Рекомендации по использованию SQLite на мобильных устройствах

Всегда используйте кодировку UTF-8

Это оптимальный вариант по быстродействию и размеру занимаемой памяти (внутри себя SQLite все запросы переводит в UTF-8, поскольку имеет встроенный парсер только этой кодировки).

Храните в базе как можно меньше данных

Этот совет не от капитана Очевидности. Не используйте SQLite просто потому, что это «модно». Продумайте альтернативы: XML, например. За использование SQLite нужно платить тратой ресурсов.

Всегда явно управляйте транзакциями

Иначе на каждую команду SQL будет создаваться и фиксироваться отдельная, неявная транзакция. Группируйте изменения данных.

Тщательно продумывайте создание индексов

Индексы ускоряют выборку, но замедляют модификацию данных. Иногда полезно использовать covering индексы, когда в определение индекса включаются поля с данными:

**CREATE** **INDEX** idx **ON** tab(key1, key2, data1, data2, data3);

Теперь, если выбирать (по условию на key1 и key2) данные столбцов data1, data2, data2, SQLite может взять их значения из самого индекса (нет необходимости дополнительного чтения данных из таблицы). Но такой индекс занимает больше места. Поэтому, следующий совет:

Лучший способ проверить что-то: тестировать на конкретном устройстве

И, разумеется, не на пустой базе. Заполните ее случайными данными, если нет подходящих. Иначе быстродействие в реальном сценарии вас неприятно удивит.

Продумайте как хранить BLOBы

На официальном сайте [приводится](http://www.sqlite.org/intern-v-extern-blob.html) тест, который показывает, что при использовании BLOBов следует:  
— поставить размер страницы побольше (8192 или 16384);  
— большие BLOBы хранить в отдельных файлах.  
Если вы решили хранить BLOBы в базе, заведите отдельную таблицу под них:

**CREATE** blobs(blobid INTEGER PRIMARY **KEY**, **data** BLOB)

и в таблицах с данными поставьте поле со ссылкой на запись в этой таблице. Это совет от самих разработчиков SQLite.

Используйте временные таблицы

Если при создании таблицы, вы указали «TEMP»

**CREATE** TEMP TABLE tab(...)

то созданная таблица будет уничтожена автоматически при закрытии базы данных.

Используйте параметризованные запросы

То есть передавайте значения в SQL запросы не в виде литералов внутри SQL, а привязывая их к параметрам. Это одновременно обезопасит ваш код от атак SQL injection и повысит быстродействие.

Добавьте AUTOINCREMENT в определение PRIMARY KEY

Столбец 

id INTEGER PRIMARY **KEY**

является синонимом поля ROWID (уникального идентификатора записи). Хитрость в том, что SQLite не гарантирует, что новые строки получат новые ключи, если ранее записи удалялись из таблицы. Если вам требуется «настоящая уникальность» идентификаторов добавьте к определению AUTOINCREMENT:

id INTEGER PRIMARY **KEY** AUTOINCREMENT

Теперь при любых обстоятельствах новая строка получит уникальное значение id (и rowid) (ну или вы исчерпаете 8-байтный счетчик).

Используйте внешние ключи (FOREIGN KEY) только если это действительно нужно

В SQLite имеется довольно развитый механизм поддержки внешних ключей. По умолчанию, он отключен (для совместимости). С одной стороны, внешние ключи гарантируют целостность. С другой — требуют ресурсов. Если вы все-таки решите их использовать, то…

По каждому полю внешнего ключа создайте индекс

Допустим, есть пара таблиц:

**CREATE** **TABLE** **master**(**mid** INTEGER PRIMARY **KEY** ...);

**CREATE** **TABLE** detail(fk\_master **REFERENCES** **master**(**mid**) **ON** **DELETE** **CASCADE** ...);

При удалении записи из master таблицы, SQLite должен проверить — не ссылается ли на нее какая-то запись в таблице detail и делает дополнительную выборку:

**SELECT** **rowid** **FROM** detail **WHERE** fk\_master = ?

Как правило, разработчики забывают создать индекс по FK ключу и эта выборка работает очень долго.

Удаление данных из базы не уменьшает размер ее файла

Чтобы сжать файл базы данных, выполните

VACUUM

Эта операция требует времени.  
  
Ну и самый главный совет.

Тщательно продумайте схему БД

Обычно вначале данные максимально нормализуют, а затем выполняют денормализацию для оптимизации. Например, если у пользователя часто требуют название его отдела, можно это название продублировать в таблице пользователей отдельным полем. Задумайтесь, какие данные меняются в вашей БД наиболее интенсивно, возможно, их следует выделить в отдельную таблицу.  
  
**Дополнено**

Как выполнить сразу несколько операторов SQL?

Объедините их в одну строку через ";".

Как быстро добавить много записей (bulk insert)?

Перед вставкой большого числа записей выполните

**PRAGMA** **synchronous** = **OFF**;

**PRAGMA** journal\_mode = **OFF**;

**BEGIN**;

Вставляйте записи. Наибольшей производительности можно достигнуть, откомпилировав (prepare) оператор вставки с параметрами и многократно выполнив его с требуемыми значениями. После вставки восстановите настройки (приводятся настройки по умолчанию):

**COMMIT**;

**PRAGMA** **synchronous** = **NORMAL**;

**PRAGMA** journal\_mode = **DELETE**;

Создавать индексы по таблице лучше после вставки, а не до. Учтите, что при отключенном журнале база может оказаться испорченной, если приложение «вылетит» в процессе вставки.