Бесплатная электронная книга

учусь

sqlite

Free unaffiliated eBook created from
Stack Overflow contributors.

#sqlite

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: sqlite

It is an unofficial and free sqlite ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official sqlite.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с sqlite

Версии

Версия	Большие перемены	Дата выхода
3.0		2004-06-18
3.7.11	SELECT max (x), y	2012-03-20
3.8.3	, CTE	2014-02-11

Examples

Монтаж

SQLite - это библиотека C, которая обычно компилируется непосредственно в приложение, загружая исходный код последней версии и добавляя файл `sqlite3.c` в проект.

Многие языки сценариев (например, Perl , Python , Ruby и т. Д.) И фреймворки (например, Android) поддерживают SQLite; это делается со встроенной копией библиотеки SQLite, которую не нужно устанавливать отдельно.

Для тестирования SQL, это может быть полезно использовать оболочку командной строки (`sqlite3` или `sqlite3.exe`). Он уже поставляется с большинством дистрибутивов Linux; в Windows загрузите предварительно скомпилированные двоичные файлы в пакете `sqlite-tools` и извлеките их где-нибудь.

Документация

SQLite уже имеет обширную документацию , которую здесь не следует дублировать.

Прочитайте Начало работы с sqlite онлайн: https://riptutorial.com/ru/sqlite/topic/1753/начало-работы-с-sqlite

глава 2: sqlite3_stmt: Подготовленное заявление (C API)

замечания

официальная документация: подготовленный объект заявления

Examples

Выполнение заявления

Инструкция построена с такой функцией, как sqlite3_prepare_v2 () .

Подготовленный объект объекта *должен* быть очищен с помощью sqlite3_finalize () . Не забывайте об этом в случае ошибки.

Если используются параметры , задайте их значения с помощью функций sqlite3_bind_xxx () .

Фактическое выполнение происходит при вызове sqlite3_step () .

```
const char *sql = "INSERT INTO MyTable(ID, Name) VALUES (?, ?)";
sqlite3_stmt *stmt;
int err;

err = sqlite3_prepare_v2(db, sql, -1, &stmt, NULL);
if (err != SQLITE_OK) {
    printf("prepare failed: %s\n", sqlite3_errmsg(db));
    return /* failure */;
}

sqlite3_bind_int (stmt, 1, 42);                          /* ID */
sqlite3_bind_text(stmt, 2, "Bob", -1, SQLITE_TRANSIENT); /* name */

err = sqlite3_step(stmt);
if (err != SQLITE_DONE) {
    printf("execution failed: %s\n", sqlite3_errmsg(db));
    sqlite3_finalize(stmt);
    return /* failure */;
}

sqlite3_finalize(stmt);
return /* success */;
```

Чтение данных из курсора

Запрос SELECT выполняется, как и любой другой оператор. Чтобы прочитать возвращаемые данные, вызовите sqlite3_step () в цикле. Он возвращает:

- SQLITE_ROW: если доступны данные для следующей строки или
- SQLITE_DONE: если строк больше нет или
- любой код ошибки.

Если запрос не возвращает никаких строк, первый шаг возвращает SQLITE_DONE.

Чтобы прочитать данные из текущей строки, вызовите функции sqlite3_column_xxx () :

```
const char *sql = "SELECT ID, Name FROM MyTable";
sqlite3_stmt *stmt;
int err;

err = sqlite3_prepare_v2(db, sql, -1, &stmt, NULL);
if (err != SQLITE_OK) {
    printf("prepare failed: %s\n", sqlite3_errmsg(db));
    return /* failure */;
}

for (;;) {
    err = sqlite3_step(stmt);
    if (err != SQLITE_ROW)
        break;

    int         id   = sqlite3_column_int (stmt, 0);
    const char *name = sqlite3_column_text(stmt, 1);
    if (name == NULL)
        name = "(NULL)";
    printf("ID: %d, Name: %s\n", id, name);
}

if (err != SQLITE_DONE) {
    printf("execution failed: %s\n", sqlite3_errmsg(db));
    sqlite3_finalize(stmt);
    return /* failure */;
}

sqlite3_finalize(stmt);
return /* success */;
```

Выполнение подготовленного заявления несколько раз

После того, как заявление было выполнено , вызов sqlite3_reset () возвращает его в исходное состояние , так что он может быть повторно выполнен.

Как правило, хотя сам вывод остается неизменным, параметры меняются:

```
const char *sql = "INSERT INTO MyTable(ID, Name) VALUES (?, ?)";
sqlite3_stmt *stmt;
int err;

err = sqlite3_prepare_v2(db, sql, -1, &stmt, NULL);
if (err != SQLITE_OK) {
    printf("prepare failed: %s\n", sqlite3_errmsg(db));
    return /* failure */;
}
```

```
for (...) {
    sqlite3_bind_int (stmt, 1, ...);   /* ID */
    sqlite3_bind_text(stmt, 2, ...);   /* name */

    err = sqlite3_step(stmt);
    if (err != SQLITE_DONE) {
        printf("execution failed: %s\n", sqlite3_errmsg(db));
        sqlite3_finalize(stmt);
        return /* failure */;
    }

    sqlite3_reset(stmt);
}

sqlite3_finalize(stmt);
return /* success */;
```

Прочитайте sqlite3_stmt: Подготовленное заявление (C API) онлайн: https://riptutorial.com/ru/sqlite/topic/5456/sqlite3-stmt--подготовленное-заявление--c-api-

глава 3: Заявления PRAGMA

замечания

Документация SQLite содержит ссылку на все инструкции PRAGMA .

Examples

PRAGMA с постоянными эффектами

Большинство PRAGMA-операторов влияют только на текущее соединение с базой данных, что означает, что они должны быть повторно применены всякий раз, когда база данных была открыта.

Однако следующие PRAGMA записывают в файл базы данных и могут быть выполнены в любое время (но в некоторых случаях, а не внутри транзакции):

- application_id
- journal_mode при включении или отключении режима WAL
- schema_version
- user_version
- wal_checkpoint

Следующие настройки PRAGMA устанавливают свойства файла базы данных, которые не могут быть изменены после создания, поэтому они должны быть выполнены до первой фактической записи в базу данных:

- auto_vacuum (также можно изменить перед VACUUM)
- кодирование
- legacy_file_format
- page_size (также может быть изменено до VACUUM)

Например:

```
-- open a connection to a not-yet-existing DB file
PRAGMA page_size = 4096;
PRAGMA auto_vacuum = INCREMENTAL;
CREATE TABLE t(x);              -- database is created here, with the above settings
```

Прочитайте Заявления PRAGMA онлайн: https://riptutorial.com/ru/sqlite/topic/5223/заявления-pragma

глава 4: Текстовые команды командной строки

Вступление

`sqlite3` оболочка командной строки реализует дополнительный набор команд (которые не доступны в программах , которые используют SQLite библиотеки). Официальная документация: специальные команды для `sqlite3`

Examples

Экспорт и импорт таблицы в виде SQL-скрипта

Экспорт базы данных - это простой двухэтапный процесс:

```
sqlite> .output mydatabase_dump.sql
sqlite> .dump
```

Экспорт таблицы довольно похож:

```
sqlite> .output mytable_dump.sql
sqlite> .dump mytable
```

Выходной файл должен быть определен с `.output` до использования `.dump` ; в противном случае текст просто выводится на экран.

Импорт еще проще:

```
sqlite> .read mytable_dump.sql
```

Прочитайте Текстовые команды командной строки онлайн:
https://riptutorial.com/ru/sqlite/topic/3789/текстовые-команды-командной-строки

глава 5: Типы данных

замечания

официальная документация: Datatypes B SQLite версии 3

Examples

Функция TYPEOF

```
sqlite> SELECT TYPEOF(NULL);
null
sqlite> SELECT TYPEOF(42);
integer
sqlite> SELECT TYPEOF(3.141592653589793);
real
sqlite> SELECT TYPEOF('Hello, world!');
text
sqlite> SELECT TYPEOF(X'0123456789ABCDEF');
blob
```

Использование булевых

Для булевых языков SQLite использует целые числа `0` и `1` :

```
sqlite> SELECT 2 + 2 = 4;
1
sqlite> SELECT 'a' = 'b';
0
sqlite> SELECT typeof('a' = 'b');
integer
```

```
> CREATE TABLE Users ( Name, IsAdmin );
> INSERT INTO Users VALUES ('root', 1);
> INSERT INTO Users VALUES ('john', 0);
> SELECT Name FROM Users WHERE IsAdmin;
root
```

Применение типов столбцов

SQLite использует динамическую типизацию и игнорирует объявленные типы столбцов:

```
> CREATE TABLE Test (
      Col1 INTEGER,
      Col2 VARCHAR(2),       -- length is ignored, too
      Col3 BLOB,
      Col4,                  -- no type required
      Col5 FLUFFY BUNNIES    -- use whatever you want
```

```
  );
> INSERT INTO Test VALUES (1, 1, 1, 1, 1);
> INSERT INTO Test VALUES ('xxx', 'xxx', 'xxx', 'xxx', 'xxx');
> SELECT * FROM Test;
1   1   1   1   1
xxx xxx xxx xxx xxx
```

(Однако объявленные типы столбцов используются для сродства типа .)

Чтобы применять типы, вам необходимо добавить ограничение с помощью функции typeof () :

```
CREATE TABLE Tab (
    Col1 TEXT   CHECK (typeof(Col1) = 'text' AND length(Col1) <= 10),
    [...]
);
```

(Если такой столбец должен быть NULLable, вы должны явно разрешить `'null'` .)

Типы даты / времени

SQLite не имеет отдельного типа данных для значений даты или времени.

Строки ISO8601

Встроенные ключевые слова `CURRENT_DATE` , `CURRENT_TIME` и `CURRENT_TIMESTAMP` возвращают строки в формате ISO8601:

```
> SELECT CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP;
CURRENT_DATE  CURRENT_TIME  CURRENT_TIMESTAMP
------------  ------------  -------------------
2016-07-08    12:34:56      2016-07-08 12:34:56
```

Такие значения также понимаются всеми встроенными функциями даты / времени :

```
> SELECT strftime('%Y', '2016-07-08');
2016
```

Юлианские числа

Встроенные функции даты / времени интерпретируют числа как юлианские дни :

```
> SELECT datetime(2457578.02425926);
2016-07-08 12:34:56
```

Функция `julianday()` преобразует любое поддерживаемое значение даты / времени в число

юлианских дней:

```
> SELECT julianday('2016-07-08 12:34:56');
2457578.02425926
```

Временные метки Unix

Встроенные функции даты / времени могут интерпретировать числа как временные метки Unix с `unixepoch` модификатора `unixepoch` :

```
> SELECT datetime(0, 'unixepoch');
1970-01-01 00:00:00
```

Функция `strftime()` может преобразовать любое поддерживаемое значение даты / времени в метку времени Unix:

```
> SELECT strftime('%s', '2016-07-08 12:34:56');
1467981296
```

неподдерживаемые форматы

Можно было бы сохранить значения даты / времени в любом другом формате в базе данных, но встроенные функции даты / времени не будут анализировать их и возвращать NULL:

```
> SELECT time('1:30:00');   -- not two digits

> SELECT datetime('8 Jul 2016');
▯
```

Прочитайте Типы данных онлайн: https://riptutorial.com/ru/sqlite/topic/5252/типы-данных

кредиты

S. No	Главы	Contributors
1	Начало работы с sqlite	CL., Community, e4c5, H. Pauwelyn
2	sqlite3_stmt: Подготовленное заявление (C API)	CL.
3	Заявления PRAGMA	CL., springy76
4	Текстовые команды командной строки	CL., e4c5, James Toomey, Lasse Vågsæther Karlsen, ravenspoint, Thinkeye
5	Типы данных	CL.