CS532 Homework #4 Answers

1. [30%] Consider the following SQL query for a banking database system. For simplicity, assume that there is only one account type and the status of a customer is bronze, silver, or gold.

   ```
   select account_id, customer_name
   from accounts
   where status = 'gold member' and customer_city = 'Binghamton';
   ```

   Suppose that (a) each account tuple occupies 100 bytes; (b) all pages are of size 4KB (i.e., 4000 bytes); (c) there are 50 customers (out of total 10,000 customers) with gold member status; and (d) 1000 customers live in Binghamton. Discuss how to evaluate the query for the following cases:

   1) [5%] When there is no index on either status or customer_city, briefly describe how the system can process this query. (Hint: There is only one method in this case.)

   Answer: Scan the entire table and evaluate the selection conditions (status = 'gold member' and customer_city = 'Binghamton') against each and every tuple.

   2) [10%] If there is a secondary index on status and a secondary index on customer_city, which index the system should use first to optimize the query? Simply say ‘status’ or ‘customer_city’ [1%]. Briefly explain your answer [9%].

   Answer: Status. As the number of the gold members is smaller than the number of the customers living in Binghamton, use the index on status to find all gold customers and then further check to find those gold customers who satisfy ‘city =Binghamton’.

   3) [15%] If there is a primary index on customer_city and a secondary index on status, the system will first use the primary index to find the customers in Binghamton and then select gold customers. For simplicity, assume that the entire B+ tree is in memory. Given that, answer the questions below:
      a. How many I/O pages the system should do [2%]? Briefly explain your answer [8%].
      b. Are they sequential or random I/O [1%]? Briefly explain your answer [4%].

   Answer:
   a. 25 I/O pages. Each page can hold 40 accounts = 4KB/100 bytes. Since there is a primary index on customer_city, the 1000 tuples for the customers residing in Binghamton will be stored in 1000/40 = 25 consecutive pages.

b. Sequential I/O since they are stored in consecutive pages.

2. [30%] Consider the join $R \infty_{R.A = S.B} S$, where R and S are two relations. Three join methods, i.e., nested loop, sort merge and hash join, are discussed in class. Nested loop and sort merge may benefit from the existence of indexes. Identify three different situations such that each of the following claims is true for one situation.

   (a) [10%] Nested loop outperforms sort merge and hash join.
   (b) [10%] Sort merge outperforms nested loop and hash join.
   (c) [10%] Hash join outperforms sort merge.

   Here the performance is compared based on the number of I/O pages. Suppose N and M are sizes of R and S in pages, respectively. Without loss of generality, assume that N > M. Also, assume that the memory buffer for the join is not large enough to hold the entire R. Justify your answer.

   Answers:

   (a) Nested loop outperforms sort merge and hash join when S is very small and R has a (primary) index on the joining attribute (i.e., R.A).

   (b) Sort merge outperforms nested loop and hash join if both tables are large such that none can be entirely held in the memory buffer, there is no index on either joining attribute, both tables are already sorted on the respective joining attributes and at least one of the joining attributes does not have repeating values.

   (c) A hash join works better than a nested loop or sort merge join in the following situation: The hash table of the smaller table can be held in memory and at least one memory page remains for the larger table, the larger table R cannot be held in memory and it is not sorted on the joining attribute, and no index exists on either joining attribute.

3. [10%] Consider the following query execution plans:

   Plan A: σ GPA $\geq$ 2(Students $\infty_{Students.SSN = Faculty.SSN}$ Faculty)
   Plan B: (σ GPA $\geq$ 2 (Students)) $\infty_{Students.SSN = Faculty.SSN}$ Faculty

   Which plan will be selected if the four rules for query optimization in Chapter 11 are applied? Is the selected plan more efficient than the other one? Briefly explain your answers.

   Answer: Plan B will be chosen by Rule 1; however, Plan A is likely to be more efficient because only a few students are faculty members at the same time.

4. [30%] Consider the following three relations:

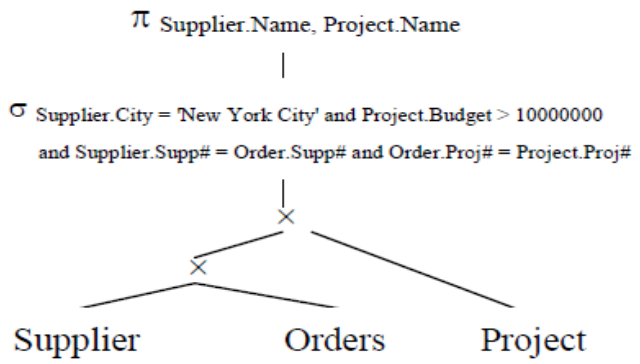
   Supplier(Supp#, Name, City, Specialty)
   Project(Proj#, Name, City, Budget)
   Order(Supp#, Proj#, Part_name, Quantity, Cost)

   Apply the four heuristic optimization rules discussed in class to find an efficient execution plan for the following SQL query. Assume that the number of suppliers in New York City is much bigger than the number of the projects whose budget is over 10 million dollars.

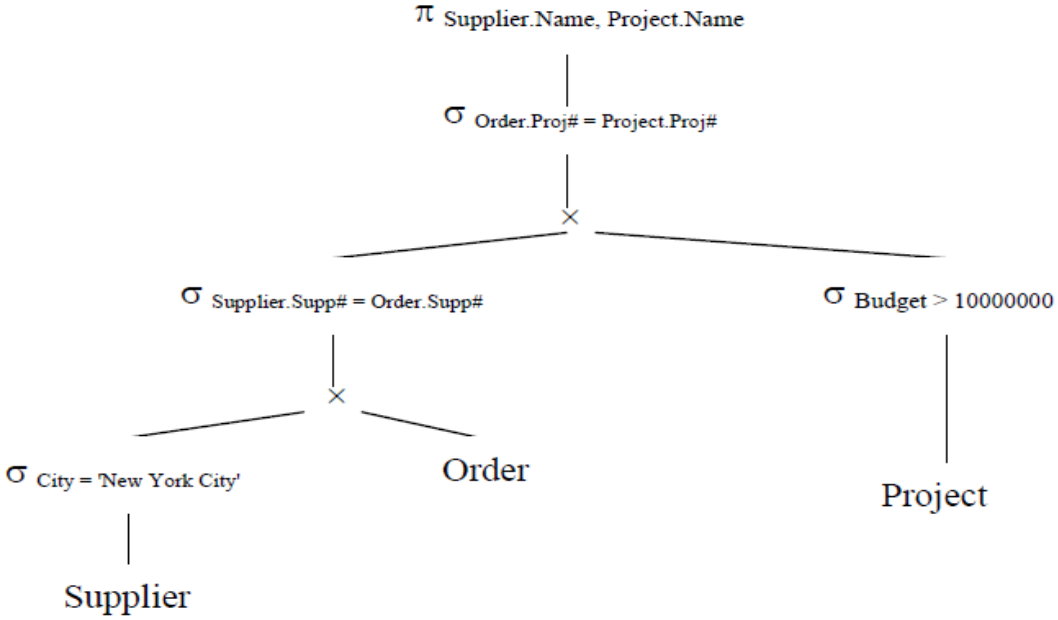
   ```
   select Supplier.Name, Project.Name
   from Supplier, Order, Project
   where Supplier.City = 'New York City' and Project.Budget > 10000000
   and Supplier.Supp# = Order.Supp# and Order.Proj# = Project.Proj#
   ```

   a) [20%] Draw the final query tree acquired by applying the four rules.

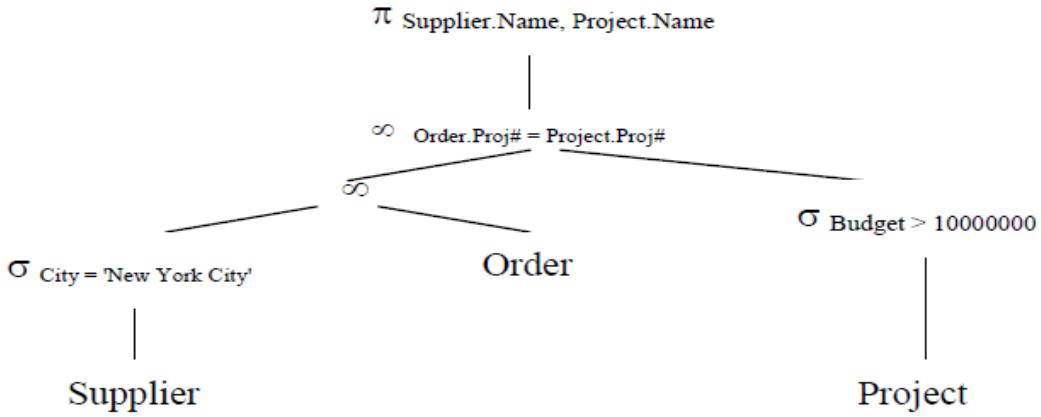
   b) [10%] Write the relational algebra expression that corresponds to the optimized query tree.



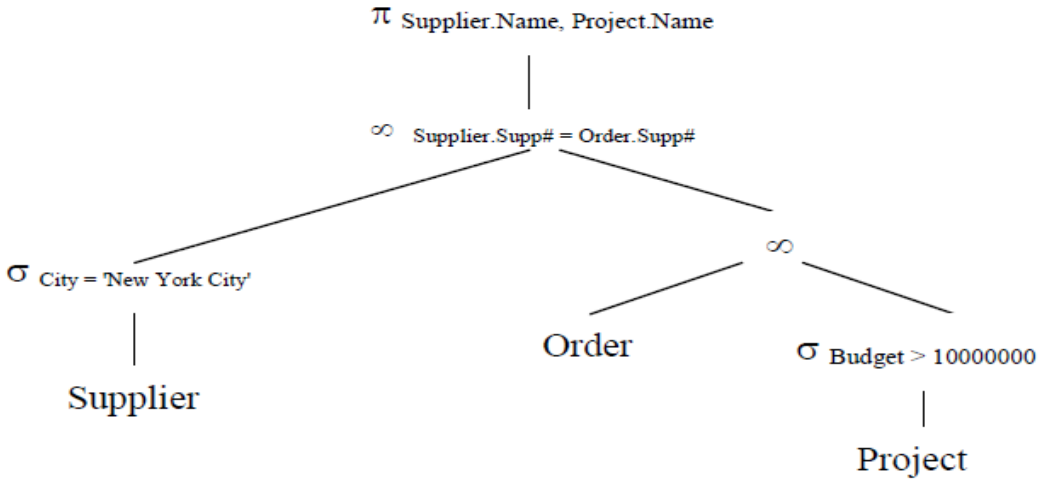
(a) Perform selections as early as possible.



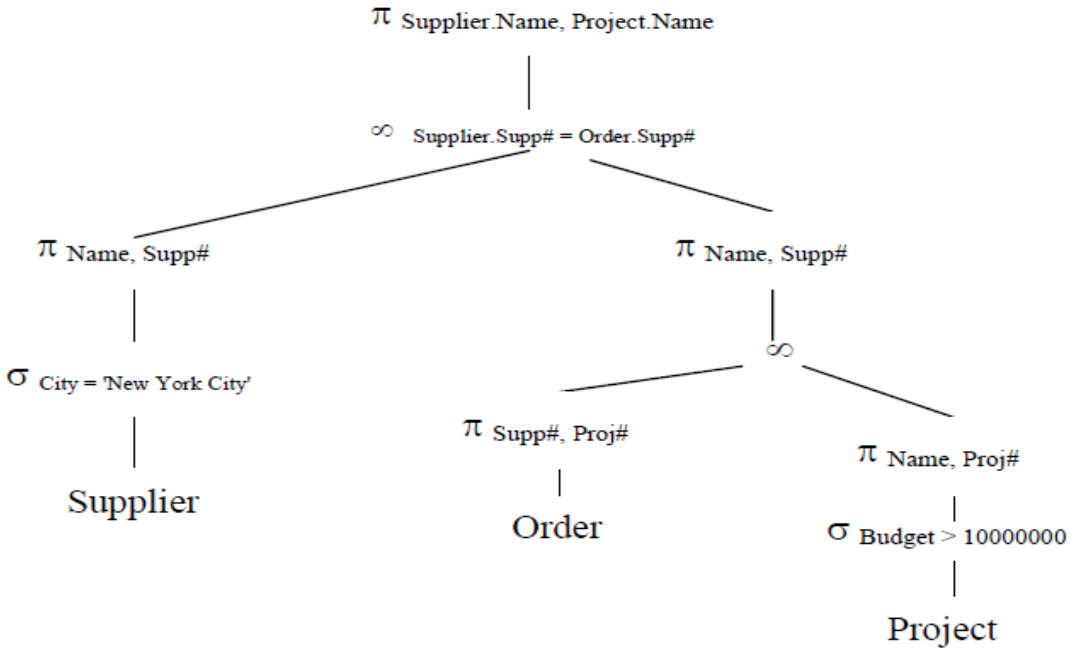
(b) Convert Cartesian products to joins whenever possible.



(c) Perform most restrictive joins first.



(d) Project out useless attributes early.



The final optimized relational algebra expression is:

$$\pi_{\text{Supplier.Name, Project.Name}} \left(\left(\pi_{\text{Name, Supp\#}} \left(\sigma_{\text{City = 'New York City'}}(\text{Supplier})\right)\right) \infty_{\text{Supplier.Supp\# = Order.Supp\#}} \left(\pi_{\text{Name, Supp\#}} \left(\pi_{\text{Supp\#, Proj\#}} (\text{Order}) \infty \pi_{\text{Name, Proj}} \left(\sigma_{\text{Budget} > 10000000} (\text{Project})\right)\right)\right)\right)$$