sqlite 数据库(一)

`HQYJ-Linux网络编程`

sqlite 数据库(一)
1.数据库简介
1.1 常用数据库
1.2 数据库管理
2.SQLITE3 客户端与命令
2.1 SQLITE3 服务器与客户端
2.2 SQLITE3 客户端命令
2.2 SQL 语句
2.2.1 SQL 语句简介
2.2.3 创建数据库表
2.2.3 插入记录
2.2.4 查询记录
2.2.5 更新记录
2.2.6 删除记录
2.2.7 删除数据库表
3.SQLITE3 API 编程
3.1 打开数据库
3.2 关闭数据库
3.3 更新 SQL
3.4 查询 SQL

1.数据库简介

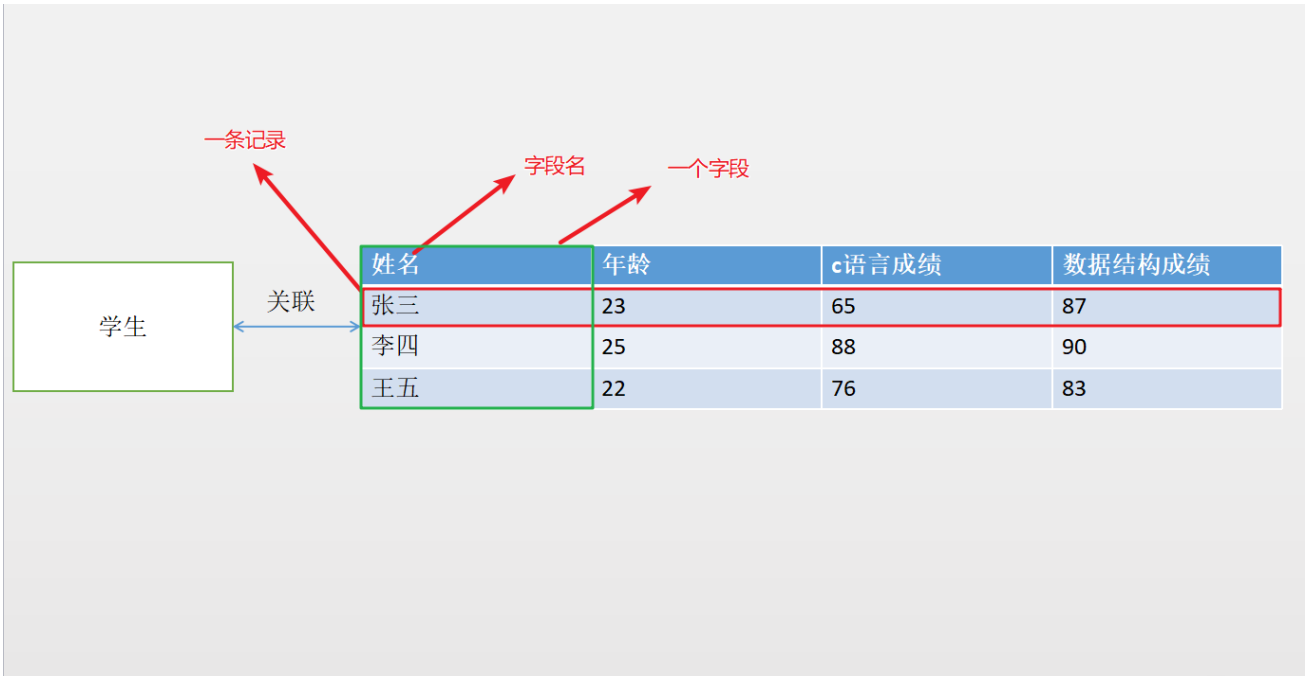
1.1 常用数据库

- **数据(Data)** 能够输入计算机并能被计算机程序识别和处理的信息集合

- **数据库(DataBase)** 是在数据库系统的管理和控制下,存放在存储介质上的数据集合
- **数据库管理系统(DataBase Mangement System) -- DBMS** 是数据库系统中对数据进行统一管理和控制的软件系统
  - 数据库定义功能（Data Definition)
  - 数据库操纵功能（Data Manpulation)
  - 数据库运行控制功能（Data Control)
  - 数据通信功能(Data Communication)
  - 支持存取海量数据(Mass Data)
- 常用数据库一般分为：
  - 大型数据库
    - **Oracle** 公司是最早开发关系数据库的厂商之一，其产品支持最广泛的操作系统平台
    - **DB2** 第一个具备网上功能的多媒体关系数据库管理系统,支持包括Linux在内的一系列平台
  - 中型数据库
    - **SQLServer** 是微软开发的数据库产品，主要支持windows平台
  - 小型数据库
    - **MYSQL** 是一个小型关系型数据管理系统，开发者为瑞典mysql AB公司，2005年被sun公司收购。开放 源码
    - **SQLite** 是专门针对嵌入式设备的数据库,属于关系型数据库,体积小，支持ACID（原子性、一致性、独立性及持久性 Atomicity、Consistency、Isolation、Durability）事物

1.2 数据库管理

- 目前大多数数据库都是 **关系型数据库**,表示一个实体在数据库关联到一张表,所以一般关系型数据库都是采用表结构的方式来管理数据
- 表中的每一行都是一条记录,具体结构如下图:



2.SQLITE3 客户端与命令

2.1 SQLITE3 服务器与客户端

- 一般数据库软件分为 **服务器** 与 **客户端**
  - 服务器负责从客户端获取指令,按照相应的指令的操作数据库
  - 客户端负责将用户的指令传输给服务器
- SQLITE3 数据库也有相应的服务器与客户端,作为用户主要与客户端进行交互

2.2 SQLITE3 客户端命令

- SQLITE3 常见命令如下:

命令	描述
.help	显示帮助信息
.quit	退出 SQLITE3
.database	显示当前打开的数据库文件

命令	描述
.tables	显示数据库中所有表名
.schema	查看表的结构

- 在 Linux 终端只需要输入 sqlite3 <*.db>,即可进入 sqlite3 客户端,并打开数据库文件

```
sqlite3 stduent.db //进入SQLITE3 客户端
```

- 当 student.db 存在时,则直接打开,如果不存在,则在创建表之后，创建数据库文件
- 可以使用 SQLITE3 支持的命令在客户端中操作.

2.2 SQL 语句

2.2.1 SQL 语句简介

- 在与数据库进行交互时,必须要使用标准的 **SQL 语句**,这主要是为了 **统一访问数据库的方式,提高通用性**.
- SQLITE3 数据库也是采用的标准 SQL 语句访问

2.2.3 创建数据库表

- 创建数据库表的标准 SQL 语句如下:

```
CREATE TABLE database_name.table_name(
   column1 datatype  PRIMARY KEY(one or more columns),
   column2 datatype,
   column3 datatype,
   .....
   columnN datatype,
);
```

- 在上述图片中出现的关键字具体解释如下:
  - **database_name** : 数据库名,一般可以省略
  - **table_name** : 数据库表名

- **columnN** : 表示第N个字段名
- **datatype** : 字段名类型
- **PRIMARY KEY** : 用于表名主键字段的关键字
- **IF NOT EXISTS** : 表示如果表存在,则不创建

- 当一个字段设置为 **PRIMARY KEY** ,一般为 id 字段,**对应的值是可以自增长的**
- 在 SQLITE3 支持常见字段的类型名如下:

存储类	描述
NULL	值是一个 NULL 值。
INTEGER	值是一个带符号的整数，根据值的大小存储在 1、2、3、4、6 或 8 字节中。
REAL	值是一个浮点值，存储为 8 字节的 IEEE 浮点数字。
TEXT	值是一个文本字符串，使用数据库编码（UTF-8、UTF-16BE 或 UTF-16LE）存储。
BLOB	值是一个 blob 数据，完全根据它的输入存储。

- Example : 创建一个 students 表,主要包括 学号(id) ,姓名(name),年龄(age),c语言成绩(score),主键为学号字段

```
root@farsight:/home/farsight/work/net/sqlite/code# sqlite3 student.db
SQLite version 3.7.2
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> create table students(id integer primary key,name text,age integer,score real);
sqlite> .table
students
sqlite> .schema
CREATE TABLE students(id integer primary key,name text,age integer,score real);
sqlite>
```

2.2.3 插入记录

- 在数据库表中插入一条记录的 SQL 如下:

```
INSERT INTO TABLE_NAME [(column1, column2, column3,...columnN)]
VALUES (value1, value2, value3,...valueN);
```

```
INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);
```

- 上述两种写法都是插入记录,不同的是第一种写法指定了字段名,而第2种写法省略
- Example : 向 student 表中插入新的记录.

```
root@farsight:/home/farsight/work/net/sqlite# sqlite3 student.db
SQLite version 3.7.2
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> create table students(id integer primary_key,name text integer,age integer,score real);
sqlite> .table
students
sqlite> .schema
CREATE TABLE students(id integer primary_key,name text integer,age integer,score real);
sqlite> insert into students values(1,'Tom',23,34.5);
sqlite> insert into students values(2,'Kevin',25,78.6);
sqlite> insert into students values(3,'Mark',37,65.5);
sqlite> insert into students values(4,'James',19,99.9);
sqlite>
```

插入4条记录

2.2.4 查询记录

- 在数据库表中查询一条记录的 SQL 如下:

```
SELECT column1, column2, columnN FROM table_name;
```

- 上述 SQL 语句主要用于查询指定字段的结果集合,如果需要 查询所有字段的结果集,则需要使用如下的 SQL 语句

```
SELECT * FROM table_name;
```

- Example : 查询 students 所有的记录

```
sqlite> .headers on
sqlite> .mode column
sqlite> select * from students;
id          name        age         score
----------  ----------  ----------  ----------
1           Tom         23          34.5
2           Kevin       25          78.6
3           Mark        37          65.5
4           James       19          99.9
sqlite>
```

显示字段名

按照列模式

- SQLITE 的 **where** 子句用于指定从一个表或多个表中获取数据的条件
  - 如果满足给定的条件，即为真（true）时，则从表中返回特定的值
  - 一般用于过滤记录,可以用于在 **SELECT** 语句、**UPDATE** 语句、**DELETE** 语句中
- where 语句具体的语法如下:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

- 上述语法中 where 后面需要接条件,对应条件表达式可以使用 **比较或逻辑运算符指定条件**,

比如 <、>、=、LIKE、NOT 等等

- Example : 查找 students 表中年龄小于25岁的学员

```
sqlite> select * from students;
id          name        age         score
----------  ----------  ----------  ----------
1           Tom         23          34.5
2           Kevin       25          78.6
3           Mark        37          65.5
4           James       19          99.9
sqlite> select * from students where age < 25;
id          name        age         score
----------  ----------  ----------  ----------
1           Tom         23          34.5
4           James       19          99.9
sqlite>
```

查找 students 表中年龄小于 25 岁的学生

- Example : 查找 students 表中年龄小于25并且分数大于60的学员

```
sqlite> select * from students;
id          name        age         score
----------  ----------  ----------  ----------
1           Tom         23          34.5
2           Kevin       25          78.6
3           Mark        37          65.5
4           James       19          99.9
sqlite> select * from students where age < 25;
id          name        age         score
----------  ----------  ----------  ----------
1           Tom         23          34.5
4           James       19          99.9
sqlite> select * from students where age < 25 and score > 60;
id          name        age         score
----------  ----------  ----------  ----------
4           James       19          99.9
sqlite>
```

2.2.5 更新记录

- SQLITE 的 **UPDATE** 查询用于修改表中已有的记录,可以使用带有 WHERE 子句的 UPDATE 查询来更新选定行,否则所有的行都会被更新,具体语法如下:

```
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

- table_name : 表示数据库表的名字
- SET : 指定需要修改的字段以及字段的值
- WHERE : 指定条件,属于可选的

- Example : 更新 id = 2 的名字为 ben

```
sqlite> update students set name='ben' where id=2;
sqlite> select * from students;
id          name        age         score
----------  ----------  ----------  ----------
1           Tom         23          34.5
2           ben         25          78.6
3           Mark        37          65.5
4           James       19          99.9
sqlite>
```

修改 id = 2 的名字为 'ben'

更新之后的记录

2.2.6 删除记录

- SQLITE 使用 **delete** 字段,具体的语法如下:

```
DELETE FROM table_name
WHERE [condition];
```

- 在语法上可以使用 WHERE 子句来指定删除的条件,如果条件为空 则删除表中所有记录
- Example : 删除 name = 'Mark' 的记录

```
sqlite> delete from students where name = 'Mark'
   ...> ;
sqlite> select * from students;
id          name        age         score
----------  ----------  ----------  ----------
1           Tom         23          34.5
2           ben         25          78.6
4           James       19          99.9
sqlite>
```

删除 name = 'Mark' 的记录

更新之后的结果

2.2.7 删除数据库表

- SQLITE3 数据库表需要使用 **DROP** 语句,具体语法如下:

```
DROP TABLE database_name.table_name;
```

- 在上述语法中 数据库名称不是必须的,可以省略不写

3.SQLITE3 API 编程

- SLIQTE3 数据库针对不同的语言提供了相应的编程接口,下面介绍的是关于 c 语言接口

3.1 打开数据库

- 在 SQLITE3 中打开数据库需要调用 **sqlite3_open** 函数,具体的函数原型如下:

```
int sqlite3_open(
  const char *filename,   /* Database filename (UTF-8) */
  sqlite3 **ppDb          /* OUT: SQLite db handle */
);
@param filename : 数据库路径名
@param ppDb : sqlite3 对象指针的地址,是输出参数

@return :
    成功 : SQLITE_OK
    失败 : 返回错误码
```

- **sqlite3** 用于描述一个数据库连接,本质上是一个结构体,具体原型如下:

```
typedef struct sqlite3 sqlite3;
```

- 相应的错误码可以通过 **sqlite3_errmsg()** 函数来打印,具体原型如下:

```
const char *sqlite3_errmsg(sqlite3*db);
@param db : sqlite3 对象指针
@return :
    返回错误码字符串描述
```

3.2 关闭数据库

- 关闭数据库需要调用 **sqlite3_close()** 函数,具体原型如下:

```
int sqlite3_close(sqlite3*db);
@param db : sqlite3 对象的指针
@return :
    成功 : SQLITE_OK
    失败 : 返回错误码
```

3.3 更新 SQL

- 更新 SQL 的 API 是 **执行更新数据库的 SQL 语句**,这里的更新数据库的 SQL 语句包含所有更改了数据库的 SQL 语句,不仅仅包含 UPDATE
- 在 c 语言中提供了 **sqlite3_exec()** 函数用于执行更新数据库的 SQL 语句,具体原型如下；

```
int sqlite3_exec(
  sqlite3*db,
  const char *sql,
  int (*callback)(void*,int,char**,char**),
  void *arg,
  char **errmsg,
);
@param db : SQLITE3 对象指针
@param sql : 执行的 SQL 字符串
@param callback : 回调函数,没有则为 NULL
@param arg : 回调函数的参数,没有则为 NULL
@param errmsg : 错误信息

@return :
    成功 : SQLITE_OK
    失败 : 返回错误码
```

- 在使用时需要注意，如果使用了 errmsg ，需要使用 **sqlite3_free()** 释放空间

3.4 查询 SQL

- 在 c 语言中查询类的 SQL 语句的执行需要调用 **sqlite3_get_table()**,具体原型如下:

```
int sqlite3_get_table(
  sqlite3 *db,           /*数据库对象的指针*/
  const char *zSql,      /*查询 SQL 语句*/
  char ***pazResult,     /*结果集的指针 */
  int *pnRow,            /*结果集的行数 */
  int *pnColumn,         /*结果集的列数 */
  char **pzErrmsg        /* 错误信息 */
);

@return :
    成功 : 返回 SQLITE3_OK
    失败 : 返回错误码
```

- 查询的操作与其他操作不同的地方是 **查询数据库都会产生结果集合**

- 查询的结果空间是由 sqlite3_get_table() 函数来分配,在使用完成之后,则需要调用 **sqlite3_free_table()** 函数来释放结果集空间

```
1.  void sqlite3_free_table(char **result);
2.  @param result : 结果集合的指针
```