University of California, Berkeley
College of Engineering
Computer Science Division – EECS

Spring 2006 Prof. Michael J. Franklin

MIDTERM II

CS 186 Introduction to Database Systems

Question 1 – SQL [6 parts, 40 points total]

Consider the following basketball schema with four relations (primary keys are underlined):

Player (pid: integer, pname: varchar(50), team: varchar(30))

Each player plays for only one team. The team field is a foreign key to Team.

Team (tname: varchar(30), city: varchar(20))

There can be multiple teams from the same city.

Game (gameid: integer, homeTeam: varchar(30), awayTeam: varchar(30), homeScore: integer, awayScore: integer)

homeTeam and awayTeam are foreign keys to Team. Two teams may play each other multiple times each season. The homeTeam and awayTeam are always different. Ties are not allowed.

Stats (pid: integer, gameid: integer, points: integer, assists: integer, rebounds: integer)

Stats records the performance statistics of a player within a game. If a player does not play in a particular game, they will not have any statistics recorded for that game. gameid is a foreign key to Games. pid is a foreign key to Player.

Write **SQL** queries for the following. **All of these are doable without creating views or temporary tables. You may do so if you feel it is necessary but we'll take off some points if you do. We will also take off points for doing unnecessary joins, distincts, etc.**:

a) [3 points] How many different players have scored more than 50 points in a game?

b) [5 points] For all the Players who have played in at least one game, list their pid and the total number of points they have scored. Return the list in *descending* order of the total number of points.

Question 1 – SQL continued
Recall the schema:

Player (pid, pname, team)
Team (tname, city)
Game (gameid, homeTeam, awayTeam, homeScore, awayScore)
Stats (pid, gameid, points, assists, rebounds)

c) [5 points] A "triple double" is when a player's number of assists, rebounds, and points in a game are all 10 or more. For each triple double, list the name (pname) and team (tname) of the player who got it, and the city where the game was played.

d) [7 points] List the names of teams who never lost a game at home.

Question 1 – SQL continued
Recall the schema:

Player (pid, pname, team)
Team (tname, city)
Game (gameid, homeTeam, awayTeam, homeScore, awayScore)
Stats (pid, gameid, points, assists, rebounds)

e) [10 points] List the name of each team that won more games at home than they lost at home.

f) [10 points] List the name of each team and a count of the number of games the team has won overall (i.e., regardless of whether they were the home team or away team).

Question 2 – Hashing [4 parts, 15 points total]
Consider a relation containing information about university students:

> Students (sid: integer, sname: varchar(50), street: varchar(50), city: varchar(30), age: integer)

consisting of 250,000 tuples, where there are 100 tuples per block (i.e., the students file is 2500 blocks long).

a) (2 points) Consider a simple, 2-pass disk-based hashing strategy as described in lecture, where in the first pass the file is partitioned and in the second pass each of the partitions is hashed (i.e., not hybrid hashing!). Ignoring any space overhead for building hash tables and assuming a perfect hash function, what is the minimum number of memory pages (same size as file blocks) required to hash the Students relation in 2 passes?

b) (3 points) In part (a) above, how many I/Os will be required? Assume that the relation is originally on disk and that the result of the hashing operation must also be written to disk.

c) (5 points) Now, suppose that the hash function used for part (a) does not work very well. Describe briefly why 2 passes may not be sufficient and how a complete algorithm would solve this problem.

d) (5 points) Now, consider a combined Hash/Aggregation operator using "hybrid hash", such as the one you implemented in HW 2 and the query "**SELECT COUNT(*) FROM Students GROUP BY age**". Given a relatively small amount of memory (say, 20 pages or so), would you expect the hybrid hash approach to perform better than the regular two-pass hashing approach on this query? Why or why not? (state any assumptions you are making and answer concisely)

Question 3 – Query Optimization [4 parts, 25 points total]
Consider the following relational schema (with primary keys underlined):

> **STUDENTS** (sid, s_name, street, city, age)
> **COURSES** (cid, c_name, prof_name)
> **REGISTERED** (sid, cid, credits)
> *Where sid and cid are foreign keys referencing STUDENTS and COURSES respectively.*

With the following characteristics:

- The **STUDENTS** relation has 10,000 tuples
- 20 tuples of **STUDENTS** fit in one disk block
- NKeys(city) for **STUDENTS** is 200 (i.e., there are 200 distinct values for city)
- NKeys(age) for **STUDENTS** is 50
- The **REGISTERED** relation has 40,000 tuples
- 50 tuples of **REGISTERED** fit in one disk block
- The **COURSES** relation has 500 tuples
- 40 tuples of **COURSES** fit in one disk block

and the following indexes:

- A hash index is defined on *sid* for **STUDENTS**
- A *clustered* B+Tree index is defined on the *city* attribute for **STUDENTS**
- An *unclustered* B+Tree index is defined on the *age* attribute for **STUDENTS**

For parts **a, b, and c**, state an *efficient method* for answering this query and state how many disk accesses will have to be made in order to answer the query using this method. Be sure to state which index(es) if any you are using. *State any assumptions you are making. Note, you cannot assume that any of the relations fit in memory*. Part of your grade will depend on how efficient a solution you choose.

a) [5 points]
```
SELECT *
FROM STUDENTS
WHERE city = 'Berkeley' and age = 30;
```

Question 3 – Query Optimization (continued)

b) [5 points] SELECT *
FROM STUDENTS
WHERE city = 'Berkeley' and sid = 123456789;

c) [7 points] Assuming there are five (5) pages of memory available for use in by the join, there are no indexes on **REGISTERED** or **COURSES**, and they are not currently sorted, choose an efficient method for computing the join of these two relations and estimate the number of disk I/Os it would incur. Be sure to state the join method you have chosen, and which relation is inner or outer if appropriate.

d) [8 points] Consider the three way join between **STUDENTS**, **COURSES**, and **REGISTERED**. Show an efficient join ordering (no need to specify a join method for this question) and estimate the number of tuples that will be in the result of each of the joins.

Question 4 – More Query Optimization [5 parts, 20 points total]
Recall the Basketball schema from Question 1:

Player (pid, pname, team); **Team** (tname, city)
Game (gameid, homeTeam, awayTeam, homeScore, awayScore)
Stats (pid, gameid, points, assists, rebounds)

a) [5 points] Consider the four-way join of these relations. Give one join ordering that a System-R optimizer would **NOT** consider, and briefly state why it would not consider it.

b) [3 points] Write a SQL SELECT statement that could take advantage of a composite B+tree index on Stats using the composite key (points, assists, rebounds) but could **not** take advantage of a composite B+tree index using (assists, rebounds, points).

c) [3 points] Write a SQL SELECT statement that could take advantage of a B+tree index on Stats using the key (points) but could **not** take advantage of a Hash index using the key (points).

d) [5 points] Assume that in the Stats relation, there are 10,000 tuples with 50 unique values for "points" and 20 unique values for "rebounds". What is the expected cardinality computed by a System R-style optimizer for the predicate "points = rebounds" on the Stats relation?

e) [4 points] Consider the "points" attribute of the Stats relation. Why might a query optimizer that maintains histograms over values of points provide better query plans than one that uses the System R approach of keeping only the high-low values and the number of distinct values?