CHAPTER 16

Query Optimization

Practice Exercises

16.1 Download the university database schema and the large university dataset from dbbook.com. Create the university schema on your favorite database, and load the large university dataset. Use the **explain** feature described in Note 16.1 on page 746 to view the plan chosen by the database, in different cases as detailed below.

a. Write a query with an equality condition on *student.name* (which does not have an index), and view the plan chosen.

b. Create an index on the attribute *student.name*, and view the plan chosen for the above query.

c. Create simple queries joining two relations, or three relations, and view the plans chosen.

d. Create a query that computes an aggregate with grouping, and view the plan chosen.

e. Create an SQL query whose chosen plan uses a semijoin operation.

f. Create an SQL query that uses a **not in** clause, with a subquery using aggregation. Observe what plan is chosen.

g. Create a query for which the chosen plan uses correlated evaluation (the way correlated evaluation is represented varies by database, but most databases would show a filter or a project operator with a subplan or subquery).

h. Create an SQL update query that updates a single row in a relation. View the plan chosen for the update query.

i. Create an SQL update query that updates a large number of rows in a relation, using a subquery to compute the new value. View the plan chosen for the update query.

16.2 Show that the following equivalences hold. Explain how you can apply them to improve the efficiency of certain queries:

a. $E_1 \bowtie_\theta (E_2 - E_3) \equiv (E_1 \bowtie_\theta E_2 - E_1 \bowtie_\theta E_3)$.

b. $\sigma_\theta({}_A\gamma_F(E)) \equiv {}_A\gamma_F(\sigma_\theta(E))$, where θ uses only attributes from A.

c. $\sigma_\theta(E_1 \bowtie E_2) \equiv \sigma_\theta(E_1) \bowtie E_2$, where θ uses only attributes from E_1.

16.3 For each of the following pairs of expressions, give instances of relations that show the expressions are not equivalent.

a. $\Pi_A(r - s)$ and $\Pi_A(r) - \Pi_A(s)$.

b. $\sigma_{B<4}({}_A\gamma_{max(B)\ \textbf{as}\ B}(r))$ and ${}_A\gamma_{max(B)\ \textbf{as}\ B}(\sigma_{B<4}(r))$.

c. In the preceding expressions, if both occurrences of *max* were replaced by *min*, would the expressions be equivalent?

d. $(r \bowtie\!\!\!\subset s) \bowtie\!\!\!\subset t$ and $r \bowtie\!\!\!\subset (s \bowtie\!\!\!\subset t)$
In other words, the natural right outer join is not associative.

e. $\sigma_\theta(E_1 \bowtie E_2)$ and $E_1 \bowtie \sigma_\theta(E_2)$, where θ uses only attributes from E_2.

16.4 SQL allows relations with duplicates (Chapter 3), and the multiset version of the relational algebra is defined in Note 3.1 on page 80, Note 3.2 on page 97, and Note 3.3 on page 108. Check which of the equivalence rules 1 through 7.b hold for the multiset version of the relational algebra.

16.5 Consider the relations $r_1(A, B, C)$, $r_2(C, D, E)$, and $r_3(E, F)$, with primary keys A, C, and E, respectively. Assume that r_1 has 1000 tuples, r_2 has 1500 tuples, and r_3 has 750 tuples. Estimate the size of $r_1 \bowtie r_2 \bowtie r_3$, and give an efficient strategy for computing the join.

16.6 Consider the relations $r_1(A, B, C)$, $r_2(C, D, E)$, and $r_3(E, F)$ of Practice Exercise 16.5. Assume that there are no primary keys, except the entire schema. Let $V(C, r_1)$ be 900, $V(C, r_2)$ be 1100, $V(E, r_2)$ be 50, and $V(E, r_3)$ be 100. Assume that r_1 has 1000 tuples, r_2 has 1500 tuples, and r_3 has 750 tuples. Estimate the size of $r_1 \bowtie r_2 \bowtie r_3$ and give an efficient strategy for computing the join.

16.7 Suppose that a B^+-tree index on *building* is available on relation *department* and that no other index is available. What would be the best way to handle the following selections that involve negation?

a. $\sigma_{\neg(building < \text{“Watson”})}(department)$

b. $\sigma_{\neg (building = \text{“Watson”})}(department)$

c. $\sigma_{\neg (building < \text{“Watson”} \vee budget < 50000)}(department)$

16.8 Consider the query:

```
select *
from r, s
where upper(r.A) = upper(s.A);
```

where “upper” is a function that returns its input argument with all lowercase letters replaced by the corresponding uppercase letters.

a. Find out what plan is generated for this query on the database system you use.

b. Some database systems would use a (block) nested-loop join for this query, which can be very inefficient. Briefly explain how hash-join or merge-join can be used for this query.

16.9 Give conditions under which the following expressions are equivalent:

$$_{A,B}\gamma_{agg(C)}(E_1 \bowtie E_2) \quad \text{and} \quad (_{A}\gamma_{agg(C)}(E_1)) \bowtie E_2$$

where *agg* denotes any aggregation operation. How can the above conditions be relaxed if *agg* is one of **min** or **max**?

16.10 Consider the issue of interesting orders in optimization. Suppose you are given a query that computes the natural join of a set of relations S. Given a subset $S1$ of S, what are the interesting orders of $S1$?

16.11 Modify the FindBestPlan(S) function to create a function FindBestPlan(S, O), where O is a desired sort order for S, and which considers interesting sort orders. A *null* order indicates that the order is not relevant. *Hints*: An algorithm A may give the desired order O; if not a sort operation may need to be added to get the desired order. If A is a merge-join, FindBestPlan must be invoked on the two inputs with the desired orders for the inputs.

16.12 Show that, with n relations, there are $(2(n-1))!/(n-1)!$ different join orders. *Hint:* A **complete binary tree** is one where every internal node has exactly two children. Use the fact that the number of different complete binary trees with n leaf nodes is:

$$\frac{1}{n}\binom{2(n-1)}{(n-1)}$$

If you wish, you can derive the formula for the number of complete binary trees with n nodes from the formula for the number of binary trees with n nodes. The number of binary trees with n nodes is:

$$\frac{1}{n+1}\binom{2n}{n}$$

This number is known as the **Catalan number**, and its derivation can be found in any standard textbook on data structures or algorithms.

16.13 Show that the lowest-cost join order can be computed in time $O(3^n)$. Assume that you can store and look up information about a set of relations (such as the optimal join order for the set, and the cost of that join order) in constant time. (If you find this exercise difficult, at least show the looser time bound of $O(2^{2n})$.)

16.14 Show that, if only left-deep join trees are considered, as in the System R optimizer, the time taken to find the most efficient join order is around $n2^n$. Assume that there is only one interesting sort order.

16.15 Consider the bank database of Figure 16.9, where the primary keys are underlined. Construct the following SQL queries for this relational database.

a. Write a nested query on the relation *account* to find, for each branch with name starting with B, all accounts with the maximum balance at the branch.

b. Rewrite the preceding query without using a nested subquery; in other words, decorrelate the query, but in SQL.

c. Give a relational algebra expression using semijoin equivalent to the query.

d. Give a procedure (similar to that described in Section 16.4.4) for decorrelating such queries.

---

branch(*branch_name*, *branch_city, assets*)
customer (*customer_name*, *customer_street, customer_city*)
loan (*loan_number*, *branch_name, amount*)
borrower (*customer_name*, *loan_number*)
account (*account_number*, *branch_name, balance*)
depositor (*customer_name*, *account_number*)

Figure 16.9 Banking database.