Search notes:

# SQLite: basic demonstration of the c interface

This is an attempt to demonstrate the basic functionality to use an SQLite database in a C program.

## Creating a table

`sqlite3_open()` opens or, if it does not exist, creates an SQLite database. We assume that `the.db` does not yet exist and thus, `sqlite3_open()` will create it.

After creating the database, `sqlite3_exec()` is given a `create table` SQL statement to be executed in the new database.
`sqlite3_exec()` performs the three API cals `sqlite3_prepare_v2()`, `sqlite3_step()` and `sqlite3_finalize()` and is thus a convenience wrapper for these.

Finally, the database is closed with `sqlite3_close()`.

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

int main() {

  sqlite3 *db;
  if (sqlite3_open("the.db", &db)) {
    printf("Could not open the.db\n");
    exit(-1);
  }

  if (sqlite3_exec(db, "create table tab(foo, bar, baz)", NULL, NULL, NULL)) {
     printf("Error executing sql statement\n");
  }
  else {
    printf("Table created\n");
  }

  sqlite3_close(db);
}
```

*Github respository about-sqlite-c-interface, path: /basic/00_create-table.c*

## Inserting some values

In this program, we use `sqlite3_exec()` again to execute two insert statements.

Note: the insert statements can be executed in *one* call by separating them with a semicolon (;):

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

int main() {

  sqlite3 *db;
  if (sqlite3_open("the.db", &db)) {
    printf("Could not open the.db\n");
    exit(-1);
  }
```

```
  if (sqlite3_exec(db,
        "insert into tab values(1, 'one', null);"
        "insert into tab values(2,  2.2 ,'two');",
        NULL, NULL, NULL)) {

     printf("Error executing sql statement\n");
  }
  else {
    printf("records inserted\n");
  }

  sqlite3_close(db);
}
```

*Github respository about-sqlite-c-interface, path: /basic/01_insert.c*

# Using bind variables

When inserting many records, bind variables allow to only prepare a statement once and the execute it multiple times.

The statement is prepared with `sqlite3_prepare_v2()` and executed with `sqlite3_step()`.

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

int main() {

    sqlite3 *db;
    if (sqlite3_open("the.db", &db)) {
      printf("Could not open the.db\n");
      exit(-1);
    }

 //
 // Prepare a statement for multiple use:
 //
    sqlite3_stmt *stmt;
    if (sqlite3_prepare_v2(db, "insert into tab values(?, ?, ?)", -1, &stmt, NULL)) {
       printf("Error executing sql statement\n");
       sqlite3_close(db);
       exit(-1);
    }

 //
 // Bind the values for the first insert:
 //
    sqlite3_bind_int (stmt, 1, 3      );
    sqlite3_bind_text(stmt, 2, "three", -1, NULL);
    sqlite3_bind_int (stmt, 3,  333   );

 //
 // Do the first insert:
 //
    sqlite3_step(stmt);

 //
 // Reset the prepared statement to the initial state.
 // This seems to be necessary in order to
 // use the prepared statement again for another
 // insert:
 //
    sqlite3_reset(stmt);

 //
 // Bind the values for the second insert
```

```
 //
    sqlite3_bind_int (stmt, 1, 4      );
    sqlite3_bind_text(stmt, 2, "four" , -1, NULL);
    sqlite3_bind_null(stmt, 3);

 //
 // To the second insert
 //
    sqlite3_step(stmt);

 //
 // Get rid of the memory allocated for stmt:
 //
    sqlite3_finalize(stmt);

    sqlite3_close(db);
}
```

*Github respository about-sqlite-c-interface, path: /basic/02_insert-bind.c*

# Selecting from the table

Since SQLite is not strongly typed (but rather *manifest* typed), each value stored in a row/column can have its own datatype.
Thus, when selecting and printing the values, we first have to determine a value's datatype with `sqlite3_column_type()`.
After determining the datatype, we can query the value with one of the `sqlite3_column_XXX()` functions.

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

void printColumnValue(sqlite3_stmt* stmt, int col) {

  int colType = sqlite3_column_type(stmt, col);

  switch(colType) {

    case SQLITE_INTEGER:
         printf("  %3d   ", sqlite3_column_int(stmt, col));
         break;

    case SQLITE_FLOAT:
         printf("  %5.2f", sqlite3_column_double(stmt, col));
         break;

    case SQLITE_TEXT:
         printf("  %-5s", sqlite3_column_text(stmt, col));
         break;

    case SQLITE_NULL:
         printf("  null");
         break;

    case SQLITE_BLOB:
         printf("  blob");
         break;
    }

}

int main() {

    sqlite3 *db;
    if (sqlite3_open("the.db", &db)) {
```

```
      printf("Could not open the.db\n");
      exit(-1);
    }

    sqlite3_stmt *stmt;
    if (sqlite3_prepare_v2(db, "select * from tab where foo >= ? order by foo", -1, &stmt, NULL)
       printf("Error executing sql statement\n");
       sqlite3_close(db);
       exit(-1);
    }

  //
  // Bind the only bindable value in the select stament (foo >= ?)
  //
    sqlite3_bind_int (stmt, 1, 2);

    while (sqlite3_step(stmt) != SQLITE_DONE) {
     //
     // Print the column values for the current record.
     // Note: the column values are 0-indexed while the
     // bind values are 1-indexed.
     //
        for (int col=0; col<=2; col++) {
          printColumnValue(stmt, col);
        }
        printf("\n");
    }

    sqlite3_finalize(stmt);

    sqlite3_close(db);
}
```

*Github respository about-sqlite-c-interface, path: /basic/03_select.c*

## See also

SQLite: C interface

Connect to SQLite from VBA using winsqlite3.dll

## Links

These files are in this github repository.

---

Index