浙江大学

大规模信息系统构建技术导论

分布式 MiniSQL 系统设计报告

2021 学年　第一学期（上）

组员信息（第一行请写组长信息）

学号	姓名
3180103772	张溢弛
3180103162	张琦
3180103501	聂俊哲

2021 年　 5 月　　4 日

目 录

一、引言

1.1 系统目标

本项目是《大规模信息系统构建技术导论》的课程项目。在大二春夏学期的《数据库系统》课程上学习的数据库基本知识与大三春学期的《大规模信息系统构建技术导论》课程上学习的分布式系统与大规模软件系统的构建的知识，完成一个分布式的 miniSQL 系统，包含 Zookeeper 集群，Master，Region Server，Client 等模块，可以完成一个分布式的关系型数据库的各类基本操作，SQL 语句的执行，并且具有**副本维护、负载均衡、容错容灾**等功能。

本系统将使用 Java 作为编程语言进行构建,并使用 Maven 作为包管理工具，Zookeeper(一种分布式协调服务，用于管理大型主机。在分布式环境中协调和管理服务是一个复杂的过程。ZooKeeper 通过其简单的架构和 API 解决了这个问题。ZooKeeper 允许开发人员专注于核心应用程序逻辑，而不必担心应用程序的分布式特性。)作为集群管理工具。可以在各个操作系统平台中跨平台使用。

1.2 设计说明

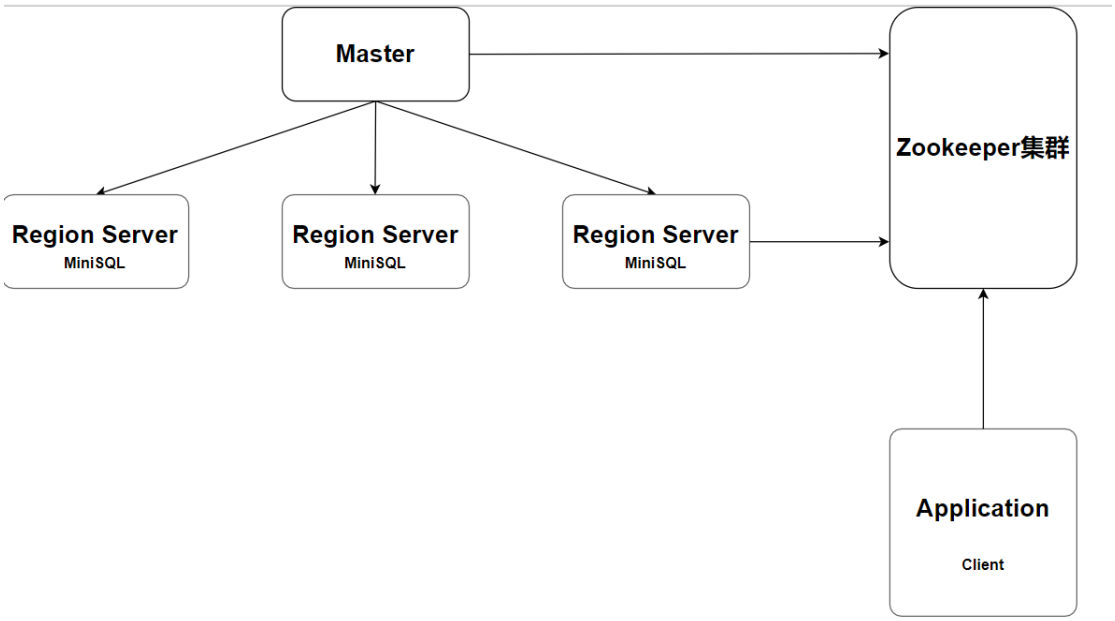
本程序采用 Java 程序设计语言，在 IDEA 集成开发环境中下编辑、编译与调试。具体程序由 3 人组成的小组开发而成。小组成员的具体分工如表 1 所示：

成员姓名	学号	分工
张溢弛	3180103772	主节点模块开发,miniSQL 中的 Interpreter 模块，API 模块，Catalog 和 Buffer 模块编写
张琦	3180103162	从节点模块开发，miniSQL 中的 Record 和 Buffer 模块的开发
聂俊哲	3180103501	客户端与通信协议模块的开发，miniSQL 中的 Index 模块开发

二、总体设计

2.1 系统总体架构设计

系统的**总体架构设计**如下图所示：



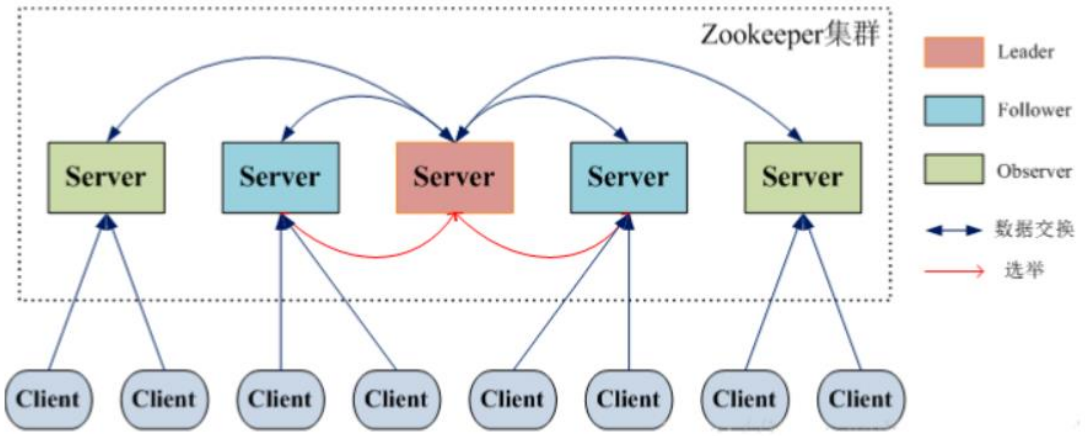
系统整体架构分为 Zookeeper 集群管理、Master、Region Server 和 Application 四个模块。其中，Region Server 底层由 miniSQL 提供服务，Application 底层由 Client 提供服务。

2.1.1 Zookeeper 集群管理

ZooKeeper 是一个分布式协调服务的开源框架。主要用来解决分布式集群中应用系统的一致性的问题，例如怎样避免同时操作同一数据造成脏读的问题。

ZooKeeper 本质上是一个分布式的小文件存储系统。提供基于类似于文件系统的目录树方式的数据存储，并且可以对树种 的节点进行有效管理。从而来维护和监控你存储的数据的状态变化。将通过监控这些数据状态的变化，从而可以达到基于数据的集群管理。诸如：统一命名服务（dubbo）、分布式配置管理（solr 的配置集中管理）、分布式消息队列（sub/pub）、分布式锁、分布式协调等功能。本系统中的 Zookeeper **集群管理提供两种类型的服务**

- Region Server 管理，Master 和 Region Server 监控 Zookeeper 中的目录。知道 Region 集群中有哪些服务器。当 Region Server 崩溃时，通过 Zookeeper 可以通知 Master，Master 做出适当的调整（容错容灾）
- 小规模数据存储，存储了从节点中的数据表等元信息，用于进行 Region Server 的调度管理。



2.1.2 Master 服务器

Master(主服务器，主节点)的主要功能有如下几条：

- 负责管理和维护表的分区信息（或者分布信息）等元数据信息。
- 维护 Region 服务器列表。
- 分配 Region（简单起见，Region 可以直接对应一个 Table， 而不需要进行切分，也不需要分裂和合并）
- 实现不同 Region 服务器之间的负载均衡管理用户对表的增加、删除、修改、查询等操作
- 对发生故障失效的 Region 服务器上的 Region 进行迁移

2.1.3 Region 服务器

Region(从节点)的主要功能有

- Region 服务器负责存储和维护分配给自己的 Region， 处理来自客户端的读写请求。
- 简单起见，Region Server 利用 miniSQL 来管理 Region， 负责 MiniSQL 的启动和管理，和 Client 的通信。
- Region 服务器提供缓存机制，对一些出现频率较高的查询进行缓存，可以优化一些高频率的查询的查询速度。

2.1.4 客户端

客户端并不是直接从 Master 主服务器上读取数据，而是在获得 Region 的存储位置信息后，直接从 Region 服务器上读取数据。

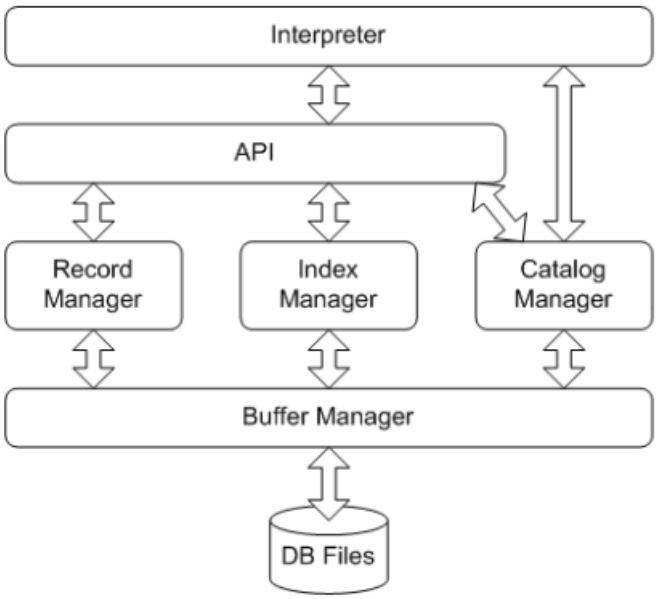
客户端可以不依赖 Master， 可以通过 Zookeeper 来获得 Region 位置信息（需要设计一套定位机制）或者从 Master 中获得，大多数客户端甚至从来不和 Master 通信，这种设计方式使得 Master 负载很小。

为减轻 Master 负担， 在客户端采用了一定的**缓存机制**，保存 Table 定位信息。更详细的内容参见第三部分模块与功能详细设计部分。

2.2 miniSQL 架构设计

miniSQL 是本分布式数据库系统项目中的核心，是运行在从节点中的单个数据库，并且可以执行一系列 SQL 语句并完成对应的操作，是一个精简的单用户 SQL 引擎，允许用户通过字符界面输入 SQL 语句实现数据库的一系列操作。

本项目中我们采用 Java 对原本的基于 C++的 miniSQL 数据库系统进行了重构，并做了如下架构设计：



主要需要实现的模块有 Interpreter，API，Record Manager，Buffer Manager 和 Catalog Manager，BufferManager 等等，主要需要实现的功能和需求有：

- **数据类型**：支持的数据类型是 `int,float,char(n)`等三种类型，并且 char 类型的长度在 1 到 255 之间
- 表的定义：一张表最多定义 32 个属性，属性可以设置为 unique 和 primary key
- **索引的建立和删除**：对于一张表的主属性自动建立 B+树索引，对于声明为 unique 的属性可以通过 SQL 语句来建立 B+树的索引，所有的索引都是单属性单值的
- **查找记录**：查找记录的过程中可以通过用 and 进行多个条件的连接
- 插入和删除记录：插入只支持单条记录的插入，删除操作支持一条和多条记录的删除
- 数据文件由一个或多个数据块组成，块大小应与缓冲区块大小相同。一个块中包含一条至多条记录，为简单起见，只要求支持定长记录的存储，且不要求支持记录的跨块存储
- 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块，块的大小应为文件系统与磁盘交互单位的整数倍，一般可定为 4KB 或 8KB
- 本系统主要通过输入一系列 SQL 语句执行来完成相应的操作和功能，SQL 语

句支持单行和多行的输入，最后必须用分号结尾作为 SQL 语句结束的标志

三、模块与功能详细设计

3.1 主节点设计

主服务器 Master Server 主要**负责管理和维护表的分区信息**，维护 Region Server 列表，分配 Region，使得负载均衡。为减轻 Master 负担，在后续的设计提升中，客户端并不是直接从 Master 主服务器上读取数据，而是在获得 Region 的存储位置信息后，直接从 Region 服务器上读取数据。

具体来讲，Master Server 需要实现以下的设计功能。管理用户对表的增加、删除、修改、查询等操作。

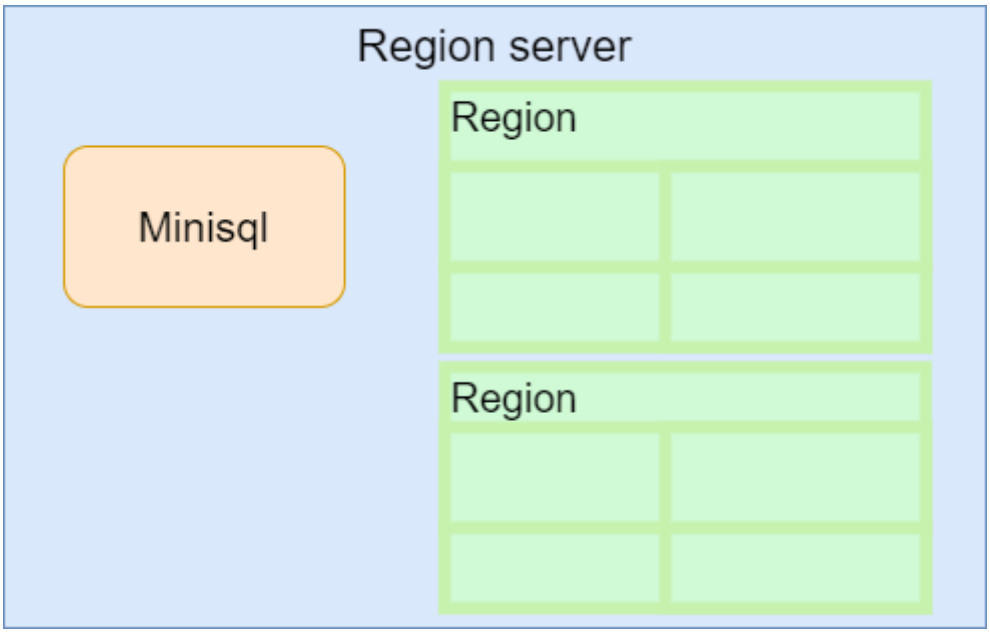
- 实现不同 Region 服务器之间的负载均衡。
- 在 Region 分裂或合并后，负责重新调整 Region 的分布。
- 对发生故障失效的 Region 服务器上的 Region 进行迁移。
- 主节点中维护了数据表的分布信息，比如每张数据表所处的从节点，表中的行列信息等元信息
- 维护一个从节点服务器列表，并管理从节点的元信息，负责从系欵点服务器的调度和分配。
- 有一定的容错容灾能力，当从节点发生故障的时候可以对失效的服务器进行数据的迁移。

3.2 从节点设计

Region server，即**从节点服务器**，可以说是分布式数据库中最核心的模块。Region server 负责存储和维护分配给自己的 Region，并利用 MiniSQL 来管理 Region。每个 Region server 负责 MiniSQL 的启动和管理，响应来自客户端的读写请求，将最终的结果返回客户端。

我们仿照 HBase 进行设计，在 HBase 中是一张大表按照行来进行分区，而在本分布式数据库中，每个 Region 都对应数据库中的一张表格，按表格来进行分区。所有表格的信息都存储在 Master 节点中的 META 元数据表中，Client 可以通过 Master 查询 META 元数据表快速的定位每个表格在哪个 Region 中，并随后访问对应的 Region server 发送读写请求。

为减轻 Master 负担，在客户端可以有缓存，保存每张表格的定位信息，从而直接访问对应的 Region server，减少客户端对 Master 的访问。



3.3 miniSQL 设计

3.3.1 Interpreter

`Interpreter` 模块直接与用户交互，主要实现以下功能：

- 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语义正确性，对正确的命令调用 API 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。

3.3.2 API

API 模块是整个系统的核心，其主要功能为提供执行 SQL 语句的接口，供 Interpreter 层调用。

该接口以 Interpreter 层解释生成的命令内部表示为输入，根据 Catalog Manager 提供的信息确定执行规则，并调用 Record Manager、Index Manager 和 Catalog Manager 提供的相应接口进行执行，最后返回执行结果给 Interpreter 模块。

3.3.3 Catalog Manager

Catalog Manager 负责管理数据库的所有模式信息，包括：

- 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。
- 表中每个字段的定义信息，包括字段类型、是否唯一等。
- 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 Interpreter 和 API 模块使用。

3.3.4 Record Manager

Record Manager 负责**管理记录表中数据的数据文件**。主要功能为实现数据文件的创建与删除（由表的定义与删除引起）、记录的插入、删除与查找操

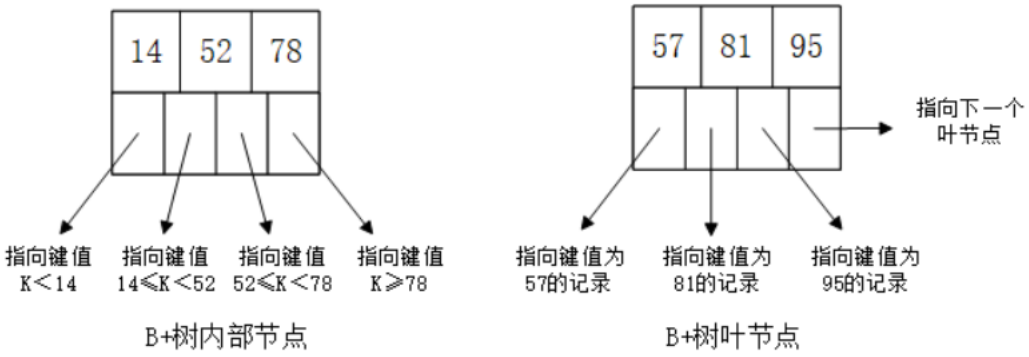
作，并对外提供相应的接口。其中记录的查找操作要求能够支持不带条件的查找和带一个条件的查找（包括等值查找、不等值查找和区间查找）。

数据文件由一个或多个数据块组成，块大小应与缓冲区块大小相同。一个块中包含一条至多条记录，为简单起见，只要求支持定长记录的存储，且不要求支持记录的跨块存储。

3.3.5 Index Manager

Index Manager 负责 B+树索引的实现，实现 B+树的创建和删除（由索引的定义与删除引起）、等值查找、插入键值、删除键值等操作，并对外提供相应的接口。

B+树中索引节点大小应与缓冲区的块大小相同(4096KB)，B+树的叉数由节点大小与索引键大小计算得到。B+树的结构可以用下面的图来表示：



当然上面只是 n=3 的情况，在本系统的 miniSQL 模块开发中，一个节点的大小和 Buffer 模块的块大小是相同的，因此 n 会比 3 大得多。

3.3.6 Buffer Manager

Buffer Manager 负责**缓冲区**的管理，主要功能有：

- 根据需要，读取指定的数据到系统缓冲区或将缓冲区中的数据写出到文
- 实现缓冲区的替换算法，当缓冲区满时选择合适的页进行替换
- 记录缓冲区中各页的状态，如是否被修改过等

- 提供缓冲区页的 pin 功能，及锁定缓冲区的页，不允许替换出去

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块，块的大小应为文件系统与磁盘交互单位的整数倍，一般可定为 4KB 或 8KB，本项目中采用 4096KB 作为块的大小。

3.3.7 数据库文件系统

DB Files 也就是数据库的文件系统，指构成数据库的所有数据文件，主要由记录数据文件、索引数据文件和 Catalog 数据文件组成。

3.3.8 支持的 SQL 语句格式

- 创建表的语句：关键字为 create table 具体的语法格式如下

```
create table table_name(
    attribution1 date_type1,
    attribution2 data_type2 (unique),
    ……
    primary key(attribution_name)
);
```

- 创建索引的语句：关键字为 create index，具体的语法格式如下

```
create index index_name on table_name (attribution_name);
```

- 删除表的语句

```
drop table table_name;
```

- 删除索引的语句

```
drop index index_name;
```

- 选择语句，关键字为 select ，只支持 select *即显示全部属性

```
select * from table_name;
select * from table_name where conditions;
```

- 插入记录语句，关键字为 insert into，支持 where 的条件表达式，语法格式如下

```
insert into table_name values (value1, value2, value3.....);
```

- 删除记录语句，关键字为 delete from，语法格式如下

```
delete from table_name;
delete from table_name where conditions;
```

- 执行 SQL 脚本文件

```
exefile file_name
```

3.4 客户端设计

本项目中的客户端是提供给用户的操作界面，用于和用户进行交互，并和服务端建立稳定的网络通信连接，**基于 RPC 通信协议进行通信**。

3.4.1 客户端工作原理

客户端并不是直接从 Master 主服务器上读取数据，而是在获得 Region 的存储位置信息后，直接从 Region 服务器上读取数据，也可以直接使用 Zookeeper 来获取 Region 的位置信息。

ZooKeeper 集群启动之后，将等待客户端连接。客户端将连接到 ZooKeeper 集合中的一个节点。它可以是领导或跟随者节点。一旦客户端被连接，节点将向特定客户端分配会话 ID 并向该客户端发送确认。如果客户端没有收到确认，它将尝试连接 ZooKeeper 集合中的另一个节点。 一旦连接到节点，客户端将以有规律的间隔向节点发送心跳确认，以确保连接不会丢失。

3.4.2 RPC 通信协议

RPC 协议是一种网络通信协议，RPC 协议假定一些传输协议的存在，如 TCP 或 UDP，为通信程序之间携带信息数据。在 OSI 模型中，RPC 跨越了传输层和应用层。RPC 使得开发包括网络分布式多程序在内的应用程序更加容易。

RPC 采用 C/S 架构。请求程序就是一个客户端，而服务提供程序就是一个服务器。首先，调用进程发送一个有进程参数的调用信息到服务进程，然后等待应答信息。在服务器端，进程保持睡眠状态直到调用信息的到达为止。当一个调用信息到达，服务器获得进程参数，计算结果，发送答复信息，然后等待下一个调用信息，最后，客户端调用过程接收答复信息，获得进程结果，然后调用执行继续进行。

本项目中客户端和服务器的通信拟采用 RPC 协议进行，同时采用 Thrift 语言来描述自定义通信格式的具体内容。

3.4.3 客户端缓存机制

客户端来进行具体的查询之前需要获得对应的表的 Region Server 的相关信息，解析这一位置信息往往是比较耗时的

因此我们为客户端设立了缓存机制，在内存中维护一个记录已知 Region Server 元信息的 HashMap，每次操作先从缓存中查询目标服务器是否已经获得其相关信息，这样可以较大地提高客户端的工作效率。

3.5 详细功能设计

本分布式数据库主要实现了数据表分区，副本维护，负载均衡和容错容灾等分布式数据库所必须具备的基本功能。

3.5.1 数据表分区

每个 Region Server 负责维护若干个数据表，并使用 miniSQL 对其进行维护，管理，添加，删除，修改和查询等操作，同时 Master Server 上也维护了每个表中数据表的分布情况，并可以进行统一的管理和调度。

3.5.2 副本维护

本项目采用主从复制的策略，在一系列 Region Server 中进行副本的保存和维护，并选择其中一个表为主副本，负责副本的复制操作，定期进行数据库文件副本的维护工作。

3.5.3 负载均衡

Master Server 可以对 Region Server 进行统一的管理和调度，当检测到一台 Region Server 繁忙的时候，Master 会将其中的某些 Region 进行重新分配，同时对于客户端发出的请求也要进行适当的调度，防止出现部分 Region Server 过热而其他 Region Server 处于饥饿状态。

Region Server 之间也需要进行数据的传输，这一个过程中也需要进行负载均衡的操作。

3.5.4 容错容灾

当某个 Region Server 失效或者发生系统崩溃等情况时，Master Server 需要将 Region 进行重新分配。

四、项目计划与进度安排

4.1 系统测试用例设计

以下给出一些测试数据的样例，最终报告中进行的测试将更加复杂，相应的测试数据也将会更加多样。

用例编号	01
描述	考察 int 类型上的等值条件查询
输入信息	select from student2 where id=1080100245;
假设	无异常
输出	返回查询结果
备注	无

用例编号	02
描述	考察 float 类型上的等值条件查询，观察数量
输入信息	select from student2 where score=98.5;
假设	无异常
输出	返回查询结果
备注	无

用例编号	03
描述	考察 char 类型上的等值条件查询，此处需观察执行时间 t1
输入信息	select from student2 where name='name245';
假设	无异常
输出	返回查询结果
备注	无

用例编号	04

描述	考察 int 类型上的不等条件查询，观察数量
输入信息	select from student2 where id<>1080109998;
假设	无异常
输出	返回查询结果
备注	无

用例编号	05
描述	考察 char 类型上的不等条件查询，观察数量
输入信息	select from student2 where name<>'name9998';
假设	无异常
输出	返回查询结果
备注	无

用例编号	06
描述	考察多条件 and 查询，观察数量
输入信息	select from student2 where score>80 and score<85;
假设	无异常
输出	返回查询结果
备注	无

用例编号	07
描述	考察 primary key 约束冲突（或 unique 约束冲突）
输入信息	insert into student2 values(1080100245,'name245',100);
假设	无异常
输出	报 primary key 约束冲突
备注	无

用例编号	08

描述	考察索引的创建，unique key 才能建立索引
输入信息	create index stuidx on student2 (name);
假设	无异常
输出	如果 name 是 unique，则创建成功
备注	无

用例编号	**09**
描述	考察在建立 b+树后再插入数据，b+树有没有做好 insert
输入信息	insert into student2 values(1080197996,'name97996',100);
假设	无异常
输出	返回插入结果
备注	无

用例编号	**10**
描述	考察 delete，同时考察 b+树的 delete
输入信息	delete from student2 where name='name97996';
假设	无异常
输出	返回删除结果
备注	无

用例编号	**12**
描述	考察 drop index
输入信息	drop index stuidx
假设	无异常
输出	返回删除结果
备注	无

用例编号	**13**

描述	考察 drop table
输入信息	drop table student2;
假设	无异常
输出	返回删除结果
备注	无

4.2 项目规划

计划时间段	进度里程碑
5 月初	完成系统设计，任务分配，搭建系统基本框架
5 月中旬-5 月下旬	基本完成代码的合作开发
6 月初	进行系统的集成与测试
6 月中旬	完成系统的各类文档与答辩 PPT，进行最终验收

五、总结

通过该项目，我们可以对分布式数据库能有一个大致的了解，对于鲁老师在课程中讲解的理论知识也能进行实践。

大规模信息技术构建导论这门课能以这个项目作为收尾，对于我们每个软件工程专业的学生来讲，既是不错的项目经历，又能初窥工业界在大规模数据库中使用的主流的技术，这是难能可贵的。

同时，项目给我们预留了很大的发挥空间，我们会在完成第三板块《模块与功能详细设计》的基础上，在运用软件工程的知识，在小组合作的基础上，对项目进行若干次功能迭代，最终实现一个稳定的、符合设计预期的分布式 miniSQL。