CS587 – Database System Implementation – Winter 2018
Homework 2, Due by midnight, Thursday March 8
Haomin He

1. Consider the following queries.
Boats (bid, btype, bname)
Reservations(sid, bid, date, rname)

Query 1:
SELECT rname
FROM Reservations R
WHERE R.date = ’10-31-2013’

Query 2:
SELECT bname, rname
FROM Boats B, Reservations R
WHERE B.bid = R.bid

Query 3:
SELECT bname, rname
FROM Boats B, Reservations R
WHERE B.bid = R.bid
AND B.btype = ‘J24’
AND R.date = ’10-31-2013’

a) For Query 1, if there is a B-tree index on date, what is the estimated cost of this query? (Estimated cost should be calculated in page I/Os as in Lecture 7 in class.)
Answer: Use the date index, Cost = Traverse Index + Retrieve Data Pages
Index traversal is small 2-3 pages (may be in buffer pool).
Big cost is retrieval of data pages.
Disk I/O Cost: $O(\log_F N) + X$ = Index Height + X

Assume the date index is clustered:
Clustered Index: sequential access - assume one disk access per page of tuples
X = # selected tuples / # tuples per page
Disk I/O Cost: $O(\log_F N) + X$ = Index Height + # selected tuples / # tuples per page

Assume the date index is unclustered:
Unclustered Index: random access - assume one disk access per tuple
X = # selected tuples

Disk I/O Cost: $O(\log_F N) + X$ = Index Height + # selected tuples

b) For Query 2, assume no indices, no sorting - what type of join would you use and why?
Answer: Since we don't consider about indices or sorting, we can use hash join; it is relatively cheaper and faster. We build hash table with one relation (or partition if greater than memory). Probe the hash table with the other relation.
Cost: # pages in B + # pages in R (if one relation fits in memory)
If neither R nor B fit in memory - Cost: 3 * (# pages in R + # pages in B)

c) For Query 3, assume no indices, no sorting - what type of join would you use and why?
Answer: We can use hash join or sort-merge join; it depends on the relation (or partition) size.
• Cost for both: 3*(# pages in R + # pages in B) IF:
– BF > sqrt(# pages in smaller relation) for hash join
– BF > sqrt(# pages in larger relation) for sort-merge join
– (BF is number of buffer pages available)

• Hash Join costs less if:
– sqrt (# pages in smaller relation) < BF < sqrt (# pages in larger relation)
– The bigger the difference in relation size, the more Hash Join is favored

• Summary: if you have at least one relatively small relation – use hash join, if both relations are huge, use sort-merge join
• Sort-Merge join is less sensitive to skew
• Sort-Merge produces a sorted result

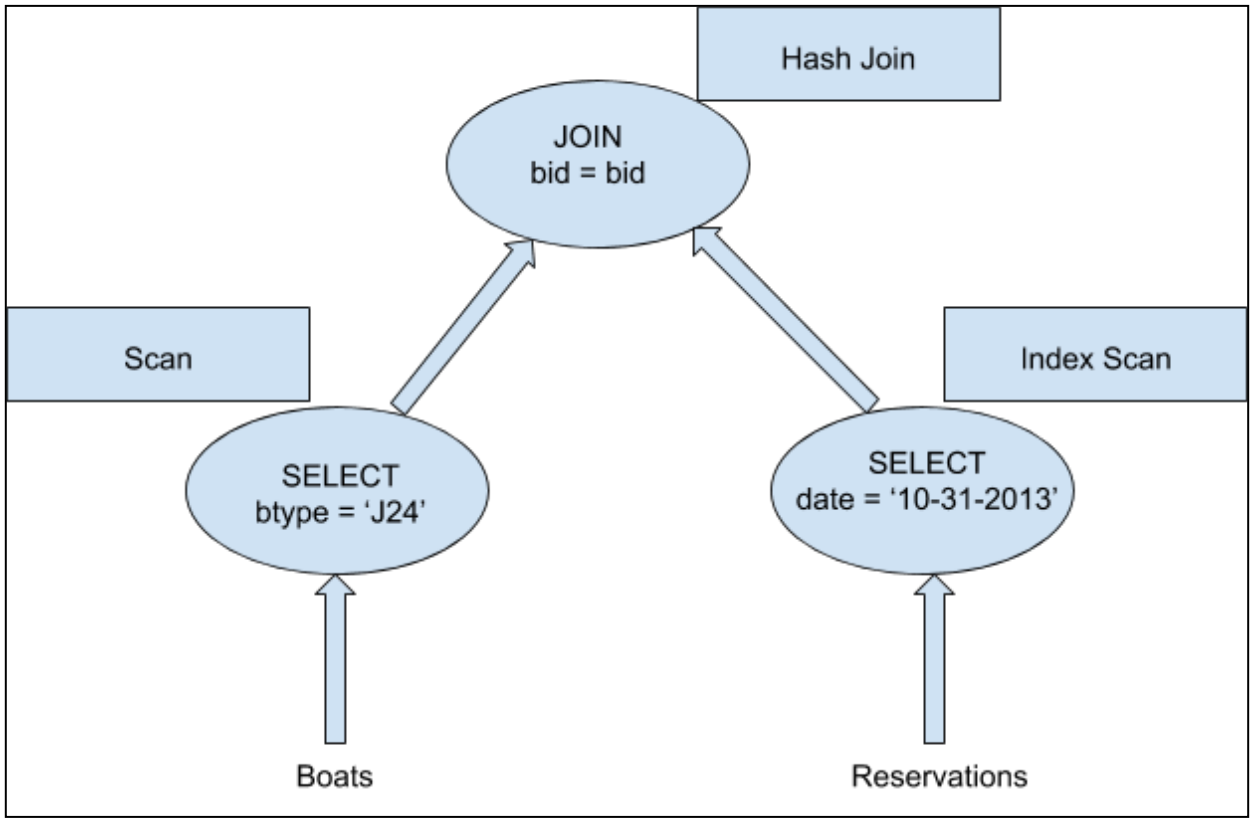
d) (Grad only) Query 3 consists of two selections a join and a projection. Give a query tree for Query 3 (like the one on slide 10 of lecture 7), for each element of the query tree select an implementation option for that operator and give the estimated cost of each implementation option.
Answer: Here I assume there is a B-tree index on date. So the index scan cost is:
Assume the date index is clustered:
Clustered Index: sequential access - assume one disk access per page of tuples
X = # selected tuples / # tuples per page
Disk I/O Cost: $O(\log_F N) + X$ = Index Height + # selected tuples / # tuples per page

Assume the date index is unclustered:
Unclustered Index: random access - assume one disk access per tuple

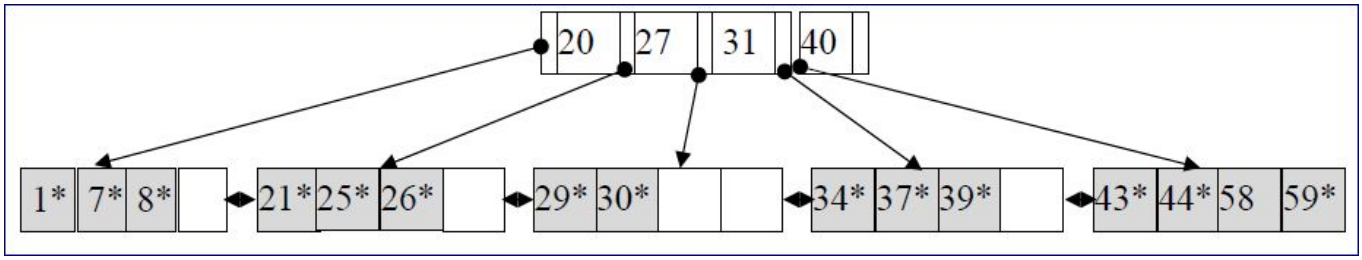
X = # selected tuples
Disk I/O Cost: $O(\log_F N) + X$ = Index Height + # selected tuples

I assume there is no index on btype. So the scan cost is:
Scan the table - Disk I/O cost: O(N) (N is # pages in relation)

We use the hash join and the cost is:
We build hash table with one relation (or partition if greater than memory). Probe the hash table with the other relation.
Cost: # pages in B + # pages in R (if one relation fits in memory)
If neither R nor B fit in memory - Cost: 3 * (# pages in R + # pages in B)

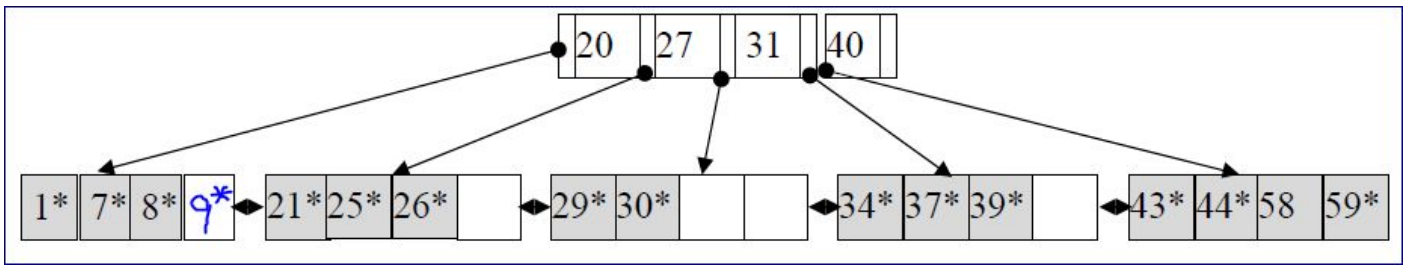


2.



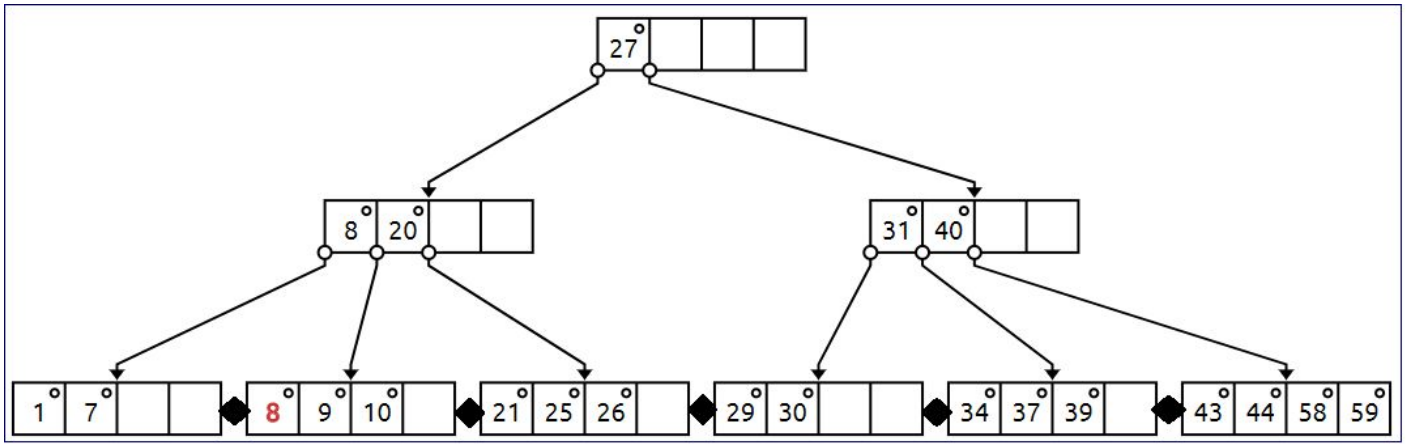
a) Draw the B-Tree after an insert of 9*.

Answer:



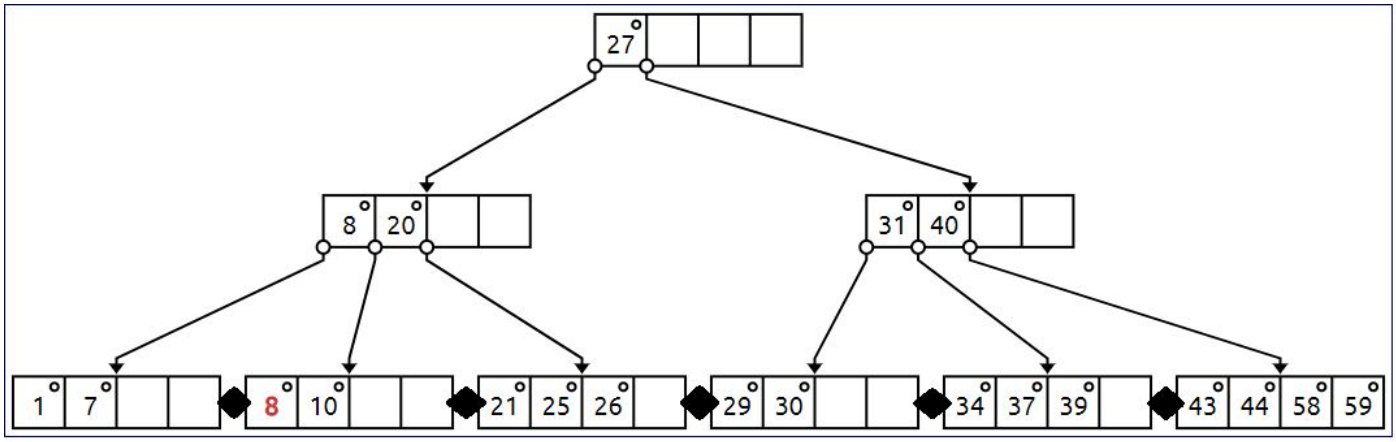
b) Draw the B-Tree after an insert of 10*.

Answer:



c) (Grad only) Draw the B-Tree after a delete of 9*. (Justify your answer if necessary.)

Answer:



3. Consider the following schema and query for freeway traffic data.

Detectors (detectorid, locationname, dtype)
Loopdata (detectorid, timestamp, speed, volume)

SELECT locationname, speed
FROM Detectors D, Loopdata L
WHERE D.detectorid = L.detectorid
AND timestamp between '2012-10-31 10:00' and '2012-10-31 12:00'
AND speed > 100

d) This query is being run on a database and it is running very slow. List three things you would check and/or do to try to improve performance.
Answer: Check for indexes. See if the indexes are picked up by the query or not. See what index structure the database utilizes. For example, hash indexes is good for equality selections; B+ tree indexes handle range queries efficiently.
There may be many ways to answer this query. Different query trees(pushing selections and projections, join orders) may improve query performance. Also, different physical operators for the same logical operator and different parameters (block size, buffer allocation, …) may lead to a better query execution plan.

e) (Grad only) What index or indices would you add to improve the performance of the query?
Answer: Since hash indexes is good for equality selections; B+ tree indexes handle range queries efficiently. I would add a B-tree indexes on timestamp and speed. And add hash indexes on detectorid.

4. Considering the discussion of Hash Join vs Sort-Merge join:
a) Give an example of a query on which you would expect Hash Join to perform better than Sort-Merge Join. Include any assumptions you make (i.e. relation sizes, etc.).
Answer:
We build hash table with one relation. Probe the hash table with the other relation.
Cost: # pages in D + # pages in S (if one relation fits in memory)

• Hash Join costs less if:
– sqrt (# pages in smaller relation) < number of buffer pages available < sqrt (# pages in larger relation)
– The bigger the difference in relation size, the more Hash Join is favored

• Summary: if you have at least one relatively small relation – use hash join, if both relations are huge, use sort-merge join

Assume department table has a small relation size and it can fit in memory.
department (did, departmentbuilding, departmentname)
student (sid, did, enrolldate, studentname)

```
SELECT departmentname, studentname
FROM department D, student S
WHERE D.did = S.did
AND D.departmentbuilding = 'FAB'
AND S.enrolldate = '1-31-2018'
```

b) Give an example of a query on which you would expect Sort-Merge Join to perform better than Hash Join. Include any assumptions you make (i.e. relation sizes, etc.).
Answer:
• Cost: 3*(# pages in D + # pages in S) IF:
– BF > sqrt(# pages in larger relation) for sort-merge join
– (BF is number of buffer pages available)

• Summary: if you have at least one relatively small relation – use hash join, if both relations are huge, use sort-merge join. Sort-Merge join is less sensitive to skew. Sort-Merge produces a sorted result

Assume both tables' relation sizes are huge.
department (did, departmentbuilding, departmentname)
student (sid, did, enrolldate, studentname)

```
SELECT departmentname, studentname
FROM department D, student S
WHERE D.did = S.did
AND D.departmentbuilding = ‘FAB’
AND S.enrolldate = ’1-31-2018’
```

References:
Lecture slides