

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | 数据库原理 |
| 姓 名： | 闫徐天任 |
| 学 院： | 计算机学院 |
| 系： | 计算机 |
| 专 业： | 软件工程 |
| 学 号： | 3160103827 |
| 指导教师： | 陈岭 |

2018年 6 月 28 日

# Record Manager 设计报告

以下是Record Manager的详细设计报告。

## Record Manager 主要功能

首先，Record Manager 实现了从内存块（Buffer提供）到记录条目（tuple）的抽象。主要功能有：

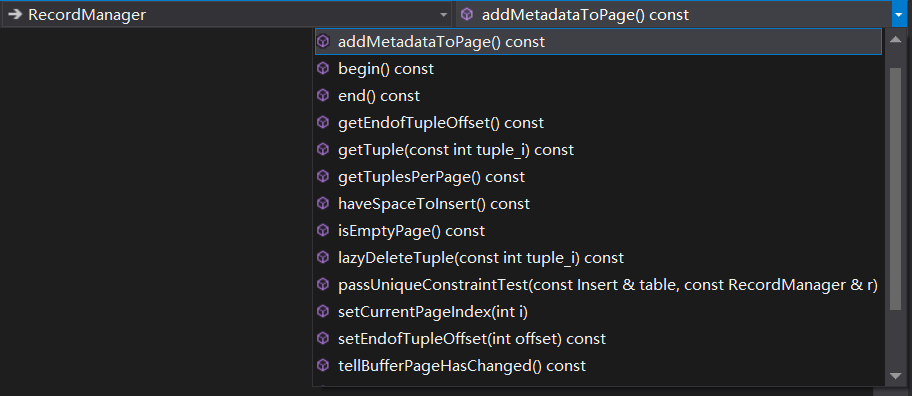
* 将记录序列化到内存块；
* 将内存块中储存的信息还原为记录；
* 修改内存块中的特定信息；

其次，record manager负责实现针对表的数据增删查的API：

* 插入记录；
* 检查要插入的记录是否符合唯一性约束；
* 根据查询条件筛选记录；
* 打印表格；
* 删除记录；
* 新建索引时，为表格中所有已有数据调用index manager的接口建立索引；
* 删除表格时，删除该表格的所有记录。

## 模块总体设计思路

虽然本模块不直接读写文件，但是为了实现数据的持久性，表的记录最终需要写入磁盘。因此需要将记录条目对象序列化，写入由buffer manager提供的内存块（每一个内存块是长度为4096的字符数组）。为避免复杂性的增加，我专门实现了几个序列化/反序列化函数、并且加入一个类 class RecordManager，将内存块封装，抽象出更高层次的接口：



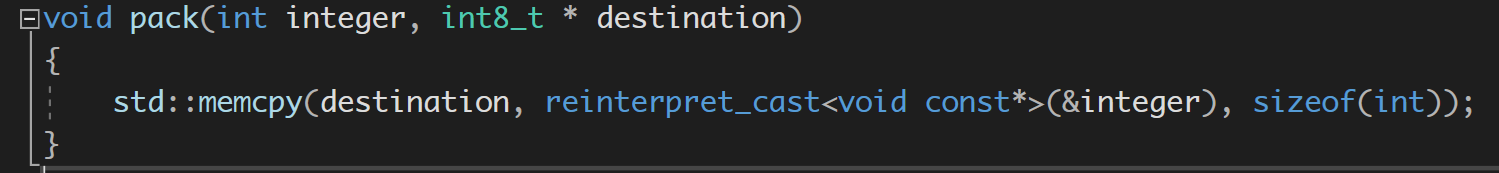
与其他模块进行交互的函数使用上图中的接口操作，不直接接触内存块。

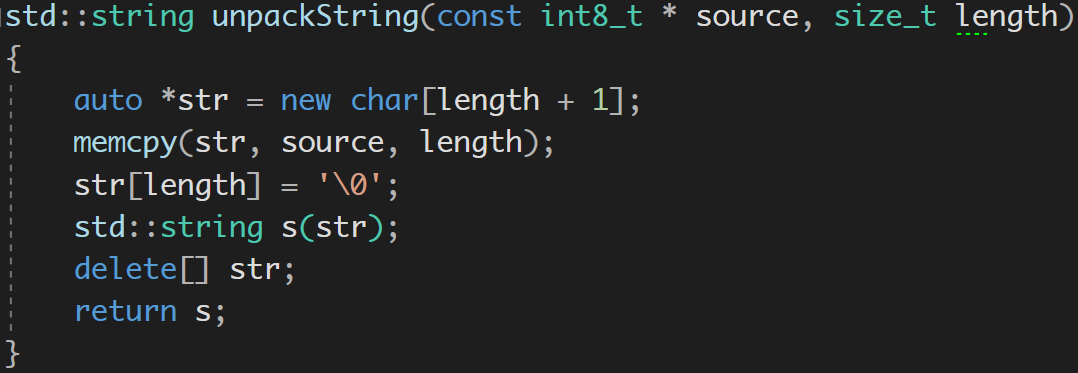
## 详细设计

以下是序列化函数、RecordManager类和记录管理模块实现的API函数的具体设计

### 序列化函数

我分别写了序列化int、float和std::string的函数，以及它们对应的逆序列化的函数。这些函数均是memcpy函数的封装，在这里仅分别举一例（int的序列化与std::string的逆序列化）





### 数据序列化方法

一个表可能有很多块，每一块的排布如下图所示：（0-4095字节）

其中地址164标记该内存块是否为空；地址165储存的是表的列数。

0-159 表名

…

160-163 条目长度

164

165

166-169 尾后偏移量

176-187 各列的长度

192-4095 按添加顺序储存记录

…

…

188-191删除首条目偏移量

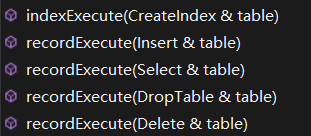
### RecordManager类

这个类主要是使用序列化与逆序列化函数对内存块数组的读取与修改。将一组操作封装为一个接口。该类的声明如下：

|  |
| --- |
| class RecordManager {     int current\_page\_index\_ = -1; // 当前内存块 id，从 0 开始     int8\_t \*memory\_page\_ = nullptr;       const Table& table\_;  ​  public:  int tuple\_size = 0; // 每个条目的长度，  // 列名到列索引的映射     std::map<string, int> column\_name\_to\_index;     RecordManager(const Table &table);  ​     bool isEmptyPage() const;     void addMetadataToPage() const;     int8\_t \* end() const;     int8\_t \* begin() const;     int getEndofTupleOffset() const;     void setEndofTupleOffset(int offset) const;     int getTuplesPerPage() const;     bool haveSpaceToInsert() const;  ​     void toNextPage();     void toPreviousPage();     int getCurrentPageIndex() const    {         return current\_page\_index\_;    }     void gotoPage(int i);     int getLastPageIndex();     void setLastPageIndex(int i);  ​     std::vector<std::string> getTuple(const int tuple\_i) const;     bool tupleHasBeenDeleted(int tuple\_i) const;     void lazyDeleteTuple(int tuple\_i) const;  ​     void tellBufferPageHasChanged() const;  ​     static bool passUniqueConstraintTest(const Insert& table, const RecordManager& r);  ​  }; |

### API的实现方法

Record Manager负责实现以下由API规定的函数：



#### 新建索引

|  |
| --- |
| for 每一条记录  for 每一列  该列是否有索引？添加索引 : continue; |

#### 插入条目

|  |
| --- |
| for 每一个有索引的列  使用索引检查新插入项的该列内容时候满足唯一性（不满足 return 错误信息）  u = 不需要再检查唯一性约束  for 每一列  该列是否有唯一性约束但是没有添加索引？u = 需要检查唯一性约束 : continue  if 不需要检查唯一性约束  读取该表的倒数第二个内存块  while(true)  读取下一个内存块  if 不需要检查唯一性约束 or 在该内存块满足唯一性约束  if 这个内存块还有空间插入  for 插入项的每一列  插入这一列  这一列有索引 ? 插入索引 : continue  else 读取下一个内存块  else  return 错误信息 |

#### 查询条目

|  |
| --- |
| for 每一列  建立表头  if对单个有索引列的等值查询  使用索引查询，调整表的间距，返回格式化后的字符串  for 每一个内存块  for 每一条有效记录  条件满足  for 每一个条件  条件不满足 ? 条件不满足, break : continue  if 条件满足  添加到待输出项，调整表的间距  输出 |

#### 删除条目

删除与查询相似，只是将符合条件的条目由输出换成标记条目为删除。

#### 删表

直接通知Buffer将要删除的表对应的文件删除。