**SQLite Кошкин дом Часть первая.**

Изучим SQLite с самых основ. Для начала мы научимся работать с SQLite без привязки к Android. Это позволит вам набить руку, лучше узнать её возможности. После этого будет легче интегрировать базу данных в Android.

Если вы знакомы с SQL, например, MySQL, то многое будет понятным. Я буду ориентироваться на новичков.

База данных нужна для долгого хранения большого количества данных, которые не пропадут после закрытия приложения. Кроме того, база данных на основе SQL позволяет производить различные манипуляции по выборе - найти самого толстого котика, найти самого молодого кота, найти только котов и только кошек и т.д.

Чтобы было легче понять принцип работы с базой данных, представьте себе следующую аналогию. Допустим на диске вы создали новую папку - это аналог базы данных. В новой папке вы можете создать несколько текстовых файлов с информацией (рецепты, дни рождения, список гостей) - это таблицы базы данных.

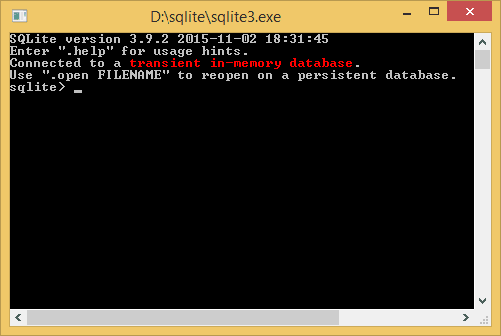
Таблица - это важная часть базы данных. Информация в таблицах упорядочена, чтобы её можно быстро найти. Таблица состоит из вертикальных столбцов (column) и горизонтальных рядов (row).

Мы построили новый пятизвёздочный отель с уникальным названием "Кошкин дом". Нам требуется учёт всех постояльцев гостиницы. Без базы данных не обойтись. Создадим базу данных**hotel** с таблицей **guests**, в которой будут столбцы **name** (имя), **city** (город), **gender** (пол), **age** (возраст).

sqlite3.exe

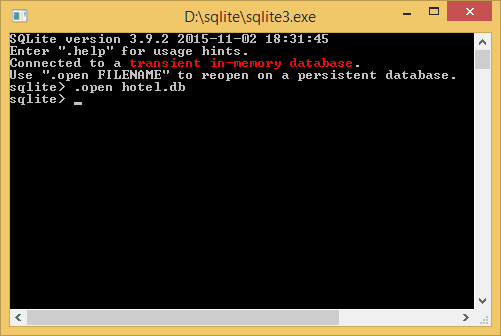
Давайте изучать SQLite. Начнём с командной строки под Windows. Хорошая новость - ничего не надо скачивать. Когда вы устанавливали инструменты разработки Android, то в папке**SDK\platform-tools** уже есть исполняемый файл **sqlite3.exe**. Для удобства я создал новую папку **sqlite** для опытов и скопировал туда этот файл.

Если сейчас запустить исполняемый файл, то увидим следующее окно.



В окне выводится подсказка, что для работы с базой данных нужно набрать команду **.open FILENAME**. Вводим:

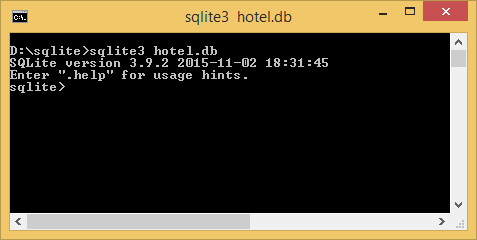
.open hotel.db



В папке рядом с **sqlite3.exe** появится новый файл **hotel.db**.

Далее в окне наберите команду .help для просмотра всех команд. Затем введите команду .quit, чтобы закрыть программу.

Теперь запустите отдельно командную строку и введите команду sqlite3 hotel.db.



Вы открыли базу данных **hotel** и программа выводит приглашение **sqlite>** для ввода специальных команд.

Прежде чем заполнять базу новыми данными, нужно ознакомиться с используемыми типами данных в SQLite:

* NULL - пустое значение
* INTEGER - целое число
* REAL - дробное число
* TEXT - строка
* BLOB - для изображений и бинарных файлов

Как видите, в SQLite не используются булевы типы. Поэтому для таких случаев используется тип **INTEGER** со значениями 0 для **false** и 1 для **true**.

Также для оптимизации используйте **INTEGER** вместо **TEXT** там, где это возможно. Например, для номерного фонда вы можете использовать значения 0, 1, 2, 3 вместо описания "одноместный", "двухместный", "люкс" и т.д.

Вернёмся к приглашению **sqlite>** и введём команду .tables. Так как у нас нет таблиц, то ничего не выводится. Запомним эту команду.

Создание таблицы

В SQL используется язык, очень похожий на разговорный английский. Так, для создания новой таблицы используется следующий синтаксис:

CREATE TABLE table\_name (column\_name\_1 data\_type\_1,

column\_name\_2 data\_type\_2,

...);

В переводе звучит так: СОЗДАТЬ ТАБЛИЦУ такую-то (первый\_столбец тип столбца, второй\_столбец тип столбца, ...).

Команды SQL принято писать заглавными буквами, хотя это не обязательно. Не путайте команды SQL с командами **sqlite3.exe**, которые начинаются с точки. Например, команду **.help**нужно вводить только в нижнем регистре.

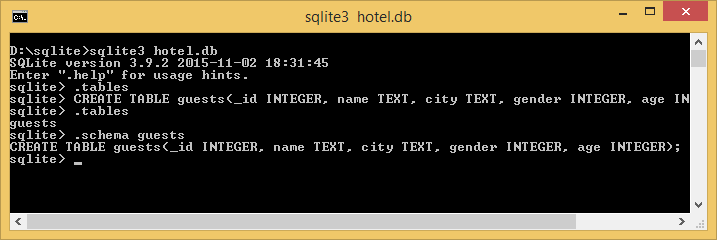
В нашем случае для создания таблицы, которая будет содержать информацию о гостях, потребуется ввести команду.

CREATE TABLE guests(\_id INTEGER, name TEXT, city TEXT, gender INTEGER, age INTEGER);

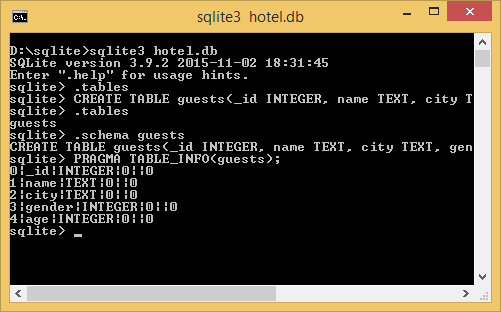
После ввода этой команды снова выполните команду **.tables**. Вы увидите созданную таблицу.



Теперь введите команду .schema, чтобы увидеть схему таблицы. Вы увидите собственную команду, которую вводили для создания таблицы.



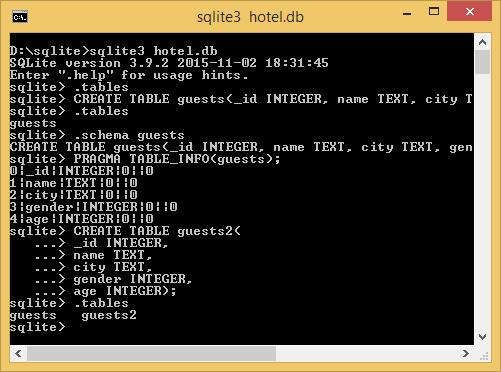
Познакомимся ещё с одной командой PRAGMA TABLE\_INFO(guests). Видим некоторое подобие таблицы с нулевыми значениями.



Если вы где-то сделали ошибку и хотите заново создать таблицу, то для удаления таблицы используется команда **DROP TABLE table\_name;**. При желании можно поставить проверочное условие **IF EXISTS**. Убедитесь, что таблицы больше нет командой .tables.

DROP TABLE IF EXISTS guests;

Небольшой совет - при вводе команд SQL принято записывать в столбик, отделяя параметры друг от друга. Когда вы находитесь в режиме ввода команд, то нажатие клавиши **Enter**приводит к переводу строки с приглашением вида **...>**. Продолжайте вводить команды в этом режиме. Когда вы закончите выражение точкой с запятой, то программа поймёт, что следующее нажатие **Enter** должно запустить команду. Для демонстрации я создал вторую таблицу **guests2**.



Небольшая подсказка: наши команды сохраняются в истории. Поэтому во время ввода вы можете нажать на клавишу "Стрелка Вверх" на клавиатуре, чтобы быстро получить доступ к длинной строке. Также можно нажать клавишу F7 - в этом случае появится диалоговое окно со списком предыдущих команд.

Экспорт данных

Можно экспортировать объекты базы данных в SQL-формате при помощи команды **.dump**. Без аргументов будет экспортирована вся база. Чтобы экспорт шел в файл, а не на экран (по умолчанию), то используйте команду **.output [filename]**. А чтобы восстановить вывод данных на экран, используйте команду **.output stdout**:

sqlite> .output guests.sql

sqlite> .dump

sqlite> .output stdout

Когда вы выполните эти команды, то у вас на диске появится файл **guests.sql** (если такой файл уже был, то он будет перезаписан).

Через командную строку:

sqlite3 hotel.db .dump

Дамп сохраняем в файле:

sqlite3 hotel.db .dump > guests.sql

Для создания новой базы данных hotel.db из нашего дампа guests.sql можно так:

sqlite3 hotel.db < guests.sql

Другой способ создания базы данных заключается в использовании опции *init*:

sqlite3 -init test.sql test2.db

В этом случае будет создана база, и мы войдем в оболочку программы. Можно добавить команду **.exit**, чтобы выйти из оболочки:

sqlite3 -init test.sql test2.db .exit

Импорт данных

Импортировать данные можно двумя способами. Если данные содержатся в SQL-формате, то можно воспользоваться командой **.read**. Если файл содержит данные в формате CSV, то используется команда **.import [file][table]**. Данные обычно разделяются вертикальной чертой |. Но можно использовать и другие разделители. Разделитель можно задать командой**.separator**, а увидеть используемый разделитель можно командой **.show**.

sqlite> .show

echo: off

eqp: off

explain: off

headers: on

mode: column

nullvalue: ""

output: stdout

colseparator: "|"

rowseparator: "\n"

stats: off

width:

В строчке **separator** вы можете видеть используемый символ.

Мы использовали для экспорта команду **.dump**, поэтому, чтобы импортировать данные из созданного файла **guests.sql**, нам подойдет команда **.read**.

.read guests.sql

SELECT \* FROM guests;

Экспорт и импорт части данных

Не всегда нужно сохранять всю базу, иногда нужно сохранить только её часть. Предположим, мы решили сохранить только те ряды таблицы, в которых имена котов заканчиваются на **ik**:

sqlite> .output ik.csv

sqlite> .separator ,

sqlite> SELECT \* FROM guests WHERE VALUE name '%ik';

sqlite> .output stdout

Теперь, если мы захотим импортировать сохраненные данные в такую же таблицу с схожей структурой, то делаем следующее:

sqlite> CREATE TABLE guests2(\_id integer PRIMARY KEY, name TEXT);

sqlite> .import ik.csv guests2

Настройка

Мы можем поменять вид приглашения, который по умолчанию имеет вид **sqlite>** через команду **.prompt**. Давайте поменяем на более знакомое и правильное:

sqlite> .prompt 'cat>'

cat>

Теперь вместо глупого приглашения используется любимое нами слово.

CRUD

Потренировавшись с созданием таблиц, перейдём к их наполнению. Существуют четыре базовых операции при работе с записями таблицы.

* **C**reate - создать новую запись
* **R**ead - прочитать запись
* **U**pdate - обновить запись
* **D**elete - удалить запись

По первым буквам операций создано сокращение CRUD для быстрого запоминания, что нужно реализовать программисту в своей программе.

Начнём с вставки новой записи в таблицу. Синтаксис команды.

INSERT INTO table\_name (column\_name\_1, column\_name\_2, ...)

VALUES (value\_1, value\_2, ...);

Переводим: ВСТАВИТЬ В такую-то таблицу (первый\_столбец, второй\_столбец, ...) Значения (первое\_значение, второе\_значение, ...)

Вставляем первую запись в таблицу.

INSERT INTO guests (\_id, name, city, gender, age) VALUES (1, "Васька", "Питер", 1, 6);

Итак, в гостинице поселился первый гость. При вводе команды важно соблюдать очерёдность столбцов и их значений. Например, можно было указать имена столбцов в обратном порядке, но тогда и значения следовало бы ввести также в обратном порядке. Но обычно стараются перечислять столбцы в том же порядке, как они создавались.

Существует укороченная запись без перечисления столбцов. В этом случае нужно указывать все значения и в том порядке, в котором создавались соответствующие столбцы.

INSERT INTO guests VALUES (1, "Васька", "Питер", 1, 6);

Проверить наличие записи можно с помощью команды:

SELECT \* from guests;

Звёздочка (\*) выводит все записи из указанной таблицы. Перед данной командой можно задать режим вывода записей через команду **.mode режим**. Доступны варианты: ascii, column, csv, html, insert, line, tabs, tcl.

Также удобно включить показ имён столбцов через команду **.header on**.

.header on .mode column SELECT \* FROM guests;



Вы можете теперь вводить новые записи, не забывая увеличивать значение идентификатора **\_id**. Но нет никакой гарантии, что однажды вы не ошибётесь и не введёте одинаковый идентификатор. Для таблицы базы данных это очень плохая ситуация, так как теряется принцип уникальности для записи.

Вторая возможная проблема - пропуск столбца. Например, возможна такая запись.

INSERT INTO guests (\_id, city, gender, age) VALUES (1, "Питер", 1, 6);

Мы пропустили столбец **name** и после вставки записи там будет пустое значение. Получается, что гость живёт в номере отеля без имени. Непорядок.

Для решения подобных проблем в SQL есть специальные ключевые слова: **PRIMARY KEY**, **AUTOINCREMENT**, **NOT NULL**, **DEFAULT value**.

**PRIMARY KEY** (первичный ключ) обеспечивает уникальность в таблице. В таблице может быть только один первичный ключ.

Первичный ключ - столбец таблицы, имеющий уникальное значение для каждой записи. Назначается при создании таблицы. Ключ не может содержать **NULL**, потому что теряется уникальность, ведь в других записях тоже может оказаться **NULL**. Значения первичного ключа должны оставаться неизменными.

Во многих случаях для этой цели создают новый столбец, который будет содержать уникальный номер. В Android столбец называют **\_id**.

Чтобы база данных сама заботилась об уникальности первичного ключа, можно добавить к нему ключевое слово **AUTOINCREMENT**, которое будет автоматически увеличивать значение на единицу при вставке новой записи.

**AUTOINCREMENT** (автоувеличение) автоматически вычисляет следующее значение ряда таблицы при добавлении. Удобно использовать у идентификаторов.

Поэтому скрипт создания таблицы должен иметь следующий вид.

CREATE TABLE guests(\_id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT,

city TEXT,

gender INTEGER,

age INTEGER);

Удалите таблицу **guests** с помощью команды DROP TABLE guest; и заново создайте таблицу с этим же именем.

Теперь для вставки новой записи вам не нужно указывать значение для первого столбца **\_id**, база данных сама сгенерирует нужное значение. Вставим новые записи.

INSERT INTO guests (name, city, gender, age) VALUES ("Васька", "Питер", 1, 6); INSERT INTO guests (name, city, gender, age) VALUES ("Мурзик", "Мурманск", 1, 4);

Если бы вы попытались использовать старый вариант с идентификатором, то получили бы сообщение об ошибке. Установленное нами правило теперь не позволяет вставлять записи с собственными идентификаторами. Мы передали эти полномочия базе данных.

**NOT NULL** не разрешает оставлять пустым определённый столбец. Если при вставке мы пропустим этот столбец, то снова увидим сообщение об ошибке.

**DEFAULT value** - если при вставке новой строки мы не зададим значения для столбца, то применится значение по умолчанию. Данный параметр можно комбинировать с предыдущим.

В этом случае мы можем пропустить столбец, но вместо ошибки база данных подставит значение по умолчанию.

Увы, но вам снова придётся удалить таблицу и воссоздать её по новому правилу.

CREATE TABLE guests(\_id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

city TEXT NOT NULL,

gender INTEGER NOT NULL DEFAULT 3,

age INTEGER NOT NULL DEFAULT 0);

Заново заселяем Ваську и Мурзика, команды не меняются. Допустим, к нам заехал необычный гость, у которого неудобно было спросить возраст и пол. Вводим только имя и город, остальное добавится автоматически по умолчанию.

INSERT INTO guests (name, city) VALUES ("Ктулху", "Москва");

Можно выводить не все записи из таблицы, а только нужные. Например, запись с идентификатором 1.

SELECT \* FROM guests WHERE \_id=1;

Если не нужны все столбцы, то перечисляем нужные через запятую вместо звёздочки.

SELECT name, city FROM guests WHERE \_id < 3;

Или такой вариант.

SELECT name, city FROM guests WHERE name !="Ктулху";

Чтобы не искать все записи, можно ограничить поиск условием **WHERE**, после которого идёт имя столбца и условие равенства. Показать всех котов, чей возраст меньше 15.

SELECT \* FROM guests WHERE age < 15;

Вместо звёздочки можно указать столбцы, которые вам нужны. Например, нам нужны только имена котов с этим же условием.

SELECT name FROM guests WHERE age < 15;

Столбцы указываются через запятую. Нам нужны имена и адреса котов с этим же условием.

SELECT name, city FROM guests WHERE age < 15;

Условие **WHERE** можно объединять с помощью ключевого слова **AND**. Список котов младше 15 лет и проживающих в Москве.

SELECT name, city FROM guests WHERE age < 15 AND city="Мурманск";

Также доступны слова **OR** (ИЛИ):

SELECT name, city FROM guests WHERE age < 15 OR city="Мурманск";

Слов **AND** и **OR** может быть несколько в одном запросе.

Проверку на **NULL** можно сделать с помощью ключевого слова **IS NULL** (если столбец таблицы создавался без **NOT NULL**)

SELECT \* FROM guests WHERE age IS NULL;

Ключевое слово **LIKE** позволяет сократить множество операторов **OR**. Например, мы хотим узнать имена котов, которые заканчиваются на "ик":

SELECT name FROM guests WHERE name LIKE '%ик';

Символ **%** в строке указывает на любое слово с нужным окончанием (представляет любое количество неизвестных символов).

Также можно использовать спецсимвол **\_** для одного символа. Ищем Ваську.

SELECT name FROM guests WHERE name LIKE '\_аська';

Стоит заметить, что в командной строке примеры могут не работать, так как там не используется кодировка UTF-8. Проверяйте на английских словах. В других программах или в Android такой проблемы не будет, там всегда используется правильная кодировка.

С помощью ключевого слова **BETWEEN** можно быстро и удобно задать диапазон.

SELECT name FROM guests WHERE age BETWEEN 10 and 20;

Сравните с более длинной записью

SELECT name FROM guests WHERE age >= 10 and age <=20;

С помощью условия **IN** за которыми в скобках идут нужные значения, можно задать нужные параметры.

SELECT name FROM guests WHERE age IN (10, 29);

Это короче, чем

SELECT name FROM guests WHERE age = 10 OR age = 29;

Ключевое слово **NOT IN** выполняет обратную задачу и позволяет получить записи, которые не входят в данное условие.

SELECT name FROM guests WHERE age NOT IN (10);

Ключевое слово **NOT** можно использовать не только с **IN**, но и с **BETWEEN**, **LIKE**.

Узнать число записей можно через функцию **COUNT**. Если запись содержит **NULL**, то она не учитывается.

SELECT COUNT(name) FROM guests;

Для показа минимального или максимального значения используются функции **MIN** или **MAX**:

SELECT name, MAX(age) FROM guests;

Если нам нужно вывести только определённое количество записей, то используйте ключевое слово **LIMIT** с указанием значения.

SELECT \* FROM guests LIMIT 3;

Существует расширенная версия, когда можно указать два значения через запятую. В первой указывается номер записи (отсчёт от 0), а вторая - число записей. Например, показать вторую запись из таблицы.

SELECT \* FROM guests LIMIT 1,1;

Мы рассмотрели половину операций с записями - CREATE и READ. Теперь нужно научиться изменять данные. Конечно, самый простой способ - удалить запись, а затем добавить новую с исправленными данными. Но это очень неуклюжий способ.

Для этого существует команда **UPDATE имя\_таблицы SET column = value WHERE условие**. После **UPDATE** указываете таблицу, после **SET** - в каком столбце нужно внести изменения и указывается новое значение, а затем указывается условие.

Можно обновлять группу столбцов, указывая их через запятую.

UPDATE имя\_таблицы SET первый\_столбец="новое значение", второй\_столбец="новое значение";

Команда **UPDATE** заменяет собой пару команд **INSERT/DELETE**. Обновить данные в нужном столбце:

UPDATE имя\_таблицы SET имя\_столбца = новое\_значение WHERE имя\_столбца = старое\_значение

Также можно производить математические действия: прибавлять, отнимать, умножать, делить. Увеличим возраст кота на день рождения.

UPDATE guests SET age=age+1 WHERE name="Мурзик";

Изменим прописку у Ктулху.

UPDATE guests SET city = "Нью-Йорк" WHERE name = "Ктулху";

Если не указать условие **WHERE**, то город изменится у всех гостей сразу. Удобно, если приехала большая делегация из одного города.

Последняя операция - удаление записи из таблицы. Команда **DELETE FROM имя\_таблицы**;

Без условия **WHERE** мы удалим все записи. Вряд ли вам это нужно. Давайте выпишем из гостиницы Ктулху, вежливо объяснив ему, что отель только для котов. Выпроводив незванного гостя, удаляем запись из таблицы.

DELETE FROM guests WHERE \_id = 3;

Вам не надо перечислять все столбцы, достаточно указать в условии нужный столбец. Условие **WHERE** работает аналогично как в команде **SELECT** и позволяет использовать ключевые слова **LIKE**, **BETWEEN** и т.д.

Кстати, вы можете посмотреть, какой идентификатор был вставлен в таблицу последним через команду:

SELECT LAST\_INSERT\_ROWID();

Добавить новый столбец в таблицу можно с помощью необязательного ключевого слова **ALTER**, за которым идёт название столбца в таблице.

ALTER TABLE guests ADD COLUMN weight INTEGER;

Чтобы указать, после какого столбца нужно добавить новый столбец, используйте ключевое слово **AFTER**.

Другие ключевые слова: **FIRST**, **BEFORE**, **LAST**, **SECOND**, **THIRD**.

Кроме **ADD**, также можно изменить имя и тип данных столбцов (**CHANGE**), изменить тип данных или позиции столбцов (), удалить столбец из таблицы (). Не все эти команды поддерживаются в SQLite, хотя часто используются в обычных SQL.

Переименовать саму таблицу (**RENAME TO**).

ALTER TABLE guests RENAME TO cats;

Преобразовать текст из указанного столбца в верхний регистр.

UPDATE guests SET name = UPPER(name);

Завершить работу с **sqlite3** можно через команду:

.exit