

CS 386/586 ASSIGNMENT 6 FALL 2011

Part I: Disk Variations

Question 1 (10 points): Some hard disk drives have a second read-write head unit located opposite the first one. What aspect of disk I/O time is this addition most likely to improve? Justify your answer.

Having a second read-write head unit could reduce the *rotational delay* since two heads could access a disk block on the opposite sides at the same time. It could also speed up the *seek time* since we could make one head access data on a track while the other head positions on the next track.

Part II: Evaluating Queries with Indexes

It is sometimes possible to evaluate a particular query using only indexes, without accessing the actual data records.

Consider a database with two tables:

Book(ISBN, title, year, publisher)

Sells(ISBN, vendor, price)

Assume three unclustered indexes, where the leaf entries have the form [search-key value, RID].

<year> on Book

<publisher> on Book

<price, ISBN> on Sells

For Questions 2-9, say which queries can be evaluated with just data from these indexes.

- If the query can, describe how.
- If the query can't, explain why.

Each question worth 10 points.

Question 2:
SELECT MAX(price)
FROM Sells;

Yes. This query can be evaluated by searching through the $\langle \text{price}, \text{ISBN} \rangle$ index for the maximum price.

Question 3:
SELECT ISBN, MIN(price)
FROM Sells
GROUP BY ISBN;

Yes. This query can be evaluated by using the $\langle \text{price}, \text{ISBN} \rangle$ index on Sells. The data needs to be sorted or hashed on ISBN to calculate the mins. But it would not be efficient since records are grouped based on ISBN while the $\langle \text{price}, \text{ISBN} \rangle$ index is sorted by price first.

Question 4:
SELECT AVERAGE(price)
FROM Sells
GROUP BY vendor;

No. None of the indexes contain the vendor information.

Question 5 (think carefully about this one!):
SELECT ISBN, AVERAGE(price)
FROM Sells
GROUP BY ISBN
HAVING COUNT(DISTINCT vendor) > 1;

Yes. The key point is that from schema `Sells(ISBN, vendor, price)` we know that ISBN and vendor combined together form the primary which means the records with the same ISBN must have a different vendor. By grouping on ISBN, the count on vendor information can be ignored (clause `HAVING COUNT(DISTINCT vendor) > 1` can be considered as clause `HAVING COUNT(RID) > 1`). Sort by ISBN to get groups, we can count RIDs (or index entries) to check the HAVING clause, then calculate the average price for qualifying groups.

Question 6:

```
SELECT COUNT(*)  
FROM Book  
WHERE year = 2003 AND publisher = 'Knopf';
```

Yes. We can get the RIDs of books from 2003 by using `<year>` index on Book and the RIDs of books published by Knopf by using `<publisher>` index on Book. And then we could count the RIDs which are in both lists of RIDs.

Question 7:

```
SELECT publisher, COUNT(ISBN)  
FROM Book  
GROUP BY publisher;
```

Yes. ISBN is the primary key for Book which means no ISBN has multiple publishers in this case. So the clause `COUNT(ISBN)` can be replaced by `COUNT(RID)`. And the `<publisher>` index on Book can be used to get the result. Sort by publisher to get groups, we can count RIDs (or index entries) for each publisher.

Question 8:

```
SELECT COUNT(DISTINCT year)  
FROM Book  
WHERE publisher = 'Penguin';
```

Yes. We can find RIDs for publisher Penguin by using `<publisher>` index on Book and search through the `<year>` index to find the years that match to those RIDs. Then we can count the distinct years.

Question 9:

```
SELECT publisher, price  
FROM Book NATURAL JOIN Sells  
WHERE year = 2003;
```

No. There is no index on Book which has ISBN as the search key to natural join Book and Sells.

Part III: Join Algorithms

Questions 10 (20 points): Consider evaluating a join operation that we know is followed by a project, such as

$\text{PROJECT}_{r.C, s.D, s.E, s.F}(\text{r JOIN}_{r.A = s.B} s)$

Assume we are using 8K pages, and that r has 100K pages and s has 500K pages.

Further assume each row in r is 200 bytes long (so 40 rows/page) and each row in s is 500 bytes long (so 16 rows/page).

Consider using the sort-merge implementation of join, with 51 page buffers (50 for input runs, 1 for output run)

(a) Calculate how many page I/Os it takes to merge sort each of r and s .

The total page number of r is $R = 100K$, the total page number of s is $S = 500K$ and the number of buffers is $B = 51$.

To sort r , the number of passes is $1 + \lceil \log_{B-1} [R/B] \rceil = 1 + \lceil \log_{50} [100K/51] \rceil = 3$

To sort s , the number of passes is $1 + \lceil \log_{B-1} [S/B] \rceil = 1 + \lceil \log_{50} [500K/51] \rceil = 4$

So the I/O cost of r is $2 \times 100K \times 3 = 600K$ and the I/O cost of s is $2 \times 500K \times 4 = 4000K$.

(b) Since we do not need all the columns in r and s for the final result, consider a variation of merge sort where on the first pass, we only write out the $r.A$ and $r.C$ columns of r rows, and the $s.B$, $s.D$, $s.E$ and $s.F$ columns of s rows. Calculate the number of page I/Os for sorting each of r and s with this modification, assuming each column is 10 bytes and that there are 8000 bytes of row storage per page.

Each column is 10 bytes, we only write out the $r.A$ and $r.C$ columns of r rows, so r has 20 bytes per row (not 200 bytes). Since the page size is still 8K, so we read in 100K pages and write out 1/10 of 100K pages in the first pass which is 10K pages. By doing 50-way merge, we can get 2000 runs of 5 pages each. Similarly, in the second pass, we read in 10K pages. By doing 50-way merge, we write out 10K pages and 40 runs of 250 pages each. In the third pass, we read in 10K pages. By doing 50-way merge, we write out 10K pages and 1 run of 10K pages. Thus the total number of page I/Os for sorting r is $(100K + 10K) + (10K + 10K) + (10K + 10K) = 150K$

And we can do the similar calculation for s and get the total number of page I/Os for sorting s is $(500K + 40K) + (40K + 40K) + (40K + 40K) + (40K + 40K) = 780K$