SQLite Tutorial

- SQLite - Home
- SQLite - Overview
- SQLite - Installation
- SQLite - Commands
- SQLite - Syntax
- SQLite - Data Types
- SQLite - Create Database
- SQLite - Attach Database
- SQLite - Detach Database
- SQLite - Create Table
- SQLite - Drop Table
- SQLite - Insert Query
- SQLite - Select Query
- SQLite - Operators
- SQLite - Expressions
- SQLite - Where Clause
- SQLite - AND & OR Clauses
- SQLite - Update Query
- SQLite - Delete Query
- SQLite - Like Clause
- SQLite - Glob Clause
- SQLite - Limit Clause
- SQLite - Order By

- SQLite - Group By
- SQLite - Having Clause
- SQLite - Distinct Keyword

Advanced SQLite

- SQLite - PRAGMA
- SQLite - Constraints
- SQLite - Joins
- SQLite - Unions Clause
- SQLite - NULL Values
- SQLite - Alias Syntax
- SQLite - Triggers
- SQLite - Indexes
- SQLite - Indexed By
- SQLite - Alter Command
- SQLite - Truncate Table
- SQLite - Views
- SQLite - Transactions
- SQLite - Sub Queries
- SQLite - Autoincrement
- SQLite - Injection
- SQLite - Explain
- SQLite - Vacuum
- SQLite - Date & Time
- SQLite - Useful Functions

SQLite Interfaces

- SQLite - C/C++

SQLite - Java
SQLite - PHP
SQLite - Perl
SQLite - Python

SQL Useful Resources

SQLite - Quick Guide
SQLite - Useful Resources
SQLite - Discussion

SQLite C/C++ Tutorial

Previous Page Next Page

Installation

Before we start using SQLite in our C/C++ programs, we need to make sure that we have SQLite library set up on the machine. You can check SQLite Installation chapter to understand installation process.

C/C++ Interface APIs

Following are important C&C++ / SQLite interface routines which can suffice your requirement to work with SQLite database from your C/C++ program. If you are looking for a more sophisticated application, then you can look into SQLite official documentation.

S.N.	API & Description
1	**sqlite3_open(const char *filename, sqlite3 **ppDb)** This routine opens a connection to an SQLite database file and returns a database connection object to be used by other SQLite routines. If the *filename* argument is NULL or ':memory:', sqlite3_open() will create an in-memory database in RAM that lasts only for the duration of the session. If filename is not NULL, sqlite3_open() attempts to open the database file by using its value. If no file by that name exists, sqlite3_open() will open a new database file by that name.
2	**sqlite3_exec(sqlite3*, const char *sql, sqlite_callback, void *data, char **errmsg)** This routine provides a quick, easy way to execute SQL commands provided by sql argument which can consist of more than one SQL command. Here, first argument *sqlite3* is open database object, *sqlite_callback* is a call back for which *data* is the 1st argument and errmsg will be return to capture any error raised by the routine. The sqlite3_exec() routine parses and executes every command given in the **sql** argument until it reaches the end of the string or encounters an error.
3	**sqlite3_close(sqlite3*)** This routine closes a database connection previously opened by a call to sqlite3_open(). All prepared statements associated with the connection should be finalized prior to closing the connection. If any queries remain that have not been finalized, sqlite3_close() will return SQLITE_BUSY with the error message Unable to close due to unfinalized statements.

Connecting To Database

Following C code segment shows how to connect to an existing database. If database does not exist, then it will be created and finally a database object will be returned.

```c
#include <stdio.h>
#include <sqlite3.h>

int main(int argc, char* argv[])
{
   sqlite3 *db;
   char *zErrMsg = 0;
   int rc;

   rc = sqlite3_open("test.db", &db);

   if( rc ){
      fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
      return(0);
   }else{
      fprintf(stderr, "Opened database successfully\n");
   }
   sqlite3_close(db);
}
```

Now, let's compile and run above program to create our database **test.db** in the current directory. You can change your path as per your requirement.

```
$gcc test.c -l sqlite3
$./a.out
Opened database successfully
```

If you are going to use C++ source code, then you can compile your code as follows:

```
$g++ test.c -l sqlite3
```

Here we are linking our program with sqlite3 library to provide required functions to C program. This will create a database file test.db in your directory and you will have the result something as follows:

```
-rwxr-xr-x. 1 root root 7383 May  8 02:06 a.out
-rw-r--r--. 1 root root  323 May  8 02:05 test.c
-rw-r--r--. 1 root root    0 May  8 02:06 test.db
```

Create a Table

Following C code segment will be used to create a table in previously created database:

```c
#include <stdio.h>
#include <stdlib.h>
#include <sqlite3.h>

static int callback(void *NotUsed, int argc, char **argv, char **azColName){
   int i;
   for(i=0; i<argc; i++){
      printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");
```

```
   }
   printf("\n");
   return 0;
}

int main(int argc, char* argv[])
{
   sqlite3 *db;
   char *zErrMsg = 0;
   int  rc;
   char *sql;

   /* Open database */
   rc = sqlite3_open("test.db", &db);
   if( rc ){
      fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
      return(0);
   }else{
      fprintf(stdout, "Opened database successfully\n");
   }

   /* Create SQL statement */
   sql = "CREATE TABLE COMPANY("  \
         "ID INT PRIMARY KEY     NOT NULL," \
         "NAME           TEXT    NOT NULL," \
         "AGE            INT     NOT NULL," \
         "ADDRESS        CHAR(50)," \
         "SALARY         REAL );";

   /* Execute SQL statement */
   rc = sqlite3_exec(db, sql, callback, 0, &zErrMsg);
   if( rc != SQLITE_OK ){
   fprintf(stderr, "SQL error: %s\n", zErrMsg);
      sqlite3_free(zErrMsg);
   }else{
      fprintf(stdout, "Table created successfully\n");
   }
   sqlite3_close(db);
   return 0;
}
```

When above program is compiled and executed, it will create COMPANY table in your test.db and final listing of the file will be as follows:

```
-rwxr-xr-x. 1 root root 9567 May  8 02:31 a.out
-rw-r--r--. 1 root root 1207 May  8 02:31 test.c
-rw-r--r--. 1 root root 3072 May  8 02:31 test.db
```

INSERT Operation

Following C code segment shows how we can create records in our COMPANY table created in above example:

```c
#include <stdio.h>
#include <stdlib.h>
#include <sqlite3.h>

static int callback(void *NotUsed, int argc, char **argv, char **azColName){
   int i;
   for(i=0; i<argc; i++){
      printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");
   }
   printf("\n");
   return 0;
}

int main(int argc, char* argv[])
{
   sqlite3 *db;
   char *zErrMsg = 0;
   int rc;
   char *sql;

   /* Open database */
   rc = sqlite3_open("test.db", &db);
   if( rc ){
      fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
      return(0);
   }else{
      fprintf(stderr, "Opened database successfully\n");
   }

   /* Create SQL statement */
   sql = "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) "  \
         "VALUES (1, 'Paul', 32, 'California', 20000.00 ); " \
         "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) "  \
         "VALUES (2, 'Allen', 25, 'Texas', 15000.00 ); "     \
         "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)" \
         "VALUES (3, 'Teddy', 23, 'Norway', 20000.00 );" \
         "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)" \
         "VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 );";

   /* Execute SQL statement */
   rc = sqlite3_exec(db, sql, callback, 0, &zErrMsg);
   if( rc != SQLITE_OK ){
      fprintf(stderr, "SQL error: %s\n", zErrMsg);
      sqlite3_free(zErrMsg);
   }else{
      fprintf(stdout, "Records created successfully\n");
   }
   sqlite3_close(db);
   return 0;
}
```

When above program is compiled and executed, it will create given records in COMPANY table and will display following two line:

```
Opened database successfully
```

```
Records created successfully
```

SELECT Operation

Before we proceed with actual example to fetch records, let me give a little detail about the callback function, which we are using in our examples. This callback provides a way to obtain results from SELECT statements. It has the following declaration:

```
typedef int (*sqlite3_callback)(
void*,    /* Data provided in the 4th argument of sqlite3_exec() */
int,      /* The number of columns in row */
char**,   /* An array of strings representing fields in the row */
char**    /* An array of strings representing column names */
);
```

If above callback is provided in sqlite_exec() routine as the third argument, SQLite will call the this callback function for each record processed in each SELECT statement executed within the SQL argument.

Following C code segment shows how we can fetch and display records from our COMPANY table created in above example:

```
#include <stdio.h>
#include <sqlite3.h>

static int callback(void *data, int argc, char **argv, char **azColName){
   int i;
   fprintf(stderr, "%s: ", (const char*)data);
   for(i=0; i<argc; i++){
      printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");
   }
   printf("\n");
   return 0;
}

int main(int argc, char* argv[])
{
   sqlite3 *db;
   char *zErrMsg = 0;
   int rc;
   char *sql;
   const char* data = "Callback function called";

   /* Open database */
   rc = sqlite3_open("test.db", &db);
   if( rc ){
      fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
      return(0);
   }else{
      fprintf(stderr, "Opened database successfully\n");
```

```
   }

   /* Create SQL statement */
   sql = "SELECT * from COMPANY";

   /* Execute SQL statement */
   rc = sqlite3_exec(db, sql, callback, (void*)data, &zErrMsg);
   if( rc != SQLITE_OK ){
      fprintf(stderr, "SQL error: %s\n", zErrMsg);
      sqlite3_free(zErrMsg);
   }else{
      fprintf(stdout, "Operation done successfully\n");
   }
   sqlite3_close(db);
   return 0;
}
```

When above program is compiled and executed, it will produce the following result:

```
Opened database successfully
Callback function called: ID = 1
NAME = Paul
AGE = 32
ADDRESS = California
SALARY = 20000.0

Callback function called: ID = 2
NAME = Allen
AGE = 25
ADDRESS = Texas
SALARY = 15000.0

Callback function called: ID = 3
NAME = Teddy
AGE = 23
ADDRESS = Norway
SALARY = 20000.0

Callback function called: ID = 4
NAME = Mark
AGE = 25
ADDRESS = Rich-Mond
SALARY = 65000.0

Operation done successfully
```

UPDATE Operation

Following C code segment shows how we can use UPDATE statement to update any record and then fetch and display updated records from our COMPANY table:

```
#include <stdio.h>
#include <stdlib.h>
#include <sqlite3.h>
```

```
static int callback(void *data, int argc, char **argv, char **azColName){
   int i;
   fprintf(stderr, "%s: ", (const char*)data);
   for(i=0; i<argc; i++){
      printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");
   }
   printf("\n");
   return 0;
}

int main(int argc, char* argv[])
{
   sqlite3 *db;
   char *zErrMsg = 0;
   int rc;
   char *sql;
   const char* data = "Callback function called";

   /* Open database */
   rc = sqlite3_open("test.db", &db);
   if( rc ){
      fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
      return(0);
   }else{
      fprintf(stderr, "Opened database successfully\n");
   }

   /* Create merged SQL statement */
   sql = "UPDATE COMPANY set SALARY = 25000.00 where ID=1; " \
         "SELECT * from COMPANY";

   /* Execute SQL statement */
   rc = sqlite3_exec(db, sql, callback, (void*)data, &zErrMsg);
   if( rc != SQLITE_OK ){
      fprintf(stderr, "SQL error: %s\n", zErrMsg);
      sqlite3_free(zErrMsg);
   }else{
      fprintf(stdout, "Operation done successfully\n");
   }
   sqlite3_close(db);
   return 0;
}
```

When above program is compiled and executed, it will produce the following result:

```
Opened database successfully
Callback function called: ID = 1
NAME = Paul
AGE = 32
ADDRESS = California
SALARY = 25000.0

Callback function called: ID = 2
NAME = Allen
AGE = 25
```

```
ADDRESS = Texas
SALARY = 15000.0

Callback function called: ID = 3
NAME = Teddy
AGE = 23
ADDRESS = Norway
SALARY = 20000.0

Callback function called: ID = 4
NAME = Mark
AGE = 25
ADDRESS = Rich-Mond
SALARY = 65000.0

Operation done successfully
```

DELETE Operation

Following C code segment shows how we can use DELETE statement to delete any record and then fetch and display remaining records from our COMPANY table:

```c
#include <stdio.h>
#include <stdlib.h>
#include <sqlite3.h>

static int callback(void *data, int argc, char **argv, char **azColName){
   int i;
   fprintf(stderr, "%s: ", (const char*)data);
   for(i=0; i<argc; i++){
      printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");
   }
   printf("\n");
   return 0;
}

int main(int argc, char* argv[])
{
   sqlite3 *db;
   char *zErrMsg = 0;
   int rc;
   char *sql;
   const char* data = "Callback function called";

   /* Open database */
   rc = sqlite3_open("test.db", &db);
   if( rc ){
      fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
      return(0);
   }else{
      fprintf(stderr, "Opened database successfully\n");
   }

   /* Create merged SQL statement */
```

```
   sql = "DELETE from COMPANY where ID=2; " \
         "SELECT * from COMPANY";

   /* Execute SQL statement */
   rc = sqlite3_exec(db, sql, callback, (void*)data, &zErrMsg);
   if( rc != SQLITE_OK ){
      fprintf(stderr, "SQL error: %s\n", zErrMsg);
      sqlite3_free(zErrMsg);
   }else{
      fprintf(stdout, "Operation done successfully\n");
   }
   sqlite3_close(db);
   return 0;
}
```

When above program is compiled and executed, it will produce the following result:

```
Opened database successfully
Callback function called: ID = 1
NAME = Paul
AGE = 32
ADDRESS = California
SALARY = 20000.0

Callback function called: ID = 3
NAME = Teddy
AGE = 23
ADDRESS = Norway
SALARY = 20000.0

Callback function called: ID = 4
NAME = Mark
AGE = 25
ADDRESS = Rich-Mond
SALARY = 65000.0

Operation done successfully
```

Previous Page Next Page

Enter email for newslet go