第五章、数据库

第一节、数据库简述：

1.1、认识数据库：

数据库（Database）是一个有组织的数据集合，用于存储、管理、检索和更新数据。它可以包含多个数据表，每个数据表都包含多个行和列，用于存储不同类型的数据。数据库还可以提供一些高级功能，如数据备份和恢复、安全性、事务管理和查询优化，以满足各种应用程序的需求。

数据库可以被应用在各种领域中，如企业管理、科学研究、社交网络、金融服务和电子商务等。常见的数据库软件包括SQLite、MySQL、Oracle、Microsoft SQL Server、PostgreSQL和MongoDB等。数据库按数据模型分类：

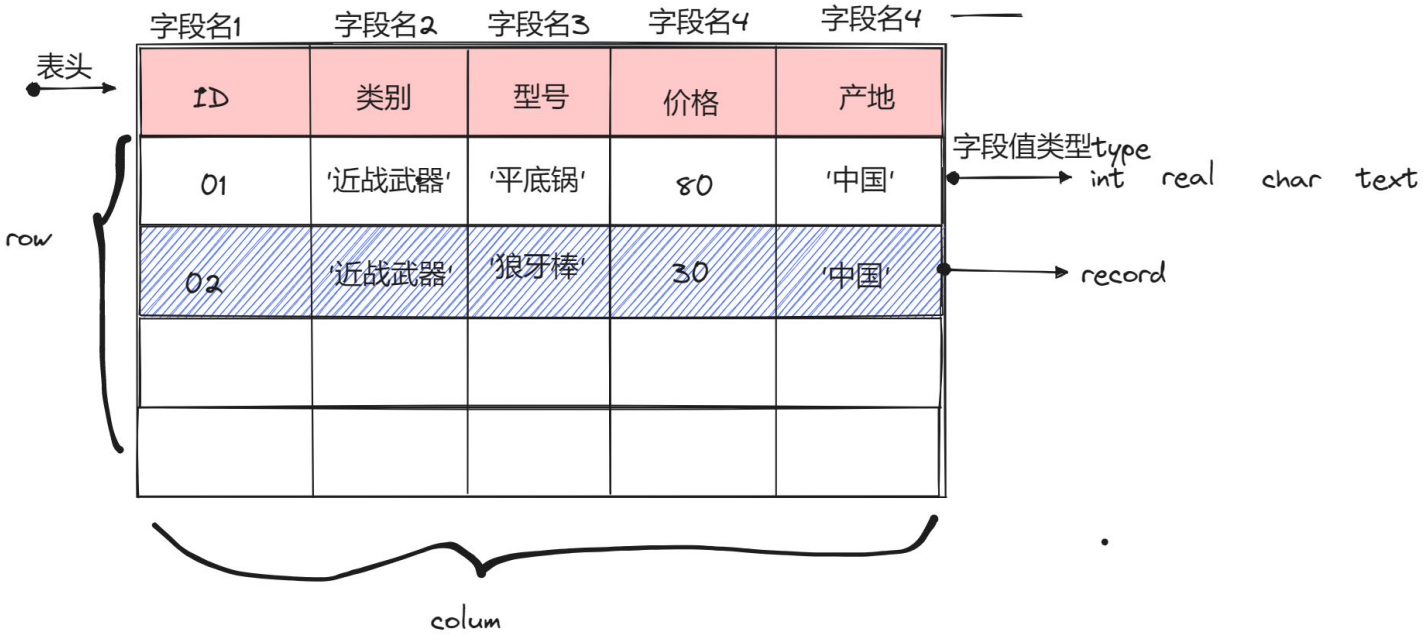
关系性数据库：以二维的表格来组织的数据形式就是关系数据库。

非关系性数据库：以键值对，或者图形或者文件组织而成的数据形式，叫非关系性数据库。

关系性数据库以sqlite3, mysql, Oracle为主。嵌入式用sqlite比较常用.

可以长期保存，一般以.db后缀保存。

非关系性数据库：MongDB以主。cache



1.2、何为SQL?

SQL的全称是Structured Query Language（结构化查询语言）。它是一种用于管理关系型数据库的编程语言，最初由IBM公司开发，现在已成为国际标准。SQL可以用于创建、修改和查询数据库中的数据，同时也可以用于管理数据库中的安全性、事务处理和备份恢复等方面。绝大多数的关系型数据库管理系统（RDBMS）都支持SQL语言，例如MySQL、Oracle、Microsoft SQL Server、PostgreSQL和SQLite等。由于SQL具有简单易学、高效可靠等优点，因此被广泛应用于企业管理、数据分析、Web开发等领域。操作数据库的术语Query

何为结构化? 在SQL中，"Structured"是指SQL语言具有一定的结构和规则，可以通过指定表、列、条件等来精确描述需要查询或修改的数据，从而使查询和修改操作更加可靠和高效。

1.3、SQLite数据库介绍：

SQLite是一个开源的，内嵌式的关系型数据库，第一个版本诞生于2000年5月，目前最高版本为SQLite3

官方下载地址：https://www.sqlite.org/index.html

Linux中在线安装SQLite3的方式： sudo apt-get install sqlite3 libsqlite3-dev

第二节、使用SQL语句操作SQLite数据库：

2.1 创建数据库文件：*.db文件

方法1.直接在终端使用sqlite3 filename的形式创建一个sqlite数据库文件：

```
sqlite3 filename.db   --------(*.db的文件后缀,有来表示是一种数据库文件，当然Linux并不以后缀为
```

方法2：进入sqlite数据程序后，使用.open + filename的形式创建一个数据库文件：

```
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open filename.db
```

在输入sqlite3之后，相当于进入了sqlite的应用程序终端，那么有两种形式的命令：

一种是以.开头的常用命令称为点命令：

```
在sqlite3中，以点（.）开头的命令称为“点命令”或“点命令”（dot command），
这些命令并不是SQL语句，而是用于管理SQLite数据库的特殊命令。
以下是一些常见的SQLite3点命令：
.help：显示可用命令的帮助信息。
.databases：显示当前连接的所有数据库。
.tables：列出当前数据库中的所有表格。
.schema：显示指定表或视图的DDL（数据定义语言）。表的纲要
.quit：退出SQLite3命令行界面。
.mode：设置输出模式。比如：按列显示内容：     .mode column
.header：启用或禁用输出标题。比如启用抬头显示：   .header on
.width 命令设置每个字段的最小显示宽度。例如，以下命令将每个字段的最小宽度设置为 10 个字符：
如果某个字段的值超过了 10 个字符，输出时该字段的宽度会自动扩展以适应数据。
这些点命令可以提高SQLite3数据库的管理效率和便利性。
```

另一种，在sqlite3的终端应用程序中，是以sql语句以;结尾的sql执行语句，下面进行介绍：

2.2常用通用的操作数据库的SQL语句：

1.在数据库中创建一张数据表：

create table 表名（字段名1 类型1，....字段名n 类型n）；

```
create table WeaponTable_02(序号 int,类别 text,型号 text,价格 int);
```

2.增：向数据表中插入一条record记录:

insert into 表名 values(字段值1，....字段值n);

```
insert into WeaponTable_02 values(1,"近战武器","解放版大砍刀",20);
```

注意：这种方式必须按照表中的字段名依次赋值，不可省略，可以使用同类型的值如0,或“”做为占位。

如果只想要对某几个字段中进行赋值，亦可以使用如下方式：指定字段赋值：

insert into 表示(要赋值的字段1，字段2， ... 字段n) values(字段值1， 字段值2，....);

比如：

省略价格时的方式：

```
insert into WeaponTable_02 values(1,"近战武器","解放版大砍刀",20);
```

3.查：查询记录select recode

数据查询语句（SELECT）：这是 SQL 语言中最常见的语句类型。SELECT 语句用于从数据库中检索数据，并且可以使用 WHERE 子句来指定检索数据的条件。在 SELECT 语句中，还可以使用 ORDER BY 子句对结果进行排序，使用 GROUP BY 子句对结果进行分组。

常用查询语句：

- **select * from 表名;**
- **select * from 表名 where ID=1 <指定字段值条件进行查询>**
- **select * from 表名 where 类别='自动步枪' order by 字段 ASC 或 DESC (升序或降序) <按条件查询并排序>**
- **select 类别 from 表名 group by 类别 <对结果进行分组返回分组名>**

```
select * from WeaponTable_02 ;
select * from WeaponTable_02 order by 价格 asc;
```

```
select * from WeaponTable_02 where 类别="自动武器";
select 类别 from WeaponTable_02 group by 类别;
```

4.改：更新记录的数据：

update 表名 set 字段=字段值 where id=xx;<找到符合条件的一条或多条记录并更新>

```
//更新一条：
update WeaponTable_02 set 产地="中国" where 序号=1;
//更新记录中的多个字段值：
update WeaponTable_02 set(型号,价格)=("平底锅",80) where 序号=1;
```

5.删：删除符合条件的一条或多条记录数据：

delete from 表名 [where condition];<where condition 是条件，按此条件删除记录>

如果直接delete from 表名，则为删除表中所有数据。

```
delete from WeaponTable_02 where 序号= 7;
```

6.删除数据表：drop table if exists 表名；

if exists为条件修饰词，意为：如果存在，则删除数据表，不存在，则不执行任何的操作也不抛出错误。这样可避免出现意外错误。

同理：也可以在创建表时，使用if not exists修饰 意为如果不存在，则创建，存在则不执行任何操作，也不抛出错误。

7.设置主键：PRIMARY KEY

使用PRIMARY KEY 修饰字段关键字在创建数时进行指定一个或多个列作为表的主键。主键是一种**用于唯一标识表中每行数据的机制**，它可以用来保证数据的一致性和完整性，以及提高查询效率。注意：如果指定了主键，则该列的值必须是唯一的，否则会抛出异常。另外，主键列不能为 NULL。

```
比如创建一个学生表格：
create table if not exists stu_table(id int primary key, 名字 text, 年龄 int, 成绩 int);
```

8.给数据表添加列属性：alter table 表名 add column 列字段 字段类型;

```
alter table WeaponTable_02 add column 产地 text;
```

第三节、sqlite3数据库操作api：

3.1、打开数据库api：sqlite3_open():

用于打开或创建SQLite3数据库文件。其函数原型如下：

```
#include <sqlite3.h>头文件
int sqlite3_open(
  const char *filename,   /* 数据库文件名 */
  sqlite3 **ppDb          /* 输出已打开的数据库实例 */
);
```

> 其中，**filename**参数为SQLite3数据库文件的路径和名称，可以是相对路径或绝对路径，如果文件不存在，则会创建一个新的数据库文件。**ppDb**参数为指向指针的指针，用于输出已打开的数据库实例，如果打开数据库文件成功，则该指针会指向一个sqlite3类型的数据库实例。
> sqlite3_open()函数返回一个整数值，用于表示打开SQLite3数据库文件的状态，常见的状态值有：
>
> - **SQLITE_OK - 成功打开数据库文件。**
> - **SQLITE_CANTOPEN - 打开数据库文件失败。**
> - **SQLITE_BUSY - 数据库文件已经被其他进程或线程打开。**
>
> **如果失败，则返回一个error code 并且可以使用sqlite3_errmsg()函数进行获取错误信息。**
> 需要**注意的是，打开SQLite3数据库文件后，需要使用sqlite3_close()或sqlite3_close_v2()函数来关闭数据库文件，以确保所有资源都被正确释放。**同时，在使用SQLite3 API时，应该避免SQL注入等安全问题(黑客技术：数据注入攻击)，确保应用程序的安全性。

以上可见一共有三个函数可以学习并使用。

代码示例：

```
#include <stdio.h>
#include <sqlite3.h>
int main(int argc, char const *argv[])
{
    sqlite3* databaseInstance = NULL;
    int ret = sqlite3_open("./MyWeapon.db",&databaseInstance);
    if(ret != SQLITE_OK)
    {
        printf("数据库打开失败\n");
        return -1;
    }
```

```
    printf("数据库成功创建\n");

    //数据库相关操作：.....

    //关闭数据库：
    sqlite3_close(databaseInstance);
    return 0;
}
```

注意：使用sqlite3的库的api接口编译时要链接sqlite3的库：-lsqlite3

3.2、解析并执行SQL语句api：sqlite3_exec()

sqlite3_exec() 函数是 SQLite3 库中的一个重要函数，用于执行 SQL 语句并处理结果。它的原型如下：

```
int sqlite3_exec(
  sqlite3*,                                  /* 数据库连接 */
  const char *sql,                           /* SQL 语句 */
  int (*callback)(void*,int,char**,char**),  /* 回调函数 */
  void *,                                    /* 回调函数的第一个参数 */
  char **errmsg                              /* 错误信息 */
);

```

> 该函数接受以下参数：
>
> - sqlite3*：指向 SQLite3 数据库连接对象的指针。
> - sql：需要执行的 SQL 语句。
> - callback：一个回调函数，用于处理查询结果。如果不需要处理结果，可以设置为 NULL。
> - void*：传递给回调函数的第一个参数，通常是一些上下文信息。
> - errmsg：如果出现错误，将返回一个指向错误信息字符串的指针。如果没有错误，则返回 NULL。
>
> **sqlite3_exec()** 函数可以用于执行任何 SQL 语句，例如 **SELECT**、**INSERT**、**UPDATE** 和 **DELETE** 等语句。它还支持参数化查询，可以使用 **?** 占位符来代替参数值。
> 回调函数通常用于处理查询结果。当执行查询时，SQLite3 将结果作为一个二维字符串数组返回，其中每个元素都表示一行和一列的值。回调函数需要使用这个结果数组进行处理，通常是将其转换为更易于使用的数据结构，例如 C 结构体或数组。
>
> - 如果返回值为 **SQLITE_OK**，表示执行 SQL 语句成功，并且没有错误。
> - 如果返回值为 **SQLITE_ABORT**，表示执行 SQL 语句被中止，通常是由于回调函数返回了非零值。
> - 如果返回值为其他非零值，表示执行 SQL 语句失败，并且返回的错误码可以使用 SQLite3 错误码列表中的常量进行解释。
>
> 总之，**sqlite3_exec()** 函数是 SQLite3 库中的一个重要函数，用于执行 SQL 语句并处理结果。它是 SQLite3 库中最常用的函数之一，也是开发 SQLite3 应用程序的基础。

```c
#include <stdio.h>
#include <sqlite3.h>
#include <stdbool.h>
//定义一个信息结构体:
typedef struct
{
    int order_index;
    char type[32];
    char vision[32];
    int price;
    char production[32];
} WeaponInfo;
//展示抬头:
void showTitle()
{
    printf("-----------------------------------------\n");
    printf("-----1.添加---2.查询---3.更改---4.删除-----\n");
    printf("-----------------------------------------\n");
}
//添加数据:
void do_insert(WeaponInfo *weaponInfo, sqlite3 *mydb)
{
    memset(weaponInfo, 0, sizeof(WeaponInfo));
    printf("请输入序号: \n");
    scanf("%d", &weaponInfo->order_index);
    printf("请输入类别: \n");
    scanf("%s", &weaponInfo->type);
    getchar();
    printf("请输入型号: \n");
    scanf("%s", &weaponInfo->vision);
    getchar();
    printf("请输入价格: \n");
    scanf("%d", &weaponInfo->price);
    printf("请输入产地: \n");
    scanf("%s", &weaponInfo->production);
    getchar();
    char sql[528] = {0};
    char *msgerr = NULL;
    sprintf(sql, "insert into WeaponTable values(%d,'%s','%s',%d,'%s')",
            weaponInfo->order_index,
            weaponInfo->type,
            weaponInfo->vision,
```

```
            weaponInfo->price,
            weaponInfo->production);
    int ret = sqlite3_exec(mydb, sql, NULL, NULL, &msgerr);
    if (ret != SQLITE_OK)
    {
        printf("err :%s \n", msgerr);
        return;
    }
}
//构建数据库查询操作:



int main(int argc, char const *argv[])
{
    sqlite3 *databaseInstance = NULL;
    // 打开数据库:
    int ret = sqlite3_open("./MyWeapon.db", &databaseInstance);
    if (ret != SQLITE_OK)
    {
        printf("数据库打开失败\n");
        return -1;
    }
    printf("数据库成功创建\n");

    // 数据库相关操作: .....
    // 在数据库文件中创建数据表: WeaponTable;
    char *msgerr = NULL;
    const char *sql = "create table if not exists WeaponTable(序号 int primary key, 类别
    ret = sqlite3_exec(databaseInstance, sql, NULL, NULL, &msgerr);
    if (ret != SQLITE_OK)
    {
        printf("err : %s", msgerr);
    }
    WeaponInfo weaponInfo = {0};
    // 展示数据库的抬头:
    int checkopt = 0;
    while (true)
    {
        showTitle();
        printf("请选择你要操作选项: \n");
        scanf("%d", &checkopt);
        switch (checkopt)
```

```
        {
        case 1:
            do_insert(&weaponInfo, databaseInstance);
            break;
        case 2:
            /* code */
            break;
        case 3:
            /* code */
            break;
        case 4:
            /* code */
            break;
        default:
            break;
        }
    }
    // 关闭数据库:
    sqlite3_close(databaseInstance);
    return 0;
}
```

3.3、 callback函数形式：查询时使用的回调函数。

在 **sqlite3_exec()** 函数中，回调函数的原型如下：

```
int callback(void *data, int argc, char **argv, char **azColName);
```

> 回调函数有四个参数：
>
> - **data：是回调函数的第一个参数，通常用于传递上下文信息。可以是任意类型的指针，由应用程序自己决定。**
> - **argc：表示结果集的列的个数。**
> - **argv：二级指针，是一个字符串数组，代表结果集中的一行数据（即一行多列的字段值，每一列字段值由一个字符指针指向）**
> - **azColName：二级指针，也是一个字符串数组，表示结果集每一列的列名（即一行的多列的字段名是什么，每一列字段名由一个字符指向指向）**

在回调函数中，通常需要遍历 **argv** 数组（**真正的每一行中的每一列的属性值的集合**），将结果转换为需要的数据类型，并进行处理。执行 **SELECT** 查询时，回调函数会被调用多次，每次调用都对应查询结果集中的一行数据。具体来说，回调函数会被调用的次数等于查询结果集中的行数。例如：如果查询结果集中有 10 行数据，那么回调函数就会被调用 10 次，每次传递一行数据。在回调函数中，你可以对传递的每一行数据进行处理，并根据需要将它们保存到应用程序中。如果查询结果集为空，则回调函数不会被调用。

回调函数通常需要返回一个整数值，其含义取决于回调函数的上下文。在 **sqlite3_exec()** 函数中，回调函数需要返回一个整数值，用于指示是否需要中止查询操作。如果返回非零值，**sqlite3_exec()** 函数将停止执行查询，并返回 **SQLITE_ABORT** 错误码。**一般情况下返回0为继续查询，每查完一次，则回调一次，止到查完记为止。如果返回值为非零将停止执行下一次的查询。**

1.使用api接口构建数据库，并插入数据记录并查询数据：

```c
#include <stdio.h>
#include <sqlite3.h>
#include <stdbool.h>
//定义一个信息结构体:
typedef struct
{
    int order_index;
    char type[32];
    char vision[32];
    int price;
    char production[32];
} WeaponInfo;
//展示抬头:
void showTitle()
{
    printf("-------------------------------------------\n");
    printf("-----1.添加---2.查询---3.更改---4.删除-----\n");
    printf("-------------------------------------------\n");
}
//添加数据:
void do_insert(WeaponInfo *weaponInfo, sqlite3 *mydb)
{
    memset(weaponInfo, 0, sizeof(WeaponInfo));
    printf("请输入序号: \n");
    scanf("%d", &weaponInfo->order_index);
    printf("请输入类别: \n");
    scanf("%s", &weaponInfo->type);
    getchar();
    printf("请输入型号: \n");
    scanf("%s", &weaponInfo->vision);
    getchar();
    printf("请输入价格: \n");
    scanf("%d", &weaponInfo->price);
    printf("请输入产地: \n");
    scanf("%s", &weaponInfo->production);
    getchar();
```

```
    char sql[528] = {0};
    char *msgerr = NULL;
    sprintf(sql, "insert into WeaponTable values(%d,'%s','%s',%d,'%s')",
            weaponInfo->order_index,
            weaponInfo->type,
            weaponInfo->vision,
            weaponInfo->price,
            weaponInfo->production);
    int ret = sqlite3_exec(mydb, sql, NULL, NULL, &msgerr);
    if (ret != SQLITE_OK)
    {
        printf("err :%s \n", msgerr);
        return;
    }
}
//查询执行的回调函数:
int select_query(void* data,int argc,char** argv,char** azColumnName)
{
    static int i = 0;
    if(i++ == 0)
    {
        printf("---%s---%s---%s---%s---%s---\n",azColumnName[0],azColumnName[1],
        azColumnName[2],azColumnName[3],azColumnName[4]);
    }
    printf("---%s---%s---%s---%s---%s---\n",argv[0],argv[1],
    argv[2],argv[3],argv[4]);
    //注意:在使用回调函数一定要返回一个整数。返回0即表示执行完毕。
    return 0;
};

//构建数据库查询操作:
void do_select(sqlite3* mydb)
{
    char* msgerr = NULL;
    const char* sql = "select * from WeaponTable";
    int ret = sqlite3_exec(mydb,sql,select_query,NULL,&msgerr);
    if(ret != SQLITE_OK)
    {
        printf("err : %s \n",msgerr);
        return;
    }
}
int main(int argc, char const *argv[])
{
```

```
    sqlite3 *databaseInstance = NULL;
    // 打开数据库:
    int ret = sqlite3_open("./MyWeapon.db", &databaseInstance);
    if (ret != SQLITE_OK)
    {
        printf("数据库打开失败\n");
        return -1;
    }
    printf("数据库成功创建\n");

    // 数据库相关操作: .....
    // 在数据库文件中创建数据表: WeaponTable;
    char *msgerr = NULL;
    const char *sql = "create table if not exists WeaponTable(序号 int primary key, 类别
    ret = sqlite3_exec(databaseInstance, sql, NULL, NULL, &msgerr);
    if (ret != SQLITE_OK)
    {
        printf("err : %s", msgerr);
    }
    WeaponInfo weaponInfo = {0};
    // 展示数据库的抬头:
    int checkopt = 0;
    while (true)
    {
        showTitle();
        printf("请选择你要操作选项: \n");
        scanf("%d", &checkopt);
        switch (checkopt)
        {
        case 1:
            do_insert(&weaponInfo, databaseInstance);
            break;
        case 2:
            do_select(databaseInstance);
            break;
        case 3:
            /* code */
            break;
        case 4:
            /* code */
            break;
        default:
            break;
        }
```

```
    }
    // 关闭数据库:
    sqlite3_close(databaseInstance);
    return 0;
}

```

3.4、另一种查询api：sqlite3_get_table（）（对内存不太友好）

sqlite3_get_table 是一个 SQLite3 C API 函数，用于执行一个 SQL 查询并将结果存储在一个字符指针数组中。该函数会返回一个二维字符指针数组，其中第一行是结果集中的列名，后面的行是结果集中的行数据。该函数的函数原型如下：

```
int sqlite3_get_table(
  sqlite3 *db,               /* 数据库句柄 */
  const char *zSql,          /* SQL 语句 */
  char ***pazResult,         /* 用于返回结果集的指针，即字符指针数组的指针地址 */
  int *pnRow,                /* 用于返回结果集的行数 */
  int *pnColumn,             /* 用于返回结果集的列数 */
  char **pzErrmsg            /* 用于返回错误消息的指针 */
);

```

> 该函数的参数包括：
>
> - db: 指向 SQLite3 数据库句柄的指针。
> - zSql: SQL 查询语句。
> - pazResult: 一个指向字符指针数组的指针，用于返回结果集。这个数组中包含了所有的查询结果，包括列名和行数据。
> - pnRow: 用于返回结果集中的行数。如要显示表头，输出时请加+1;
> - pnColumn: 用于返回结果集中的列数。
> - pzErrmsg: 用于返回错误消息的指针。
>
> 该函数的返回值为整数，表示函数执行成功与否。如果返回值为 **SQLITE_OK**，表示函数执行成功，查询结果被存储在 **pazResult** 指向的数组中。如果返回值为其他错误代码，例如 **SQLITE_ERROR**，则表示函数执行失败，可以通过 **pzErrmsg** 指针获取错误消息。

```
//使用sqlite3_get_table接口来构建查询结果的输出函数;
void do_get_table(sqlite3* mydb)
{
    const char* sql = "select * from WeaponTable";
    char** azResult = NULL;
    int row = 0;
```

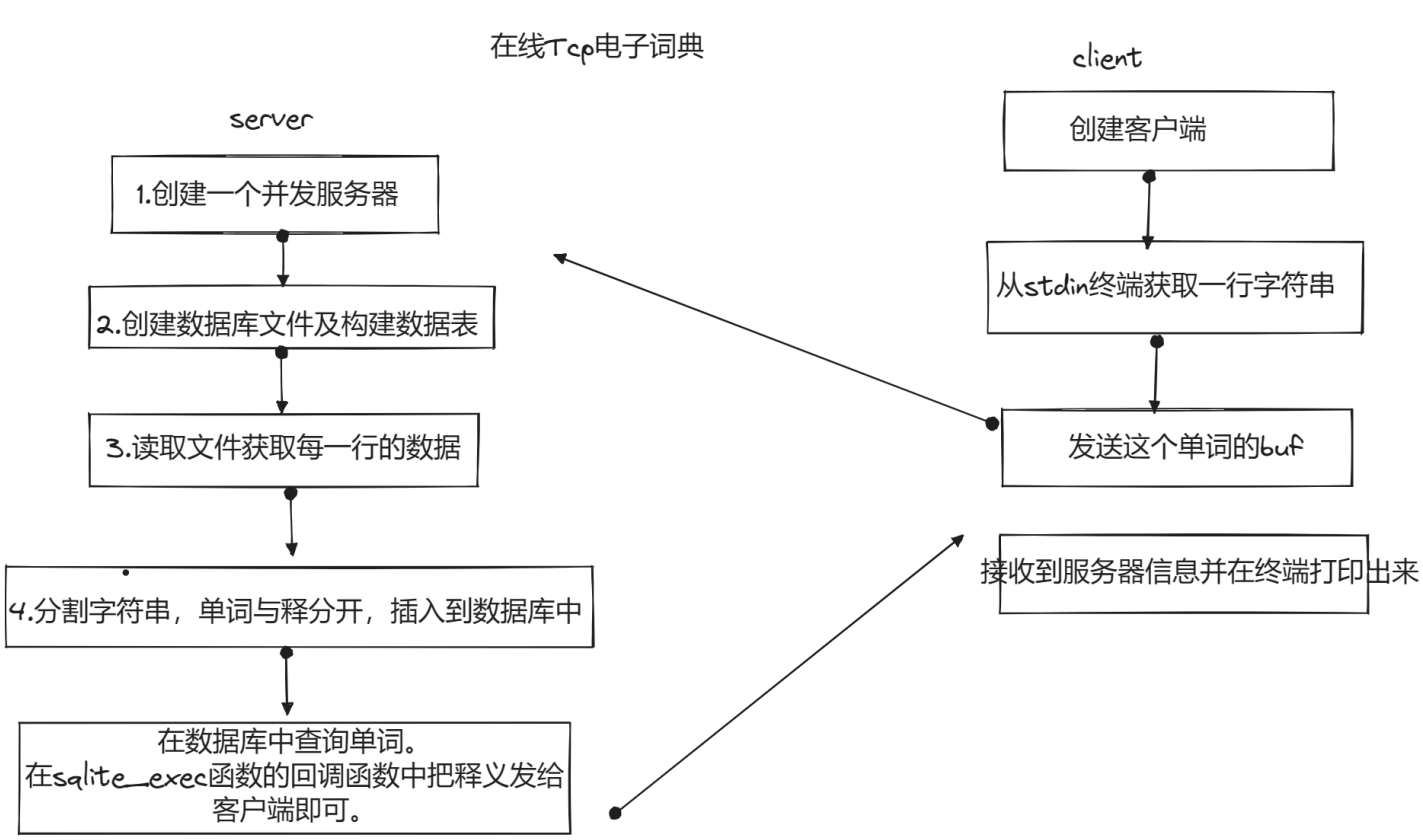
```
    int column = 0;
    char* msgErr = NULL;
    //get_table的使用。
    int ret = sqlite3_get_table(mydb,sql,&azResult,&row,&column,&msgErr);
    if(ret != SQLITE_OK)
    {
        printf("获取结果集失败 : %s \n",msgErr);
        return ;
    }
    printf("-------\n");
    //遍历这个集合，注意加上表头所占的一行。
    for(int i = 0; i < (row+1)*column; i++)
    {
        if(i % column == 0)
        {
            printf("\n");
        }
        else
        {
            printf("---%s---",azResult[i]);
        }

    }

    printf("\n");
}

```

最后练习、TCP电子词典小项目构建思路：


在线Tcp电子词典
server
1.创建一个并发服务器
2.创建数据库文件及构建数据表
3.读取文件获取每一行的数据
4.分割字符串，单词与释分开，插入到数据库中
在数据库中查询单词。
在sqlite_exec函数的回调函数中把释义发给客户端即可。
client
创建客户端
从stdin终端获取一行字符串
发送这个单词的buf
接收到服务器信息并在终端打印出来