Homework Assignment #3

Database Systems course

Objectives

To understand and be able to design integrity constraints.

To understand query execution tree

To understand and apply query optimization

Submission

This homework is not for submission, it can be used to self test your knowledge on these issues.

Question 1 (Integrity Constraints)

Consider the following relational schema. An employee can work in more than one department; the pct time field of the Works relation shows the percentage of time that a given employee works in a given department.

Emp(eid: integer, ename: string, age: integer, salary: real)

Works(eid: integer, did: integer, pct time: integer)

Dept(did: integer, budget: real, managerid: integer)

Write integrity constraints (domain, key, foreign key, or CHECK constraints; or assertions) to ensure each of the following requirements, considered independently.

1. Employees must make a minimum salary of $1000.
2. Every manager must be also be an employee.
3. A manager must always have a higher salary than any employee that he or she manages.

Question 2 (Query execution)

Consider the following scenario:

Dept(did: integer, projid: integer, budget: real, status: char(10))

Proj(projid: integer, code: integer, report: varchar)

Assume that each Dept record is 40 bytes long and each Proj record is 2000 bytes long on average. There are 5000 tuples in Dept (note that did is not a key by itself) and 1000 tuples in Proj. Each department, identified by did, has 10 projects on average. The file system supports 4000 byte pages, and 12 buffer pages are available. All following questions are based on this information. You can assume uniform distribution of values. State any additional assumptions. The cost metric to use is the number of page I/Os.

Consider the following query:

```
SELECT D.did, COUNT(*)
FROM Dept D, Proj P
WHERE D.projid=P.projid AND D.budget>99000
GROUP BY D.did
```

Assume that department budgets are uniformly distributed in the range 0 to 100,000.

1. Draw the most naive query execution tree.
2. Assuming no prior indices, what is the best join algorithm to use? Be sure to show that there is enough buffer space to run the algorithm you chose.
3. Using the algorithm you chose above, compute the cost of executing this j join.
4. Now draw the best query execution plan you can think of for this query.
5. How does this change the cost of the join algorithm you chose above?
6. Now suppose that there is a clustered B+ tree index on {D.did, D.budget} and a hash index on {P.projid}. Describe the plan with the lowest estimated cost.

<u>Question 3 (Query Optimization)</u>

Consider the following two sections referring to the following relational schema, which includes two relations:

Movie(title, director, year, genre, length)

Revenues(year, average)

(A)

Consider the following query. This query is meant to get the longest movies for years having movies over 100 minutes long:

```
SELECT year, Max(length)
FROM   Movies
WHERE  genre = "comedy"
GROUP BY year
HAVING Max(length) > 100
```

Can you propose transformations on the above query that is likely to reduce the cost of evaluating the query?

This query is a modification which meant to get the longest movies for years having movies over 100 minutes long and counting all comedies in each of those years as well.

```
SELECT year, Max(length), count(*)
FROM   Movies
WHERE  genre = "comedy"
GROUP BY year
HAVING Max(length) > 100
```

Would the above optimizations you proposed above still be valid? Why or why not?

(B)
Consider the following query, which asks for revenue on comedies per year:

```
SELECT year, Count(title) * Revenues.average AS YearTotal
FROM Revenues, Movie
WHERE Movie.year=Revenues.year AND genre=comedy
GROUP BY year
```

Can you propose transformations on the above query that is likely to reduce the cost of evaluating the query?

Suppose the query was modified to also return the number of comedies made per year, i.e.,

```
SELECT year, Count(title) * Revenues.average AS YearTotal, Count(*)
FROM Revenues, Movie
WHERE Movie.year=Revenues.year AND genre=comedy
GROUP BY year
```

Would the above optimizations you proposed above still be valid? Why or why not?