YourSQL 项目文档

杨乐、郑立言

一、概述

YourSQL 项目为本学期数据库的课程大作业。据课程要求，YourSQL 是一个数据库管理系统，支持基本的增删改查等基本操作，并在其上拓展了若干附加功能。小组成员有杨乐和郑立言两名同学，项目地址为 https://github.com/zhengly123/YourSQL。

二、功能分析

YourSQL 项目实现了如下功能：

1. 系统管理
    a) show databases 展示所有数据库
    b) create database 创建数据库
    c) drop database 删除数据库
    d) use database 使用数据库
    e) close 关闭数据库
    f) create table 创建关系
    g) drop table 删除关系
2. 基本查询
    a) insert 插入子句
    b) delete 删除子句
    c) update 更新子句
    d) select 查询子句
3. 查询优化
    a) create index 创建索引
    b) drop index 删除索引
4. 简单的外键约束（创建关系时可指定 primary key 主键与 foreign key 外键）
5. 更多的数据类型支持（支持 date 日期类型）
6. 三个表以上的数据连接（select 查询子句拓展）
7. 聚集查询
    a) avg、sum、min、max 聚集查询
    b) group by 分组
    c) order by 排序
8. 模糊查询（在 where 子句中使用 like 关键字，使用 “%,_” 进行匹配）
9. 网络多用户（通过 c-s 模式使得多用户可同时使用同一数据库）

三、模块分析

针对上述的功能分析、助教提供的课程资料以及 Standford Redbase 课程资料，本项目分为以下模块进行开发：

1. **文件系统模块(PF)**:主要负责底层对文件进行直接读写，以及增加缓存来加速读写毒素。
2. **记录管理模块（RM）**：在文件系统之上，管理与存储数据库记录以及元数据的文件。
3. **索引模块（IX）**：为存储在文件中的记录建立 B+树索引，加快查找速度。
4. **系统管理模块（SM）**：支持系统管理功能，对数据库和数据表进行管理。
5. **查询解析模块（QL）**：支持基本查询功能，对数据库的数据进行增产改查的基本操作。
6. **命令解析模块(Parser)**:根据用户输入的 SQL 命令，解析成参数并调用对应的系统函数。
7. **输出模块（Printer）**：对于 select 查询子句，实现对信息的缓存以及再输出。
8. **错误处理模块（Errorhandle）**：对于系统可能出现的错误进行处理，并输出对应的信息。
9. **正则匹配模块（Regex）**：用于模糊查询中的匹配正则表达式。
10. **网络模块（Client/Server）**：系统的网络交互模块，支持用户可通过网络来对数据库进行操作。
11. **测试模块（GTest）**：采用 Google Test，对数据库进行功能测试，确保正确性。

四、项目周期与分工情况

项目开发的时间为 2018-2019 秋季学期上半学期。组内分工情况如下：

杨乐：记录管理模块（RM）、查询解析模块（QL）、命令解析模块（Parser）、错误处理模块（Errorhandle）以及测试模块（GTest）。

郑立言：索引模块（IX）、系统管理模块（SM）、查询解析模块（QL）、输出模块（Printer）、正则匹配模块（Regex）、网络模块（Client/Server）以及测试模块（GTest）。

五、各模块分析

1. 记录管理模块（RM）

记录管理模块仿照 StanfordCS346 Redbase 来设计端口，并加以实现；以下介绍每个部分的端口与功能。

（1） RID， 记录 id

我们为每个记录单独设置一个 id 来标志它。该 id 有两个域，pageNum 和 slotNum，分别记录该 id 在文件中的页数和槽数。

RID 除了初始化，是不能修改的；外部只能根据该 ID 作为唯一的凭据来访问某一个与之相关的记录。

（2） RM_Record，记录

在用户使用 RID 向系统询问时，我们返回一个 RM_Record 类代表查询的结果。RM_Record 有两个域，一个为所查询记录数据的*拷贝*；二则为该记录的 RID。

用户在修改记录时，需要对拷贝进行修改后，再向系统通过 RID 和修改的数据提出更新请求。

（3） RM_FileHandle，文件管理器

文件管理器是与用户交互的核心。用户可以提供一个RID，系统会返回一个记录（RM_Record）；用户可以插入一个记录，然后得到一个可以访问该记录的 RID；用户可以提供一个 RID，来删除该记录；用户可以提供一个更新后的记录，来更新文件中的对应记录。

```
class RM_FileHandle
{
   RC GetRec       (const RID &rid, RM_Record &rec) const; // Get a record
   RC InsertRec     (const char *pData, RID &rid);      // Insert a new record, return
record id
   RC DeleteRec     (const RID &rid);                 // Delete a record
   RC UpdateRec     (const RM_Record &rec);            // Update a record
};
```

（4） RM_FileScan，文件扫描器

文件扫描器用于扫描记录中满足一定条件的所有记录。系统支持查找记录中某一个属性等于、大于、不大于、小于、不小于、不等于某个值的记录。这里并没有采用索引进行优化，故是线性的查询机制。

```
class RM_FileScan {
   RC OpenScan    (const RM_FileHandle &fileHandle,  // Initialize file scan
                  AttrType     attrType,
                  int attrLength,
                  int attrOffset,
                  CompOp       compOp,
                  void *value);
   RC GetNextRec  (RM_Record &rec);              // Get next matching record
   RC CloseScan   ();                           // Terminate file scan
};
```

（5） RM_Manager，记录管理器

记录管理器是记录管理模块的顶端结构。用户可以向它申请创造文件，删除文件，打开与关闭文件。在打开文件后，用户可以和它提供的文件管理器作进一步交互。

```
class RM_Manager
{
   RC CreateFile  (const char *fileName, int recordSize); // Create a new file
   RC DestroyFile (const char *fileName);      // Destroy a file
   RC OpenFile    (const char *fileName, RM_FileHandle &fileHandle); // Open a file
   RC CloseFile   (RM_FileHandle &fileHandle);  // Close a file
};
```

2. 索引模块（IX）

总体上，API 采用 Stanford CS 346 Redbase 提供的接口，但由于 PF 不同，也少许修改。

（1） IX_Manager（索引管理）

Manager 模块主要进行与文件相关的管理。我们采用了 map<pair<string, int>, int> openedFile 管理已经打开的文件，防止文件的非法关闭。

```
class IX_Manager
{
   RC CreateIndex  (const char *fileName,        // Create new index
                    int       indexNo,
                    AttrType   attrType,
                    int       attrLength);
   RC DestroyIndex (const char *fileName,        // Destroy index
                    int       indexNo);
   RC OpenIndex    (const char *fileName,        // Open index
                    int       indexNo,
                    IX_IndexHandle &indexHandle);
   RC CloseIndex   (IX_IndexHandle &indexHandle);  // Close index
}
```

（2） IX_IndexHandle（索引遍历）

我们采用了 B+树作为索引的数据结构，参考了邓俊辉老师的数据结构课程中关于 B 树原理讲解的幻灯片。

```
class IX_IndexHandle
{
   IX_IndexHandle(FileManager &fm,
          BufPageManager &bpm,
          IX_Manager &ixManager,
          IX_Header &ixHeader);
   RC open (FileManager &fm,
           BufPageManager &bpm,
           IX_Manager &ixManager,
           int fileID,
           const char *fileName, int indexNo);
   RC close();
}
```

以下为 B+树的节点声明。

```
class BPlusTreeNode
{
   RC insert(void* key,const RID &value);
   RC remove(void* key,const RID &value);
   bool less(void * a, void * b);
   bool equal(void *a, void *b);
   bool full();
   bool overfull();
   void* getKey(int k);
   int firstGreaterIndex(void *key);
};
```

（3） IX_IndexScan（索引查询）

于在 B+树中的所有叶子节点形成了链，所以只用先用树上查找目标 Key，然后遍历该链即可得到所有结果。

```
class IX_IndexScan {
   RC OpenScan     (const IX_IndexHandle &indexHandle, // Initialize index scan
                    CompOp     compOp,
                    void       *value,
                    ClientHint  pinHint = NO_HINT);
   RC GetNextEntry  (RID &rid);                      // Get next matching entry
   RC CloseScan     ();                             // Terminate index scan
};
```

（4） 优化

在 Select 或其他语句中，可能会出现有多个索引的情况。为了作出最优选择，我们统计了 B+树中不同 Key 的节点数量，以便挑选选择性最强的索引。

3. 系统管理模块（SM）

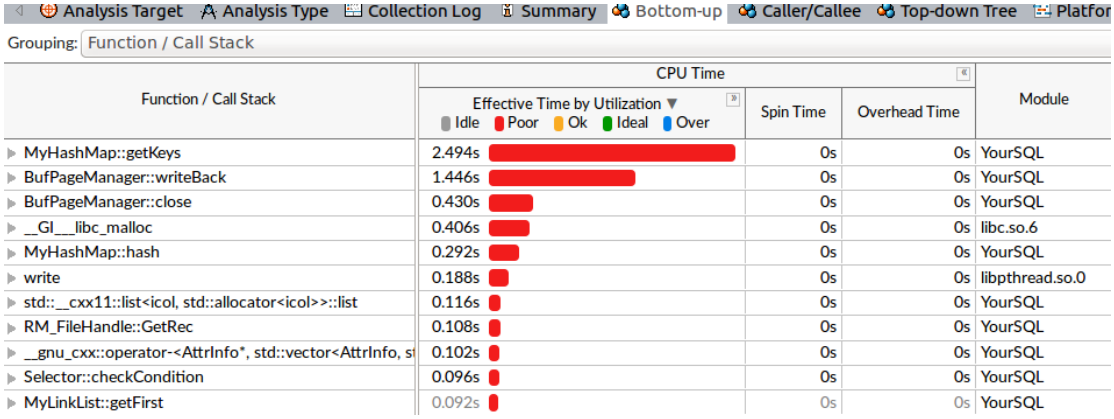
（1）接口定义

```
class SM_Manager {
   RC CreateDb    (const char *dbName);
   RC OpenDb      (const char *dbName);             // Open database
   RC CloseDb     ();                              // Close database
   RC CreateTable (const char *relName,             // Create relation
                   int       attrCount,
                   AttrInfo   *attributes);
   RC DropTable   (const char *relName);            // Destroy relation
   RC CreateIndex (const char *relName,             // Create index
                   const char *attrName);
   RC DropIndex   (const char *relName,             // Destroy index
                   const char *attrName);
}
```

（2） Database 操作

在代码中，我们将创建、删除以及切换数据库分别对应 mkdir，rmdir 与 chdir 这三个 Linux 函数，并做了异常判断。值得一提的是，在调用这三个函数时，需要严格检查参数是否合法，考虑各种可能情况，避免发生危险操作。

（3）管理 Handle



Function / Call Stack	CPU Time			Module
	Effective Time by Utilization ▼ (Idle Poor Ok Ideal Over)	Spin Time	Overhead Time	
MyHashMap::getKeys	2.494s	0s	0s	YourSQL
BufPageManager::writeBack	1.446s	0s	0s	YourSQL
BufPageManager::close	0.430s	0s	0s	YourSQL
__GI___libc_malloc	0.406s	0s	0s	libc.so.6
MyHashMap::hash	0.292s	0s	0s	YourSQL
write	0.188s	0s	0s	libpthread.so.0
std::__cxx11::list<icol, std::allocator<icol>>::list	0.116s	0s	0s	YourSQL
RM_FileHandle::GetRec	0.108s	0s	0s	YourSQL
__gnu_cxx::operator-<AttrInfo*, std::vector<AttrInfo, st	0.102s	0s	0s	YourSQL
Selector::checkCondition	0.096s	0s	0s	YourSQL
MyLinkList::getFirst	0.092s	0s	0s	YourSQL

通过性能测试，可以发现由于 BufPageManager 的特性，关闭文件是非常昂贵的操作，主要耗时都集中在 BufPageManager 的 close 操作。为了避免文件开关的 overhead，我们采用在 SM_Manager 中集中管理 RM_FileHandle 和 IX_IndexHandle，运行中不进行关闭文件的操作，而是将 Handle 放在 SM_Manager 中保存，等待后续使用。

4. 查询解析模块（QL）

本模块支持系统调用的 Insert、Delete、Update 与 Select 语句，并与之前的系统管理模块结合，一同完成对数据库 SQL 语句的支持。

（1）模块接口

分别对四条语句实现了各自的接口，如下所示，其中参数已在注释中标明：

```
RC Select (int         nSelAttrs,       // # attrs in Select clause
        const RelAttr selAttrs[],      // attrs in Select clause
        std::list<std::string> rellist,
        int         nConditions,     // # conditions in Where clause
        const Condition conditions[]);  // conditions in Where clause
RC Insert (const char  *relName,          // relation to insert into
        int        nValues,          // # values to insert
        const Value values[]);         // values to insert
RC Delete (const char *relName,           // relation to delete from
        int       nConditions,       // # conditions in Where clause
        const Condition conditions[]);  // conditions in Where clause
RC Update (const char *relName,           // relation to update
        int   nSet,
        const Condition sets[],
        int   nConditions,            // # conditions in Where clause
        const Condition conditions[]);  // conditions in Where clause
```

（2）where 子句支持与索引优化

由于 delete、update、select 需要用到 where 子句，我们定义了一个单独的 Selector 类，进行筛选；而涉及到的索引优化查询，可以统一在这个类中进行实现。

（3）多表连接

为了支持多表连接，我们建立了一个 selectResult 类，用于记录当前的 select 结果。对于多个表的情况，我们首先会对其中约束较为严格的表进行筛选。然后再对后续的表进行筛选，通过笛卡尔积的方式将后表与前表的结果相乘，并保存在新的 selectResult 中。

（4）聚集查询

对于聚集查询，我们只需要对最后的 selectResult 中的结果进行进一步数据处理即可 ;sum、avg 只需要对应列进行求和，而 min、max 可以调用对应的比较器进行比较。group by 子句可以忽略掉其余的列进行查询和比对；order by 子句直接调用 sort 函数进行排序即可。

5. 命令解析模块（Parser）

YourSQL 采用 flex 和 bison 进行输入命令的解析。命令首先需要经过 flex 进行词法分析，然后经过 bison 进行语法分析，最后分析得到结果。
这里需要注意的是，在助教给出的语法分析表中，不支持日期、浮点数以及更复杂带转义符号的字符串。这些都需要进行补充：前两者实现较为简单，而后者需要用到 flex 中的状态机来额外对字符串内部进行识别，而不能使用统一的正则表达式进行匹配。

6. 输出模块（Printer）

YourSQL 支持屏幕直接输出、CSV 文件输出和基于 Socket 的输出，且三种方式可以切换。实现时通过一个抽象类 Printer，派生出不同的子类，如屏幕输出的 StdoutPrinter 与 CSV 输出的 CSVPrinter；并将 Printer 与 SM 模块和 QL 模块相结合，在切换模式时，只需要将其中的 Printer 进行切换，即可采用不同的输出模式。

7. 网络模块（Server-Client 模式）

YourSQL 支持 Server-Client 模式。

通过学习 Redis 的服务器设计架构，YourSQL 采用了 IO 多路复用技术，通过事件队列实现单线程多连接管理，实现操作的序列化。

与 Redis 的双层事件队列不同，YourSQL 采用了 Linux 的 epoll 机制，能够高效管理网络通信。

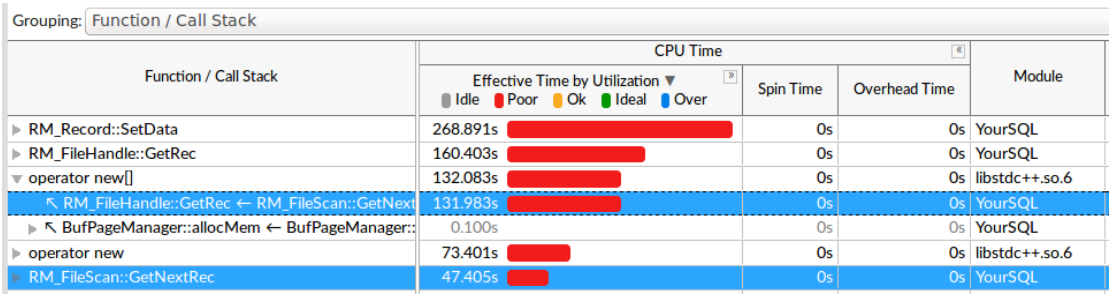
由于单个 Server 进程只能管理一个数据库，需要 Client 进行配置。Client 分为 Client_administrator 和 Client_user， Client_administrator 拥有完整的数据库操纵权限，但 Client_user 不能进行数据库的创建、删除操作，防止数据库结构被破坏。

8. 测试模块（GTest）

YourSQL 采用了 Google Test 测试框架，总共设计了 60 个自动化测例和另外的手动测例。这些测例覆盖了各个模块的多数函数，保证了系统具有一定的鲁棒性。

六、性能优化

我们对助教提供的 10W 条插入数据、以及部分的 select 查询进行性能测试，结果如下：



Grouping: Function / Call Stack

Function / Call Stack	CPU Time			Module
	Effective Time by Utilization ▼ (Idle Poor Ok Ideal Over)	Spin Time	Overhead Time	
RM_Record::SetData	268.891s	0s	0s	YourSQL
RM_FileHandle::GetRec	160.403s	0s	0s	YourSQL
operator new[]	132.083s	0s	0s	libstdc++.so.6
↖ RM_FileHandle::GetRec ← RM_FileScan::GetNext	131.983s	0s	0s	YourSQL
↖ BufPageManager::allocMem ← BufPageManager::	0.100s	0s	0s	YourSQL
operator new	73.401s	0s	0s	libstdc++.so.6
RM_FileScan::GetNextRec	47.405s	0s	0s	YourSQL

在插入为主的操作中，YourSQL 耗费了较多时间在 RM_Record::SetData 分配内存上，可以考虑避免插入时的内存申请和复制，以进行优化。同时，现在的 RM_FileHandle::GetRec 需要将数据复制一次，产生了较大的 Overhand。这是因为目前 BufPageManager 不支持固定 buffer 等操作，可以考虑修改 BufPageManager 以实现进一步的优化。

在 Select 为主的操作中，主要热点同样为 RM_FileHandle::GetRec，可以采用上述的优化方法。