# 实验2：记录管理器实现

**作者**：邹兆年  
**单位**：哈尔滨工业大学计算学部  
**邮箱**：[znzou@hit.edu.cn](mailto:znzou@hit.edu.cn)  
**项目地址**：<https://gitee.com/HIT-DB/hit-db-class-rucbase-lab>

## 一、实验目的

掌握Rucbase记录管理器的实现方法。

## 二、相关知识

1. 分槽页面布局
2. 堆文件组织

## 三、实验内容

本实验包括2项任务。

### 任务1：记录操作实现

补全 RMFileHandle 类，实现文件记录的获取、插入、删除和修改操作。

每个 RMFileHandle 对象对应一个文件，当 RMManager 执行打开文件操作时，会创建一个指向 RMFileHandle 对象的指针。RMFileHandle 类的接口如下：

class RmFileHandle {  
public:  
 RmFileHandle(DiskManager \*disk\_manager, BufferPoolManager \*buffer\_pool\_manager, int fd);  
   
 // 不考虑事务的记录操作（事务将在后续实验使用）  
 std::unique\_ptr<RmRecord> get\_record(const Rid &rid, Context \*context) const;  
 Rid insert\_record(char \*buf, Context \*context);  
 void delete\_record(const Rid &rid, Context \*context);  
 void update\_record(const Rid &rid, char \*buf, Context \*context);  
   
 // 辅助函数  
 RmPageHandle create\_new\_page\_handle();  
 RmPageHandle fetch\_page\_handle(int page\_no) const;  
 RmPageHandle create\_page\_handle();  
 void release\_page\_handle(RmPageHandle &page\_handle);  
};

具体完成如下任务：

#### （1）阅读代码

阅读 src/record 目录下的代码：

* src/record/rm\_defs.h
* src/record/rm\_manager.h
* src/record/rm\_file\_handle.h
* src/record/rm\_file\_handle.cpp

理解 RMManager 类、RMFileHandle 类和 RMPageHandle 类的设计。

* RMManager 类提供了创建、打开、关闭、删除记录文件的接口，其内部实现调用了实验1实现的 DiskManager 和 BufferPoolManager 类的接口。
* Rid 类定义了记录ID。
* RmReocrd 类定义了记录存储结构。
* RMPageHandle 类定义了分槽页面布局。
* RMFileHandle 类定义了文件记录的获取、插入、删除和修改操作。

参考下图理清重要概念之间的关系。

#### （2）实现 RmFileHandle::get\_record 函数

**函数声明**：

std::unique\_ptr<RmRecord> RmFileHandle::get\_record(const Rid& rid, Context\* context) const;

**功能**：获取一条记录。参数和返回值的含义参考代码注释。

**实现**：参考代码注释。

#### （3）实现 RmFileHandle::insert\_record 函数

**函数声明**：

Rid RmFileHandle::insert\_record(char\* buf, Context\* context);

**功能**：插入一条记录。参数和返回值的含义参考代码注释。

**实现**：参考代码注释。对于堆文件组织形式，只需找到一个有足够空间存放该记录的页面即可。如果所有已分配页面中都没有足够空间，则申请一个新页面来存放该记录。注意更新页中的位图（bitmap），它记录了每个槽（slot）中是否存放了记录。此外，如果当前页在插入后变满，则需要更新 file\_hdr 的第一个空闲页。

#### （4）实现 RmFileHandle::delete\_record 函数

**函数声明**：

void RmFileHandle::delete\_record(const Rid& rid, Context\* context);

**功能**：删除一条记录。参数和返回值的含义参考代码注释。

**实现**：参考代码注释。先获取page handle，然后将页中的位图（bitmap）中与删除记录槽位对应的位置0。如果删除操作导致该页面恰好从已满变为未满，则需要调用 release\_page\_handle 函数。

#### （5）实现 RmFileHandle::update\_record 函数

**函数声明**：

void RmFileHandle::update\_record(const Rid& rid, char\* buf, Context\* context);

**功能**：更新一条记录。参数和返回值的含义参考代码注释。

**实现**：参考代码注释。先获取page handle，然后直接更新页的数据即可。

#### （6）实现 RmFileHandle::fetch\_page\_handle 函数

**函数声明**：

RmPageHandle RmFileHandle::fetch\_page\_handle(int page\_no) const;

**功能**：获取指定页面对应的 RmPageHandle 对象。参数和返回值的含义参考代码注释。

**实现**：参考代码注释。需要调用 BufferPoolManager::fetch\_page() 获取指定页面。

#### （7）实现 RmFileHandle::create\_new\_page\_handle 函数

**函数声明**：

RmPageHandle RmFileHandle::create\_new\_page\_handle();

**功能**：创建一个新的 RmPageHandle 对象。返回值的含义参考代码注释。

**实现**：参考代码注释。需要调用 BufferPoolManager::new\_page()，在缓冲池中创建新页，并更新 page\_hdr 和 file\_hdr 中各项内容。

#### （8）实现 RmFileHandle::create\_page\_handle 函数

**函数声明**：

RmPageHandle RmFileHandle::create\_page\_handle();

**功能**：创建或获取一个空闲的 RmPageHandle 对象。返回值的含义参考代码注释。

**实现**：参考代码注释。基本实现逻辑是先判断第一个空闲页是否存在，如果存在，就调用 fetch\_page\_handle 函数获取它；否则，调用 create\_new\_page\_handle 函数创建一个新的 RmPageHandle 对象。

#### （9）实现 RmFileHandle::release\_page\_handle 函数

**函数声明**：

void RmFileHandle::release\_page\_handle(RmPageHandle &page\_handle);

**功能**：当page handle中的page从已满变成未满的时候调用此函数。参数的含义参考代码注释。

**实现**：参考代码注释。更新 page\_hdr 的下一个空闲页和 file\_hdr 的第一个空闲页。

#### （10）单元测试

单元测试代码在文件 src/test/storage/record\_manager\_test.cpp 中。

执行下列命令，进行单元测试：

cd build  
make record\_manager\_test  
./bin/record\_manager\_test

#### （11）注意事项

不允许修改任何公有函数的声明。

### 任务2：记录迭代器实现

补全 RmScan 类，实现对文件记录的遍历。

RmScan 类继承于 RecScan 类，它们的接口如下：

class RecScan {  
public:  
 virtual ~RecScan() = default;  
 virtual void next() = 0;  
 virtual bool is\_end() const = 0;  
 virtual Rid rid() const = 0;  
};  
  
class RmScan : public RecScan {  
public:  
 RmScan(const RmFileHandle \*file\_handle);  
 void next() override;  
 bool is\_end() const override;  
 Rid rid() const override;  
};

具体完成如下任务：

#### （1）阅读代码

阅读下列文件中的代码：

* src/record/rm\_scan.h
* src/record/rm\_scan.cpp

理解 RmScan 类的设计。

#### （2）实现 RmScan::RmScan 构造函数

**函数声明**：

RmScan::RmScan(const RmFileHandle \*file\_handle);

**功能**：构造函数。参数的含义参考代码注释。

**实现**：参考代码注释。传入 file\_handle，初始化 rid。RmScan 内部存放了 rid，用于指向一个记录。

#### （3）实现 RmScan::next 函数

**函数声明**：

void RmScan::next();

**功能**：找到文件中下一个存放记录的位置。

**实现**：参考代码注释。对于当前页面，基于位图（bitmap）找到值为1的位对应的槽号（slot\_no）。如果当前页面的所有槽中都没有存放记录，就找下一个页面。

#### （4）实现 RmScan::is\_end 函数

**函数声明**：

bool RmScan::is\_end() const;

**功能**：判断是否到达文件末尾，即最后一个页面的最后一个槽。返回值的含义参考代码注释。

**实现**：参考代码注释。可以自主定义末尾的标识符，如 RM\_NO\_PAGE。

#### （5）单元测试

单元测试代码在文件 src/test/storage/record\_manager\_test.cpp 中。

执行下列命令，进行单元测试：

cd build  
make record\_manager\_test  
./bin/record\_manager\_test

#### （6）注意事项

不允许修改任何公有函数的声明。

## 四、考核方法

### 1. 实验完成情况（80%）

根据单元测试通过情况和代码理解情况给分。

* **任务1：记录操作实现（50%）**
* **任务2：记录迭代器实现（30%）**

### 2. 实验报告（20%）

根据实验报告的完整性、科学性和规范性评分。