# #0 PROJECT MINISQL OVERVIEW

**2024年5月31日：修复了框架的一些问题，请大家重新拉取代码**

## 框架链接：ZJU GitLab链接，请使用内网访问

## 一、实验目的

1. 设计并实现一个精简型单用户SQL引擎MiniSQL，允许用户通过字符界面输入SQL语句实现基本的增删改查操作，并能够通过索引来优化性能。
2. 通过对MiniSQL的设计与实现，提高学生的系统编程能力，加深对数据库管理系统底层设计的理解。

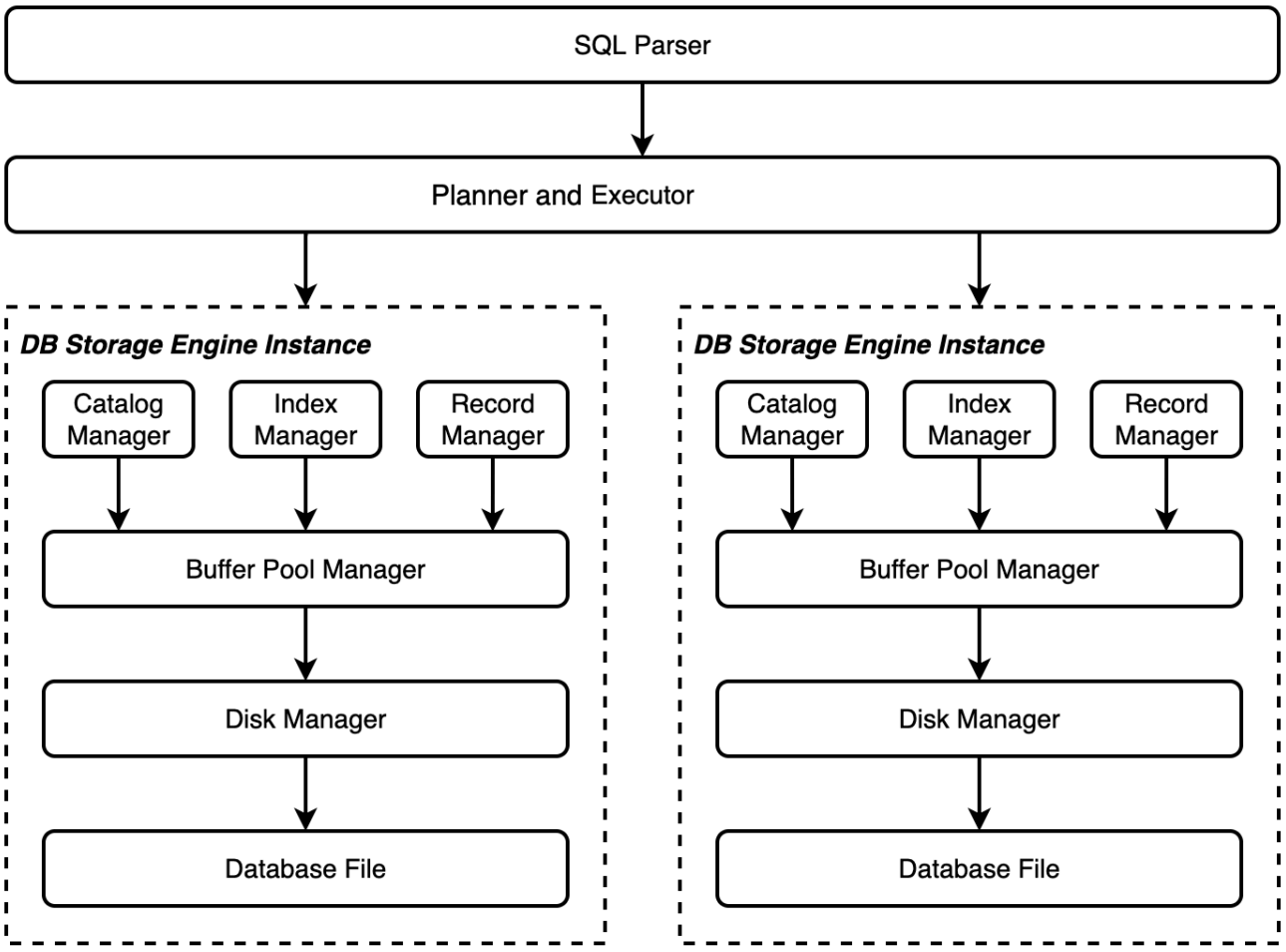
## 二、实验需求

1. 数据类型：要求支持三种基本数据类型：`integer`，`char(n)`，`float`。
2. 表定义：一个表可以定义多达32个属性，各属性可以指定是否为`unique`，支持单属性的主键定义。
3. 索引定义：对于表的主属性自动建立B+树索引，对于声明为`unique`的属性也需要建立B+树索引。
4. 数据操作: 可以通过`and`或`or`连接的多个条件进行查询，支持等值查询和区间查询。支持每次一条记录的插入操作；支持每次一条或多条记录的删除操作。
5. 在工程实现上，使用源代码管理工具（如Git）进行代码管理，代码提交历史和每次提交的信息清晰明确；同时编写的代码应符合代码规范，具有良好的代码风格。

## 三、系统架构与模块概述

### 3.1 系统架构示意图

- 在系统架构中，解释器`SQL Parser`在解析SQL语句后将生成的语法树交由执行器`Executor`处理。执行器则根据语法树的内容对相应的数据库实例（`DB Storage Engine Instance`）进行操作。
- 每个`DB Storage Engine Instance`对应了一个数据库实例（即通过`CREATE DATABSAE`创建的数据库）。在每个数据库实例中，用户可以定义若干表和索引，表和索引的信息通过`Catalog Manager`、`Index Manager`和`Record Manager`进行维护。目前系统架构中已经支持使用多个数据库实例，不同的数据库实例可以通过`USE`语句切换（即类似于MySQL的切换数据库），在初步实现时，可以先考虑单个数据库实例的场景，在单个实例跑通后再支持多个实例。




## 3.2 系统模块概述

### 3.2.1 Disk Manager

- Database File（DB File）是存储数据库中所有数据的文件，其主要由记录（Record）数据、索引（Index）数据和目录（Catalog）数据组成（即共享表空间的设计方式）。与书上提供的设计（每张表通过一个文件维护，每个索引也通过一个文件维护，即独占表空间的设计方式）有所不同。共享表空间的优势在于所有的数据在同一个文件中，方便管理，但其同样存在着缺点，所有的数据和索引存放到一个文件中将会导致产生一个非常大的文件，同时多个表及索引在表空间中混合存储会导致做了大量删除操作后可能会留有有大量的空隙。在本实验中，为了方便同学们实现，我们采取共享表空间的设计方式，即将所有的数据和索引放在同一个文件中。学有余力的同学可以额外尝试使用独占表空间的设计方式进行设计。
- Disk Manager负责DB File中数据页的分配和回收，以及数据页中数据的读取和写入。
- 对应实验：#1 DISK AND BUFFER POOL MANAGER。

### 3.2.2 Buffer Pool Manager

- Buffer Manager 负责缓冲区的管理，主要功能包括：

1. 根据需要，从磁盘中读取指定的数据页到缓冲区中或将缓冲区中的数据页转储（Flush）到磁盘；
2. 实现缓冲区的替换算法，当缓冲区满时选择合适的数据页进行替换；
3. 记录缓冲区中各页的状态，如是否是脏页（Dirty Page）、是否被锁定（Pin）等；
4. 提供缓冲区页的锁定功能，被锁定的页将不允许替换。

- 为提高磁盘 I/O 操作的效率，缓冲区与文件系统交互的单位是数据页（Page），数据页的大小应为文件系统与磁盘交互单位的整数倍。在本实验中，数据页的大小默认为 4KB。
- 对应实验：#1 DISK AND BUFFER POOL MANAGER。

#### 3.2.3 Record Manager

- Record Manager 负责管理数据表中记录。所有的记录以堆表（Table Heap）的形式进行组织。Record Manager 的主要功能包括：记录的插入、删除与查找操作，并对外提供相应的接口。其中查找操作返回的是符合条件记录的起始迭代器，对迭代器的迭代访问操作由执行器（Executor）进行。
- 堆表是由多个数据页构成的链表，每个数据页中包含一条或多条记录，支持非定长记录的存储。不要求支持单条记录的跨页存储（即保证所有插入的记录都小于数据页的大小）。堆表中所有的记录都是无序存储的。
- 需要额外说明的是，堆表只是记录组织的其中一种方式，除此之外，记录还可以通过顺序文件（按照主键大小顺序存储所有的记录）、B+树文件（所有的记录都存储在B+树的叶结点中，MySQL中InnoDB存储引擎存储记录的方式）等形式进行组织。学有余力的同学可以尝试着使用除堆表以外的形式来组织数据。
- 对应实验：#2 RECORD MANAGER。

#### 3.2.4 Index Manager

- Index Manager 负责数据表索引的实现和管理，包括：索引（B+树等形式）的创建和删除，索引键的等值查找，索引键的范围查找（返回对应的迭代器），以及插入和删除键值等操作，并对外提供相应的接口。
- B+树索引中的节点大小应与缓冲区的数据页大小相同，B+树的叉数由节点大小与索引键大小计算得到。
- 对应实验：#3 INDEX MANAGER。

#### 3.2.5 Catalog Manager

- Catalog Manager 负责管理数据库的所有模式信息，包括：

1. 数据库中所有表的定义信息，包括表的名称、表中字段（列）数、主键、定义在该表上的索引。
2. 表中每个字段的定义信息，包括字段类型、是否唯一等。
3. 数据库中所有索引的定义，包括所属表、索引建立在那个字段上等。

- Catalog Manager 还必需提供访问及操作上述信息的接口，供执行器使用。
- 对应实验：#4 CATALOG MANAGER。

#### 3.2.6 Planner and Executor

- Planner（执行计划生成器）的主要功能是根据解释器（Parser）生成的语法树，通过Catalog Manager 提供的信息检查语法树中的信息是否正确，如表、列是否存在，谓词的值类型是否与column类型对应等等，随后将这些词语转换成可以理解的各种 c++ 类。解析完成后，Planner根据改写语法树后生成的Statement结构，生成对应的Plannode，并将Plannode交由Executor进行执行。
- Executor（执行器）的主要功能是遍历Planner生成的计划树，将树上的 PlanNode 替换成对应的 Executor，并调用 Record Manager、Index Manager 和 Catalog Manager 提供的相应接口进行执行。Executor采用的是火山模型，提供迭代器接口，每次调用时会返回一个元组和相应的 RID，直到执行完成。
- 语法树的相关结构请参考#5 PLANNER AND EXECUTOR。

#### 3.2.7 SQL Parser

- 程序流程控制，即“启动并初始化 → ‘接收命令、处理命令、显示命令结果’循环 → 退出”流程。
- 接收并解释用户输入的命令，生成命令的内部数据结构表示，同时检查命令的语法正确性和部分语义正确性，对正确的命令生成语法树，然后调用执行器层提供的函数执行并显示执行结果，对不正确的命令显示错误信息。

## 四、评分标准

本实验有一定的规模，实现上有一定的复杂度，推荐1~3人一组**（请各小组于夏学期第1周结束前完成组队信息填写）**，完成一个完整的系统，推荐每个同学都参与到每个模块的设计中（但也可以每个同学完成单独的模块），分工合作完成模块功能实现、测试用例编写、模块功能测试、性能调优以及文档撰写工作。具体的评分标准如下：

| 评分项 | 说明 | |
|---|---|---|
| 小组总体设计报告（20%） | 总体报告得分各小组成员相同，在总体报告中应给出各成员负责的模块。详细报告根据各人的任务单独给分；报告及时提交则根据报告的质量进行给分；未及时提交则在报告质量的基础上降一级；未提交报告或抄袭，相应报告得分为`0`。 | |
| 个人详细设计报告（20%） | | |
| 系统测试与验收（60%） | 功能性测试（50%） | 多人协作完成一个完整的系统，经验收功能完善且几乎没有错误，则组内各成员验收等级为`A`；如某模块功能不完善或有较多错误，则对负责该模块的成员进行扣分；若只有单人参与实现，但无法进行系统演示，根据完成度评定等级为`B`、`C`、`D`或`0`，原则上不评定为`A`；若单人完成，且能够进行系统演示，完成度较高，可获得加分，但平时分不会溢出50分多人协作完成一个系统，但最后系统无法联合运行，则按各人完成他负责的模块进行处理；若程序编写工作基本完成，但无法运行或无法进行测试，则根据程序质量给评定验收等级为`C`、`D`；若基本上未编写程序或程序纯属抄袭，验收得分为`0`。 |
| 正确性测试（10%） | 对于系统中每个模块，除了提供的参考测试代码外，每个小组需要自行设计测试代码并运行通过，测试代码中的测试用例应尽可能涵盖所有可能出现的情况。测试代码的设计说明也应体现在最终的设计文档中； | |

| 评分项 | 说明 | |
| --- | --- | --- |
| 附加项（Bonus） | 完成实验中要求的附加项并通过验收可获得一定的加分；对MiniSQL代码框架提出建设性意见的，经采纳可获得一定的加分；Bonus直接加在平时分上，但平时分总分不超过50分。 | |

附：验收等级与得分对照

| | |
| --- | --- |
| A | **90~100** |
| B | 80~89 |
| C | 70~79 |
| D | 60~69 |
| E | <60 |