

电子科技大学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

Linux 环境高级编程



课题名称 YA-DB: 微型数据库系统

学 院 计算机科学与工程学院

学 号 202222080626

作者姓名 王乐卿

指导教师 李林

目 录

第一章 需求分析与可行性研究	1
1.1 目标概述	1
1.2 可行性研究与分析	1
1.3 项目调研	1
1.3 需求分析	2
第二章 概要设计	4
2.1 系统架构	4
2.2 系统泳道	5
第三章 详细设计及实现	7
3.1 数据表文件.table	7
3.2 持久化 B+树搜索	8
3.3 持久化 B+树插入	10
第四章 系统测试	11
4.1 系统测试概述	11
4.2 系统测试环境	11
4.3 系统测试用例及结果	12
第五章 开发计划	24

第一章 需求分析与可行性研究

1.1 目标概述

本项目的主要任务是实现一个简易的微型数据库系统，称为 YA-DB，用户通过启动 YA-DB 客户端，通过输入类 SQL 指令，实现新建表、新建索引、插入数据和查询数据等功能。

1.2 可行性研究与分析

针对本项目的课题目标，主要从技术实现、经济成本、社会因素等几个方面来分析本课题的可行性：

（一）技术实现方面：本人具有专业的计算机科学基础，在项目开发过程中遇到难题时，有能力开发文档解决开发问题。此外，笔者在字节跳动实习过程中有过后台数据库开发经验，对项目开发的技术有一定的掌握，因此技术方面是可行的。

（二）经济成本方面：本项目采用 C++ 开发，部署至远程服务器上进行测试，因此本项目开发的经济成本主要来源于服务器的开销，在短期内是可控的。

（三）社会因素方面：本项目并不提供实际业务，而只是给出数据存储的一种的解决方案，因此不涉及非法使用等侵权问题；此外，本项目的研究逻辑和代码实现都由作者亲自实现，不存在抄袭等不符合道德和法律的社会问题。

综上所述，本次项目在技术实现、经济成本和社会因素方面都是可行的。

1.3 项目调研

本项目通过快速原型法进行开发，充分调研了市面上如 MySQL 等各种成熟数据库软件，在本节对调研结果做简要介绍。

MySQL 中存在两类常用殷勤，分别是 MyISAM 和 InnoDB 引擎，InnoDB 引擎对于主键索引的实现采用聚集索引，其余采用非聚集索引，而 MyISAM 引擎对所有索引均采用非聚集索引。在 InnoDB 的 B+树存储中，其叶子节点存储了所有数据，因此当主键索引时，可直接从叶子节点中获取目标数据；而采用非主键索引时，其 B+树的叶子节点中仅存储了对应的主键的 ID，需要二次调用主键索引查询，才可以获取目标数据。

此外，InnoDB 引擎引入了表空间的概念，本段介绍 MySQL 5.66 版本引入的

独立表空间。表空间是一个抽象的逻辑概念，InnoDB 把数据保存在表空间，本质上是一个或多个磁盘文件组成的虚拟文件系统。InnoDB 引擎会将数据表存储到数据表对应的独立表空间中去，数据表的独立表空间文件的名称和数据表名相同，拓展名为.ibd，该文件中同时存储了表索引和表数据。

此外，在 Linux 计算机存储中，以页为单位管理内存，无论是将磁盘中的数据加载到内存中，还是将内存中的数据写回磁盘，操作系统都会以页面为单位进行操作，为了使用顺序 IO 提高磁盘加载速度，InnoDB 引入页、区和段等概念，加快磁盘加载速度。

在本项目的设计中，采用主键索引为聚集索引，而非主键索引为非聚集索引，需要进行二次索引的思想，且将索引和数据文件以若干 B+树的形式，存储于自定义后缀的.table 文件中，具体的文件空间分配将在下文讲解。

1.3 需求分析

本项目通过快速原型法进行开发，在充分调研了市面上各种成熟数据库软件后，对项目进行简单的需求分析并进行用例图设计，其基本用例图如图 1-1 所示：

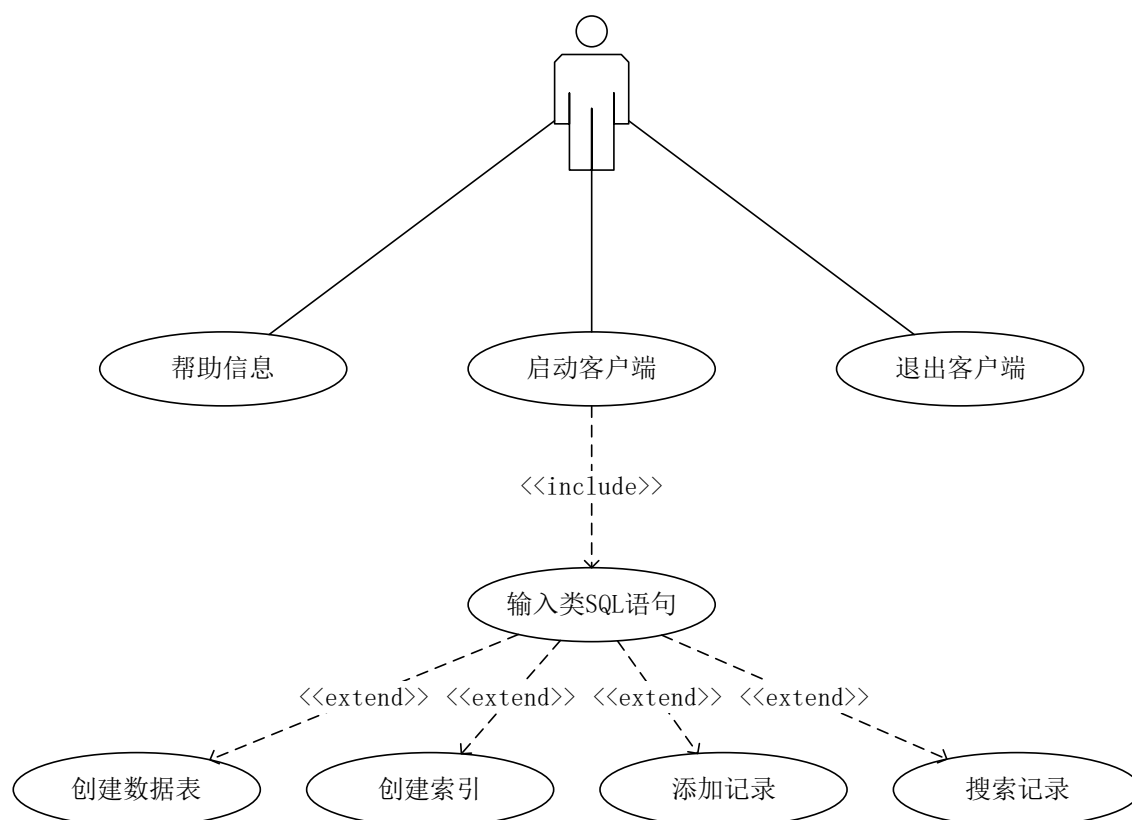


图 1-1 YR-DB 基本用例图

当用户启动 YA-DB 客户端后,通过输入类 SQL 语句,服务端将利用词法分析器对 SQL 语句进行解释,目前支持执行如下四种操作:

(1) 新建数据表

用户通过输入 CREATE TABLE 的类 SQL 语句,新建一个数据表,服务端执行后将在约定目录下新建一个与数据表同名的.table 文件,存储该数据表的数据和索引信息。该表支持任意列和任意行,每一列的属性约定为 int64_t。

(2) 新建索引

用户通过输入 CREATE INDEX 的类 SQL 语句,为表格的某一个属性建立索引结构,以实现快速搜索。继承 InnoDB 引擎的设计思想,YA-DB 约定第一列的为数据唯一的主键索引列,即便用户未创建索引,也会默认创建主键索引。用户新建的索引仅索引到对应的主键值,需要对主键进行二次索引,才可以获得最终数据。

(3) 添加记录

用户可以通过输入 INSERT 的类 SQL 语句,对指定数据表插入数据,服务端将数据插入主键的聚集索引中,同时更新所有已经创建的非主键索引,保证数据搜索的一致性。

(4) 搜索记录

用户可以通过输入 SELECT 的类 SQL 语句,对数据进行精确搜索和范围搜索。当搜索的列存在索引时,将使用索引加速搜索,否则,将对主键索引进行全表遍历。

第二章 概要设计

2.1 系统架构

为了提高代码的结构性、可重用性、可读性和可维护性，YA-RPC 系统基于分层架构模式的思想来构建，具体的系统架构如图 2-1 所示：

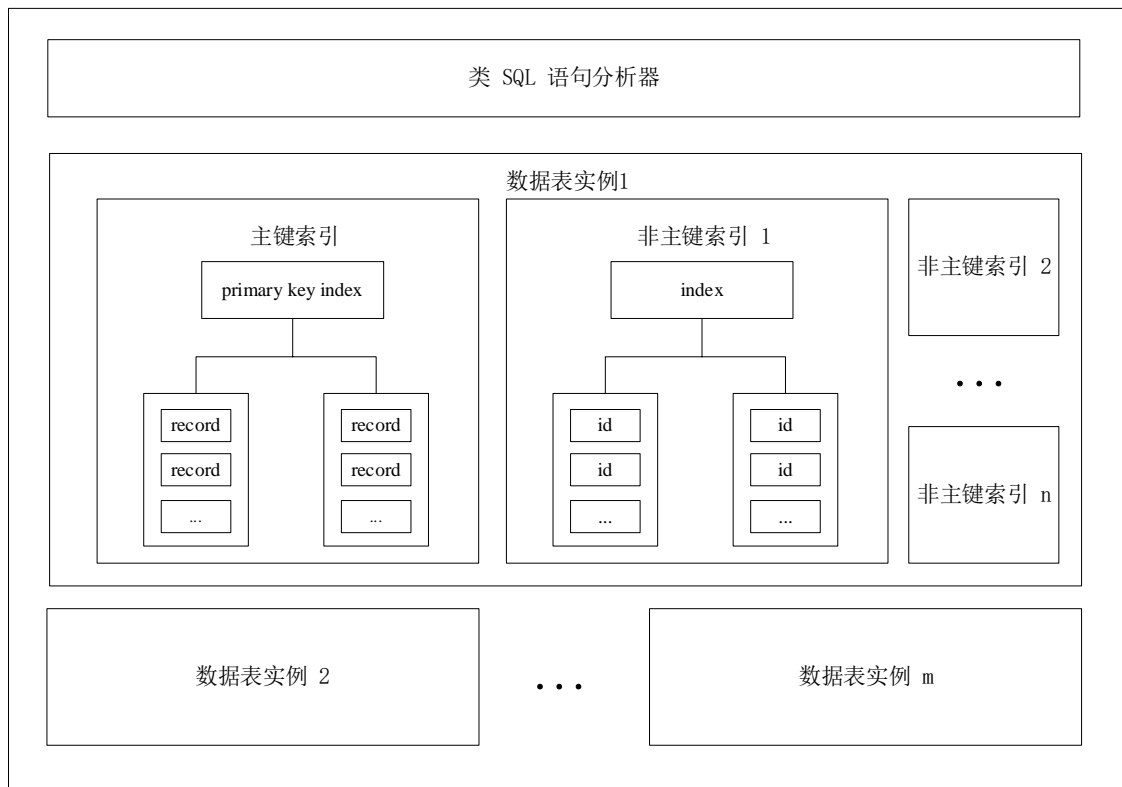


图 2-1 YA-DB 系统架构图

结合上述分离架构思想，将 YA-DB 系统分为三层：类 SQL 语句分析器对用户输入的一类 SQL 语句进行词法和语法分析，定位到特定的数据表实例，基于操作的不同对索引实例进行操作。

(1) 类 SQL 语句分析器：YA-DB 项目目前支持创建数据表、创建索引、插入数据和删除数据四种功能，类 SQL 语句分析器将对用户输入的一类 SQL 语句进行词法和语法分析，执行特定的功能。

(2) 数据表：用户可以自由创建任意数量的数据表，YA-DB 将对创建的所有数据表创建一个.table 文件。该文件中存储了数据表的元数据和两类索引数据，分别是主键索引和非主键索引，二者以 B+树的形式存储于.table 文件中，数据表实例，将根据用户的一类 SQL 请求，对不同的索引进行相应的操作。

(3) 索引：YA-DB 的索引分为主键索引和非主键索引，主键索引的 B+树的叶子节点存储了数据表的行数据，而非主键索引的 B+树的叶子节点仅存储对应的主键数据，需要进行二次查找。

YA-DB 默认采用 15 阶 B+树，因此即便是千万量级的数据，也仅需要 5 次节点 IO 即可查询到，因此，YA-DB 二次索引设计在保证查询效率的同时，有效的节约了索引的存储空间。当然，值得一提的是，用户可以通过修改默认配置，自由选择 B+树的阶数。

2.2 系统泳道

为了更好的描述 YA-DB 系统的执行流程，建模系统泳道图，如图 2-1 所示：

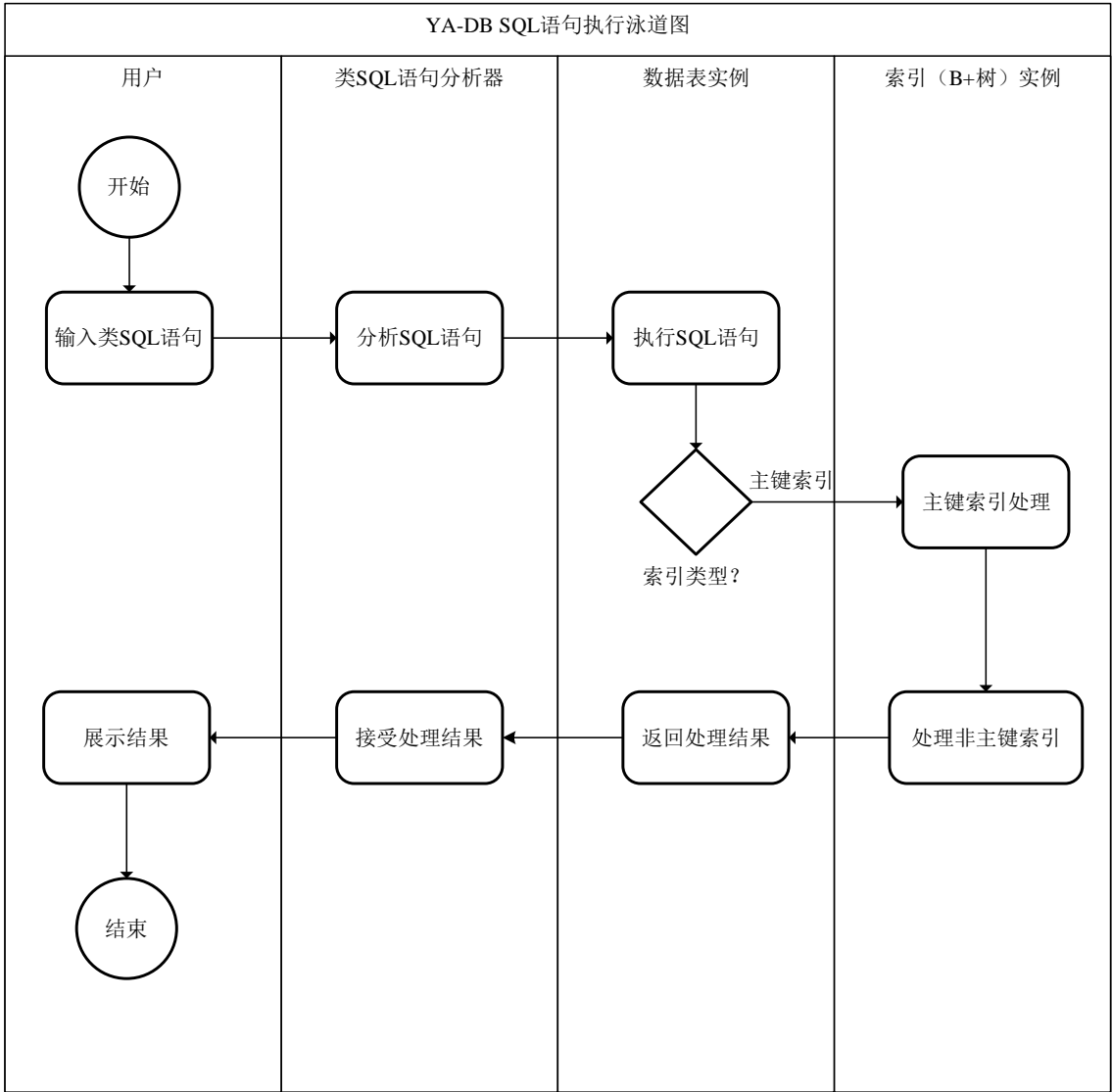


图 2-1 YA-DB 系统 SQL 语句执行流程图

当用户启动 YA-DB 客户端后,通过输入类 SQL 语句,服务端将利用词法分析器对 SQL 语句进行分析,构建指定的数据表实例,每个数据表实例从同名的.table 文件中反序列化获取,该文件的字节分配设计将在详细设计中介绍。构建数据表实例后,将根据执行的 SQL 语句类型对不同的索引实例进行操作,每个索引实例都是 B+树实例。

当用户执行插入操作时,首先构建主键索引实例,将数据插入到主键索引中,然后通过数据表元数据,获取已经创建索引的数据行,构建对应的非主键索引实例,更新非主键索引实例。

当用户执行搜索操作时,若搜索列是主键索引,则构建主键索引实例进行搜索;若搜索列是非主键索引,则构建非主键索引实例,获取主键值,随后构建主键索引实例进行二次搜索;若搜索列既不是主键索引,也不是非主键索引,则构建主键索引实例,进行全表搜索。

当执行完用户的类 SQL 语句请求后,将执行结果做可视化处理,响应给用户。

第三章 详细设计及实现

3.1 数据表文件.table

用户可以自由创建任意数量的数据表，YA-DB 将对创建的所有数据表创建一个对应的同名.table 文件，该文件中存储了数据表的元数据和两类索引数据，数据表文件的示意图如图 3-1 所示：

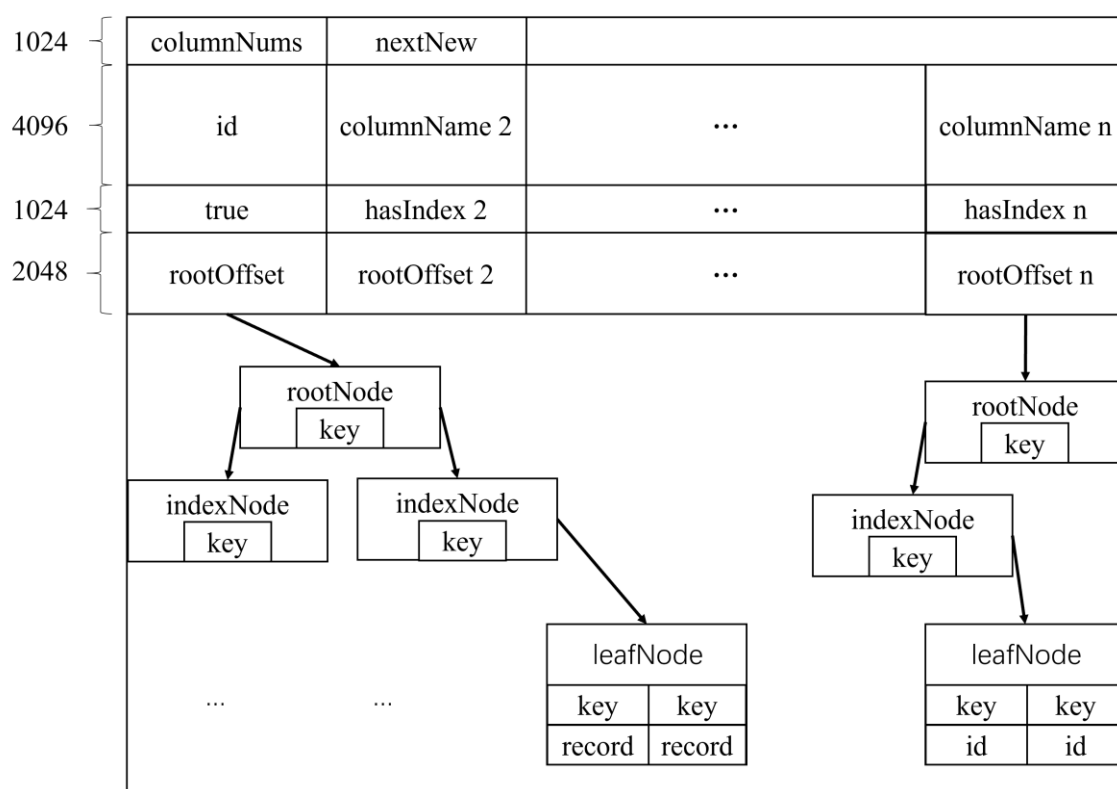


图 3-1 数据表.table 文件结构示意图

数据表文件主要分为三类，分别是数据表元数据，主键索引数据和其余非主键索引数据：

(1) 数据表元数据：元数据共 8092 字节，位于文件的头部。分配 1024 字节，存储特殊元数据，目前存储了两个数据，分别是表格的记录的列的数量和下一个新建节点的起始偏移量。分配 4096 字节存储列名，YA-DB 项目默认首列的列名必须是 id，YA-DB 将其作为主键建立聚集索引。分配 1024 的字节，存储对应的列是否创建了索引。分配 2048 的字节，存储对应的列的索引 B+树的根节点在文件中的偏移量，仅 hasIndex 为 true 的列才存储该数据，否则置为空。

(2) 主键索引：主键索引被实现为聚集索引，因此其叶子节点存储了记录行

数据 record。当建立主键索引的 B+树时，首先从数据表元数据中获取其根节点在文件中的偏移量，随后从文件中反序列化出该根节点，此时则完成主键索引 B+树的构建。

(3) 非主键索引：非主键索引被实现为非聚集索引，因此其叶子节点进出了主键索引，即 id 的值，需要二次索引才能获取叶子节点的数据。当建立非主键索引的 B+树时，先通过数据表元数据判断该列是否存在索引，若存在，则再从数据表元数据中获取非主键索引 B+树的根节点在文件中的偏移量，随后从文件中反序列化出该根节点，此时则完成非主键索引 B+树的构建。

值得注意的是，文件中的数据是紧凑存储的，示意图中为了便于理解，并未画的紧凑，将在后续小节以具体的实例讲解主键索引和非主键索引在文件中具体的存储，以及二者与数据表元素的联动。

3.2 持久化 B+树搜索

YA-DB 将对创建的所有数据表创建一个对应的同名 .table 文件，该文件中存储了数据表的元数据和两类索引数据，下面以一个实例讲解两类索引在磁盘上的持久化存储，以及和数据表元数据的联动，.table 文件实例图如图 3-2 所示。

columnNums = 3 nexNew = 9392			
columnName	id	age	name
hasIndex	true	true	false
rootOffset	8792	9292	
SelfOffset: 8192 rightBrother: 8492 Size: 300		SelfOffset: 8492 rightBrother: NULL Size: 300	
id 1	id 2	id 3	id 4 id 5
(1, 26, tom)	(2, 20, jack)	(3, 18, sam)	(4, 36, alice) (5, 14, bob)
SelfOffset: 8792 Size: 100		SelfOffset: 8892 rightBrother: 9092 Size: 200	
id 3		age 14 age 18	
8192	8492	(5) (3)	
SelfOffset: 9092 rightBrother: NULL Size: 200		SelfOffset: 9292 Size: 100	
age 20 age 26 age 36		age 20	
(2) (1) (4)		8892 9092	
...			

图 3-2 .table 文件示例

索引共分为两类，分别是主键索引和非主键索引，而两种索引又分为内部节点和叶子节点，其中根据树的高度，根节点可能是叶子节点，也可能是内部节点。

在实际存储的过程中，将为每种节点分配其最大可能需要的存储空间，根据 B+树的阶数和数据行的列数的不同，节点的大小均不相同，此处更清晰易懂的讲解存储逻辑，将各个节点的大小定义为：主键索引叶子节点（300 bytes）、非主键索引叶子节点（200 bytes）、内部节点（100 bytes）。

本次讲解的 B+树的阶数定义为 5 阶，数据表共三列，分别是 id、age 和 name，共插入了 5 条数据，其中为 id 建立主键的聚集索引，为 age 建立列的非聚集索引，图 3-2 展示了上述操作之后的文件存储空间分配：

(1) 数据表元数据：共 8192 bytes，前 1024 bytes 中，columnNums 指列的数量，共 3 列，newNew 是下一个将分配的节点在文件中的起始偏移量，其余空间置为空；随后的 4096 bytes 中存储了列名，已知当前数据表共三列，分别是 id、age 和 name；随后的 1024 字节存储了列是否建立了索引，已知 id 和 age 建立了索引，因此将其置为 true，name 未建立索引，因此置为 false；最后的 2048 字节存储了索引 B+树的根节点在文件中的起始偏移量，id 为主键索引，其根节点的文件起始偏移量为 8792，age 索引的根节点的文件起始偏移量为 9292。

(2) 主键索引搜索：当搜索 id=1 的行记录时，首先根据 rootOffset 获取主键索引的根节点的文件起始偏移量为 8792，随后使用 lseek 函数定位到 8792，由于根节点的大小为 100 bytes，于是读取 100 bytes 的数据，使用反序列化构建根节点实例。

然后通过比较根节点的 key 和输入 id 的值的大小，定位到特定的孩子节点，此处由于 $1 < 3$ ，于是定位到孩子节点的文件起始偏移量为 8192，使用 lseek 函数定位到 8192，由于主键索引的叶子节点的大小为 300 bytes，于是读取 300 bytes 的数据，使用反序列化构建主键索引的叶子节点实例。

最后在叶子节点中比较 id 和 key 的大小，主键索引的索引即数据，其叶子节点中可以直接获取到记录行 (1, 26, tom)。

(3) 非主键索引搜索：当用户搜索 age = 26 的行记录时，首先通过 hasIndex 的数据表元数据判断 age 列是否存在索引，此处为 true，通过 rootOffset 的数据表元数据，获取 age 列索引的根节点的文件起始偏移量为 9292，随后使用 lseek 函数定位到 9292，由于根节点的大小为 100 bytes，于是读取 100 bytes 的数据，使用反序列化构建根节点实例。

接着通过比较根节点的 key 和输入的 age 的值的大小，定位到特定的孩子节点，此处由于 $26 > 20$ ，于是定位到孩子节点的文件起始偏移量为 9092，使用 lseek 函数定位到 9092，由于非主键索引的叶子节点的大小为 200 bytes，于是读取 200 bytes 的数据，使用反序列化构建非主键索引的叶子节点实例。

然后在叶子节点中比较 id 和 key 的大小，非主键索引的叶子节点中存储的是对应数据行的主键索引值，即 id 值，本次示例在叶子节点中获取 id 值为 1。

最后，通过该 id 值再次通过主键索引进行二次索引，获取最终的目标记录行 (1, 26, tom)。

(4) 无主键索引搜索：当用户搜索 `name = tom` 时，首先通过 `hasIndex` 的数据表元数据判断 `name` 列是否存在索引，此处为 `false`，即未对 `name` 建立索引，于是直接建立主键索引，定位到主键索引的最左侧的叶子节点，通过 `rightBrother` 指针不断向右搜索，直到不存在右兄弟叶子节点，对数据表进行全表搜索。

3.3 持久化 B+树插入

本次讲解的 B+树的阶数定义为 5 阶，数据表共三列，分别是 `id`、`age` 和 `name`，图 3-2 中插入了 5 条数据，其中为 `id` 建立主键的聚集索引，为 `age` 建立列的非聚集索引，为了讲解 B+树的插入，再次插入 2 两条数据，分别是 (6, 53, joey) 和 (7, 12, linda)，插入后的 `.table` 文件示意图如图 3-3 所示。

columnNums = 3 nexNew = 9692			
columnName	id	age	name
hasIndex	true	true	false
rootOffset	8792	9292	
SelfOffset: 8192 rightBrother: 8492 Size: 300		SelfOffset: 8492 rightBrother: 9392 Size: 300	
id 1 id 2		id 3 id 4	
(1, 26, tom) (2, 20, jack)		(3, 18, sam) (4, 36, alice)	
SelfOffset: 8792 Size: 100		SelfOffset: 8892 rightBrother: 9092 Size: 200	SelfOffset: 9092 rightBrother: NULL Size: 200
id 3 id: 5		age 12 age 14 age 18	age 20
8192 8492 9392		(7) (5) (3)	age 20
		(2) (1) (4) (6)	8892 9092
SelfOffset: 9392 rightBrother: NULL Size: 300		...	
id 5 id 6 id 7			
(5, 14, bob) (6, 53, joey) (7, 12, linda)			

图 3-3 插入数据后 `.table` 文件示例

首先建立主键索引，将数据插入主键索引中，由于 B+树的阶数为 5 阶，叶子节点分裂，为其在 `nexNew` 的起始偏移量后分配 300 bytes 的空间，同时更新其余涉及的节点。

随后，通过 `hasIndex` 判断哪些列建立了非主键索引，同时更新所有的非主键索引了，由于非主键索引的叶子节点未滿，不对其分裂。

第四章 系统测试

4.1 系统测试概述

本项目在开发过程中经过详细的单元测试。黑盒测试作为系统测试的常用方法之一，在黑盒测试中，不关心程序内部结构及特性，而仅仅对程序的接口进行测试，检测在按照特定的使用方式进行使用时，程序能否接受使用数据，并产生符合预期的输出结果。

本项目的主要任务是实现一个简易的数据库，称为 YA-DB，用户可以通过实现如下功能：

- (1) 创建任意列数、任意行数的数据表
- (2) 创建索引
- (3) 插入记录行
- (4) 搜索记录行，支持指定搜索和范围搜索

4.2 系统测试环境

本次系统测试在 ubuntu 18.04 进行，为了尽可能的展示测试用例，将建立两张表：

(1) **FuncTest**：该表用于进行功能测试，B+树阶数默认 5 阶，初始化后 4 列 1000 行数据，其表结构表 4-1 所示：

表 4-1 FuncTest 表实例

id	columnOne	columnTwo	columnThree
1	random	1	1
2	random	2	2
...

在后续的测试中，id 作为主键，以 1 开始顺序增长，后续将为 columnOne 建立索引，其数据随机，为了便于观察数据是否正确，将 columnTwo 和 columnThree 的值设置为与 id 相同。

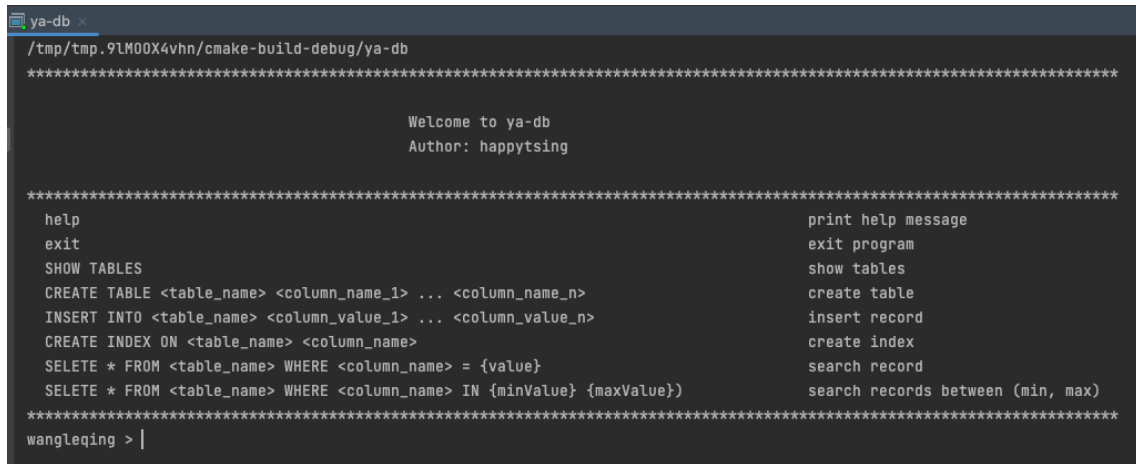
功能测试主要测试系统的功能是否可以正确运行。

(2) **LoadTest**：该表用于进行负载测试，B+树阶数默认 15 阶，共 100 列 1000000 行数据，其中 id 作为主键，以 1 开始顺序增长，其余数据全部随机。

主要用于测试系统在大量数据的情况下，其运行速度如何。

(3) 客户端

在用户启动客户端后，YD-DB 的启动和使用界面如图 4-1 所示：



```

ya-db
/tmp/tmp.9LM00X4vhn/cmake-build-debug/ya-db
*****
Welcome to ya-db
Author: happytsing

*****
help                                print help message
exit                                exit program
SHOW TABLES                        show tables
CREATE TABLE <table_name> <column_name_1> ... <column_name_n>      create table
INSERT INTO <table_name> <column_value_1> ... <column_value_n>      insert record
CREATE INDEX ON <table_name> <column_name>                          create index
SELETE * FROM <table_name> WHERE <column_name> = {value}             search record
SELETE * FROM <table_name> WHERE <column_name> IN {minValue} {maxValue}) search records between (min, max)
*****
wangleqing > |
  
```

图 4-1 YA-DB 客户端页面

4.3 系统测试用例及结果

(一) 功能测试

由于 100 列过长，不便于展示测试结果，因此构建一个 4 列的表 FuncTens 进行测试，其测试用例如表 4-1 所示：

表 4-1 功能测试用例

编号	内容描述	过程描述	预期结果	测试结果
101	创建表	创建表 FuncTest 和 LoadTest，将在默认文件夹 tables 下创建同名持久化存储文件 FuncTest.table 和 LoadTest.table	创建成功	与预期结果一致
102	展示表	展示当前创建的所有表	展示表成功	与预期结果一致
103	创建索引	为 FuncTest 的 columnOne 列创建索引	创建索引成功	与预期结果一致
104	插入数据	插入数据，同时将自动更新索引	插入数据并更新索引成功	与预期结果一致

105	精确主键索引搜索	使用主键索引进行精准搜索	搜索成功	与预期结果一致
106	精确非主键索引搜索	使用非主键索引进行精准搜索	搜索成功	与预期结果一致
107	精确无索引搜索	不使用索引进行精准搜索	搜索成功	与预期结果一致
108	范围主键索引搜索	使用主键索引进行范围搜索	搜索成功	与预期结果一致
109	范围非主键索引搜索	使用非主键索引进行范围搜索	搜索成功	与预期结果一致
110	范围无索引搜索	不使用索引进行范围搜索	搜索成功	与预期结果一致

(1) 用例 101

创建表 FuncTest，如图 4-2 所示：

```
wangleqing > CREATE TABLE FuncTest id columnOne columnTwo columnThree
CREATE TABLE FuncTest id columnOne columnTwo columnThree
finish in 0.000147 seconds.
```

图 4-2 创建表 FuncTest

定位到预定的文件夹 tables 查看，发现已经创建了和数据表同名的 FuncTest.table 文件，该文件中将存储索引和数据。

```
ubuntu@VM-16-2-ubuntu /tmp/tmp.9lM00X4vhn/tables
) ls
FuncTest.table LoadTest.table
```

图 4-3 展示表文件

以同样的方式创建表 LoadTest，由于过于冗长在此不做展示。

(2) 用例 102

查看当前创建的所有数据表，其结果如 4-4 所示：

```
wangleqing > SHOW TABLES
SHOW TABLES
+-----+
| LoadTest |
| FuncTest |
+-----+
finish in 0.000087 seconds.
```

图 4-4 展示表

(3) 用例 103

在使用创建索引之前,已经为该表初始化插入了 1000 条数据。为其 columnOne 创建索引,如图 4-5 所示

```
ya-db x
[BPT::insert INFO] 情况 (2) 叶子节点已满, 分裂叶子节点
[BPT::insert INFO] 情况 (2.2) (2.3) 抽象出递归函数处理
[BPT::InsertInternalNode INFO] 情况 (2.2) 父节点未滿, 直接插入
[BPT::locateLeafNode INFO] 定位叶子节点成功
[BPT::insert INFO] 情况 (1) 叶子节点未滿, 直接插入
[BPT::locateLeafNode INFO] 定位叶子节点成功
[BPT::insert INFO] 情况 (1) 叶子节点未滿, 直接插入
[BPT::locateLeafNode INFO] 定位叶子节点成功
[BPT::insert INFO] 情况 (1) 叶子节点未滿, 直接插入
[BPT::locateLeafNode INFO] 定位叶子节点成功
[BPT::insert INFO] 情况 (2) 叶子节点已满, 分裂叶子节点
[BPT::insert INFO] 情况 (2.2) (2.3) 抽象出递归函数处理
[BPT::InsertInternalNode INFO] 情况 (2.2) 父节点未滿, 直接插入
[BPT::locateLeafNode INFO] 定位叶子节点成功
[BPT::insert INFO] 情况 (1) 叶子节点未滿, 直接插入
[Table::createIndex INFO] column 'columnOne' create index success.
finish in 0.104343 seconds.
```

图 4-5 创建索引

(4) 用例 104

向 FuncTest 插入一条数据, 将同步更新索引, 即如果如图 4-6 所示。


```
wangleqing > INSERT INTO FuncTest 1001 99999999 1001 10001
INSERT INTO FuncTest 1001 99999999 1001 10001
[BPT::locateLeafNode INFO] 定位叶子节点成功
[BPT::insert INFO] 情况 (2) 叶子节点已满, 分裂叶子节点
[BPT::insert INFO] 情况 (2.2) (2.3) 抽象出递归函数处理
[BPT::InsertInternalNode INFO] 情况 (2.2) 父节点未满, 直接插入
[Table::insertRecord INFO] update column 「columnOne」 index.
[BPT::locateLeafNode INFO] 定位叶子节点成功
[BPT::insert INFO] 情况 (2) 叶子节点已满, 分裂叶子节点
[BPT::insert INFO] 情况 (2.2) (2.3) 抽象出递归函数处理
[BPT::InsertInternalNode INFO] 情况 (2.2) 父节点未满, 直接插入
finish in 0.000438 seconds.
```

图 4-6 插入数据

如图所示, 成功插入数据, 并且更新了列 columnOne 的索引。

(5) 用例 105

```
wangleqing > SELECT * FROM FuncTest WHERE id = 1001
SELECT * FROM FuncTest WHERE id = 1001
[Table::selectRecord INFO] 主键索引
+-----+-----+-----+-----+
| id      | columnOne | columnTwo | columnThree |
+-----+-----+-----+-----+
| 1001    | 99999999 | 1001      | 10001       |
+-----+-----+-----+-----+
finish in 0.000191 seconds.
```

图 4-7 精确主键索引搜索

(6) 用例 106

```
wangleqing > SELECT * FROM FuncTest WHERE columnOne = 99999999
SELECT * FROM FuncTest WHERE columnOne = 99999999
[Table::selectRecord INFO] 非主键索引
+-----+-----+-----+-----+
| id      | columnOne | columnTwo | columnThree |
+-----+-----+-----+-----+
| 1001    | 99999999 | 1001      | 10001       |
+-----+-----+-----+-----+
finish in 0.000170 seconds.
```

图 4-8 精确非主键索引搜索

(7) 用例 107

```
wangleqing > SELECT * FROM FuncTest WHERE columnTwo = 1001
SELECT * FROM FuncTest WHERE columnTwo = 1001
[Table::selectRecord INFO] 无索引
+-----+-----+-----+-----+
| id      | columnOne | columnTwo | columnThree |
+-----+-----+-----+-----+
| 1001    | 99999999 | 1001      | 10001       |
+-----+-----+-----+-----+
finish in 0.003540 seconds.
```

图 4-9 精确无索引搜索

(8) 用例 108

```
wangleqing > SELECT * FROM FuncTest WHERE id in 5 10
SELECT * FROM FuncTest WHERE id in 5 10
[Table::selectRecord INFO] 主键索引
+-----+-----+-----+-----+
| id      | columnOne | columnTwo | columnThree |
+-----+-----+-----+-----+
| 5       | 672513670 | 5         | 5           |
| 6       | 1706014008 | 6         | 6           |
| 7       | 317570514  | 7         | 7           |
| 8       | 370148691  | 8         | 8           |
| 9       | 851095448  | 9         | 9           |
| 10      | 738786720  | 10        | 10          |
+-----+-----+-----+-----+
finish in 0.003612 seconds.
```

图 4-10 范围主键索引搜索

(9) 用例 109

```
wangleqing > SELECT * FROM FuncTest WHERE columnOne in 3000000 10000000
SELECT * FROM FuncTest WHERE columnOne in 3000000 10000000
[Table::serialize INFO] deSerialize Table 「FuncTest」
[Table::useTable INFO] use Table 「FuncTest」 success.
[Table::selectRecord INFO] 非主键索引
+-----+-----+-----+-----+
| id      | columnOne | columnTwo | columnThree |
+-----+-----+-----+-----+
| 624     | 3104546   | 624       | 624         |
| 758     | 3115111   | 758       | 758         |
| 611     | 9840775   | 611       | 611         |
+-----+-----+-----+-----+
finish in 0.003788 seconds.
```

图 4-11 范围非主键索引搜索

(10) 用例 110

```
wangleqing > SELECT * FROM FuncTest WHERE columnTwo in 5 10
SELECT * FROM FuncTest WHERE columnTwo in 5 10
[Table::selectRecord INFO] 无索引
+-----+-----+-----+-----+
| id      | columnOne | columnTwo | columnThree |
+-----+-----+-----+-----+
| 5        | 672513670 | 5         | 5           |
| 6        | 1706014008 | 6         | 6           |
| 7        | 317570514 | 7         | 7           |
| 8        | 370148691 | 8         | 8           |
| 9        | 851095448 | 9         | 9           |
| 10       | 738786720 | 10        | 10          |
+-----+-----+-----+-----+
finish in 0.003665 seconds.
```

图 4-12 范围非无索引搜索

（二）负载测试

功能测试已经完成，本次负载测试主要在 100 列和 100w 条记录的情况下，对执行速度进行测试。

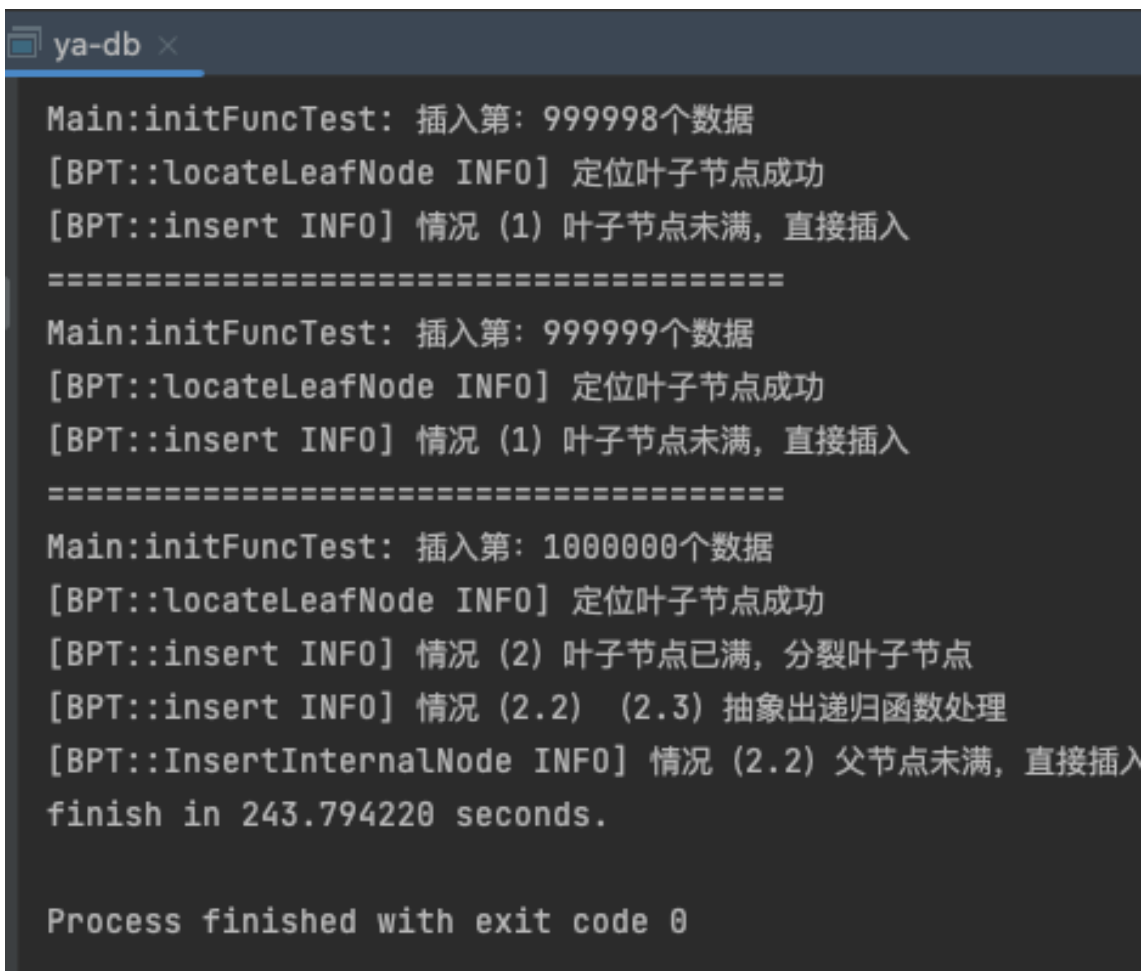
表 4-2 负载测试用例

编号	内容描述	过程描述	预期结果	测试结果
201	插入 100w 条数据	向拥有 100 列的表 LoadTest 插入 100w 条数据，测试时间	速度良好	与预期结果一致
202	创建索引	为 LoadTest 的 columnOne 创建索引	速度良好	与预期结果一致
203	插入数据	插入数据，同时将自动更新索引	速度良好	与预期结果一致
204	精确主键索引搜索	使用主键索引进行精准搜索	速度良好	与预期结果一致
205	精确非主键索引搜索	使用非主键索引进行精准搜索	速度良好	与预期结果一致

206	精确无索引搜索	不使用索引进行精准搜索	速度良好	与预期结果一致
207	范围主键索引搜索	使用主键索引进行范围搜索	速度良好	与预期结果一致
208	范围非主键索引搜索	使用非主键索引进行范围搜索	速度良好	与预期结果一致
209	范围无索引搜索	不使用索引进行范围搜索	速度良好	与预期结果一致

(1) 用例 201

当插入 100w 条 100 列的数据时，共耗时 230.56 秒。



```

ya-db x
Main:initFuncTest: 插入第: 999998个数据
[BPT::locateLeafNode INFO] 定位叶子节点成功
[BPT::insert INFO] 情况 (1) 叶子节点未滿, 直接插入
=====
Main:initFuncTest: 插入第: 999999个数据
[BPT::locateLeafNode INFO] 定位叶子节点成功
[BPT::insert INFO] 情况 (1) 叶子节点未滿, 直接插入
=====
Main:initFuncTest: 插入第: 1000000个数据
[BPT::locateLeafNode INFO] 定位叶子节点成功
[BPT::insert INFO] 情况 (2) 叶子节点已滿, 分裂叶子节点
[BPT::insert INFO] 情况 (2.2) (2.3) 抽象出递归函数处理
[BPT::InsertInternalNode INFO] 情况 (2.2) 父节点未滿, 直接插入
finish in 243.794220 seconds.

Process finished with exit code 0

```

图 4-13 插入 100w 条数据

(2) 用例 202

对已经插入 100w 条 100 列的数据表，创建索引共需要 338.94 秒

```
[Table::createIndex INFO] column 「One_a」 create index success.  
finish in 338.949365 seconds.
```

图 4-14 创建索引

最终创建完 100w 条数据和索引的存储文件大小为 1658899624 Bytes，约 1.545G，如图 4-15 所示：

```
) ls -la  
total 1620160  
drwx----- 2 ubuntu ubuntu      4096 Nov 30 17:27 .  
drwxr-xr-x 6 ubuntu ubuntu      4096 Nov 29 17:18 ..  
-rw-rw-r-- 1 ubuntu ubuntu    130192 Nov 30 17:27 FuncTest.table  
-rw-rw-r-- 1 ubuntu ubuntu 1658899624 Nov 30 17:24 LoadTest.table
```

图 4-15 文件大小

（3）用例 203

当插入数据时，更新索引速度十分快。

```
wangleqing > INSERT INTO LoadTest 1000001 99999999 99999999 99999999  
INSERT INTO LoadTest 1000001 99999999 99999999 99999999  
[BPT::locateLeafNode INFO] 定位叶子节点成功  
[BPT::insert INFO] 情况 (1) 叶子节点未滿，直接插入  
[Table::insertRecord INFO] update column 「One_a」 index.  
[BPT::locateLeafNode INFO] 定位叶子节点成功  
[BPT::insert INFO] 情况 (1) 叶子节点未滿，直接插入  
finish in 0.000491 seconds.
```

图 4-16 插入数据

（4）用例 204

在 100w 条数据中使用主键索引搜索精确值仅花费 0.48 秒。

```
wangleqing > SELECT * FROM LoadTest WHERE id = 900000
SELECT * FROM LoadTest WHERE id = 900000
[Table::selectRecord INFO] 主键索引
+-----+-----+-----+-----+-----+
| id      | One_a    | two_a    | three_a   | four_a    |
+-----+-----+-----+-----+-----+
| 900000  | 338504198 | 661388820 | 2038615949 | 1577769984 |
+-----+-----+-----+-----+-----+
finish in 0.488009 seconds.
```

图 4-17 精确主键索引搜索

(5) 用例 205

在 100w 条数据中使用非主键索引搜索精确值仅花费 0.49 秒。

```
wangleqing > SELECT * FROM LoadTest WHERE One_a = 338504198
SELECT * FROM LoadTest WHERE One_a = 338504198
[Table::selectRecord INFO] 非主键索引
+-----+-----+-----+-----+-----+-----+
| id      | One_a    | two_a    | three_a   | four_a    |
+-----+-----+-----+-----+-----+-----+
| 900000  | 338504198 | 661388820 | 2038615949 | 1577769984 |
+-----+-----+-----+-----+-----+-----+
finish in 0.489296 seconds.
```

图 4-18 精确非主键索引搜索

(6) 用例 206

在 100w 条数据中无索引时搜索精确值平均需要花费 5.5 秒。

```
wangleqing > SELECT * FROM LoadTest WHERE two_a = 661388820
SELECT * FROM LoadTest WHERE two_a = 661388820
[Table::selectRecord INFO] 无索引
+-----+-----+-----+-----+-----+-----+
| id      | One_a    | two_a    | three_a   | four_a    |
+-----+-----+-----+-----+-----+-----+
| 900000  | 338504198 | 661388820 | 2038615949 | 1577769984 |
+-----+-----+-----+-----+-----+-----+
finish in 5.575569 seconds.
```

图 4-19 精确无索引搜索

(7) 用例 207

主键范围搜索也仅需要 0.49 秒

```
wangleqing > SELECT * FROM LoadTest WHERE id in 900000 900005
SELECT * FROM LoadTest WHERE id in 900000 900005
[Table::selectRecord INFO] 主键索引
+-----+-----+-----+-----+-----+-----+
| id      | One_a    | two_a    | three_a   | four_a    | five_a    | six_a    |
+-----+-----+-----+-----+-----+-----+
| 900000   | 338504198 | 661388820 | 2038615949 | 1577769984 | 1052993490 | 63      |
| 900001   | 643741960 | 1008770439 | 244811379  | 2006885136 | 995712702  | 45      |
| 900002   | 1023895719 | 742750092 | 910922210  | 1045658693 | 333760277  | 14      |
| 900003   | 2146972262 | 1186725285 | 228899015  | 1923789905 | 2134423794 | 33      |
| 900004   | 1799951842 | 1140474570 | 1910161782 | 1249685681 | 1818478955 | 19      |
| 900005   | 1077552441 | 477902550  | 1308964954 | 1741034787 | 795066997  | 12      |
+-----+-----+-----+-----+-----+-----+
finish in 0.495335 seconds.
```

图 4-20 范围主键索引搜索

(8) 用例 208

在范围小的情况下，非主键索引搜索速度也很快，但如果是大范围的情况下，非主键索引由于需要对 id 进行二次索引，因此速度将会逐渐趋于无索引范围搜索。

```
wangleqing > SELECT * FROM LoadTest WHERE One_a in 338504198 338504200
SELECT * FROM LoadTest WHERE One_a in 338504198 338504200
[Table::serialize INFO] deSerialize Table 「LoadTest」
[Table::useTable INFO] use Table 「LoadTest」 success.
[Table::selectRecord INFO] 非主键索引
[Table::selectRecord INFO] id900000
+-----+-----+-----+-----+-----+
| id      | One_a    | two_a    | three_a   | four_a    |
+-----+-----+-----+-----+-----+
| 900000   | 338504198 | 661388820 | 2038615949 | 1577769984 |
+-----+-----+-----+-----+-----+
finish in 0.489451 seconds.
```

图 4-21 范围非主键索引搜索

(9) 用例 208

在搜索范围较小的情况下，无索引范围搜索速度仅是较慢，但如果是大范围搜

索，速度将很不理想。

```
wangleqing > SELECT * FROM LoadTest WHERE two_a in 661388820 661389000
SELECT * FROM LoadTest WHERE two_a in 661388820 661389000
[Table::serialize INFO] deSerialize Table 「LoadTest」
[Table::useTable INFO] use Table 「LoadTest」 success.
[Table::selectRecord INFO] 无索引
+-----+-----+-----+-----+-----+-----+
| id      | One_a    | two_a    | three_a  | four_a   | five_a   |
+-----+-----+-----+-----+-----+-----+
| 900000  | 338504198 | 661388820 | 2038615949 | 1577769984 | 1052993490 |
+-----+-----+-----+-----+-----+-----+
finish in 4.896376 seconds.
```

图 4-22 范围无索引搜索

第五章 开发计划

本课题由作者独自设计并开发完成，在本课题的推进和实施过程中，针对复杂工程问题的执行指定了详尽的工程计划，具体内容如表 5-1 所示：

表 5-1 工程计划管控与执行计划

工程计划	详细内容	执行情况
项目需求分析	针对实际项目背景和使用场景，进行详细需求分析	按期完成
可行性研究和分析	针对需求分析和复杂工程问题的归纳，从技术、经济成本和社会因素等方面进行可行性分析	按期完成
技术知识学习	针对理论研究和项目开发所需的知识技能，进行针对性的学习，如 Linux 文件系统 API、B+树等。	按期完成
竞品调研	调研市场上成熟的数据库系统，并学习其架构原理	按期完成
系统架构设计及实现	对 YA-DB 项目框架设计，并使用代码实现详细设计中的工程项目，对复杂工程问题进行攻关	按期完成
系统单元测试	对实现的系统的各个功能进行白盒单元测试，确保功能单元的正常运行	按期完成
系统功能测试	对系统整体进行黑盒功能测试，确保各个功能单元的联动功能正常运行	按期完成