Руководство пользователя по SQLiteStudio

**Содержание**

* [Установка и запуск](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#installation-and-running)
  + [Поддерживаемая платформа](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#supported-platforms)
  + [Установка двоичного пакета](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#installing-binary-package)
  + [Установка из исходного кода](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#installing-from-source-code)
* [Основы](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#basics)
  + [Создание новой базы данных](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#creating-new-database)
    - [Постоянная и временная базы данных](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#permanent-vs-transient-database)
  + [Использование существующей базы данных](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#using-existing-database)
  + [Управление списком баз данных](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#managing-database-list)
    - [Редактирование сведений о базе данных](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#editing-database-details)
    - [Организация списка баз данных](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#organizing-database-list)
    - [Удаление базы данных из списка](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#deleting-database-from-the-list)
  + [Выполнение SQL-запросов](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#executing-sql-queries)
    - [Окно редактора SQL](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#sql-editor-window)
    - [Кнопки на панели инструментов](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#buttons-on-toolbar)
    - [Ввод SQL-запросов](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#typing-sql-queries)
    - [Завершение кода](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#code-completion)
    - [История выполнения запроса](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#query-execution-history)
    - [Встроенные функции SQL](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#built-in-sql-functions)
    - [Редактирование данных](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#editing-the-data)
      * [Диалоговое окно редактора значений](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#value-editor-dialog)
    - [Прозрачное прикрепление базы данных](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#transparent-database-attaching)
  + [Создание и редактирование таблиц](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#creating-and-editing-tables)
    - [Диалоговое окно "столбец"](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#column-dialog)
      * [Ограничения столбцов](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#column-constraints)
    - [Таблица ограничений](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#table-constraints)
    - [Без таблиц ROWID](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#without-rowid-tables)
    - [Виртуальные столы](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#virtual-tables)
  + [Создание и редактирование индексов](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#creating-and-editing-indexes)
    - [Частичные индексы](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#partial-indexes)
  + [Создание и редактирование триггеров](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#creating-and-editing-triggers)
    - [Триггер для представления](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#trigger-for-a-view)
  + [Создание и редактирование видов](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#creating-and-editing-views)
  + [Перемещение и копирование объектов между базами данных](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#moving-and-copying-objects-between-databases)
  + [Экспорт схемы и данных](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#exporting-schema-and-data)
  + [Импортировать данные](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#importing-data)
  + [Полезные сочетания клавиш](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#useful-keyboard-shortcuts)
  + [Диалог настройки](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#configuration-dialog)
    - [Настройка редакторов типов данных](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#customizing-data-type-editors)
  + [Опция командной строки](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#command-line-options)
  + [Файл конфигурации SQLiteStudio](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#sqlitestudio-configuration-file)
  + [Обновления SQLiteStudio](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#sqlitestudio-updates)
    - [Автомат](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#automatic)
    - [Руководство](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#manual)
* [Расширенное использование](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#advanced-usage)
  + [Пользовательские функции SQL](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#custom-sql-functions)
  + [Пользовательские параметры сортировки](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#custom-collations)
  + [Расширения SQLite](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#sqlite-extensions)
  + [Заполнение таблиц](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#populating-tables)
  + [История DDL](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#ddl-history)
  + [Преобразование версии базы данных SQLite](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#converting-sqlite-database-version)
* [Интерфейс командной строки (CLI)](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#command-line-interface-cli)
* [Устранение неисправностей](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#troubleshooting)
  + [SQLiteStudio разбился! Что мне делать?](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#sqlitestudio-crashed-what-do-i-do)
  + [Я сохранил двоичные данные в базу данных SQLite 2, но теперь они повреждены, пусты или усечены!](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#i-have-saved-binary-data-into-sqlite-2-database-but-now-its-damaged-empty-or-truncated)
  + [Я использую метод ввода Fcitx, но он не работает в SQLiteStudio](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#im-using-fcitx-input-method-but-it-doesnt-work-in-sqlitestudio)
  + [Смотреть также](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#see-also)

**Установка и запуск**

**Поддерживаемая платформа**

| **Платформа** | **Статус** | **Двоичные файлы** | **Примечания** |
| --- | --- | --- | --- |
| Windows (32bit) | Поддерживаемый | ДА |  |
| Windows (64bit) | Поддерживаемый | ДА |  |
| Linux (32bit) | Поддерживаемый | ДА |  |
| Linux (64bit) | Поддерживаемый | ДА |  |
| Mac OS X (ix86 64bit) | Поддерживаемый | ДА | Это включает в себя большинство современных Маков, см. <http://support.apple.com/kb/ht3696>для получения подробной информации. |
| Mac OS X (ix86 32bit) | Не поддерживаемый | НЕТ |  |
| Mac OS X (PowerPC) | Не поддерживаемый | НЕТ |  |
| операционная система FreeBSD | Должен работать, а не тестироваться | НЕТ |  |
| Солярис | Должен работать, а не тестироваться | НЕТ |  |
| Другая операционная система | Не поддерживаемый | НЕТ |  |

**Установка двоичного пакета**

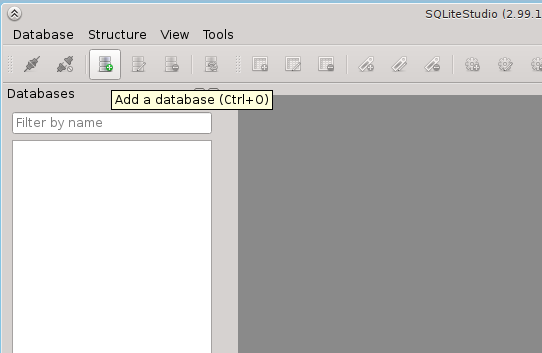
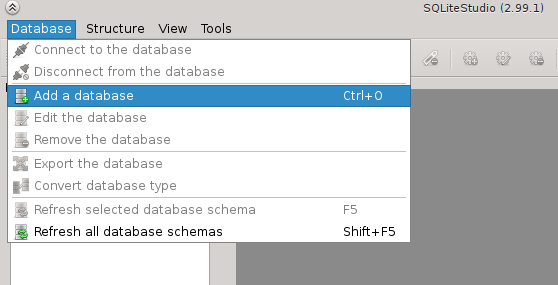
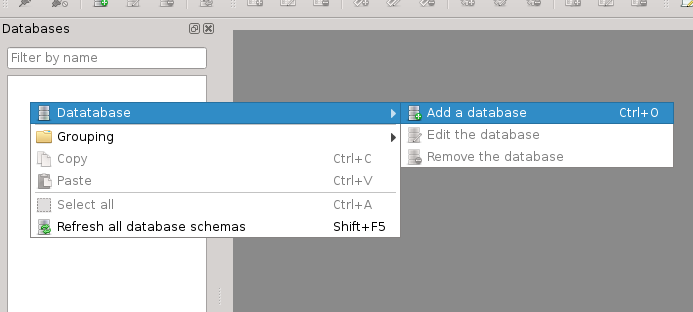
**Установка из исходного кода**

Смотрите [компиляции приложения из источники](https://github.com/pawelsalawa/sqlitestudio/wiki/Compiling_application_from_sources).

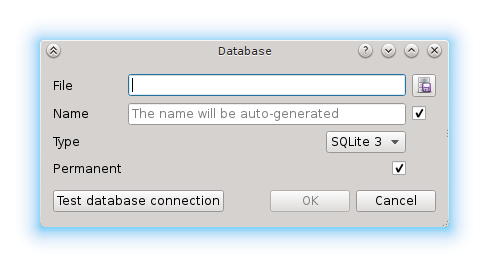
**Основы**

**Создание новой базы данных**

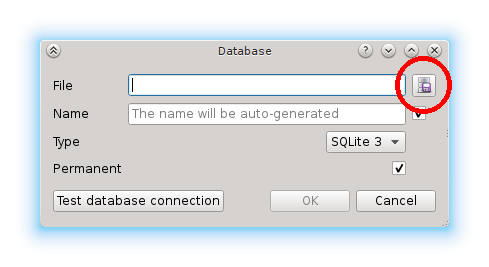
Вы можете создать новую базу данных тремя способами:

1. Нажмите кнопку на панели инструментов: 
2. Нажмите на пункт главного меню: 
3. Щелкните правой кнопкой мыши по списку базы данных и выберите пункт из контекстного меню: 

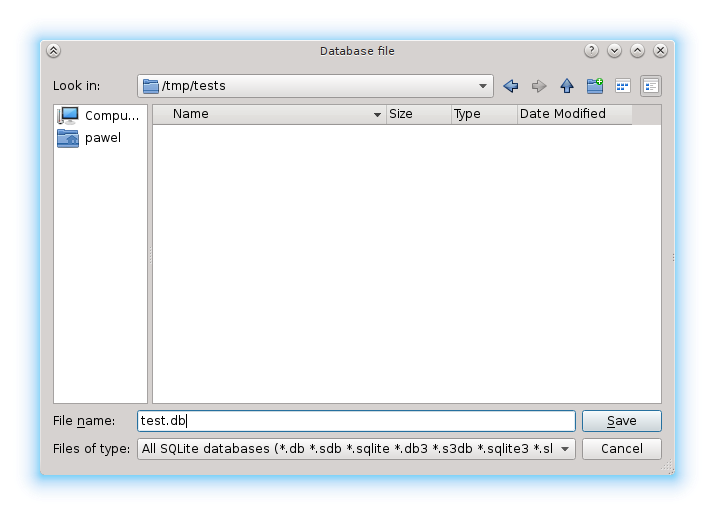
Затем появится диалоговое окно базы данных:



Нажмите на кнопку обзор:



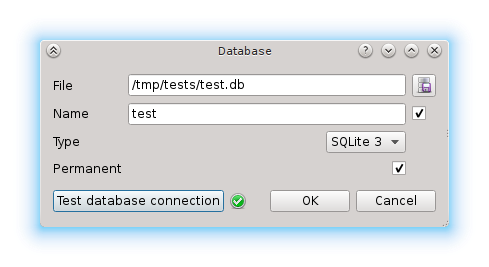
Вы увидите диалоговое окно просмотра файловой системы (это может выглядеть по-разному в различных операционных системах):



В этом диалоговом окне перейдите в каталог, где вы хотите создать свою базу данных и введите имя файла для вашей базы данных (на рисунке выше это " тест.децибел.)" Нажмите Кнопку "Сохранить". Теперь диалоговое окно базы данных содержит Имя файла и имя базы данных. Вы можете изменить имя на все, что вы хотите (в этом случае вы должны отключить флажок на правой стороне), или просто оставить его, как есть.

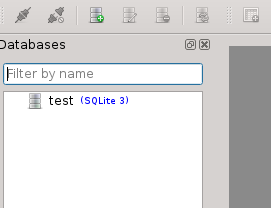
**Имя базы данных-это просто символическое имя, псевдоним, используемый SQLiteStudio для представления базы данных в списке. Вы можете выбрать любое имя, которое вы хотите, он просто имеет быть уникальным в контексте баз данных, уже добавленных в SQLiteStudio.**

Вы можете нажать "Test connection", чтобы убедиться, что ваш файл базы данных может быть создан в указанной локализации (в отношении разрешений каталога, свободного места и т.д.). Если все в порядке, то вы увидите зеленый значок рядом с ним:

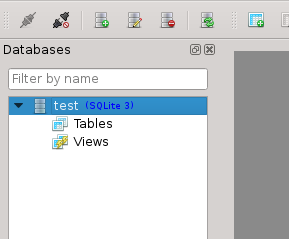


Если возникнут проблемы, появится красный значок.

Теперь вы можете нажать "Ok", чтобы создать базу данных. Вы увидите его в списке базы данных слева:

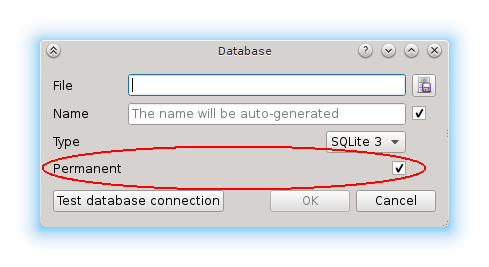


Дважды щелкните, чтобы открыть его:



**Постоянная и временная базы данных**

Возможно, вы заметили" постоянную " проверку в диалоговом окне базы данных:



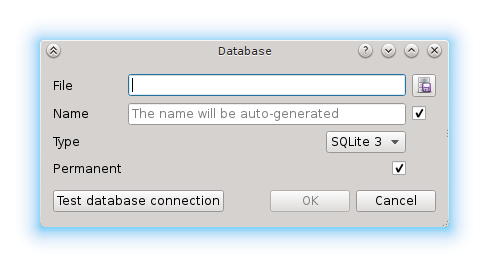
Когда эта опция включена, то база данных постоянно добавляется в SQLiteStudio. Это означает, что при закрытии SQLiteStudio и запуске его снова база данных останется в списке.

Если этот параметр не установлен, то базы данных исчезнут из списка после перезапуска SQLiteStudio.

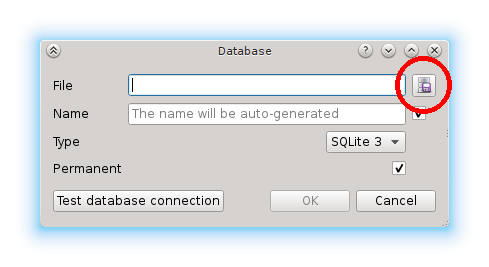
**Использование существующей базы данных**

Добавление существующей базы данных в список очень похоже на создание новой базы данных.

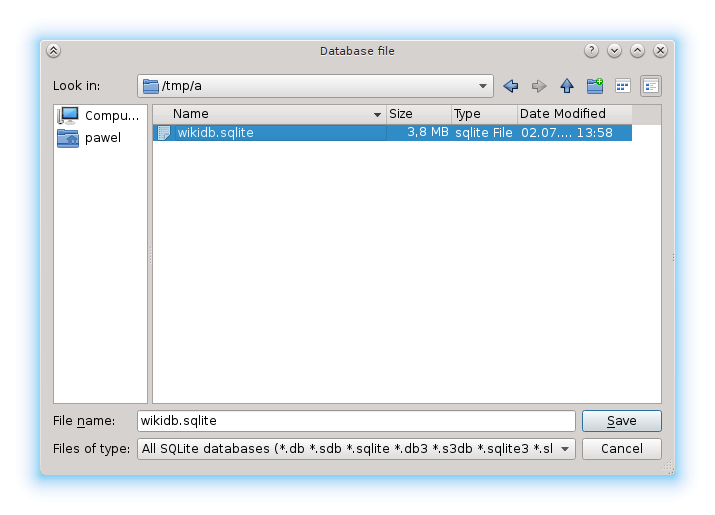
Повторите шаги сверху, util вы видите диалоговое окно базы данных:



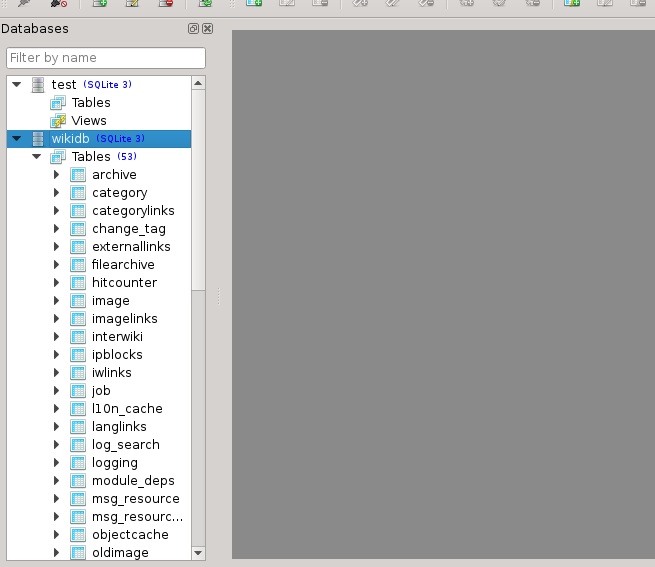
Затем вы нажимаете (как и раньше) на кнопку обзор:



В диалоговом окне просмотра файлов выберите существующий файл базы данных:



Затем нажмите "Сохранить" и " ОК " в следующем диалоговом окне, и ваша база данных будет добавлена в список. Вы можете дважды щелкнуть на нем, чтобы открыть его:



Также можно использовать механизм перетаскивания: перетащите файл базы данных из файлового менеджера и поместите его в список базы данных, затем подтвердите диалог базы данных.

**Управление списком баз данных**

**Редактирование сведений о базе данных**

Вы можете редактировать данные базы данных в любой момент. Просто щелкните правой кнопкой мыши на нем и выбрать "редактировать базу данных" из контекстного меню. Появится стандартное диалоговое окно базы данных.

Изменение имени базы данных разрешено, так как новое имя уникально для всех баз данных, находящихся в списке.

**Имя базы данных-это просто символическое имя, псевдоним, используемый SQLiteStudio для представления базы данных в списке.**

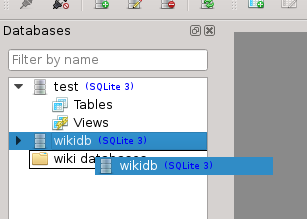
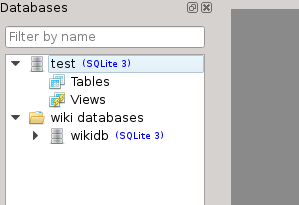
При изменении пути к файлу базы данных это будет означать, что под тем же именем вы получите доступ к новому файлу базы данных. **Он не будет копировать, ни переместить старый файл базы данных в новое место.**

При редактировании сведений о базе данных невозможно изменить ее тип (SQLite2, SQLite3 или любой другой), поскольку это разрешено только при создании новой базы данных. Если вы хотите конвертировать SQLite версии в другую, использовать [базы данных преобразования диалог](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#converting-sqlite-database-version).

**Организация списка баз данных**

Вы можете перемещать базы данных вверх и вниз с помощью мыши "drag & drop". Можно создать виртуальные "каталоги", в которых можно группировать базы данных. Это помогает поддерживать большое количество баз данных в списке.

Чтобы создать группу, щелкните правой кнопкой мыши на группе базы данных и выберите "группировка" → "создать группу". Затем можно перетаскивать базы данных в эту группу:

Группы могут быть вложенными.

Если вы щелкните правой кнопкой мыши на базе данных и выберите "создать группу", то база данных будет автоматически перемещена в группу после создания группы.

**Удаление базы данных из списка**

Чтобы удалить базу данных из списка, выберите:

* щелкните правой кнопкой мыши на нем и выбрать "удалить базу данных",
* или нажмите "удалить" с клавиатуры.

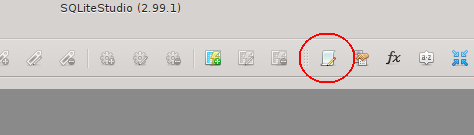
Вас попросят подтвердить, прежде чем база данных будет фактически удалена из списка.

**Удаление базы данных из списка не удаляет файл базы данных! Он удаляет его только из списка баз данных в SQLiteStudio.** Нет никакого способа, чтобы удалить сам файл базы данных с помощью SQLiteStudio. Это по соображениям безопасности-если вам действительно нужно удалить файл базы данных, сделать это с помощью operatins system utils (файловый менеджер, командная строка, ...).

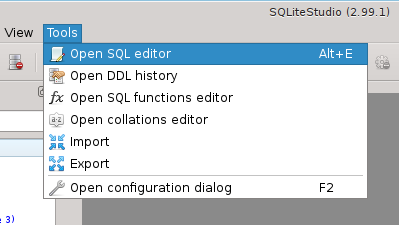
**Выполнение SQL-запросов**

**Окно редактора SQL**

Для выполнения SQL запросов к базе данных SQLite необходимо открыть окно редактора SQL:

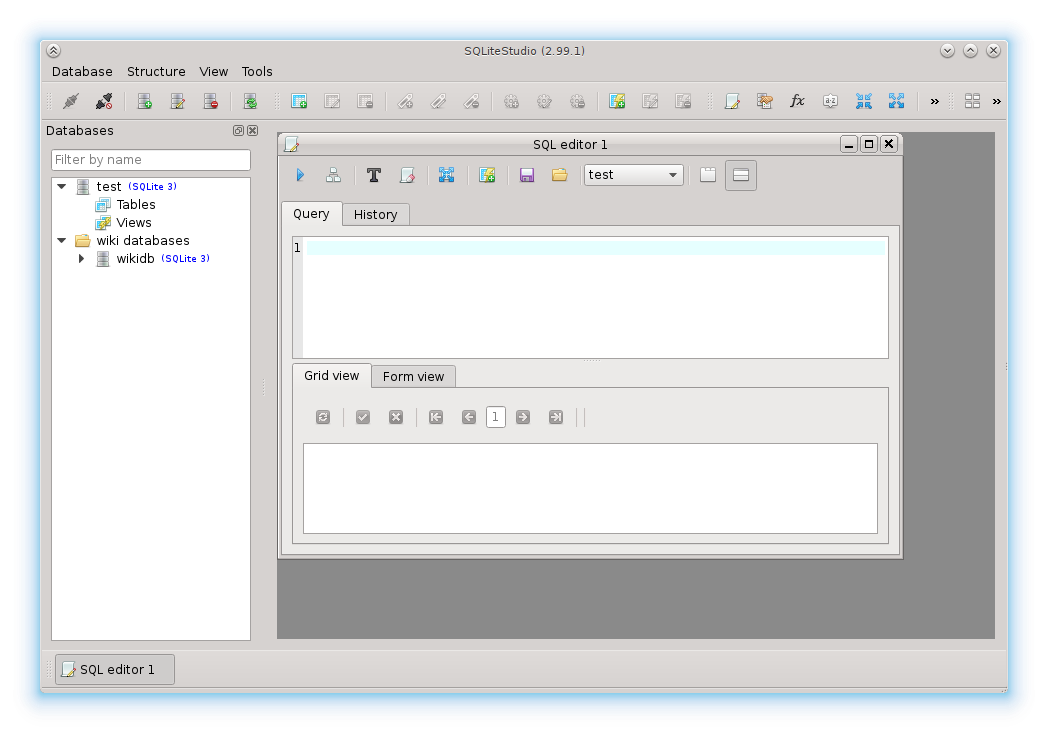


Вы можете найти его также в меню "Инструменты":

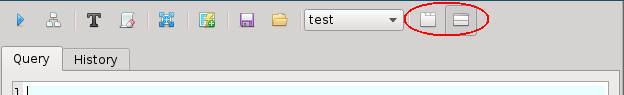


Он также доступен под сочетанием клавиш "Alt+e" (по умолчанию может быть изменен).

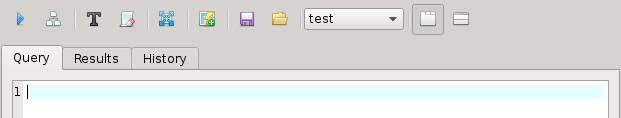
Редактор SQL имеет верхнее текстовое редактирование для ввода запросов SQL и нижнее представление для отображения результатов запроса:



Если вы предпочитаете иметь поле редактирования запроса на одной странице и результаты на другой странице, вы можете переключить его очень быстро с панели инструментов:



И в результате вы получаете 3 вкладки в редакторе, а не 2:



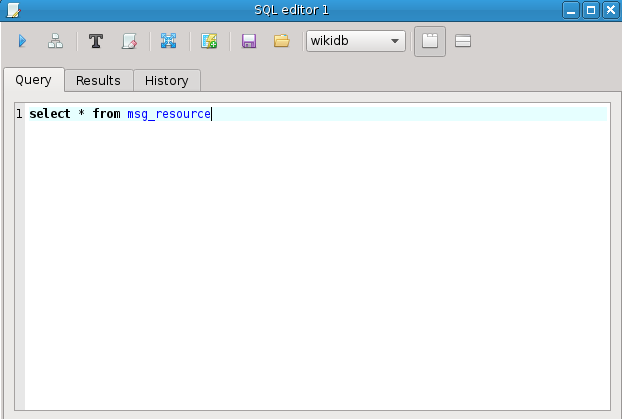
**Кнопки на панели инструментов**

* 1-й кнопкой (https://github.com/pawelsalawa/sqlitestudio/wiki/images/Exec_query.png) выполняет запрос вы набрали в запросе текстовое поле ниже.
* 2 кнопки (https://github.com/pawelsalawa/sqlitestudio/wiki/images/Explain_query.png) выполняет **EXPLAIN** заявление для запроса ниже.
* 3-й кнопкой (https://github.com/pawelsalawa/sqlitestudio/wiki/images/Format_sql.png) использует в настоящее время настроен [SqlFormatterPlugin](https://github.com/pawelsalawa/sqlitestudio/wiki/SqlFormatterPlugin) в формат запросов набирается ниже.
* 4-й кнопкой (https://github.com/pawelsalawa/sqlitestudio/wiki/images/Clear_history.png) очищает выполнения запроса истории (доступен в последней вкладке SQL в окне редактора).
* 5-я кнопка (https://github.com/pawelsalawa/sqlitestudio/wiki/images/Table_export.png) открывает [Export\_dialog](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual" \l "exporting-schema-and-data) для экспорта результатов запроса введенный ниже.
* 6-й кнопкой (https://github.com/pawelsalawa/sqlitestudio/wiki/images/Createviewfromquery.png) создает вид из **SELECT** запросов набирается ниже.
* 7-я кнопка (https://github.com/pawelsalawa/sqlitestudio/wiki/images/Save_sql_file.png сохранить содержимое запроса текстовое поле ниже в файл.
* 8-й кнопкой (https://github.com/pawelsalawa/sqlitestudio/wiki/images/Open_sql_file.png) загружает содержимое выбранного файла в запрос редактировать поля ниже.
* 9-я позиция-это поле со списком, в котором можно выбрать текущую рабочую базу данных для окна редактора SQL. Все запросы выполняются для базы данных, выбранной в этом combobox.
* 10 и 11 кнопок (https://github.com/pawelsalawa/sqlitestudio/wiki/images/Results_in_tab.png https://github.com/pawelsalawa/sqlitestudio/wiki/images/Results_below.png)- это ярлык для настройки SQL редактор результаты режима презентации - в отдельную вкладку, или ниже полей запроса.

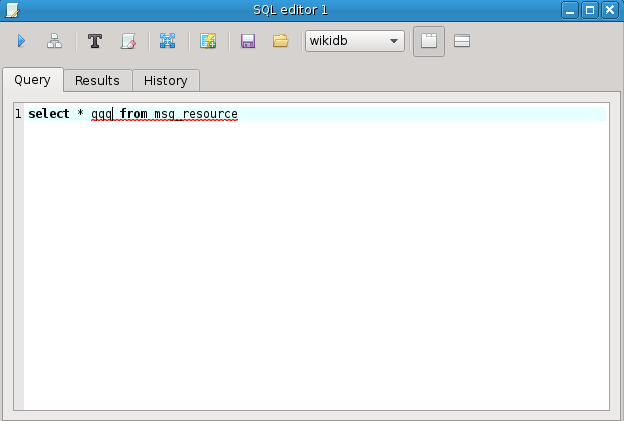
Список может отличаться, если любой плагин добавляет свои собственные кнопки на панель инструментов.

**Ввод SQL-запросов**

При вводе SQL-запроса в редакторе он выделит синтаксис SQL, но также если Вы введете правильное имя любого объекта (то есть существующую таблицу, индекс, триггер, представление, базу данных или имя столбца), имя будет выделено синим цветом (по умолчанию, его можно настроить):



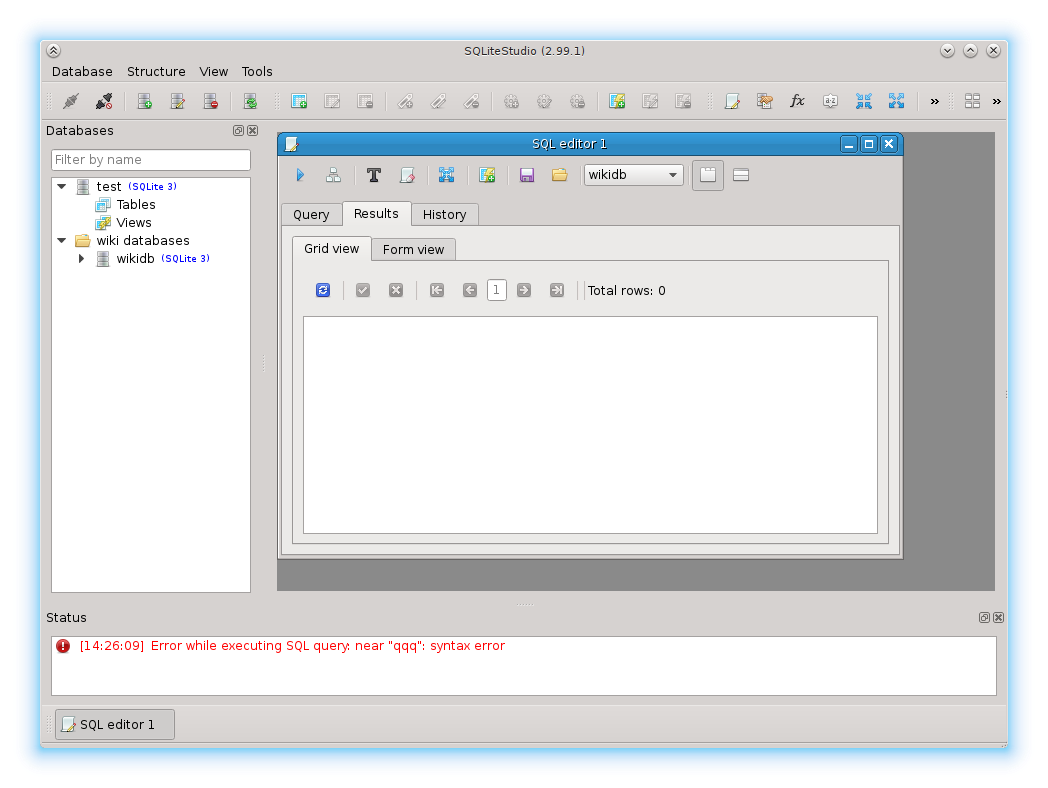
Если вы используете неверный синтаксис SQL, проблемная часть будет подчеркнута волнистой линией:



Для выполнения запроса, используйте кнопку на панели инструментов https://github.com/pawelsalawa/sqlitestudio/wiki/images/Exec_query.png, или сочетание клавиш "Ф9" (ярлык настраивается).

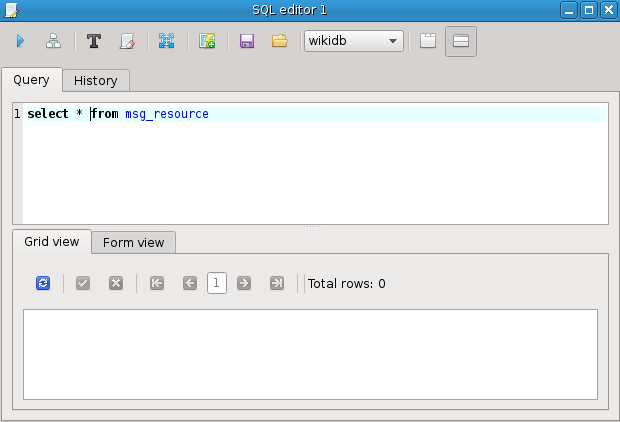
**Обработка ошибок осуществляется с помощью статического синтаксического анализа, поэтому есть шанс, что некоторые ошибки обнаруживаются по ошибке. Если вы думаете, что отмеченная ошибка не является ошибкой, вы можете выполнить запрос и SQLiteStudio не будет анализировать его. Вместо этого он будет помещать запрос непосредственно в SQLite драйвер для его выполнения. Если это было действительно правильно, то он будет возвращать данные, но в таких случаях SQLiteStudio не может извлечь метаданные из запроса и результаты не будут редактируемыми. Только запросы, которые не были отмечены маркером ошибки, вернут полные метаданные, и результаты будут доступны для редактирования. Это ограничение также влияет на прикрепление прозрачной базы данных.**

Даже sqlitestudio утверждает, что синтаксис является неправильным, вы все еще можете попробовать выполнить запрос:

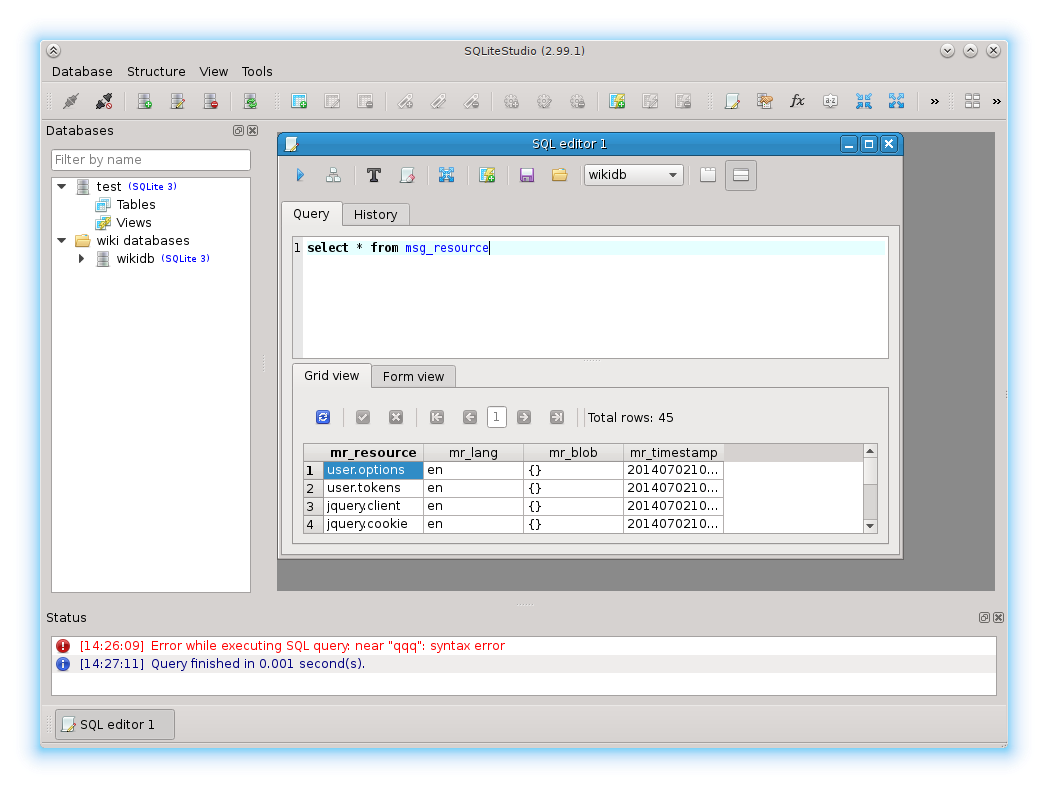


В этом случае запрос был недопустимым, очевидно.

Теперь вернемся к исправлению запроса (удалите из него "qqq"), а также переключим режим представления результатов, чтобы результаты отображались под запросом, а не в отдельной вкладке:

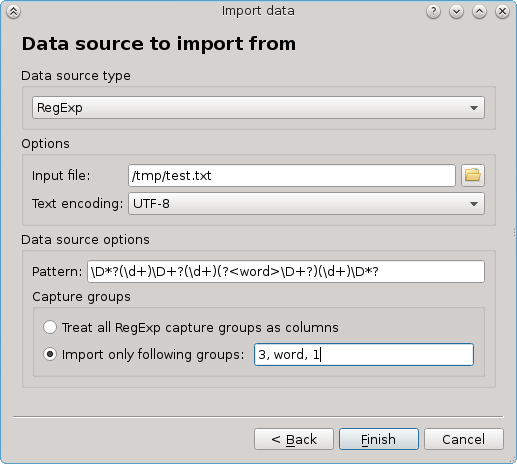


Выполнение запроса:

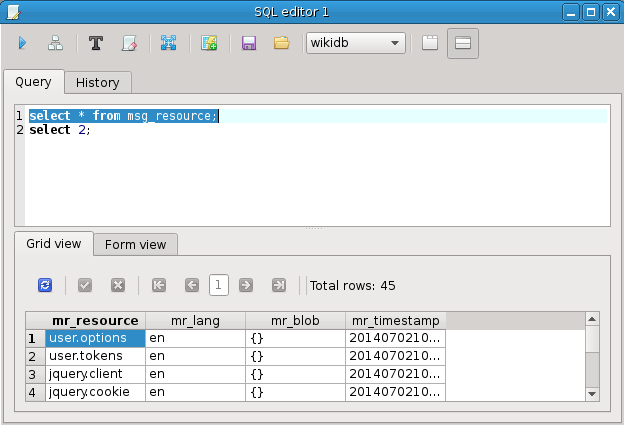


Результаты появились в представлении под запросом, а соответствующее сообщение о состоянии выполнения появилось в нижней области состояния.

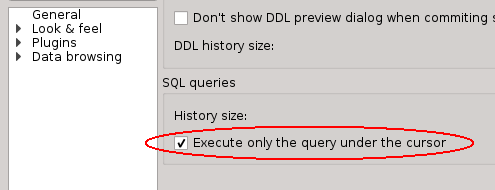
Можно ввести несколько запросов, один за другим, разделяя их точкой с запятой. В этом случае SQLiteStudio будет выполнять все типы запросов в редакторе, но будет представлять только результаты последнего запроса:



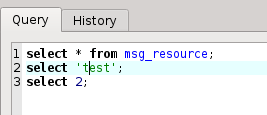
Если вы хотите выполнить только один запрос (набрано еще больше запросов), выберите этот запрос и нажмите " выполнить":



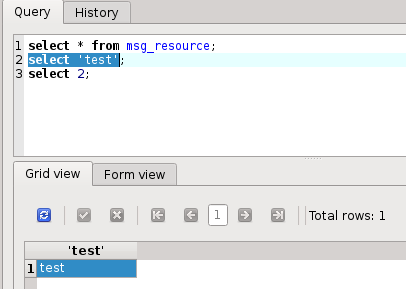
Поведение выполнения всех введенных запросов по умолчанию, но вы можете изменить его, поэтому выполняется только запрос, в котором находится курсор ввода. Для этого откройте [диалог настройки](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#configuration-dialog) (кнопка \_on\_main\_toolbar), зайдите в "общем" разделе и включите эту опцию:



Теперь, когда Вы нажмете "Execute" в редакторе SQL, будет выполняться только один запрос, тот под курсором ввода:

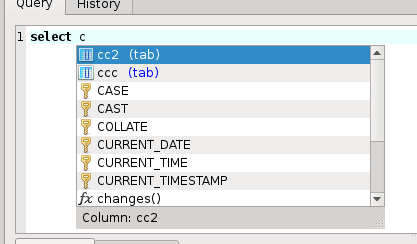


В этом режиме, после выполнения текущего запроса, будет выбран весь запрос, так что вы можете быть уверены, какой именно запрос был выполнен:

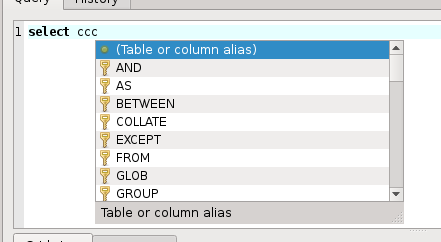


**Завершение кода**

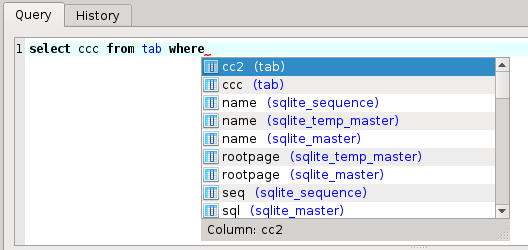
SQLiteStudio предоставляет помощник по коду, поэтому он может помочь Вам с синтаксисом SQL. Когда вы нажмете *Ctrl+пробел* (это настраиваемый ярлык), он покажет вам все возможные значения для текущей позиции курсора. Допустим, вы ввели "SELECT c" и нажмите *сочетание клавиш Ctrl+пробел*:



Вы можете видеть, что он показывает только допустимые записи, которые начинаются с буквы "c". Давайте выберем "ccc", затем попросите помощника насчет следующего предложения:

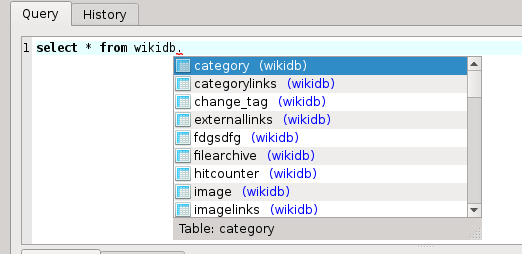


Первая позиция (со значком пули) является псевдо-значением, которое говорит вам, что вы можете поместить псевдоним для таблицы или столбца здесь. Вы ввели " ccc " раньше, что является именем столбца, поэтому здесь вы можете ввести псевдоним для него. Мы не будем этого делать. Вместо этого мы выбираем "FROM" и добавить "tab where" и попросите помощника снова:



Помощник предлагает несколько столбцов, которые можно использовать в предложении "where", но обратите внимание, что столбцы из таблицы "tab" перечислены в начале, поскольку вы уже используете эту таблицу в запросе. Это то, как помощник пытается быть умным с предложением значений - когда они, скорее всего, будут использоваться в контексте, он будет перечислять их в верхней части.

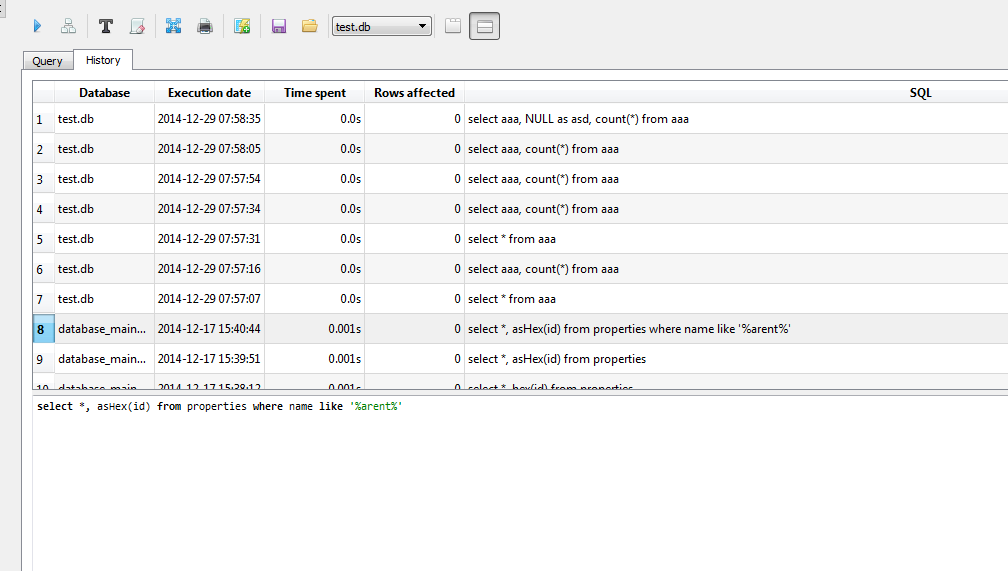
Помощник запускается вручную с помощью клавиш, указанных выше, но он также запускает автоматически, когда вы введете "name.", где имя-это имя существующей базы данных или таблицы. В следующем примере я не использовал сочетания клавиш, это появилось после того, как я типа "wikidb.":



**Есть небольшая проблема с помощником на Linux с включенной композицией KWin, которая заставляет помощника отображаться с небольшой задержкой (около половины секунды). Первопричина этой проблемы пока не установлена. Если это вас беспокоит, отключение композиции в KWin исправляет эту проблему.**

**История выполнения запроса**

Вкладка история SQL хранит историю SQL-запросов, которые вы когда-либо выполняли. Количество запросов хранится в истории можно изменить в [диалоговом окне Настройки](https://github.com/pawelsalawa/sqlitestudio/wiki/Configuration_dialog) и по умолчанию 1000 записей. При превышении этого ограничения самая старая запись удаляется.



Вы можете просматривать и просматривать запросы из истории. Дважды щелкните запись, чтобы автоматически скопировать ее в редактор SQL, так что вы можете повторно выполнить его.

Вы можете стереть всю историю запросов SQL, нажав на кнопку" стереть историю " на панели инструментов и подтвердив свой выбор.

**Встроенные функции SQL**

Sqlitestudio определяет несколько функций SQL в базах данных, к которым он подключается, поэтому их можно использовать при выполнении запросов из sqlitestudio.

**Функции, зарегистрированные SQLiteStudio, доступны только в SQLiteStudio. Они не будут доступны в других приложениях, подключающихся к базе данных, если эти приложения не определяют такие функции самостоятельно**

| **Функция** | **Описание** |
| --- | --- |
| regexp(pattern, arg) | Матчи *арг* против *шаблону*, где *шаблон* представляет собой [регулярное выражение](http://en.wikipedia.org/wiki/Regular_expression). Это также делает **REGEXP** оператора в SQL-запросы. Возвращает 1 на матч, или 0, если *параметр* не соответствует. |
| sqlfile(file) | Считывает данный *файл* как текстовый файл, содержащий SQL-запросы и выполняет все эти запросы. Возвращает первый столбец первой строки из результатов последнего выполненного запроса из *файла*. |
| readfile(file) | Считывает данный *файл* как двоичный файл и возвращает байт из него. |
| writefile(file, data) | Пишет с учетом *данных* байт, в данном *файле*. Возвращает количество фактически записанных байт. |
| langs() | Возвращает список скриптовых языков, существующих в SQLiteStudio на данный момент (это зависит от того, что [скриптовый язык Плагины](https://github.com/pawelsalawa/sqlitestudio/wiki/Official_plugins#Scripting_languages_plugins) загружены в данный момент). |
| script(language, code) | Оценивает приведенный *код* , используя [скриптовый плагин](https://github.com/pawelsalawa/sqlitestudio/wiki/Official_plugins#scripting-languages-plugins) , который поддерживает данный *язык* и возвращает результат оценки. Использовать **Лэнг()** функция, чтобы узнать поддерживаемых языков. |
| charsets() | Возвращает список кодировок, поддерживаемых SQLiteStudio (используется например в аргументах для import() функции) |
| import\_formats() | Возвращает список форматов импорта, поддерживаемых SQLiteStudio (зависит от загружаемых модулей импорта) |
| import\_options(format) | Возвращает список используемых в настоящее время импорт настроек для определенного **формата** (на **формат** должен быть один из форматов вернулся из import\_formats()). Каждая установка в отдельной строке. Каждая линия a setting\_name=setting\_value |
| import(file, format, table, charset, options) | Выполняет импорт процесс с использованием **файла** для ввода, **формат** по выбору импортировать плагин (должен быть одним из значений, возвращаемых import\_formats()). Импорт делается в **таблице**. Если Таблица не существует, она будет создана. Этот **набор символов** не является обязательным и должен быть одним из значений, возвращаемых из charsets() (например 'UTF-8'). По умолчанию используется UTF-8. В **опциях** не является обязательным и должен быть в том же формате, как вернулся со import\_options() (что является одним из вариантов в каждой строке, каждой линии option\_name=value), хотя это нормально, чтобы обеспечить только часть вариантов - то остальные настройки останутся. |
| html\_escape(string) | Заменяет HTML-код специальных символов в *строку* с их HTML-коды побег и возвращает модифицированную строку. Например ' < 'будет заменен на'<'. |
| url\_encode(string) | Заменяет [URL-адрес](http://en.wikipedia.org/wiki/Url) конкретные персонажи с их процентов защитить коды и возвращает модифицированную строку. Например ' / 'будет заменен на '%2F'. |
| url\_decode(string) | Заменяет процентов защитить кодов [URL-адреса](http://en.wikipedia.org/wiki/Url) с их фактическим символов и возвращает модифицированную строку. Например, '%2F 'будет заменен на'/'. |
| base64\_encode(data) | Кодирует данные байты с [помощью base64](http://en.wikipedia.org/wiki/Base64) кодирование и возвращает строки base64. |
| base64\_decode(data) | Декодирует данные байты из [в base64](http://en.wikipedia.org/wiki/Base64) кодировку и возвращает декодированный байт. |
| md4\_bin(data) | Вычисляет [алгоритмов MD4](http://en.wikipedia.org/wiki/Md4) хэш-функции для данных и возвращает вычисленное значение в байтах. |
| md4(data) | Вычисляет [алгоритмов MD4](http://en.wikipedia.org/wiki/Md4) хэш-функции для данных и возвращает вычисленное значение в виде шестнадцатеричной строки. |
| md5\_bin(data) | Вычисляет [MD5 хэш](http://en.wikipedia.org/wiki/Md5) хэш-функции для данных и возвращает вычисленное значение в байтах. |
| md5(data) | Вычисляет [MD5 хэш](http://en.wikipedia.org/wiki/Md5) хэш-функции для данных и возвращает вычисленное значение в виде шестнадцатеричной строки. |
| sha1(data) | Вычисляет [алгоритм SHA-1](http://en.wikipedia.org/wiki/Sha1) хэш-функции для данных и возвращает вычисленное значение в байтах. |
| sha224(data) | Вычисляет [алгоритм SHA-224](http://en.wikipedia.org/wiki/SHA-2) хэш-функции для данных, и возвращает вычисленное значение в байтах. |
| sha256(data) | Вычисляет [алгоритм SHA-256](http://en.wikipedia.org/wiki/SHA-2) хэш-функции для данных, и возвращает вычисленное значение в байтах. |
| sha384(data) | Вычисляет [алгоритм SHA-384](http://en.wikipedia.org/wiki/SHA-2) хэш-функции для данных, и возвращает вычисленное значение в байтах. |
| sha512(data) | Вычисляет [алгоритм SHA-512](http://en.wikipedia.org/wiki/SHA-2) хэш-функции для данных, и возвращает вычисленное значение в байтах. |
| sha3\_224(data) | Вычисляет [SHA3-224](http://en.wikipedia.org/wiki/SHA-3) хэш-функции для данных, и возвращает вычисленное значение в байтах. |
| sha3\_256(data) | Вычисляет [SHA3-256](http://en.wikipedia.org/wiki/SHA-3) хэш-функции для данных, и возвращает вычисленное значение в байтах. |
| sha3\_384(data) | Вычисляет [SHA3-384](http://en.wikipedia.org/wiki/SHA-3) хэш-функции для данных, и возвращает вычисленное значение в байтах. |
| sha3\_512(data) | Вычисляет [SHA3-512](http://en.wikipedia.org/wiki/SHA-3) хэш-функции для данных, и возвращает вычисленное значение в байтах. |

**Редактирование данных**

В **табличном виде** данных, вы можете начать редактирование значений:

* двойной щелчок на ячейке (старое значение останется и будет изначально выбран),
* выбор ячейки и нажатие клавиши Enter (старое значение останется и будет изначально выбрано),
* выделите ячейку и просто начните вводить новое значение (старое значение будет перезаписано немедленно),
* щелчок правой кнопкой мыши по ячейке и выбор "редактировать значение в редакторе" из контекстного меню (значение будет изменено в отдельном диалоговом окне).

Любые изменения, внесенные в данные, не передаются в базу данных немедленно. Вы должны зафиксировать их вручную, нажав на кнопку "commit" на панели инструментов, или нажать "Ctrl+ввод" клавиш (клавиш можно изменить в [диалоговом окне Настройки](https://github.com/pawelsalawa/sqlitestudio/wiki/Configuration_dialog).

Если вы решили удалить какие-либо изменения в данных Вы сделали, нажмите на кнопку "Откат" на панели инструментов, или нажмите "Ctrl+Backspace" сочетание клавиш (это также настраивается).

Можно выборочно фиксировать/откатывать ячейки (для фиксации / отката только некоторых текущих изменений). Для этого выделите ячейки, которые вы хотите зафиксировать / откатить, и щелкните их правой кнопкой мыши. В контекстном меню вы увидите позиции "фиксация выделенных ячеек" и "Откат выделенных ячеек".

В **форме просмотра** данных, вы можете редактировать значения по одной строке за раз. Каждая ячейка из строки представлена здесь как вид/редактор с вкладками. Нажимая "Commit" или "Rollback" в представлении формы, Вы фиксируете/откатываете только текущую строку.

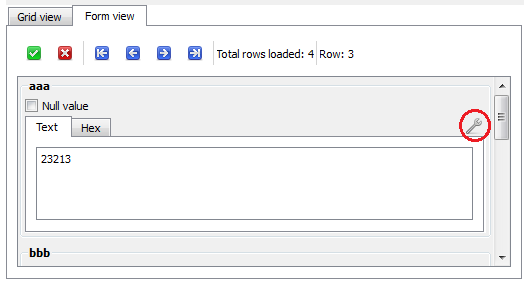
Представление формы работает только со строками, которые в данный момент видны в представлении сетки, поэтому, когда вы достигаете последней строки из представления сетки и хотите редактировать следующие строки, вам нужно переключиться обратно на представление сетки, перейти на следующую страницу данных, а затем перейти к представлению формы и продолжить работу.

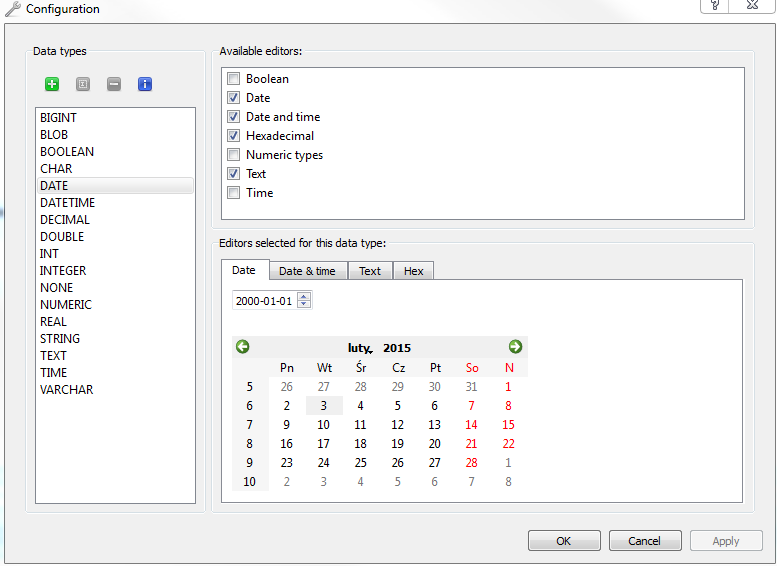
Обратите внимание, что перед изменением страниц в представлении сетки необходимо зафиксировать / откатить текущие изменения данных, в противном случае они будут потеряны.

**Диалоговое окно редактора значений**

Если столбец редактировать это "BLOB" типа, отдельном редакторе откроется диалог. Редактор диалог имеет несколько вкладок, где вы можете просматривать и редактировать данные, используя различные режимы (текстовый, двоичный и т.д.). То же самое диалоговое окно откроется, если вы щелкните правой кнопкой мыши на ячейке и выберите "Правка в редакторе значений".

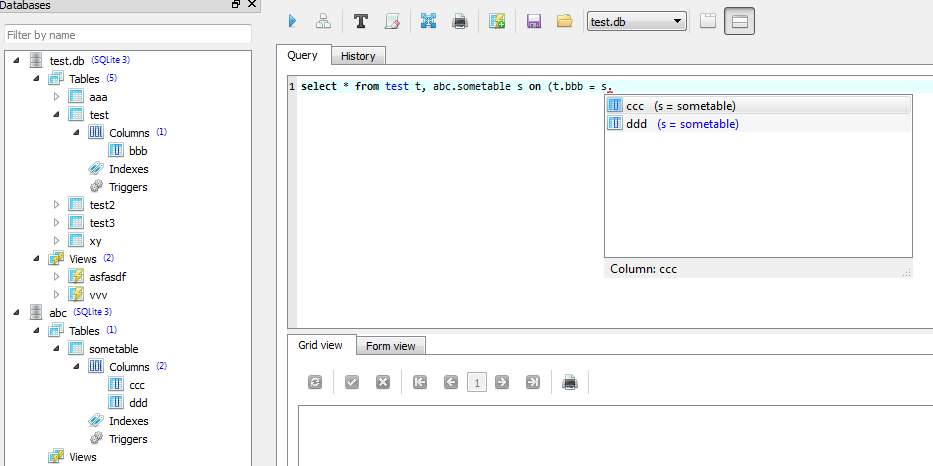
Одни и те же данные Редакторы/просмотров используются на **формы**, поэтому, если вы планируете редактировать все колонки с помощью редактора чисел, вы можете просто как хорошо перейти к форме просмотра.

Редактор значений может быть настроен, поэтому он использует определенные режимы редактора/представления для определенного типа данных столбца. Для этого используйте значок в правой части представления редактора: 

В левой части конфигурации вы видите список типов данных, распознанных редактором значений, где вы можете выбрать тип данных, который вы хотите настроить. В правом верхнем углу вы можете увидеть список всех доступных виджетов редактора (они предоставляются плагинами, поэтому список зависит от того, какие плагины вы в настоящее время загрузили, но многие плагины уже встроены в приложение). В правой нижней области у вас есть пример того, как ваша текущая конфигурация для выбранного типа данных выглядит. Вы также можете перетаскивать вкладки редактора, чтобы изменить их порядок. Самый левый редактор является редактором по умолчанию для этого типа данных (что означает, что левая вкладка всегда открыта сначала для этого типа данных). 

Если существует тип данных, не указанный в списке конфигурации, и вы входите в этот тип где-то в базе данных, SQLiteStudio выберет редакторы по умолчанию (Text и Hex).

**Прозрачное прикрепление базы данных**

Прозрачный прикрепление функция в SQLiteStudio является возможность вызова ATTACH и DETACH автоматически с помощью SQLiteStudio, когда это необходимо. SQL-запрос можно ввести в редакторе SQL, используя таблицы из нескольких баз данных, используя имена баз данных, представленные в списке баз данных: 

Поскольку прозрачные крепления звонки ATTACH и DETACH внутренне, она влияла на все ограничения, которые относятся к этим заявлениям. Это не поддерживает подключение баз данных различного типа. Это означает, что нельзя смешивать базы данных SQLite 2 и SQLite 3 в одном запросе или любом другом типе базы данных mix ups.

**Создание и редактирование таблиц**

TODO: помните о сортировке нескольких столбцов TODO: объясните разницу между ограничениями столбцов и ограничениями таблицы

**Представление (View)**

Представление (View) хранится только в базе данных в SQLite заявлении соответствующими именами. Вид (View) на самом деле является предопределенным форма SQLite запрос комбинации таблицы.

Все строки в окне просмотра (View) может содержать таблицы или выбранные строки из одной или нескольких таблиц. Вид (View) могут быть созданы из одной или нескольких таблиц, в зависимости от точки зрения вы хотите создать запрос SQLite. ,

Представление (View) представляет собой виртуальную таблицу, которая позволяет пользователям достичь следующего:

* Поиск пользователя или структура данных группы способ более естественным или интуитивно.
* Ограничение доступа к данным, пользователи видят только ограниченные данные, вместо полной таблицы.
* Сводные данные из различных таблиц, используемых для создания отчетов.

SQLite вид только для чтения, и, следовательно, не может быть выполнено удаление, вставка или UPDATE заявление о представлении. Но вы можете создать триггер на представлении, когда вы пытаетесь удалить, или UPDATE INSERT сработавшей, должно быть сделано в содержании определения действия триггера.

## Создание представления

SQLite вид создается с помощью**CREATE VIEW заявление.**SQLite представления могут быть созданы из одной таблицы, несколько таблиц или других представлений.

CREATE VIEW Базовый синтаксис выглядит следующим образом:

CREATE [TEMP | ВРЕМЕННОЕ] VIEW имя\_представления AS

ВЫБОР column1, column2 .....

FROM table\_name

WHERE [состояние];

Вы можете включать в себя несколько таблиц в SELECT, заявление, которое в обычном SQL SELECT запрос очень похожим образом. Если вы используете дополнительный TEMP или TEMPORARY ключевое слово, это создаст представление во временной базе данных.

## примеров

Предположим, что таблица Компания имеет следующие записи:

ID ИМЯ ВОЗРАСТ АДРЕС ЗАРПЛАТА

---------- ---------- ---------- ---------- ----------

1 Paul 32 California 20000.0

2 Allen 25 Texas 15000,0

3 Teddy 23 Норвегия 20000,0

4 Mark 25 Рич-Mond 65000,0

5 David 27 Texas 85000,0

6 Ким 22 Юго-зал 45000,0

7 Джеймс 24 Хьюстон 10000,0

Теперь, вот представление из таблицы компании, чтобы создать экземпляр. Вид выбрать лишь несколько столбцов из таблицы КОМПАНИИ:

SQLite> CREATE VIEW COMPANY\_VIEW AS

ВЫБОР ID, имя, возраст

ОТ КОМПАНИИ;

Теперь вы можете запросить COMPANY\_VIEW, и запрос фактического столов аналогичным образом. Ниже приведены примеры:

SQLite> SELECT \* FROM COMPANY\_VIEW;

Это дает следующие результаты:

ID NAME AGE

---------- ---------- ----------

1 Павел 32

2 Allen 25

3 Teddy 23

4 Mark 25

5 David 27

6 Ким 22

7 Джеймс 24

## Удалить представление

Чтобы удалить представление, просто используйте DROP VIEW заявление с **имя\_представления.**DROP VIEW Базовый синтаксис выглядит следующим образом:

SQLite> DROP VIEW имя\_представления;

Следующая команда удалит вид COMPANY\_VIEW мы создали ранее:

SQLite> DROP VIEW COMPANY\_VIEW;

**Опция командной строки**

Ниже приведен список параметров, которые можно передавать в качестве аргументов при запуске SQLiteStudio. Эти параметры действительны только для GUI приложений, для [командной строки](https://github.com/pawelsalawa/sqlitestudio/wiki/CLI) есть отдельный набор настроек (нажать [командной строки](https://github.com/pawelsalawa/sqlitestudio/wiki/CLI) , чтобы увидеть подробности).

| **Параметры SQLiteStudio** |
| --- |
| - х, -- help |
| - v, -- version |
| - d, -- debug |
| -- debug-stdout |
| -- debug-lemon |
| -- debug-sql |
| -- debug-sql-db |
| -- list-плагины |
| -- мастер-конфиг |

| **Аргументы sqlitestudio** |
| --- |
| folder |

**Файл конфигурации SQLiteStudio**

Вся конфигурация sqlitestudio хранится в одном файле базы данных SQLite3. Более подробную информацию по [конфигурации расположения каталога](https://github.com/pawelsalawa/sqlitestudio/wiki/Configuration_directory_location) страницы.

**Обновления SQLiteStudio**

**Автомат**

Автоматическое обновление:

* проверка доступности обновлений,
* загрузка обновлений,
* установка обновлений.

**Автоматическое обновление работает только для Переносимого дистрибутива. Если приложение было установлено из диспетчера пакетов Linux, то этот менеджер пакетов должен обновить приложение.**

Проверка доступности обновлений всегда выполняется при запуске приложения, но может быть отключена пользователем (и повторно включена в диалоге настройки). Он также может быть вручную запускается пользователем из меню *Справка* / *проверить наличие обновлений*. Перед загрузкой обновлений у пользователя всегда запрашивается подтверждение.

Для установки обновлений могут потребоваться права администратора. Если да, SQLiteStudio будет запрашивать у пользователя пароль (в Linux и MacOS X) или подтверждение привилегированной работы (в Windows).

**Http прокси пока не поддерживается, поэтому, если для подключения HTTP требуется настройка прокси, автоматическое обновление не будет работать. Поддержка конфигурации прокси-сервера планируется в будущих версиях.**

**Руководство**

Ручное обновление является просто о загрузке свежей версии с [Скачать страницы](http://sqlitestudio.pl/?act=download) и заменив старую версию с ней.

**Расширенное использование**

**Пользовательские функции SQL**

SQLiteStudio позволяет пользователю создавать дополнительные функции SQL, которые будут использоваться в SQL-запросах. Можно создать, например, функция "test123()", которая возвращает случайную строку для вас, и вы сможете набрать:

Вставить в таблицу mytable значения (test123());

Пользовательские функции SQL в SQLiteStudio очень полезны, если ваше приложение использует некоторые пользовательские функции SQL для себя, и вы хотите иметь возможность эмулировать среду приложения в sqlitestudio, поэтому вы можете выполнять похожие запросы к базе данных.

Другой вариант использования-когда вы хотите упростить некоторые задачи обслуживания базы данных. Создание функции, которая выполняет несколько более сложных операций, используя некоторые простые входные значения, может рассматриваться как процедура SQLite, потому что вы можете выполнить ее с помощью:

Выберите имя\_функции(input\_arguments);

Чтобы создать/изменить / удалить функции SQL, откройте редактор пользовательских функций SQL: https://github.com/pawelsalawa/sqlitestudio/wiki/images/Tools_functions.png

В редакторе функций нужно определить несколько вещей:

1. Имя функции. Он должен быть уникальным.
2. Является ли ваша функция goind Скалярной или Агрегатной?
   1. Скалярные функции применяются к каждой отдельной строке данных отдельно. Примером такой функции является "length()".
   2. Агрегатные функции используются совместно с GROUP BY п. и они применяются ко всему результирующему набору. Примером такой функции является "min()".
3. Язык реализации-язык программирования, который вы будете использовать, чтобы определить, что функция на самом деле делать.
4. Input arguments-список именованных входных аргументов. В настоящее время они просто символические имена, используемые помощником по синтаксису SQL, чтобы предложить ожидаемые входные аргументы. Больше ничего.
5. Databases-для каких баз данных функция должна быть зарегистрирована SQLiteStudio.
6. Для Скалярной функции-требуется одно поле реализации. Это код на выбранном языке программирования, который выполняет работу функции. Возвращаемое значение этого кода становится результатом вызова функции.
7. Для Агрегатной функции существует 3 поля реализации:
   1. Код инициализации-этот код выполняется один раз, в начале сгруппированных результатов запроса. Вы можете использовать его для инициализации некоторых переменных до" 0 " значений или что-то в этом роде. Это поле является необязательным.
   2. Per Step code-этот код выполняется для каждой строки данных из результатов. Каждая итерация использует один и тот же контекст, что означает, что все переменные являются общими для всех шагов.
   3. Код заключительного шага-выполняется после последнего кода" per step". Вы можете рассчитать конечный результат и переучивать любые приобретенные вами ресурсы. Этот код должен возвращать результат агрегатной функции.

Аргументы, передаваемые функции, доступны в коде как переменные, имена которых зависят от выбранного языка реализации. Для TCL аргументы доступны в "$argv". Для qtscript они в "arguments" массива. Чтобы узнать, как обрабатывать входные аргументы на определенном языке, прочитайте о конкретном плагине, который обеспечивает поддержку этого языка. Официальные Плагины, описанные на [языках сценариев, Плагины, страницы](https://github.com/pawelsalawa/sqlitestudio/wiki/Official_plugins#scripting-languages-plugins)).

**Пользовательские функции SQL, созданные в SQLiteStudio, существуют только в контексте sqlitestudio. Функции не сохраняются в базе данных, это просто не так, как работают базы данных SQLite. Если вы хотите использовать какую-либо функцию в своем собственном приложении, Вы должны зарегистрировать эту функцию в SQLite из своего приложения.**

**Примеры: Функции пользователя**

**Понятие функции пользователя**

При реализации на языке SQL сложных алгоритмов, которые могут потребоваться более одного раза, сразу встает вопрос о сохранении разработанного кода для дальнейшего применения. Эту задачу можно было бы реализовать с помощью хранимых процедур, однако их архитектура не позволяет использовать процедуры непосредственно в выражениях, т.к. они требуют промежуточного присвоения возвращенного значения переменной, которая затем и указывается в выражении. Естественно, подобный метод применения программного кода не слишком удобен. Многие разработчики уже давно хотели иметь возможность вызова разработанных алгоритмов непосредственно в выражениях.

Возможность создания пользовательских функций была предоставлена в среде MS SQL Server 2000. В других реализациях SQL в распоряжении пользователя имеются только встроенные функции, которые обеспечивают выполнение наиболее распространенных алгоритмов: поиск максимального или минимального значения и др.

Функции пользователя представляют собой самостоятельные объекты базы данных, такие, например, как хранимые процедурыили триггеры. Функция пользователя располагается в определенной базе данных и доступна только в ее контексте.

В SQL Server имеются следующие классы функций пользователя:

* Scalar – функции возвращают обычное скалярное значение, каждая может включать множество команд, объединяемых в один блок с помощью конструкции BEGIN...END;
* Inline – функции содержат всего одну команду SELECT и возвращают пользователю набор данных в виде значения типа данных TABLE;
* Multi-statement – функции также возвращают пользователю значение типа данных TABLE, содержащее набор данных, однако в теле функции находится множество команд SQL (INSERT, UPDATE и т.д.). Именно с их помощью и формируется набор данных, который должен быть возвращен после выполнения функции.

Пользовательские функции сходны с хранимыми процедурами, но, в отличие от них, могут применяться в запросах так же, как и системные встроенные функции. Пользовательские функции, возвращающие таблицы, могут стать альтернативой просмотрам. Просмотры ограничены одним выражением SELECT, а пользовательские функции способны включать дополнительные выражения, что позволяет создавать более сложные и мощные конструкции.

**Функции Scalar**

Создание и изменение функции данного типа выполняется с помощью команды:

<определение\_скаляр\_функции>::=

{CREATE | ALTER } FUNCTION [владелец.]

имя\_функции

( [ { @имя\_параметра скаляр\_тип\_данных

[=default]}[,...n]])

RETURNS скаляр\_тип\_данных

[WITH {ENCRYPTION | SCHEMABINDING}

[,...n] ]

[AS]

BEGIN

<тело\_функции>

RETURN скаляр\_выражение

END

Рассмотрим назначение параметров команды.

Функция может содержать один или несколько входных параметров либо не содержать ни одного. Каждый параметр должен иметь уникальное в пределах создаваемой функции имя и начинаться с символа "@". После имени указывается тип данных параметра. Дополнительно можно указать значение, которое будет автоматически присваиваться параметру (DEFAULT), если пользователь явно не указал значение соответствующего параметра при вызове функции.

С помощью конструкции RETURNS скаляр\_тип\_данных указывается, какой тип данных будет иметь возвращаемое функциейзначение.

Дополнительные параметры, с которыми должна быть создана функция, могут быть указаны посредством ключевого слова WITH. Благодаря ключевому слову ENCRYPTION код команды, используемый для создания функции, будет зашифрован, и никто не сможет просмотреть его. Эта возможность позволяет скрыть логику работы функции. Кроме того, в теле функции может выполняться обращение к различным объектам базы данных, а потому изменение или удаление соответствующих объектов может привести к нарушению работы функции. Чтобы избежать этого, требуется запретить внесение изменений, указав при создании этой функции ключевое слово SCHEMABINDING.

Между ключевыми словами BEGIN...END указывается набор команд, они и будут являться телом функции.

Когда в ходе выполнения кода функции встречается ключевое слово RETURN, выполнение функции завершается и как результат ее вычисления возвращается значение, указанное непосредственно после слова RETURN. Отметим, что в теле функцииразрешается использование множества команд RETURN, которые могут возвращать различные значения. В качестве возвращаемого значения допускаются как обычные константы, так и сложные выражения. Единственное условие – тип данных возвращаемого значения должен совпадать с типом данных, указанным после ключевого слова RETURNS.

**Пример 11.1.** Создать и применить функцию скалярного типа для вычисления суммарного количества товара, поступившего за определенную дату. Владелец функции – пользователь с именем user1.

CREATE FUNCTION

user1.sales(@data DATETIME)

RETURNS INT

AS

BEGIN

DECLARE @c INT

SET @c=(SELECT SUM(количество)

FROM Сделка

WHERE дата=@data)

RETURN (@c)

END

Пример 11.1. Создание функции скалярного типа для вычисления суммарного количества товара, поступившего за определенную дату.

В качестве входного параметра используется дата. Функция возвращает значение целого типа, полученное из оператора SELECTпутем суммирования количества товара из таблицы Сделка. Условием отбора записей для суммирования является равенство даты сделки значению входного параметра функции.

Проиллюстрируем обращение к функции пользователя: определим количество товара, поступившего за 02.11.01:

DECLARE @kol INT

SET @kol=user1.sales ('02.11.01')

SELECT @kol

**Функции Inline**

Создание и изменение функции этого типа выполняется с помощью команды:

<определение\_табл\_функции>::=

{CREATE | ALTER } FUNCTION [владелец.]

имя\_функции

( [ { @имя\_параметра скаляр\_тип\_данных

[=default]}[,...n]])

RETURNS TABLE

[ WITH {ENCRYPTION | SCHEMABINDING}

[,...n] ]

[AS]

RETURN [(] SELECT\_оператор [)]

Основная часть параметров, используемых при создании табличных функций, аналогична параметрам скалярной функции. Тем не менее создание табличных функций имеет свою специфику.

После ключевого слова RETURNS всегда должно указываться ключевое слово TABLE. Таким образом, функция данного типа должна строго возвращать значение типа данных TABLE. Структура возвращаемого значения типа TABLE не указывается явно при описании собственно типа данных. Вместо этого сервер будет автоматически использовать для возвращаемого значения TABLEструктуру, возвращаемую запросом SELECT, который является единственной командой функции.

Особенность функции данного типа заключается в том, что структура значения TABLE создается автоматически в ходе выполнения запроса, а не указывается явно при определении типа после ключевого слова RETURNS.

Возвращаемое функцией значение типа TABLE может быть использовано непосредственно в запросе, т.е. в разделе FROM.

**Пример 11.2.** Создать и применить функцию табличного типа для определения двух наименований товара с наибольшим остатком.

CREATE FUNCTION user1.itog()

RETURNS TABLE

AS

RETURN (SELECT TOP 2 Товар.Название

FROM Товар INNER JOIN Склад

ON Товар.КодТовара=Склад.КодТовара

ORDER BY Склад.Остаток DESC)

Пример 11.2. Создание функции табличного типа для определения двух наименований товара с наибольшим остатком.

Использовать функцию для получения двух наименований товара с наибольшим остатком можно следующим образом:

SELECT Название

FROM user1.itog()

**Функции Multi-statement**

Создание и изменение функций типа Multi-statement выполняется с помощью следующей команды:

<определение\_мульти\_функции>::=

{CREATE | ALTER }FUNCTION [владелец.]

имя\_функции

( [ { @имя\_параметра скаляр\_тип\_данных

[=default]}[,...n]])

RETURNS @имя\_параметра TABLE

<определение\_таблицы>

[WITH {ENCRYPTION | SCHEMABINDING}

[,...n] ]

[AS]

BEGIN

<тело\_функции>

RETURN

END

Использование большей части параметров рассматривалось при описании предыдущих функций.

Отметим, что функции данного типа, как и табличные, возвращают значение типа TABLE. Однако, в отличие от табличных функций, при создании функций Multi-statement необходимо явно задать структуру возвращаемого значения. Она указывается непосредственно после ключевого слова TABLE и, таким образом, является частью определения возвращаемого типа данных. Синтаксис конструкции <определение\_таблицы> полностью соответствует одноименным структурам, используемым при создании обычных таблиц с помощью команды CREATE TABLE.

Набор возвращаемых данных должен формироваться с помощью команд INSERT, выполняемых в теле функции. Кроме того, в теле функции допускается использование различных конструкций языка SQL, которые могут контролировать значения, размещаемые в выходном наборе строк. При работе с командой INSERT требуется явно указать имя того объекта, куда необходимо вставить строки. Поэтому в функциях типа Multi-statement, в отличие от табличных, необходимо присвоить какое-то имя объекту с типом данных TABLE – оно и указывается как возвращаемое значение.

Завершение работы функции происходит в двух случаях: если возникают ошибки выполнения и если появляется ключевое слово RETURN. В отличие от функций скалярного типа, при использовании команды RETURN не нужно указывать возвращаемое значение. Сервер автоматически возвратит набор данных типа TABLE, имя и структура которого была указана после ключевого слова RETURNS. В теле функции может быть указано более одной команды RETURN.

Необходимо отметить, что работа функции завершается только при наличии команды RETURN. Это утверждение верно и в том случае, когда речь идет о достижении конца тела функции – самой последней командой должна быть команда RETURN.

**Пример 11.3.** Создать и применить функцию (типа multi-statement), которая для некоторого сотрудника выводит список всех его подчиненных (подчиненных как непосредственно ему, так и опосредствованно через других сотрудников).

Список сотрудников с указанием каждого руководителя представлен в таблице emp\_mgr со следующей структурой:

CREATE TABLE emp\_mgr

(emp CHAR(2) PRIMARY KEY,-- сотрудник

mgr CHAR(2)) -- руководитель

Пример данных в таблице emp\_mgr показан ниже. Для упрощения иллюстрации имена сотрудников и их начальников представлены буквами латинского алфавита. У директора организации начальника нет (NULL).

emp mgr

---------

a NULL

b a

c a

d a

e f

f b

g b

i c

k d

CREATE FUNCTION fn\_findReports(@id\_emp

CHAR(2))

RETURNS @report TABLE(empid CHAR(2)

PRIMARY KEY,

mgrid CHAR(2))

AS

BEGIN

DECLARE @r INT

DECLARE @t TABLE(empid CHAR(2)

PRIMARY KEY,

mgrid CHAR(2),

pr INT DEFAULT 0)

INSERT @t SELECT emp,mgr,0

FROM emp\_mgr

WHERE emp=@id\_emp

SET @r=@@ROWCOUNT

WHILE @r>0

BEGIN

UPDATE @t SET pr=1 WHERE pr=0

INSERT @t SELECT e.emp, e.mgr,0

FROM emp\_mgr e, @t t

WHERE e.mgr=t.empid

AND t.pr=1

SET @r=@@ROWCOUNT

UPDATE @t SET pr=2 WHERE pr=1

END

INSERT @report SELECT empid, mgrid

FROM @t

RETURN

END

Пример 11.3. Создание функции, которая для некоторого сотрудника выводит список всех его подчиненных.

Применим созданную функцию для определения списка подчиненных сотрудника ‘b’:

SELECT \* FROM fn\_findReports('b')

Оператор возвращает следующие значения:

emp mgr

-----------

b a

e f

f b

g b

Список подчиненных сотрудника ‘a’ создается с помощью оператора

SELECT \* FROM fn\_findReports('a')

emp mgr

---------

a NULL

b a

c a

d a

e f

f b

g b

i c

k d

Другой оператор формирует список подчиненных сотрудника ‘e’:

SELECT \* FROM fn\_findReports('e')

emp mgr

--------

e f

Список подчиненных сотрудника ‘c’ создает следующий оператор:

SELECT \* FROM fn\_findReports('c')

emp mgr

--------

c a

i c

Удаление любой функции осуществляется командой:

DROP FUNCTION {[ владелец.] имя\_функции }

[,...n]

**Встроенные функции**

Встроенные функции, имеющиеся в распоряжении пользователей при работе с SQL, можно условно разделить на следующие группы:

* математические функции;
* строковые функции;
* функции для работы с датой и временем;
* функции конфигурирования;
* функции системы безопасности;
* функции управления метаданными;
* статистические функции.

**Математические функции**

Краткий обзор математических функций представлен в таблице.

|  |  |
| --- | --- |
| Таблица 11.1. | |
| ABS | вычисляет абсолютное значение числа |
| ACOS | вычисляет арккосинус |
| ASIN | вычисляет арксинус |
| ATAN | вычисляет арктангенс |
| ATN2 | вычисляет арктангенс с учетом квадратов |
| CEILING | выполняет округление вверх |
| COS | вычисляет косинус угла |
| COT | возвращает котангенс угла |
| DEGREES | преобразует значение угла из радиан в градусы |
| EXP | возвращает экспоненту |
| FLOOR | выполняет округление вниз |
| LOG | вычисляет натуральный логарифм |
| LOG10 | вычисляет десятичный логарифм |
| PI | возвращает значение "пи" |
| POWER | возводит число в степень |
| RADIANS | преобразует значение угла из градуса в радианы |
| RAND | возвращат случайное число |
| ROUND | выполняет округление с заданной точностью |
| SIGN | определяет знак числа |
| SIN | вычисляет синус угла |
| SQUARE | выполняет возведение числа в квадрат |
| SQRT | извлекает квадратный корень |
| TAN | возвращает тангенс угла |

SELECT Товар.Название, Сделка.Количество,

Round(Товар.Цена\*Сделка.Количество

\*0.05,1)

AS Налог

FROM Товар INNER JOIN Сделка

ON Товар.КодТовара=

Сделка.КодТовара

Пример 11.4. Использование функции округления до одного знака после запятой для расчета налога.

**Строковые функции**

Краткий обзор строковых функций представлен в таблице.

|  |  |
| --- | --- |
| Таблица 11.2. | |
| ASCII | возвращает код ASCII левого символа строки |
| CHAR | по коду ASCII возвращает символ |
| CHARINDEX | определяет порядковый номер символа, с которого начинается вхождение подстроки в строку |
| DIFFERENCE | возвращает показатель совпадения строк |
| LEFT | возвращает указанное число символов с начала строки |
| LEN | возвращает длину строки |
| LOWER | переводит все символы строки в нижний регистр |
| LTRIM | удаляет пробелы в начале строки |
| NCHAR | возвращает по коду символ Unicode |
| PATINDEX | выполняет поиск подстроки в строке по указанному шаблону |
| REPLACE | заменяет вхождения подстроки на указанное значение |
| QUOTENAME | конвертирует строку в формат Unicode |
| REPLICATE | выполняет тиражирование строки определенное число раз |
| REVERSE | возвращает строку, символы которой записаны в обратном порядке |
| RIGHT | возвращает указанное число символов с конца строки |
| RTRIM | удаляет пробелы в конце строки |
| SOUNDEX | возвращает код звучания строки |
| SPACE | возвращает указанное число пробелов |
| STR | выполняет конвертирование значения числового типа в символьный формат |
| STUFF | удаляет указанное число символов, заменяя новой подстрокой |
| SUBSTRING | возвращает для строки подстроку указанной длины с заданного символа |
| UNICODE | возвращает Unicode-код левого символа строки |
| UPPER | переводит все символы строки в верхний регистр |

SELECT Фирма, [Фамилия]+""

+Left([Имя],1)+"."

+Left([Отчество],1)

+"." AS ФИО

FROM Клиент

Пример 11.5. Использование функции LEFT для получения инициалов клиентов.

**Функции для работы с датой и временем**

Краткий обзор основных функций для работы с датой и временем представлен в таблице.

|  |  |
| --- | --- |
| Таблица 11.3. | |
| DATEADD | добавляет к дате указанное значение дней, месяцев, часов и т.д. |
| DATEDIFF | возвращает разницу между указанными частями двух дат |
| DATENAME | выделяет из даты указанную часть и возвращает ее в символьном формате |
| DATEPART | выделяет из даты указанную часть и возвращает ее в числовом формате |
| DAY | возвращает число из указанной даты |
| GETDATE | возвращает текущее системное время |
| ISDATE | проверяет правильность выражения на соответствие одному из возможных форматов ввода даты |
| MONTH | возвращает значение месяца из указанной даты |
| YEAR | возвращает значение года из указанной даты |

SELECT Year(Дата) AS Год, Month(Дата)

AS Месяц,

Sum(Количество) AS Общ\_Количество

FROM Сделка

GROUP BY Year(Дата), Month(Дата)

Пример 11.6. Использование функций YEAR и MONTH для определения общего количества товара, проданного за каждый месяц каждого года.

DECLARE @d DATETIME

DECLARE @y INT

SET @d=’29.10.03’

SET @y=DATEPART(yy,@d)

SELECT @y

Пример 11.7. Пример выделения из даты значения года.

**Пользовательские параметры сортировки**

Чтобы добавить/изменить / удалить пользовательскую последовательность параметров сортировки, откройте редактор параметров сортировки: https://github.com/pawelsalawa/sqlitestudio/wiki/images/Tools_collations.png

Последовательность параметров сортировки-это определение алгоритма, которое используется SQLite для сравнения двух значений и выяснения, является ли одно меньше/больше/равно другому. Параметры сортировки могут использоваться в нескольких контекстах SQLite - например, столбец таблицы может быть определен с помощью определенных параметров сортировки, которые будут влиять на то, как выполняется сортировка для этого столбца, как группировка по этому столбцу и т. д. Ознакомиться с [SQLite в документации](https://www.sqlite.org/datatype3.html" \l "collation) для более подробной информации.

Сопоставление реализации, как писать [пользовательские функции SQL,](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#custom-sql-functions) реализация 2 Правил:

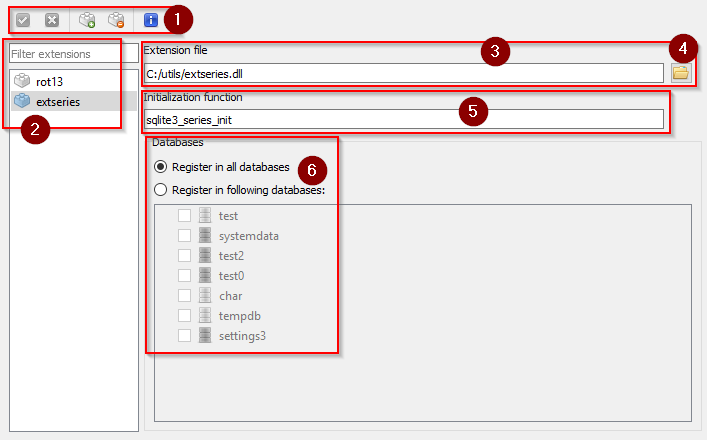
* Функция сортировки всегда будет снабжена ровно 2 аргументами-значениями для сравнения.
* Функция сортировки должна возвращать значения -1, 0 или 1, если первый аргумент меньше или больше второго.

**Расширения SQLite**

Расширения SQLite поддерживаются только в SQLite 3. [SQLite в документации](http://sqlite.org/loadext.html) дополнительные сведения о том, что SQLite для среды выполнения подгружаемые расширения.

Чтобы загрузить расширение SQLite в соединения с базой данных SQliteStudio, откройте Диспетчер расширений: https://github.com/pawelsalawa/sqlitestudio/wiki/images/Extension_manager_button.png

Окно Extension manager похоже на окно редактора параметров сортировки. С левой стороны есть список (2) уже зарегистрированных в приложении расширений, которые загружаются для определенного набора (или всех) баз данных. В верхней части (1) находится панель инструментов, где вы можете добавлять/удалять расширения и фиксировать/откатывать любые изменения, сделанные в окне менеджера. Также на этой панели инструментов есть кнопка справки, указывающая на этот раздел руководства пользователя.



Основная часть (3, 4, 5, 6) окна Extension manager будет включена/отключена в соответствии с вашим текущим выбором в списке (2) - Если в списке ничего не выбрано, то основная часть останется отключенной. При добавлении / редактировании определенного расширения необходимо указать путь к файлу библиотеки, который предоставляет расширение. Это то, что вы делаете в первом поле (3). Для упрощения можно использовать File dialog (4).

Поле функции инициализации (5) является необязательным и должно использоваться, если SQLite не смог инициализировать расширение из-за отсутствия функции инициализации. Как вы могли уже прочитать в [SQLite в документации](http://sqlite.org/loadext.html), инициализация функция наименование прогнозируется на основе библиотеки/расширения файла, так что если имя файла не соответствует инициализации функционировать соответствующим образом, необходимо обеспечить функцию инициализации вручную и вот что вы можете делать в поле (5). В противном случае вы можете оставить его пустым.

Список баз данных в области (6) похож на редактор параметров сортировки и редактор функций. Здесь можно определить, должно ли быть загружено определенное расширение для всех баз данных (только для SQLite версии 3), используемых в приложении, или оно должно быть загружено для предварительно выбранного набора баз данных.

SQLiteStudio будет пытаться загрузить (зонд) расширение, которое вы редактируете (в то время как вы вводите путь к файлу), так что вы будете уведомлены, если файл является недопустимым/нет или есть какие-либо другие проблемы (например, не найдена функция инициализации). Если что-то не так с файлом расширения, вы увидите пульсирующую красную точку в левом верхнем углу записи файла. Как только вы исправите путь к файлу или имя функции расширения инициализации, красная точка автоматически исчезнет.

Не забудьте зафиксировать ваши издания. В противном случае они не будут применяться.

SQLiteStudio версии 3.1.X и старше поддерживает расширение загрузки только через [load\_extension()](http://sqlite.org/lang_corefunc.html) функция SQL, выполненных в Редакторе SQL.

**Заполнение таблиц**

Заполнение таблиц-это способ вставки большого количества данных в таблицы, основываясь на некоторых правилах, чтобы иметь некоторые тестовые данные для работы. Он автоматизирует вставку большого количества данных, которые имеют предсказуемые значения, такие как последовательности o чисел, случайных чисел, случайный текст, случайные числа, и т.д., и т.д.

Заполнение осуществляется с помощью правил, которые определены [заполнение плагинов](https://github.com/pawelsalawa/sqlitestudio/wiki/Official_plugins#populating-plugins). Каждый плагин имеет свои входные аргументы (например, минимальные / максимальные значения и т. д.) и производит набор значений на основе входных аргументов и внутренних алгоритмов.

Чтобы заполнить таблицу данными, нажмите кнопку "заполнить таблицу" на панели инструментов окна таблицы. В диалоговом окне заполнить вы сможете определить, сколько строк данных Вы хотите создать и выбрать, какие столбцы должны быть заполнены. Невыбранные столбцы будут заполнены значениями по умолчанию. При выборе столбцов для заполнения необходимо выбрать плагин, используемый для заполнения-это определяет, какие данные будут вставлены в этот столбец. Большинство плагинов также имеют конфигурацию, доступную для указания некоторых деталей производства данных.

**История DDL**

При управлении многочисленными базами данных и выполнении большого количества изменений схемы полезно иметь инструмент для отслеживания и воспроизведения изменений, внесенных в схему. SQLiteStudio показывает предварительный просмотр DDL перед выполнением инструкции создания/изменения объекта базы данных (если эта опция не была отключена пользователем). Он получает работу, но это может быть неуклюжим, чтобы скопировать DDL и вставить в некоторые его в какой-то другой текстовый файл, чтобы сохранить историю. Это также рискованно, потому что иногда можно забыть скопировать DDL.

Вот где появляется окно истории DDL. Он хранит историю любых изменений DDL (Create statements, DROP statements и любых других операторов, связанных с изменениями объектов базы данных), независимо от того, включено или нет диалоговое окно предварительного просмотра DDL. Он хранит информацию о дате и времени выполнения и базе данных, на которой был выполнен DDL.

Вместе с SQL история [SQL редактор](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#sql-editor), ДДЛ окне истории делает любые изменения, сделанные на любой базе из SQLiteStudio воспроизводимы.

Чтобы открыть окно ddl история, нажмите на кнопку "ddl история" на панели инструментов: https://github.com/pawelsalawa/sqlitestudio/wiki/images/Tools_ddl_history.png

**Преобразование версии базы данных SQLite**

**Интерфейс командной строки (CLI)**

Если вы ищете параметры командной строки для запуска основных SQLiteStudio приложения см. В разделе [параметры командной строки](https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual#command-line-options).

CLI-это отдельное консольное приложение только для текста. Ознакомиться с [интерфейсом командной строки](https://github.com/pawelsalawa/sqlitestudio/wiki/CLI) для более подробной информации.

**Устранение неисправностей**

**SQLiteStudio разбился! Что мне делать?**

Сообщить об этом на [bugs.sqlitestudio.pl](http://bugs.sqlitestudio.pl/)или - если вы предпочитаете - вы можете сделать это с SQLiteStudio. Перейдите в меню " Справка "и выберите"Сообщить об ошибке".

Вы также можете обратиться за помощью на [forum.sqlitestudio.pl](http://forum.sqlitestudio.pl/). Где-то есть обходной путь для ошибки, так что вы можете применять их до того, что ошибка будет исправлена и выйдет в новой версии.

**Я сохранил двоичные данные в базу данных SQLite 2, но теперь они повреждены, пусты или усечены!**

**Следующее относится только к SQLite 2.**

**Длинная короткая история:** старый SQLite в 2 библиотеки очень ограничены в отношении типов данных, особенно когда дело доходит до двоичных данных. Лучше использовать обычные текстовые столбцы и хранить данные в кодировке base64.

**Более подробно:** чтобы иметь возможность хранить двоичные данные в SQLite таблице 2, необходимо определить тип данных столбца, как BLOB, или CLOB, так SQLiteStudio знает, что вы будете хранить двоичные данные в этом столбце. Без этого SQLiteStudio не может правильно записывать и читать двоичные данные из этого столбца.

Кроме того, на SQLite 2 версии есть баг, который делает хранение двоичных данных, чтобы быть усечены до первого возникновения *нулевого* байта (байт со значением = 0), как это делается для обычных строк. SQLiteStudio ничего не может с этим поделать. Если вы храните данные со всех *ненулевых* байт, единственным решением является использование регулярной TEXT основе столбцов и хранить данные в кодировке base64.

**Я использую метод ввода Fcitx, но он не работает в SQLiteStudio**

Не уверен насчет Windows или MacOS X, но в Linux вы можете использовать [О](http://en.wikipedia.org/wiki/Fcitx) с SQLiteStudio. Вот как это сделать.

* Если вы используете [Linux в управляемых пакет](https://github.com/pawelsalawa/sqlitestudio/wiki/Unix-like_application_distribution" \o "wikilink):

Единственное, что вам может понадобиться сделать, это изменить ярлык по умолчанию для sqlitestudio code assistant (Ctrl + пробел), так как он сталкивается с fcitx вызывая ярлык.

* Если вы используете [портативный дистрибутив](https://github.com/pawelsalawa/sqlitestudio/wiki/Portable_distribution) (бинарные пакеты загружаются с SQLiteStudio Домашняя страница):

Скопировать platforminputcontexts/libfcitxplatforminputcontextplugin.so из своего интервала QT установки в SQLiteStudio/platforminputcontexts/libfcitxplatforminputcontextplugin.so (вам придется создать platforminputcontexts папку), перезагрузка SQLiteStudio и все тут!

Другими словами, то, что вы должны сделать, это:

sudo apt-get install fcitx-frontend-qt5

ЛН-ы в /usr/lib в/`арка`-линукс-дистрибутив GNU/qt5/Плагины/platforminputcontexts/ /путь/к/SQLiteStudio3/

Просто не забудьте изменить код sqlitestudio помощник ярлык (Ctrl + пробел), так что он не сталкивается с Fcitx.

**Смотреть также**

1. [Вопросы и ответы](https://github.com/pawelsalawa/sqlitestudio/wiki/FAQ)
2. [форум](http://forum.sqlitestudio.pl/)