

**院 系：计 算 机 学 院**

**实验课程：数据库系统原理**

**实验项目：位图索引**

**指导老师：？？？？？？？？？**

**专 业：计算机科学与技术**

**班 级：？？？？？？？？？**

**学 生：？？？？？？？？？**

**华南师范大学教务处**

**目录**

[1 实验内容 4](#_Toc90503733)

[1.1 实验题目：实现位图索引 4](#_Toc90503734)

[1.2 实验要求 4](#_Toc90503735)

[2 设计思路 5](#_Toc90503736)

[2.1 记录 5](#_Toc90503737)

[2.2 系统架构 5](#_Toc90503738)

[2.3 缓冲池管理器设计 5](#_Toc90503739)

[2.4 位图索引管理器设计 6](#_Toc90503740)

[2.5 位图索引管理器设计 6](#_Toc90503741)

[2.6 位图索引设计 6](#_Toc90503742)

[2.7 位图方法设计 7](#_Toc90503743)

[3 具体实现 8](#_Toc90503744)

[3.1 实现工具 8](#_Toc90503745)

[3.2 SQL解析实现 8](#_Toc90503746)

[3.2.1 SQL语言定义 8](#_Toc90503747)

[3.2.2 SQL词法分析 9](#_Toc90503748)

[3.2.3 SQL文法分析 10](#_Toc90503749)

[3.3 缓冲池管理器实现 11](#_Toc90503750)

[3.3.1 结构图 11](#_Toc90503751)

[3.3.2 数据成员： 11](#_Toc90503752)

[3.3.3 各方法的实现 12](#_Toc90503753)

[3.4 位图索引管理器实现 13](#_Toc90503754)

[3.4.1 数据成员： 13](#_Toc90503755)

[3.4.2 各方法的实现： 13](#_Toc90503756)

[3.5 位图索引实现 15](#_Toc90503757)

[3.5.1 数据成员： 15](#_Toc90503758)

[3.5.2 各方法实现： 15](#_Toc90503759)

[3.6 位图实现： 16](#_Toc90503760)

[3.6.1 数据成员： 16](#_Toc90503761)

[3.6.2 各方法实现 17](#_Toc90503762)

[3.7 文件结构 19](#_Toc90503763)

[3.7.1 数据文件: 19](#_Toc90503764)

[3.7.2 位图索引文件： 19](#_Toc90503765)

[4 实验结果 20](#_Toc90503766)

[4.1 测试与分析工具 20](#_Toc90503767)

[4.2 测试 20](#_Toc90503768)

[4.2.1 单元测试样例: 20](#_Toc90503769)

[4.2.2 大规模集成测试样例 20](#_Toc90503770)

[4.2.3 测试结果 21](#_Toc90503771)

[4.3 性能评测： 21](#_Toc90503772)

[4.3.1 评测样例 21](#_Toc90503773)

[4.3.2 性能评测结果： 22](#_Toc90503774)

[4.4 SQL界面测试测试： 22](#_Toc90503775)

# 实验内容

## 实验题目：实现位图索引

## 实验要求

（1）说明设计思想及实现方案

（2）使用程序实现位图索引，语言工具不限

（3）说明指定题目的详细方案（包括数据结构、文件结构、实现细节等）

（4）实验数据要求有小数据量的模拟及大数据量的验证

（5）实验的效率分析（代价及具体时间）

（6）提交实验报告（按照教务处提供的实验报告格式）

# 设计思路

## 记录

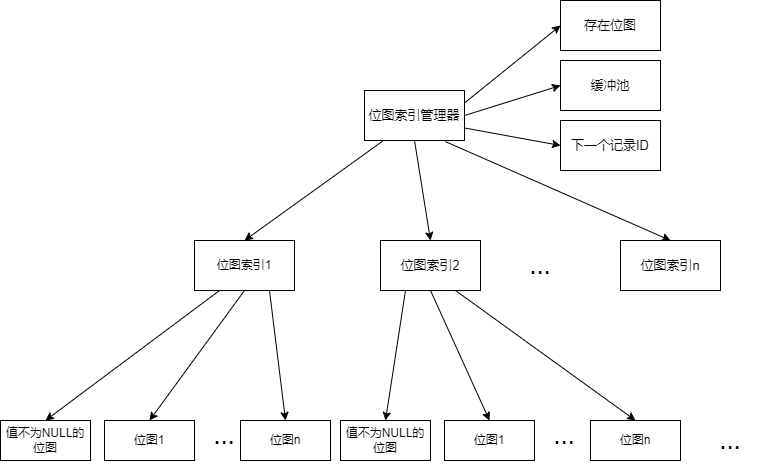
每一个记录有一个记录号（记录号是连续递增的整数，使得记录号连续，记录的存储连续)，每一条记录里面有多个属性。

## 系统架构

每一张表有一个位图索引管理器。它里面管理着很多个位图索引，同时也管理着一个存在位图，管理着一个缓冲池。

表里的每一个属性对应一个位图索引。一个位图索引管理着很多个位图，一个属性的每个值对应一个位图，同时也管理着一个值不为NULL的位图。

**具体的架构图如下：**



## 缓冲池管理器设计

缓冲池管理器提供如下方法：

1. 从缓冲池中取指定的文件页。
2. 将指定的页放回（unpin）缓冲池。
3. 向文件中附加（append）新页，并将新页取回到缓冲池同时取出。
4. 将指定的文件页刷盘。

## 位图索引管理器设计

位图索引管理器提供如下方法：

1. 统计满足条件的记录的条数。
2. 根据条件删除记录。
3. 插入一条新的记录。
4. 根据条件更新记录。
5. 查询所有满足的记录。

## 位图索引管理器设计

位图索引管理器提供如下方法：

1. 统计满足条件的记录的条数。
2. 根据条件删除记录。
3. 插入一条新的记录。
4. 根据条件更新记录。
5. 查询所有满足的记录。

## 位图索引设计

位图索引提供如下方法：

1. 更新大小，用于更新所有其管理位图的大小
2. 设置某个值的位图的对应位为1
3. 将其管理的所有位图中的某个位置的位设置为0
4. 通过某个条件(a = 3)查询位图
5. 返回其管理的所有值位图

## 位图方法设计

位图提供如下方法：

1. 更新大小，用于更新本位图的大小
2. 将某个位置上的位设置为1
3. 将某个位置上的位设置为0
4. 统计本位图中1的个数
5. 使用popcnt指令集统计本位图中1的个数
6. 序列化方法，用于将位图序列化为一个01字符串，同时采用位图压缩算法进行压缩
7. 反序列化方法，用于将一个位图压缩后的01字符串还原为未压缩之前的01字符串。
8. 获得位图的长度
9. 初始化位图帮助数组
10. 迭代器接口，用于使用迭代器遍历位图
11. 所有的位运算操作。

另外，在位图取反操作实现时需要注意，**不能直接对每一个位按位取反**，原因如下：1.因为原先有一些位为0是因为这些位不存在，按位取反后变为1后这些就存在了，错误；2. 同样的问题发生在原先有一些位为0是因为这些位对应记录的值为NULL的情况上。

**解决方案：**按位取反后所得结果的bitmap还要与“值不为NULL的bitmap”和“existence bitmap”两个bitmap相与。

# 具体实现

## 实现工具

实现语言：C++，语言标准ISO C++20

编译器：MinGW 8.1.0，内置编译器g++ 8.1.0

编译链工具：CMAKE 3.21

词法分析工具：Flex

## SQL解析实现

### SQL语言定义

由于需要用户输入查询条件，所以需要定义一套SQL语言。这里定义的SQL语言形如：

*select attr1=value1 and attr2<=value2 and attr3 IS NULL or attr4 IS NOT NULL…*

*insert attr1=value1 attr2=value2 attr3=value3 attr4=value4 …*

*update attr1=value1 attr2=value2 … where attr2<=value2 …*

*delete attr1=value1 and attr2=value2*

*count attr1=value1 and attr2=value2*

接着需要处理简单SQL语句到内部类型的转换。

表示蓝色与红色部分的条件的内部结构是std::vector<std::variant<Token, std::tuple<std::string, Token, std::string>>>。红色的部分对应Token，蓝色的部分对应std::tuple<std::string, Token, std::string>，其中左边的std::string保存的是attr，中间的Token保存的是<=、IS NULL等条件，右边的std::string保存的是value（如果是IS NULL或者IS NOT NULL则右边的std::string为空）。

表示橙色部分的属性的内部结构是std::vector<std::pair<std::string, std::string >>，左边的std::string是属性名attr，右边的std::string是值value。

### SQL词法分析

对于词法分析的部分使用flex词法分析程序生成器进行生成，**flex的定义程序如下**：





flex词法分析程序

### SQL文法分析

完成后写出所需SQL语句的文法：

*SQL -> select A | update D where A | count A | delete A | insert D*

*A -> A { or B } | B*

*B -> B { and C } | C*

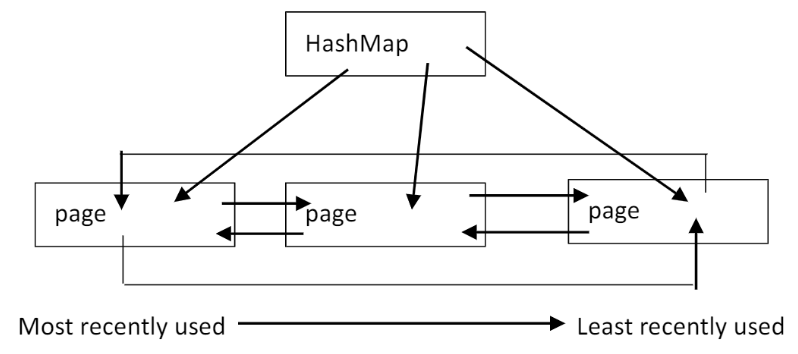
*C -> (A) | attr [6种关系运算符] value | value [6种关系运算符] attr | attr IS NULL | attr IS NOT NULL*

*D -> { attr = value | value = attr }*

按照递归下降分析法写出分析这个SQL的程序，并在递归下降的过程中将SQL语句转换为系统识别的内部类型。接着就可以根据解析内部类型了。

## 缓冲池管理器实现

### 结构图



缓冲池管理器结构图

### 数据成员：

|  |  |  |
| --- | --- | --- |
| 变量名 | 类型 | 解释 |
| m\_pages | Page \* | 缓冲池块(Frame)构成的数组 |
| m\_fileStore | FileStore \* | 用于以页为单位读写磁盘 |
| m\_pageTable | std::map<std::pair<FileType, PageIDType>, FrameIDType> | 维护磁盘页ID到装有该磁盘页的缓冲池块Frame ID之间的映射 |
| m\_replacer | Replacer \* | 用于基于LRU策略找出要从内存中淘汰的页 |
| m\_freeList | std::list<FrameIDType> | 空闲的Frame的列表 |

### 各方法的实现

1. 构造方法：

根据传入的缓冲池大小参数创建对应大小的缓冲池进行管理。

1. 析构方法：

回收缓冲池所占有的内存空间。

1. fetchPage（从缓冲池中拿取指定的文件页）：

通过m\_pageTable查询指定的文件页是否已经缓存在内存中。如果没有，查看是否空闲的Frame，如果存在将文件页从磁盘中读取到该Frame中，将该Frame的pinCount加一，并将其返回；如果没有，那么通过m\_lruReplacer基于LRU淘汰策略选择出要淘汰的页（要淘汰的页pinCount需为0），将其淘汰，并将文件页从磁盘中读取到其原先占有的Frame中，将该Frame的pinCount加一，并返回。在淘汰页时，如果页中的Dirty位为1，会将页写盘。

1. flushPage（将指定的文件页刷盘）：

如果指定的文件在内存中，且其Dirty位为1则将其刷盘。

1. unpinPage（将指定的文件页放回缓冲池）：

将指定的文件页pinCount减一，如果持有页时对页的内容有改写，那么还会将页的Dirty位置为1。

1. appendNewPage（向文件中附加（append）新页，并将新页取回到缓冲池同时取出）：

向文件末尾写入一个新页，并将该新页放入到缓冲池中（过程与fetchPage方法相似），将放有该页的Frame的pinCount加一，并将该新页返回。

## 位图索引管理器实现

### 数据成员：

|  |  |  |
| --- | --- | --- |
| 变量名 | 类型 | 解释 |
| m\_tableName | std::string | 表名 |
| m\_nextRecordID | RecordIDType | 下一个要分配的记录ID |
| m\_existenceBitmap | Bitmap | 存在位图，维护着当前哪些记录号是使用中的 |
| m\_bitmapIndices | std::map<std::string, BitmapIndex> | 维护属性名到位图索引的映射 |
| m\_bufferPoolManager | BufferPoolManager & | 缓冲池管理器 |

### 各方法的实现：

1）构造函数：

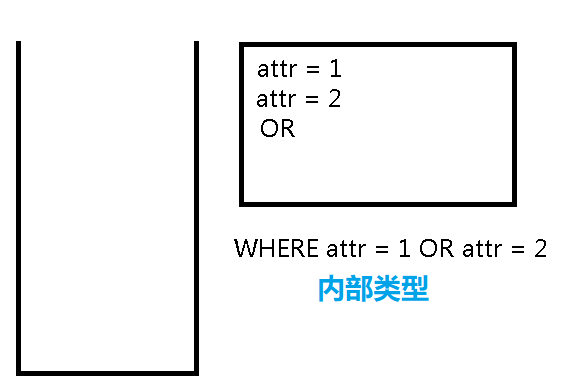
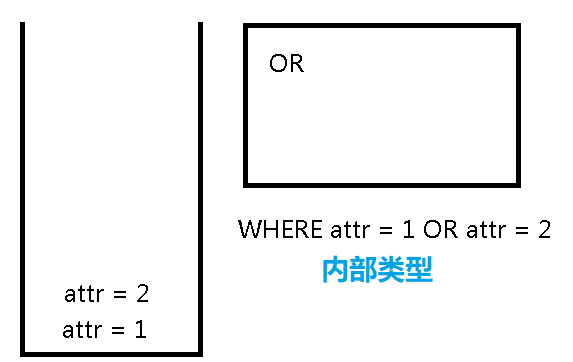
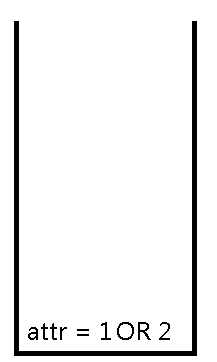
从位图索引文件中读取保存好的位图索引管理器，并重建位图索引管理器，在重建的过程中需要相将01字符串使用位图解压算法反序列化后再设置到对应的位图中。

2）析构函数：

将位图索引管理器中的所有位图保存到文件中，同时在保存前需要对位图进行压缩存储。在存储时要保存一些元信息，比如当前位图索引管理器一共管理了几个位图索引，每个位图索引有几个位图等。

3）SQL内部结构到位图的转换函数：

利用栈来完成具体的操作。遍历前面词法分析和语法分析处理好的内部结构，如果碰到形如a=1的形式就调用位图索引的getBitmap方法获取对应的位图并压栈，如果是操作符就弹出栈中的操作数进行对应的操作。And就使用位图的与操作，Or就使用位图的或操作。最后返回栈顶元素与存在位图的与。

4）统计满足条件的记录的条数:

首先调用函数将条件转换为对应的位图，然后调用位图的popCount函数进行位的统计并返回。

5）根据条件删除记录：

首先调用函数将条件转换为对应的位图，遍历这个取得的位图上的位，调用所有位图索引的移除位方法移除这个位，最后再将存在位图中这个位设置为0并返回条件位图中1的个数代表影响的行数。

6）插入一条新的记录：

首先在存在位图中寻找0，如果找到了0就代表有可以用的空位，直接调用插入函数插入。如果没有找到就增加nextRecordID的值，然后调用所有位图索引的更新大小方法在所有的位图索引后面添加一个位并调用插入函数插入记录。

插入函数首先会将存在位图对应的位设置为1，然后在缓冲池中根据记录号获取对应的页，并将记录写入该页中，对于新记录的每一个属性，找到对应的位图索引，如果是插入已存在的值（找到了对应值的位图）就将对应值的位图的对应位设置为1，如果插入新值（找不对对应值的位图），就在这个位图索引中创建一个新的位图，将对应位设置为1，其余位为0。

7）根据条件更新记录:

首先调用函数将条件转换为对应的位图，遍历这个取得的位图上的位，清除所有的位图索引的所有位图上的对应位。然后开始设置新的属性，先尝试寻找对应的位图，对于更新已存在的值（找到了对应值的位图）的情况，将对应值的位图的对应位设置为1，对于更新新值（找不对对应值的位图）的情况，在这个位图索引中创建一个新的位图，将对应位设置为1，其余位为0。

同时，更新记录时根据记录号从缓冲池中取出对应的页，将更新后的记录写入到对应的页中。

8）查询所有满足的记录：

首先调用函数将条件转换为对应的位图，然后利用这个位图构造一个记录迭代器，返回这个记录迭代器。每次对迭代器进行解引用操作时就从缓冲区中取出对应的记录并返回，执行++操作时就寻找位图中的下一个1，为下一次解引用做准备。

## 位图索引实现

### 数据成员：

|  |  |  |
| --- | --- | --- |
| 变量名 | 类型 | 解释 |
| m\_bitmapLength | uint64\_t & | 位图长度，引用位图索引管理器中nextRecordID |
| m\_bitmaps | std::map<ValueType, Bitmap> | 属性值到位图的映射 |
| m\_notNullBitmap | Bitmap | 记录非NULL记录的位图，记录着哪些记录的值不为NULL |

### 各方法实现：

1）更新大小，用于更新所有其管理位图的大小：

调用所有位图的更新大小方法，同时调用不空位图的更新大小方法。

2）设置某个值的位图的对应位为1

如果这个位图不存在，就创建一个新的位图并插入到当前的位图索引中，然后将对应的位设置为1，如果这个位图已经存在了，就直接将这个位图对应的位设置为1。还需要将不空位图对应位设置为1。

3）将其管理的所有位图中的某个位置的位设置为0

将所有位图对应的位设置为0，如果发现一个位图的所有位均为0，就直接将这个位图从位图索引的map中移除。还需要将不空位图中的对应位设置为0。

4）通过某个条件(a = 3)查询位图：

如果查询条件是IS NULL就返回不空位图的取反；

如果查询条件是IS NOT NULL就返回不空位图；

如果查询条件是=，如果值对应的位图存在就返回对应的位图，如果值对应的位图不存在就返回空位图；

如果查询条件是!=，如果值对应的位图存在就返回对应的位图的取反与不空位图的与，如果值对应的位图不存在就返回所有位图的或；

如果查询条件是>，就使用upper\_bound找到所有比当前值大的位图并返回它们的或；

如果查询条件是>=，就使用lower\_bound找到所有大于等于当前值的位图并返回它们的或；

如果查询条件是<，就使用begin + lower\_bound找到所有小于当前值的位图并返回它们的或；

如果查询条件是<=，就使用begin + upper\_bound找到所有小于等于当前值的位图并返回它们的或；

5）返回其管理的所有值位图：

直接返回成员变量中的map的引用。

## 位图实现：

### 数据成员：

|  |  |  |
| --- | --- | --- |
| 变量名 | 类型 | 解释 |
| m\_bitmap | std::vector<uint64\_t> | 存储位图的动态数组 |
| m\_bitmapLength | uint64\_t & | 位图的长度，引用位图索引管理器中的nextRecordID |
| m\_bitCount | uint64\_t | 当前位图有多少个位为1 |
| ms\_bitmapMask | static uint64\_t[] | 用来执行位操作的帮助数组 |
| ms\_bitmapNotMask | static uint64\_t[] | 用来执行位操作的帮助数组 |

### 各方法实现

刚开始将ms\_bitmapMask[i]设置为1<<i，将ms\_bitmapNotMask[i]设置为~ ms\_bitmapMask[i]。

需要设置一个整数的第i位时，直接位或 ms\_bitmapMask[i]，要清除一个整数的第i位时，直接与ms\_bitmapNotMask[i]。

1）更新大小，用于更新本位图的大小

根据nextRecordID来对更新存储位图vector的大小

2）将某个位置上的位设置为1

如果这个位是0，将bitCount加一，如果这个位是1，就什么都不做。然后把这个位设置为1。

3）将某个位置上的位设置为0

如果这个位是1，将bitCount减一，如果这个位是0，就什么都不做。然后把这个位设置为0。

4）统计本位图中1的个数

直接返回m\_bitCount

5）使用popcnt指令集统计本位图中1的个数

遍历这个整数数组，使用\_\_builtin\_popcountll统计1的个数并求和返回。

6）序列化方法（用于将位图序列化为一个01字符串，同时采用位图压缩算法进行压缩）：

首先将整个vector中的整数逐位提取，变成01字符串，然后对这个01字符串使用位图压缩算法进行压缩，并返回这个压缩好的01字符串。

7）反序列化方法（用于将一个位图压缩后的01字符串还原为未压缩之前的01字符串）：

将一个位图压缩后的01字符串还原为未压缩之前的01字符串，这个函数并不构造vector数组，构造的工作留给位图索引管理器的构造函数来做

8）获得位图的长度：

返回m\_bitmapLength

9）迭代器接口，用于使用迭代器遍历位图：

迭代器刚开始找到第一个1的位置并将其保存在m\_currentPos中，每次解引用直接返回这个保存的值，当执行++操作时，从当前位置开始寻找下一个1的位置并保存到m\_currentPos中，以备下一次解引用的操作。

10）所有的位运算操作：

与等操作：将vector中的所有整数取出来位与另外一个位图对应下标的整数；

或等操作：将vector中的所有整数取出来位与另外一个位图对应下标的整数；

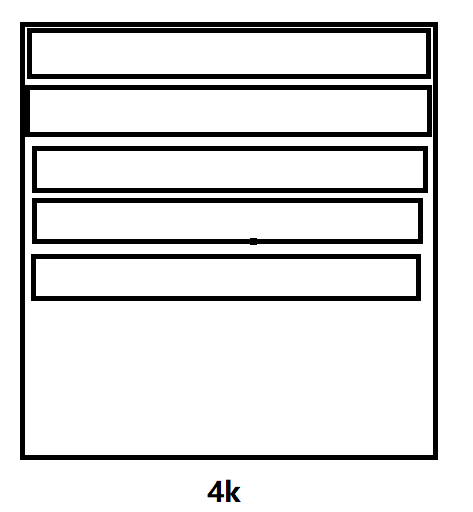
非操作：新建一个空白位图，将vector中的所有整数取反放入新建的空白位图中；

与操作：新建一个空白位图，将两个vector中的所有整数相与后放入新建的空白位图中；

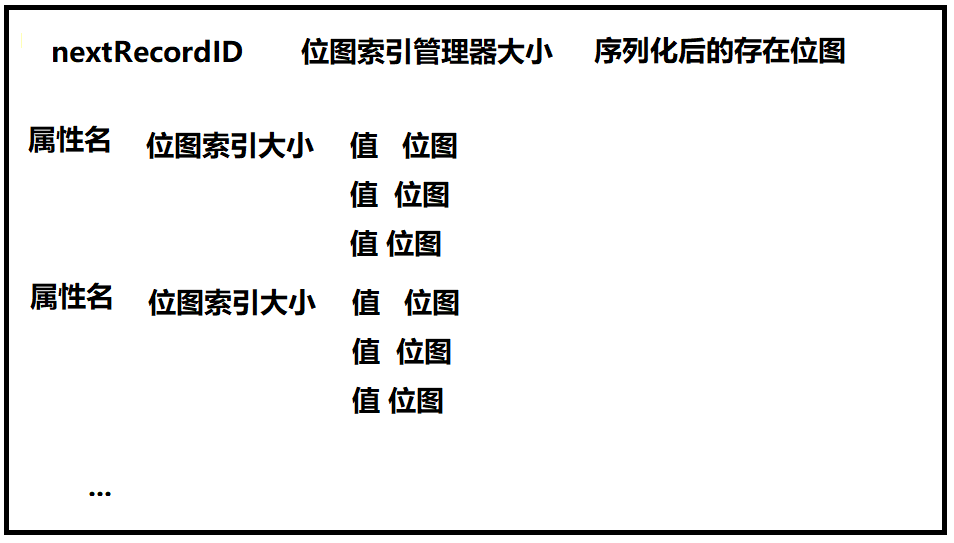
或操作：新建一个空白为徒，将两个vector中的所有整数相或后放入新建的空白位图中；

## 文件结构

### 数据文件:



### 位图索引文件：



# 实验结果

## 测试与分析工具

测试框架：Google Test

效率分析框架：Google Benchmark

## 测试

### 单元测试样例:

InsertTest：测试插入功能

SelectTest：测试查询功能

UpdateTest：测试更新功能

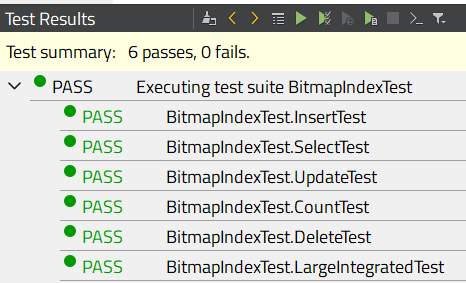
CountTest：测试Count计数功能

DeleteTest：测试删除功能

### 大规模集成测试样例

LargeIntegratedTest：插入10000条随机数据，随机select出来检查数据是否正确，随机更新数据后select出来检查数据是否被更新成功，使用count功能测试插入的条数是否正确，删除所有插入的数据，查看删除的条数是否正确。

### 测试结果



测试结果

## 性能评测：

### 评测样例

Insert: 插入10000条数据到文件中，所有数据的所有字段都被设置了

Select: 在上面的插入执行完成之后，所有的数据都被插入到了数据库里面。在本测试中将用同样的字段值将数据取回来并迭代完成，每次选出1条数据。

SelectLarge: 与上面的测试不同，这个测试在数据上执行10000次选择操作，每次选出67条数据。

Count: 这个测试执行1次count计数操作，统计当前数据库中有多少条数据。

CountLarge: 这个测试执行10000次count计数操作，每次统计67条数据。

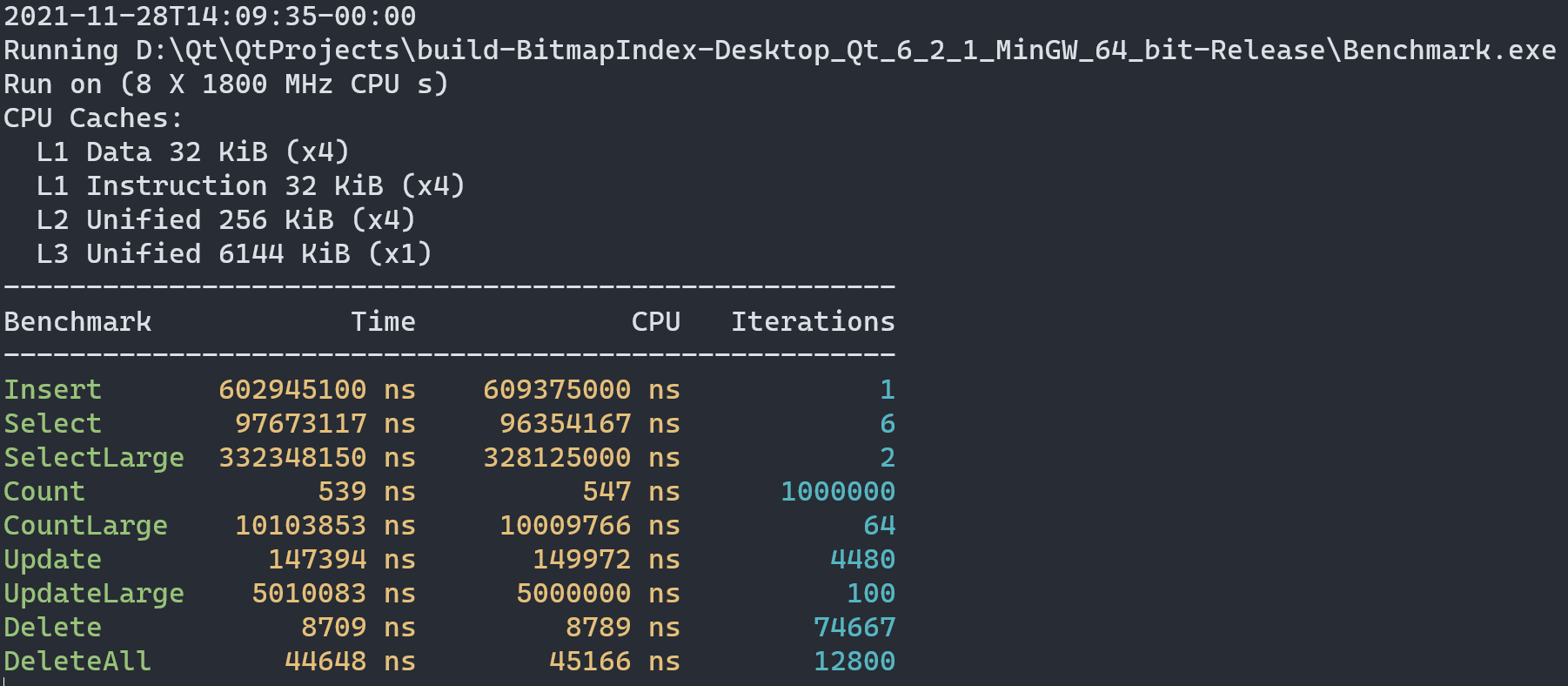
Update: 在数据库上执行150次更新操作，每次更新67条数据，一共更新10000条数据。

UpdateLarge: 在数据库上执行1次更新操作，一共更新10000条数据。

Delete: 删除数据库中的部分数据，一共2500条数据受到影响。

DeleteAll: 删除数据库中所有的记录，一共7500条数据受到影响。

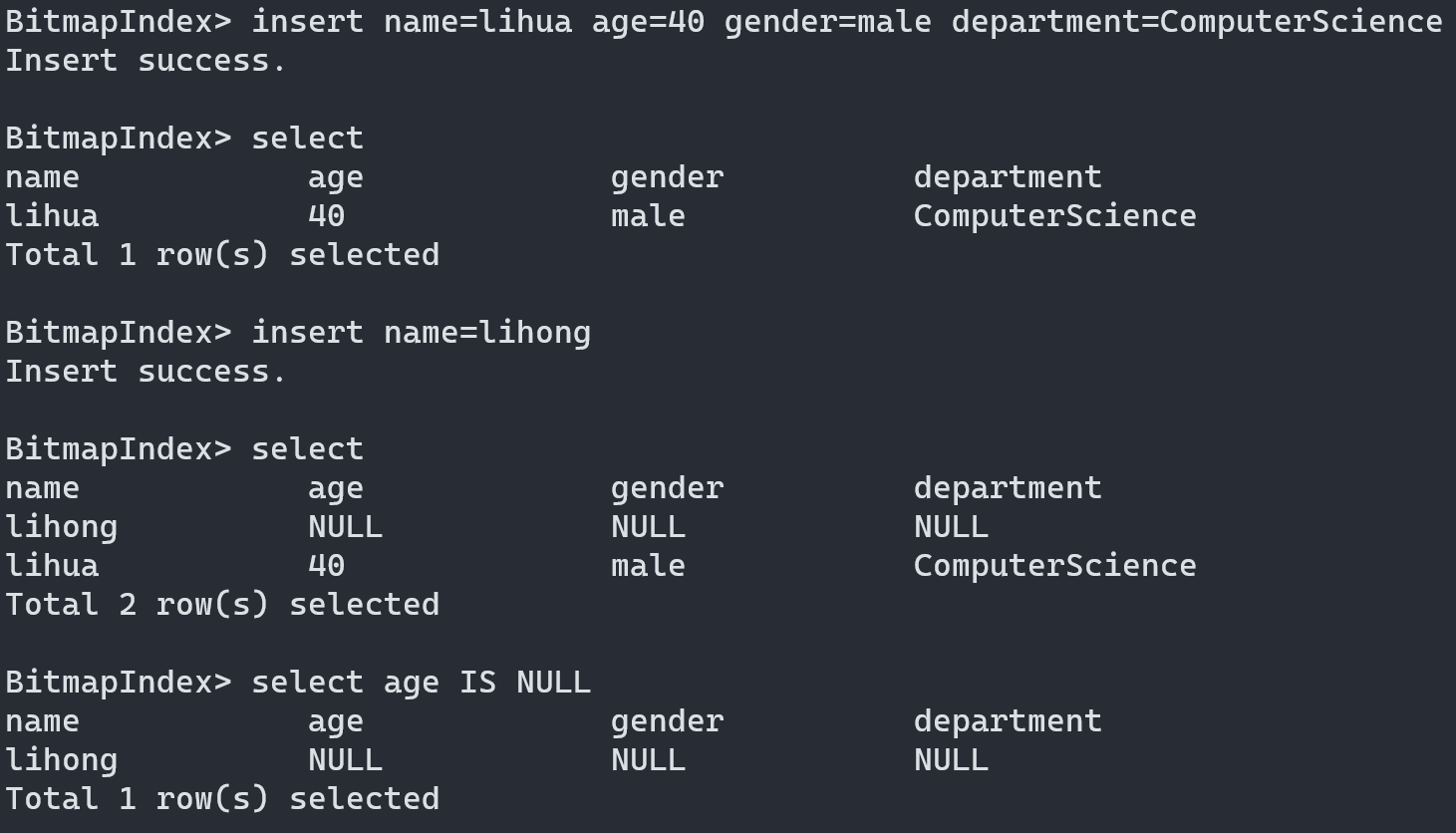
### 性能评测结果：



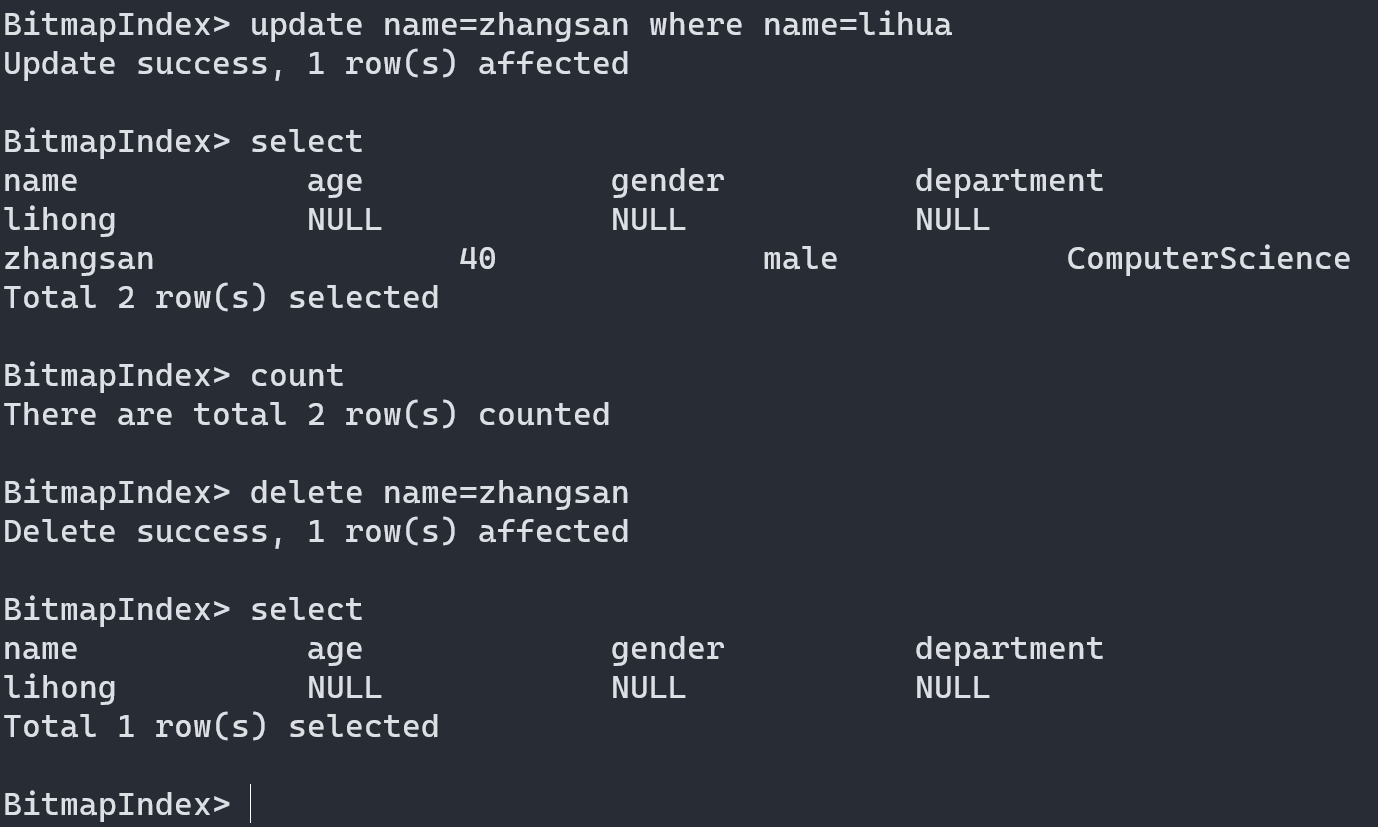
性能评测结果

## SQL界面测试测试：

经测试，SQL界面正常执行：



SQL界面测试1



SQL界面测试2