The University of British Columbia

Computer Science 404: Database Systems

Midterm Examination #2, March 16, 2005. Instructor: E.M. Knorr

Time: 48 minutes. Only a simple, non-programmable calculator is permitted.
Closed book. No notes, cell phones, PDAs, etc.

Name_____ Student No_____

(PRINT) (Last) (First)

Signature_____

The examination has 6 pages, and that includes this cover sheet. Check that you have a complete paper.

Print your name and ID at the top of this page, and provide your signature. Have your student ID ready.

In the interest of time, it is sufficient to print your initials at the top of pages 2-6.

A simple, non-programmable, non-communicating calculator is permitted. No other aids are allowed.

Work quickly and do the easy questions first. Part marks are available.

The marks for each question are given in braces. **Do not spend too much time on any one question.**

To minimize disruptions during the exam, **please avoid asking the invigilators for help, tips, or explanations.** *Please write down any reasonable assumptions that you are making, if you believe that a question is ambiguous.*

Marks

Page	Max.	Achieved
2	7	
3	5	
4	5	
5	5	
6	6	
Total	28	

1. {3 marks} Provide 3 good reasons for why a DBMS needs to use a sorting operation? (I've provided one example, please give me 3 others).

- user wants the SQL query results to be sorted (e.g., ORDER BY or GROUP BY)
-
-
-

2. {4 marks} Suppose we start with the following Linear Hashing index. Assume that a split occurs every time we have to allocate a new overflow page. There are a maximum of 3 entries per bucket. We'll hash modulo the last *n* bits of the key value (i.e., same hash function as in class).

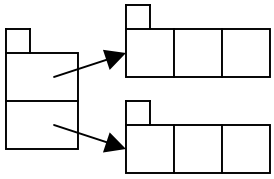
000	00	0	16
		next = 1	
01	01	1	5
10	10	2	6
11	11	3	7
100	00	4	12

Show the final state of the index structure. Insert entries in this order: 8, 10, 14, 28

3. {5 marks} Suppose we have the following entries that we're going to insert into the Extendible Hash index structure shown below. You can assume that each bucket holding the data entries can hold up to 3 entries. Use the same hash function (i.e., modulo last *n* bits of each binary value) just like in class and your assignment. The index already contains the values 15, 18, 26, and 4.

Insert the following new values (in the following order) into the index structure shown. Show the final state of the index structure. Be sure to list the correct values for the nubs (i.e., local and global depths).

Add these new entries: 1, 17, 23, 3, 16



4. Query Evaluation. Consider these table schemas, with primary keys underlined:

HockeyPlayer (playerName, teamName, salary, goals, assists, penalties, address, ...)—has 1200 tuples, and each record contains 400 bytes
HockeyFan (Name, FavouriteTeam, FavouritePlayerName, fanSalary, ...)—has 3,000,000 tuples, and each record contains 200 bytes
HockeyTeam (teamName, teamOwner, ...)—has 30 tuples, and each record contains 100 bytes

A hockey fan is someone who likes to watch hockey. Let's assume that a hockey fan has exactly one favourite hockey team, and exactly one favourite hockey player. But, the fan's favourite hockey player does not have to be on the fan's favourite hockey team. Assuming a uniform distribution for the data, answer the questions on this page and the next 2 pages.

a) {5 marks} The hockey league wants to match up all hockey players with their fans. It plans to create personalized greeting cards containing all the personal information about both the player and the individual fan (to eventually mail to the respective fans and players). The query is as follows:

```
SELECT      *
FROM        HockeyPlayer, HockeyFan
WHERE       playerName = FavouritePlayerName;
```

There are *no indexes* on the tables. For this question, use a sort-merge join to compute the number of I/O's that would be required using 75 buffers. The page size is 4K.

(b) {5 marks} Here are the same schemas, repeated for your convenience.

HockeyPlayer (playerName, teamName, salary, goals, assists, penalties, address, ...)—has 1200 tuples, and each record contains 400 bytes
HockeyFan (Name, FavouriteTeam, FavouritePlayerName, fanSalary, ...)—has 3,000,000 tuples, and each record contains 200 bytes
HockeyTeam (teamName, teamOwner, ...)—has 30 tuples, and each record contains 100 bytes

Let's create an Alternative 2 clustering index on FavouritePlayerName. Compute the number of index pages (leaf and internal) at each level. You can assume that strings are 30 bytes long, numbers are 4 bytes long, and all pointers are 10 bytes long.

(c) {6 marks} Once more, here are the same schemas, repeated for your convenience.

HockeyPlayer (playerName_ teamName, salary, goals, assists, penalties, address, ...)—has 1200 tuples, and each record contains 400 bytes
HockeyFan (Name, FavouriteTeam, FavouritePlayerName, fanSalary, ...)—has 3,000,000 tuples, and each record contains 200 bytes
HockeyTeam (teamName, teamOwner, ...)—has 30 tuples, and each record contains 100 bytes

From part (b), there is a B+-tree clustering index (on FavouritePlayerName). Let us also assume that there is a hash index available on the playerName field. Compute the number of I/O's required for the following query. You are free to use any reasonable calculations that you computed on previous pages for these tables. List any extra assumptions that you need to make. As usual, assume 2.5 I/O's per B+ tree index probe, and 1.2 I/O's for a hash index probe.

```
SELECT      *
FROM        HockeyPlayer, HockeyFan
WHERE       playerName = FavouritePlayerName
            AND  teamName = "Vancouver Canucks";
```

For this question, let us assume that each of the players has approximately the same number of fans. This will greatly simplify the analysis. Do an index-nested loop join (INL) with the most appropriate index.