МИНИСТЕРСТВО НАУКИ и высшего образования РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ»

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

филиал «РКТ» МАИ в г. Химки Московской области

**Специальность 09.02.03 «Программирование в компьютерных системах»**

**ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**

**ПМ.02 «Разработка и администрирование баз данных**»

**Студент**

**Группы МП-32-17\_\_Гурушкин В.В\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)**

**Руководитель**

**практики от организации \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)**

**Руководитель**

**практики от филиала Шумаев А.Ю. / ( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ )**

**2020г.**

**ПРОГРАММА ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ**

По специальности 09.02.03 «Программирование в компьютерных системах»

|  |  |  |
| --- | --- | --- |
| Дата | Наименование выполняемых работ | Количество дней практики |
| 11.06.20 | Ознакомление с программой практики. Прохождение инструктажа по технике безопасности и охране труда, изучение внутреннего распорядка организации и правил работы. Разработка задания на производственную практику. | 4 |
| 12.06.20 | Описание структуры сети, в которой может функционировать разработанная база данных. | 4 |
| 13.06.20 | Описание сетевого оборудования необходимого для работы с базой данных. | 4 |
| 15.06.20 | Описание программного обеспечения необходимого для работы с базой данных по сети. | 4 |
| 16.06.20-18.06.20 | Разработка клиентских интерфейсов для клиент-серверных приложений. | 16 |
| 20.06.20 | Создание инфокоммуникационной системы.  Выполнение сетевых настроек для взаимодействия с СУБД. | 4 |
| 22.06.20-24.06.20 | Создание концептуальной модели данных.  Создание логической модели данных.  Создание физической модели данных.  Описание механизмов обеспечения целостности базы данных. | 16 |
| 25.06.20-01.07.20 | Описание СУБД представленной базы данных и ее возможностей. Внесение различных данных в базу данных.  Структурирование запросов базы данных.  Определение методов создания хранимых процедур и триггеров. Создание хранимых процедур и триггеров базы данных. | 50 |
| 02.07.20-03.07.20 | Определение способов управления правами пользователей.  Описание распределения прав пользователей и управления ими в базе данных. | 12 |
| 03.07.20-04.07.20 | Определение методов создания и синхронизации реплик базы данных. Описание существующих механизмов репликации в базе данных. Определение методов создания резервных копий базы данных. Описание существующих механизмов резервного копирования в базе данных. | 12 |
| 06.07.20 | Составление отчёта по практике | 6 |
| 07.07.20 | Подготовка к защите отчёта по практике. | 6 |
| 08.07.20 | Итоговая аттестация по производственной практике - зачёт | 6 |

Руководитель практики от филиала «РКТ» МАИ преподаватель

Шумаев А.Ю.

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись)*

**АТТЕСТАЦИОННЫЙ ЛИСТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**

студент \_\_\_\_\_\_\_\_\_Гурушкин Виктор Валерьевич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*ФИО*

обучающийся на 3-ем курсе по специальности СПО 09.02.03 «Программирование в компьютерных системах»

успешно прошел производственную практику по профессиональному модулю   
**ПМ.02 «Разработка и администрирование баз данных»**

в объеме 144 часа с 11.06.2020г. по 08.07.2020г.

в организации филиала «РКТ» МАИ

Виды и качество выполнения работ

|  |  |  |
| --- | --- | --- |
| Виды работ, выполненных обучающимся(ейся) во время практики | Объем работ | Качество выполнения работ в соответствии с технологией и (или) требованиями организации, в которой проходила практика |
| Ознакомление с программой практики. Прохождение инструктажа по технике безопасности и охране труда, изучение внутреннего распорядка организации и правил работы. Разработка задания на производственную практику. | 4 |  |
| Описание структуры сети, в которой может функционировать разработанная база данных. | 4 |  |
| Описание сетевого оборудования необходимого для работы с базой данных. | 4 |  |
| Описание программного обеспечения необходимого для работы с базой данных по сети. | 4 |  |
| Разработка клиентских интерфейсов для клиент-серверных приложений. | 16 |  |
| Создание инфокоммуникационной системы.  Выполнение сетевых настроек для взаимодействия с СУБД. | 4 |  |
| Создание концептуальной модели данных.  Создание логической модели данных.  Создание физической модели данных.  Описание механизмов обеспечения целостности базы данных. | 16 |  |
| Описание СУБД представленной базы данных и ее возможностей. Внесение различных данных в базу данных.  Структурирование запросов базы данных.  Определение методов создания хранимых процедур и триггеров. Создание хранимых процедур и триггеров базы данных. | 50 |  |
| Определение способов управления правами пользователей.  Описание распределения прав пользователей и управления ими в базе данных. | 12 |  |
| Определение методов создания и синхронизации реплик базы данных. Описание существующих механизмов репликации в базе данных. Определение методов создания резервных копий базы данных. Описание существующих механизмов резервного копирования в базе данных. | 12 |  |
| Составление отчёта по практике | 6 |  |
| Подготовка к защите отчёта по практике. | 6 |  |
| Итоговая аттестация по производственной практике - зачёт | 6 |  |

Руководитель практики от филиала «РКТ» МАИ Шумаев А.Ю.

|  |  |  |
| --- | --- | --- |
| Дата\_\_\_\_\_\_\_\_\_\_\_\_\_ | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *подпись* | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *Расшифровка подписи* |

**ДНЕВНИК** **ПРОХОЖДЕНИЯ ПРАКТИКИ**

|  |  |  |
| --- | --- | --- |
| Дата | Наименование выполняемых работ | Подпись  руководителя |
| 11.06.20 | Ознакомление с программой практики. Прохождение инструктажа по технике безопасности и охране труда, изучение внутреннего распорядка организации и правил работы. Разработка задания на производственную практику. |  |
| 12.06.20 | Описание структуры сети, в которой может функционировать разработанная база данных. |  |
| 13.06.20 | Описание сетевого оборудования необходимого для работы с базой данных. |  |
| 15.06.20 | Описание программного обеспечения необходимого для работы с базой данных по сети. |  |
| 16.06.20-18.06.20 | Разработка клиентских интерфейсов для клиент-серверных приложений. |  |
| 20.06.20 | Создание инфокоммуникационной системы.  Выполнение сетевых настроек для взаимодействия с СУБД. |  |
| 22.06.20-24.06.20 | Создание концептуальной модели данных.  Создание логической модели данных.  Создание физической модели данных.  Описание механизмов обеспечения целостности базы данных. |  |
| 25.06.20-01.07.20 | Описание СУБД представленной базы данных и ее возможностей. Внесение различных данных в базу данных.  Структурирование запросов базы данных.  Определение методов создания хранимых процедур и триггеров. Создание хранимых процедур и триггеров базы данных. |  |
| 02.07.20-03.07.20 | Определение способов управления правами пользователей.  Описание распределения прав пользователей и управления ими в базе данных. |  |
| 03.07.20-04.07.20 | Определение методов создания и синхронизации реплик базы данных. Описание существующих механизмов репликации в базе данных. Определение методов создания резервных копий базы данных. Описание существующих механизмов резервного копирования в базе данных. |  |
| 06.07.20 | Составление отчёта по практике |  |
| 07.07.20 | Подготовка к защите отчёта по практике. |  |
| 08.07.20 | Итоговая аттестация по производственной практике - зачёт |  |

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 7](#_Toc54046814)

[1 СУБД SQLITE 8](#_Toc54046815)

[2.1 Определение 8](#_Toc54046816)

[2.2 Особенности 10](#_Toc54046817)

[2.3 Использование 16](#_Toc54046818)

[2.4 Инструменты для разработки 22](#_Toc54046819)

[2 КЛАССЫ И КОНСТАНТЫ 27](#_Toc54046820)

[2.1 Предопределенные константы 27](#_Toc54046821)

[2.2 Класс SQLite3 28](#_Toc54046822)

[2.2 Класс SQLite3Stmt 30](#_Toc54046823)

[2.3 Класс SQLite3Result 31](#_Toc54046824)

[3 СИСТЕМА ОТСЛЕЖИВАНИЯ ОШИБОК 32](#_Toc54046825)

[3.1 Малозначимые ошибки в работе 32](#_Toc54046826)

[3.2 Ошибки при запросах 34](#_Toc54046827)

[ЗАКЛЮЧЕНИЕ 36](#_Toc54046828)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 37](#_Toc54046829)

# ВВЕДЕНИЕ

Практическая выполнена в соответствии с рабочей программой ПМ.02 «Разработка и администрирование баз данных». В работе представлены основные этапы анализа и разработки предметной области «Разработка и администрирование базы данных», навыки работы с GitHub и SQLite.

База данных играет важную роль в большинстве предметных областей. Благодаря динамической природе автоматизированных отраслей сейчас приложения требуют некоторых механизмов хранения, доступа и изменения данных. Поскольку важность баз данных стремительно растёт, реляционные системы управления базами данных набирают свою популярность.

В качестве программного обеспечения для создания интерфейса выбран редактор Notepad++. А для написания кода выбраны: язык гипертекстовой разметки HTML, язык таблиц стилей CSS и для работы с базами данных PHP. Использован веб-интерфейс phpmyadmin и СУБД SQLite.

В процессе выполнения проекта должны быть получены знания и опыт в области технологии СУБД «SQLite» Также мною будут решаться вопросы администрирования базы данных и реализовываться методы и технологии её защиты.

# 1 СУБД SQLITE

## 2.1 Определение

SQLite — это встраиваемая кроссплатформенная БД, которая поддерживает достаточно полный набор команд SQL и доступна в исходных кодах (на языке C). Позиция функциональности SQLite между MySQL и PostgreSQL. Однако, на практике, SQLite оказывается в 2-3 раза быстрее. Такое возможно благодаря высокоупорядоченной внутренней архитектуре и устранению необходимости в соединениях типа «сервер-клиент» и «клиент-сервер». На рисунке 1 представлена диаграмма со сравнением скорости операций трех разных СУБД

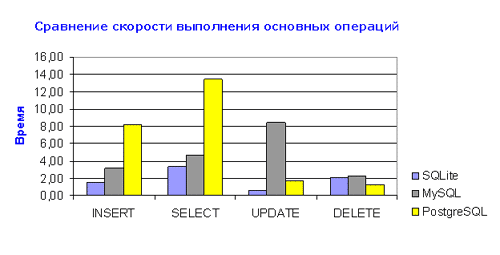


Рисунок 1. «Сравнение скорости операции СУБД»

Отличие SQLite от MySQL и аналогичных СУБД: Классические СУБД, такие как MySQL (а также MS SQL, Oracle, PostgreeSQL) состоят из отдельного сервера, поддерживающего работу базы данных и прослушивающих определённый порт, на предмет обращения клиентов. В качестве клиента может выступать в том числе и расширение PHP, реализующего интерфейс, с помощью которого осуществляются запросы к базе. Движок SQLite и интерфейс к ней реализованы в одной библиотеке, что увеличивает скорость выполнения запросов. Такой сервер часто называют встроенным.

Встроенный сервер имеется и у других баз данных, например, у MySQL, но его использование требует лицензионных отчислений, поэтому не получило широкое распространение в мире открытых исходных кодов.

SQLite является бестиповой базой данных. Точнее, есть только два типа – целочисленный «integer» и текстовый «text». Причём «integer» используется преимущественно для первичного ключа таблицы, а для остальных данных пойдёт «text». Длина строки, записываемой в текстовое поле, может быть любой. Все базы данных хранятся в файлах, по одному файлу на базу. Количество баз данных, а также таблиц в них, ограниченно только свободным местом, имеющимся на сайте. А максимально возможный объём одной базы данных составляет 2 Тб. Так как все данные хранятся в файлах, проблем с переносом базы данных с одного хостинга на другой не существует – достаточно лишь скопировать соответствующие файлы.

Всё это, собранное в один пакет, лишь немногим больше по размеру клиентской части библиотеки MySQL, является впечатляющим достижением для полноценной базы данных. Используя высоко эффективную инфраструктуру, SQLite может работать в крошечном объёме выделяемой для неё памяти, гораздо меньшем, чем в любых других системах БД. Это делает SQLite очень удобным инструментом с возможностью использования практически в любых задачах, возлагаемых на базу данных.

Движок БД представляет библиотеку, с которой программа компонуется и SQLite становится составной частью программы. Вся БД хранится в единственном стандартном файле на машине, на которой исполняется программа. Несколько процессов или потоков могут одновременно без каких-либо проблем читать данные из одной базы. Запись в базу можно осуществить только в том случае, если никаких других запросов в данный момент не обслуживается; в противном случае попытка записи оканчивается неудачей, и в программу возвращается код ошибки. Другим вариантом развития событий является автоматическое повторение попыток записи в течение заданного интервала времени.

Поддерживаемые типы данных: NULL: NULL-значение; INTEGER: целое со знаком, хранящееся в 1, 2, 3, 4, 6, или 8 байтах; REAL: число с плавающей запятой, хранящееся в 8-байтовом формате IEEE; TEXT: текстовая строка с кодировкой UTF-8, UTF-16BE или UTF-16LE; BLOB: тип данных, хранящийся точно в таком же виде, в каком и был получен.

Сама библиотека SQLite написана на C; существует большое количество привязок к другим языкам программирования, в том числе Apple Swift, Delphi, C++, Java, C#, VB.NET, Python, Perl, Node.js, PHP, PureBasic, Tcl (средства для работы с Tcl включены в комплект поставки SQLite), Ruby, Haskell, Scheme, Smalltalk, Lua и Parser, а также ко многим другим.

Простота и удобство встраивания SQLite привели к тому, что библиотека используется в браузерах, музыкальных плеерах и многих других программах. В частности, SQLite используется в: Adobe Integrated Runtime — среда для запуска приложений (частично); Gears; Autoit; Фреймворк Qt; Платформа XUL на движке Gecko 1.9+, XULRunner 1.9+ и, потенциально, все приложения, основанные на этой платформе, в том числе: Mozilla Firefox (начиная с версии 3.0); SQLite Manager; Skype; Viber;

Многие программы поддерживают SQLite в качестве формата хранения данных (особенно в Mac OS и iOS, Android), в том числе: 1С Предприятие; Adobe Photoshop Lightroom; AIMP; Google Chrome; Opera (начиная с версии 10.50); Safari; XnView;

## 2.2 Особенности

Использование SQLite в многопоточных приложениях:

SQLite может быть собран в однопоточном варианте (параметр компиляции SQLITE\_THREADSAFE = 0). В этом варианте его нельзя одновременно использовать из нескольких потоков, поскольку полностью отсутствует код синхронизации. Проверить, есть ли многопоточность можно через вызов sqlite3\_threadsafe (): если вернула 0, то это однопоточный SQLite. По умолчанию, SQLite собран с поддержкой потоков (sqlite3.dll). Есть два способа использования многопоточного SQLite: serialized и multi-thread.

Serialized (надо указать флаг SQLITE\_OPEN\_FULLMUTEX при открытии соединения). В этом режиме потоки могут как угодно дергать вызовы SQLite, никаких ограничений. Но все вызовы блокируют друг друга и обрабатываются строго последовательно.

Multi-thread (SQLITE\_OPEN\_NOMUTEX). В этом режиме нельзя использовать одно и то же соединение одновременно из нескольких потоков (но допускается одновременное использование разных соединений разными потоками). Обычно используется именно этот режим.

Формат данных:

База данных SQLite может хранить (текстовые) данные в UTF-8 или UTF-16. Набор вызовов API состоит из вызовов, которые получают UTF-8 (sqlite3\_XXX) и вызовов, которые получают UTF-16 (sqlite3\_XXX16). Если тип данных интерфейса и соединения не совпадает, то выполняется конвертация «на лету».

Поддержка UNICODE:

По умолчанию — поддержки нет. Нужно создать свой collation (способ сравнения) через sqlite3\_create\_collation. И определить свои встроенные функции like(), upper(), lower() через www.sqlite.org/c3ref/create\_function.html. Есть проект «International Components for Unicode», ICU и некоторые собирают SQLite DLL уже с ним.

Немного про работу ICU и SQLite:

Значение внутри БД может принадлежать к одному из следующих типов хранения (storage class): NULL, INTEGER (занимает 1,2,3,4,6 или 8 байт), REAL (число с плавающей точкой, 8 байт в формате IEEE), TEXT (строка в формате данных базы, обычно UTF-8), BLOB (двоичные данные, хранятся «как есть»).

Порядок сортировки значений разных типов:

NULL меньше всего (включая другой NULL);

INTEGER и REAL меньше любого TEXT и BLOB, между собой сравниваются арифметически;

TEXT меньше любого BLOB, между собой сравниваются на базе своих collation;

BLOB сравниваются между собой через memcmp().

SQLite выполняет неявные преобразования типов «на лету» в нескольких местах:

при занесении значения в столбец (тип столбца задает рекомендацию по преобразованию);

при сравнении значений между собой.

Столбец может иметь следующие рекомендации приведения типа: TEXT, NUMERIC, INTEGER, REAL, NONE.

Значения BLOB и NULL всегда заносятся в любой столбец «как есть».

В столбец TEXT значения TEXT заносятся «как есть», значения INTEGER и REAL становятся строками. В столбец NUMERIC, INTEGER числа записываются «как есть», а строки становятся числами, если могут (то есть допустимо обратное преобразование «без потерь»). Для столбца REAL правила похожи на INTEGER(NUMERIC); отличие в том, что все числа представлены в формате с плавающей запятой. В столбец NONE значения заносятся «как есть» (этот тип используется по умолчанию, если не задан другой).

При сравнении значений разного типа между собой может выполняться дополнительное преобразование типов.

При сравнении числа со строкой, если строка может быть преобразована в число «без потерь», она становится числом.

В SQLite в уникальном индексе может быть сколько угодно NULL значений (с этим согласен Oracle и не согласен MS SQL).

Если в вызове sqlite3\_open() передать имя файла как ":memory:", то SQLite создаст соединение к новой (чистой) БД в памяти. Это соединение абсолютно неотличимо от соединения к БД в файле по логике использования: доступен тот же набор SQL команд. Сейчас это исправлено и можно открыть два соединения к одной БД в памяти.

rc = sqlite3\_open("file:memdb1?mode=memory&cache=shared", &db);

ATTACH DATABASE 'file:memdb1?mode=memory&cache=shared' AS aux1;

Чтобы открыть соединение к БД используется вызов sqlite3\_open(). В любой момент времени мы можем к открытому соединению присоединить еще до 10 баз данных через SQL команду ATTACH DATABASE.

sqlite3\_open('toy-store.sqlite3', &db); // откроем соединение к БД в файле "toy-store.sqlite3"

sqlite3\_exec(&db, "ATTACH 'bar.sqlite3' AS bar", ... ); // присоединим "bar.sqlite3"

Теперь все таблицы БД в файле db1.sqlite3 стали прозрачно доступны в нашем соединении. Для разрешения конфликтов имен следует использовать имя присоединения (основная база называется «main»):

SELECT \* FROM main.my\_table UNION SELECT \* FROM bar.my\_table

Ничего не мешает присоединить к БД новую базу в памяти и использовать ее для кэширования и пр.

sqlite3\_open('toy-store.sqlite3', &db); // откроем соединение к БД в файле "toy-store.sqlite3"

sqlite3\_exec(&db, "ATTACH ':memory:' AS mem", ... ); // присоединим новую БД в памяти

Передайте пустую строку вместо имени файла в sqlite3\_open() и будет создана временная БД в файле на диске. Причем, после закрытия соединения к БД, она будет удалена с диска.

SQL команда PRAGMA служит для задания всевозможных настроек у соединения или у самой БД:

PRAGMA name; // запросить текущее значение параметра name

PRAGMA name = value; // задать параметр name значением value

Настройку соединения (очевидно) следует проводить сразу после открытия и до его использования.

Некоторые параметры:

PRAGMA page\_size = bytes; // размер страницы БД; страница БД - это единица обмена между диском и кэшом, разумно сделать равным размеру кластера диска

PRAGMA cache\_size = -kibibytes; // задать размер кэша соединения в килобайтах, по умолчанию он равен 2000 страниц БД

PRAGMA encoding = "UTF-8"; // тип данных БД, всегда используйте UTF-8

PRAGMA foreign\_keys = 1; // включить поддержку foreign keys, по умолчанию - ОТКЛЮЧЕНА

PRAGMA journal\_mode = DELETE | TRUNCATE | PERSIST | MEMORY | WAL | OFF; // задать тип журнала

PRAGMA synchronous = 0 | OFF | 1 | NORMAL | 2 | FULL; // тип синхронизации транзакции

Журнал и фиксация транзакций:

SQLite тщательно соблюдает целостность данных в БД (ACID), реализуя механизм изменения данных через транзакции. Кратко о транзакциях: транзакция либо полностью накатывается, либо полностью откатывается. Промежуточных состояний быть не может. Если вы не используете транзакции явно (BEGIN; ...; COMMIT;), то всегда создается неявная транзакция. Она стартует перед выполнением команды и использует коммит сразу после. Отсюда жалобы на «медленность» SQLite. SQLite может вставлять и до 50 тыс записей в секунду, но фиксировать транзакций он не может больше, чем ~ 50 в секунду. Именно поэтому, не получается вставлять записи быстро, используя неявную транзакцию.

При настройках по умолчанию SQLite гарантирует целостность БД даже при отключении питания в процессе работы. Достигается подобное изумительное поведение ведением журнала (специального файла) и хитроумным механизмом синхронизации изменений на диске. Обновление данных в БД работает так:

— до любой модификации БД SQLite сохраняет изменяемые страницы из БД в отдельном файле (журнале), то есть просто копирует их туда; — убедившись, что копия страниц создана, SQLite начинает менять БД; — убедившись, что все изменения в БД «дошли до диска» и БД стала целостной, SQLite стирает журнал.

Если SQLite открывает соединение к БД и видит, что журнал уже есть, он соображает, что БД находится в незавершенном состоянии и автоматически откатывает последнюю транзакцию. То есть механизм восстановления БД после сбоев, фактически, встроен в SQLite и работает незаметно для пользователя. По умолчанию журнал ведется в режиме DELETE.

PRAGMA journal\_mode = DELETE

Это означает, что файл журнала удаляется после завершения транзакции. Сам факт наличия файла с журналом в этом режиме означает для SQLite, что транзакция не была завершена, база нуждается в восстановлении. Файл журнала имеет имя файла БД, к которому добавлено "-journal".

В режиме TRUNCATE файл журнала обрезается до нуля (на некоторых системах это работает быстрее, чем удаление файла).

В режиме PERSIST начало файла журнала забивается нулями (при этом его размер не меняется, и он может занимать кучу места).

В режиме MEMORY файл журнала ведется в памяти и это работает быстро, но не гарантирует восстановление базы при сбоях (копии данных-то нету на диске).

А можно и совсем отключить журнал (PRAGMA journal\_mode = OFF). В этой ситуации перестает работать откат транзакций (команда ROLLBACK) и база, скорее всего, испортится, если программа будет завершена аварийно. Для базы данных в памяти режим журнала может быть только либо MEMORY, либо OFF.

Мы знаем, что современные системы используют хитроумное кэширование для повышения производительности и могут откладывать запись на диск. Допустим, SQLite завершил запись в БД и хочет стереть файл журнала, чтобы отметить факт фиксации транзакции. Если в этот промежуток времени отключится питание, то журнала уже не будет, а БД еще не будет целостной — потеря данных!

PRAGMA synchronous задает степень «паранойи» SQLite на это счет.

Режим OFF (или 0) означает: SQLite считает, что данные фиксированы на диске сразу после того как он передал их ОС (то есть сразу после вызова соответственного API ОС). Это означает, что целостность гарантирована при аварии приложения (поскольку ОС продолжает работать), но не при аварии ОС или отключении питания.

Режим синхронизации NORMAL (или 1) гарантирует целостность при авариях ОС и почти при всех отключениях питания. Существует ненулевой шанс, что при потере питания в самый неподходящий момент база испортится. Это некий средний, компромиссный режим по производительности и надежности.

Режим FULL гарантирует целостность всегда и везде и при любых авариях. Но работает, разумеется, медленнее, поскольку в определенных местах делаются паузы ожидания. И это режим по умолчанию.

Режим журнала WAL

По умолчанию, режим журнала БД всегда «возвращается» в DELETE. Допустим, мы открыли соединение к БД и установили режим PERSIST. Изменили данные, закрыли соединение. На диске остался файл журнала (начало которого забито нулями). Открываем соединение к БД снова. Если не задать режим журнала в этом соединении, он опять будет работать в DELETE. Как только мы обновим данные, механизм фиксации транзакций сотрет файл журнала.

Режим журнала WAL работает иначе — он «постоянный». Как только мы перевели базу в режим WAL, она останется в этом режиме, пока ей явно не поменяют режим журнала на другой.

Изначально SQLite проектировалась как встроенная БД. Архитектура разделения одновременного доступа к данным была устроена примитивно: одновременно несколько соединений могут читать БД, а вот записывать в данный момент времени может только одно соединение. Это, как минимум, означает, что пишущее соединение ждет «освобождения» БД от читающих. При попытке записать в «занятую» БД приложение получает ошибку SQLITE\_BUSY (не путать с SQLITE\_LOCKED!). Достигается этот механизм разделения доступа через API блокировки файлов (которые плохо работают на сетевых дисках, поэтому там не рекомендуется использовать SQLite; узнать больше )

В режиме WAL (Write-Ahead Logging) «читатели» БД и «писатели» в БД уже не мешают друг другу, то есть допускается модификация данных при одновременном чтении. Короче говоря, это шаг в сторону больших и серьезных СУБД, в которых все так и есть. Утверждается также, что SQLite в WAL работает быстрее.

Но есть и недостатки: — требуется некоторые дополнительные ништяки от ОС (unix и Windows имеют эти ништяки); — БД занимает несколько файлов (файлы «XXX-wal» и «XXX-shm»); — плохо работает на больших транзакциях (условно, если транзакция больше 50 Мбайт); — нельзя открыть такую БД в режиме «только чтение»; — возникает дополнительная операция checkpoint.

Фактически, в режиме WAL данные БД разделяются между БД и файлом журнала. Операция checkpoint переносит данные в БД. По умолчанию, это делается автоматически, если журнал занял 1000 страниц БД. То есть, идут быстрые COMMIT-ы и вдруг какой-то COMMIT задумался и начал делать checkpoint. Если такое поведение нежелательно, можно делать checkpoint вручную (когда все спокойно), можно это делать и в отдельном процессе.

Пределы:

Несмотря на миниатюрность, SQLite в реальности не накладывает серьезных ограничений на размеры полей, таблиц или БД.

По умолчанию, BLOB или строкое значение могут занимать 1 Гбайт и это же ограничение размера одной записи (можно поднять до 2^31 — 1, параметр SQLITE\_MAX\_LENGTH).

Количество столбцов: 2000 (можно поднять до 32767, SQLITE\_MAX\_COLUMN).

Размер SQL оператора: 1 МБайт (1073741824 байт, SQLITE\_MAX\_SQL\_LENGTH).

Одновременный join: 64 таблицы.

Присоединить баз к соединению: 10 (до 62, SQLITE\_MAX\_ATTACHED)

Максимальное количество страниц в БД: 1073741823 (до 2147483646, SQLITE\_MAX\_PAGE\_COUNT).

Если задать размер страницы 65636 байт, то максимальный размер БД будет примерно 14 Терабайт.

Максимальное число записей в таблице: 2^64 — 1, но на практике, конечно, ограничение размера вступит раньше

## 2.3 Использование

Установка SQLite: в PHP5+ поддержка SQLite установлена и включена по умолчанию. На рисунке 2 показан SQLite в программной среде Open Server 5.2.2.

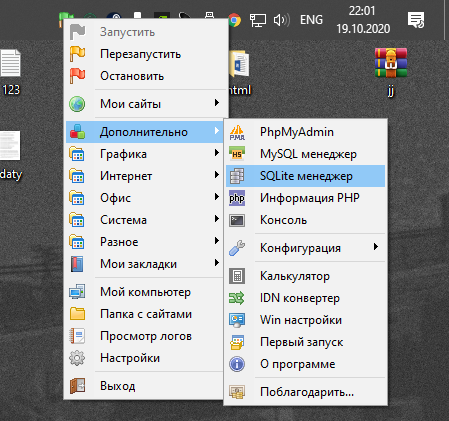


Рисунок 2. «SQLite менеджер в Open Server»

Установка под Windows: Для установки SQLite необходимо скачать и скопировать в папку с расширениями библиотеку «php\_sqlite.dll», которую можно загрузить по ссылке: http://snaps.php.net/win32/PECL\_STABLE/php\_sqlite.dll. Затем необходимо раскомментировать (или добавить) строку «extension=php\_sqlite.dll» в файле «php.ini». Для нормального функционирования SQLite также необходимо раскомментировать строку «extension=php\_pdo.dll». Если используется полная версия PHP в zip-архиве, а не в виде инсталлятора, соответствующие библиотеки расширения должны находится в директории ext. Библиотека «php\_pdo.dll» должна загружаться до загрузки «php\_sqlite.dll». То есть в php.ini строка «extension=php\_sqlite.dll» должна стоять после «extension=php\_pdo.dll».

Установка под Unix: Скачайте свежую версию SQLite с официального сайта (http://sqlite.org/download.html). Прочтите файл «INSTALL», поставляемый с исходными тестами модуля. Или просто воспользуйтесь командой установки PEAR: «pear install sqlite».

Синтаксис: Синтаксис SQL декларативен и читается подобно естественным языкам.

Утверждения выражаются в императивной форме, начиная с глагола, описывающего действие, далее следует субъект и предикат. Заметно, что оператор читается как нормальное предложение. На рисунке 3 изображен синтаксис SQLite.

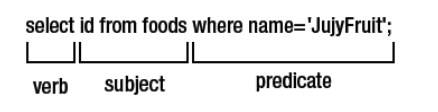


Рисунок 3. «Синтаксис в SQLite»

Операторы: Текст на языке SQL представляет собой последовательность операторов.

Операторы обычно разделяются символом точка-с-запятой, который отмечает конец оператора. Например, текст на рисунке 4 состоит из трех операторов.

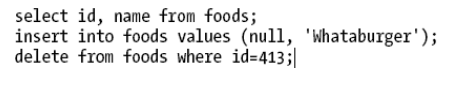


Рисунок 4. «Операторы в SQLite»

Точка-с-запятой используется в SQL как разделитель операторов (command terminator). Она отмечает конец оператора, который можно выполнять. Разделитель операторов ассоциируется с интерактивными программами, спроектированными для немедленного выполнения запросов в БД.

Константы: Константами, так же называемые литералами, явно записываются величины. Выделяется три типа констант: строки, числа и двоичные значения.

Строка - это одна или несколько символов в одинарных кавычках. Например:

’Toys’

‘Cars’

‘RKT\_MAI’

SQLite позволяет записывать строки в двойных кавычках тоже, настоятельно рекомендуется использовать только одинарные, для языка SQL стандартом является использование одинарных кавычек. Если одинарная кавычка является частью строки, надо вместо одной набрать две подряд. Числа можно записывать как целые, с десятичной точкой и научной нотации.

Двоичные константы записываются как пары шестнадцатеричных цифр (0-9A-F) в одинарных кавычках с лидирующим символом x.

Ключевые слова и идентификаторы: Слова, имеющие в SQL специальный смысл, называются ключевыми. В частности, к ним относятся: select, update, insert, create, drop, begin и так далее. Идентификаторы (имена) указывают на специальные объекты в базе данных, такие как таблицы или индексы. Ключевые слова зарезервированы и не могут быть использованы как идентификаторы. SQL не чувствителен к регистру относительно имен и ключевых слов. В строках, по умолчанию, регистр букв различается.

Комментарии: Комментарии в SQLite обозначаются двумя последовательными минусами (–), которые коментируют остаток строки. Многострочные комментарии, как в языке C, обозначаются парами символов ( /\* \*/). Пример представлен на рисунке 5.

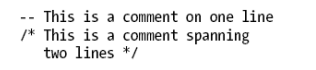


Рисунок 5. «Комментарии в SQLite»

Процедурный интерфейс к SQLite почти такой же, как у MySQL и других расширений БД. По большей части переход к SQLite потребует только изменить mysql/pq/etc... префикс функции на sqlite.

<?php

// Создадим базу данных

$db = sqlite\_open("toy-store.db");

if (!$db) exit("Не удалось создать базу данных!");

?>

Для работы с SQLite, как и любой реляционной базой данных используется язык запросов SQL. Поэтому создать таблицу данных можно при помощи традиционного запроса CREATE TABLE, вставить запись при помощи оператора INSERT, извлечь запись при помощи SELECT, а обновить существующую запись при помощи запроса UPDATE.

// создаём таблицу products

sqlite\_query($db, "CREATE TABLE products (id INTEGER PRIMARY KEY, name CHAR(255))");

// добавляем что-нибудь для примера

sqlite\_query($db, "INSERT INTO toy-store (name) VALUES (product1)");

sqlite\_query($db, "INSERT INTO toy-store (name) VALUES (product2)");

sqlite\_query($db, "INSERT INTO toy-store (name) VALUES (product3)");

// выполняем запрос

$result = sqlite\_query($db, "SELECT \* FROM products");

// проходим в цикле выборкой по ячейкам

while ($row = sqlite\_fetch\_array($result)) {

print\_r($row);

/\* каждый результат будет выглядеть примерно так

Array

(

[0] => 1

[id] => 1

[1] => product1

[name] => product1

)

\*/

}

// закрываем соединение с базой

sqlite\_close($db);

?>

Редактрирование записи: Для изменения поля воспользуемся функцией sqlite\_query() и передадим ей запрос на обновление (UPDATE).

<?php

// Создадим таблицу "table1" в базе

$query\_table = sqlite\_query($db, "CREATE TABLE table1

(id INTEGER PRIMARY KEY,

/\* id автоматически станет автоинкрементным \*/

field1 TEXT,

field2 TEXT);

");

if (!$query\_table) exit("Невозможно создать таблицу в базе данных!");

// Запишем что-нибудь в таблицу

sqlite\_query($db, "INSERT INTO table1(field1, field2) VALUES ('test', 'igrushka1');");

sqlite\_query($db, "INSERT INTO table1(field1, field2) VALUES ('test2', igrushka2');");

sqlite\_query($db, "INSERT INTO table1(field1, field2) VALUES ('Посетите ', 'наш магазин игрушек');");

// Изменим поле с id=1

sqlite\_query($db, "UPDATE table1 SET field2='igrushka novaya' WHERE id=1;");

// Сделаем выборку данных

$query = sqlite\_query($db, "SELECT \* FROM table1;");

// В цикле выведем все полученные данные

while ($array = sqlite\_fetch\_array($query))

{

echo($array['field1'].$array['field2']." (id записи:".$array['id'].")<br />");

}

?>

В результате получим:

1. test igrushka 1+ igrushka novaya (id записи:1)
2. test2 igrushka 2 (id записи:2)
3. посетите наш магазин игрушек (id записи:3)

Удаление записи из таблицы: Чтобы удалить запись из таблицы, нужно передать функции sqlite\_query() запрос на удаление (DELETE).

// Удалим поле с id=2

sqlite\_query($db, "DELETE FROM table1 WHERE id=2;");

// Сделаем выборку данных

$query = sqlite\_query($db, "SELECT \* FROM table1;");

// В цикле выведем все полученные данные

while ($array = sqlite\_fetch\_array($query))

{

echo($array['field1'].$array['field2']." (id записи:".$array['id'].")<br />");

}

В результате получим:

1. test igrushka 1+ igrushka novaya (id записи:1)
2. посетите наш магазин игрушек (id записи:3)

## 2.4 Инструменты для разработки

При разработке базы данных, удобно использовать инструмент Adminer, в котором и будет редактироваться БД проекта. Сначала стоит перейти в сам инструмент, послек чего в форме выбрать движок, сервер и авторизоваться (по умолчанию логин и пароль комбинация root-root». Формы для входа в Adminer представлены на рисунке 6.

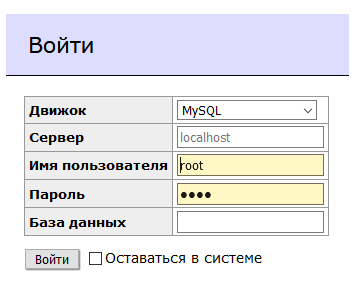


Рисунок 6. «Вход в Adminer»

После чего необходимо выбрать базу данных, на которой нужно работать. Список баз данных представлен на рисунке 7.

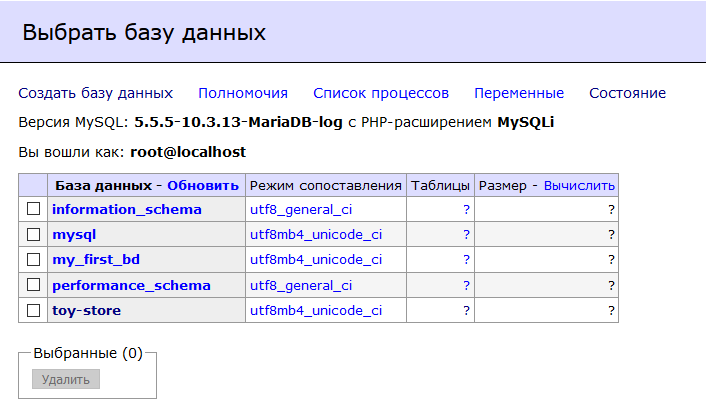


Рисунок 7. «Список баз данных в Adminer»

После входа и выбора базы данных, нам будет представлены списки таблиц. Можно создавать новые таблицы, работать с ними в SQL, импортировать и экспортировать текущие. Список таблиц базы данных представлен на рисунке 8.

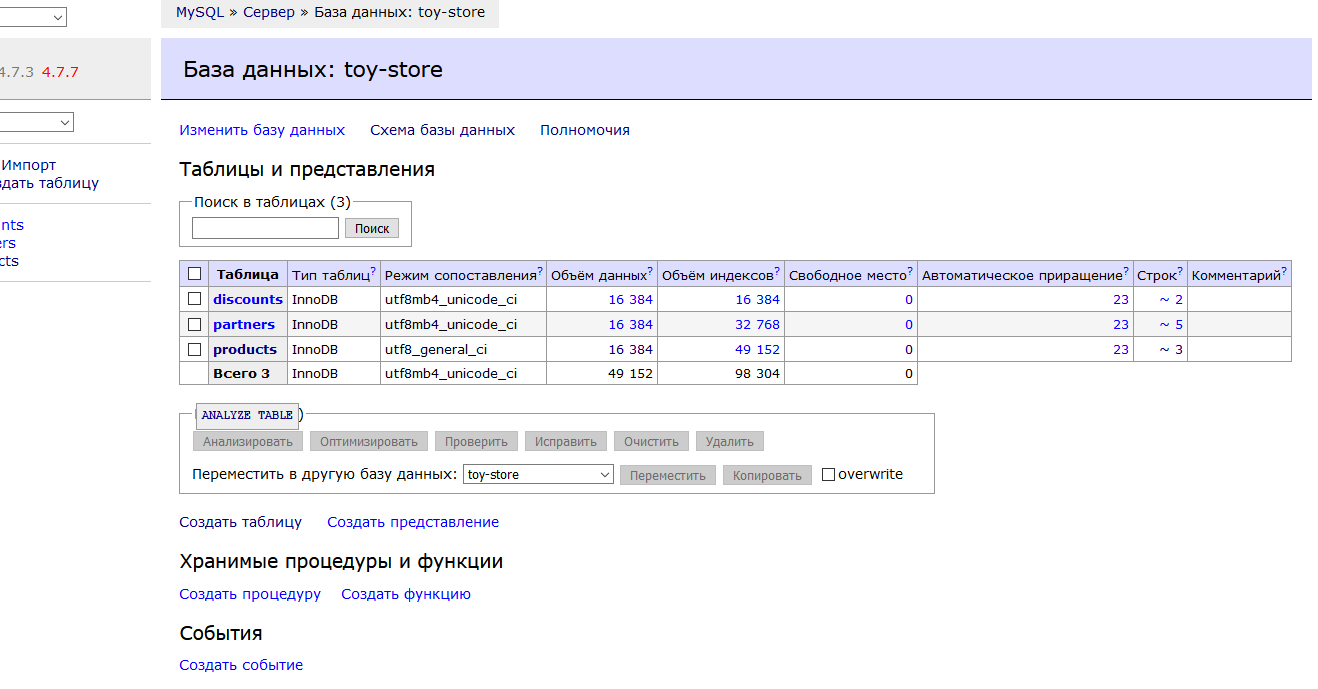


Рисунок 8. «Списки таблиц в Adminer»

Далее выберем таблицу «products». На рисунке 9 изображена категория «продукты» с ее компонентами.

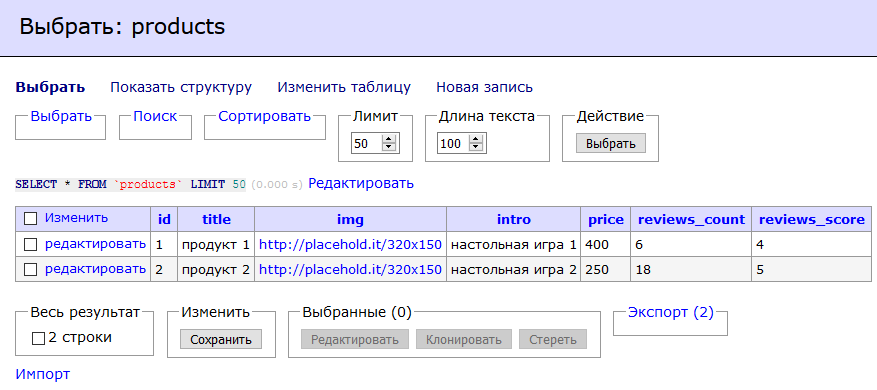


Рисунок 9. «products»

Нажав на ссылку «редактировать», мы перейдем в режим редактирования значения. На рисунке 10 изображено окно редактирования одного из продуктов категории «products»

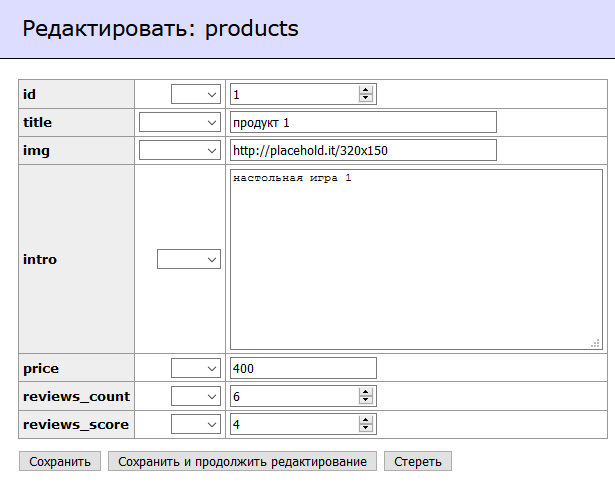


Рисунок 10. «Редактировать products»

Помимо основных функций, Adminer способен изменить название базы данных, показать схему базы данных и указать полномочия для пользователей, а для удобства навигации присутствует поиск в таблицах. На рисунке 11-15 изображены возможные действия с базой данных.

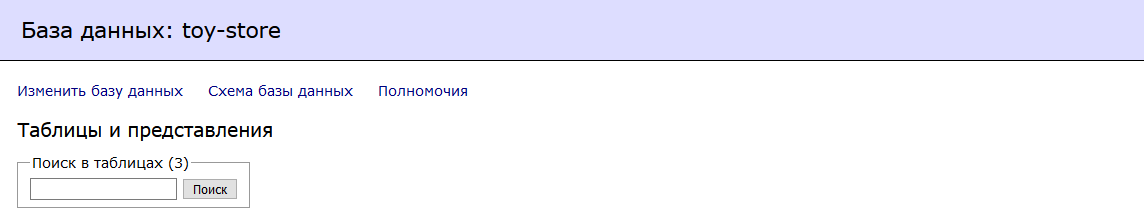


Рисунок 11. «Функции в Adminer»

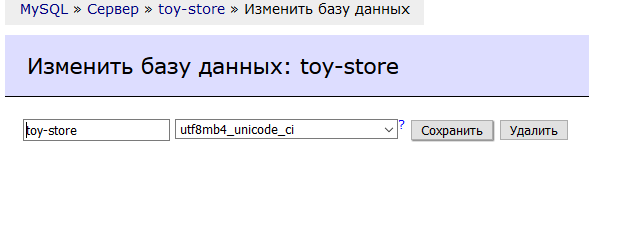


Рисунок 12. «Изменить название базы данных в Adminer»

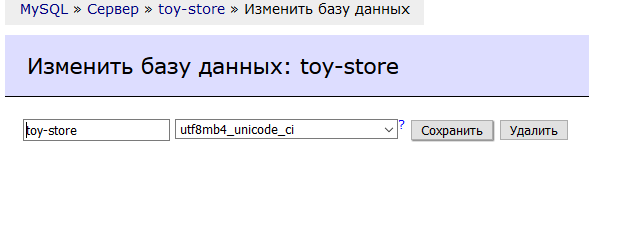


Рисунок 13. «Изменить название базы данных в Adminer»

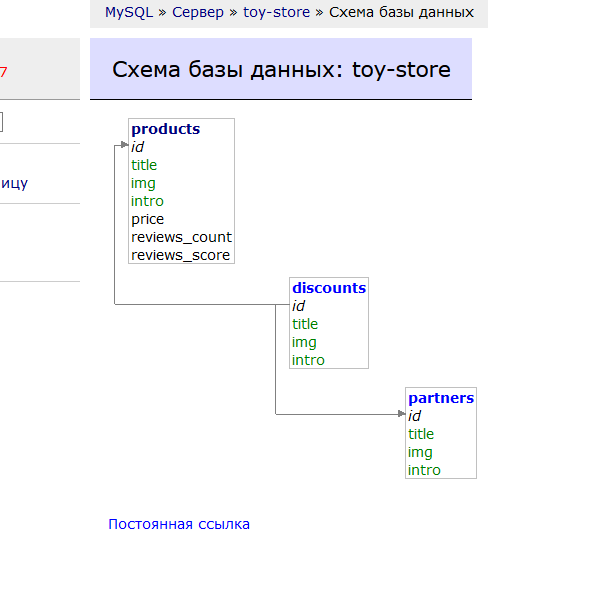


Рисунок 14. «Схемы базы данных в Adminer»

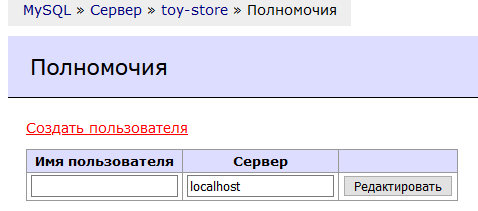


Рисунок 15. «Полномочия в Adminer»

# 2 КЛАССЫ И КОНСТАНТЫ

## 2.1 Предопределенные константы

Перечисленные ниже константы определены данным расширением и могут быть доступны только в том случае, если PHP был собран с поддержкой этого расширения или же в том случае, если данное расширение было подгружено во время выполнения.

* 1. SQLITE3\_ASSOC (integer)
  2. Указывает на то, что Sqlite3Result::fetchArray() метод должен вернуть массив, индексированный по имени столбца в соответствующий набор результатов.
  3. SQLITE3\_NUM (integer)
  4. Указывает на то, что Sqlite3Result::fetchArray() метод должен вернуть массив, индексированный по номеру столбца в соответствующий набор результатов, начиная со столбца 0.
  5. SQLITE3\_BOTH (integer)
  6. Указывает на то, что Sqlite3Result::fetchArray() метод должен вернуть массив, индексированный по имени столбца и его номеру в соответствующий набор результатов, начиная со столбца 0.
  7. SQLITE3\_INTEGER (integer)
  8. Представляет класс хранения типа SQLite3 INTEGER.
  9. SQLITE3\_FLOAT (integer)
  10. Представляет класс хранения типа SQLite3 REAL (FLOAT).
  11. SQLITE3\_TEXT (integer)
  12. Представляет класс хранения типа SQLite3 TEXT.
  13. SQLITE3\_BLOB (integer)
  14. Представляет класс хранения типа SQLite3 BLOB.
  15. SQLITE3\_NULL (integer)
  16. Представляет класс хранения типа SQLite3 NULL.
  17. SQLITE3\_OPEN\_READONLY (integer)
  18. Указывает на то, что база данных SQLite3 доступна только для чтения.
  19. SQLITE3\_OPEN\_READWRITE (integer)
  20. Указывает на то, что база данных SQLite3 доступна как для чтения, так и для записи.
  21. SQLITE3\_OPEN\_CREATE (integer)
  22. Указывает на то, что файл базы данных SQLite3 будет создан в том случае, если он еще не существует.
  23. SQLITE3\_DETERMINISTIC (integer) Указывает, что функция созданная SQLite3::sqliteCreateFunction() детерминированна. Она всегда возвращает одинаковый результат для для одинаковых входных данных в одном выражении SQL. (Доступно с PHP 7.1.4)

## 2.2 Класс SQLite3

Класс, предоставляющий доступ к API SQLite 3 базе данных.

SQLite3 {

/\* Методы \*/

public backup ( SQLite3 $destination\_db [, string $source\_dbname = "main" [, string $destination\_dbname = "main" ]] ) : bool

public busyTimeout ( int $msecs ) : bool

public changes ( void ) : int

public close ( void ) : bool

public \_\_construct ( string $filename [, int $flags = SQLITE3\_OPEN\_READWRITE | SQLITE3\_OPEN\_CREATE [, string $encryption\_key = "" ]] )

public createAggregate ( string $name , mixed $step\_callback , mixed $final\_callback [, int $argument\_count = -1 ] ) : bool

public createCollation ( string $name , callable $callback ) : bool

public createFunction ( string $name , mixed $callback [, int $argument\_count = -1 [, int $flags = 0 ]] ) : bool

enableExceptions ([ bool $enableExceptions = FALSE ] ) : bool

public static escapeString ( string $value ) : string

public exec ( string $query ) : bool

public lastErrorCode ( void ) : int

public lastErrorMsg ( void ) : string

public lastInsertRowID ( void ) : int

public loadExtension ( string $shared\_library ) : bool

public open ( string $filename [, int $flags = SQLITE3\_OPEN\_READWRITE | SQLITE3\_OPEN\_CREATE [, string $encryption\_key = "" ]] ) : void

public openBlob ( string $table , string $column , int $rowid [, string $dbname = "main" [, int $flags = SQLITE3\_OPEN\_READONLY ]] ) : resource

public prepare ( string $query ) : SQLite3Stmt

public query ( string $query ) : SQLite3Result

public querySingle ( string $query [, bool $entire\_row = FALSE ] ) : mixed

public static version ( void ) : array

}

Содержание:

* + 1. SQLite3::backup — Резервное копирование одной базы данных в другую базу данных
    2. SQLite3::busyTimeout — Установить обработчик "занято" на соединение
    3. SQLite3::changes — Получить количество строк, которые были изменены/удалены/вставлены последним запросом
    4. SQLite3::close — Закрыть соединение с базой данных
    5. SQLite3::\_\_construct — Создает экземпляр объекта SQLite3 и открывает соединение с базой
    6. SQLite3::createAggregate — Зарегистрировать функцию PHP в качестве агрегирующей функции SQL
    7. SQLite3::createCollation — Регистрирует функцию PHP для использования в качестве функции сортировки SQL
    8. SQLite3::createFunction — Регистрирует функцию PHP для использования в качестве скалярной функции SQL
    9. SQLite3::enableExceptions — Включить выброс исключений
    10. SQLite3::escapeString — Возвращает правильно экранированную строку
    11. SQLite3::exec — Выполняет запрос без результата к текущей базе данных
    12. SQLite3::lastErrorCode — Возвращает числовой код результата последнего неудачного запроса SQLite
    13. SQLite3::lastErrorMsg — Возвращает текст на английском, описывающий последний неудачный запрос SQLite
    14. SQLite3::lastInsertRowID — Возвращает идентификатор строки последней вставки (INSERT) в базу данных
    15. SQLite3::loadExtension — Попытаться загрузить библиотеку модуля SQLite
    16. SQLite3::open — Открыть базу данных SQLite
    17. SQLite3::openBlob — Открывает ресурс потока для чтения BLOB
    18. SQLite3::prepare — Подготавливает SQL-запрос для выполнения
    19. SQLite3::query — Выполняет SQL-запрос
    20. SQLite3::querySingle — Выполняет запрос и возвращает одиночный результат
    21. SQLite3::version — Возвращает версию библиотеки SQLite3, содержит как строковую константу, так и числовую

## 2.2 Класс SQLite3Stmt

Класс, предоставляющий доступ к подготовленным запросам в SQLite 3 расширении.

SQLite3Stmt {

/\* Методы \*/

public bindParam ( mixed $sql\_param , mixed &$param [, int $type ] ) : bool

public bindValue ( mixed $sql\_param , mixed $value [, int $type ] ) : bool

public clear ( void ) : bool

public close ( void ) : bool

public execute ( void ) : SQLite3Result

public getSQL ([ bool $expanded = FALSE ] ) : string

public paramCount ( void ) : int

public readOnly ( void ) : bool

public reset ( void ) : bool

}

Содержание:

* + 1. SQLite3Stmt::bindParam — Связывает параметр с переменной подготовленного запроса
    2. SQLite3Stmt::bindValue — Связывает значение параметра с переменной подготовленного запроса
    3. SQLite3Stmt::clear — Удаляет все текущие привязанные параметры
    4. SQLite3Stmt::close — Закрывает подготовленный запрос
    5. SQLite3Stmt::execute — Выполняет подготовленный запрос и возвращает объект с результирующим набором
    6. SQLite3Stmt::getSQL — Получить SQL-запрос в виде строки из запроса
    7. SQLite3Stmt::paramCount — Возвращает количество параметров в подготовленном запросе
    8. SQLite3Stmt::readOnly — Проверяет, является ли подготовленный запрос только для чтения
    9. SQLite3Stmt::reset — Сбрасывает подготовленный запрос

## 2.3 Класс SQLite3Result

Класс, предоставляющий доступ к результирующему набору расширения SQLite 3.

SQLite3Result {

/\* Методы \*/

public columnName ( int $column\_number ) : string

public columnType ( int $column\_number ) : int

public fetchArray ([ int $mode = SQLITE3\_BOTH ] ) : array

public finalize ( void ) : bool

public numColumns ( void ) : int

public reset ( void ) : bool

}

Содержание:

1. SQLite3Result::columnName — >Возвращает имя n-ного столбца
2. SQLite3Result::columnType — Возвращает тип n-ного столбца
3. SQLite3Result::fetchArray — Выбирает одну строку из результирующего набора и помещает ее в ассоциативный или нумерованный массив, или в оба сразу
4. SQLite3Result::finalize — Закрывает результирующий набор
5. SQLite3Result::numColumns — Возвращает количество столбцов в результирующем наборе
6. SQLite3Result::reset — Сбрасывает указатель результирующего набора на первую строку

# 3 СИСТЕМА ОТСЛЕЖИВАНИЯ ОШИБОК

## Малозначимые ошибки в работе

При входе в инструмент Adminer у многих пользователей может выявляться ошибка, представленная на рисунке 16.

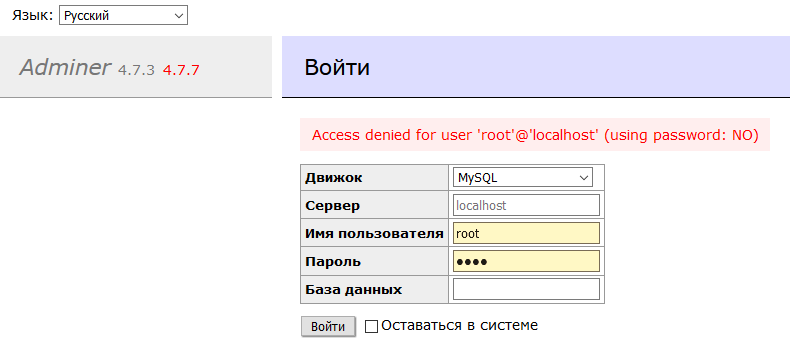


Рисунок 16. «Не получается войти в Adminer»

Чтобы свободно получить доступ в Adminer и в движке MySQL в целом, должно совпасть три параметра, описывающих пользователя базы данных — имя, название машины и пароль. Если есть несовпадения, доступ будет запрещен. Самая простая причина проблемы — неправильный ввод пароля. Кроме этого, вызывать ошибку может неправильный синтаксис. В системе MySQL нет простой зависимости имя пользователя – пароль, название хоста играет важную роль в получении доступа к БД. Оно может иметь вид IP-адреса, доменного имени, ключевого слова (например, localhost) или символа, объединяющего несколько машин в группу (например, % — любой хост, кроме локального).

Если ошибка Access denied for user ‘root’@’localhost’ (Using password: YES и NO) появляется с указанием Using password: YES, проблема заключается в неправильном вводе пароля. Проверить это можно, открыв таблицу mysql.user, в которой хранятся данные обо всех пользователях. На рисунке 17 изображен пример с таблицей данных.

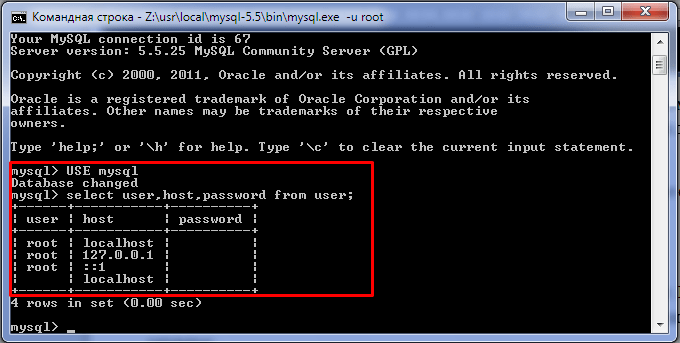


Рисунок 17. «Таблица данных пользоваталей»

Порядок действия для решения проблемы:

1. Откройте таблицу пользователей.
2. Проверьте, существует ли пользователь root с хостом localhost. Если он есть, смотрите на поле «password». Если там пусто, зайти в базу можно без ввода пароля. Если там что-то есть, значит, вы вводите неправильный пароль.
3. Смените пароль командой SET PASSWORD.
4. Если пользователя root нет, создайте его, установите пароль и предоставьте ему права.
5. После этого в базу данных можно зайти.

Если изменить данные не получается, следует использовать параметр —skip-grant-tables, который отменяет все настройки разрешений.

Если ошибка появляется с ключом (Using password: NO), нужно сделать следующее изменить файл config.inc.php, указав в нем правильные данные. На рисунке 18 изображен текстовый редактор с открытым файлом «config.inc.php»

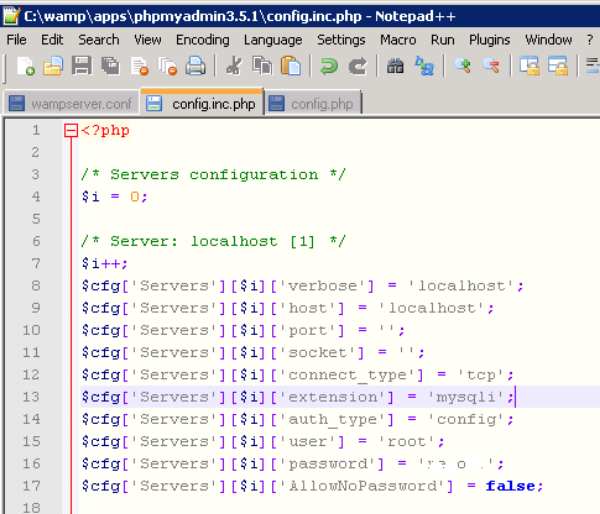


Рисунок 18. «config.inc.php»

Если проблема возникает при установке MySQL, нужно удалить базы данных старой версии программы или сменить пароль для доступа к ним, используя режим —skip-grant-tables.

Заключение: Таким образом, ошибка Access denied for user ‘root’@’localhost’ (Using password: YES или NO) возникает при несоответствии пароля и имени пользователя и легко исправляется заменой данных для входа.

## Ошибки при запросах

Существует запрос на удаление:

"DELETE FROM Products WHERE ProductName=1"

Возникает ошибка: sqlite3.OperationalError: near "Products": syntax error.

Правильный синтаксис:

"DELETE FROM Customers WHERE ProductName.id=1"

Запрос на обновление:

nickname = input('Введите ник:')

user\_login = int(1)

sql.execute(f'UPDATE users SET nick = {nickname} WHERE login = "{user\_login}"') db.commit()

Возникает ошибка: sqlite3.OperationalError: no such column: Max2005

Правильный синтаксис:

sql.execute(f'UPDATE users SET nick = ? WHERE login = ?', (nickname, user\_login))

Заключение: Таким образом, ошибки в синтаксисе возникают при несоответствии формату документации и исправляется простой заменой синтаксиса из официальной документации.

# ЗАКЛЮЧЕНИЕ

На практике ПМ.02 «Разработка и администрирование баз данных» были изучены и применены навыки создания серверной части базы данных. В частности, изучена компактная и встраиваемая СУБД SQLite и софт для администрирования PhpMyAdmin. Полученные знания были применены для создания серверной части базы данных на архитектуре «клиент-сервер», её настройки и разработки системы управления веб-сайтом «toy-store». Были получены знания в области СУБД «SQLite», а именно ее особенности и инструменты. Множество фрэймворков таких как Django, Ruby on Rails и web2py по умолчанию используют SQLite3, а многие браузеры используют данный инструмент для хранения локальных данных. Так же SQLite используется в качестве хранилища данных таких ОС как Android и Windows Phone 8.

В итоге закреплены такие навыки как работа с базами данных, веб-интерфейс, SQL разметка, синтаксис, классы и константы, а также система отслеживания ошибок.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. SQLite: [Электронный ресурс]/ https://ru.bmstu.wiki/SQLite - статья в интернете.
2. Руководство SQLite: [Электронный ресурс]/ https://proglib.io/p/sqlite-tutorial - статья в интернете.
3. SQLite документация: [Электронный ресурс]/ https://www.php.net/ - официальная документация.
4. Краткое вступление в SQLite: [Электронный ресурс]/ http://www.php.su/articles/?cat=phpdb&page=007 - статья в интернете.