений>)»*** , при данном сравнении учитывается регистр букв:

```
sqlite> SELECT 'понедельник' IN ('понедельник', 'вторник', 'среда');
1
sqlite> SELECT 'Понедельник' IN ('понедельник', 'вторник', 'среда');
0
```

Извлечение данных из таблицы

Работа с условиями в инструкциях *WHERE* и *HAVING*

Инструкции ***WHERE*** и ***HAVING*** позволяют ограничить набор выводимых, изменяемых или удаляемых записей с помощью некоторого условия внутри которого могут быть указаны соответствующие операторы сравнения:

11. Условие на проверку соответствия шаблону, выполняемое с помощью ***«LIKE <шаблон>»*** , при этом в шаблоне применяются следующие специальные символы, которые могут быть размещены в любом месте шаблона:

1) ***«%»*** – любое количество символов или полное их отсутствие;

2) ***«_»*** – любой одиночный символ.

ПРИМЕР, который позволяет определить наличие вхождения слова «среда» в шаблон:

```
sqlite> SELECT 'понедельник вторник среда' LIKE '%среда%';
1
```

Также следует отметить, что шаблон для поиска может иметь сложную структуру:

```
sqlite> SELECT 'понедельник вторник среда' LIKE
'%ик_%вт_ик%';
0
```

Извлечение данных из таблицы

Работа с условиями в инструкциях *WHERE* и *HAVING*

Инструкции ***WHERE*** и ***HAVING*** позволяют ограничить набор выводимых, изменяемых или удаляемых записей с помощью некоторого условия внутри которого могут быть указаны соответствующие операторы сравнения:

На практике возникают ситуации, когда необходимо найти символы ***«%»*** и ***«_»***, которые являются специальными. Сделать это позволяет функция ***ESCAPE <символ>***:

```
sqlite> SELECT '15%' LIKE '15\%' ESCAPE '\';
1
sqlite> SELECT '15$' LIKE '15%';
1
```

Как уже было отмечено ранее, при сравнении с шаблоном букв латинского алфавита регистр символов не учитывается. Для того, чтобы такой учет выполнялся необходимо параметру ***«case_sensitive_like»*** в SQL-команде ***PRAGMA*** присвоить одно из значений: ***«true», «1», «yes», «on»*:**

```
sqlite> PRAGMA case_sensitive_like = true;
sqlite> SELECT 'w' LIKE 'W';
0
```

Извлечение данных из таблицы

Работа с индексами

В целях ускорения выполнения запросов применяются индексы, или ключи. В *SQLite* существуют следующие виды индексов:
1) первичный ключ (для создания такого индекса используется ключевое слово ***PRIMARY KEY***);
2) уникальный индекс (когда применяется составной первичный ключ);
3) обычный индекс.

Для того, чтобы посмотреть, каким образом будет выполняться запрос и какие индексы при этом будут использоваться, позволяет SQL-команда ***EXPLAIN***, которая имеет следующий формат:

```
EXPLAIN QUERY PLAN <SQL-запрос>;
```

Выполним поиск в обычном поле, не являющемся первичным ключом:

```
sqlite> EXPLAIN QUERY PLAN SELECT * FROM items WHERE
id_supplier =1;
QUERY PLAN
-- SEARCH TABLE items
```

Выполним поиск в поле, являющемся первичным ключом:

```
sqlite> EXPLAIN QUERY PLAN SELECT * FROM supplier
WHERE id_supplier =1;
QUERY PLAN
-- SEARCH TABLE supplier USING INTEGER PRIMARY KEY
```

Извлечение данных из таблицы

Работа с индексами

Для создания индекса применяется SQL-команда ***CREATE INDEX***, которая имеет следующий формат записи:

CREATE UNIQUE INDEX [IF NOT EXISTS] <имя индекса> ON <имя таблицы> (<имя поля>);

*** Если указывается ключевое слово ***UNIQUE***, то создается уникальный индекс, что обеспечивает отсутствие дублирования данных в поле. Если слово ***UNIQUE*** не указано, то создается обычный индекс.

ПРИМЕР создание обычного индекса для номера поставщика и его проверка с помощью SQL-команды ***EXPLAIN***:

```
sqlite> EXPLAIN QUERY PLAN SELECT * FROM items WHERE id_ supplier =1;
QUERY PLAN
-- SCAN TABLE items
sqlite> CREATE INDEX index_supplier ON items (id_supplier);
sqlite> EXPLAIN QUERY PLAN SELECT * FROM items WHERE id_supplier =1;
QUERY PLAN
-- SEARCH TABLE items USING INDEX index_supplier (id_supplier =?)
```

Извлечение данных из таблицы

Работа с индексами

Над индексами можно выполнять следующие операции:

1) удаление обычного или уникального индекса, что выполняет SQL-команда ***DROP INDEX***, имеющая следующий формат:

```
DROP INDEX [IF NOT EXISTS] <имя индекса>;
```

2) получение статистической информации об индексах, которая помещается в специальную таблицу ***«sqiite_stat1»***, изначально указанная таблица пуста, для сбора и помещения статистических данных в таблицу ***«sqiite_stat1»*** применяется команда ***ANALYZE***, формат записи которой имеет вид:

```
ANALYZE <имя таблицы>;
```

```
sqlite> SELECT * FROM sqlite_stat1;
Error: no such table: sqlite_statl
sqlite> ANALYZE;
sqlite> SELECT * FROM sqlite_stat1;
items|index_supplier|4 2
supplier||3
buyer||2
```

Извлечение данных из таблицы

Создание вложенных запросов

При создании таблицы с помощью вложенного запроса применяется следующий формат записи:

```
CREATE [TEMP/TEMPORARY] TABLE [IF NOT EXISTS] <имя таблицы> AS <инструкция SELECT>;
```

ПРИМЕР создания временной копии таблицы ***«supplier»*** и вывода ее содержимого:

```
sqlite> CREATE TEMP TABLE supplier1 AS SELECT * FROM supplier;
sqlite> SELECT * FROM supplier1;
1|Петров
2|Зайцев
3|Краснов
```

Извлечение данных из таблицы

Создание вложенных запросов

Вложенные запросы можно использовать
и в инструкции ***INSERT***, структура записи которой имеет следующий вид:

```
INSERT [OR <алгоритм>] INTO <имя таблицы> (<имя поля 1>, <имя поля 2>,..... <имя поля n>) <инструкция SELECT>;
```

ПРИМЕР удаления всех данных из созданной ранее временной таблицы ***«supplier1»***, а затем ее наполнение с помощью вложенного запроса:

```
sqlite> DELETE FROM supplier1;
sqlite> INSERT INTO supplier1 SELECT * FROM supplier WHERE id_supplier<3;
sqlite> SELECT * FROM supplier1;
1|Петров
2|Зайцев
```

Извлечение данных из таблицы

Создание вложенных запросов

Инструкция ***WHERE*** также позволяет использовать вложенные запросы. При этом вложенный запрос размещается в операторе ***IN***.

ПРИМЕР вывода товаров, предоставленных поставщиком по фамилии Зайцев:

```
sqlite> SELECT * FROM items WHERE id_ supplier IN (
...> SELECT id_ supplier FROM supplier
...> WHERE name_ supplier ='Зайцев');
3|2|микроволновая печь
2|2|пылесос
2|2|ноутбук
```