**Chapter12 Indexing and Hashing**

1. 区分主索引(primary index)与辅助索引(secondary index)

2. 区分稠密索引(dense index)与稀疏索引(sparse index)

主索引可以是稠密的也可以是稀疏的，辅助索引只能是稠密的

3. 索引的更新（插入和删除），具体实现都要依据索引是稠密的还是稀疏的

4. B+树索引文件：

1）每个节点（不是根节点和叶节点），有⎡*n*/2⎤到n个子节点，也就是有⎡*n*/2-1⎤到n-1个值；

2）叶节点有⎡(*n*–1)/2 ⎤到n个值；

3）如果根节点不是叶节点，它至少有两个子节点，如果是叶节点，有0到n-1个值；

4）在B+树索引文件上做查询，⎡ log⎡*n*/2⎤(*K*)⎤，*log*50(1,000,000) = 4

5）在B+树上进行插入，删除，注意每一步要满足1），2），3）的约束

5. hash索引

* Hash文件组织里每个bucket里存放的已经是文件记录，Hash索引里每个bucket里存放的是搜索码和指针。
* Hash索引通常是辅助索引。
* 区分静态hash和动态hash，理解动态hash的实现方法（对bucket建立了一个bucket address table）

6.静态hash索引与动态hash索引的比较

顺序索引与hash索引的比较

7.SQL语句中index的使用

**create index** <index-name> **on** <relation-name> (<attribute-list>)

**drop index** <index-name>

**chapter13 Query Processing**

**1.基本概念**

Annotated expression specifying detailed evaluation strategy is called an evaluation-plan.

Query Optimization: Amongst all equivalent evaluation plans choose the one with lowest cost.

**2.估算查询的代价**

为了简单，只考虑从磁盘传输和寻道的时间代价：*b \* tT + S \* tS* （忽略了真实情况中存在的CPU时间）

**3. selection operation**

* 使用文件扫描算法（file scan--用于定位、检索满足选择条件的记录的搜索算法）

1）linear search：每个文件块都要被扫描，并且所有记录都要被测试

Cost =*br* block transfers + 1 seek = *tS* + *br*\* *tT*

如果扫描是在键属性上进行的，由于键的不重复性，一旦找到就不再继续扫描，cost= *br* /2 block transfers + 1 seek

2）binary search：适用条件为 文件按某一属性排序，且选择条件是该属性上的等值比较

Cost= ⎡log2(*br)*⎤ \* (*tT* + *tS*)

* 使用索引扫描算法（index scan—选择条件必须基于索引的搜索码）

1）primary index, equality on key（主索引，码属性等值比较）

如果适用B+树，*Cost* = (*hi* + 1) \* (*tT* + *tS*)

2）primary index, equality on nonkey（主索引，非码属性等值比较）

由于是非码属性，所以检索的结果不只一个（设为b个）

*Cost* = *hi \** (*tT* + *tS*)+ *tS* + *tT* \*b

3）secondary index, equality（辅助索引，等值比较）

如果搜索码是候选码，*Cost = (hi* + 1) \* (*tT* + *tS*)

如果搜索码不是候选码，Cost = (*hi* + *n) \** (*tT* + *tS*)

* 涉及比较的选择运算

实现 σ*A*≤*V* (*r*) or σ*A* ≥ *V*(*r*)可以通过线性和二分扫描，也可以通过使用索引

1）primary index, comparision（主索引，比较）(Relation is sorted on A)

对于σ*A* ≥ *V*(*r*)，使用索引找到第一个大于v的元组，然后顺序扫描接下来的元组

对于σ*A*≤*V* (*r*)，只用一直顺序扫描元组，直到遇到大于v的第一个元组，不需要使用索引

2）secondary index, comparision（辅助索引，比较）

对于σ*A* ≥ *V*(*r*)，使用索引找到第一个大于v的元组对应的索引，然后顺序扫描接下来的索引，找到对应的记录

对于σ*A*≤*V* (*r*)，只需要扫描索引对应的记录，直到遇到大于v的元组

* 复杂选择的实现

**Conjunction: σθ1∧ θ2∧. . . θ*n*(*r)***

1）conjunction selection using one index

选择某个θ*i*以及A1-A7的算法之一，使得到的结果对于σθ*i* (*r)*代价最小，将结果放入内存缓冲区后，对剩余的条件做较少的测试即可。

2） conjunction selection using multiple-key index（组合索引—在多个属性上建立的索引）

3）conjunction selection by intersection of identifiers（通过标识符的交集实现合取运算）

**Disjunction: σθ1∨ θ2 ∨. . . θ*n* (*r)***

disjunctive selection by union of identifiers

**4. sorting**

能全部放入内存的关系，可以使用快速排序

不能全部放入内存的关系，可以使用外部归并排序

**External sort-merge**

分为两步：创建归并段，N路归并

代价分析：

第一阶段，关系的每个数据块读入写出，共需2*br*次磁盘访问

第二阶段，共需归并趟数为⎡log*M*–1(*br / M)*⎤，每趟归并，每个数据块读写一次，最后一次可以不写入磁盘

total= 2*br  +br\**(*2* ⎡log*M*–1(*br / M)*⎤ - 1) = *br ( 2* ⎡log*M*–1(*br / M)*⎤ + 1)

寻道代价：

第一阶段，一次寻道用来读每个归并段，一次寻道用来写每个归并段 *2* ⎡*br / M*⎤

第二阶段，对于每趟归并需要*2* ⎡*br / bb*⎤次寻道，最后一次不需要写入磁盘

Total= *2* ⎡*br / M*⎤ + ⎡*br / bb*⎤ (*2* ⎡log*M*–1(*br / M)*⎤ -1)

**5.join operation**

1）nested-loop join

在这里要区分一下连接的外层关系和内层关系

最差情况：*nr* \* *bs* + *br*次块存取，*nr* + *br*次寻道

最好情况：*br*  + *bs*次块存取，2次寻道

2）block nested-loop join

最坏情况： *br* \* *bs + br*次块存取，2 \* *br*  次寻道

最好情况：*br*  + *bs*次块存取，2次寻道

对嵌套循环连接和块嵌套循环连接的改进方案。

3）indexed nested-loop join

Cost： *br* (*tT + tS*) + *nr* \* *c*

4）merge join

Cost：*br + bs* 次块存取，⎡*br / bb*⎤ + ⎡*bs / bb*⎤次寻道，另外还有排序的代价

5）hash join

不需递归划分的情况：3(*br* + *bs)*+ 4 \* *nh* 块存取，2( ⎡*br / bb*⎤ + ⎡*bs / bb*⎤) +2 \* *nh*寻道

需要递归划分的情况：2*(br + bs*)⎡*logM–*1(*bs*) – 1⎤ + *br + bs* 块存取，2(⎡*br / bb*⎤ + ⎡*bs / bb*⎤) ⎡*logM–*1(*bs*) – 1⎤ 次寻道

做课后习题13.3、13.5