实验0：磁盘管理器实现

邹兆年，哈尔滨工业大学计算学部，znzou@hit.edu.cn

https://gitee.com/HIT-DB/hit-db-class-rucbase-lab

一、实验目的

1. 掌握Rucbase开发环境的配置方法。
2. 掌握Rucbase磁盘管理器的实现方法。
3. 掌握Rucbase的测试方法。

二、相关知识

1. 数据库管理系统的系统架构
2. 文件操作
3. 文件页面访问

三、相关技能

1. Git的使用
2. 基本Shell命令的使用

四、相关软件

完成该实验，你需要以下软件和库：

- `git`
- `flex`
- `bison`
- `cmake`
- `gcc`
- `gdb`
- `graphviz`
- `libreadline-dev`

你可以使用任何你熟悉的代码编辑器，如 `vim`、`emacs`等。建议使用Visual Studio Code (VSCode)。

如果你使用Windows操作系统，你还需要一个运行Linux操作系统的虚拟机。为了便于开发，建议在虚拟机上运行`ssh`服务，开发时可以在Windows上直接 `ssh`连接虚拟机上的Linux主机，比较方便。

在Linux上安装`flex`和`bison`的方法如下：

```
sudo apt-get install flex bison
```

在Linux上安装`libreadline-dev`的方法如下：

```
sudo apt-get install libreadline-dev
```

五、实验内容

本实验包括3项任务。

任务1：开发环境配置

（1）下载Rucbase代码框架

创建目录`rucbase-lab`

```
mkdir rucbase-lab
```

进入目录`rucbase-lab`

```
cd rucbase-lab
```

将rucbase-lab仓库中全部文件克隆到该目录，请注意这里的命令选项`--recursive`。

```
git clone --recursive https://gitee.com/HIT-DB/hit-db-class-rucbase-lab.git .
```

（2）安装GoogleTest

在`rucbase-lab`目录下，进入目录`deps/googletest`

```
cd deps/googletest
```

如果你的`deps/googletest`目录下没有任何文件，说明你在克隆仓库的时候忘记使用`--recursive`选项。请执行以下命令下载GoogleTest的相关源文件。

```
git submodule init
git submodule update
```

创建目录`build`

```
mkdir build
```

进入目录`build`

```
cd build
```

创建Makefile

```
cmake ..
```

编译GoogleTest

```
make
```

安装GoogleTest

```
sudo make install
```

（3）编译Rucbase服务端程序

在`rucbase-lab`目录下，创建目录`build`

```
mkdir build
```

进入目录`build`

```
cd build
```

创建Makefile

```
cmake ..
```

编译Rucbase服务端程序`rmdb`

```
make
```

（4）编译Rucbase客户端程序

在`rucbase-lab`目录下，进入目录`rucbase_client`

```
cd rucbase_client
```

创建目录`build`

```
mkdir build
```

进入目录`build`

```
cd build
```

创建Makefile

```
cmake ..
```

编译Rucbase客户端程序`rucbase_client`

```
make
```

（5）运行Rucbase服务端程序

在`rucbase-lab`目录下，进入目录`build`

```
cd build
```

编译Rucbase服务端程序`rmdb`

```
make
```

运行`rmdb`

```
./bin/rmdb <database_name>
```

这里的参数`<database_name>`代表数据库的名字，你可以给数据库取一个你喜欢的名字。

退出服务端程序的方法是`Ctrl+C`。

（6）运行Rucbase客户端程序

在`rucbase-lab`目录下，进入目录`rucbase_client/build`

```
cd rucbase_client/build
```

编译Rucbase客户端程序`rucbase_client`

```
make
```

运行`rucbase_client`

```
./rucbase_client
```

可以运行多个客户端程序。

（7）使用Rucbase客户端程序

在`rucbase_client`中执行命令和SQL语句并获得返回结果。

Rucbase的一些基本命令如下（命令必须以`;`结尾）：

- `show tables`：列出数据库中全部关系。

- `desc <table_name>`：显示关系`<table_name>`的模式，这里`<table_name>`是关系的名字。
- `exit`：退出客户端程序。

此时你还没有为编写任何代码，在`rucbase_client`中执行SQL语句会出现程序运行错误。

任务2：磁盘管理器实现

补全`DiskManager`类，实现磁盘管理器的下列功能。

1. 创建、删除、打开、关闭文件。
2. 分配、读、写文件页面。

具体完成如下任务。

（1）阅读代码

阅读`src/storage`目录下的代码。

- `src/storage/page.h`
- `src/storage/disk_manager.h`
- `src/storage/disk_manager.cpp`

理解`PageId`结构体的设计，并回答下列问题：

1. `PageId::fd`和`PageId::page_no`的含义是什么？

理解`DiskManager`类的设计，并回答下列问题：

1. `path2fd_`的作用是什么？
2. `fd2path_`的作用是什么？
3. `path2fd_`和`fd2path_`的关系是什么？
4. `fd2pageno_`的作用是什么？

自学Linux操作系统`/usr/include/unistd.h`文件声明的`read`、`write`、`open`、`close`、`unlink`等函数。

（2）实现`DiskManager::create_file`函数

函数声明：

```
void DiskManager::create_file(const std::string &path);
```

功能：创建文件。参数的含义参考代码注释。

实现：参考代码注释。基本实现方法是调用`open`函数，使用`O_CREAT`模式。注意不能重复创建相同文件，否则抛出异常（阅读`src/test/storage/disk_manager_test.cpp`文件，了解这种情况下抛出异常的类型）。

（3）实现`DiskManager::destroy_file`函数

函数声明：

```
void DiskManager::destroy_file(const std::string &path);
```

功能：删除文件。参数的含义参考代码注释。

实现：参考代码注释。基本实现方法是调用`unlink`函数。注意不能删除未关闭的文件，否则抛出异常（阅读`src/test/storage/disk_manager_test.cpp`文件，了解这种情况下抛出异常的类型）。

（4）实现`DiskManager::open_file`函数

函数声明：

```
int DiskManager::open_file(const std::string &path);
```

功能：打开文件。参数和返回值的含义参考代码注释。

实现：参考代码注释。基本实现方法是调用`open`函数，使用`O_RDWR`模式，然后更新打开文件的列表，即`path2fd_`和`fd2path_`。注意不能重复打开相同文件，否则抛出异常（阅读`src/test/storage/disk_manager_test.cpp`文件，了解这种情况下抛出异常的类型）。

（5）实现`DiskManager::close_file`函数

函数声明：

```
void DiskManager::close_file(int fd);
```

功能：关闭文件。参数的含义参考代码注释。

实现：参考代码注释。基本实现方法是调用`close`函数，然后更新打开文件的列表，即`path2fd_`和`fd2path_`。注意不能重复打开相同文件，否则抛出异常（阅读`src/test/storage/disk_manager_test.cpp`文件，了解这种情况下抛出异常的类型）。

（6）实现`DiskManager::read_page`函数

函数声明：

```
void DiskManager::read_page(int fd, page_id_t page_no, char *offset, int num_bytes);
```

功能：读页面。参数的含义参考代码注释。

实现：参考代码注释。基本实现逻辑如下：

1. 根据`page_no`和`PAGE_SIZE`计算指定页面在文件中的偏移量。
2. 调用`read`函数。

一般情况下，`num_bytes = PAGE_SIZE`。但也可以是`num_bytes < PAGE_SIZE`，如只读取页头。

（7）实现`DiskManager::write_page`函数

函数声明：

```
void DiskManager::write_page(int fd, page_id_t page_no, const char *offset, int
num_bytes);
```

功能：写页面。参数的含义参考代码注释。

实现：参考代码注释。基本实现逻辑如下：

1. 根据`page_no`和`PAGE_SIZE`计算指定页面在文件中的偏移量。
2. 调用`write`函数。

一般情况下，`num_bytes = PAGE_SIZE`。但也可以是`num_bytes < PAGE_SIZE`，如只读取页头。

（8）注意事项

不允许修改任何公有函数的声明。

任务3：磁盘管理器测试

单元测试代码在文件`src/test/storage/disk_manager_test.cpp`中。

执行下列命令，进行单元测试。

```
cd build
make disk_manager_test
./bin/disk_manager_test
```

（1）单元测试

GoogleTest是谷歌公司开发的一种单元测试工具。Rucbase-lab提供了一些GoogleTest单元测试，可以测试代码正确性（但是有可能无法覆盖全部特殊情况）。

以磁盘管理器单元测试为例，执行步骤如下：

在`rucbase-lab`目录下，进入`build`目录

```
cd build
```

构建单元测试程序`disk_manager_test`

```
make disk_manager_test
```

执行单元测试程序`disk_manager_test`

```
./bin/disk_manager_test
```

阅读`disk_manager_test`的输出结果，了解单元测试的通过情况。

（2）集成测试

1. 编译并运行Rucabase的服务端程序和客户端程序。
2. 在客户端程序执行命令和SQL语句，并观察执行结果。

下面给出2个测试用例。（只有完成了全部实验才能执行）

测试用例1

```
create table student (id int, name char(32), major char(32));
create index student (id);
create table grade (course char(32), student_id int, score float);
create index grade (student_id);

show tables;
desc student;

begin;
insert into student values (1, 'Tom', 'Computer Science');
insert into student values (2, 'Jerry', 'Computer Science');
insert into student values (3, 'Jack', 'Electrical Engineering');
commit;

begin;
select * from student where id>=1;
update student set major = 'Electrical Engineering' where id = 2;
select * from student where id>=1;
delete from student where name = 'Jack';
select * from student where id>=1;
commit;

begin;
insert into grade values ('Data Structure', 1, 90.0);
insert into grade values ('Data Structure', 2, 95.0);
insert into grade values ('Calculus', 2, 82.0);
insert into grade values ('Calculus', 1, 88.5);
abort;

begin;
insert into grade values ('Data Structure', 1, 90.0);
insert into grade values ('Data Structure', 2, 95.0);
insert into grade values ('Calculus', 2, 82.0);
insert into grade values ('Calculus', 1, 88.5);
commit;

select * from student, grade;
select id, name, major, course, score from student, grade where student.id =
grade.student_id;
```

```
select id, name, major, course, score from student join grade where student.id =
grade.student_id;

drop index student (id);
desc student;

drop table student;
drop table grade;
show tables;

exit;
```

测试用例2

```
create table test (id int, val int); // 创建关系

show tables; // 列出数据库中全部关系，检查关系test是否建立成功
desc test; // 列出test的关系模式

insert into test values (1, 111); // 插入元组
select * from test; // 检查元组是否插入成功
insert into test values (2, 222);
select * from test;
insert into test values (3, 333);
select * from test;

select val, id from test; // 检查投影查询
select * from test where id = 1; // 检查点查询
select * from test where id > 1; // 检查区间查询
select * from test where id = 1 and val = 111; // 检查复合查询条件

update test set val = 444 where id = 1;
select * from test;
update test set id = 4 where val = 444;
select * from test;
update test set id = 1 where id = 4;
select * from test;
update test set val = 111 where val = 444;
select * from test;
```