**Search (there is conjunction too) and Join Algorithms Examples**  
Suppose that the **EMPLOYEE** file in has

**rE = 10,000 records stored in**

**bE = 2,000 disk blocks with blocking factor**

**bfrE = 5 records/block and the following access paths:**

1. A clustering index on Salary, with levels **xSalary = 3** and average selection cardinality **sSalary = 20**. (This corresponds to a selectivity of slSalary = 20/10000 = 0.002.)
2. A **secondary index** on the key attribute Ssn, with xSsn = 4 (sSsn = 1, slSsn = 0.0001).
3. A **secondary** index on the nonkey attribute Dno, with **xDno = 2** and first-level index blocks **bI1Dno = 4**. There are NDV(Dno, EMPLOYEE) = 125 distinct values for Dno, so the selectivity of Dno is slDno = (1/ NDV(Dno, EMPLOYEE)) = 0.008, and the selection cardinality is **sDno = (rE \* slDno) = (rE/NDV(Dno, EMPLOYEE)) = 80.**
4. A **secondary** index on Sex, with **xSex = 1**. There are NDV(Sex, EMPLOYEE) = 2 values for the Sex attribute, so the average selection cardinality is **sSex = (rE/NDV(Sex, EMPLOYEE)) = 5000.**

(Note that in this case, a histogram giving the percentage of male and female employees may be useful, unless the percentages are approximately equal.)

We illustrate the use of cost functions with the following examples:

OP1: **σSsn=‘123456789’** (EMPLOYEE)

OP2: **σDno>5**(EMPLOYEE)

OP3: **σDno=5**(EMPLOYEE)

OP4: **σDno=5 AND SALARY>30000 AND Sex=‘F’** (EMPLOYEE)

The cost of the

**Brute force (linear search or file scan)** option S1 will be estimated as

**CS1a = bE = 2000** (for a selection on a nonkey attribute) or

**CS1b = (bE/2) = 1,000** (average cost for a selection on a key attribute).

**For OP1 (selection on a key, answer will contain one record at most)**

we can use either method S1 (linear search) or

method S6a (index); the cost estimate for S6a is

**CS6a = xSsn + 1 = 4 + 1 = 5**, and it is chosen over method S1, whose average cost is CS1b = 1,000 blocks.

**For OP2 (range query on nonkey)**

we can use either method S1 (**linear search**)

**(with estimated cost CS1a = 2,000) or**

method S6b (use **secondary** index on nonkey)

(with estimated cost **CS6b = xDno + (bI1Dno/2) + (rE /2) = 2 + (4/2) + (10,000/2) = 5,004),** so we choose the linear search approach for OP2.

**For OP3 (equality on nonkey)**

we can use either method S1 (with estimated cost CS1a = 2,000) or

method S6a (index)(with estimated cost

**CS6a = xDno + sDno = 2 + 80 = 82),** so we choose method S6a.

**Finally, consider OP4, which has a conjunctive selection condition.**

We need to estimate the cost of using any one of the three components of the selection condition to retrieve the records, plus the linear search approach. The latter gives cost estimate **CS1a = 2000 accesses to disk.**

Using the condition (**Dno = 5**) first gives the cost estimate CS6a = **82**.

Using the condition (**Salary > 30000**) first gives a cost estimate

**CS4 = xSalary + (bE/2) = 3 + (2000/2) = 1003.**

Using the condition (**Sex = ‘F’**) first gives a cost estimate

**CS6a = xSex + sSex = 1 + 5000 = 5001.**

**The optimizer would then choose method S6a on the secondary index on Dno because it has the lowest cost estimate**.

The condition (Dno = 5) is used to retrieve the records, and the remaining part of the conjunctive condition (Salary > 30,000 AND Sex = ‘F’) is checked for each selected record after it is retrieved into memory. Only the records that satisfy these additional conditions are included in the result of the operation.

JOIN

**J1—Nested-loop join.**

**J2—Index-based nested-loop join (using an access structure to retrieve the matching record(s))**

**J3—Sort-merge join.**

**J4—Partition–hash join (or just hash join)**

Suppose that we have the EMPLOYEE file described in the example in the previous section, and assume that the **DEPARTMENT** file consists of

**rD = 125 records stored in**

**bD = 13 disk blocks.**

Consider the following two join operations:

**OP6: EMPLOYEE ∞Dno=Dnumber DEPARTMENT**

**OP7: DEPARTMENT ∞Mgr\_ssn=Ssn EMPLOYEE**

Suppose that we have

a primary index on **Dnumber** of DEPARTMENT with **xDnumber= 1** level and

a secondary index on **Mgr\_ssn** of DEPARTMENT with selection cardinality

**sMgr\_ssn= 1 and**

**levels x Mgr\_ssn= 2.**

Assume that the join selectivity for OP6 is jsOP6 = (1/|DEPARTMENT|) = 1/125 because Dnumber is a key of DEPARTMENT.

Also assume that the blocking factor for the resulting join file is **bfrED= 4 records per block.**

We can estimate the worst-case costs for the JOIN operation OP6 using the applicable methods J1 (nested loop join) and J2 (index based nested loop join) as follows:

Using method J1 with EMPLOYEE as outer loop:

**CJ1 = bE + (bE \* bD) + (( jsOP6 \* rE\* rD)/bfrED)**

**= 2,000 + (2,000 \* 13) + (((1/125) \* 10,000 \* 125)/4) = 30,500**

Using method J1 with DEPARTMENT as outer loop:

**CJ1 = bD + (bE \* bD) + (( jsOP6\* rE\* rD)/bfrED)**

**= 13 + (13 \* 2,000) + (((1/125) \* 10,000 \* 125/4) = 28,513**

Using method J2 (index) with EMPLOYEE as outer loop:

**CJ2c = bE + (rE \* (xDnumber+ 1)) + (( jsOP6 \* rE \* rD)/bfrED**

= 2,000 + (10,000 \* 2) + (((1/125) \* 10,000 \* 125/4) = 24,500

Using method J2 with DEPARTMENT as outer loop:

**CJ2a = bD + (rD \* (xDno + sDno)) + (( jsOP6 \* rE \* rD)/bfrED)**

= 13 + (125 \* (2 + 80)) + (((1/125) \* 10,000 \* 125/4) = 12,763

Using method J4 (hash) gives

**CJ4 = 3\* ( bD + bE ) + (( jsOP6 \* rE \* rD)/bfrED) = 3\* (13+2,000) + 2,500 = 8,539**

Case 5 has the lowest cost estimate and will be chosen.

Notice that in case 2 above, if 15 memory buffer blocks (or more) were available for executing the join instead of just 3, 13 of them could be used to hold the entire DEPARTMENT relation (outer loop relation) in memory, one could be used as buffer for the result, and one would be used to hold one block at a time of the EMPLOYEE file (inner loop file), and the cost for case 2 could be drastically reduced to just bE + bD + (( jsOP6 \* rE \* rD)/bfrED) or 4,513.

If some other number of main memory buffers was available, say nB = 10, then the cost for case 2 would be calculated as follows, which would also give better performance than case 4:

**CJ1 = bD + (⎡bD/(nB – 2)⎤ \* bE) + ((js \* |R| \* |S|)/bfrRS)**

**= 13 + ( ⎡13/8⎤ \* 2,000) + (((1/125) \* 10,000 \* 125/4) = 28,513 = 13 + (2 \* 2,000) + 2,500 = 6,513**