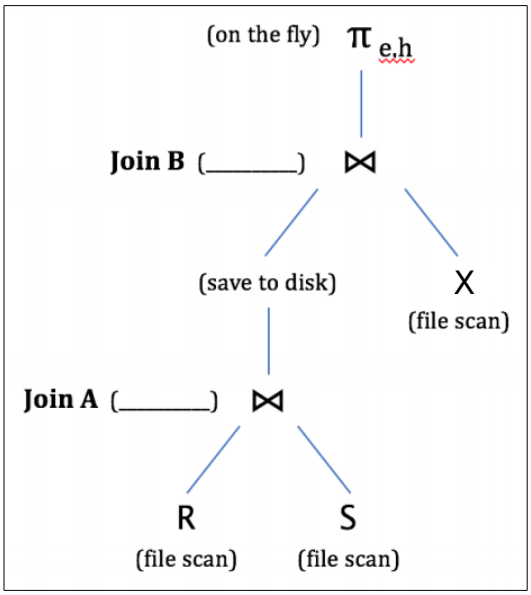
Section 7 Handout: Updates in red

---

1. (414 SP17 Final) We have the query plan and meta data below for R(e, f), S(f, g), and X(g, h):

Table	# tuples	# blocks
R	1,000	100
S	5,000	200
X	100,000	10,000
R ⋈ S	5,000	20

Column	# distinct	Low	High
R.f	100	1	1,000
S.f	1,000	1	2,000
S.g	5,000	1	2,000
X.g	1,000	1	10,000



We have 11 memory pages available.

a. What is the estimated cost of Join B in the plan if we implement it with a **block nested loop** join?

b. What is the estimated cost of Join B the if we use an **indexed** join? Assume that we have a **clustered** index on X(g).

c. What is the estimated cost of Join A in the plan if we implement it with a **block nested loop** join?

d. What is the estimated cost of Join A if we use an **indexed** join? Assume that we have an **unclustered** index on S(f).

e. What is the total cost (IOs) of this plan if we use the best choice of join algorithm for A and B (from above)?

---

2. Create the index that is easiest to create that will make the following queries run faster. Assume there are no previous indexes.

a.

```
SELECT * FROM R
WHERE R.f > 100 AND R.f < 700
```

b.

```
SELECT * FROM S
WHERE S.g = 344
```

3. We have the relation V(m, n, p), W(p, q, r) and the following two queries. For each of the unclustered indexes below identify which queries will run faster under that index versus no index. Assume all attributes range from 0 to 1000 and are distributed uniformly.

(A)
```
SELECT * FROM V
WHERE V.m = 344
```

(B)
```
SELECT * FROM V
WHERE V.m = 344 AND
      V.p = 311
```

(C) (assume nested loop join)
```
SELECT * FROM V, W
WHERE V.p = W.p
```

a. `INDEX idx1 on V(m)`

b. `INDEX idx2 on V(m,p)`

c. `INDEX idx3 on V(p,m)`

---

4. (344 AU16 MT)

Purchase(pid, custId, quantity, price)

T(Purchase) = 1000
B(Purchase) = 100
V(Purchase, price) = 100
Price range = [0, 200)

Customer(custId, name, city)

T(Customer) = 3000
B(Customer) = 200
V(Customer, custId) = 3000

Number of memory pages available = 20

a. Consider the query below and circle the indexes that will produce a speed up:

```
SELECT * FROM Purchase P, Customer C
WHERE P.custId = C.custId AND
      P.price < 100 AND
      C.custId = 42
```

(1) Hashtable index on Purchase(price)	(2) B-tree index on Purchase(pid, price)
(3) Hashtable index on Customer(custId)	(4) Hashtable index on Purchase(custId)
(5) B-tree index on Purchase(price, pid)	(6) Hashtable index on Purchase(price, pid)

b. Which join algorithm would you use to execute the join in a) to minimize execution time? Assume that there are no indexes available. Be clear about how the join will be executed, i.e., what attribute will you sort on if sorting is involved, what relation will you construct a hashtable on if one is needed, etc. Briefly explain why.