More SQLite programming with C

CMPUT 391

The reading you need to do

Point your browser to: https://www.sqlite.org/cintro.html

Make sure you bookmark that page and read, today:

- Binding Parameters and Reusing Prepared Statements
- Extending SQLite

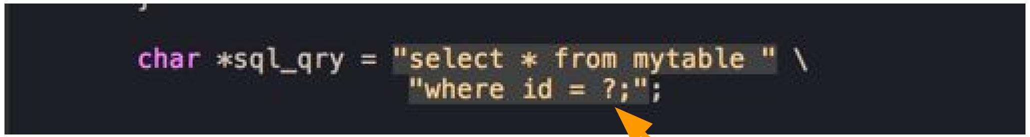
We suggest you also read the section on

- Configuring SQLite

Parameterized SQL statements

What to do if the actual query depends on user input?

Example:



```
char *sql_qry = "select * from mytable " \
                 "where id = ?;";
```

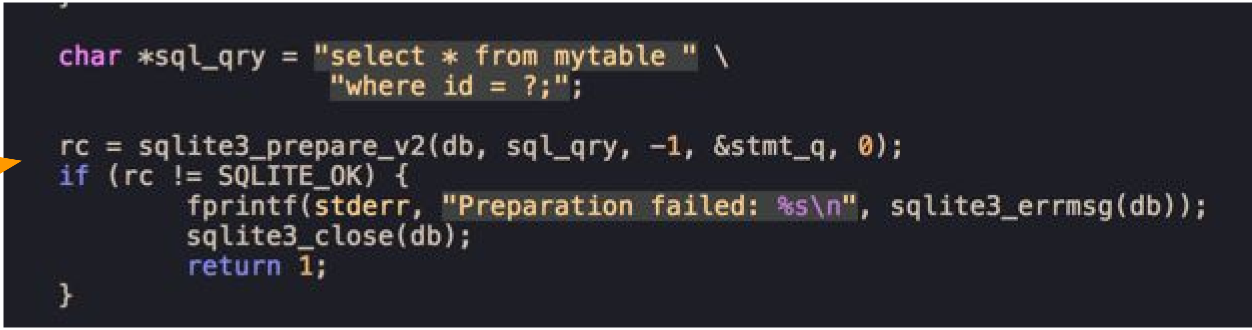
? is the parameter

Note: this is part of the SQL standard

→ most DBMSs, including SQLite support it

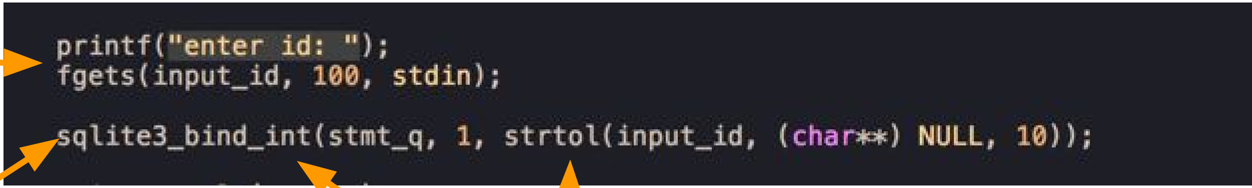
Read up https://www.sqlite.org/c3ref/bind_blob.html

Binding parameters to a SQL query



```
char *sql_qry = "select * from mytable " \
                 "where id = ?;";

rc = sqlite3_prepare_v2(db, sql_qry, -1, &stmt_q, 0);
if (rc != SQLITE_OK) {
        fprintf(stderr, "Preparation failed: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return 1;
}
```

prepare the statement as usual



```
printf("enter id: ");
fgets(input_id, 100, stdin);

sqlite3_bind_int(stmt_q, 1, strtol(input_id, (char**) NULL, 10));
```

read the input from STDIN

bind the input to the SQL parameter

convert the parameter to an **int**

Re-using a prepared statement

Preparing a statement means compiling it into SQLite3 byte-code (and takes time)

If you need to issue the same statement many times:

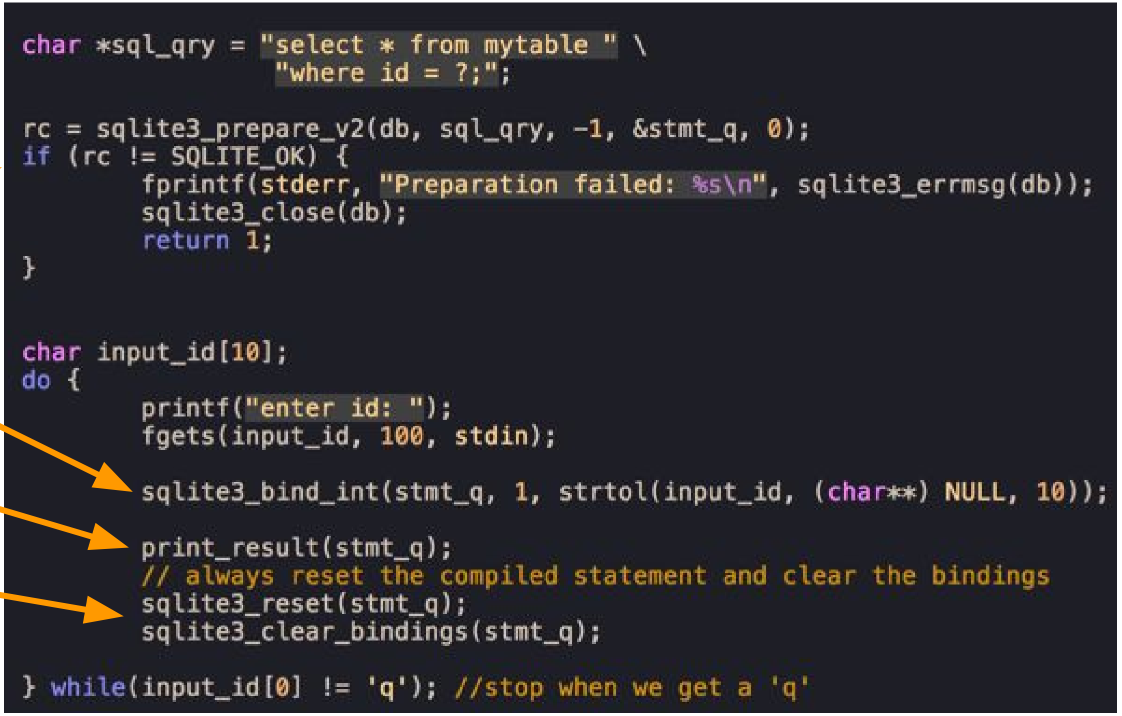
prepare it **once**

call many times

bind parameters

execute

reset bindings



```
char *sql_qry = "select * from mytable " \
                 "where id = ?;";

rc = sqlite3_prepare_v2(db, sql_qry, -1, &stmt_q, 0);
if (rc != SQLITE_OK) {
        fprintf(stderr, "Preparation failed: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return 1;
}


char input_id[10];
do {
        printf("enter id: ");
        fgets(input_id, 100, stdin);

        sqlite3_bind_int(stmt_q, 1, strtol(input_id, (char**) NULL, 10));

        print_result(stmt_q);
        // always reset the compiled statement and clear the bindings
        sqlite3_reset(stmt_q);
        sqlite3_clear_bindings(stmt_q);

} while(input_id[0] != 'q'); //stop when we get a 'q'
```

Read carefully

Among other important things, https://www.sqlite.org/c3ref/bind_blob.html states:

“The second argument is the index of the SQL parameter to be set. The leftmost SQL parameter has an index of 1.”

Also read:

https://www.sqlite.org/c3ref/clear_bindings.html

https://www.sqlite.org/c3ref/reset.html

Using custom functions in SQL statements

SQL has a very limited list of built-in functions

Most DBMSs offer ways for you to add custom functions to be used in SQL statements

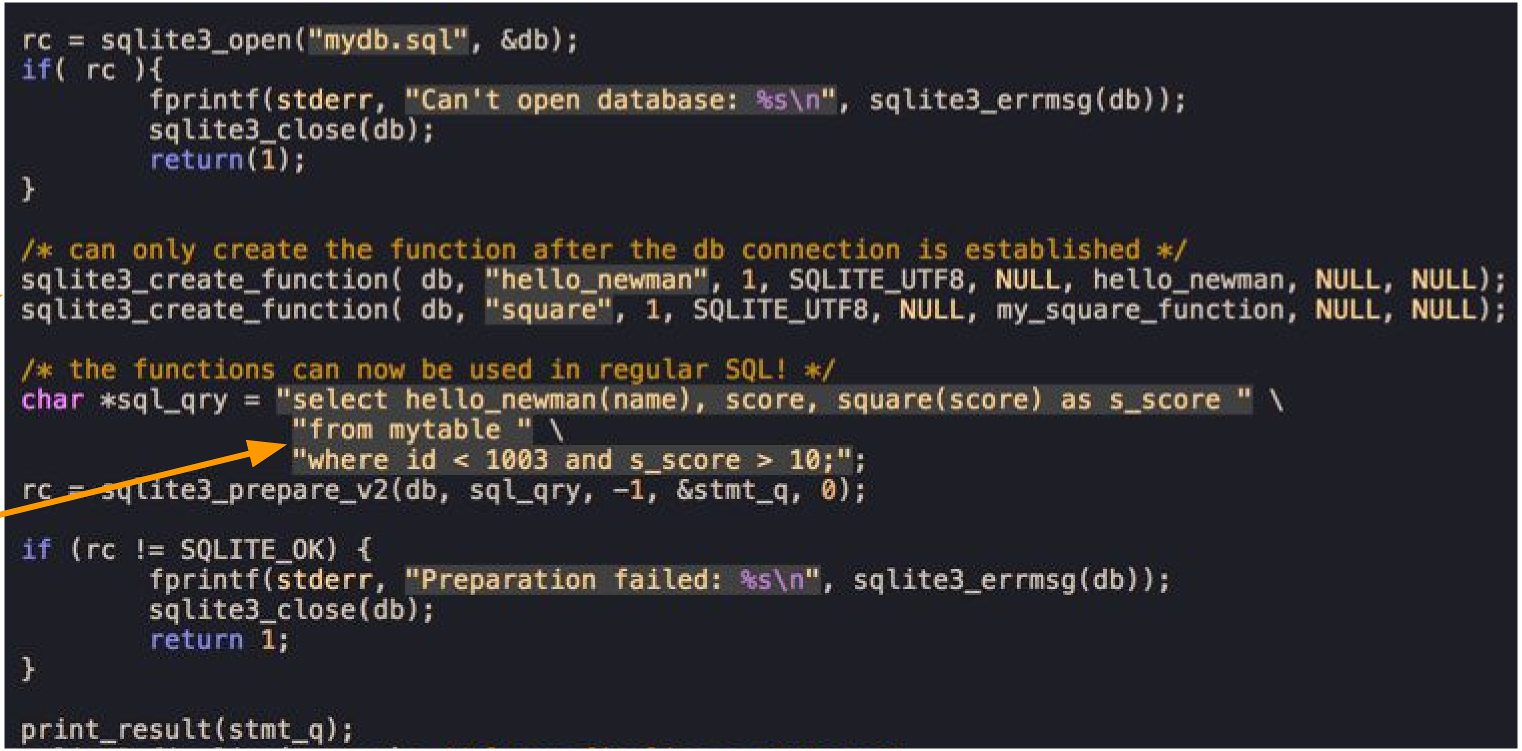
SQLite does not support the official SQL programming standard :(

But it allows the same functionality

Examples of scalar functions

declare the functions

use the functions in SQL

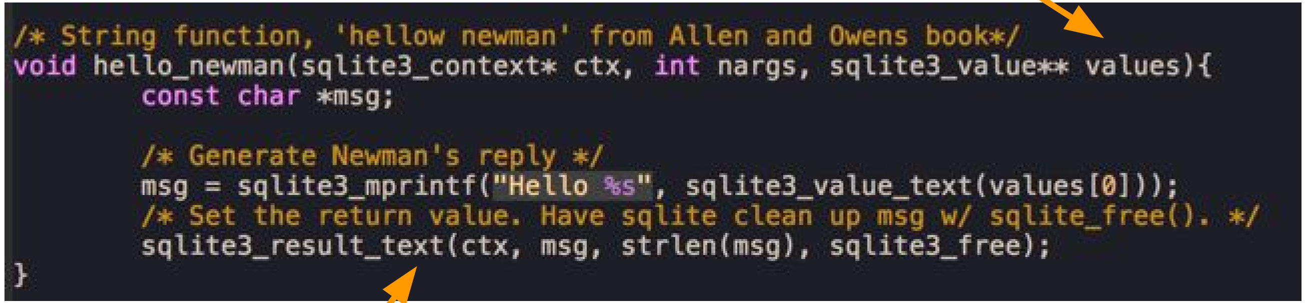


```
rc = sqlite3_open("mydb.sql", &db);
if( rc ){
        fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return(1);
}

/* can only create the function after the db connection is established */
sqlite3_create_function( db, "hello_newman", 1, SQLITE_UTF8, NULL, hello_newman, NULL, NULL);
sqlite3_create_function( db, "square", 1, SQLITE_UTF8, NULL, my_square_function, NULL, NULL);

/* the functions can now be used in regular SQL! */
char *sql_qry = "select hello_newman(name), score, square(score) as s_score " \
                 "from mytable " \
                 "where id < 1003 and s_score > 10;";
rc = sqlite3_prepare_v2(db, sql_qry, -1, &stmt_q, 0);

if (rc != SQLITE_OK) {
        fprintf(stderr, "Preparation failed: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return 1;
}

print_result(stmt_q);
```

The function implementation

Always the same signature

argument array

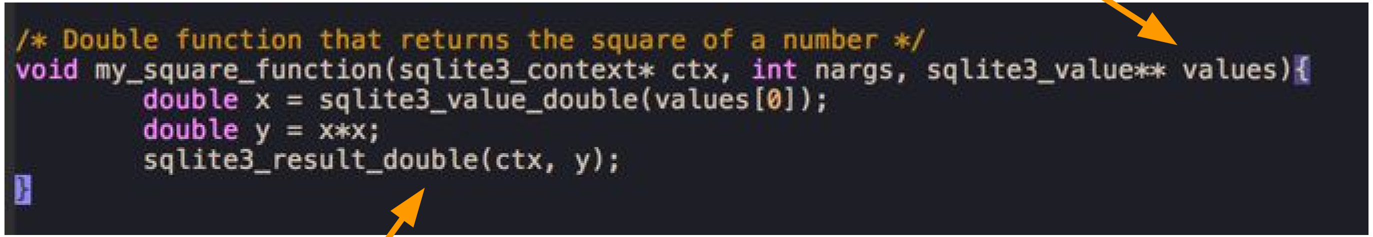


```c
/* String function, 'hellow newman' from Allen and Owens book*/
void hello_newman(sqlite3_context* ctx, int nargs, sqlite3_value** values){
        const char *msg;

        /* Generate Newman's reply */
        msg = sqlite3_mprintf("Hello %s", sqlite3_value_text(values[0]));
        /* Set the return value. Have sqlite clean up msg w/ sqlite_free(). */
        sqlite3_result_text(ctx, msg, strlen(msg), sqlite3_free);
}
```

result type

The function implementation

Always the same signature



```
/* Double function that returns the square of a number */
void my_square_function(sqlite3_context* ctx, int nargs, sqlite3_value** values){
        double x = sqlite3_value_double(values[0]);
        double y = x*x;
        sqlite3_result_double(ctx, y);
}
```

argument array

result type

Read carefully

Among other important things https://www.sqlite.org/c3ref/create_function.html expains how to create **aggregate** functions as well.

Sample code

Parameterized SQL

https://webdocs.cs.ualberta.ca/~denilson/teaching/cmput391/parameterized_sql.c

Sample functions

https://webdocs.cs.ualberta.ca/~denilson/teaching/cmput391/sample_functions.c