Лабораторная работа 2.20 Основы работы с SQLite3

Цель работы: *исследовать базовые возможности системы управления базами данных SQLite3.*

Теоретические сведения

Назначение реляционной базы данных и СУБД

Прежде чем говорить о реляционной базе данных и системе управления базами данных (СУБД), надо определиться с тем, что такое база данных вообще.

Понятие базы данных (БД) абстрактно. Конкретными реализациями являются базы данных чего-либо. Например, база данных библиотеки, сайта или база данных магазина, в которой хранятся сведения о сотрудниках, товарах, поставщиках и покупателях.

Удобнее всего такую информацию хранить в таблицах. Например, база данных может состоять из следующих таблиц: "Сотрудники", "Поставщики", "Покупатели". Каждую таблицу будут формировать свои столбцы и строки.

Так таблица "Сотрудники" может включать столбцы "ФИО", "Должность", "Зарплата". Каждая строка этой таблицы будет содержать сведения об одном человеке. Так создаются таблицы в базах данных. Каждая строка называется записью, каждая ячейка строки – полем. Содержание конкретного поля определяется его столбцом.

Следующий вопрос: где хранить таблицы? Очевидно в файлах или даже одном файле. Например, мы можем открыть Excel или другой табличный процессор и заполнить несколько таблиц. Получится база данных. Нужно ли что-то еще?

Представим, что есть большая база данных, скажем, предприятия. Это очень большой файл, его используют множество человек сразу, одни изменяют данные, другие выполняют поиск информации. Табличный процессор не может следить за всеми операциями и правильно их обрабатывать. Кроме того, загружать в память большую БД целиком – не лучшая идея.

Здесь требуется программное обеспечение с другими возможностями. ПО для работы с базами данных называют системами управления базами данных, то есть СУБД.

Таким образом, у нас должен быть файл определенной структуры, содержащий базу данных, а также ПО, обеспечивающее работу с этим файлом.

Стандартным общепринятым языком для описания баз данных и выполнения к ним запросов является язык SQL.

С другой стороны, существует большое количество различных СУБД. Например: SQLite, MySQL, PostgreSQL и другие. Каждая из них имеет некоторые отличия от других, в результате чего накладывает небольшую специфику на используемый SQL, формируя его диалект.

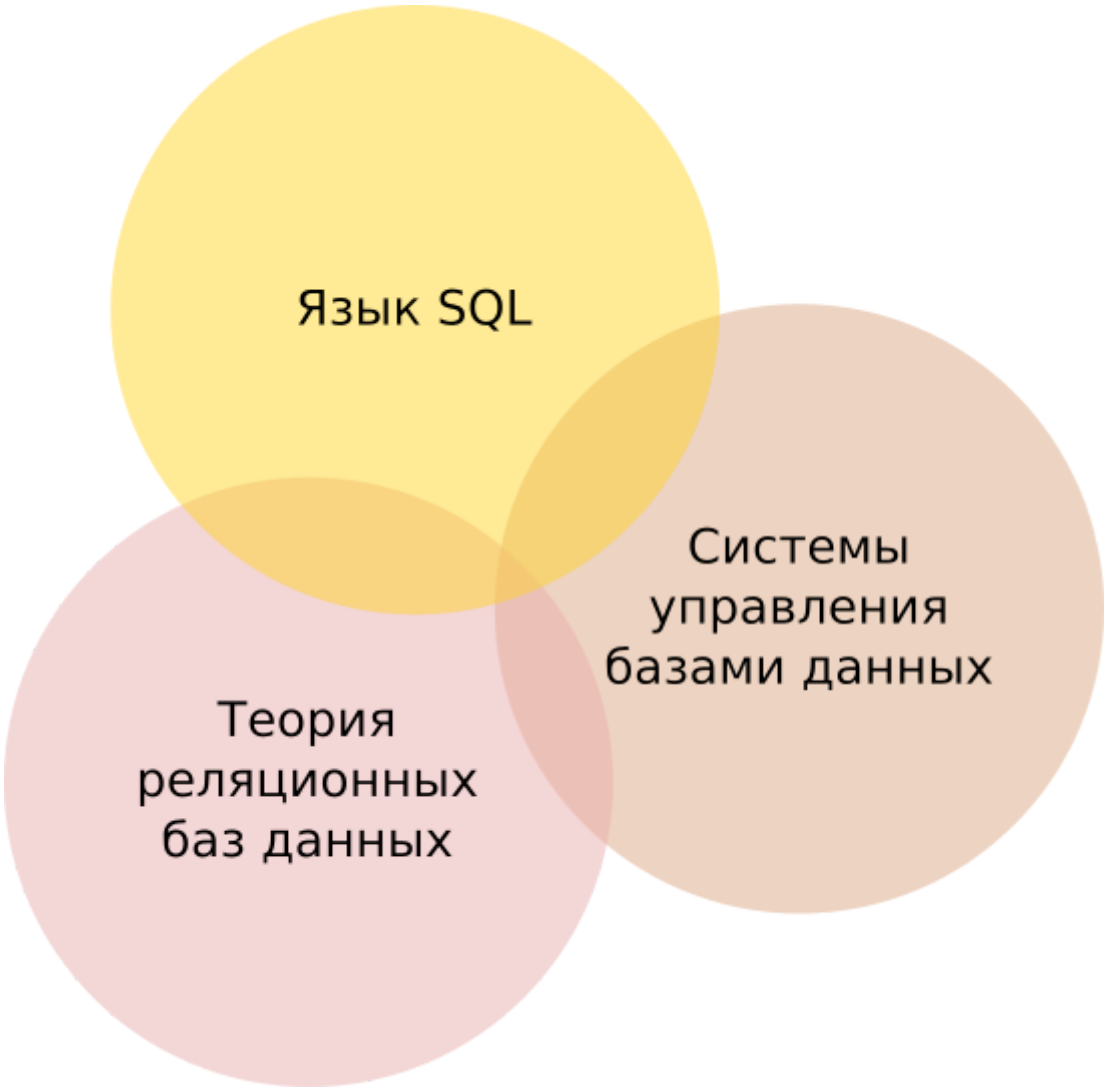
Таким образом, изучая работу с базами данных, вы, с одной стороны, изучаете универсальный SQL, с другой – приобретаете опыт работы с конкретной СУБД. При этом в последствии перейти с одной СУБД на другую относительно легко.

Теперь вернемся к вопросу о том, что такое реляционная базы данных (РБД). Слово "реляция" происходит от "relation", то есть "отношение". Это означает, что в РБД существуют механизмы установления связей между таблицами. Делается это с помощью так называемых первичных и внешних ключей.

Допустим, мы разрабатываем базу данных для сайта. Одна из таблиц будет содержать сведения о страницах сайта. Вторая таблица будет содержать описание разделов сайта. Каждая строка-запись первой таблицы должна в одном из своих полей содержать указание на раздел, к которому принадлежит описываемая этой записью страница.

Таким образом, мы разделяем разные сущности (страницы и разделы) по таблицам, но устанавливаем между ними связь. В последствии используя язык SQL мы сможем, например, создать запрос, который извлечет сведения о конкретном разделе и принадлежащих ему страницах. Хотя такой таблицы исходно нет.

Существуют определенные правила создания реляционных баз данных, их нормализации в основном с целью устранения избыточности. Теория разработки РБД – это целая наука.



Хранение информации в базах данных дает преимущество не только с точки зрения обеспечения к ним быстрого доступа множества процессов. Базы данных, особенно реляционные, позволяют структурировать данные, манипулирования ими и легко наращивать объем.

Можно сказать, что в одной таблице содержатся ассоциированные данные, а в разных таблицах одной БД находятся связанные данные.

Язык SQL

SQL – это язык программирования декларативного типа. В отличие от привычных нам процедурных языков, в которых есть условия, циклы и функции, в декларативных языках подобных алгоритмических конструкций почти нет. Декларативные выражения представляют собой скорее запросы, описание того, что хочет получить человек.

В случае SQL человек формулирует запрос на извлечение или модификацию данных, а алгоритм его выполнения почти полностью ложится на плечи конкретной СУБД. Хотя если один и тот же результат может быть получен с помощью разных запросов, программисту лучше выбрать тот, который создаст меньшую нагрузку на СУБД. То есть программисту желательно иметь представление о том, как работает СУБД.

Запрос производится к таблицам базы данных, результатом обработки запроса также является таблица, которую при желании можно сохранить.

Язык SQL предназначен для создания и изменения реляционных баз данных, а также извлечения из них данных. Другими словами, SQL – это инструмент, с помощью которого человек управляет базой данных. При этом ключевыми операциями являются создание таблиц, добавление записей в таблицы, изменение и удаление записей, выборка записей из таблиц, изменение структуры таблиц.

Однако в процессе развития языка SQL в нем появились новые средства. Стало возможно описывать и хранить такие объекты как индексы, представления, триггеры и процедуры. То есть в современных диалектах SQL есть элементы процедурных языков.

Язык SQL и СУБД обычно не используются сами по себе, а выполняют функцию промежуточного встроенного компонента, обеспечивающего связь между прикладным ПО или программой, которую пишет программист, и базой данных. В языках программирования существуют свои библиотеки, обеспечивающие API для различных СУБД.

Сам язык SQL состоит из операторов, инструкций и вычисляемых функций. Зарезервированные слова, которыми обычно выступают операторы, принято писать заглавными буквами. Однако написание их не прописными, а строчными буквами к ошибке не приводит.

SQLite

SQLite – это система управления базами данных, отличительной особенностью которой является ее встраиваемость в приложения. Это значит, что большинство СУБД являются самостоятельными приложениями, взаимодействие с которыми организовано по принципу клиент-сервер. Программа-клиент посылает запрос на языке SQL, СУБД, которая в том числе может находиться на удаленном компьютере, возвращает результат запроса.

В свою очередь SQLite является написанной на языке C библиотекой, которую динамически или статически подключают к программе. Для большинства языков программирования есть свои привязки (API) для библиотеки SQLite. Так в Python СУБД SQLite импортируют командой `import sqlite3`. Причем модуль sqlite3 входит в стандартную библиотеку языка и не требует отдельной установки.

С другой стороны, библиотеку SQLite можно скачать с сайта разработчика. Она встроена в консольную утилиту sqlite3, с помощью которой можно на чистом SQL создавать базы данных и управлять ими. Также существуют включающие SQLite приложения с графическим интерфейсом пользователя от сторонних разработчиков.

СУБД SQLite во многом поддерживает стандартный SQL. Диалект языка SQL, используемый в SQLite, по синтаксису схож с тем, который используется в PostgreSQL. Однако SQLite накладывает ряд специфических особенностей на SQL.

Следует различать саму SQLite как содержащую СУБД библиотеку и базу данных как таковую. С помощью SQLite создаются базы данных, представляющие собой один кроссплатформенный текстовый файл. Файл базы данных, в отличие от SQLite, не встраивается в приложение, не становится его частью, он существует отдельно. Так можно создать базу данных, пользуясь консольным sqlite3, после чего использовать ее в программе с помощью библиотеки SQLite языка программирования. При этом файл базы данных также хранится на локальной машине.

Приложение, включающее в себя SQLite, использует ее функциональность посредством простых вызовов функций. Поскольку функции вызываются в том же процессе, что работает приложение, вызовы работают быстрее, чем это было бы в случае межпроцессного взаимодействия.

Уход от клиент-серверной модели вовсе не означает, что SQLite – это учебная или урезанная СУБД. Это означает лишь специфику ее применения в роли встраиваемого компонента. Существует множество типов приложений, от "записных книжек" до браузеров и операционных систем, нуждающихся в небольших локальных базах данных.

Поскольку SQLite работает в рамках другого приложения, во время записи файл базы данных блокируется. Таким образом, записывать данные можно только последовательно. В то же время читать базу могут сразу несколько процессов. SQLite – не лучший выбор, если предполагаются частые обращения к БД на запись.

Не считая NULL, SQLite поддерживает всего четыре типа данных – INTEGER, REAL, TEXT и BLOB. Последний тип – это двоичные данные. При этом в столбец, объявленный одним типом, могут записываться данные любого другого. Если SQLite не может преобразовать переданные данные в заявленный для столбца тип, то оставляет их как есть.

В SQLite нет контроля доступа к БД посредством пароля или с помощью оператора GRANT. Доступ контролируется на уровне файловой системы путем разрешений на файл базы данных для пользователей и групп.

В SQLite поддержка внешних ключей по умолчанию отключена, но ее можно включить.

В Ubuntu установить sqlite3 можно командой `sudo apt install sqlite3`. В этом случае утилита вызывается командой `sqlite3`. Также можно скачать с сайта https://sqlite.org архив с последней версией библиотеки, распаковать и вызвать в терминале утилиту:

```
pl@comp:~$ cd sqlite/
pl@comp:~/sqlite$ ls
sqldiff  sqlite3  sqlite3_analyzer
pl@comp:~/sqlite$ ./sqlite3
SQLite version 3.28.0 2019-04-16 19:49:53
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .exit
pl@comp:~/sqlite$
```

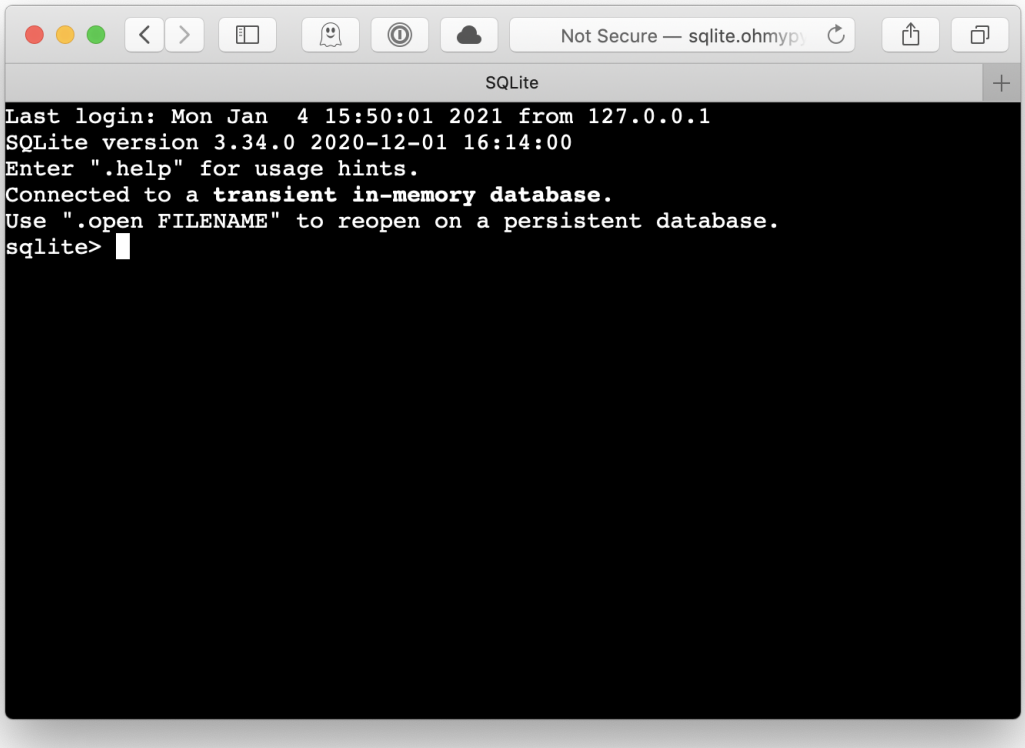
Для операционной системы Windows скачивают свой архив (sqlite-tools-win32-*.zip) и распаковывают. Далее настраивают путь к каталогу, добавляя адрес каталога к переменной PATH (подобное можно сделать и в Linux). Возможно как и в Linux работает вызов утилиты по ее адресу. Android же имеет уже встроенную библиотеку SQLite.

Следует различать операторы и выражения языка SQL и команды самой утилиты sqlite3. Команды утилиты обязательно пишутся с маленькой буквы и начинаются с точки. В конце точка с запятой не ставится. Команды языка SQL заканчиваются точкой с запятой и содержат операторы, которые обычно пишут прописными буквами.

Песочница

Песочница — это SQLite, который работает прямо в браузере: sqlite.ohmypy.ru

Нормально работает только на десктопе, с мобильного использовать не получится.



⚠ **Песочница vs локальная версия**Возможно, на вашем компьютере уже установлен SQLite. Скорее всего, это старая версия, и часть команд из лекций может не работать. Поэтому я рекомендую пройти весь первый модуль в песочнице, чтобы не закопаться сейчас в технических деталях и не растерять мотивацию. Когда закончите модуль, если будет желание — установите свежую версию SQLite по инструкции. Либо оставайтесь в песочнице хоть до конца курса — это неплохой вариант.

Попробуйте повторить шаги в песочнице — все должно работать:

```
sqlite> .mode box
sqlite> create table city (id integer primary key, name text);
sqlite> insert into city (name) values ('Москва'), ('Санкт-Петербург'),
('Новосибирск');
sqlite> select * from city;
┌────┬─────────────────┐
│ id │      name       │
├────┼─────────────────┤
│ 1  │ Москва          │
│ 2  │ Санкт-Петербург │
│ 3  │ Новосибирск     │
└────┴─────────────────┘
```

Песочница держит базу данных в памяти и не записывает на диск — это значит, что сохранить изменения не получится. Когда вы зайдете в следующий раз — база будет пустой. Особенности облачного доступа, увы. Чтобы вам не выполнять каждый раз команды с нуля, я подготовил специальные предзаполненные базы для каждого урока. Когда потребуется — буду давать ссылки в песочницу прямо на них.

Создание базы данных и таблиц

Создание и открытие базы данных

С помощью sqlite3 создать или открыть существующую базу данных можно двумя способами. Во-первых, при вызове утилиты sqlite3 в качестве аргумента можно указать имя базы данных. Если БД существует, она будет открыта. Если ее нет, она будет создана и открыта.

```
$ sqlite3 your.db
```

Во вторых, работая в самой программе, можно выполнить команду

```
.open your.db
```

Выяснить, какая база данных является текущей, можно с помощью команды `.databases` утилиты sqlite3. Если вы работаете с одной БД, а потом открываете другую, то текущей становится вторая БД.

```
pl@comp:~/sqlite$ ls
sqldiff  sqlite3  sqlite3_analyzer
pl@comp:~/sqlite$ ./sqlite3 first.db
SQLite version 3.28.0 2019-04-16 19:49:53
Enter ".help" for usage hints.
sqlite> .databases
main: /home/pl/sqlite/first.db
sqlite> .open second.db
sqlite> .databases
main: /home/pl/sqlite/second.db
sqlite> .quit
pl@comp:~/sqlite$ ls
first.db  second.db  sqldiff  sqlite3  sqlite3_analyzer
pl@comp:~/sqlite$
```

Создание и удаление таблицы

Таблицы базы данных создаются с помощью директивы CREATE TABLE языка SQL. После CREATE TABLE идет имя таблицы, после которого в скобках перечисляются имена столбцов и их тип:

```
sqlite> CREATE TABLE pages (
   ...> title TEXT,
   ...> url TEXT,
   ...> theme INTEGER,
   ...> num INTEGER);
```

Имена как таблицы, так и столбцов принято писать строчными буквами. Если имя включает два слова, обычно их соединяют с помощью нижнего подчеркивания. Команды можно писать в одну строку, а не так, как показано выше.

Чтобы увидеть список таблиц базы данных используется команда `.tables`.

Для удаления целой таблицы из базы данных используется директива DROP TABLE, после которой идет имя удаляемой таблицы.

```
sqlite> .open first.db
sqlite> .tables
sqlite> CREATE TABLE pages (
   ...> title TEXT,
   ...> url TEXT,
   ...> theme INTEGER,
   ...> num INTEGER);
sqlite> .tables
pages
sqlite> DROP TABLE pages;
sqlite> .tables
sqlite>
```

Первичный ключи и автоинкремент

Для реляционных баз данных важно, чтобы каждую запись-строку таблицы можно было однозначно идентифицировать. То есть в таблицах не должно быть полностью совпадающих строк. Записи должны отличаться хотя бы по одному полю.

С этой целью принято создавать дополнительное поле, которое часто называют ID или подобно. В базах данных под Android по соглашению столбец для уникального идентификатора записей называют _id.

```
sqlite> CREATE TABLE pages (
   ...> _id INTEGER,
   ...> title TEXT,
   ...> url TEXT,
   ...> theme INTEGER,
   ...> num INTEGER);
```

При таком создании таблицы следить за уникальностью поля _id каждой записи должен будет человек. Для SQLite столбец _id ничем не отличается от любого другого. Мы вполне можем сделать несколько записей с одинаковым ID.

Чтобы исключить возможность ввода одинаковых идентификаторов, столбец ID назначают **первичным ключом**. PRIMARY KEY – ограничитель, который заставляет СУБД проверять уникальность значения данного поля у каждой добавляемой записи.

```
sqlite> CREATE TABLE pages (
   ...> _id INTEGER PRIMARY KEY,
   ...> title TEXT,
   ...> url TEXT,
   ...> theme INTEGER,
   ...> num INTEGER);
```

Если нам не важно, какие конкретно идентификаторы будут записываться в поле _id, а важна только уникальность поля, следует назначить полю еще один ограничитель – **автоинкремент** – AUTOINCREMENT.

```
sqlite> CREATE TABLE pages (
   ...> _id INTEGER PRIMARY KEY AUTOINCREMENT,
   ...> title TEXT,
   ...> url TEXT,
   ...> theme INTEGER,
   ...> num INTEGER);
```

В этом случае SQLite будет сам записывать в поле уникальное целочисленное значение по нарастающей от записи к записи. Поскольку это поле заполняется автоматически, то при добавлении записи в таблицу его игнорируют.

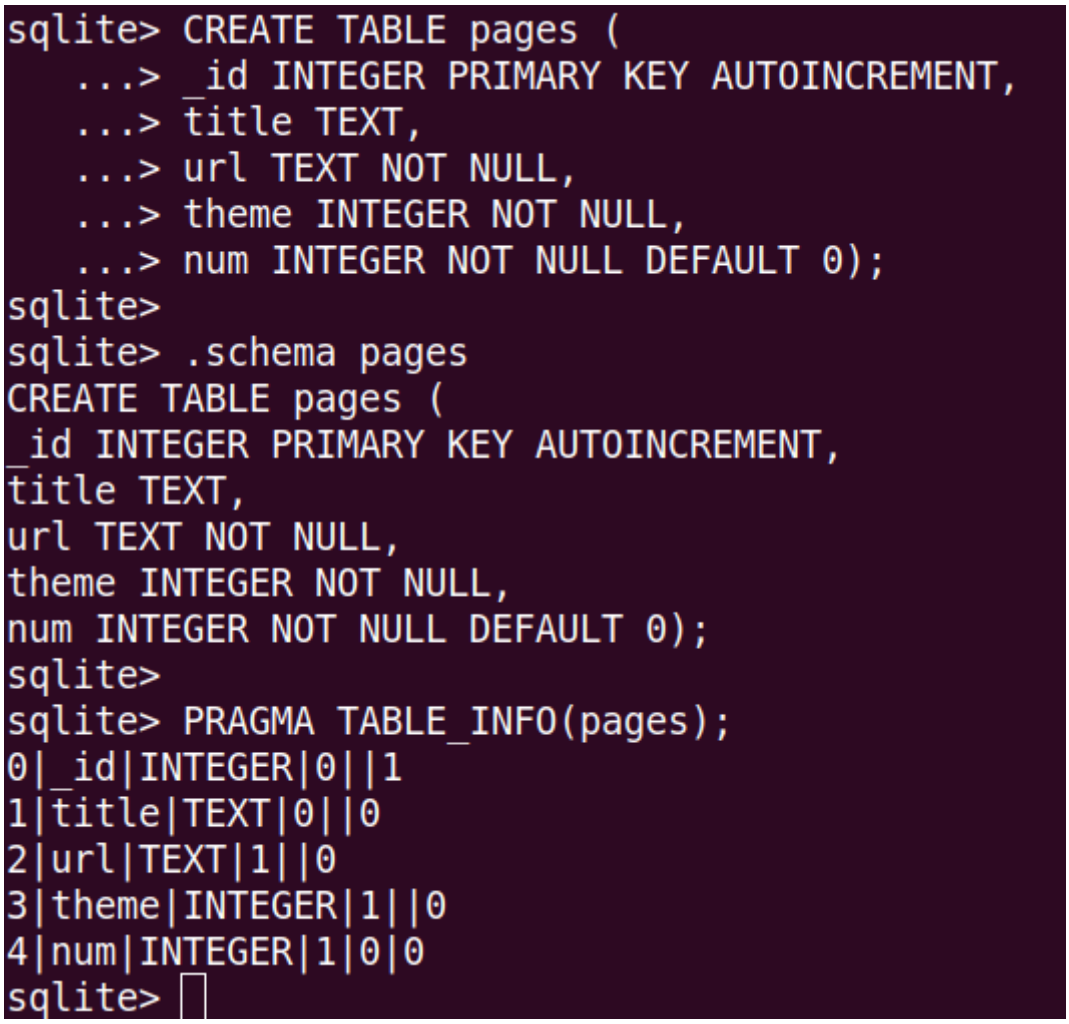
NOT NULL и DEFAULT

Ограничитель NOT NULL используют, чтобы запретить оставление поля пустым. По умолчанию, если поле не является первичным ключом, в него можно не помещать данные. В этом случае полю будет присвоено значение NULL. В случае NOT NULL вы не сможете добавить запись, не указав значения соответствующего поля.

Однако, добавив ограничитель DEFAULT, вы сможете не указывать значение. DEFAULT задает значение по умолчанию. В результате, когда данные в поле не передаются при добавлении записи, поле заполняется тем, что было указано по умолчанию.

Допустим, в таблице поля url, theme и num не должны быть пустыми. При этом если значение для num не передается, то полю присваивается 0. В этом случае команда для создания таблицы будет такой:

```
sqlite> CREATE TABLE pages (
   ...> _id INTEGER PRIMARY KEY AUTOINCREMENT,
   ...> title TEXT,
   ...> url TEXT NOT NULL,
   ...> theme INTEGER NOT NULL,
   ...> num INTEGER NOT NULL DEFAULT 0);
```



С помощью команд `.schema` и PRAGMA TABLE_INFO() можно посмотреть схему таблицы.

Внешний ключ

С помощью внешнего ключа устанавливается связь между записями разных таблиц. Внешний ключ в одной таблице для другой является первичным. Внешние ключи не обязаны быть уникальными. В одной таблице может быть несколько внешних ключей, при этом каждый будет устанавливать связь со своей таблицей, где он является первичным.

Представим, что у нас есть вторая таблица, в которой перечислены темы-разделы, а их номера являются уникальными идентификаторами.

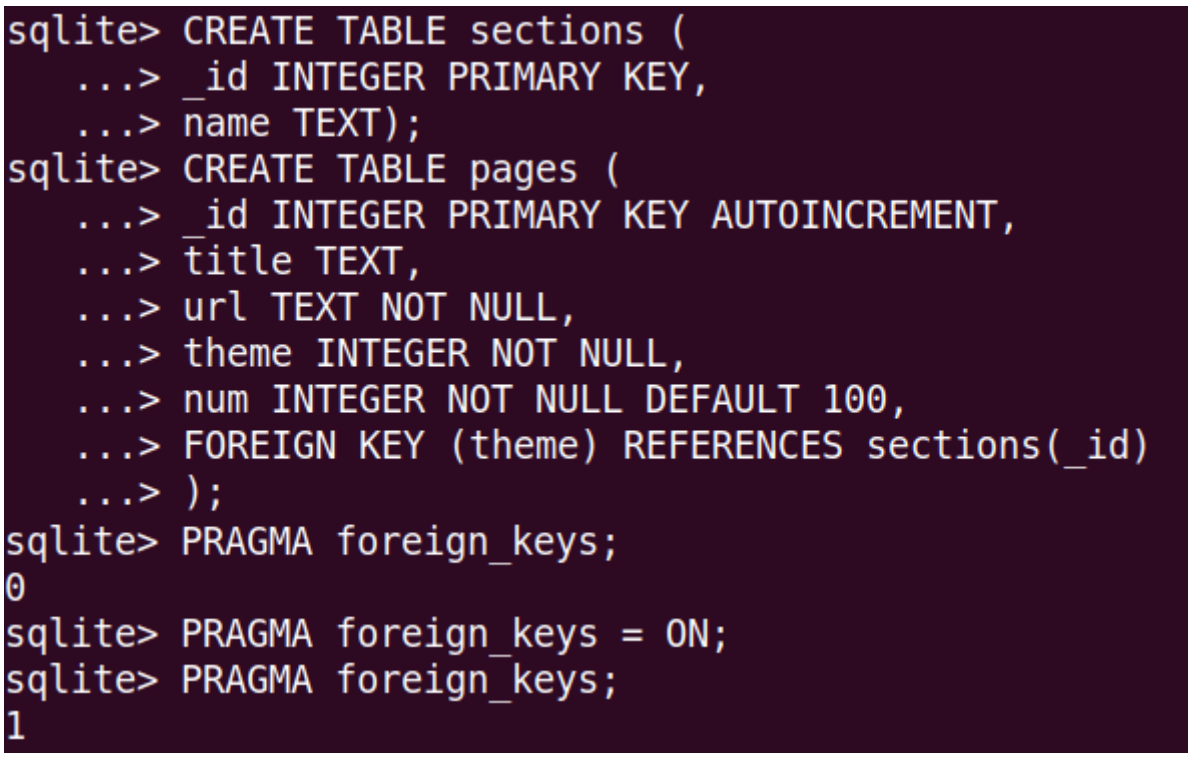
```
sqlite> CREATE TABLE sections (
   ...> _id INTEGER PRIMARY KEY,
   ...> name TEXT);
```

Тогда в первой таблице в столбце theme следует хранить номера тем – их ID, взятые из второй таблицы. Это будут внешние ключи, представляющие собой первичные в таблице с разделами. Внешние ключи уникальными не будут, так как разные страницы могут принадлежать к одной и той же теме.

```
sqlite> CREATE TABLE pages (
   ...> _id INTEGER PRIMARY KEY AUTOINCREMENT,
   ...> title TEXT,
   ...> url TEXT NOT NULL,
   ...> theme INTEGER NOT NULL,
   ...> num INTEGER NOT NULL DEFAULT 100,
   ...> FOREIGN KEY (theme) REFERENCES sections(_id)
   ...> );
```

FOREIGN KEY является ограничителем, так как не дает нам записать в поле столбца theme какое-либо иное значение, которое не встречается в качестве первичного ключа в таблице sections. Однако в SQLite поддержка внешнего ключа по умолчанию отключена. Поэтому, даже назначив столбец внешним ключом, вы сможете записывать в его поля любые значения.

Чтобы включить поддержку внешних ключей в sqlite3, надо выполнить команду `PRAGMA foreign_keys = ON;`. После этого добавить в таблицу запись, в которой внешний ключ не совпадает ни с одним первичным из другой таблицы, не получится.



INSERT и SELECT – добавление и выборка данных

В теории реляционных баз данных часто фигурирует акроним CRUD, образованный от слов Create, Read, Update, Delete. Обозначает он тот факт, что данные в БД можно

- добавлять,
- считывать их оттуда,
- изменять,
- удалять.

То есть CRUD обозначает четыре базовых действия с данными, хранимыми в таблицах БД.

При этом операторы языка SQL не обязательно совпадают со словами create, read, update, delete. Так в SQL нет оператора READ, вместо него используется SELECT. Записи-строки в таблицу вставляются не с помощью CREATE, а оператором INSERT.

Оператор INSERT

С помощью оператора INSERT языка SQL выполняется вставка данных в таблицу. Синтаксис команды:

```
INSERT INTO <table_name>
(<column_name1>, <column_name2>, ...)
VALUES
(<value1>, <value2>, …);
```

После INSERT INTO указывается имя таблицы, после в скобках перечисляются столбцы. После слова VALUES перечисляются данные, вставляемые в поля столбцов. Например:

```
sqlite> INSERT INTO sections
   ...> (_id, name) VALUES
   ...> (1, 'information');
```

При этом столбцы не обязательно должны перечисляться в том же порядке, в котором задавались при создании таблицы:

```
sqlite> INSERT INTO sections
   ...> (name, _id)
   ...> VALUES
   ...> ('Boolean Algebra', 3);
```

Однако, поскольку можно вообще не указывать имена столбцов, в этом случае при указании значений их порядок должен совпадать со схемой таблицы:

```
sqlite> INSERT INTO sections
   ...> VALUES (2, 'Digital Systems');
```

```
pl@comp:~/sqlite$ ./sqlite3 inf1.db
SQLite version 3.28.0 2019-04-16 19:49:53
Enter ".help" for usage hints.
sqlite> .tables
pages     sections
sqlite> .schema sections
CREATE TABLE sections (
_id INTEGER PRIMARY KEY,
name TEXT);
sqlite> INSERT INTO sections ( id, name) VALUES (1, 'Information');
sqlite> INSERT INTO sections VALUES (2, 'Digital Systems');
sqlite> INSERT INTO sections (name, _id) VALUES ('Boolean Algebra', 3);
sqlite>
sqlite> SELECT * FROM sections;
1|Information
2|Digital Systems
3|Boolean Algebra
sqlite>
```

Если в таблице есть столбцы с ограничителями AUTOINCREMENT и DEFAULT, то есть автозаполняемые, значения для них можно не указывать. Однако в этом случае должен быть указан перечень столбцов для остальных значений:

```
sqlite> INSERT INTO pages VALUES
   ...> (1, 'What is Information',
   ...> 'information', 1, 1);
sqlite> INSERT INTO pages
   ...> (title, url, theme, num)
   ...> VALUES
   ...> ('Amount of Information',
   ...> 'amount-information', 1, 2);
```

Здесь в первом случае мы вручную задаем значение для поля _id и передаем данные во все остальные поля. Поэтому можем опустить перечисление столбцов. Во втором случае СУБД будет самостоятельно заполнять поле _id. Во избежание неоднозначности мы должны перечислить остальные столбцы.

```
sqlite> PRAGMA foreign_keys = ON;
sqlite> .schema pages
CREATE TABLE pages (
_id INTEGER PRIMARY KEY AUTOINCREMENT,
title TEXT,
url TEXT NOT NULL,
theme INTEGER NOT NULL,
num INTEGER NOT NULL DEFAULT 100,
FOREIGN KEY (theme) REFERENCES sections(_id)
);
sqlite> INSERT INTO pages VALUES
   ...> (1, 'What is Information', 'information', 1, 1);
sqlite> INSERT INTO pages (title, url, theme, num) VALUES
   ...> ('Amount of Information', 'amount-information', 1, 2);
sqlite> SELECT * FROM pages;
1|What is Information|information|1|1
3|Amount of Information|amount-information|1|2
sqlite>
```

Обратите внимание, в SQLite мы должны включать поддержку внешнего ключа, чтобы работал ограничитель FOREIGN KEY и не давал нам добавлять записи с номерами тем, которых нет в таблице sections.

Причина, по которой СУБД присвоила второй записи идентификатор 3, а не 2, заключается в том, что раньше в таблицу уже помещались данные, но были удалены.

Оператор SELECT

С помощью оператора SELECT осуществляется выборочный просмотр данных из таблицы. В простейшем случае оператор имеет следующий синтаксис, где вместо <table_name> указывается имя таблицы:

```
SELECT * FROM <table_name>;
```

Такая команда отображает значения всех столбцов и строк заданной таблицы. На выборку всех столбцов указывает звездочка после слова SELECT. А все строки будут выбраны потому, что после имени таблицы нет оператора WHERE языка SQL. WHERE позволяет задавать условие, согласно которому отображаются только удовлетворяющие ему строки.

Утилита sqlite3 позволяет отображать данные таблицы в различных форматах, которые задаются с помощью команды .mode, после которой указывается необходимый режим. Команда `.mode` без аргумента показывает текущий режим.

Обратите внимание, на скринах выше разделителями между столбцами является вертикальная черта. Это режим list. Посмотреть все доступные режимы можно с помощью команды `.help .mode`.

```
sqlite> .help .mode
.mode MODE ?TABLE?       Set output mode
   MODE is one of:
     ascii    Columns/rows delimited by 0x1F and 0x1E
     csv      Comma-separated values
     column   Left-aligned columns.  (See .width)
     html     HTML <table> code
     insert   SQL insert statements for TABLE
     line     One value per line
     list     Values delimited by "|"
     quote    Escape answers as for SQL
     tabs     Tab-separated values
     tcl      TCL list elements
sqlite> .mode csv
sqlite> SELECT * FROM pages;
1,"What is Information",information,1,1
3,"Amount of Information",amount-information,1,2
sqlite> .mode html
sqlite> SELECT * FROM pages;
<TR><TD>1</TD>
<TD>What is Information</TD>
<TD>information</TD>
<TD>1</TD>
<TD>1</TD>
</TR>
<TR><TD>3</TD>
<TD>Amount of Information</TD>
<TD>amount-information</TD>
<TD>1</TD>
<TD>2</TD>
</TR>
```

Чтобы отобразить заголовки используется команда `.header on`. Они хорошо сочетаются с режимом column. Для отключения заголовков используется `.header off`:

```
sqlite> .mode column
sqlite> .header on
sqlite> SELECT * FROM pages;
_id         title                url          theme       num
----------  -------------------  -----------  ----------  -------
1           What is Information  information  1           1
3           Amount of Informati  amount-info  1           2
sqlite>
```

Когда требуется просмотреть только отдельные столбцы, то вместо звездочки их имена перечисляются через запятую:

```
sqlite> SELECT title, theme FROM pages;
What is Information|1
Amount of Information|1
```

WHERE

Условие WHERE используется не только с оператором SELECT, также с UPDATE и DELETE. С помощью WHERE определяются строки, которые будут выбраны, обновлены или удалены. По сути это фильтр.

После ключевого слова WHERE записывается логическое выражение, которое может быть как простым (содержащим операторы = или ==, >, <, >=, <=, !=, BETWEEN), так и сложным (AND, OR, NOT, IN, NOT IN). Примеры:

```
sqlite> SELECT * FROM pages
   ...> WHERE _id == 3;

sqlite> SELECT * FROM pages WHERE
   ...> theme == 2 AND num == 100;

sqlite> SELECT * FROM pages WHERE
   ...> theme <= 2;
```

```
sqlite> SELECT * FROM pages;
1|What is Information|information|1|1
3|Amount of Information|amount-information|1|2
4|Binary System|binary|2|100
5|Octal System|octal|2|1
6|Lows of Logic Algebra|logic-low|3|100
sqlite> SELECT * FROM pages WHERE _id == 3;
3|Amount of Information|amount-information|1|2
sqlite> SELECT * FROM pages WHERE theme == 2 AND num == 100;
4|Binary System|binary|2|100
sqlite> SELECT * FROM pages WHERE theme <= 2;
1|What is Information|information|1|1
3|Amount of Information|amount-information|1|2
4|Binary System|binary|2|100
5|Octal System|octal|2|1
sqlite>
```

Примеры с BETWEEN и IN:

```
sqlite> SELECT _id, title
   ...> FROM pages WHERE
   ...> _id BETWEEN 2 AND 8;
3|Amount of Information
4|Binary System
5|Octal System
6|Lows of Logic Algebra

sqlite> SELECT _id, title
   ...> FROM pages WHERE
   ...> _id IN (1,2);
```

```
1|What is Information

sqlite> SELECT _id, title
   ...> FROM pages WHERE
   ...> _id NOT IN (1,3);
4|Binary System
5|Octal System
6|Lows of Logic Algebra
```

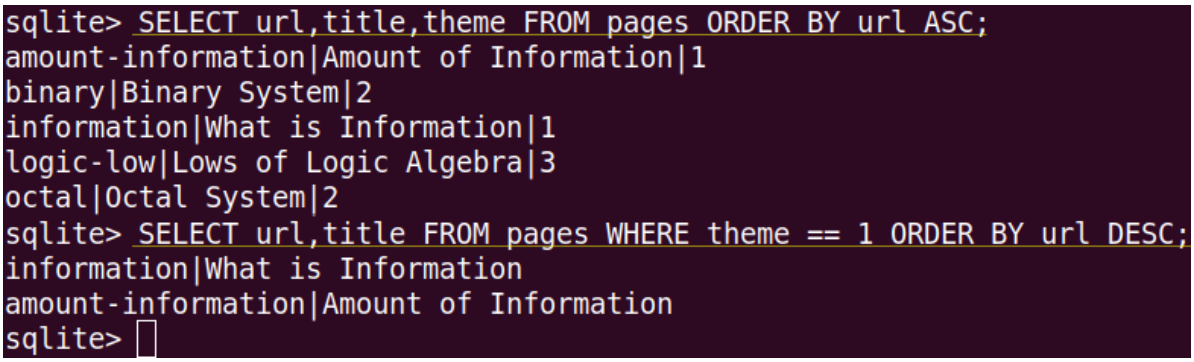
ORDER BY

При выводе данных их можно не только фильтровать с помощью WHERE, но и сортировать по возрастанию или убыванию с помощью оператора ORDER BY.

```
sqlite> SELECT url, title, theme
   ...> FROM pages
   ...> ORDER BY url ASC;
amount-information|Amount of Information|1
binary|Binary System|2
information|What is Information|1
logic-low|Lows of Logic Algebra|3
octal|Octal System|2

sqlite> SELECT url, title FROM pages
   ...> WHERE theme == 1
   ...> ORDER BY url DESC;
information|What is Information
amount-information|Amount of Information
```

ASC – сортировка от меньшего значения к большему. DESC – сортировка от большего значения к меньшему.



UPDATE и DELETE – обновление и удаление данных

Операторы UPDATE и DELETE надо использовать с осторожностью. Если с помощью WHERE не заданы обновляемые или удаляемые строки, будут обновлены или удалены все записи таблицы. Поэтому данные команды почти всегда используются совместно с WHERE.

UPDATE ... SET – обновление полей записи

Синтаксис команды:

```
UPDATE имя_таблицы
SET имя_столбца = новое_значение
WHERE условие;
```

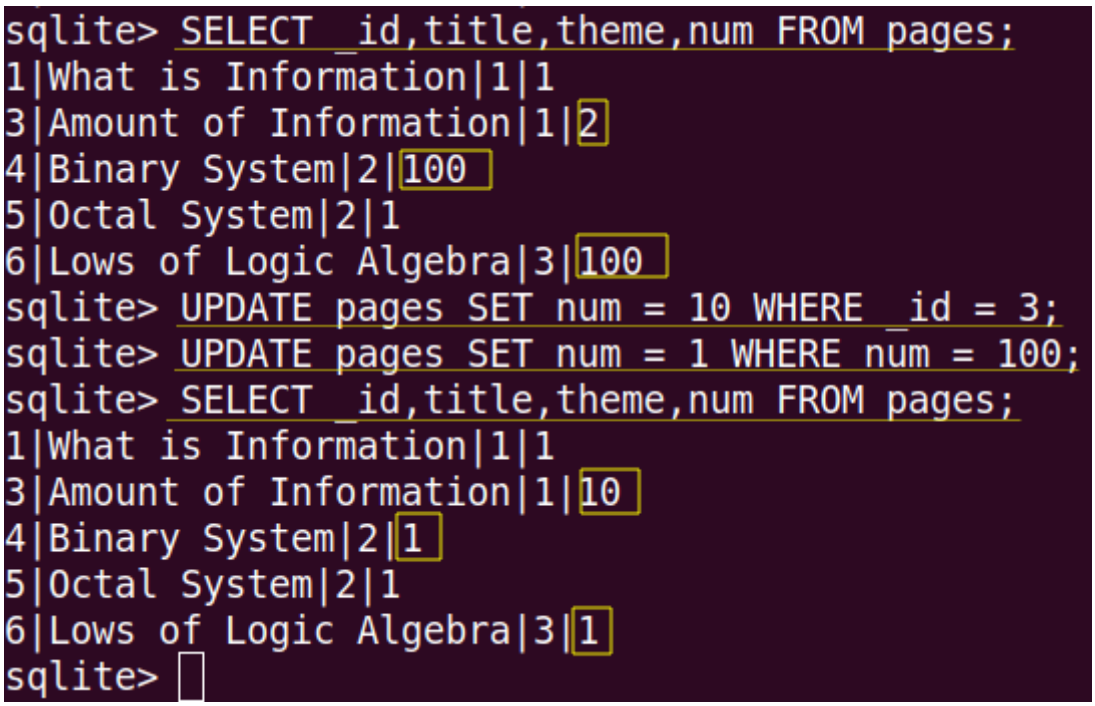
Чаще всего условием является ID конкретной записи, в результате чего обновляется только она:

```
sqlite> UPDATE pages SET num = 10
   ...> WHERE _id = 3;
```

Однако можно указывать другие столбцы:

```
sqlite> UPDATE pages SET num = 1
   ...> WHERE num = 100;
```

При этом будут обновлены все записи, в которых указанное в условии поле имеет соответствующее значение.



DELETE FROM – удаление записей таблицы

Синтаксис команды удаления из таблицы одной или нескольких записей:

```
DELETE FROM имя_таблицы WHERE условие;
```

Без WHERE будут удалены все строки, однако сама таблица останется. Она будет пустой. Для удаления самой таблицы из базы данных используется команда `DROP TABLE имя_таблицы;`.

Примеры:

```
sqlite> DELETE FROM pages WHERE _id = 6;
sqlite> DELETE FROM pages WHERE theme = 2;
```

```
sqlite> SELECT _id,title,theme,num FROM pages;
1|What is Information|1|1
3|Amount of Information|1|10
4|Binary System|2|1
5|Octal System|2|1
6|Lows of Logic Algebra|3|1
sqlite> DELETE FROM pages WHERE _id = 6;
sqlite> DELETE FROM pages WHERE theme = 2;
sqlite> SELECT _id,title,theme,num FROM pages;
1|What is Information|1|1
3|Amount of Information|1|10
sqlite>
```

Агрегирование и группировка

Как быть, если надо посчитать общее количество строк таблицы или найти запись, содержащую максимальное значение, или посчитать сумму значений столбца? Для этих целей в языке SQL предусмотрены различные функции агрегирования данных. Наиболее используемые – count(), sum(), avr(), min(), max(). Используют их совместно с оператором SELECT.

Вывод количества столбцов таблицы:

```
sqlite> SELECT count() FROM pages;
```

Поиск максимального ID:

```
sqlite> SELECT max(_id) FROM pages;
```

Количество различных вариантов значения столбца:

```
sqlite> SELECT count(DISTINCT theme)
   ...> FROM pages;
```

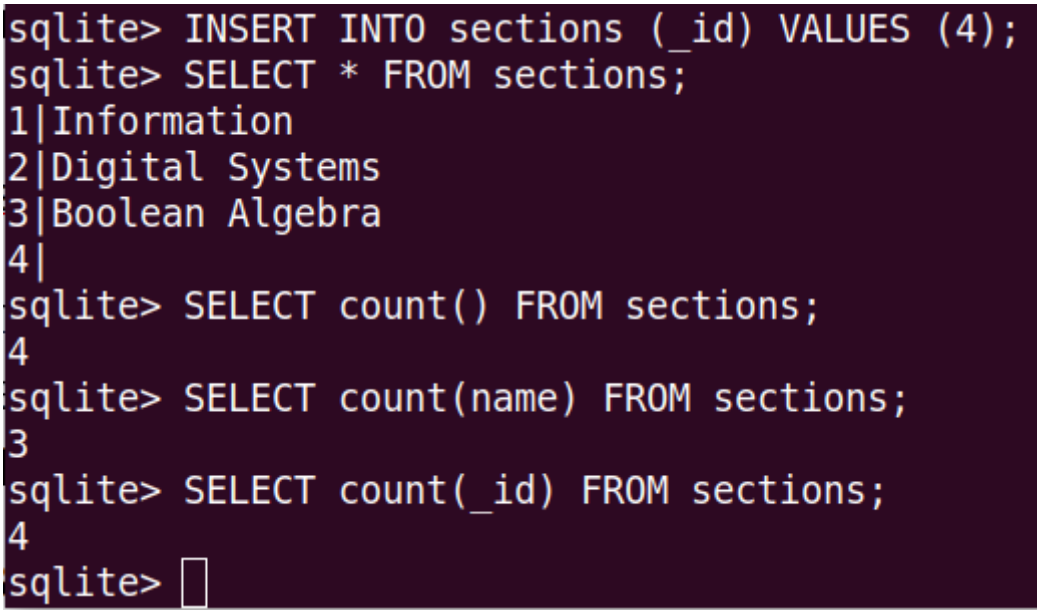
```
sqlite> SELECT title,theme FROM pages;
What is Information|1
Amount of Information|1
Binary System|2
Boolean Lows|3
sqlite> SELECT count() FROM pages;
4
sqlite> SELECT count(DISTINCT theme) FROM pages;
3
sqlite> SELECT max(_id) FROM pages;
8
sqlite> SELECT max(_id),title FROM pages;
8|Boolean Lows
sqlite>
```

На скрине команда с DISTINCT возвращает 3 потому, что у нас встречается три разных значения в столбце theme – это значения 1, 2 и 3. Тема 1 встречается у двух записей, но благодаря агрегированию они учитываются как одна.

Вообще DISTINCT перед именем столбца выводит его различающиеся значения. Например, мы хотим узнать, какие темы используются в таблице pages:

```
sqlite> SELECT DISTINCT theme FROM pages;
1
2
3
```

Если в функцию count() передается просто имя столбца, например count(theme), то она возвращает количество записей с не NULL значениями. Если в указанном столбце нигде не встречается NULL, то результат будет совпадать с общим количеством записей.



Перед агрегированием можно выполнить фильтрацию. Данная команда посчитает количество страниц определенной темы:

```
sqlite> SELECT count() FROM pages
   ...> WHERE theme = 1;
2
```

Обратим внимание, в SQL сначала выполняется фильтрация, то есть оператор WHERE. И только после этого агрегирование, то есть функция count(). Таким образом, из всей таблицы сначала фильтруются записи с темой 1. После этого считается их количество.

Если было бы наоборот, то приведенная выше команда не сработала, потому что функция count() вернула бы число-количество строк таблицы, и фильтровать из него было бы уже нечего.

В SQL кроме функций агрегирования есть оператор GROUP BY, который выполняет группировку записей по вариациям заданного поля. То есть GROUP BY группирует все записи, в которых встречается одно и то же значение в указанном столбце, в одну строку. Так следующая команда выведет не количество тем, а их номера:

```
sqlite> SELECT theme FROM pages
   ...> GROUP BY theme;
1
2
3
```

Таким образом мы можем узнать, на какие темы имеются страницы в базе данных.

Часто группировка и агрегирование фигурируют в одной команде. Например, надо выяснить количество записей в каждой группе:

```
sqlite> SELECT theme,count()
   ...> FROM pages
   ...> GROUP BY theme;
```

Здесь будут выведены два столбца. В первом будет номер темы, во втором – количество страниц темы. Функция count() будет выполняться после группировки по темам. Она будет считать количество записей в каждой теме.

```
sqlite> SELECT title,theme FROM pages;
What is Information|1
Amount of Information|1
Binary System|2
Boolean Lows|3
sqlite> SELECT theme FROM pages GROUP BY theme;
1
2
3
sqlite> .header on
sqlite> .mode column
sqlite> SELECT theme,count() FROM pages GROUP BY theme;
theme       count()
----------  ----------
1           2
2           1
3           1
sqlite> SELECT theme, count() AS num FROM pages GROUP BY theme;
theme       num
----------  ----------
1           2
2           1
3           1
sqlite>
```

На скрине в последней команде используется переименование столбца count() в столбец num. Делается это с помощью ключевого слова AS.

Пример совместного использования группировки и функции max():

```
sqlite> SELECT theme, max(num)
   ...> FROM pages GROUP BY theme;
theme       max(num)
----------  ----------
1           10
2           100
3           100
```

Здесь сначала происходит группировка записей по темам. Потом в каждой группе ищется запись с максимальным значением столбца num.

Для вывода можно указывается столбец таблицы, группировка по которому не выполняется:

```
sqlite> SELECT title,theme,max(num)
   ...> FROM pages GROUP BY theme;
Amount of Information|1|10
Binary System|2|100
Boolean Lows|3|100
```

В примере выше это поле title. Однако подобное не всегда уместно:

```
sqlite> SELECT title, theme, count()
   ...> FROM pages GROUP BY theme;
What is Information|1|2
Binary System|2|1
Boolean Lows|3|1
```

У нас две страницы первой темы, но поскольку мы выводим в одной строке целую группу, название другой страницы первой темы не показано. Таким образом, хотя можно указывать столбцы по которым группировка не выполняется, иногда в этом нет смысла.

JOIN – соединение таблиц

При выборке данных из таблицы pages мы можем увидеть номер темы, к которой относится та или иная страницы, но не название темы. Чтобы увидеть названия тем, надо вывести вторую таблицу.

```
sqlite> .header on
sqlite> .mode column
sqlite> SELECT title,theme FROM pages;
title                theme
-------------------  ----------
What is Information  1
Amount of Informati  1
Binary System        2
Boolean Lows         3
sqlite> SELECT * FROM sections;
_id         name
----------  -----------
1           Information
2           Digital Sys
3           Boolean Alg
4
```

Как получить сводную таблицу, в которой для каждой страницы будет указано название ее темы? Фактически нам надо вывести два столбца. Столбец title из таблицы pages и столбец name из таблицы sections.

При этом должно быть выполнено сопоставление по номеру темы, который в одной таблице является внешним ключом, а в другой – первичным. Так, если в записи таблицы pages в поле theme указан внешний ключ 1, то из таблицы sections должна выбираться запись, чье значение поля первичного ключа _id равно 1. Далее из этой записи берется значение поля name.

В SQL для соединения данных из разных таблиц используется оператор JOIN. В случае с нашим примером запрос будет выглядеть так:

```
sqlite> SELECT pages.title,
   ...> sections.name AS theme
   ...> FROM pages JOIN sections
   ...> ON pages.theme == sections._id;
```

```
sqlite> SELECT pages.title, sections.name AS theme
   ...> FROM pages JOIN sections
   ...> ON pages.theme == sections._id;
title                theme
-------------------  -----------
What is Information  Information
Amount of Informati  Information
Binary System        Digital Sys
Boolean Lows         Boolean Alg
```

Подвыражение `AS theme` можно опустить. Тогда в качестве заголовка столбца будет указано его оригинальное имя – name.

После SELECT указываются столбцы, которые необходимо вывести. Перед именем столбца пишется имя таблицы. Указывать таблицу не обязательно, если имя столбца уникальное:

```
sqlite> SELECT title, name
   ...> FROM pages JOIN sections
   ...> ON theme = sections._id;
```

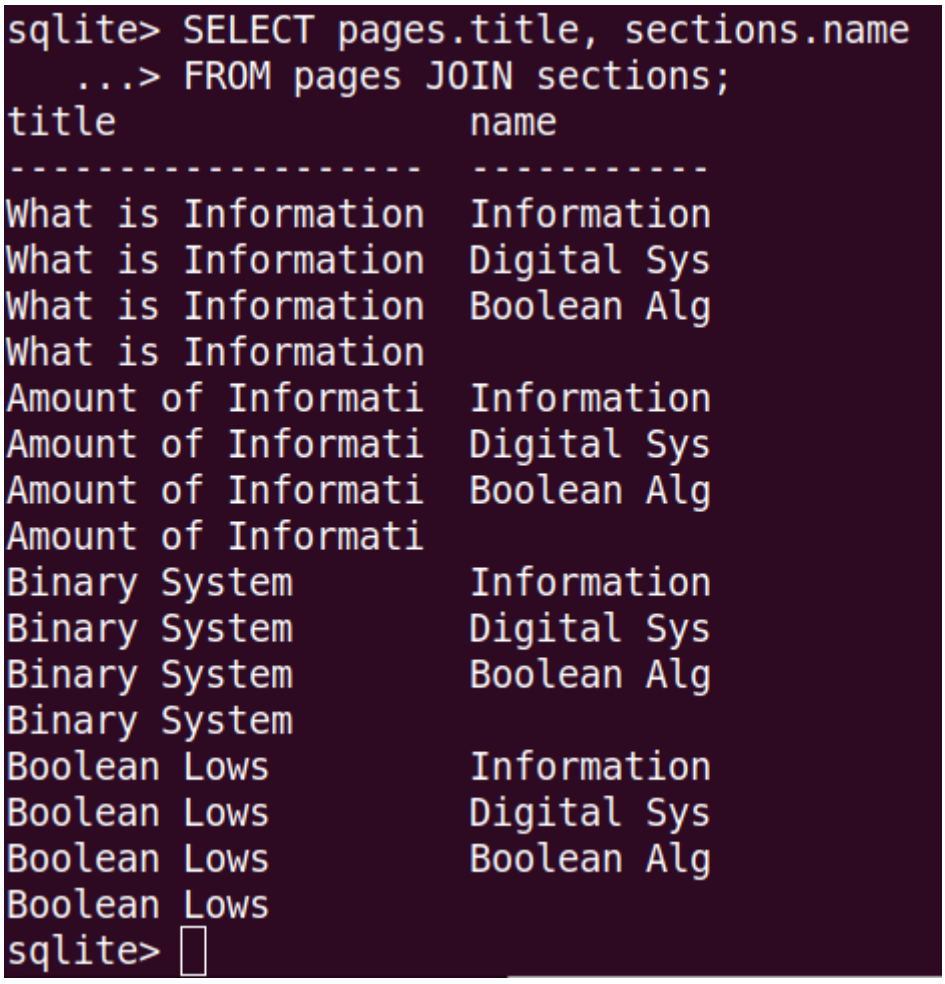
Здесь имя таблицы используется только с _id, так как столбец с таким именем есть в обоих таблицах.

Если после SELECT будет стоять звездочка, будут выведены все столбцы из обоих таблиц.

После FROM указываются обе сводимые таблицы через JOIN. В данном случае неважно, какую указывать до JOIN, какую после.

После ключевого слова ON записывается условие сведения. Условие сообщает, как соединять строки разных таблиц. В данном случае каждая запись из таблицы pages дополняется полями той записи из таблицы sections, чье поле _id содержит такое же значение, какое содержит поле theme таблицы pages.

Если записать команду без части ON, то каждая строка первой таблицы будет соединена по очереди со всеми строками второй. В сводной таблице каждое соединение будет отдельной записью.



Однако если часть ON заменить на WHERE с тем же условием, то соединение таблиц вернет нужный нам результат.

```
sqlite> SELECT pages.title, sections.name
   ...> FROM sections JOIN pages
   ...> WHERE pages.theme == sections._id;
title                name
-------------------  -----------
What is Information  Information
Amount of Informati  Information
Binary System        Digital Sys
Boolean Lows         Boolean Alg
```

На самом деле здесь выполняется фильтрация результата предыдущего примера.

JOIN писать не обязательно. После FROM таблицы можно перечислить через запятую (это верно как при использовании WHERE, так и ON):

```
sqlite> SELECT pages.title, sections.name
   ...> FROM pages, sections
   ...> WHERE pages.theme == sections._id;
```

Можно комбинировать WHERE и JOIN ON. Например, мы хотим вывести страницы только второй и третьей тем:

```
sqlite> SELECT pages.title, sections.name
   ...> FROM pages JOIN sections
   ...> ON pages.theme == sections._id
   ...> WHERE pages.theme == 2
   ...> OR pages.theme == 3;
title          name
-------------  ---------------
Binary System  Digital Systems
Boolean Lows   Boolean Algebra
```

Соединение можно использовать совместно с группировкой. Узнаем, сколько в каждой теме статей:

```
sqlite> SELECT sections.name AS theme,
   ...> count() AS qty_articles
   ...> FROM pages JOIN sections
   ...> ON pages.theme == sections._id
   ...> GROUP BY sections.name
   ...> ORDER BY sections._id;
theme        qty_articles
-----------  ------------
Information  2
Digital Sys  1
Boolean Alg  1
```

В этом запросе сначала была получена сводная таблица, в которой была выполнена группировка по столбцу name и с помощью функции count() посчитано количество записей в каждой группе.

Нормализация

Нормализация – центральная идея реляционных баз данных. Нормализация – это процесс разработки базы данных с учетом так называемых нормальных форм. Каждая нормальная форма представляет собой правило, соблюдая которое в базе данных уменьшается избыточность, неоднозначность, противоречивость, сложность извлечение данных и т. п.

В нормализованной базе данных отношения между таблицами соответствуют реальным отношениям между данными. В этом смысле нормализация сводится к соблюдению здравого смысла.

Рассмотрим несколько базовых принципов нормализации реляционных баз данных.

В таблице каждая строка должна содержать одинаковое число столбцов. В принципе по-другому и быть не может, ведь при создании таблицы с помощью SQL однозначно определяются столбцы и их типы. Однако некоторые записи могут вообще не предполагать заполнение каких-либо столбцов.

Представим, что в нашей базе мы должны хранить даты создания и изменения страниц. Если дополнить таблицу pages столбцами дат, то у одной записи могут быть заполнены все эти столбца, потому что страница часто правилась, а у другой – только один, потому что страница была создана, но больше не правилась. Да и количество столбцов под даты заранее неизвестно.

Чтобы привести базу в нормальную форму, надо создать другую таблицу. В ней будут храниться даты. При создании или правке страницы в эту таблицу добавляется запись – ID страницы и дата правки.

```
sqlite> CREATE TABLE dates (
   ...> _id INTEGER PRIMARY KEY AUTOINCREMENT,
   ...> page_id INTEGER NOT NULL,
   ...> date TEXT,
   ...> FOREIGN KEY (page_id)
   ...> REFERENCES pages(_id)
   ...> );
```

```
sqlite> PRAGMA foreign_keys = ON;
sqlite> CREATE TABLE dates (
   ...> _id INTEGER PRIMARY KEY AUTOINCREMENT,
   ...> page_id INTEGER NOT NULL,
   ...> date TEXT,
   ...> FOREIGN KEY (page_id) REFERENCES pages(_id)
   ...> );
sqlite> SELECT _id FROM pages;
1
3
7
8
9
sqlite> INSERT INTO dates VALUES (1, 1, '2019-05-25');
sqlite> INSERT INTO dates VALUES (2, 3, '2019-05-26');
sqlite> INSERT INTO dates VALUES (3, 7, '2019-05-26');
sqlite> INSERT INTO dates VALUES (4, 8, '2019-05-30');
sqlite> INSERT INTO dates VALUES (5, 9, '2019-06-01');
sqlite> INSERT INTO dates VALUES (6, 3, '2019-06-03');
sqlite> INSERT INTO dates VALUES (7, 8, '2019-06-04');
sqlite> SELECT * FROM dates;
1|1|2019-05-25
2|3|2019-05-26
3|7|2019-05-26
4|8|2019-05-30
5|9|2019-06-01
6|3|2019-06-03
7|8|2019-06-04
```

Вывод данных о правках в более информативном виде:

```
sqlite> SELECT dates.date, pages.title
   ...> FROM dates, pages
   ...> ON dates.page_id == pages._id
   ...> ORDER BY date(dates.date) DESC;
2019-06-04|Boolean Lows
2019-06-03|Amount of Information
2019-06-01|What is Algorithm
2019-05-30|Boolean Lows
2019-05-26|Amount of Information
2019-05-26|Binary System
2019-05-25|What is Information
```

В SQLite нет типов данных под даты и время. Однако функция date() позволяет преобразовывать текст в дату и сравнивать даты между собой.

Запрос о последних правках каждой страницы:

```
sqlite> SELECT max(date(dates.date)),
   ...> pages.url
   ...> FROM dates, pages
   ...> ON dates.page_id == pages._id
   ...> GROUP BY pages.url
   ...> ORDER BY date(dates.date) DESC;
2019-06-04|boolean
2019-06-03|amount-information
2019-06-01|algorithm
2019-05-26|binary
2019-05-25|information
```

Здесь в каждой группе находится максимальное значение в поле даты. Сортировка выполняется уже после.

Если мы заходим узнать историю конкретной страницы, то можем выполнить запрос

```
sqlite> SELECT date FROM dates
   ...> WHERE page_id == 8
   ...> ORDER BY date DESC;
```

```
sqlite> SELECT _id FROM pages WHERE url == 'boolean';
8
sqlite> SELECT date FROM dates WHERE page_id == 8 ORDER BY date DESC;
2019-06-04
2019-05-30
sqlite>
```

Следующий принцип нормализации: **в таблице не должно быть полностью идентичных записей.** Записи должны различаться как минимум по уникальному ключу, которым чаще всего является первичный ключ. Однако могут быть таблицы с составным первичным ключом, когда уникальность записи определяется несколькими полями.

Запись таблицы должна описывать только одну сущность. Например, есть таблица, где описаны продукты, их количество и для каждого продукта указан поставщик. В эту таблицу нельзя помещать адрес поставщика, так как адрес поставщика не относится к продукту. Столбец "поставщик" следует сделать внешним ключом к другой таблице, в которой описывается сам поставщик, его адрес и другие данные.

В нашей базе данных мы могли бы темы указывать словами непосредственно в таблице pages и не заводить таблицу sections. Однако есть целый ряд причин для выноса тем в отдельную таблицу:

- Если мы захотим поменять название какой-нибудь темы, то проще это сделать в таблице sections, так как тема там встречается один раз. Если бы в pages указывались названия, а не ID тем, пришлось бы найти все строки с необходимым названием и обновить их.
- В sections мог быть еще одни столбец, например "описание темы", который относится к теме, а не странице. Иначе мы бы нарушили нормальную форму, согласно которой запись должна описывать одну сущность. Если же потребуется получить полные сведения о какой-нибудь странице, в том числе описание ее темы, это можно сделать через JOIN.
- На таблицу sections может быть внешний ключ из какой-нибудь другой таблицы, а не только из pages.

Разработка базы данных и ее нормализация сложный процесс, предшествующий заполнению БД и работе с ней. Чтобы грамотно разработать сложную базу данных, надо хорошо знать предметную область, для которой создается БД, особенности SQL и, нередко, специфику конкретной СУБД.

Подзапросы и шаблоны

Вспомним запрос, с помощью которого мы узнавали даты правки отдельно взятой страницы. Если не известен ID страницы, сначала надо выполнить запрос к таблице pages:

```
sqlite> SELECT _id FROM pages
   ...> WHERE url == 'boolean';
8
sqlite> SELECT date FROM dates
   ...> WHERE page_id == 8
   ...> ORDER BY date DESC;
2019-06-04
2019-05-30
```

Однако язык SQL позволяет комбинировать запросы – первый запрос сделать подзапросом во втором:

```
sqlite> SELECT date FROM dates
   ...> WHERE page_id ==
   ...> (SELECT _id FROM pages
   ...> WHERE url == 'boolean')
   ...> ORDER BY date DESC;
2019-06-04
2019-05-30
```

В конце подзапроса точка с запятой не ставятся, подзапрос заключается в круглые скобки. Его результат подставляется в основной запрос. В данном случае page_id будет сравниваться с найденным _id из таблицы pages.

Представим, что нам неизвестны полное название или URL страницы. Как ее найти, не просматривая всю таблицу? Для этих целей в SQL предусмотрен оператор LIKE, после которого в одинарных кавычках записывается шаблон, на соответствие которому ищутся записи.

В шаблонах используют символы % и _. Первый соответствует любому количеству неизвестных символов, в том числе ни одному. Знак подчеркивания соответствует любому, но одному символу. Так если нам известно, что url страницы начинается с 'b', найти ее можно следующим образом:

```
sqlite> SELECT _id,url FROM pages
   ...> WHERE url LIKE 'b%';
7|binary
8|boolean
```

Запрос с оператором LIKE также можно использовать в качестве подзапроса. Однако, если шаблону будет соответствовать несколько записей, то в запрос из подзапроса будет взята только первая попавшаяся:

```
sqlite> SELECT date,page_id
   ...> FROM dates WHERE page_id ==
   ...> (SELECT _id FROM pages
   ...> WHERE url LIKE 'b%');
2019-05-26|7
```

Поэтому подзапросы с LIKE следует делать более конкретными:

```
sqlite> SELECT date,page_id
   ...> FROM dates WHERE page_id ==
   ...> (SELECT _id FROM pages
   ...> WHERE url LIKE 'b__l%n');
2019-05-30|8
2019-06-04|8
```

Views – представления

Бывает удобно сохранить результат выборки для дальнейшего использования. Для этих целей в языке SQL используется оператор CREATE VIEW, который создает представление – виртуальную таблицу. В эту виртуальную таблицу как бы сохраняется результат запроса.

Таблица виртуальная потому, что на самом деле ее нет в базе данных. В такую таблицу не получится вставить данные, обновить их или удалить. Можно только посмотреть хранящиеся в ней данные, сделать из нее выборку.

С другой стороны, если вы вносите изменения в реальные таблицы, они будут отражены и в виртуальных, потому что СУБД каждый раз, когда запрашивается представление, использует SQL выражение представления для обновления данных. Рассмотрим простой пример:

```
sqlite> CREATE VIEW title_url AS
   ...> SELECT title,url FROM pages;

sqlite> SELECT * FROM title_url;
What is Information|information
Amount of Information|amount-information
Binary System|binary
Boolean Lows|boolean

sqlite> INSERT INTO pages (title, url, theme)
   ...> VALUES
   ...> ('What is Algorithm', 'algorithm', 4);

sqlite> SELECT * FROM title_url;
What is Information|information
Amount of Information|amount-information
Binary System|binary
Boolean Lows|boolean
What is Algorithm|algorithm
```

Сначала было создано представление title_url, затем – добавлена еще одна запись в реальную таблицу pages. При выборке из представления мы видим эту запись.

Часто в представления объединяют данные из нескольких таблиц:

```
sqlite> CREATE VIEW change_page AS
   ...> SELECT pages._id, pages.title,
   ...> sections.name, dates.date
   ...> FROM pages JOIN dates JOIN sections
   ...> ON pages._id = dates.page_id
   ...> AND sections._id = pages.theme
   ...> ORDER BY dates.date DESC;

sqlite> SELECT * FROM change_page;
8|Boolean Lows|Boolean Algebra|2019-06-04
3|Amount of Information|Information|2019-06-03
9|What is Algorithm|Algorithm|2019-06-01
8|Boolean Lows|Boolean Algebra|2019-05-30
3|Amount of Information|Information|2019-05-26
7|Binary System|Digital Systems|2019-05-26
1|What is Information|Information|2019-05-25
```

Удаляются представления с помощью команды DROP VIEW:

```
sqlite> DROP VIEW title_url;
```

Что если нам нужны только пять страниц, которые последними претерпели изменения. Как вывести определенную часть таблицы? Для этих целей есть оператор LIMIT:

```
sqlite> SELECT * FROM change_page LIMIT 5;
8|Boolean Lows|Boolean Algebra|2019-06-04
3|Amount of Information|Information|2019-06-03
9|What is Algorithm|Algorithm|2019-06-01
8|Boolean Lows|Boolean Algebra|2019-05-30
3|Amount of Information|Information|2019-05-26
```

Работает он как с виртуальными, так и реальными таблицами:

```
sqlite> SELECT * FROM dates LIMIT 3;
1|1|2019-05-25
2|3|2019-05-26
3|7|2019-05-26

sqlite> SELECT * FROM dates LIMIT 2, 3;
3|7|2019-05-26
4|8|2019-05-30
5|9|2019-06-01
```

Если после LIMIT указано два числа, то первое обозначает смещение, и только второе – количество выбираемых строк. Кроме того, можно указывать смещение с помощью ключевого слова OFFSET:

```
sqlite> SELECT * FROM dates LIMIT 3 OFFSET 2;
3|7|2019-05-26
4|8|2019-05-30
5|9|2019-06-01
```

Загрузка данных из файлов

Чтобы не добавлять города вручную, возьмем готовый набор данных — city.csv (↓ скачать). Скачаем файл и загрузим данные (песочница):

```
$ sqlite3 city-1.db
SQLite version 3.34.0 2020-12-01 16:14:00
Enter ".help" for usage hints.
sqlite> .mode box
sqlite> .import --csv city.csv city
sqlite> select count(*) from city;
┌──────────┐
│ count(*) │
├──────────┤
│ 1117     │
└──────────┘
```

Команда `.import` автоматически создала таблицу `city` со всеми столбцами из `city.csv` и загрузила данные из файла. Неплохо!

Посмотрим, какие столбцы есть в таблице:

```
sqlite> .schema city
CREATE TABLE city(
  "address" TEXT,
  "postal_code" TEXT,
  "country" TEXT,
  "federal_district" TEXT,
  "region_type" TEXT,
  "region" TEXT,
  "area_type" TEXT,
  "area" TEXT,
  "city_type" TEXT,
  "city" TEXT,
  "settlement_type" TEXT,
  "settlement" TEXT,
  "kladr_id" TEXT,
  "fias_id" TEXT,
  "fias_level" TEXT,
  "capital_marker" TEXT,
  "okato" TEXT,
  "oktmo" TEXT,
  "tax_office" TEXT,
  "timezone" TEXT,
  "geo_lat" TEXT,
  "geo_lon" TEXT,
  "population" TEXT,
  "foundation_year" TEXT
);
```

И взглянем на содержимое:

```
select federal_district, city, population
from city limit 10;
```

```
┌──────────────────┬──────────────────┬──────────────┐
| federal_district |       city       | population |
├──────────────────┼──────────────────┼──────────────┤
| Южный            | Адыгейск         | 12689      |
| Южный            | Майкоп           | 144055     |
| Сибирский        | Горно-Алтайск    | 62861      |
| Сибирский        | Алейск           | 28528      |
| Сибирский        | Барнаул          | 635585     |
| Сибирский        | Белокуриха       | 15072      |
| Сибирский        | Бийск            | 203826     |
| Сибирский        | Горняк           | 13040      |
| Сибирский        | Заринск          | 47035      |
| Сибирский        | Змеиногорск      | 10569      |
└──────────────────┴──────────────────┴──────────────┘
```

Условие `limit 10` указывает, что вернется не более 10 записей. Рекомендую всегда использовать `limit`, если запрос может вернуть много записей.

Анализ данных

Выполним несколько запросов, чтобы освежить знания SQL (песочница).

Группировка и сортировка

Сколько городов в каждом из федеральных округов?

```
select
  federal_district as district,
  count(*) as city_count
from city
group by 1
order by 2 desc
;
```

```
┌──────────────────┬──────────────┐
|     district     | city_count |
├──────────────────┼──────────────┤
| Центральный      | 304        |
| Приволжский      | 200        |
| Северо-Западный  | 148        |
| Уральский        | 115        |
| Сибирский        | 114        |
| Южный            | 96         |
| Дальневосточный  | 82         |
| Северо-Кавказский | 58         |
└──────────────────┴──────────────┘
```

Если не встречали раньше форму записи вроде `group by 1` и `order by 2` — это короткое обращение к столбцам из `select` по порядковому номеру. Можно записать то же самое, но длиннее:

```
select
  federal_district as district,
  count(*) as city_count
from city
group by federal_district
order by count(*) desc
;
```

Фильтрация

У каких городов в названии есть слово «Красный»?

```
select address
from city
where city like '%Красный%';
```

```
┌──────────────────────────────────┐
│             address              │
├──────────────────────────────────┤
│ Ростовская обл, г Красный Сулин  │
│ Саратовская обл, г Красный Кут   │
│ Тверская обл, г Красный Холм     │
└──────────────────────────────────┘
```

Какие города появились за последние 30 лет?

```
select region, city, foundation_year
from city
where foundation_year between 1990 and 2020;
```

```
┌────────────┬────────────┬──────────────────┐
│  region    │   city     │ foundation_year  │
├────────────┼────────────┼──────────────────┤
│ Ингушетия  │ Магас      │ 1995             │
│ Татарстан  │ Иннополис  │ 2012             │
└────────────┴────────────┴──────────────────┘
```

Или так:

```
select region, city, foundation_year
from city
where
  foundation_year between date('now', '-30 years') and date('now')
;
```

```
┌────────────┬────────────┬──────────────────┐
│  region    │   city     │ foundation_year  │
├────────────┼────────────┼──────────────────┤
│ Ингушетия  │ Магас      │ 1995             │
│ Татарстан  │ Иннополис  │ 2012             │
└────────────┴────────────┴──────────────────┘
```

Сколько городов в Приволжском и Уральском округах?

```
select count(*)
from city
where
  federal_district in ('Приволжский', 'Уральский')
;
┌──────────┐
│ count(*) │
├──────────┤
│ 315      │
└──────────┘
```

Подзапросы

Сколько городов было основано в каждом веке?

```
with history as (
  select
    city,
    (foundation_year/100)+1 as century
  from city
)

select
  century || '-й век' as dates,
  count(*) as city_count
from history
group by century
order by century desc
;
┌──────────┬──────────────┐
│  dates   │ city_count │
├──────────┼──────────────┤
│ 21-й век │ 1          │
│ 20-й век │ 263        │
│ 19-й век │ 189        │
│ 18-й век │ 191        │
│ 17-й век │ 137        │
│ 16-й век │ 79         │
│ 15-й век │ 39         │
│ 14-й век │ 38         │
│ 13-й век │ 27         │
│ 12-й век │ 44         │
│ 11-й век │ 8          │
│ 10-й век │ 6          │
│ 9-й век  │ 4          │
│ 5-й век  │ 2          │
│ 3-й век  │ 1          │
│ 1-й век  │ 88         │
└──────────┴──────────────┘
```

Разберем запрос по порядку:

```
select
  city,
  (foundation_year/100)+1 as century
from city
```

Селект для каждого города определяет век, в котором тот был основан.

```
with history as (
  select ...
)
```

Выражение `with history as (...)` создает именованный запрос. Название — `history`, а содержание — селект в скобках (век основания для каждого города). К `history` можно обращаться по имени в остальном запросе, что мы и делаем.

> Строго говоря, селект в блоке `with` называют «табличным выражением» (common table expression, CTE). Так что если встретите в документации — не удивляйтесь. Запомните главное: это обычный селект, к которому можно для краткости обращаться по имени, как к таблице.

```
with history as (...)

select
  century || '-й век' as dates,
  count(*) as city_count
from history
group by century
order by century desc
;
```

Для каждого века посчитали количество городов, отсортировали по убыванию века.

> Возможно, вы заметили, что подозрительно много городов были основаны в первом веке. Действительно, это неспроста: в столбце foundation_year есть проблемные значения, из-за которых запрос «врет». Разберемся с этим в следующем модуле.

```
with history as (
  select
    city,
    (foundation_year/100)+1 as century
  from city
)

select
  century || '-й век' as dates,
  count(*) as city_count
from history
group by century
order by century desc
;
```

Выгрузка данных

Всем хороша таблица `city`, только данных в ней перебор. Что, если нам нужны только идентификаторы и названия городов Самарской области? Давайте выгрузим их в CSV:

```
$ sqlite3 city-1.db
SQLite version 3.34.0 2020-12-01 16:14:00
Enter ".help" for usage hints.
sqlite> .mode csv
sqlite> .once samara.csv
sqlite> select kladr_id, city from city where region = 'Самарская';
sqlite> .exit
```

> В песочнице запрещена запись на диск. Поэтому там команды из этого урока будут выводить результаты не в файл, а на экран.

Разберемся, что здесь произошло:

1. `.mode csv` включает вывод результатов запроса в формате CSV (вместо обычного табличного)
2. `.once samara.csv` направляет вывод результатов запроса в файл `samara.csv` (вместо печати на экран)
3. `select ...` выполняет запрос и выводит результаты в соответствии с `.mode` и `.once` — в файл `samara.csv` в CSV-формате.

В результате в `samara.csv` ровно то, что требуется:

```
6300000200000,"Жигулевск"
6300001000000,"Кинель"
6301700100000,"Нефтегорск"
6300000300000,"Новокуйбышевск"
6300000400000,"Октябрьск"
6300000500000,"Отрадный"
6300000900000,"Похвистнево"
6300000100000,"Самара"
6300000800000,"Сызрань"
6300000700000,"Тольятти"
```

Кроме `.once` есть команда `.output`, которая тоже направляет вывод в указанный файл. Вот в чем разница:

- `.once samara.csv` действует только для следующей команды (`select from city` в нашем примере). Если выполнить еще один селект — его результаты уже пойдут не в файл, а на экран.
- `.output samara.csv` действует до тех пор, пока не будет явно отменена. Сколько бы селектов вы не выполнили, их результаты SQLite запишет в `samara.csv`. Отменить можно, выполнив еще один `.output` без параметров.

С `.output` легко наделать ошибок, поэтому я предпочитаю всегда использовать `.once`.

`.once` создает файл в кодировке UTF-8, так что если захотите открыть его в Excel — используйте функцию «Данные > Из CSV-файла».

Экспорт в CSV

Если вы когда-нибудь экспортировали CSV из других программ, то, возможно, встречали настройки экспорта:

- выводить заголовки столбцов или нет;
- какой разделитель полей использовать;
- кавычить значения или нет;
- какую кодировку использовать.

Посмотрим, как меняются эти параметры в SQLite. Для наглядности будем выводить результаты не в файл, а на экран.

Вывод по умолчанию

Без заголовков, разделитель — запятая.

```
.mode csv
select kladr_id, city
from city
where region = 'Самарская'
limit 3;
6300000200000,"Жигулевск"
6300001000000,"Кинель"
6301700100000,"Нефтегорск"
```

С заголовками

```
.mode csv
.headers on
select kladr_id, city
from city
where region = 'Самарская'
limit 3;
kladr_id,city
6300000200000,"Жигулевск"
6300001000000,"Кинель"
6301700100000,"Нефтегорск"
```

С другим разделителем

```
.mode csv
.headers on
.separator ;
select kladr_id, city
from city
where region = 'Самарская'
limit 3;
kladr_id;city
6300000200000;"Жигулевск"
6300001000000;"Кинель"
6301700100000;"Нефтегорск"
```

Не кавычить значения

```
.mode list
.headers on
.separator ,
select kladr_id, city
from city
where region = 'Самарская'
limit 3;
kladr_id,city
6300000200000,Жигулевск
6300001000000,Кинель
6301700100000,Нефтегорск
```

Здесь используется `.mode list` вместо `.mode csv`. Режим `csv` всегда кавычит значения, так что если кавычки категорически не нужны — используйте `list`.

Изменить кодировку

А вот кодировку никак не изменить. SQLite выгружает данные в UTF-8.

CSV — проверенный универсальный формат. Но что, если удобнее выгрузить в другом? SQLite это умеет.

JSON

```
.mode json
select kladr_id, city
from city
where region = 'Самарская'
limit 3;
[{"kladr_id":"6300000200000","city":"Жигулевск"},
{"kladr_id":"6300001000000","city":"Кинель"},
{"kladr_id":"6301700100000","city":"Нефтегорск"}]
```

Команды INSERT

Удобно, если хотите быстро вставить данные в другую БД. Можно указать имя таблицы, в которую INSERT вставит данные (`cities` вместо `city` в примере):

```
.mode insert cities
select kladr_id, city
from city
where region = 'Самарская'
limit 3;
INSERT INTO cities(kladr_id,city) VALUES('6300000200000','Жигулевск');
INSERT INTO cities(kladr_id,city) VALUES('6300001000000','Кинель');
INSERT INTO cities(kladr_id,city) VALUES('6301700100000','Нефтегорск');
```

Markdown и HTML

```
.mode markdown
select kladr_id, city
from city
where region = 'Самарская'
limit 3;
|   kladr_id    |    city    |
|---------------|------------|
| 6300000200000 | Жигулевск  |
| 6300001000000 | Кинель     |
| 6301700100000 | Нефтегорск |
.mode html
select kladr_id, city
from city
where region = 'Самарская'
limit 3;
<TR><TH>kladr_id</TH>
<TH>city</TH>
</TR>
<TR><TD>6300000200000</TD>
<TD>Жигулевск</TD>
</TR>
<TR><TD>6300001000000</TD>
<TD>Кинель</TD>
</TR>
<TR><TD>6301700100000</TD>
<TD>Нефтегорск</TD>
</TR>
```

Здорово, правда? Нечасто встретишь такое разнообразие форматов.

Экспорт и импорт

Помните, в уроке «Загружаем данные» мы импортировали `city.csv` в таблицу `city`? CSV-файл тогда был «каноничного» формата, с запятыми-разделителями и заголовками.

Конечно, так бывает не всегда. Допустим, есть у нас файл `samara.csv`:

```
6300000200000|"Жигулевск"
6300001000000|"Кинель"
6301700100000|"Нефтегорск"
6300000300000|"Новокуйбышевск"
6300000400000|"Октябрьск"
6300000500000|"Отрадный"
6300000900000|"Похвистнево"
6300000100000|"Самара"
6300000800000|"Сызрань"
6300000700000|"Тольятти"
6300000600000|"Чапаевск"
```

Попробуем загрузить его в таблицу `samara` (песочница):

```
.import --csv samara.csv samara
.mode box
select * from samara limit 5;
┌──────────────────────────────────┐
│   6300000200000|"Жигулевск"      │
├──────────────────────────────────┤
│ 6300001000000|"Кинель"           │
│ 6301700100000|"Нефтегорск"       │
│ 6300000300000|"Новокуйбышевск"   │
│ 6300000400000|"Октябрьск"        │
│ 6300000500000|"Отрадный"         │
└──────────────────────────────────┘
```

Явно не то, что нужно.

Получается, при импорте нужны такие же настройки, как при экспорте — разделители, заголовки, кавычки. Логично, что и настраиваются они точно так же:

```
drop table if exists samara;
create table samara (kladr_id, name);
.mode csv
.headers on
.separator |
.import samara.csv samara
.mode box
select * from samara limit 5;
┌─────────────────┬──────────────────┐
│   kladr_id      │      name        │
├─────────────────┼──────────────────┤
│ 6300000200000 | Жигулевск       |
│ 6300001000000 | Кинель          |
│ 6301700100000 | Нефтегорск      |
│ 6300000300000 | Новокуйбышевск  |
│ 6300000400000 | Октябрьск       |
└─────────────────┴──────────────────┘
```

Аппаратура и материалы

1. Компьютерный класс общего назначения с конфигурацией ПК не хуже рекомендованной для ОС Windows 10 с подключением к глобальной сети Интернет.
2. Операционная система Windows 10.
3. Система контроля версий Git.
4. Браузер для доступа к web-сервису GitHub, рекомендован к использованию Google Chrome.
5. СУБД SQLite3.

Указания по технике безопасности

При работе на ЭВМ без разрешения руководителя занятия запрещается:

- подавать (снимать) напряжение на ПЭВМ и электрические розетки с распределительного щита;
- включать и выключать блоки питания ПЭВМ и мониторы;

- извлекать ПЭВМ из защитного кожуха;
- устранять неисправности, возникшие в ходе выполнения лабораторной работы.

Методика и порядок выполнения работы

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и выбранный Вами язык программирования (выбор языка программирования будет доступен после установки флажка **Add .gitignore**).
3. Выполните клонирование созданного репозитория на рабочий компьютер.
4. Дополните файл `.gitignore` необходимыми правилами для выбранного языка программирования и интегрированной среды разработки.
5. Добавьте в файл README.md информацию о группе и ФИО студента, выполняющего лабораторную работу.
6. Добавьте файл `README` и зафиксируйте сделанные изменения.
7. Решите задачу: выполните в песочнице команды:

   ```
   create table customer(name);

   select *
   from customer;

   .schema customer
   ```

   Вот что здесь происходит:

   - Первая команда (`create table`) создает таблицу `customer` с единственным столбцом `name`.
   - Вторая команда (`select`) показывает содержимое таблицы `customer` (она пустая).
   - Третья команда (`.schema`) показывает список и структуру всех таблиц в базе.

   `create` и `select` — это SQL-запросы, часть стандарта SQL. Запрос может занимать несколько строк, а в конце всегда ставится точка с запятой.

   `.schema` — это специальная команда SQLite, не часть стандарта SQL. Специальные команды всегда начинаются с точки, занимают ровно одну строку, а точку запятой в конце ставить не надо. Что вернула команда `.schema`?

8. Решите задачу: с помощью команды `.help` найдите в песочнице команду, которая отвечает за вывод времени выполнения запроса. Если ее включить, в результатах запроса добавится строчка:

   ```
   Run Time: real XXX user XXX sys XXX
   ```

   Например:

   ```
   sqlite> .SOMETHING on
   sqlite> select count(*) from city;
   1117
   Run Time: real 0.000 user 0.000106 sys 0.000069
   ```

Какая команда должна быть вместо SOMETHING?

9. Решите задачу: загрузите файл city.csv в песочнице:

```
.import --csv city.csv city
```

Затем выполните такой запрос:

```
select max(length(city)) from city;
```

Какое число он вернул?

10. Решите задачу: загрузите файл city.csv в песочнице с помощью команды `.import`, но без использования опции `--csv`. Эта опция появилась только в недавней версии SQLite (3.32, май 2020), так что полезно знать способ, подходящий для старых версий.

    Вам поможет команда `.help import`. Всего должно получиться две команды:

```
do_something
.import city.csv city
```

Какая команда должна быть вместо `do_something`?

11. Решите задачу: напишите в песочнице запрос, который посчитает количество городов для каждого часового пояса в Сибирском и Приволжском федеральных округах. Выведите столбцы `timezone` и `city_count`, отсортируйте по значению часового пояса:

```
┌──────────┬──────────────┐
| timezone | city_count |
├──────────┼──────────────┤
| UTC+3    | xxx        |
| UTC+4    | xx         |
| UTC+5    | xx         |
| UTC+6    | x          |
| UTC+7    | xx         |
| UTC+8    | xx         |
└──────────┴──────────────┘
```

Укажите в ответе значение `city_count` для `timezone = UTC+5`.

12. Решите задачу: напишите в песочнице запрос, который найдет три ближайших к Самаре города, не считая саму Самару.

    Укажите в ответе названия этих трех городов через запятую в порядке удаления от Самары. Например:

```
Нижний Новгород, Москва, Владивосток
```

> Чтобы посчитать расстояние между двумя городами, используйте формулу из школьного курса геометрии:
>
> $$distance^2 = (lat_1 - lat_2)^2 + (lon_1 - lon_2)^2 \quad (1)$$
>
> Где (lat_1, lon_1) — координаты первого города, а (lat_2, lon_2) — координаты второго.

13. Решите задачу: напишите в песочнице запрос, который посчитает количество городов в каждом часовом поясе. Отсортируйте по количеству городов по убыванию. Получится примерно так:

```
┌──────────┬────────────┐
| timezone | city_count |
├──────────┼────────────┤
| UTC+3    | xxx        |
| UTC+5    | xxx        |
| UTC+7    | xxx        |
| UTC+4    | xxx        |
| ...      |            |
└──────────┴────────────┘
```

    А теперь выполните этот же запрос, но так, чтобы результат был

    - в формате CSV,
    - с заголовками,
    - с разделителем «pipe» |

    Как выглядит четвертая строка результата?

14. Выполните индивидуальное задание. Каждый запрос к базе данных сохраните в файл с расширением sql. Зафиксируйте изменения.
15. Добавьте отчет по лабораторной работе в *формате PDF* в папку *doc* репозитория. Зафиксируйте изменения.
16. Отправьте изменения в локальном репозитории в удаленный репозиторий GitHub.
17. Проконтролируйте изменения, произошедшие в репозитории GitHub.
18. Отправьте адрес репозитория GitHub на электронный адрес преподавателя.

Индивидуальное задание

Загрузите в SQLite выбранный Вами датасет в формате CSV (датасет можно найти на сайте Kaggle). Сформируйте более пяти запросов к таблицам БД. Выгрузите результат выполнения запросов в форматы CSV и JSON.

Содержание отчета и его форма

Отчет по лабораторной работе оформляется электронно в формате PDF, должен содержать ответы на контрольные вопросы, скриншоты терминала с командами SQLite и результатами выполнения этих команд.

Вопросы для защиты работы

1. Каково назначение реляционных баз данных и СУБД?
2. Каково назначение языка SQL?
3. Из чего состоит язык SQL?
4. В чем отличие СУБД SQLite от клиент-серверных СУБД?
5. Как установить SQLite в Windows и Linux?
6. Как создать базу данных SQLite?
7. Как выяснить в SQLite какая база данных является текущей?
8. Как создать и удалить таблицу в SQLite?

9. Что является первичным ключом в таблице?
10. Как сделать первичный ключ таблицы автоинкрементным?
11. Каково назначение инструкций NOT NULL и DEFAULT при создании таблиц?
12. Каково назначение внешних ключей в таблице? Как создать внешний ключ в таблице?
13. Как выполнить вставку строки в таблицу базы данных SQLite?
14. Как выбрать данные из таблицы SQLite?
15. Как ограничить выборку данных с помощью условия WHERE?
16. Как упорядочить выбранные данные?
17. Как выполнить обновление записей в таблице SQLite?
18. Как удалить записи из таблицы SQLite?
19. Как сгруппировать данные из выборке из таблицы SQLite?
20. Как получить значение агрегатной функции (например: минимум, максимум, количество записей и т. д.) в выборке из таблицы SQLite?
21. Как выполнить объединение нескольких таблиц в операторе SELECT?
22. Каково назначение подзапросов и шаблонов при работе с таблицами SQLite?
23. Каково назначение представлений VIEW в SQLite?
24. Какие существуют средства для импорта данных в SQLite?
25. Каково назначение команды `.schema`?
26. Как выполняется группировка и сортировка данных в запросах SQLite?
27. Каково назначение "табличных выражений" в SQLite?
28. Как осуществляется экспорт данных из SQLite в форматы CSV и JSON?
29. Какие еще форматы для экспорта данных Вам известны?