# 《DBMS》实验报告三

**报告人：**冷友方；王大伟；罗传文；赵丹；杨婧如.

**学院：**信息学院

**日期：**2016年11月30日

## 1 iterator 迭代器的设计

首先，由于对一个表的各种操作，都有获取的元组的操作，为此设计了一个迭代器来方

便的取出表中的元组。

迭代器的结构如下所示：

|  |
| --- |
| struct iterator {  int extent;  int blk;  char b[PAGE\_SIZE];  int t;  int len;  };  struct iterator open\_iter(struct logic\_table \*ltb);  int getnext\_iter(struct iterator \*iter, char \*rec);  int close\_iter(struct iterator \*iter); |

其中extent 用于表示当前的区间，blk表示当前的块，b[PAGE\_SIZE]表示当前取出的块，t用于记录在块中的位置，len表示记录的长度。open\_iter()是打开一个迭代器，并进行初使化；close\_iter()是关闭这个法代器；getnext\_iter()是得到下一条元组。具体的实现在 scantable.c 中。

由于我们对记录的存储是聚簇的，所以我们的迭代器是一次取出一个块，然后一条一条的抛出记录，这个块中的记录用完之后再读下一个块， 直到读出了这个表中的所有的块， 得到了表中的所有记录。

## 2 SPJ结果的输出

在进行完选择和连接操作之后，得到的结果我们想返问给用户。我们实现了 Oracle 里 SQL\*plus 界面里的返回结果的样式，即把结果表用字符界面显示出来。在文件scantable.c中实现，代码如下所示:

|  |
| --- |
| int print\_tuple(char \*rec, struct table\_def \*tdef)  int print\_tbhdr(struct table\_def \*tdef) |

print\_tbhdr()是显示结果表的表头， print\_tuple()是显示结果表里的所有记录。在实现上，因为我们的每一条记录都是以char[]存储在数据文件里面，所以这个函数就是从记录头里面得到记录的模式信息，然后按记录模式逐个解析这条元组，然后按一定的形式显示。

## 3 Select

#### 3.1 无条件选择

无条件选择(我们实现的是表扫描)，就是对表中的所有元组进行一次遍历，实现上只需用表扫描算法即可。由于已经实现了迭代器，所以就是从迭代器中得到每一条记录，把记录放到结果集里面。因为我们的数据库存储，在每一个索引上面都不是聚簇的，所以在进行全表扫描的时候并没有用索引扫描。

首先建表，从磁盘中批量的导入数据，再从表中逐条把数据载入缓冲区里，逐条的写入。具体实现代码在select.c文件的tot\_sel(const char \*tbname)函数中。

#### 3.2 等值条件选择

对于选择条件是等值的情况，当选择条件上面建了索引的时候，可以用索引进行选择操作。即在索引里面查找条件里的键值，得到记录里面满足条件的所有记录的地址，然后从这些地址里面取出所对应的各条记录，因此并不需要扫描整个表。

首先建表，向表批量的导入数据，接着对主键建立索引，最后基于B+树的查找操作来查找索引。具体实现代码在select.c文件的equal \_sel (int key, const char \*tbname)函数中。

#### 3.3 范围条件选择的实现

对于范围查询，由于已经建立了 B+树索引的时候，可以用索引来进行。首先从索引里

面找到范围里面最左边的值的记录，若没有的话，找到比该值大的最小的 key 值的记录。然后在索引树上面向右逐个取出记录对应的地址，从数据里面得到相应的记录，直到索引树上的键值不在这个范围内时停止。

对应这个查找过程，修改了一下 B+树索引里面查找的一点实现，使它能像迭代器一

样一个个地返回当前位置右边的记录地址。具体实现代码在select.c文件的range\_sel(int left, int right, const char \*tbname)函数中。

## 4 Join

#### 4.1 笛卡尔积实现

对于笛卡尔积操作，我们对两个表的任何两条记录分别进行连接，依次取出第一个表的

每一条记录，分别与第二个表中记录做笛卡尔积。两个表做笛卡尔积的时候，对 CPU 时间和 I/O 时间的花费非常大，而且不能够进行优化，所以实际对数据库表进行操作的时候应该避免这方面的操作。

首先建立两个表，将数据批量导入表中，载入其中一个表的一条记录与另一个表的每一条记录按索引建立连接。具体实现代码在join.c文件的product(const char \*tb1, const char \*tb2)函数中。

#### 4.2 基于块的嵌套循环连接实现

每次将尽可能多的第一个表中的元组放在内存，然后对它们建立一个内存查找结构；然后依次取出第二个表中的元组，根据这个元组连接属性上的值查询我们建立的第一个表的内存查找结构，若能找出能和这个元组连接的第一个表中的元组，则把连接结果加入到结果表中。不断重复这样的步骤，直到我们取出了第一个表中的所有元组为止。

首先建立两个表，导入数据，将其中一个表的一条记录和另一个表的每条记录进行对比，如果不相等，就建立连接。具体实现代码在join.c文件的nest\_blk\_join(const char \*tb1, const char \*tb2)函数中。

#### 4.3 索引连接实现

若我们在某一个表上面的连接属性上建立了索引，我们可以进行使用索引的连接，算法

是：对第一个表的每一条元组，使用索引查找第二个表中具有这个元组连接属性上的值的所

有元组，然后分别和这些符合条件的元组进行连接。当第一个表很小时，而且第二个表索引上的重复值不是太多时，这种算法可以达到很高的效率。

首先建立两个表，导入数据，在一个表中建立B+树索引，从左表中载入一条记录到右表查找key所对应的记录。具体实现代码在join.c文件的index\_join(const char \*tb1, const char \*tb2)函数中。

#### 4.4 排序连接实现

当两个表都比较大时，我们可以考虑用排序连接：首先对两个表分别根据连接属性上的

值创建缓冲区大小的排序子表；然后将每个子表的第一块调进缓冲区；重复查找所有的子表

上面最小的连接属性值；分别找出两个表中具有这个最小值的元组，然后进行连接，并删除

这些元组；如果一个子表内存中的块被处理完毕，则装入这个子表的下一块；直到所有的子

表都被处理完毕。

首先建立两个表，将数据导入两个表中，按主键升序聚簇存储，从一个表中载入一条记录，将另一个表作为内循环，直到值大于左表中的值时停止。具体实现代码在sortjoin.c文件中。

#### 4.5 散列连接的实现

同样，当要连接的两个表规模比较大时，可以用散列来把关系进行划分然后分别进行连

接：首先分别将两个关系按照连接属性上的值划分到静态的散列桶中并写回磁盘；然后分别

加载两个关系相对应的桶，对两个桶中的元组进行嵌套循环连接。

首先建立两个表，将数据批量的导入两个表中，在一个表中建立哈希索引，从左表载入一条记录到右表的哈希桶中去查找。具体实现代码在hashjoin.c文件中。

## 5 实验中遇到的问题