### 使用libsqlite

sudo apt-get install libsqlite3-dev

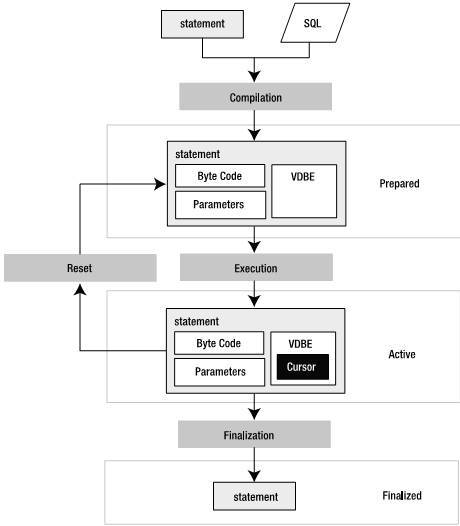
下面代码例示了在SQLite上执行一个query的一般过程。

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include"sqlite3.h"  #include<string.h>  int main(int argc,char \*\*argv)  {  // 校验参数有效性  char \*dbname = argv[1];  char \*tablename = argv[2];  if (argc < 3) {  printf("Error: Must specific a db file and select table name\n");  printf("\nUsage:\n\t%s file.db table\_name\n", argv[0]);  exit(1);  }  printf("db file %s:SELECT \* FROM %s\n", dbname, tablename);  int rc, i, ncols;  sqlite3 \*db;  sqlite3\_stmt \*stmt;  char \*sql;  const char\*tail;  // 打开数据  rc = sqlite3\_open(dbname, &db);  if(rc) {  fprintf(stderr,"Can'topendatabase:%sn",sqlite3\_errmsg(db));  sqlite3\_close(db);  exit(1);  }  // 构造执行语句  sql = malloc(strlen("SELECT \* FROM ") + strlen(tablename));  if (sql == NULL) {  sqlite3\_close(db);  exit(1);  }  sprintf(sql, "SELECT \* FROM %s", tablename);  // 预处理  rc = sqlite3\_prepare(db, sql, (int)strlen(sql), &stmt, &tail);  if (rc != SQLITE\_OK){  fprintf(stderr, "SQLerror:%s\n", sqlite3\_errmsg(db));  }  rc = sqlite3\_step(stmt);  ncols = sqlite3\_column\_count(stmt);  while (rc == SQLITE\_ROW){  for (i=0; i<ncols; i++){  fprintf(stderr, "'%s'", sqlite3\_column\_text(stmt, i));  }  fprintf(stderr, "\n");  rc = sqlite3\_step(stmt);  }  // 释放statement  sqlite3\_finalize(stmt);  // 关闭数据库  sqlite3\_close(db);  printf("\n");  return(0);  } |
|  |

编译

|  |
| --- |
| gcc -o executeselect executeselect.c `pkg-config sqlite3 --cflags --libs` |

一个SQL语句在SQLite中的执行过程如下：



预处理查询(Prepared Query)是SQLite执行所有SQL命令的方式，包括下面三个步骤：

(1) 准备(preparation)：

分词器(tokenizer) 、分析器(parser)和代码生成器(code generator)把SQL语句编译成VDBE字节码，编译器会创建一个语句句柄(sqlite3\_stmt)，它包括字节码以及其它执行命令和遍历结果集所需的全部资源。相应的C API为sqlite3\_prepare()，位于prepare.c文件中。

(2) 执行(execution)：

虚拟机执行字节码，执行过程是一个步进(stepwise)的过程，每一步(step)由sqlite3\_step()启动，并由VDBE执行一段字节码。当第一次调用sqlite3\_step()时，一般会获得一种锁，锁的种类由命令要做什么(读或写)决定。对于SELECT语句，每次调用sqlite3\_step()使用语句句柄的游标移到结果集的下一行。对于结果集中的每一行，它返回SQLITE\_ROW，当到达结果末尾时，返回SQLITE\_DONE。对于其它SQL语句(INSERT、UPDATE、DELETE等)，第一次调用sqlite3\_step()就导致VDBE执行整个命令。

(3) 定案(finalization)：

VDBE关闭语句，释放资源。相应的C API为sqlite3\_finalize()，它导致VDBE结束程序运行并关闭语句句柄。如果事务是由人工控制开始的，它必须由人工控制进行提交或回卷，否则sqlite3\_finalize()会返回一个错误。当sqlite3\_finalize()执行成功，所有与语句对象关联的资源都将被释放。在自动提交模式下，还会释放关联的数据库锁。

每一步(preparation、execution和finalization)都关联于语句句柄的一种状态(prepared、active和finalized)。Pepared表示所有资源都已分配，语句已经可以执行，但还没有执行。现在还没有申请锁，一直到调用sqlite3\_step()时才会申请锁。Active状态开始于对sqlite3\_step()的调用，此时语句正在被执行并拥有某种锁。Finalized意味着语句已经被关闭且所有相关资源已经被释放。

### 编译sqlite源码

|  |
| --- |
| root@NJVV-11048100:~/work/mycode/sqlite# ./configure  root@NJVV-11048100:~/work/mycode/sqlite# make |

./configure 配置选项

--enable-shared 编译会生成动态库，默认为yes

生成的静态库libsqlite3.a和动态库libsqlite3.so在隐藏目录.libs目录下。

|  |
| --- |
| root@NJVV-11048100:~/work/mycode/sqlite# ls -alh .libs/  total 22M  drwxrwxrwx 0 root root 4.0K Oct 26 11:37 .  drwxrwxrwx 0 root root 4.0K Oct 26 11:37 ..  -rwxrwxrwx 1 root root 8.9M Oct 26 11:37 libsqlite3.a  lrwxrwxrwx 1 root root 16 Oct 26 11:37 libsqlite3.la -> ../libsqlite3.la  -rwxrwxrwx 1 root root 959 Oct 26 11:37 libsqlite3.lai  lrwxrwxrwx 1 root root 19 Oct 26 11:37 libsqlite3.so -> libsqlite3.so.0.8.6  lrwxrwxrwx 1 root root 19 Oct 26 11:37 libsqlite3.so.0 -> libsqlite3.so.0.8.6  -rwxrwxrwx 1 root root 3.7M Oct 26 11:37 libsqlite3.so.0.8.6  -rwxrwxrwx 1 root root 8.6M Oct 26 11:36 sqlite3.o |

用编译的源码库

|  |
| --- |
| gcc -o executeselect executeselect.c -L/root/work/mycode/sqlite/.libs/ -lsqlite3 -I /root/work/mycode/sqlite/ |

启动gdb调试

|  |
| --- |
| LD\_LIBRARY\_PATH=/mnt/f/workdir/mycode/sqlite/.libs/ gdb ./executeselect |

|  |
| --- |
| 设置断点  0x01 查看全局和静态变量  info variables  1  0x02 查看当前stack frame局部变量  info locals  1  0x03 查看当前stack frame参数  info args |

tui enable->