1. 题目一：存储管理

题目种类：内核代码填充

推荐知识点：存储管理的基本原理、文件存储组织、记录存储组织、缓冲区管理

题目描述：

在提供的代码框架的基础上，提供相关接口的实现，完成指定功能。本题目包含若干子任务，每个任务对应不同的测试点，只有通过相应测试点才可以获得相应的分数。

提示：

l 测试代码会调用指定接口来进行测试，参赛队伍不能修改已有的接口，也不能删除已有的数据结构及数据结构中的变量，但是可以增添新的接口、数据结构、变量。

l 提供了项目结构文档来帮助参赛队伍理解代码框架，同时在代码注释中，也对数据结构及接口进行了说明，参赛队伍可以通过阅读代码注释来辅助理解代码框架。

1、磁盘管理器

在本任务中，参赛队伍需要实现磁盘管理器DiskManager的相关接口，磁盘管理器负责文件操作、读写页面等。在完成本任务之前，参赛队伍需要阅读项目结构文档中磁盘管理器的相关说明，以及代码框架中src/errors.h、src/storage/disk\_manager.h、src/storage/disk\_manager.cpp、src/common/config.h文件。

参赛队伍需要实现以下接口：

（1）void DiskManager::create\_file(const std::string &path);

该接口的参数path为文件名，该接口的功能是创建文件，其文件名为path参数指定的文件名。

（2）void DiskManager::open\_file(const std::string &path);

该接口的参数path为文件名，该接口的功能是打开文件名参数path指定的文件。

（3）void DiskManager::close\_file(const std::string &path);

该接口的参数path为文件名，该接口的功能是关闭文件名参数path指定的文件。

（4）void DiskManager::destroy\_file(const std::string &path);

该接口的参数path为文件名，该接口的功能是删除文件名参数path指定的文件。

（5）void DiskManager::write\_page(int fd, page\_id\_t page\_no, const char \*offset, int num\_bytes);

该接口负责在文件的指定页面写入指定长度的数据，该接口从指定页面的起始位置开始写入数据。

（6）void DiskManager::read\_page(int fd, page\_id\_t page\_no, char \*offset, int num\_bytes);

该接口需要从文件的指定页面读取指定长度的数据，该接口从指定页面的起始位置开始读取数据。

2、缓冲池管理器

在本任务中，参赛队伍需要实现缓冲池管理器BufferPoolManager和缓冲池替换策略Replacer相关的接口，缓冲池管理器负责管理缓冲池中的页面在内外存的交换，缓冲池替换策略主要负责缓冲区页面的淘汰和查找。在完成本任务之前，参赛队伍需要首先阅读项目结构文档中缓冲池管理器的相关说明，以及代码框架中src/storage和src/replacer文件夹下的代码文件。

对于缓冲池替换策略Replacer，参赛队伍需要实现一个Replacer的子类LRUReplacer，LRUReplacer实现了缓冲池页面替换的LRU策略，需要实现的接口如下：

(1) bool LRUReplacer::victim(frame\_id\_t\* frame\_id);

该接口的功能是选择并淘汰缓冲池中一个页面。如果成功找到要淘汰的页面，则函数返回值为true；否则，返回值为false。被淘汰的页面所在的帧由参数frame\_id返回。

(2) void LRUReplacer::pin(frame\_id\_t frame\_id);

该接口的功能是固定住一个帧中的页面，代表该页面正在使用，不可被换出，参数frame\_id指定了帧的编号。

(3) void LRUReplacer::unpin(frame\_id\_t frame\_id);

该接口的功能是取消固定一个帧中的页面，当该页面使用完毕时调用unpin函数取消对该页面的固定，参数frame\_id指定了帧的编号。

对于缓冲池管理器BufferPoolManager，参赛队伍需要管理缓冲池中的页面，并对缓冲池进行并发控制，需要实现的接口如下：

(1) Page \*BufferPoolManager::new\_page(PageId \*page\_id);

该成员函数用于在缓冲池中申请创建一个新页面。如果创建新页面成功，则返回指向该页面的指针，同时通过参数page\_id返回新建页面的编号。

(2) Page \*BufferPoolManager::fetch\_page(PageId page\_id);

该成员函数用于获取缓冲池中的指定页面。待获取页面的编号由参数page\_id给出。

(3) bool BufferPoolManager::find\_victim\_page(frame\_id\_t \*frame\_id);

当缓冲池中没有可用的空闲帧时，该成员函数用于寻找需要淘汰的页面。如果成功找到要淘汰的页面，则函数返回值为true；否则，返回值为false。被淘汰的页面所在的帧由参数frame\_id返回。在实现这个成员函数时，需要调用LRUReplacer::victim函数。

(4) void BufferPoolManager::update\_page(Page \*page, PageId new\_page\_id, frame\_id\_t new\_frame\_id);

当缓冲池想要把某个帧中的页面置换为新页面或者删除该帧中的页面时，会调用update\_page函数，该函数把指定帧中的原有页面刷入磁盘中，并将新页面和该帧建立映射，即更新页表。

(5) bool BufferPoolManager::unpin\_page(PageId page\_id, bool is\_dirty);

当某个操作完成某个页面的使用之后，需要调用该函数将取消该操作对该页面的固定。当所有操作都完成该页面的使用之后，需要在Replacer中调用Unpin函数取消该页面的固定。

(6) bool BufferPoolManager::delete\_page(PageId page\_id);

用于在缓冲池中删除指定页面，同时将该页面所在的帧置为空闲帧。如果当前页面正在被某个操作使用，则该页面不能被删除。

(7) bool BufferPoolManager::flush\_page(PageId page\_id);

用于强制刷新缓冲池中的指定页面到磁盘上，无论该页是否正在被使用，或者是否为脏页，都需要把该页面的数据刷入磁盘中。

(8) void BufferPoolManager::flush\_all\_pages(int fd);

用于将指定文件中的存在于缓冲池的所有页面都刷新到磁盘。

提示：在缓冲池中，需要淘汰某个脏页时，需要将脏页写入磁盘。

3、记录管理器

在本任务中，参赛队伍需要填充记录管理器涉及的RmFileHandle类和RmScan类，RmFileHandle类负责对表的记录进行操作，RmScan类用于遍历表文件中存放的记录。

对于RmFileHandle类。在完成本文任务之前，参赛队伍需要阅读项目结构文档中记录管理器的相关说明，以及代码框架中src/record文件夹下的代码文件。

参赛队伍需要实现的接口如下：

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

该函数用于获取表中某一条指定位置的记录，每条记录由Rid来进行唯一标识。

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

该函数负责向表中插入一条新记录，在该函数中未指定记录的插入位置，参赛队伍需要选择一个空闲位置插入记录并同步更新表的元数据信息。

(3) void RmFileHandle::insert\_record(const Rid &rid, char \*buf);

该函数负责向表中的指定位置插入一条记录，该函数主要用于事务的回滚和系统故障恢复。

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

该函数负责删除表中指定位置的记录。

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

该函数负责把表中指定位置的记录更新为新的值。

对于RmScan类，参赛队伍需要实现的接口如下：

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

该函数为RmScan类的构造函数，需要初始化相关成员变量。

(2) void RmScan::next() override;

该函数负责找到表文件中下一个存放了合法记录的位置。

(3) bool RmScan::is\_end() const override;

该函数负责判断是否已经扫描到文件的末尾位置。

测试示例：

unit\_test.cpp中提供了参考测试示例，最终测试包括但不限于unit\_test.cpp中提供的测试。

题目二：（存储管理）

题目种类：基础功能

推荐知识点：逻辑查询优化、查询解析、基本算子实现、查询执行框架、元数据存储组织

题目描述：

在实现题目一功能的基础上增加查询执行模块，使得系统支持通过测试需要的元数据管理、DDL语句、DQL语句、DML语句。

提示：

l 本题目提供了空缺的样例代码，参赛队伍可以选择进行补全，也可以选择自行构建。如果选择补全空缺的样例代码，需要实现“元数据管理与DDL语句”和“DQL语句和DML语句”。

l 本题目通过sql进行测试，分为五个测试点，它们依次是“尝试建表”、“单表插入与条件查询”、“单表更新与条件查询”、“单表删除与条件查询”、“连接查询”。

l 对于所有不合法的sql语句，都需要输出failure。（包括语法错误、删除不存在的表、创建已经存在的表、where条件中出现不存在的字段等）

l 注意浮点数的精度问题，在最后一个测试点basic\_query\_test5中涉及到了浮点数精度的测试。（每个测试点的测试知识点参考测试说明文档）

1、元数据管理与DDL语句

本任务要求参赛队伍对数据库的元数据进行管理，并实现基本的DDL语句，包括create table和drop table两种语句。大赛提供的框架中，与元数据管理和DDL语句相关的代码文件位于src/system文件夹下，代码框架中提供了create table语句的实现方式，参赛队伍需要补充完善sm\_manager.cpp文件中未实现的函数（index相关的函数除外）。

2、DQL语句和DML语句

DML语句要求实现基本的增删改，即insert、delete、update语句。DQL语句要求实现select语句。在完成本任务之前，参赛队伍需要首先阅读项目结构文档中查询解析、查询优化、查询执行的相关说明，以及代码框架中src/analyze、src/optimizer、src/execution文件夹下的代码文件，参赛队伍需要实现代码框架中未提供的功能，包括语义检查、查询执行计划的生成、执行算子等。语义检查包括但不限于创建已有的表、删除不存在的表等。

在大赛提供的框架中，analyze模块提供了insert、select、delete语句的相关代码，把parser生成的抽象语法树转化成了查询计划生成所需要的Query列表，并进行了部分语义检查，参赛队伍需要补充完善update语句的相关代码。optimizer模块提供了查询计划的生成，但并未对查询计划进行逻辑优化和物理优化。execution模块中提供了insert算子的完整实现，参赛队伍需要实现扫描算子、连接算子、投影算子、update算子和delete算子。

本题目通过SQL来进行测试，因此参赛队伍可以对代码进行重构。

测试示例（见测试要求文档）

测试输出要求：

本题目的输出要求写入数据库文件夹下的output.txt文件中，例如测试数据库名称为execution\_test\_db，则在测试时使用./bin/rmdb execution\_test\_db命令来启动服务端，对应输出应写入buid/execution\_test\_db/output.txt文件中。