Sqlite3小结（小型数据库中增删改查的操作）

<https://blog.csdn.net/nan_lei/article/details/84344003>

原创nan\_lei 发布于2018-11-22 16:31:32 阅读数 6339 收藏

展开

说一点：学习的话还是要用印象笔记~没有为什么~：数据库浅学

一、数据库简介

1、数据库简介

    数据库(Database)是按照数据结构来组织、存储和管理数据的仓库。

    在信息化社会，充分有效地管理和利用各类信息资源，是进行科学研究和决策管理的前提条件。数据库技术是管理信息系统、办公自动化系统、决策支持系统等各类信息系统的核心部分，是进行科学研究和决策管理的重要技术手段。

    数据库的使用需要依赖数据库管理系统（Database Management System，简称DBMS）。数据库管理系统是对数据库中的数据进程同一管理和控制的软件系统。通常情况下，DBMS有以下功能：

    -数据库定义

    -数据库操作

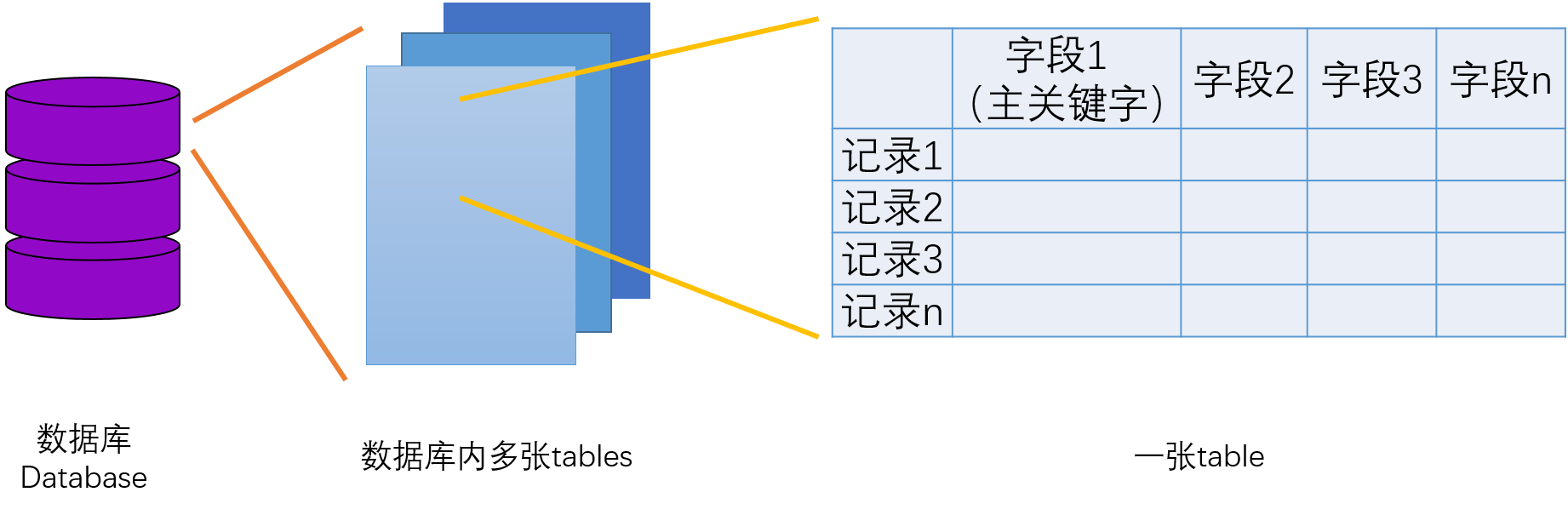
    -数据库运行控制

    -数据通信

    -海量存取数据

    最早期的数据库可以追溯到上世纪五十年代，当时雷明顿兰德公司（Remington Rand Inc）为世界上第一台电子计算机Univac I发明了一种一秒钟可以输入数百条纸带的驱动器，可以看做数据库最早的原型。

    真正意义上第一个数据库程序诞生于1961年。美国通用电气公司Bachman等人在1961年开发了IDS（Integrated Data Store）。后来在1964年通用电气公司（General ElectricCo.）的Charles Bachman 成功地开发出世界上第一个网状DBMS也即第一个数据库管理系统——集成数据存储（Integrated Data Store IDS），奠定了网状数据库的基础，并在当时得到了广泛的发行和应用。



2、常见数据库简介

目前比较流行的数据库有以下几种：

    1.大型数据库Oracle Database

    Oracle公司（甲骨文公司）是全球最大的企业级软件公司，其公司旗下著名产品有Java编程语言与OracleDatabase。Oracle公司主要面向大型企业，为各大企业提供专业级的数据库解决方案。

    2.办公用数据库Microsoft Access

    Access是微软在Office办公套件内提供的数据库软件，具有数据库引擎和图形化交互界面，并且可以使用VBA编程环境对其进行深度编程操作。

    3.小型数据库MySQL

    由瑞典MySQLAB公司开发（该公司已被Oracle公司收购），经常应用于WEB前端领域。由于体积小、速度快、代码开源深受程序开发人员喜爱。还可以配合PHP和Apache搭建数据库服务器。

    4.嵌入式数据库SQLite

    SQLite数据库是专门针对嵌入式开发的数据库软件，占用资源非常少（仅需几百K内存即可），处理速度快，而且可以直接使用C语言编程，同时也可以配合C#、PHP、Java等其他语言。SQLite诞生于2000年，2015年发布了新版本SQLite3，一经推出马上变成了最流行的嵌入式数据库软件。

//SQL中文翻译为“结构化查询语言”，是单词"Structured Query Language"的缩写。SQL是一种操作数据库的高级的非过程化编程语言，用户使用SQL命令通过DBMS对数据库内的数据进行访问。1987年，SQL得到国际标准组织ISO的支持成为数据库的国际标准，不过各种数据库的SQL命令略有不同，因此并不能够完全通用。

//SQL到底应该读作"S-Q-L"三个字母还是读作"色Q尔"整体读音，这个争    论时至今日仍没有定论

二、SQLite3数据库使用

//使用SQLite3需要安装，若当前主机没有SQLite3软件可以输入命令sudo apt-get install sqlite3安装

使用SQLite3数据库有两种方式：

    1.手动创建

    直接在terminal内输入sqlite即可进入SQLite3数据库软件，通过手动输入SQL命令行完成数据库的创建。

    2.代码创建

    由于SQLite3是基于C语言进行编程，因此可以直接在C语言代码中以函数调用和传参的形式操作数据库。在程序运行中，当需要操作数据库时，应用程序会打开数据库，若此时数据库不存在则会创建一个数据库（类似"w"模式打开文件）。注意使用过数据库编程后需要关闭。



1、SQLite3创建数据库

在terminal输入

    sqlite3 mydb.db

创建一个本地的数据库文件mydb.db

当成功创建数据库后，会出现以下提示信息：

    SQLite version 3.7.9 2011-11-01 00:52:41

    Enter ".help" for instructions

    Enter SQL statements terminated with a ";"

    sqlite>

其中"sqlite>"相当于terminal的命令提示符，在后面输入数据库操作命令

如果想退出数据库，则可以使用命令

    .quit（简写为.q）

或者

    .exit（简写为.e）

退出数据库。

2、创建表和删除表

数据库创建完毕后，还需要创建表来存储数据。输入命令

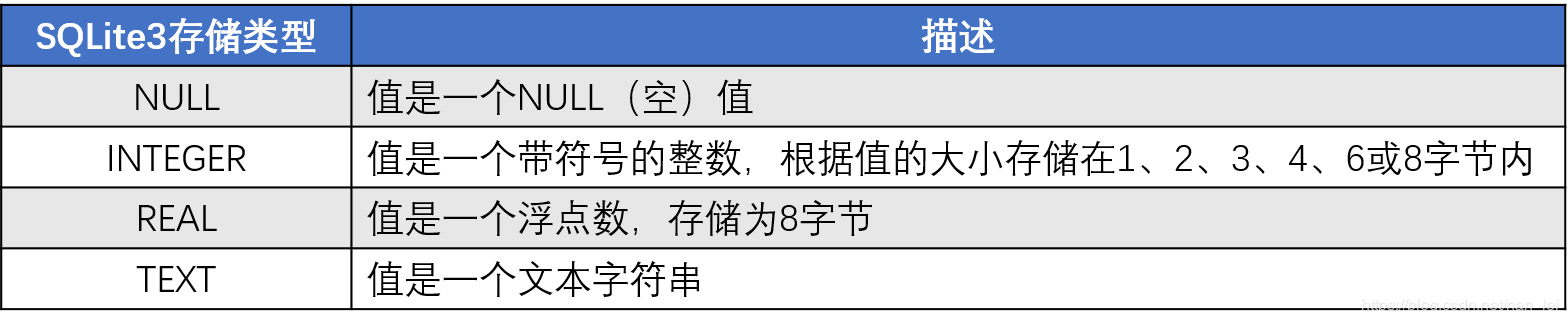
    create table table\_name(column1 datatype PRIMARY KEY, column2 datatype, ……);//注意命令后的分号

来创建一张表。

//PRIMARY KEY表示该列为该表的“主关键字”，主关键字用于唯一索引表内的某一条记录，主关键字必须唯一且主关键字的列值不能为空

//一张表允许省略主关键字

创建表时注意每一列需要指定数据类型。SQLite的数据类型



输入命令

    .tables

查看创建表是否成功。

如果想获取表格的格式信息，则可以输入命令

    .schema

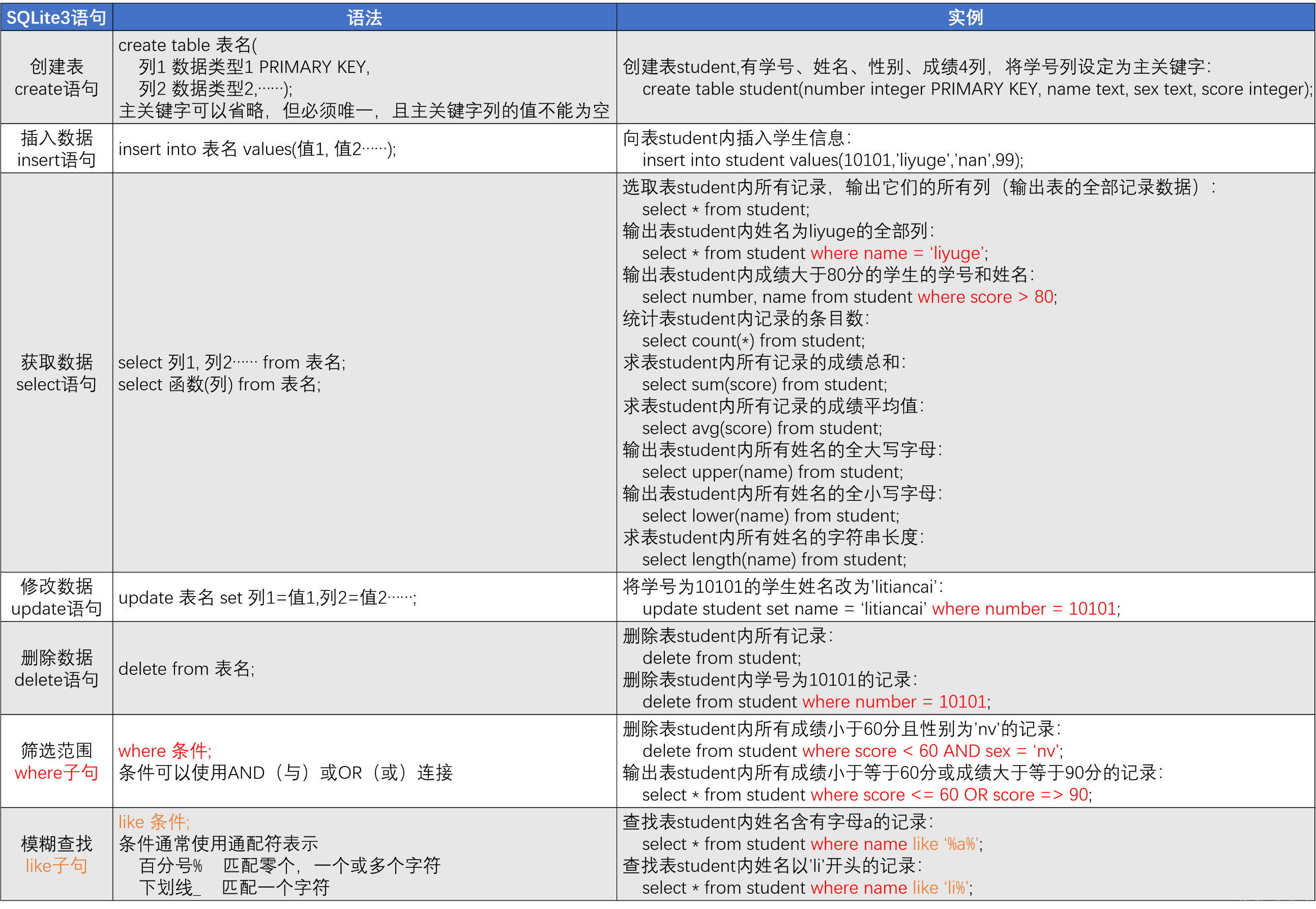
查看表格中列的完整信息。

如果想删除某表，则可以输入命令

    drop table table\_name;

来删除一张表。

//SQLite语句与子句用法



3、向表中插入数据——insert语句

在创建了一张表之后，我们就可以向表中插入数据了。输入命令

    insert into table\_name values(value1,value2……);

向指定名称的表中插入一个记录。

4、设定操作数据条件——where子句

在SQLite语句中，where子句用于设定筛选条件，如果给定的条件为真则从表中获取该记录。

where子句还可以搭配AND和OR运算符连接条件。其中：

    AND表示“与”，使用AND连接的所有条件必须都为真，整个条件才为真

    OR表示“或”，使用OR连接的所有条件只要有其中一条为真，则整个条件为真

where子句通常与select语句、update语句和delete语句搭配使用，具体见下面语句的操作。

5、删除表中的数据——delete语句

我们可以使用delete语句将表内的一个记录删除。输入命令

    delete from table\_name where condition;

其中删除记录需要在where后输入附加条件，如果有多个条件，则可以使用AND/OR连接条件。例如：

    delete from student where number = 10102;

表示从表student中删除number等于10102的记录。

    delete from student where score < 60 AND sex = 'nan';

表示从表student中删除score小于60并且sex为nan的记录。

如果想删除某表内所有记录，则可以输入命令

    delete from table\_name;

6、修改表内记录——update语句

如果想修改表内某个记录的某个或某几个列的值，则可以输入命令

    update table\_name set column1 = value1, column2 = value2…… where condition;

其中修改记录需要在where后输入附加条件，如果有多个条件，则可以使用AND/OR连接条件。

7、查询表内记录——select语句

使用select语句可以输出一个表内指定的列信息，然后使用where子句确定筛选条件。

    select column1,column2…… from table\_name where condition;

如果想输出表内所有列，则可以将column指定为\*

    select \* from table\_name;//输出表内所有列的信息

/\*为了输出美观，可以在执行select语句输出表内信息之前使用

    .header on

和

    .mode column

设定输出格式\*/

除此之外，select语句还可以搭配一些SQLite功能函数使用，常用的有：

    count(\*)        统计表内行数（记录数）    sqlite&gt; select count(\*) from stu;

    max(column)        计算该列的最大值        columen 是一列的名字

    min(column)        计算该列的最小值

    avg(column)        计算该列的平均值

    sum(column)        计算该列的总和

    upper(column)    输出选中的列的所有字符串的全大写字母

    lower(column)    输出选中的列的所有字符串的全小写字母

    length(column)    计算该列的所有字符串的长度

将这些函数放在select语句内可以完成一些计算工作。例如输入命令

    select sum(score) from student;

可以求出表student内所有学生的成绩总和。

8、模糊查询——like子句

除了使用where子句设定筛选条件外，我们还可以使用like子句模糊匹配表内的记录。like语句搭配两个通配符一起使用：

    百分号%    表示零个、一个或多个数字或字符

    下划线\_    表示一个数字或字符

如果某条记录内的某一列能与like后的条件匹配，则该记录被选定。例如：

    select \* from student where name like %l%;//从表student中选取名字内带字母l的所有记录

练习：使用SQLite3创建数据库，建立一个简易的图书管理系统。

数据库内包括三张表：

    1.图书基本信息数据表。列：图书编号、图书名称、出版社、作者、出版日期……

    2.办理借阅卡的学生表。列：借阅卡编号、学生学号、学生姓名、所在班……

    3.学生借阅图书表。列：借阅卡编号、学生编号、图书编号、借阅日期、借阅时间、是否归还……

在三张表内添加适当记录，并进行以下操作：

    1.查询指定出版社的所有图书

    2.查询指定作者的所有图书

    3.查询指定学号的学生的借阅卡信息

    4.查询指定学号的学生的图书借阅信息

    5.查询所有未归还图书

    6.若某图书被借阅，在借阅图书表内记录借阅信息

    7.若某图书被归还，删除该图书在借阅图书表内记录

三、SQLite数据库与C语言编程应用

1、sqlite编程步骤

    除了在SQLite3的终端界面操作数据库之外，SQLite3另一个特点就是可以直接作用于多种编程语言的代码内（例如C/C++、Java、PHP、Python等语言）。当这些编程语言需要数据库时，直接调用SQLite3预留给该编程语言的接口函数即可实现该数据库的操作或得到数据记录。

以C语言为例，在C语言代码内操作SQLite3数据库类似于文件IO操作，具体步骤如下：

    1.打开数据库sqlite3\_open()

    2.使用函数sqlite3\_exec()或其他非回调函数形式操作数据库

    3.关闭数据库sqlite3\_close()

如果在C语言代码中操作SQLite3数据库，需要在程序开头添加头文件

    #include<sqlite3.h>

并且在编译时需要添加额外的库

    gcc \* -lsqlite3

注意：sqlite3库并非Linux系统自带C语言函数库，因此在编译时可能会提示缺少该库。若缺少该库，可以输入命令

    sudo apt-get install libsqlite3-dev

来安装该库。

2、打开/关闭数据库

我们可以使用sqlite3\_open()函数来打开一个数据库，使用sqlite3\_close()函数来关闭一个数据库。

    函数sqlite3\_open()

    所需头文件：#include<sqlite3.h>

    函数原型：int sqlite3\_open(const char \*filename, sqlite3 \*\*db)

    函数参数：

        filename    需要操作的数据库的路径，若不存在则会创建该数据库

        db            操作该数据库的句柄指针，必须为sqlite\*类型的指针并且使用地址传递

    函数返回值：

        成功：0

        失败：错误码（大于0）

当打开数据库成功后，第二个参数db指针就指向该数据库，后续的操作都借助该指针进行。该指针也被称为“句柄”。

与文件IO类似，数据库使用完毕后也需要关闭

    函数sqlite3\_close()

    所需头文件：#include<sqlite3.h>

    函数原型：int sqlite3\_close(sqlite3 \*db)

    函数参数：

        db        需要关闭的数据库的句柄指针

    函数返回值：

        成功：0

        失败：错误码（大于0）

示例：使用sqlite3\_open()和sqlite3\_close()来打开和关闭一个数据库

//注意：编译时需添加-lsqlite3库

#include<stdio.h>

#include<sqlite3.h>

int main(int argc, const char \*argv[])

{

    if(argc<2)

    {

        printf("too few argument\n");

        printf("Usage: %s <database>\n",argv[0]);

        return 0;

    }

    sqlite3 \*db;

    if(sqlite3\_open(argv[1],&db)!=0)//打开数据库

    {

        perror("cannot open database");

    }

    //对数据库的操作

    sqlite3\_close(db);//关闭数据库

    return 0;

}

3、使用回调函数操作数据库

    SQLite3的C语言编程中，所有之前学习过的SQLite语句都可以借助sqlite3\_exec()函数对该数据库的表进行相关操作。

    在sqlite3\_exec()函数中，第三个参数可以传参一个指定类型的函数指针，这样当对数据库执行SQLite语句成功后会自动运行该回调函数。该参数并非必须，某些操作（例如create或者delete等）可以不需要回调函数。若不需要，则第三个参数可以设定为NULL。

    若回调函数指针非空，当我们使用sqlite3\_exec()操作数据库的表时，每一次找到符合要求的记录都会执行一次回调函数，此时该回调函数的第二个形参存储该记录的字段（列）的数目，第三个形参存储该记录每个字段（列）的值，第四个形参存储该记录每个字段（列）的名称。

    函数sqlite3\_exec()

    所需头文件：#include<sqlite3.h>

    函数原型：

        typedef int (\*sqlite3\_callback)(void\*,int,char\*\*,char\*\*);    //定义sqlite3\_exec()需要的回调函数的函数指针类型

        int sqlite3\_exec(sqlite3 \*db, const char \*sql, sqlite3\_callback callback, void\* para, char \*\*errmsg)

    函数参数：

        db            需要操作的数据库的句柄指针

        sql            需要对数据库执行的SQLite语句

        callback    回调函数，当执行SQLite语句（第二个参数）成功后会自动运行该函数。如果不需要则该参数可以设定为NULL

        para        传递给回调函数第一个参数的指针。如果不需要则该参数可以设定为NULL。

        errmsg        存储错误信息指针，必须使用地址传递传参。若sqlite3\_exec()执行失败则会在堆区开辟内存空间存放错误信息，并将该空间首地址返回给errmsg

    函数返回值：

        成功：0

        失败：错误码（大于0）

第三个参数需要一个指定函数类型的函数指针。回调函数的函数定义为：

    函数sqlite3\_exec()需要的回调函数sqlite3\_callback

    函数原型：

        int function(void \*para, int f\_num, char \*f\_value[], char \*f\_name[])

    函数参数：

        para        传递给回调函数的参数（即sqlite3\_exec()的第四个参数）

        f\_num        记录中包含的字段数目

        f\_value        包含每个字段值的指针数组

        f\_name        包含每个字段名称的指针数组

注意：sqlite3\_exec()内para参数在传参时必须强转为void\*类型，在回调函数内再强转回原来的指针类型

示例1：使用sqlite3\_exec()创建表student。表student内的字段有

    学号（主关键字）-->    整数

    姓名 ------------->    字符串

    性别 ------------->    字符串

    年龄 ------------->    整数

    成绩 ------------->    整数

#include<stdio.h>

#include<sqlite3.h>

int main(int argc, const char \*argv[])

{

    if(argc<2)

    {

        printf("too few argument\n");

        printf("Usage: %s <database>\n",argv[0]);

        return 0;

    }

    char create[]="create table student(number integer PRIMARY KEY, name text, sex text, age integer, score integer);";

    char \*errmsg;

    sqlite3 \*db;

    if(sqlite3\_open(argv[1],&db)!=0)

    {

        perror("cannot open database");

    }

    if(sqlite3\_exec(db,create,NULL,NULL,&errmsg)!=0)

    {

        printf("%s\n",errmsg);

    }

    sqlite3\_close(db);

    return 0;

}

示例2：在示例1的基础上，向表student内插入5个学生的成绩信息

学号    姓名    性别    年龄    成绩

10101    liyuge    nan        18        99

10102    tiancai    nan        18        98

10103    hehe    nv        19        96

10104    xixi    nv        19        94

10105    haha    nan        18        90

#include<stdio.h>

#include<sqlite3.h>

int main(int argc, const char \*argv[])

{

    if(argc<2)

    {

        printf("too few argument\n");

        printf("Usage: %s <database>\n",argv[0]);

        return 0;

    }

    //char create[]="create table student(number integer PRIMARY KEY, name text, sex text, age integer, score integer);";

    char \*insert[]={

        "insert into student values(10101,'liyuge','nan',18,99);",

        "insert into student values(10102,'tiancai','nan',18,98);",

        "insert into student values(10103,'hehe','nv',19,96);",

        "insert into student values(10104,'xixi','nv',19,94);",

        "insert into student values(10105,'haha','nan',18,90);"

    };

    int len = sizeof(insert)/sizeof(insert[0]);

    int i;

    char \*errmsg;

    sqlite3 \*db;

    if(sqlite3\_open(argv[1],&db)!=0)

    {

        perror("cannot open database");

    }

    /\*上文执行过的代码本次不再执行，下同

  //创建表student

    if(sqlite3\_exec(db,create,NULL,NULL,&errmsg)!=0)

    {

        printf("%s\n",errmsg);

    }

    \*/

    //向表内插入学生信息

    for(i=0;i<len;i++)

    {

        if(sqlite3\_exec(db,insert[i],NULL,NULL,&errmsg)!=0)

        {

            printf("%s\n",errmsg);

        }

    }

    sqlite3\_close(db);

    return 0;

}

示例3：在示例2的基础上，使用回调函数，输出表student内所有学生信息

#include<stdio.h>

#include<sqlite3.h>

int print\_callback(void \*para, int f\_num, char \*f\_value[], char \*f\_name[])

{

    int i;

    //printf("f\_num is %d\n",f\_num);//测试输出

    for(i=0;i<f\_num;i++)

    {

        printf("%s:\t%s\n",f\_name[i],f\_value[i]);

    }

    printf("--------------------------\n");

    return 0;

}

int main(int argc, const char \*argv[])

{

    if(argc<2)

    {

        printf("too few argument\n");

        printf("Usage: %s <database>\n",argv[0]);

        return 0;

    }

    //char create[]="create table student(number integer PRIMARY KEY, name text, sex text, age integer, score integer);";

    /\*

    char \*insert[]={

        "insert into student values(10101,'liyuge','nan',18,99);",

        "insert into student values(10102,'tiancai','nan',18,98);",

        "insert into student values(10103,'hehe','nv',19,96);",

        "insert into student values(10104,'xixi','nv',19,94);",

        "insert into student values(10105,'haha','nan',18,90);"

    };

    int len = sizeof(insert)/sizeof(insert[0]);

    \*/

    char select[]="select \* from student;";

    //int i;

    char \*errmsg;

    sqlite3 \*db;

    if(sqlite3\_open(argv[1],&db)!=0)

    {

        perror("cannot open database");

    }

    /\*

    //创建表student

    if(sqlite3\_exec(db,create,NULL,NULL,&errmsg)!=0)

    {

        printf("%s\n",errmsg);

    }

    \*/

    /\*

    //向表内插入学生信息

    for(i=0;i<len;i++)

    {

        if(sqlite3\_exec(db,insert[i],NULL,NULL,&errmsg)!=0)

        {

            printf("%s\n",errmsg);

        }

    }

    \*/

    //执行回调函数打印表内数据

    if(sqlite3\_exec(db,select,print\_callback,NULL,&errmsg)!=0)

    {

        printf("%s\n",errmsg);

    }

    sqlite3\_close(db);

    return 0;

}

示例4：在示例3的基础上，建立一个结构体数组，将数据库内的数据存放在结构体数组内

#include<stdio.h>

#include<string.h>

#include<sqlite3.h>

struct Student

{

    int number;

    char name[16];

    char sex[4];

    int age;

    int score;

};

int offset=0;//使用全局变量offset记录回调函数执行次数

//注意回调函数第一个形参的用法

int callback(void \*para, int f\_num, char \*f\_value[], char \*f\_name[])

{

    int i;

    struct Student \*p = (struct Student\*)para+offset;

    //printf("f\_num is %d\n",f\_num);//测试输出

    /\*

    for(i=0;i<f\_num;i++)

    {

        printf("%s:\t%s\n",f\_name[i],f\_value[i]);

    }

    printf("--------------------------\n");

    \*/

    p->number = atoi(f\_value[0]);

    strcpy(p->name,f\_value[1]);

    strcpy(p->sex,f\_value[2]);

    p->age = atoi(f\_value[3]);

    p->score = atoi(f\_value[4]);

    offset++;//每次回调函数执行offset加1

    return 0;

}

void print\_student\_array(struct Student stu[],int len)

{

    int i;

    printf("number\tname\tsex\tage\tscore\n");

    for(i=0;i<len;i++)

    {

        printf("%d\t%s\t%s\t%d\t%d\n",stu[i].number,stu[i].name,stu[i].sex,stu[i].age,stu[i].score);

    }

}

int main(int argc, const char \*argv[])

{

    if(argc<2)

    {

        printf("too few argument\n");

        printf("Usage: %s <database>\n",argv[0]);

        return 0;

    }

    //char create[]="create table student(number integer PRIMARY KEY, name text, sex text, age integer, score integer);";

    /\*

    char \*insert[]={

        "insert into student values(10101,'liyuge','nan',18,99);",

        "insert into student values(10102,'tiancai','nan',18,98);",

        "insert into student values(10103,'hehe','nv',19,96);",

        "insert into student values(10104,'xixi','nv',19,94);",

        "insert into student values(10105,'haha','nan',18,90);"

    };

    int len = sizeof(insert)/sizeof(insert[0]);

    \*/

    char select[]="select \* from student;";

    //int i;

    int len = 5;

    char \*errmsg;

    sqlite3 \*db;

    struct Student stu[len];

    if(sqlite3\_open(argv[1],&db)!=0)

    {

        perror("cannot open database");

    }

    /\*

    //创建表student

    if(sqlite3\_exec(db,create,NULL,NULL,&errmsg)!=0)

    {

        printf("%s\n",errmsg);

    }

    \*/

    /\*

    //向表内插入学生信息

    for(i=0;i<len;i++)

    {

        if(sqlite3\_exec(db,insert[i],NULL,NULL,&errmsg)!=0)

        {

            printf("%s\n",errmsg);

        }

    }

    \*/

    //执行回调函数将表内数据存储在结构体数组内

    //注意第四个参数的用法

    if(sqlite3\_exec(db,select,callback,(void\*)&stu,&errmsg)!=0)

    {

        printf("%s\n",errmsg);

    }

    print\_student\_array(stu,len);//打印结构体数组内数据

    sqlite3\_close(db);

    return 0;

}

4、不使用回调函数操作数据库

    除了使用sqlite3\_exec()和回调函数操作SQLite3数据库，SQLite3还提供使用非回调函数的形式操作数据库。

    我们可以使用sqlite3\_get\_table()函数访问数据库内数据，该函数无需回调函数。

    函数sqlite3\_get\_table()

    所需头文件：#include<sqlite3.h>

    函数原型：

        int sqlite3\_get\_table(sqlite3 \*db, const char \*sql, char \*\*\*resultp, int \*nrow, int \*ncloumn, char \*\*errmsg)

    函数参数：

        db            需要操作的数据库的句柄指针

        sql            需要对数据库执行的SQLite语句

        resultp        存储SQLite语句执行结果的指针，必须使用地址传递。

        nrow        满足条件的记录数（行数），必须使用地址传递。

        ncloumn        每条记录的字段（列）的数目，必须使用地址传递。

        errmsg        存储错误信息指针，必须使用地址传递传参。若sqlite3\_exec()执行失败则会在堆区开辟内存空间存放错误信息，并将该空间首地址返回给errmsg

    函数返回值：

        成功：0

        失败：错误码（大于0）

如果sqlite3\_get\_table()执行成功，则会在堆区创建一个字符指针数组，存放执行结果，并将堆区空间首地址返回给第三个参数resultp。同时，将获取到的记录的个数（行数）返回给第四个参数nrow，获取到的字段数（列数）返回给第五个参数ncolumn。

注意resultp指向的字符指针数组内的结果是包含表头行的（但是nrow是不包含表头行的），因此在循环输出时需要注意是否需要输出表头信息，同时注意循环次数。

如果sqlite3\_get\_table()执行失败，则会在堆区开辟内存空间存放错误信息，并将该空间首地址返回给第六个参数errmsg。

示例：使用sqlite3\_get\_table()，获取表student内所有学生信息并输出

#include<stdio.h>

#include<sqlite3.h>

int main(int argc, const char \*argv[])

{

    if(argc<2)

    {

        printf("too few argument\n");

        printf("Usage: %s <database>\n",argv[0]);

        return 0;

    }

    char select[]="select \* from student;";

    int nrow,ncolumn;

    char \*\*result;

    char \*errmsg;

    int i,j;

    sqlite3 \*db;

    if(sqlite3\_open(argv[1],&db)!=0)

    {

        perror("cannot open database");

    }

    if(sqlite3\_get\_table(db,select,&result,&nrow,&ncolumn,&errmsg)!=0)

    {

        printf("%s\n",errmsg);

        return 0;

    }

    printf("nrow is %d, ncolumn is %d\n",nrow,ncolumn);//测试输出

    for(i=1;i<=nrow;i++)//result内结果包含表头信息，因此result[0]是表头信息，实际数据范围为result[1]~result[nrow]

    {

        for(j=0;j<ncolumn;j++)

        {

            printf("%s:%s\n",result[j],result[i\*ncolumn+j]);

        }

        printf("------------------\n");

    }

    printf("result is %p\n",result);

    sqlite3\_close(db);

    return 0;

}

————————————————

版权声明：本文为CSDN博主「nan\_lei」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/nan\_lei/article/details/84344003