#MicroSQL 开发文档

## 一. 总体框架

#### 1.1 概述

MicroSQL是一个轻量级的关系型数据库, 框架的设计思路主要根据斯坦福大学的[Database System Implementation(CS346)](https://web.stanford.edu/class/cs346/2015/)课程提供的Redbase数据库框架来设计. 主要包含5个模块: - PageFile(PF) - RecordManager(RM) - IndexManager(IX) - SystemManager(SM) - QueryManager(QM) 在具体的实现中, 为了遵循模块化程序设计的原则, 将各个模块按照不同的功能分成多个类来实现, 具体实现细节将在后面介绍

#### 1.2 层次结构

#### 1.3 基本功能

* 支持的数据类型: INT, FLOAT, STRING(需要指定大小).
* 数据库的建立和删除，以及数据库的统计信息查询
* 表的建立和删除，以及表的统计信息查询
* 索引的建立和删除
* 使用与MySQL语法相似的SQL语句对表或数据库中的数据执行增、删、改、查操作

#### 1.4 设计语言和运行环境

开发环境：Visual Studio 2015 编程语言：C++（主要使用C++11推荐的语法）

## 二. 各模块的具体功能

#### 2.1 Page File Manager(PF)

PF模块是系统的最底层模块, 主要向高层次的结构(Index Manager和Record Manager)提供以页为基本单元的文件IO操作. 在该模块中, 主要实现了创建, 删除, 打开和关闭文件四个操作. 对于单个文件(PageFile类)的操作, 必须先从PF模块打开一个PageFile实例为操作对象, 以页(Page)为最小访问单元. PageFile提供了从文件中获取一个新的Page, 获取一个特定页号的Page, 强制更新(Force)文件中特定页号的Page等方法.

#### 2.2 Record Manager(RM)

RM模块是管理记录文件的模块, 主要面向Query Manager和System Manager两个模块. RM模块只提供了对记录文件的创建, 删除, 打开和关闭四个操作. 在MicroSQL的实现中, 每个表中有若干条记录, 一个表中的所有记录必须存在同一个记录文件(RecordFile)中. 对单个记录文件的操作, 必须通过RecordFile实例进行操作. RecordFile提供了对该文件中记录的增删改查四个操作.

#### 2.3 Index Manager(IX)

IX模块是管理索引的模块, 主要提供了对某个表的属性进行创建, 删除索引, 并对已创建的索引的打开, 关闭四个操作. 关于索引的具体实现, MicroSQL的索引与大多数数据库引擎类似, 使用B+树作为索引的数据结构以达到较高的索引效率. 其他模块对于单个索引的数据插入, 删除, 查询, 必须先通过Index Manager获取一个IndexHandle实例, 并通过这个IndexHandle实例进行操作.

#### 2.4 System Manager(SM)

SM模块直接面向Command Parser，提供一些可能会影响整个数据库系统的操作，包括：

* 创建、删除一个数据库/关系表/索引等（DDL）
* 维护系统中各个数据库和表的Catalog
* 从文件中导入数据
* 设置数据库或表的参数
* 输出关系表

#### 2.5 Query Manager(QM)

QM模块与SM模块类似，主要向Command Parser提供了用于执行特定的SQL操作。当前的实现支持的SQL操作有Select（包括多表联查），Insert，Delete，Update四种操作。

## Page/PageFile

PF模块中除了构造函数与析构函数之外的每个函数都返回一个整数. 返回值为0表示正常完成. 所有非0返回值均暗示了有异常情况或错误发生. 返回值为正数表示遇到异常(比如读到文件尾或关闭一个未打开的文件等对程序影响不大的错误); 返回值为负表示发生了系统无法自动处理的错误.

### 页缓冲池

获取一个文件中某页面上的数据需要先把页面存入缓冲池(维护在内存中), 然后在缓冲池中操作(读写)数据. 当一个内存中的页面和数据正在被操作时, 必须先将页面的状态设为locked (固定). 当某个进程对页面的所有操作均结束后, 必须立即把页面状态改为unlocked, 但此时并不需要把页面移出缓冲池. 只有当需要读一个新的页, 并且缓冲池中的内存不足时才会选择一个unlocked的页面移出. 所使用的选择算法为Least-Recently-Used (LRU). 当一个页面被移出缓冲池时, 只有该页面被标记为dirty才会将硬盘上页面对应的文件重写以更新页面文件, 即非dirty的页面被移出缓冲池时不需要做任何工作. 但也提供了接口使得PF模块可以强制将一个非dirty的页面数据写入硬盘, 以及在不移出页面的情况下将所有在缓冲池中的dirty页面的数据写入硬盘. PageManager必须保证每个页面只能同时被一个事务使用, 即一个locked状态下的页面不能被locked.

### 页面号

一个文件中的页面需要用页面号标识, 页面号对应其在文件中的位置. 当新建一个文件并分配页面时, 页面号是顺序增长的. 但一个页面被删除之后, 新分配的页面其页面号不一定是顺序的, 而是在之前分配过的页面上寻找一个最近被删除的页面(用栈存储删除的页面号)来存放该页面数据. 若栈为空时才在之前的页面之后再分配一个新的页面.

注: 1. 每个页面的大小(字节)用PF\_PAGE\_SIZE表示, 默认为4K (4096字节) 2. 内存池中的页面数用PF\_BUFFER\_SIZE表示, 默认为40 3. 所有作为存储函数返回结果的指针传进来时应保证其原来的数据(如果有的话)已经无需再使用了

## RecordManager

RM模块提供的类和方法用于把记录存储在文件中, 相当于PF模块的客户端. 在这个模块中需要调用PF模块中已经实现的函数.

### 文件头

为了便于管理文件内容, 可以把每个文件的第一个Page作为一个特殊的Header Page, 用于存储空闲空间的信息, 文件存储的记录数, 每个页面存储的记录数, 文件的当前页面数以及其他与整个文件有关的信息. 每个页面也包含一个页面头.

### 记录标识符

RecordIdentifier类对一个指定文件中的所有记录提供唯一标识, 因此一条指定记录的标识符应该是保持不变的. 即一个标识符的属性在记录更新或其他记录插入/删除的条件下都是恒定的. 每个文件的页面有数量不等的Slot(根据每个文件的Record的大小确定), 但一个文件内的所有页面Slot数相同. 在这个系统中只用页面号(PageNum)以及槽号(SlotNum)来构成为RecordIdentifier.

### 记录空闲空间

当需要插入记录时, 不应该线性搜索有空余的页面. 为了提高搜索效率, 可以使用一个页的链表来存储有空余Slot的页面.

每个文件中存储的记录必须是等长的, 这样能更方便地管理每个页面上的记录和空闲空间, 而且保证了每个记录的位置都能够方便地访问到. 最好每个表单独存储在一个文件中.

### 访问记录

当主程序需要访问表Table中一条标识符为Id的记录时, 分两个步骤: 首先通过RecordManager->OpenFile(Table)申请一个RecordFile(实际上, 只能通过这个类来访问数据库的记录), 通过RecordFile->GetRec(Id)来获取一个Record对象. 但是由于这样效率比较低下, 因此主要还是用FileScan类来让客户端与数据库进行交互

## Index Manager

## System Manager

## Query Manager

## 三. 重要接口

## 四. 使用语法

## 五. 实际测试

## 六. 组内分工