SQLite

**Содержание обзора:**

1. [Что представляет из себя SQLite и как всё устроено в SQLite?](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#1)
2. [Кто использует SQLite?](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#2)
3. [Какие инструменты использовать для администрирования баз данных SQLite?](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#3)
4. [Какие компоненты есть в Delphi для доступа к SQLite?](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#4)
5. [Установка и настройка LiteDAC](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#5)
6. [Первая база данных SQLite в Delphi](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#6)
   1. [Создаем базу данных с помощью SQLite Expert](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#6_1)
   2. [Создание базы данных с помощью LiteDAC](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#6_2)
7. [Шифрование базы данных с помощью LiteDAC](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#7)
   1. [Шифруем существующую базу данных](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#7_1)
   2. [Создаем или открываем зашифрованную базу данных](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#7_2)
   3. [Меняем ключ шифрования базы данных](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#7_3)
8. [Запись данных в базу SQLite](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#8)
   1. [Используем транзакции](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#8_1)
   2. [Используем настройки базы данных](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#8_2)
      1. [PRAGMA synchronous](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#8_2_1)
      2. [PRAGMA temp\_store](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#8_2_2)
   3. [Используем возможности LiteDAC](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#8_3)
   4. [График скорости записи данных в базу SQLite](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#8_4)
   5. [Настройка TLiteSQL в Design-Time](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#8_5)
9. [Как исключить дублирование записей в таблице?](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#9)
10. [Как сократить размер файла базы данных?](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#10)
11. [Использование обзоров (VIEW) в SQLite](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#11)
12. [Дополнительные возможности по работе с SQLite в Delphi при использовании LiteDAC](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#12)
    1. [Создание дампа базы SQLite](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#12_1)
    2. [Макросы в LiteDAC](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#12_2)
    3. [Мониторинг запросов к SQLite в LiteDAC](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#12_3)
13. [Подведем итог](http://www.webdelphi.ru/2016/08/sqlite-v-delphi-bolshoj-obzor-i-litedac-v-primerax/#13)

**1. Что представляет из себя SQLite и как всё устроено в SQLite?**

SQLite — это компактная встраиваемая реляционная база данных. Для нас, как для разработчиков это говорит о том, что:

1. SQLite легко настраивать. Для работы с SQLite не требуется установка дополнительного программного обеспечения (разве что для удобства разработки поставить какую-нибудь утилиту администрирования базы данных, но об этом чуть ниже). Всё, что необходимо, чтобы начать работу — это положить рядом с exe-файлом вашей программы небольшую динамическую библиотеку sqlite3.dll и использовать базовые методы (SQLite API) этой dll.
2. SQLite просто использовать. Хотите — используйте методы dll, хотите — специальные компоненты, как платные, так и бесплатные. На сегодняшний день для Delphi имеется целый ряд компонентов, позволяющих работать с SQLite, в том числе и поставляемые вместе с RAD Studio.

База данный SQLite представляет из себя обычный файл, который может располагаться где угодно, лишь бы вы смогли получить к этому файлу доступ — даже на CD- или DVD-диске, правда, в этом случае вы сможете только читать базу, но не записывать в неё данные (что характерно). В итоге получаем: один файл — одна база данных. Легко, просто, понятно. Естественно, что если Вы только выбираете какую базу данных использовать в своем приложении, то одним из критериев выбора являются ограничения той или иной СУБД. Давайте посмотрим на ограничения СУБД SQLite.

Со всем списком ограничений SQLite Вы можете ознакомиться [на официальном сайте](http://www.sqlite.org/limits.html), я же обращу внимание только на основные моменты. Итак ограничения, используемые по умолчанию:

|  |  |  |
| --- | --- | --- |
| **ОПИСАНИЕ** | **ЗНАЧЕНИЕ** | **КОНСТАНТА В ИСХОДНИКЕ** |
| Максимальная длина строки или BLOB-поля | 1 000 000 000 | SQLITE\_MAX\_LENGTH |
| Максимальное количество колонок | 2 000 | SQLITE\_MAX\_COLUMN |
| Максимальная длина SQL-выражения | 1 000 000 | SQLITE\_MAX\_SQL\_LENGTH |
| Максимальное количество таблиц в выражениях с JOIN | 64 |  |
| Максимальная глубина дерева выражений | 1 000 | SQLITE\_MAX\_EXPR\_DEPTH |
| Максимальное количество аргументов функции | 100 | SQLITE\_MAX\_FUNCTION\_ARG |
| Максимальное количество термов в объединённом выражении с SELECT | 500 | SQLITE\_MAX\_COMPOUND\_SELECT |
| Максимальная длина шаблона как аргумента операторов LIKE или GLOB | 50 000 | SQLITE\_MAX\_LIKE\_PATTERN\_LENGTH |
| Максимальное количество символов-заменителей в одном SQL-выражении | 999 | SQLITE\_MAX\_VARIABLE\_NUMBER |
| Максимальная глубина рекурсии триггеров | 1 000 | SQLITE\_MAX\_TRIGGER\_DEPTH |
| Максимальное количество присоединённых баз | 10 | SQLITE\_MAX\_ATTACHED |
| Максимальный размер страницы базы данных | 32 768 | SQLITE\_MAX\_PAGE\_SIZE |
| Максимальное количество страниц в файле базы данных | 1 073 741 823 | SQLITE\_MAX\_PAGE\_COUNT |

Как видите, ограничения более, чем приемлемые для разработки более менее серьезных приложений. Конечно, в случае острой необходимости Вы всегда сможете поменять некоторые ограничения в большую сторону (например, установить максимальное количество присоединенных баз данных в максимально возможное значение — 125), однако, лично мне с трудом представляется то, какое приложение надо писать в Delphi, чтобы превысить лимиты по умолчанию. Что же касается максимального размера файла базы данных, то, согласно официальной документации, в настоящее время максимально возможный размер файла БД SQLite составляет **140 Тб,**однако конкретно это ограничение не тестировалось, так как разработчики SQLite не имеют доступа к программным и аппаратным средствам способных достичь такого лимита…что уж говорить нам — простым смертным пользователям SQLite.

**2. Кто использует SQLite?**

Еще один немаловажный момент при выборе СУБД, который волнует разработчиков —**кто это использует?** В особенности, если рассматриваются бесплатные инструменты и средства с открытым исходным кодом (к которым относится и SQLite). Ведь, как часто бывает, даже самый гениальный проект с открытым исходным кодом может внезапно уйти в небытие и разработчик останется ни с чем: морально устаревающий со временем инструмент, отсутствие поддержки и так далее. И здесь в пользу того или иного компонента или инструмента разработчика играет наличие в рядах пользователей наиболее титулованных и популярных компаний — уж, если они не боятся использовать что-то бесплатное, то всяко-разно есть шанс, что поддержка продукта будет идти регулярно и долго (по данным официального сайта, SQLite планируется поддерживать вплоть [до 2050 года](http://www.sqlite.org/about.html)).

Вот небольшой перечень программных продуктов, которые активно используют SQLite:

1. Google Chrome
2. Skype
3. Opera
4. FireFox
5. ThunderBird
6. Safari
7. Практически все программное обеспечение для Android, которому необходимо хранить какие-либо данные (в Android встроена поддержка SQLite)

В дополнение (могу, конечно, сейчас немного ошибиться, так как давно не заглядывал в работы этим приложений) также скажу, что SQLite использовался такими клиентами облачных хранилищ как Dropbox и Google Drive.

**3. Какие инструменты использовать для администрирования баз данных SQLite?**

Как говориться, на вкус и цвет фломастеры разные и каждый сам выбирает что использовать для администрирования баз данных SQLite: кому-то (как мне, например) по душе «тяжеловесные» программы типа SQLite Expert в которых можно сделать с базой всё, что угодно, другому — подавай легкий маленький и шустрый инструмент, чтобы только посмотреть структуру, добавить пару строк или таблиц в БД и забыть. Разные потребности, соответственно, рождают самые различные предложения. А, учитывая достаточно высокую популярность SQLite, можно смело говорить о том, что выбор программы для администрирования баз SQLite более, чем достаточен даже для самого привередливого пользователя. Когда я начинал свое знакомство с SQLite, то проводил небольшой [обзор решение в области администрирования SQLite](http://www.webdelphi.ru/2010/11/obzor-programm-dlya-administrirovaniya-baz-dannyx-sqlite), но, как я уже писал выше, всё меняется и рынок программных продуктов также претерпевает свои изменения. Поэтому в этой статье, я лишь перечислю рассмотренные ранее программные продукты, а также дополню обзор другими приложениями, о которых мне сообщали Вы, уважаемые читатели, в своих комментариях к статьям.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название** | **Стоимость** | **Ссылка на сайт** | **Скрин окна программы**  **(кликабельно)** | **Дата обновления** |
| SQLite Database Browser | бесплатно | [ссылка](http://sqlitebrowser.org/) | [SQLite Database Browser](http://www.webdelphi.ru/wp-content/uploads/2016/08/sqlitebrowser.png) | 04/05/2016 |
| SQLite Administrator | бесплатно | [ссылка](http://sqliteadmin.orbmu2k.de/) | [SQLite Administrator](http://www.webdelphi.ru/wp-content/uploads/2016/08/main.jpg) | нет информации |
| SQLiteManager | $49 | [ссылка](http://www.sqlabs.com/sqlitemanager.php) | [SQLiteManager](http://www.webdelphi.ru/wp-content/uploads/2016/08/detail2@2x.png) | 17/06/2016 |
| SQLite Maestro | от $79 до $499  trial на 30 дней | [ссылка](http://www.sqlmaestro.com/) | [SQLite Maestro](http://www.webdelphi.ru/wp-content/uploads/2016/08/1393252808-920x875-1.png) | 12/01/2015 |
| SQLite Expert | версия «Personal» — бесплатно  версии «Professional» от $59 | [ссылка](http://www.sqliteexpert.com/index.html) | [SQLite Expert](http://www.webdelphi.ru/wp-content/uploads/2016/08/execute_sql_large.png) | 20/07/2016 |
| Не рассмотренные ранее программные продукты | | | | |
| SQLiteSpy | бесплатно | [ссылка](http://www.yunqa.de/delphi/products/sqlitespy/index) | sqlitespy | 20/06/2016 |
| SQLiteStudio | бесплатно | [ссылка](http://sqlitestudio.pl/) | [SQLite Studio](http://www.webdelphi.ru/wp-content/uploads/2016/08/full_49.png) | 10/06/2016 |

Таким образом, у нас в наличии семь различных программных продуктов для управления базами SQLite — более, чем достаточно, чтобы выбрать инструмент по своим потребностям. Я же, в дальнейшем буду использовать давно испытанный **SQLite Expert Personal**.

**4. Какие компоненты есть в Delphi для доступа к SQLite?**

В этой части, конечно, не такое большое разнообразие, как в части программ для управления базами SQLite, но, тем не менее и того, что есть в наличии сейчас, когда актуальной версией Delphi является Delphi 10.1 Berlin, более чем достаточно, чтобы начать работу с SQLite.

В теперь уже далеком 2010-м году, когда мы радовались выходу Delphi 2010 и обсуждали её новинки, на рынке компонентов Delphi для SQLite было совсем не радостно. Конечно, уже тогда были свои решения для работы с SQLite (всё-таки SQLite стартовала ещё раньше — в 2000-м году), но, тем не менее сама Delphi не могла похвастаться работой с SQLite «из коробки». Однако, тем, кому было необходимо использовать эту СУБД в Delphi-проектах, было вполне достаточно на первоначальном этапе использовать **[Delphi wrapper for Sqlite 3](http://www.itwriting.com/blog/?page_id=659" \t "_blank)** от Тима Андерсона ([Tim Anderson](http://www.itwriting.com/blog/" \t "_blank)). Эти простые в работе классы для SQLite в Delphi и сейчас доступны для скачивания и, даже, со временем стали поддерживать Unicode. Этим же wrapper’ом [пользовался и я в свое время](http://www.webdelphi.ru/2010/09/sqlite-v-delphi-2010/), когда только начинал разбираться с SQLite.

На базе **Delphi wrapper for Sqlite 3**также развивались такие обёртки для SQLite в Delphi, как [Delphi SQLite Wrapper](http://www.ararat.cz/doku.php/en:sqlitewrap" \t "_blank) от разработчика библиотеки Synapse и компоненты от [sv-utils](https://code.google.com/archive/p/sv-utils/downloads" \t "_blank). Все эти обёртки вносили какие-то улучшения в работу с SQLite (добавлялась работа с параметрами, поддержка Unicode, кроссплатформенность и т.д.), однако, повторюсь, они базировались на обёртке от Тима Андерсона. В настоящее время все эти замечательные миниатюрные компоненты, видимо, заброшены авторами, так как даты последних обновлений не особенно вселяют надежду на их дальнейшее развитие — 2012-2013 год. Из более менее «живых» бесплатных компонентов можно выделить [ZeosLib](https://sourceforge.net/projects/zeoslib/" \t "_blank), которой я пользовался ещё в бытность Delphi 7, а также [компоненты DISQLite3](http://www.yunqa.de/delphi/products/sqlite3/index) от разработчика SQLiteSpy.

Это, в принципе, и понятно, потому как прошло несколько лет и Embarcadero представила нам новый релиз — **Delphi XE3**, которая позволяла [работать с базами SQLite, используя DBExpress](http://www.webdelphi.ru/2012/10/sqlite-v-delphi-xe3). И, как говориться, жизнь стала налаживаться. Прошло совсем немного времени и уже [5 февраля 2013 года](http://www.webdelphi.ru/2013/02/novoe-priobretenie-embarcadero-kakie-peremeny-ozhidayutsya-v-rad-studio/) Embarcadero заявила в своем [пресс-релизе](https://www.embarcadero.com/ru/press-releases/embarcadero-technologies-acquires-high-performance-enterprise-database-connectivity-technology-for-developers) о покупке компонентов AnyDAC (ныне известных под названием FireDAC) — мощной библиотеки компонентов для доступа практически к любым базам данных, включая и SQLite. Таким образом, мы уже имели возможность после покупки лицензии на Delphi свободно использовать для доступа к базам данных SQLite:

1. **Бесплатные обёртки** от различных разработчиков «Delphi SQLite Wrapper», «[Delphi wrapper for SQLite 3](http://www.webdelphi.ru/2010/09/sqlite-v-delphi-2010/" \t "_blank)«, [sv-utils](http://www.webdelphi.ru/2011/10/sqlite-dlya-krossplatformennoj-razrabotki-v-delphi-xe2/" \t "_blank) и т.д.
2. **Бесплатные библиотеки** компонентов ([ZeosLib](https://sourceforge.net/projects/zeoslib/" \t "_blank), FireDAC)
3. [Технологию **DBExpress**](http://www.webdelphi.ru/2012/10/sqlite-v-delphi-xe3)
4. **Компоненты**([DISQLite](http://www.yunqa.de/delphi/products/sqlite3/index" \t "_blank)).

Однако, обилие различных бесплатных и относительно бесплатных решений в области работы с **SQLite в Delphi** нисколько не затормозило разработку платных компонентов для доступа к SQLite. И здесь, наиболее ярким представителем, на мой взгляд, является компания [Devart](https://www.devart.com/" \t "_blank) с их замечательными компонентами для доступа к SQLite — **[LiteDAC](http://www.webdelphi.ru/2013/07/sqlite-v-delphi-obzor-litedac/" \t "_blank)** ([стоимость компонентов на данный момент составляет от 6800 руб.](https://allsoft.ru/software/software-engineering/vcl/litedac-/?partner=9737)).

Таким образом, на сегодняшний день можно сделать вывод, что для работы с **SQLite в Delphi** нет никаких препятствий. Ваши возможности ограничиваются лишь вашими потребностями в той или иной функциональности компонентов. Я же, как и три года назад, в работе использую компоненты от Devart **LiteDAC**. И теперь перейдем непосредственно к работе с SQLite в Delphi.

**5. Установка и настройка LiteDAC**

[Скачать пробную версию](https://www.devart.com/litedac/) компонентов можно с официального сайта Devart. После того, как установщик скачан, запускаем его и устанавливаем компоненты. После установки компонентов на вкладке LiteDAC в Delphi появится вот такой набор компонентов:

[LiteDAC](http://www.webdelphi.ru/wp-content/uploads/2016/08/LiteDAC.png)

Так как компоненты LiteDAC распространяются в двух редакциях: **Standard** и **Professional**, то набор компонентов у них разный. Так в версии LiteDAC **Standard** **Edition** Вы сможете воспользоваться следующими компонентами для доступа к SQLite:

|  |  |  |
| --- | --- | --- |
| tliteconnection | TLiteConnection | Компонент для создания подключения к БД SQLite. |
| tlitequery | TLiteQuery | Компонент для выполнения запросов и работы наборами данных. |
| tlitesql | TLiteSQL | Компонент для выполнения SQL-запросов, которые не возвращают наборы данных |
| tlitetable | TLiteTable | Компонент позволяет получать и изменять данные в одной таблице БД без написания SQL-запросов |
| tliteupdatesql | TLiteUpdateSQL | Компонент позволяет настроить операции обновления для наборов данных. |
| tlitedatasource | TLiteDataSource | Компонент для создания связей между компонентами LiteDAC, содержащими наборы данных и элементами управления на форме. |
| tlitescript | TLiteScript | Компонент для выполнения последовательности SQL-запросов |
| tlitesqlmonitor | TLiteSQLMonitor | Компонент для мониторинга выполнения SQL-запросов к БД SQLite |
| tliteconnectdialog | TLiteConnectDialog | Компонент для создания пользовательских диалогов для запроса логинов/паролей и ключей шифрования БД. |
| tvirtualtable | TVirtualTable | Компонент для хранения наборов данных в памяти. |

Профессиональная версия LiteDAC (**Professional Edition**) расширяется следующими компонентами

|  |  |  |
| --- | --- | --- |
| tliteuserfunction | TLiteUserFunction | Компонент для определения пользовательских функций для будущего использования в SQL-запросах |
| tliteloader | TLiteLoader | Компонент обеспечивает быструю загрузку внешних данных в базу |
| tlitedump | TLiteDump | Компонент обеспечивает создание дампов для базы данных или отдельных её частей и восстановления БД из этих дампов. |
| tlitemetadata | TLiteMetaData | Компонент для извлечения мета-данных из базы данных |
| tliteencryptor | TLiteEncryptor | Компонент для шифрования/дешифрования базы данных SQLite |
| tcrbatchmove | TCRBatchMove | Компонент обеспечивает обмен данными между всеми потомками TDataSet. |

Со всеми [отличиями версий LiteDAC](https://www.devart.com/litedac/editions.html) вы можете ознакомиться на официальном сайте, я же обращу внимание на три важных, на мой взгляд, **преимущества профессиональной версии**:

1. Возможность использовать прямой доступ (Direct Mode) к БД SQLite
2. Возможность использовать шифрование базы данных (в режиме Direct Mode)
3. Возможность разработки приложений для работы с базами данных под Android и iOS

Перейдем к разработке нашего первого приложения для работы с SQLite в Delphi.

**6. Первая база данных SQLite в Delphi**

Пусть в нашей базе данных будет храниться, например, информация о курсе валют от ЦБ РФ. Для этого разберемся со структурой нашей будущей базы данных и типами данных.

В SQLite, согласно [официально документации](http://www.sqlite.org/faq.html#q2), поддерживаются следующие типы данных: **INTEGER, REAL, TEXT, BLOB, NULL**. Остальные типы данных могут эмулироваться через строки.

Так как мы будем использовать [API ЦБ РФ](http://www.cbr.ru/scripts/Root.asp?PrtId=SXML) по курсам валют,  то нам потребуется создать в нашей  базе данных как минимум две таблицы:

1. Справочник кодов валют
2. Данные по курсам валют на определенную дату.

**Таблица справочника кодов** валют будет содержать следующие столбцы:

1. идентификатор записи, например, *1*
2. идентификатор валюты, например, *R01010*
3. номинал, например, *10*
4. наименование валюты на английском языке, *Australian Dollar*
5. наименование валюты на русском языке, например, *Австралийский доллар*

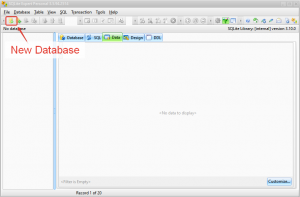
**Таблица данных о курсах валют** будет содержать следующую информацию:

1. идентификатор записи, например, *1*
2. идентификатор валюты, например, *R01010*
3. Дата на которую установлен курс, например, *10.08.2016*
4. Курс валюты по отношению к рублю, например, *49,9973*

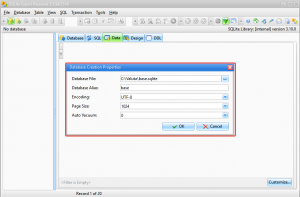
**6.1. Создаем базу данных с помощью SQLite Expert**

Создавать базу данных можно как с помощью наших компонентов непосредственно в Delphi, так и, используя одну из программ, рассмотренных выше. Например, создать в SQLite Expert таблицу со справочником кодов можно следующим образом:

1.Запускаем SQLite Expert и жмем кнопку «New Database»

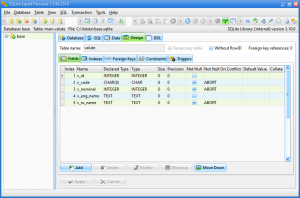
[](http://www.webdelphi.ru/wp-content/uploads/2016/08/2016-08-11_04-50-04.png)

2. Задаем имя базы данных, её расположение, кодировку текста и другие параметры

[](http://www.webdelphi.ru/wp-content/uploads/2016/08/2016-08-11_04-54-40.png)

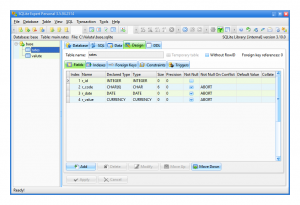
О таком параметре как Vacuum мы поговорим чуть позднее. Пока же жмем «Ok» и переходим к следующему шагу — созданию таблицы.

3. Для создания новой таблицы в базе данных вы можете кликнуть правой кнопкой мыши на названии базы в дереве слева или же воспользоваться разделом меню «Table» и выбрать пункт «New Table». После этого откроется редактор для в котором необходимо указать название новой таблицы, определить необходимые поля, а также указать другие параметры, если это необходимо. В нашем случае, таблицы *«valute»* будет выглядеть следующим образом:

[](http://www.webdelphi.ru/wp-content/uploads/2016/08/2016-08-11_05-09-39.png)

Как видите, несмотря на то, что в SQLite нет такого типа, как CHAR, я все равно его использовал при создании таблицы и это не будет иметь никаких негативных последствий для работы с нашей базой данных.

Аналогичным образом создадим таблицу с курсами валют, опять же используя такие типы данных как DATE и CURRENCY:

[](http://www.webdelphi.ru/wp-content/uploads/2016/08/2016-08-11_05-15-45.png)

Теперь рассмотрим другой вариант создания базы данных — с помощью LiteDAC.

**6.2. Создание базы данных с помощью LiteDAC**

Для создания базы данных воспользуемся компонентом ***TLiteConnection***. Для этого создаем в Delphi новый VCL-проект, бросаем на главную форму компонент TLiteConnection с вкладки LiteDAC и устанавливаем свойство *TLiteConnection.Options.ForceCreateDatabase* в значение *True*. Это позволит нам, в случае отсутствия файла базы данных по указанному в свойстве *Database* пути создать его.

Теперь создадим обработчик события OnBeforeConnect компонента и напишем следующий код:

|  |
| --- |
| **procedure** TForm3.LiteConnection1BeforeConnect(Sender: **TObject**);  **begin**  *//указываем путь к файлу базы данных*  LiteConnection1.Database:=ExtractFilePath(ParamStr(0))+'base.db';  *//указываем путь к клиентской библиотеке sqlite*  LiteConnection1.ClientLibrary:=ExtractFilePath(ParamStr(0))+'sqlite3.dll';  **end**; |

Теперь напишем небольшую процедуру, которая будет создавать необходимые таблицы в базе данных и поместим её в обработчик OnAfterConnect компонента:

|  |
| --- |
| **procedure** TForm3.CreateTables;  **begin**  LiteConnection1.ExecSQL('CREATE TABLE IF NOT EXISTS valute (v\_id INTEGER PRIMARY KEY AUTOINCREMENT, ' + 'v\_code CHAR(6), v\_nominal INTEGER, v\_eng\_name TEXT, v\_ru\_name TEXT)');  LiteConnection1.ExecSQL('CREATE TABLE IF NOT EXISTS rates (r\_id INTEGER PRIMARY KEY AUTOINCREMENT, ' + 'r\_code CHAR(6), r\_date DATE, r\_value CURRENCY)');  **end**; |

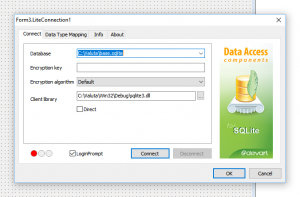
В процедуре CreateTables мы проверяем существует ли необходимая нам таблица в базе данных и, если таблицы нет, то создаем её. Теперь нам только остается подключиться к базе данных и проверить как работает наша программа. Пишем в обработчике OnCreate главной формы приложения всего одну строку:

|  |
| --- |
| **procedure** TForm3.FormCreate(Sender: **TObject**);  **begin**  LiteConnection1.Connected := **True**;  **end**; |

После первого запуска программы рядом с exe-файлом появится файл базы данных с необходимыми таблицами. При этом не забывайте, что рядом с exe-файлом должен находиться файл sqlite3.dll.

Если воспользоваться работой LiteDAC в режиме Direct Mode (прямой доступ), то наличие DLL рядом с exe-файлом не обязательно

В том случае, если Вам необходимо протестировать подключение к базе данных или настроить его в Design-Time, вы можете воспользоваться редактором свойств подключения у компонента TLiteConnection, который вызывается двойным кликом мыши по компоненту или вызовом через контекстное меню (пункт «Connection Editor…»). Выглядит редактор следующим образом:

[](http://www.webdelphi.ru/wp-content/uploads/2016/08/2016-08-11_22-15-20.png)

Здесь вы можете указать путь к базе данных и клиентской библиотеке, а также, если включите чек-бокс «Direct» (режим прямого доступа к БД), то дополнительно сможете указать алгоритм шифрования базы данных и ключ шифрования.

Так как на данный момент нет никаких строгих требований к шифрованию данных в SQLite, то LiteDAC использует свои решения, которые не совместимы ни с какими другими. **Поэтому использовать шифрование мы может исключительно в базах SQLite, созданных непосредственно с использованием LiteDAC**. В свою очередь, никакие сторонние приложения не смогут использовать нашу базу данных без использования LiteDAC. Таким образом, если база данных была создана в SQLite Expert’е или другой программе, то шифровать вы её не сможете.

**7. Шифрование базы данных с помощью LiteDAC**

Если Вам необходимо зашифровать базу данных или сами данные, то LiteDAC поддерживает следующие операции:

1. шифровать базу данных;
2. создать новую зашифрованную базу данных;
3. подключения и работы с зашифрованной базе данных;
4. изменить ключ шифрования зашифрованной базы данных;
5. расшифровать зашифрованную базу данных.

Для шифрования/расшифровки файла базы данных, в LiteDAC используется один из следующих алгоритмов:

* Triple DES;
* Blowfish;
* [AES](http://www.webdelphi.ru/2016/05/algoritm-rijndael-aes-v-delphi/)-128;
* AES-192;
* AES-256;
* Cast-128;
* RC4.

Рассмотрим операции шифрования базы данных в LiteDAC.

**7.1. Шифруем существующую базу данных**

Для шифрования существующей базы данных необходимо выполнить следующую последовательность операций:

|  |
| --- |
| LiteConnection1.Database := ExtractFilePath(ParamStr(0)) + 'base.db';  LiteConnection1.Options.Direct:=**True**; *//только в Direct Mode можно использовать шифрование*  *//шифрование базы данных*  LiteConnection1.Options.ForceCreateDatabase := **False**; *//проверяем есть ли файл БД на диске*  LiteConnection1.Options.EncryptionAlgorithm := leAES256; *//указываем алгоритм шифрования*  LiteConnection1.EncryptionKey := ''; *//база не зашифрована - оставляем пока ключ пустым*  LiteConnection1.Open; *//подключаемся к БД*  LiteConnection1.EncryptDatabase ('masterkey'); *//шифруем БД с ключом masterkey* |

После этого база будет зашифрована с использованием алгоритма AES-256 с ключом AES-256.

**7.2. Создаем или открываем зашифрованную базу данных**

Чтобы открыть зашифрованную ранее базу данных необходимо немного модифицировать показанный выше код:

|  |
| --- |
| LiteConnection1.Database := ExtractFilePath(ParamStr(0)) + 'base.db';  LiteConnection1.Options.Direct:=**True**;  *//шифрование базы данных*  LiteConnection1.Options.ForceCreateDatabase := **True**; *//если базы нет, то она будет создана и зашифрована*  LiteConnection1.Options.EncryptionAlgorithm := leAES256;  LiteConnection1.EncryptionKey := 'masterkey'; *//указываем ключ шифрования*  LiteConnection1.Open; |

**7.3. Меняем ключ шифрования базы данных**

Если необходимо сменить ключ шифрования базы данных, то сделать это можно следующим образом:

|  |
| --- |
| LiteConnection1.Database := ExtractFilePath(ParamStr(0)) + 'base.db';  LiteConnection1.Options.Direct:=**True**;  *//шифрование базы данных*  LiteConnection1.Options.ForceCreateDatabase := **False**; *//проверяем есть ли файл БД на диске*  LiteConnection1.Options.EncryptionAlgorithm := leAES256;  LiteConnection1.EncryptionKey := 'masterkey'; *//указываем старый ключ шифрования*  LiteConnection1.Open; *//подключаемся к БД*  LiteConnection1.EncryptDatabase ('root'); *//шифруем БД с новым ключом root* |

Соответственно, если Вам необходимо расшифровать базу данных, то вам необходимо подключиться к ней и вызвать метод EncryptDatabase, указав в качестве нового ключа пустую строку.

**8. Запись данных в базу SQLite**

После подключения к базе данных мы можем добавлять/редактировать и удалять данные. Скорее всего, что я не ошибусь, если скажу, что большинство тех, кто только начал разбираться с устройством SQLite и забросил эти начинания на начальном этапе, столкнулись с проблемой скорости записи данных в базу. Поэтому в этой части обзора я рассмотрю все возможные способы увеличения скорости записи в базу данных, которые вы можете использовать. При этом большая часть способов будут универсальными, то есть использовать Вы их сможете вне зависимости от того установлен ли у вас LiteDAC или нет.

Для того, чтобы продемонстрировать скорость записи данных в базу я буду заполнять таблицу «Справочник валют» (‘valute’).

Так как рассмотрение вопросов, связанных с получением данных от сервера ЦБ РФ через его API выходит за рамки этой статьи, то в статье будет представлена только та часть кода, которая относится непосредственно к работе с базой данных. Полный код программы Вы всегда можете скачать в конце статьи или [со страницы исходников](http://www.webdelphi.ru/razrabotchiku/isxodniki-delphi/).

Информация по каждой валюте будет содержаться вот в таком объекте:

|  |
| --- |
| TValute = **class**  **private**  FCode: **string**;  FNominal: **cardinal**;  FRuName: **string**;  FEngName: **string**;  **public**  **constructor** Create(**const** ACode, ARuName, AEngName: **string**; ANominal: **cardinal**);  **destructor** Destroy;**override**;  **property** Code: **string** **read** FCode **write** FCode;  **property** Nominal: **cardinal** **read** FNominal **write** FNominal;  **property** RuName: **string** **read** FRuName **write** FRuName;  **property** EngName: **string** **read** FEngName **write** FEngName;  **end**; |

Соответственно, весь справочник загружается в виде XML с сайта и сохраняется в списке:

|  |
| --- |
| FValutes: TObjectList; |

Рассмотрим вариант записи данных, который вы можете встретить наиболее часто в Сети при рассмотрении вопросов работы с базами данных (не только с SQLite). Для этого воспользуемся удобным компонентом ***TLiteSQL***. Бросаем на форму приложения компонент TLiteSQL и кнопку TButton. В обработчике OnClik кнопки поместим следующий код:

|  |
| --- |
| **var** Timer: TStopWatch;  **begin**  *{В этом месте мы загружаем список валют с сайта ЦБ РФ и сохраняем его в списке}*  Timer:=TStopwatch.StartNew;  Timer.Start; *//запускаем таймер*  LiteSQL1.SQL.Text := 'INSERT INTO valute (v\_code, v\_nominal, v\_eng\_name, v\_ru\_name) VALUES (:code, :nominal, :eng\_name, :ru\_name)';  **for** I := 0 **to** Pred(CBR.Valutes.Count) **do**  **begin**  LiteSQL1.ParamByName('code').AsString := CBR.Valutes[I].Code;  LiteSQL1.ParamByName('nominal').AsInteger := CBR.Valutes[I].Nominal;  LiteSQL1.ParamByName('eng\_name').AsString := CBR.Valutes[I].EngName;  LiteSQL1.ParamByName('ru\_name').AsString := CBR.Valutes[I].RuName;  LiteSQL1.Execute;  **end**;  Timer.Stop;  ShowMessage(Timer.Elapsed.ToString);*//показываем затраты времени*  **end**; |

Чтобы воспользоваться объектом TStopWatch в uses необходимо подключить модуль System.Diagnostics

В этом примере мы добавляем новые записи в таблицу valutes, используя параметры. Посмотрим, что покажет таймер:

На добавление 60 записей в таблицу ушло 00:00:05.83

Нетрудно предположить какое время мы бы затратили, если бы добавляли не 60, а 6000 записей. Но всё совсем не так печально, как может показаться на первый взгляд. Рассмотрим способы ускорения записи данных в базу.

**8.1. Используем транзакции**

О том, что необходимо использовать транзакции [говорят](http://www.sqlite.org/faq.html#q19) и сами разработчики SQLite. Допишем наш код следующим образом:

|  |
| --- |
| LiteConnection1.StartTransaction; *//запускаем транзакцию*  **try**  LiteSQL1.SQL.Text := 'INSERT INTO valute (v\_code, v\_nominal, v\_eng\_name, v\_ru\_name) VALUES (:code, :nominal, :eng\_name, :ru\_name)';  **for** I := 0 **to** Pred(CBR.Valutes.Count) **do**  **begin**  LiteSQL1.ParamByName('code').AsString := CBR.Valutes[I].Code;  LiteSQL1.ParamByName('nominal').AsInteger := CBR.Valutes[I].Nominal;  LiteSQL1.ParamByName('eng\_name').AsString := CBR.Valutes[I].EngName;  LiteSQL1.ParamByName('ru\_name').AsString := CBR.Valutes[I].RuName;  LiteSQL1.Execute;  **end**;  LiteConnection1.Commit;*//подтверждаем запись*  Timer.Stop;  ShowMessage(Timer.Elapsed.ToString);  **except**  LiteConnection1.Rollback; *//откатываем изменения*  **end**; |

Проверим, что покажет теперь наш счётчик:

00:00:00.1420682

Неплохое ускорение, правда? Но и это ещё не всё. Ускорить запись можно также, используя различные настройки базы данных.

**8.2. Используем настройки базы данных**

Для изменения настроек базы SQLite используются выражения PRAGMA с использованием различных параметров. Рассмотрим, какие из настроек могут ускорить запись данных в базу. Чтобы сменить какую-либо настройку базы данных с использованием LiteDAC, достаточно сделать следующий вызов, например:

|  |
| --- |
| LiteConnection1.ExecSQL('PRAGMA synchronous = OFF'); |

**8.2.1. PRAGMA synchronous**

По умолчанию база SQLite создается с настройками **максимальной безопасности**. В частности, флаг *synchronous* установлен в значение **FULL**. Использование этого значения гарантирует, что все данные будут записаны в базу данных и авария или сбой питания не нарушат целостность базы SQLite. Однако, за безопасность мы платим, конечно же, временем. Поэтому, если Вы хотите ускорить запись данных в базу, пожертвовав тем самым безопасностью, то можете использовать следующие значения synchronous:

* **0 или OFF** — синхронизация полностью отключена. При таком режиме работы, в случае аварии, возможен выход из строя базы данных
* **1 или NORMAL** — SQLite синхронизирует данные только в самых критичных ситуациях,  синхронизация запускается намного реже, чем при режиме FULL.
* **2 или FULL** — максимальный уровень безопасности.

Проверим, насколько ускорится запись данных в случае отключенной синхронизации. Для этого в обработчике OnAfterConnect компонента LiteConnection пишем:

|  |
| --- |
| LiteConnection1.ExecSQL('PRAGMA synchronous = OFF') |

Запускаем приложение и смотрим на значение таймера:

00:00:00.011

По сравнению с предыдущим значением, **скорость записи возросла в 10 раз**. Однако не забывайте, что в этом случае нам никто не гарантирует в случае сбоя целостность базы данных. Используйте эту настройку на свой страх и риск.

**8.2.2. PRAGMA temp\_store**

Еще один способ ускорения работы с БД SQLite — указать место хранения временных таблиц и индексов.  
temp\_store указывает место хранения временных таблиц и индексов и может принимать следующие значения:

* 0 или DEFAULT
* 1 или FILE
* 2 или MEMORY

Использовать этот параметр стоит с осторожностью, так как его изменение, например, в Android может привести к тому, что ваша БД будет «поломана». Посмотрим, какой результат покажет таймер при использовании этого флага со значением *MEMORY*.

00:00:00.11

Как видите, влияние этого флага на скорость записи не так значительно, как при использовании режима синхронизации, но, тем не менее ускорение есть.

**8.3. Используем возможности LiteDAC**

Компоненты LiteDAC дают нам уникальную возможность ускорения записи данных в таблицу используя пакетные операции (Batch operations). Рассмотрим как ими воспользоваться. Для этого модифицируем наш код добавления данных по валютам в базу SQLite следующим образом:

|  |
| --- |
| LiteSQL1.SQL.Text := 'INSERT INTO valute (v\_code, v\_nominal, v\_eng\_name, v\_ru\_name) VALUES (:code, :nominal, :eng\_name, :ru\_name)';  *//определяем типы данных параметров*  LiteSQL1.Params[0].DataType:=ftString;  LiteSQL1.Params[1].DataType:=ftInteger;  LiteSQL1.Params[2].DataType:=ftString;  LiteSQL1.Params[3].DataType:=ftString;  *//определяем количество записей в пакете*  LiteSQL1.Params.ValueCount:=CBR.Valutes.Count;  *//собираем пакет*  **for** I := 0 **to** Pred(CBR.Valutes.Count) **do**  **begin**  LiteSQL1.Params[0][i].AsString := CBR.Valutes[I].Code;  LiteSQL1.Params[1][i].AsInteger := CBR.Valutes[I].Nominal;  LiteSQL1.Params[2][i].AsString := CBR.Valutes[I].EngName;  LiteSQL1.Params[3][i].AsString := CBR.Valutes[I].RuName;  **end**;  *//выполняем пакетную операцию*  LiteSQL1.Execute(CBR.Valutes.Count); |

Смотрим на значение таймера:

00:00:00.10

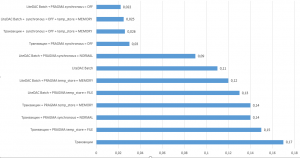
Значение сопоставимо с тем, которое мы получали при использовании транзакций, хотя в приведенном примере с batch operations транзакции не использовались. Более того, даже, если вы будете использовать в приведенном выше примере транзакции — на скорости работы программы это практически никак не отразиться. Здесь стоит обратить на один немаловажный момент, а именно на то, как в примере определялись параметры.  
Думаю, что, Вы согласитесь с тем, что логичнее было бы обращаться к параметру по его имени, например, так:

|  |
| --- |
| LiteSQL1.ParamByName('code').AsString := CBR.Valutes[I].Code; |

Однако разработчики LiteDAC не рекомендуют использовать такой подход применительно к пакетным операциям, так как в этом случае заполнение массива данных будет происходить медленнее в силу того, что каждый параметр придётся искать в каждой итерации цикла, что неизбежно отразиться на скорости записи данных в базу SQLite.

**8.4. График скорости записи данных в базу SQLite**

В качестве небольшого вывода по этой части обзора я приведу график скорости записи в базу данных 1000 элементов в зависимости от различных настроек базы данных, чтобы вы могли сами определиться с тем, какие настройки использовать (кликните на рисунок, чтобы увеличить):

[](http://www.webdelphi.ru/wp-content/uploads/2016/08/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA.png)

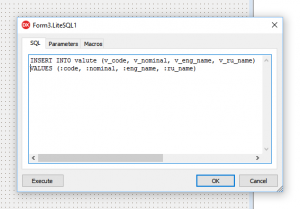
Каждый тест проводился не менее 5 раз и рассчитывалось среднее значение затраченного времени в секундах. Добавление 1000 записей без использования транзакций, batch и настроек БД заняла около 130 секунд, поэтому на графике этот тест не отражен.

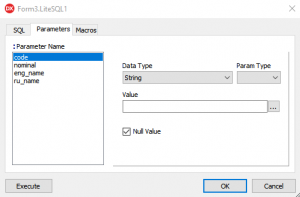
Какой режим записи данных использовать — решать только вам. Но не стоить забывать, что в погоне за скоростью мы теряем в надежности. Для себя я определил следующий режим работы:

1. PRAGMA synchronous = NORMAL
2. PRAGMA temp\_store = DEFAULT

**8.5. Настройка TLiteSQL в Design-Time**

Ещё один способ чуточку ускорить запись данный в SQLite — настроить необходимые параметры компонента в Design-Time. С одной стороны — это упростит исходный код программы, а с другой стороны — немного (самую малость) ускорит работу программы при записи данных. Продемонстрирую это на последнем примере, когда мы использовали batch. Двойным кликом по TLiteSQL вызываем редактор запроса и переносим запрос на вставку новой записи:

[](http://www.webdelphi.ru/wp-content/uploads/2016/08/2016-08-12_03-37-39.png) Теперь переходим на вкладку Parameters в редакторе и определяем типы данных параметров:

[](http://www.webdelphi.ru/wp-content/uploads/2016/08/2016-08-12_03-38-56.png)

После этого можно закрыть редактор и удалить из процедуры добавления новых записей теперь уже лишние строки, то есть, оставить следующий код:

|  |
| --- |
| LiteSQL1.Params.ValueCount := CBR.Valutes.Count;  **for** I := 0 **to** Pred(CBR.Valutes.Count) **do**  **begin**  LiteSQL1.Params[0][I].AsString := CBR.Valutes[i].Code;  LiteSQL1.Params[1][I].AsInteger := CBR.Valutes[i].Nominal;  LiteSQL1.Params[2][I].AsString := CBR.Valutes[i].EngName;  LiteSQL1.Params[3][I].AsString := CBR.Valutes[i].RuName;  **end**;  LiteSQL1.Execute(CBR.Valutes.Count); |

Теперь, разобравшись немного с запись данных перейдем к следующему вопросу.

**9. Как исключить дублирование записей в таблице?**

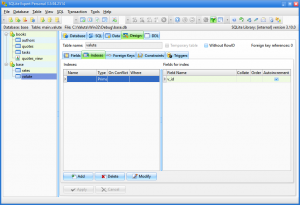
Этот вопрос касается не столько работы с компонентами в Delphi, сколько работы с самой СУБД SQLite. Самый очевидный алгоритм, который приходит в голову (и, опять же, самый часто используемый) выглядит следующим образом:

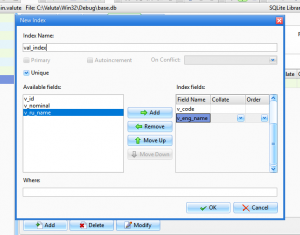
1. Делаем запрос SELECT, используя WHERE и определяем есть ли запись в базе
2. Если записи нет — добавляем

Например, если нам надо определить, если в нашей таблице запись, касающаяся доллара, то мы может сделать такой запрос SELECT к базе:

|  |
| --- |
| **SELECT** \* **FROM** valute **WHERE** v\_eng\_name='US Dollar' |

Если результат вызова будет содержать запись, то, соответственно, добавлять ничего не нужно. Вместе с этим, есть и другой способ исключения дублирования записей в базу SQLite. Для этого необходимо создать уникальный индекс и с помощью него «отсеивать» повторяющиеся записи. Чтобы создать индекс в SQLite Expert необходимо выбрать интересующую нас таблицу (это будет valute), затем перейти на вкладку Design—>Indexes:

[](http://www.webdelphi.ru/wp-content/uploads/2016/08/2016-08-12_04-04-14.png)Теперь нажимаем кнопку «Add» под списком индексов и выбираем поля, которые будут составлять наш уникальный индекс. Пусть это будут поля *v\_code* и *v\_eng\_name.*Таким образом, Вы должны получить следующий результат:

[](http://www.webdelphi.ru/wp-content/uploads/2016/08/2016-08-12_04-07-10.png)

Обратите внимание, что, если при создании уникального индекса в таблице уже есть записи у которых повторяются поля, составляющие индекс, то Вы получите ошибку «SQLite UNIQUE constraint failed:»

Ту же самую операцию, но с использованием LiteDAC можно сделать, например, так:

|  |
| --- |
| LiteConnection1.ExecSQL('CREATE UNIQUE INDEX IF NOT EXISTS val\_idx ON valute (v\_code, v\_eng\_name)'); |

И, теперь, модифицируем наш код запроса на добавление новой записи следующим образом:

|  |
| --- |
| **INSERT** **OR** **IGNORE** **INTO** valute (v\_code, v\_nominal, v\_eng\_name, v\_ru\_name)  **VALUES** (:code, :nominal, :eng\_name, :ru\_name) |

Теперь в таблицу будут попадать записи только с уникальным сочетанием кода и названия на английском языке. На скорости работы программы это никак не отразиться, однако, код программы станет чуть меньше и понятнее.

**10. Как сократить размер файла базы данных?**

Ещё один момент, который может несколько смутить начинающих разбираться с SQLite — размер базы данных. Со временем вы можете заметить, что размер базы данных очень сильно растёт (особенно, если данных очень много) и не уменьшается даже в том случае, если вы удаляете данные из таблиц. Это связано с тем, что SQLite при удалении записей из таблицы свободное пространство файла не удаляется, а используется в последующей работе для вставки новых записей. Чтобы сжать файл после удаления записей можно пойти двумя путями

1. В необходимый момент вызвать команду **VACUUM**, например, так:

|  |
| --- |
| LiteConnection1.ExecSQL('VACUUM') |

В этом случае вы сами определяете, когда необходимо сжать файл.  
2. При создании базы данных включить автоматическое сжатие с помощью команды **PRAGMA auto\_vacuum**:

|  |
| --- |
| LiteConnection1.ExecSQL('PRAGMA auto\_vacuum = FULL') |

В этом случае SQLite будет сама сжимать файл после удаления записей, но при этом незначительно снизиться производительность. На этом, пожалуй, все вопросы, которые так или иначе волновали тех, кто только присматривался к SQLite, исчерпаны: рассмотрели предельные значения SQLite, как повысить производительность SQLite, как сократить размер файла базы данных, а также, каким образом можно шифровать базу данных с использованием LiteDAC. Перейдем к следующему вопросу — получению данных из базы SQLite.

**11. Использование обзоров (VIEW) в SQLite**

На данный момент у нас есть две таблицы — справочник валют и курсы валют. Допустим, нам необходимо выбрать из обоих таблиц следующие поля:

1. Название валюты на русском языке ( таблица valute)
2. Дату на которую установлен курс (таблица rates)
3. Курс (таблица rates)

Выбрать необходимые данные можно довольно легко, используя следующий запрос:

|  |
| --- |
| **SELECT** valute.[v\_ru\_name], rates.[r\_date], rates.[r\_value] **FROM** valute, rates **WHERE** valute.[v\_code]=rates.[r\_code] |

В этом запросе нет ничего сложного. Однако, если запрос будет достаточно сложный и громоздкий или вам необходимо часто обращаться к этим таблицам, фильтруя данные, то можно его представить с помощью VIEW и работать с таким обзором как с обычной таблицей. Сделать обзор (VIEW) по представленному выше запросу можно так:

|  |
| --- |
| LiteConnection1.ExecSQL('CREATE VIEW IF NOT EXISTS [val\_rates] AS select valute.[v\_ru\_name], rates.[r\_date], rates.[r\_value] from valute, rates where valute.[v\_code]=rates.[r\_code]'); |

Здесь мы создали обзор с именем val\_rates. Теперь мы можем просматривать данные, обращаясь к обзору как к обычной таблице:

|  |
| --- |
| **SELECT** \* **FROM** val\_rates |

Или использовать компонент TLiteTable:

|  |
| --- |
| LiteTable1.TableName:='val\_rates';  LiteTable1.DataTypeMap.AddFieldNameRule('v\_ru\_name', ftString);  LiteTable1.DataTypeMap.AddFieldNameRule('r\_date', ftDate);  LiteTable1.DataTypeMap.AddFieldNameRule('v\_value', ftCurrency);  LiteTable1.Open; |

При работе с обзорами в SQLite необходимо отметить следующее:

1. Обзоры (view) являются — это виртуальные таблицы. Их содержимое является результатом выполнения запросов к другим таблицам. И, хотя обзоры похожи на настоящие таблицы, они ими не являются. Содержимое обзора динамически генерируется во время обращения к обзору.
2. Вы не можете применять к обзоры операции вставки, обновления и удаления данных.

**12. Дополнительные возможности по работе с SQLite в Delphi при использовании LiteDAC**

Использование LiteDAC позволяет нам без проблем не только добавлять/удалять/редактировать данные, но, так же и предоставляет нам некоторые дополнительные удобные возможности по работе с базами данных, например, лёгкое создание дампов баз данных, бэкапов, работу с макросами и т.д. Рассмотрим некоторые из них

**12.1. Создание дампа базы SQLite**

Для создания дампа базы данных предусмотрен отдельный компонент TLiteDump.

Компонент **TLiteDump** содержит следующие свойства:

|  |  |
| --- | --- |
| Mode | Режим создания дампа. Может принимать одно из трех значений:   * **dmAll** — создается бэкап для объектов схемы, всех таблиц и данных в этих таблицах. Этот режим установлен по умолчанию. * **dmData** — создается дамп только для данных * **dmSchema** — создается дамп только для объектов схемы. |
| ObjectTypes | Типы объектов для которых будет создаваться дамп. Может содержать следующие значения из множества TLiteDumpObjects: **doTables**, **doViews**, **doTriggers**, **doIndexes**. |
| Options | Задает поведение компонента и может содержать следующие элементы множества:   * **AddDrop** — добавляет запрос на удаление объекта перед его созданием * **GenerateHeader** — добавляет специальный заголовок в скрипт * **QuoteNames** — заключает все имена объектов БД в кавычки. |

Рассмотрим пример создания дампа нашей базы данных. Бросаем на главную форму приложения компонент TLiteDump и оставляем свойства компонента, установленные по умолчанию. Теперь, например, в обработчике OnClick кнопки можно написать следующий код:

|  |
| --- |
| **procedure** TForm3.Button3Click(Sender: **TObject**);  **begin**    LiteDump1.BackupToFile('bakup.sql'); *// создаем дамп и сохраняем его в поток*  **end**; |

Таким образом, рядом с exe-файлом создастся файл с именем backup.sql, содержащий всю информацию из базы данных. Аналогичным образом можно восстановить данные БД из дампа:

|  |
| --- |
| LiteDump1.RestoreFromFile('bakup.sql'); |

Также компонент TLiteDump поддерживает сохранение и восстановление базы данных из потока. Для этого используются, соответственно, методы:

|  |
| --- |
| LiteDump1.BackupToStream(AStream: TStream)  LiteDump1.RestoreFromStream(AStream: TStream) |

Просто, удобно и функционально.

**12.2. Макросы в LiteDAC**

Макросы позволяют менять весь SQL-запрос динамически. Например, мы часто используем SELECT для выборки данных из различных таблиц SQLite. Такие запросы мы можем выполнять как обычно, например:

|  |
| --- |
| LiteQuery1.SQL.Text:='SELECT \* FROM valute';  LiteQuery1.Open; *//поработали с записями*  LiteQuery1.Close;  *//новый запрос для другой таблицы*  LiteQuery1.SQL.Text:='SELECT \* FROM rates';  LiteQuery1.Open; |

А можем воспользоваться макросом. Имя макроса всегда начинается с символа **&**. Например, мы можем переписать пример выше с использованием макроса так:

|  |
| --- |
| LiteQuery1.SQL.Text:='SELECT \* FROM &amp;TableName';  LiteQuery1.MacroByName('TableName').Value:='valute';  LiteQuery1.Open; *//поработали с записями*  LiteQuery1.Close;  LiteQuery1.MacroByName('TableName').Value:='rates';  LiteQuery1.Open; |

Используя макросы, мы можем менять и другие части запросов, например, изменять перечень запрашиваемых полей, изменять содержимое WHERE и так далее. Таким образом макросы в LiteDAc позволяют, опять же, сократить исходный код программы и, возможно, сделать его чуть более наглядным (хотя здесь смотря с какой стороны посмотреть).

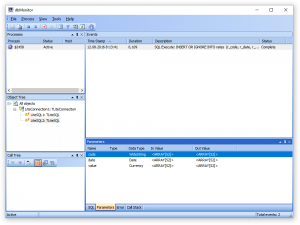
**12.3 Мониторинг запросов к SQLite в LiteDAC**

В приведенных выше примерах по вставке данных в таблицу SQLite для замера затраченного времени я использовал модуль System.Diagnostics. Однако, если Вам необходимо для отладки отслеживать вообще все запросы к БД, то можно воспользоваться компонентом **TLiteSQLMonitor** и получать всю необходимую информацию по обращению к базе.

Чтобы воспользоваться этой замечательной возможностью Вам необходимо:

1. Скачать с сайта Devart программу DBMonitor. Скачивается она бесплатно по ссылке в меню LiteDAC в Delphi.
2. Бросить на форму компонент TLiteSQLMonitor и установить его свойство Active в значение True
3. Для всех компонентов, которые вы хотите отслеживать в DBMonitor необходима выставить свойство Debug в значение True.
4. Запустить программу
5. Запустить DBMonitor
6. Отслеживать обращения к базе данных.

Ниже представлен скрин работающего DBMonitor:

[](http://www.webdelphi.ru/wp-content/uploads/2016/08/2016-08-12_08-14-23.png)

Как видите, здесь нам показываются и затраты времени на выполнение той или иной операции и значения параметров и многое другое, что может потребоваться для отладки вашего приложения.

**13. Подведем итог**

Как видите, SQLite — достаточно удобная и легкая в использовании СУБД с массой возможностей и широким набором самых различных настроек. И то, что на первый взгляд, может показаться в ней непродуманным (например, растущий размер файла) далеко не всегда является таковым. С выходом Delphi XE3 появилась полноценная поддержка работы с SQLite, что называется «из коробки». Однако, наравне с компонентами, поставляемыми вместе с Delphi, развиваются (и достаточно успешно) различные сторонние компоненты для доступа к SQLite, как, например, приведенные в этой статье компоненты LiteDAC. Конечно, каждый выбирает самостоятельно компоненты для работы: кому-то непременно надо использовать только то, что есть в Delphi, кто-то использует исключительно Open Source. Я же предпочитаю использовать то, что мне кажется наиболее удобным и простым в использовании, поэтому уже несколько лет для работы с SQLite в Delphi использую исключительно LiteDAC.

Ниже представлен исходник программы, примеры из которой я приводил выше в статье, однако обращу ваше внимание — это не полноценный продукт для работы с курсами валют. Это лишь набросок программы, демонстрирующий работу с SQLite, который, возможно Вам пригодится.