Sqlite开发

# 环境

配置日期：2015年6月1日

Sqlite: sqlite-amalgamation-3081002.zip (源文件)

# An Introduction To The SQLite C/C++ Interface

SQLite C/C++接口简介

## Executive Summary

概要

The following two objects and eight methods comprise the essential elements of the SQLite interface:

以下2个对象和8个函数组成了SQLite接口的基本元素。

### sqlite3

The database connection object. Created by sqlite3\_open() and destroyed by sqlite3\_close().

数据库连接对象。由sqlite3\_open()创建，sqlite3\_close()销毁。

### sqlite3\_stmt

The prepared statement object. Created by sqlite3\_prepare() and destroyed by sqlite3\_finalize().

预处理语句对象。由sqlite3\_prepare()创建，sqlite3\_finalize()销毁。

### sqlite3\_open()

Open a connection to a new or existing SQLite database. The constructor for sqlite3.

打开一个新的或已存在的SQLite数据库连接。sqlite3对象的构造函数。

### sqlite3\_prepare()

Compile SQL text into byte-code that will do the work of querying or updating the database. The constructor for sqlite3\_stmt.

将SQL语句编译为字节码，用以对数据库进行查询或修改。sqlite3\_stmt对象的构造函数。

### sqlite3\_bind()

Store application data into parameters of the original SQL.

将程序数据存储到原始SQL中。

### sqlite3\_step()

Advance an sqlite3\_stmt to the next result row or to completion.

移动sqlite3\_stmt对象到下一个结果行或结束位置。

### sqlite3\_column()

Column values in the current result row for an sqlite3\_stmt.

sqlite3\_stmt对象当结果行的列值。

### sqlite3\_finalize()

Destructor for sqlite3\_stmt.

sqlite3\_stmt对象的析构函数。

### sqlite3\_close()

Destructor for sqlite3.

sqlite3对象的析构函数

### sqlite3\_exec()

A wrapper function that does sqlite3\_prepare(), sqlite3\_step(), sqlite3\_column(), and sqlite3\_finalize() for a string of one or more SQL statements.

对一个或多个SQL语句字符串的包装函数，依次调用sqlite3\_prepare(), sqlite3\_step(), sqlite3\_column(), 和sqlite3\_finalize()。

## Introduction

SQLite currently has over 200 distinct APIs. This can be overwhelming to a new programmer. Fortunately, most of the interfaces are very specialized and need not be considered by beginners. The core API is small, simple, and easy to learn. This article summarizes the core API.

A separate document, [The SQLite C/C++ Interface](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\intro.html), provides detailed specifications for all C/C++ APIs for SQLite. Once the reader understands the basic principles of operation for SQLite, [that document](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\intro.html) should be used as a reference guide. This article is intended as introduction only and is neither a complete nor authoritative reference for the SQLite API.

## Core Objects And Interfaces

The principal task of an SQL database engine is to evaluate SQL statements. of SQL. To accomplish this, the developer needs two objects:

* The [database connection](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\sqlite3.html) object: sqlite3
* The [prepared statement](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) object: sqlite3\_stmt

Strictly speaking, the [prepared statement](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) object is not required since the convenience wrapper interfaces, [sqlite3\_exec](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\exec.html) or [sqlite3\_get\_table](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\free_table.html), can be used and these convenience wrappers encapsulate and hide the [prepared statement](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) object. Nevertheless, an understanding of [prepared statements](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) is needed to make full use of SQLite.

The [database connection](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\sqlite3.html) and [prepared statement](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) objects are controlled by a small set of C/C++ interface routine listed below.

* [sqlite3\_open()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\open.html)
* [sqlite3\_prepare()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\prepare.html)
* [sqlite3\_step()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\step.html)
* [sqlite3\_column()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\column_blob.html)
* [sqlite3\_finalize()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\finalize.html)
* [sqlite3\_close()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\close.html)

Note that the list of routines above is conceptual rather than actual. Many of these routines come in multiple versions. For example, the list above shows a single routine named [sqlite3\_open()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\open.html) when in fact there are three separate routines that accomplish the same thing in slightly different ways: [sqlite3\_open()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\open.html), [sqlite3\_open16()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\open.html) and [sqlite3\_open\_v2()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\open.html). The list mentions [sqlite3\_column()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\column_blob.html) when in fact no such routine exists. The "sqlite3\_column()" shown in the list is place holders for an entire family of routines to be used for extracting column data in various datatypes.

Here is a summary of what the core interfaces do:

|  |  |
| --- | --- |
| [sqlite3\_open()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\open.html) | This routine opens a connection to an SQLite database file and returns a [database connection](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\sqlite3.html) object. This is often the first SQLite API call that an application makes and is a prerequisite for most other SQLite APIs. Many SQLite interfaces require a pointer to the [database connection](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\sqlite3.html) object as their first parameter and can be thought of as methods on the [database connection](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\sqlite3.html) object. This routine is the constructor for the [database connection](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\sqlite3.html) object. |
| [sqlite3\_prepare()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\prepare.html) | This routine converts SQL text into a [prepared statement](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) object and returns a pointer to that object. This interface requires a [database connection](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\sqlite3.html) pointer created by a prior call to [sqlite3\_open()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\open.html) and a text string containing the SQL statement to be prepared. This API does not actually evaluate the SQL statement. It merely prepares the SQL statement for evaluation.  Think of each SQL statement as a small computer program. The purpose of [sqlite3\_prepare()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\prepare.html) is to compile that program into object code. The [prepared statement](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) is the object code. The [sqlite3\_step()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\step.html) interface then runs the object code to get a result.  New applications should always invoke [sqlite3\_prepare\_v2()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\prepare.html) instead of [sqlite3\_prepare()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\prepare.html). The older [sqlite3\_prepare()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\prepare.html) is retained for backwards compatibility. But [sqlite3\_prepare\_v2()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\prepare.html) provides a much better interface. |
| [sqlite3\_step()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\step.html) | This routine is used to evaluate a [prepared statement](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) that has been previously created by the [sqlite3\_prepare()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\prepare.html) interface. The statement is evaluated up to the point where the first row of results are available. To advance to the second row of results, invoke [sqlite3\_step()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\step.html) again. Continue invoking [sqlite3\_step()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\step.html) until the statement is complete. Statements that do not return results (ex: INSERT, UPDATE, or DELETE statements) run to completion on a single call to [sqlite3\_step()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\step.html). |
| [sqlite3\_column()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\column_blob.html) | This routine returns a single column from the current row of a result set for a [prepared statement](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) that is being evaluated by [sqlite3\_step()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\step.html). Each time [sqlite3\_step()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\step.html) stops with a new result set row, this routine can be called multiple times to find the values of all columns in that row.  As noted above, there really is no such thing as a "sqlite3\_column()" function in the SQLite API. Instead, what we here call "sqlite3\_column()" is a place-holder for an entire family of functions that return a value from the result set in various data types. There are also routines in this family that return the size of the result (if it is a string or BLOB) and the number of columns in the result set.   * [sqlite3\_column\_blob()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\column_blob.html) * [sqlite3\_column\_bytes()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\column_blob.html) * [sqlite3\_column\_bytes16()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\column_blob.html) * [sqlite3\_column\_count()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\column_count.html) * [sqlite3\_column\_double()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\column_blob.html) * [sqlite3\_column\_int()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\column_blob.html) * [sqlite3\_column\_int64()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\column_blob.html) * [sqlite3\_column\_text()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\column_blob.html) * [sqlite3\_column\_text16()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\column_blob.html) * [sqlite3\_column\_type()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\column_blob.html) * [sqlite3\_column\_value()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\column_blob.html) |
| [sqlite3\_finalize()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\finalize.html) | This routine destroys a [prepared statement](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) created by a prior call to [sqlite3\_prepare()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\prepare.html). Every prepared statement must be destroyed using a call to this routine in order to avoid memory leaks. |
| [sqlite3\_close()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\close.html) | This routine closes a [database connection](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\sqlite3.html) previously opened by a call to [sqlite3\_open()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\open.html). All [prepared statements](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) associated with the connection should be [finalized](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\finalize.html) prior to closing the connection. |

### Typical Usage Of Core Routines And Objects

An application will typically use [sqlite3\_open()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\open.html) to create a single [database connection](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\sqlite3.html) during initialization. Note that [sqlite3\_open()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\open.html) can be used to either open existing database files or to create and open new database files. While many applications use only a single [database connection](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\sqlite3.html), there is no reason why an application cannot call [sqlite3\_open()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\open.html) multiple times in order to open multiple [database connections](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\sqlite3.html) - either to the same database or to different databases. Sometimes a multi-threaded application will create separate [database connections](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\sqlite3.html) for each threads. Note that a single [database connection](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\sqlite3.html) can access two or more databases using the [ATTACH](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\lang_attach.html) SQL command, so it is not necessary to have a separate database connection for each database file.

Many applications destroy their [database connections](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\sqlite3.html) using calls to [sqlite3\_close()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\close.html) at shutdown. Or, for example, an application that uses SQLite as its [application file format](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\appfileformat.html) might open [database connections](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\sqlite3.html) in response to a File/Open menu action and then destroy the corresponding [database connection](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\sqlite3.html) in response to the File/Close menu.

To run an SQL statement, the application follows these steps:

1. Create a [prepared statement](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) using [sqlite3\_prepare()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\prepare.html).
2. Evaluate the [prepared statement](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) by calling [sqlite3\_step()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\step.html) one or more times.
3. For queries, extract results by calling [sqlite3\_column()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\column_blob.html) in between two calls to [sqlite3\_step()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\step.html).
4. Destroy the [prepared statement](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) using [sqlite3\_finalize()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\finalize.html).

The foregoing is all one really needs to know in order to use SQLite effectively. All the rest is optimization and detail.

## Convenience Wrappers Around Core Routines

The [sqlite3\_exec()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\exec.html) interface is a convenience wrapper that carries out all four of the above steps with a single function call. A callback function passed into [sqlite3\_exec()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\exec.html) is used to process each row of the result set. The [sqlite3\_get\_table()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\free_table.html) is another convenience wrapper that does all four of the above steps. The [sqlite3\_get\_table()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\free_table.html) interface differs from [sqlite3\_exec()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\exec.html) in that it stores the results of queries in heap memory rather than invoking a callback.

It is important to realize that neither [sqlite3\_exec()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\exec.html) nor [sqlite3\_get\_table()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\free_table.html) do anything that cannot be accomplished using the core routines. In fact, these wrappers are implemented purely in terms of the core routines.

## Binding Parameters and Reusing Prepared Statements

In prior discussion, it was assumed that each SQL statement is prepared once, evaluated, then destroyed. However, SQLite allows the same [prepared statement](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) to be evaluated multiple times. This is accomplished using the following routines:

* [sqlite3\_reset()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\reset.html)
* [sqlite3\_bind()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\bind_blob.html)

After a [prepared statement](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) has been evaluated by one or more calls to [sqlite3\_step()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\step.html), it can be reset in order to be evaluated again by a call to [sqlite3\_reset()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\reset.html). Think of [sqlite3\_reset()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\reset.html) as rewinding the [prepared statement](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) program back to the beginning. Using [sqlite3\_reset()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\reset.html) on an existing [prepared statement](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) rather than creating a new [prepared statement](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) avoids unnecessary calls to [sqlite3\_prepare()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\prepare.html). For many SQL statements, the time needed to run [sqlite3\_prepare()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\prepare.html) equals or exceeds the time needed by [sqlite3\_step()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\step.html). So avoiding calls to [sqlite3\_prepare()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\prepare.html) can give a significant performance improvement.

It is not commonly useful to evaluate the exact same SQL statement more than once. More often, one wants to evaluate similar statements. For example, you might want to evaluate an INSERT statement multiple times with different values. Or you might want to evaluate the same query multiple times using a different key in the WHERE clause. To accommodate this, SQLite allows SQL statements to contain [parameters](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\lang_expr.html#varparam) which are "bound" to values prior to being evaluated. These values can later be changed and the same [prepared statement](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) can be evaluated a second time using the new values.

SQLite allows a [parameter](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\lang_expr.html#varparam) wherever a string literal, numeric constant, or NULL is allowed. (Parameters may not be used for column or table names.) A [parameter](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\lang_expr.html#varparam) takes one of the following forms:

* **?**
* **?***NNN*
* **:***AAA*
* **$***AAA*
* **@***AAA*

In the examples above, *NNN* is an integer value and *AAA* is an identifier. A parameter initially has a value of NULL. Prior to calling [sqlite3\_step()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\step.html) for the first time or immediately after [sqlite3\_reset()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\reset.html), the application can invoke the [sqlite3\_bind()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\bind_blob.html) interfaces to attach values to the parameters. Each call to [sqlite3\_bind()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\bind_blob.html) overrides prior bindings on the same parameter.

An application is allowed to prepare multiple SQL statements in advance and evaluate them as needed. There is no arbitrary limit to the number of outstanding [prepared statements](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html). Some applications call [sqlite3\_prepare()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\prepare.html) multiple times at start-up to create all of the [prepared statements](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) they will ever need. Other applications keep a cache of the most recently used [prepared statements](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) and then reuse [prepared statements](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) out of the cache when available. Another approach is to only reuse [prepared statements](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\stmt.html) when they are inside of a loop.

## Configuring SQLite

The default configuration for SQLite works great for most applications. But sometimes developers want to tweak the setup to try to squeeze out a little more performance, or take advantage of some obscure feature.

The [sqlite3\_config()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\config.html) interface is used to make global, process-wide configuration changes for SQLite. The [sqlite3\_config()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\config.html) interface must be called before any [database connections](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\sqlite3.html) are created. The [sqlite3\_config()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\config.html) interface allows the programmer to do things like:

* Adjust how SQLite does [memory allocation](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\malloc.html), including setting up alternative memory allocators appropriate for safety-critical real-time embedded systems and application-defined memory allocators.
* Set up a process-wide [error log](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\errlog.html).
* Specify an application-defined page cache.
* Adjust the use of mutexes so that they are appropriate for various [threading models](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\threadsafe.html), or substitute an application-defined mutex system.

After process-wide configuration is complete and [database connections](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\sqlite3.html) have been created, individual database connections can be configured using calls to [sqlite3\_limit()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\limit.html) and [sqlite3\_db\_config()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\db_config.html).

## Extending SQLite

SQLite includes interfaces that can be used to extend its functionality. Such routines include:

* [sqlite3\_create\_collation()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\create_collation.html)
* [sqlite3\_create\_function()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\create_function.html)
* [sqlite3\_create\_module()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\create_module.html)
* [sqlite3\_vfs\_register()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\vfs_find.html)

The [sqlite3\_create\_collation()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\create_collation.html) interface is used to create new [collating sequences](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\datatype3.html#collation) for sorting text. The [sqlite3\_create\_module()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\create_module.html) interface is used to register new [virtual table](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\vtab.html) implementations. The [sqlite3\_vfs\_register()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\vfs_find.html) interface creates new [VFSes](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\vfs.html).

The [sqlite3\_create\_function()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\create_function.html) interface creates new SQL functions - either scalar or aggregate. The new function implementation typically makes use of the following additional interfaces:

* [sqlite3\_aggregate\_context()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\aggregate_context.html)
* [sqlite3\_result()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\result_blob.html)
* [sqlite3\_user\_data()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\user_data.html)
* [sqlite3\_value()](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\value_blob.html)

All of the built-in SQL functions of SQLite are created using exactly these same interfaces. Refer to the SQLite source code, and in particular the [date.c](http://www.sqlite.org/src/doc/trunk/src/date.c) and [func.c](http://www.sqlite.org/src/doc/trunk/src/func.c) source files for examples.

Shared libraries or DLLs can be used as [loadable extensions](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\loadext.html) to SQLite.

## Other Interfaces

This article only mentions the foundational SQLite interfaces. The SQLite library includes many other APIs implementing useful features that are not described here. A [complete list of functions](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\funclist.html) that form the SQLite application programming interface is found at the [C/C++ Interface Specification](file:///D:\devdata\sqlite\SQLite%203.8.10.2\sqlite-doc-3081002\c3ref\intro.html). Refer to that document for complete and authoritative information about all SQLite interfaces.

# NOTE

## VS2008下的调试

VS2008不支持超过32768行源代码的调试（由于pdb文件行号长度限制），无法调试单一sqlite.c文件。之前sqlite提供了分割的文件如sqlite-amalgamation32k-201408060029.zip，将单一.c文件分割为多个.c文件，避免该问题。似乎当前版本（3.10.2）没有提供分割的版本。如需调试，可升级至新版VS。

## 列信息/列名获取

### 使用SQL查询语句

通过SQL查询语句 pragma table\_info(table\_name)查询table\_name的表结构，获取列名等信息。推荐使用该方法。

### 使用内部函数sqlite3FindTable

不建议使用该方式。因为该函数声明为SQLITE\_PRIVATE（即static，仅在sqlite.c文件中能够使用），SQLITE\_PRIVATE函数的直接调用可能带来未知的问题。

* 直接调用（不建议）

可以在SQLITE.C中将其声明和定义更改为SQLITE\_API，并在调用前添加如下声明：

extern "C"

{

struct Table;

Table \*sqlite3FindTable(sqlite3\*,const char\*, const char\*);

};

* 间接调用

在SQLITE.C中新增一个SQLITE\_API类型的函数，完成查询并返回结果，外部调用该函数。

* 注意事项

打开数据库后，直接调用sqlite3FindTable返回结果为NULL，需要首先进行一次查询才可以获取到Table信息：

char\* szSqlSelTable = "select name from sqlite\_master where type = 'table'";

nRet = sqlite3\_exec(pdb, szSqlSelTable, 0, 0, 0);

if (nRet != SQLITE\_OK)

{

fprintf(stderr, "szSqlSelTable error\n");

}

Table \*pTab;

pTab = sqlite3FindTable(pdb, "memberu", 0);