EXERCISE

You have a database with the following schema:

Employee (name, address, phone, dept)
Department (name, floor, manager, dept, area, budget)
Contracts (company, contact, area, rating, state)

E.dept is a foreign key to D.dept
C.area is a foreign key to D.area
D.budget is between $5000 and $30000
C.state has 50 unique values
C.rating is between 1 and 10

There are 10,000 employees, 10 per page, 1,000 pages
There are 50 departments, 10 per page, 5 pages
There are 1,000,000 contracts, 100 per page, 10,000 pages

For B-Tree indexes, there are 100 keys per node.

Unclustered B+Tree Index on E.dept, 50 unique values
Unclustered B+Tree Index on D.dept, 50 unique values
Unclustered Hash Index on C.company, 2,000 unique values
Clustered B+Tree Index on C.state, C.rating

Query 1

```
SELECT    *
FROM      employee E, department D
WHERE     E.dept = D.dept AND D.budget > $10000
```

1) Begin the process of query optimization, by determining all the cheapest and interesting access methods for each relation and their costs. (Pass 1)
2) Enumerate all the two-way joins that the optimizer should estimate costs for, you can use nested loops, index nested loops, hash, and sort-merge join algorithms. You do not need to estimate the costs for each plan

Query 2

```
SELECT    *
FROM      employee E, department D, contracts C
WHERE     E.dept = D.dept AND D.area = C.area AND
          D.budget > $10000 AND C.state = CA AND C.rating > 5
```

1) Enumerate and estimate costs for all plans in Pass 1
2) Enumerate plans considered in Pass 2

Physical Database Tuning

For the Employee relations, assume there is only an index on E.name.

Given the following workload
Q1: DELETE FROM Employee WHERE name = x;
Q2: INSERT INTO Employee VALUES (x,y,z…);
Q3: UPDATE Employee SET address = x WHERE name = y;
Q4: SELECT name FROM employee WHERE dept = x;
Q5: SELECT * FROM employee WHERE address > x AND address < y
[such that 8% of the tuples are retrieved]

Q1 is run 20x an hour
Q2 is run 20x an hour
Q3 is run 10x an hour
Q4 is run 200x an hour
Q5 is run 1x an hour

Is it worth adding an unclustered index to the E.address? Assume it would take 1.1 I/Os (on average) to update the address index after the correct index leaf page is found. Also assume the buffer manager starts out empty, and is only able to store index pages in memory.