CS186

Fall 1998

Midterm 2

J.Hellerstein

Midterm Exam: Introduction to Database Systems

This exam has four problems, for a total of 70 points. There are also 10 points of extra credit available. Each problem is made up of multiple questions. You should read through the exam quickly and plan your time-management accordingly. Before beginning to answer a question, be sure to read it carefully and to answer all parts of every question!

1. Cost Estimation

Cage/Fish & Associates hires you to help keep track of their clients and cases. Theyre currently stored as 2 relations (primary keys in **bold**)

`Clients(name, **id**, birthday)`

`-5000 tuples, 50 bytes/tuple (2 tuples/page)`

`Cases(**caseid**, clientid, lawyer, courtdate)`

`-10,000 tuples, 2000 bytes/tuple, each attribute of the tupe is 500 bytes long. (2 tuples/page)`

Some additionaly notes on this schema:

- A page holds 4000 bytes
- caseid has values ranging from 0 to 10,000
- clientid has values ranging from 50 to 5050, and is a foreign key to Clients.
- Unless stated otherwise, a B+Tree index is available on each of the primary keys.

a. [2 points] Consider the following two queries:

i. SELECT* FROM Clients C WHERE C.birthday=4/25/77
ii. SELECT* FROM Cases S WHERE S.lawyer=Georgia Thomas

Assume that the two queries return the same number of qualifying tuples. If youre building indexes on the selected attributed to speed up the queries, for which is a clustered index(as opposed to unclustered) more important, and why? Be sure to state any assumptions that you make.

a. [4 points] Estimate the number of tuples that would pass the selection Cases.casesid = Cases.clientid.

What assumptions are you making in this approximation?

b. [9 points] Assume that you have two indexes on Cases. The first is a clustered B+tree of height 2 on (lawyer, caseid), containing 201 pages (note that key compression makes this possible, even though

the keys themselves are 500 bytes long!). The second is an unclustered B+tree of height 2 on (caseid) containing 101 pages. Consider the query:

SELECT lawyer

FROM Cases

WHERE caseid > 5000

i. Using the clustered B+-tree, what is the cost of running this query?
ii. Using the unclustered B+tree, what is the cost of running this query?
iii. What would be the cost if you used the unclustered tree but sorted the rids of the matching tuples before fetching the tuples from the heap file? Assume that the rids fit in memory

2) Query Processing

Consider a system containing three join algorithms: (I) Block Nested Loops, (ii) Sort Merge and (iii) Hash Join.

a. [3 points] For each of the join algorithms in the system, discuss how its performance is affected by skew in the data values of the join columns.
b. [3 points] For each of the join algorithms in the system, discuss how its performance is affected by making a major change in the size of only one input relation.
c. [9 points] Two relations R and S have sizes (in #pages) of M=500 and N=100, respectively. You are to perform an equijoin of R and S. Given B=12 buffers, how many I/Os would be required for each of the join algorithms in the system?

3. Query Optimization

a. [6 points] Assume you have a simple PC database system that contains only two join algorithms: simple nested-loops join, and index nested loops join. You have a database with the following schema (primary keys in **bold**):

```
web_pages(URL: text, author: text, content:text, last_edited: date)

links(from_URL: text, to_URL: text, last_clicked: date)
```

There is only one index, a B+tree on web_pages.URL. Assuming the system uses a System-R style optimizer (i.e. the algorithm we learned in class), give *three* different query execution plans that it might produce for the following query. Each of your plans should have a different cost for a typical database (you need not write down the cost in your answer). Your answer can be three query plan trees, or three plans *concisely* described in English:

```
SELECT URL, to_URL, last_clicked

FROM web_pages, links

WHERE URL = from_URL

AND author = Rasputin:
```

- [9 points] There are three parts of an SQL query that can cause a column (or columns) to be considered an "interesting order". *Briefly* describe each of these, and for each one *briefly* explain why an ordered input can lower the cost of evaluating that part of the query.
- [5 points] Consider the following query:

```
SELECT *

FROM T
WHERE W.URL = http://db.cs.berkeley.edu

AND EXISTS (SELECT *

FROM links L

WHERE L.last_clicked = w.last_edited);
```

Describe in English 2 query plans for this query, and for each one describe how many times the links table is scanned.

- **[Extra Credit: 10 pts]**
  Assume you have an SQL query of the following form:

```
SELECT *

FROM T

WHERE EXISTS SubQuery1

AND EXISTS SubQuery2;
```

Assume that each subquery is correlated with T, and that there are no indexes in the system. Assum the optimizer has a built-in function call

`RF_Exists()` that can give an accurate reduction factor for an EXISTS clause. Given this information and what you know from class, can you describe how to choose the best plan for the query?

4. SQL

A bookmaker (also known as a "bookie") is a person who takes bets on various sprting events. He determines odds on the outcome of these events, and collects and payrs off the bets made by others through him.

Bob is a bookie that doesnt really understand anything about sports or gambling; hence, he has a large customer base. His customer base is growing too large for him to handle his operation manually. Since he knows a little about databases, he decides to use a database to keep track of customer bets on sprting events,

Hoping eventually he will be able to bring his operation online.

He decides on the following schema for his initial database (primary keys are in **bold**, referential integrity to be enforced on all cross-table references):

`Customers(`**`c_id`**`: integer, name: string, loc: string, favorite_limb: string)`

This table contains the customers, their names, where they live, and their favorite limbs(if they cant pay their bets, Bob takes their favorite limb).

`Teams(`**`team_id`**`: integer, name: string, league: string, home: string)`

This table holds the teams Bob allows his customers to bet on. It contains name of the team, the league to which the team belongs, i.e. "NFL" or "NCAA", and the home location of the team.

`Games(`**`game_id`**`: integer, fav_team: integer, underdog_team: integer, date:, DATE, odds: float)`

This table holds the games Bob allows his customers to bet on. It contains a unique game_id, the team_id of the team from the Teams table favored to win (fav_team), the team_id of the opponent in the Teams table (underdog_team), the date of the game, the odds for the game.

`Bets(`**`c_id`**`: integer, game_id: integer, pick: integer, betted_amount: float)`

This table contains the bets made by customers from the Customers table on specific games from the Games table. Each records the customers bet of an amount (betted_amount) on a team they pick from the Teams table (pick).

The betting rules work as follows:

- Customers pay Bob the betted_amount for each bet.
- If a customoer bets on a game, and the team they betted for wins, and that team is the favorite team, then Bob gives the customer 2 times the betted_amount.
- If the customer bets on a game, and the team they betted for wins, and that team is the underdog, then Bob gives the customer betted_amount + odds*betted_amount.
- If the customer bets on a game, and the team they betted for loses, Bob keeps the betted_amount.

- **Note**

: the key of Bets is (c_id, game_id), so a customer can only bet once per game.

Assume there are no nulls in any tables of the database.

You need to help Bob figure out some of the SQL issues. Feel free to use views if they help you clarify your thinking (or Bobs)!

a. [4 points] Write the SQL DLL for the Games table.
b. [4 points] Write an SQL query to find the names of teams that were picked to win in more than 50 bets.
c. [5 points] Suppose that all the underdogs win. Write an SQL query to give they list of customers who lost all their bets. The list should contain customers names, locations, how much they owe, and their favorite limbs.
d. [7 points] Write an SQL integrity constraint that assures no team in the NFL league plays more than once a day.