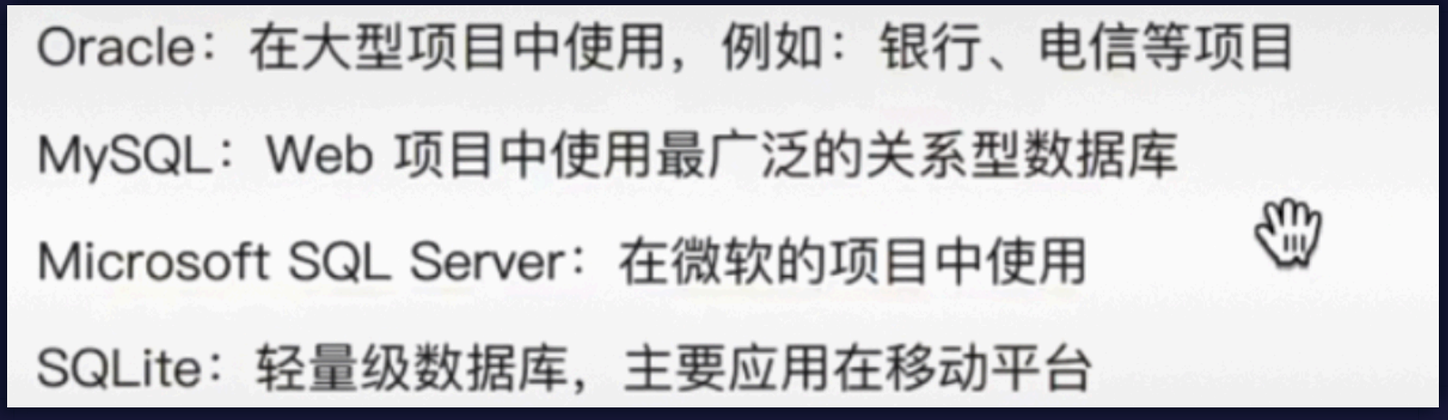
1.数据库

1.1数据库的简介

简单的说，数据库（database）就是一个存放数据的仓库，这个仓库是按照一定的数据结构（数据结构是指数据的组织形式或数据之间的联系）来组织、存储的，我们可以通过数据提供的多种方法来管理数据库里的数据。



1.2seqlite3数据库的优缺点

SQLite 是一款轻量级的关系型数据库，它的优点如下(sqlite3官网):

1. 体积小：SQLite 数据库文件只有几百KB大小，可以在硬盘上存储更多的数据。
2. 无服务器：SQLite 数据库不需要安装服务器，可以直接在本地使用，方便快捷。
3. 功能强大：SQLite 支持标准的 SQL 语法，可以实现增删改查等复杂操作。
4. 性能好：SQLite 数据库的性能比其他数据库要好，可以满足大多数应用的需求。
5. 免费开源：SQLite 是开源的，可以，不需要支付任何费用。

1.3数据库准备工作

1.3.1数据库的安装

Ubuntu上安装sqlite3数据库：`sudo apt-get install sqlite3 libsqlite3-dev sqlitebrowser`

1.3.2数据库的创建

可以使用sqlite3 stu.db来创建一个数据库文件，数据库文件的后缀一般为.db结尾

```
linux@ubuntu:~/work/database$ sqlite3  stu.db
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
sqlite>
```

1.3.3数据库命令行常见操作

`sqlite>.help` #查看帮助

`sqlite>.database` #列出数据库的路径及文件名

`sqlite>.open new.db` #关闭原有数据库，打开新的数据库

`sqlite>.table` #列出表名字

`sqlite>.schema` #列出表的结构

`sqlite>.quite` #退出数据库

`sqlite>.headers on|off` #打开或者关闭表头

`sqlite>.mode column` #左对齐列（可视化性强）

`sqlite>.width NUM1 NUM2` #设置显示时候列宽（和上述.mode结合使用）

`sqlite>.show` #显示当前设置变量的数值

1.4数据库的SQL语句

1.4.1创建表

```
CREATE TABLE stu(id INT,name CHAR,score FLOAT);
```

注：

INT等价于INTEGER

CHAR等价于TEXT

```
CREATE TABLE IF NOT EXISTS stu(id INT,name CHAR,score FLOAT);
```

如果表不存在就创建，存在就不创建了

1.4.2插入数据

```
INSERT INTO stu VALUES(1101,'zhangsan',88.9);
```

注：

在sqlite3中不分字符和字符串统认为是字符串，字符串需要加单引号或者双引号

1.4.3查询表

```
SELECT * FROM stu;
```
#查询stu表中所有的数据

```
SELECT name,score FROM stu;
```
#查询表中name和score的列

```
SELECT * FROM stu WHERE id=1101;
```
#查询id为1101的记录

```
SELECT * FROM stu WHERE score=99 AND name='zhangsan';
```

#查询名字为zhangsan和score为99的记录

```
SELECT * FROM stu WHERE score=99 OR name='zhangsan';
```

#查询名字为zhangsan或者score为99的记录

```
SELECT * FROM stu ORDER BY score ASC;
```

#查询所有的数据按照升序排序

```
SELECT * FROM stu ORDER BY score DESC;
```

#查询所有的数据按照降序排序

1.4.4更新表

```
UPDATE stu SET name='wangwu' WHERE id=1103;
```

#将id为1103的记录name字段改为wangwu

```
UPDATE stu SET name='tianqi',score=100 WHERE name='wangwu';
```

#将name为wangwu的记录name和score更新为tianqi和100

1.4.5删除表中记录

```
DELETE FROM stu WHERE id=1103;
```

#从表中删除id为1103的记录

1.4.6删除表

```
DROP TABLE stu;
```
#删除stu表

1.4.7给表添加列

```
ALTER TABLE stu ADD COLUMN sex CHAR;
```

#给表插入sex列

1.4.8删除表中的列

1. 利用原表有用信息生成临时表

   ```
   CREATE TABLE tmp AS SELECT id,name,score from stu;
   ```

2. 删除旧表

   ```
   DROP TABLE stu;
   ```

3. 临时表更新名字

   ```
   ALTER TABLE tmp RNAME TO stu;
   ```

> **sqlite3**中不支持删除列，可以先将原表中有用的信息生成一个临时表
>
> 删除旧表，将临时表改名为原表

1.4.9主键 PRIMARY KEY

```
CREATE TABLE IF NOT EXISTS stu(id INT PRIMARY KEY AUTOINCREMENT,name char);
```

> #在建表时 可以指定一个字段为主键 指定之后 以后再插入时 主键不允许冲突

1.4.10联接表

```
sqlite> select * from stu_base;
1           zhangsan    m
2           lisi        m
3           tianqi      w
sqlite> select * from stu_info;
1           beijing     2021
2           shanghai    2022
3           chengdu     2023
```

比如上述的数据库中有两张表，有一个相同的ID此时我们就可以使用联接表的功能了。

```
select stu_base.id,name,sex,addr,year from stu_base,stu_info where
stu_base.id=stu_info.id;
```

```
sqlite> select stu_base.id,name,sex,addr,year from stu_base,stu_info where stu_base.id=stu_info.id;
1           zhangsan    m           beijing     2021
2           lisi        m           shanghai    2022
3           tianqi      w           chengdu     2023
sqlite>
```

1.4.11虚表的创建

在使用上述的联接表的时候，查询命令非常的复杂，可以创建一个虚表区查看

相关信息，虚表只用于查看，但不能修改。

```
CREATE VIEW 表名 AS 联接表命令
```

```
create view stu as select stu_base.id,name,sex,addr,year from stu_base,stu_info where stu_base.id=stu_info.id;
```

```
sqlite> create view stu as select stu_base.id,name,sex,addr,year from stu_base,stu_info
 where stu_base.id=stu_info.id;
sqlite> select * from stu;
1           zhangsan    m           beijing     2021
2           lisi        m           shanghai    2022
3           tianqi      w           chengdu     2023
```

1.4.12数据库中获取时间函数

`date(‘now’)` #获取当前日期

`date(‘now’,'+2 month','+1 day')` #获取当前日期

`time('now')` #获取当前时间，没有加时区间

`time('now','+8 hour')`#获取当前时间，加上8小时（东8区）

`datetime('now','localtime')`#获取本地日期和时间

例如：

```
insert into stu_log values(date('now'),time('now','+8 hour'));
```

#将当前的日期和时间插入到stu_log的表中

1.4.13触发器的使用

```
CREATE TRIGGER  触发器名  [BEFORE|AFTER] [INSERT|UPDATE|DELETE]
ON  <tableName >  //dbo代表该表的所有者
BEGIN
--do something
END ;
```

```
sqlite> .schema
CREATE TABLE stu_base(id int,name text,sex char);
CREATE TABLE stu_info(id int,addr text,year int);
CREATE TABLE stu_log(id INT,date TEXT,time TEXT,op TEXT);
```

```
create trigger stu_insert afrer insert on stu_base begin insert into stu_log v
alues(1,date('now'),time('now','+8 hour'),'insert'); end;
```

#添加对stu_base表insert的触发器，只要添加就将时间写在stu_log表中

```
create trigger stu_update after update on stu_base begin insert into stu_log va
lues(2,date('now'),time('now','+8 hour'),'update'); end;
```

#添加对stu_base表update的触发器，只要更新就将时间写在stu_log表中

```
create trigger stu_delete after delete on stu_base begin insert into stu_log va
lues(3,date('now'),time('now','+8 hour'),'delete'); end;
```

#添加对stu_base表delete的触发器，只要删除数据就将时间写在stu_log表中

当对stu_base表进行操作的时候就可以看到如下的log日志信息

```
sqlite> select * from stu_log;
1|2023-08-15|16:44:37|insert
2|2023-08-15|16:45:19|update
3|2023-08-15|16:45:39|delete
```

1.5数据库事务操作

事务（Transaction）是一个对数据库执行工作单元。事务（Transaction）是以逻辑顺序完成的工作单位或序列，可以是由用户手动操作完成，也可以是由某种数据库程序自动完成。

事务（Transaction）是指一个或多个更改数据库的扩展。例如，如果您正在创建一个记录或者更新一个记录或者从表中删除一个记录，那么您正在该表上执行事务。重要的是要控制事务以确保数据的完整性和处理数据库错误。

实际上，您可以把许多的 SQLite 查询联合成一组，把所有这些放在一起作为事务的一部分进行执行。

事务（Transaction）具有以下四个标准属性，通常根据首字母缩写为 ACID:

- **原子性（Atomicity）：**确保工作单位内的所有操作都成功完成，否则，事务会在出现故障时终止，之前的操作也会回滚到以前的状态。
- **一致性（Consistency）：**确保数据库在成功提交的事务上正确地改变状态。
- **隔离性（Isolation）：**使事务操作相互独立和透明。
- **持久性（Durability）：**确保已提交事务的结果或效果在系统发生故障的情况下仍然存在。

`sqlite>begin;` #开始事务操作

`sqlite>drop table stu;` #具体想做内容

`sqlite>rollback;` #撤销

`sqlite>begin;` #开始事务操作

`sqlite>drop table stu;` #具体想做内容

`sqlite>commit` #提交

1.6数据库的常用API

```
#include <sqlite3.h>
int sqlite3_open(
  const char *filename,   /* Database filename (UTF-8) */
  sqlite3 **ppDb          /* OUT: SQLite db handle */
);
功能: 打开数据库
参数:
    @filename:数据库的路径及名字
    @ppDb:数据库的句柄
返回值:  成功返回SQLITE_OK,失败返回error code

const char *sqlite3_errmsg(sqlite3*db);
功能: 获取错误信息
参数:
    @db:数据库指针
返回值: 返回错误信息

int sqlite3_close(sqlite3*db);
功能: 关闭数据库
参数:
    @db:数据库的句柄
返回值: 成功返回SQLITE_OK, 失败返回error code
```

```
int sqlite3_exec(
  sqlite3*db,                                  /* An open database */
  const char *sql,                            /* SQL to be evaluated */
  int (*callback)(void*,int,char**,char**),  /* Callback function */
  void *arg,                                     /* 1st argument to callback */
  char **errmsg                               /* Error msg written here */
);
功能: 执行sql语句
参数:
    @db:打开的数据库
  @sql:sql语句
    @callback:回调函数 （只有查询是需要）
        int (*callback)(void*arg,int column,char**f_value,char**f_name)
    @arg:给回调函数第一个参数传参
    @errmsg:错误信息
返回值: 成功返回SQLITE_OK, 失败返回error code
void sqlite3_free(void*)
功能: 将errmsg占用的内存空间释放

int sqlite3_get_table(
  sqlite3 *db,          /* An open database */
  const char *zSql,     /* SQL to be evaluated */
  char ***pazResult,    /* Results of the query */
  int *pnRow,           /* Number of result rows written here */
  int *pnColumn,        /* Number of result columns written here */
  char **pzErrmsg       /* Error msg written here */
);
功能: 查询数据库函数
参数:
  @db:打开的数据库
  @sql:sql语句
    @pazResult:查询到的结果
    @pnRow:结果的行
    @pnColumn:结果的列
  @pzErrmsg:错误信息
返回值: 成功返回SQLITE_OK, 失败返回error code
void sqlite3_free_table(char **result);
功能: 释放结果所占的内存空间
参数:
    @result:结果的首地址
返回值: 无
```

1.7数据库编程实例

```c
#include <head.h>
#include <sqlite3.h>
#define DB_NAME "stu.db"
typedef struct stu_info {
    int id;
    char name[20];
    int age;
} stu_t;

enum {
    CALLBACK_MODE,
    GETTABLE_MODE,
};

int sqlite3_init_table(sqlite3** pdb)
{
    int ret;
    // 1.打开数据库
    if ((ret = sqlite3_open(DB_NAME, pdb)) != SQLITE_OK) {
        printf("sqlite3_open error: %s\n", sqlite3_errmsg(*pdb));
        return ret;
    }
    // 2.在数据库中创建表 PRIMARY KEY AUTOINCREMENT
    char sql[] = "CREATE TABLE IF NOT EXISTS stu(id integer PRIMARY KEY,name text,age
integer);";
    if ((ret = sqlite3_exec(*pdb, sql, NULL, NULL, NULL)) != SQLITE_OK) {
        printf("sqlite3_exec error: %s\n", sqlite3_errmsg(*pdb));
        return ret;
    }
    return SQLITE_OK;
}
void input_sql_data(stu_t* info)
{
    int ret;
retry:
    printf("input (id,name,age) >");
    ret = scanf("%d %s %d", &info->id, info->name, &info->age);
    if (ret != 3) {
        printf("input error,try again\n");
        while (getchar() != '\n')
            ;
        goto retry;
    }
}
int sqlite3_insert_data(sqlite3* pdb)
{
    int ret;
    char sql[100] = { 0 };
    stu_t info;
    input_sql_data(&info);
    snprintf(sql, sizeof(sql), "INSERT INTO stu VALUES(%d,'%s',%d);", info.id,
info.name, info.age);
```

```c
    if ((ret = sqlite3_exec(pdb, sql, NULL, NULL, NULL)) != SQLITE_OK) {
        printf("sqlite3_exec error: %s\n", sqlite3_errmsg(pdb));
        return ret;
    }
    return SQLITE_OK;
}
//有一项数据callback被回调一次
int check_callback(void* arg, int column, char** f_value, char** f_name)
{
    if (*(int*)arg == 1) {
        *(int*)arg = 0;
        for (int i = 0; i < column; i++) {
            printf("%-10s", f_name[i]);
        }
        putchar(10);
    }
    for (int i = 0; i < column; i++) {
        printf("%-10s", f_value[i]);
    }
    putchar(10);
    return SQLITE_OK; // 这里一定要返回OK，否则会认为回调错误
}
void table_show(char** pazResult, int row, int column)
{
    for (int i = 0; i <= row; i++) { //这里的row是不包括标题行的，所以循环多走一次
        for (int j = 0; j < column; j++) {
            printf("%-10s", pazResult[i * column + j]);
        }
        putchar(10);
    }
}
int sqlite3_check_data(sqlite3* pdb, int mode)
{
    int ret, flags = 1;
    char sql[] = "SELECT * FROM stu;";
    switch (mode) {
    case CALLBACK_MODE:
        if ((ret = sqlite3_exec(pdb, sql, check_callback, &flags, NULL)) != SQLITE_OK)
{
            printf("sqlite3_exec select error:ret = %d,errmsg = %s\n", ret,
sqlite3_errmsg(pdb));
            return ret;
        }
        break;
    case GETTABLE_MODE: {
        char** pazResult;
        int row, column;
        if ((ret = sqlite3_get_table(pdb, sql, &pazResult, &row, &column, NULL)) !=
SQLITE_OK) {
            printf("sqlite3_get_table error ret = %d, errmsg = %s\n", ret,
sqlite3_errmsg(pdb));
            sqlite3_free_table(pazResult);
```

```
            return ret;
        }
        table_show(pazResult, row, column);
        sqlite3_free_table(pazResult);
    } break;
    }
    return SQLITE_OK;
}

int main(int argc, const char* argv[])
{
    sqlite3* pdb;
    int ret;
    // 1.初始化数据库
    if ((ret = sqlite3_init_table(&pdb)) != SQLITE_OK) {
        return ret;
    }
    // 2.插学生入数据
    for (int i = 0; i < 3; i++)
        if ((ret = sqlite3_insert_data(pdb)) != SQLITE_OK) {
            return ret;
        }
    // 3.查询数据
    if ((ret = sqlite3_check_data(pdb, CALLBACK_MODE)) != SQLITE_OK) {
        return ret;
    }
    printf("----------------------\n");
    if ((ret = sqlite3_check_data(pdb, GETTABLE_MODE)) != SQLITE_OK) {
        return ret;
    }
    // 6.关闭数据库
    if ((ret = sqlite3_close(pdb)) != SQLITE_OK) {
        printf("sqlite3_close error: %s\n", sqlite3_errmsg(pdb));
        return ret;
    }
    return 0;
}
```