CHAPTER 13

Query Optimization

This chapter describes how queries are optimized. It starts off with statistics used for query optimization, and outlines how to use these statistics to estimate selectivities and query result sizes used for cost estimation. Equivalence rules are covered next, followed by a description of a query optimization algorithm modeled after the classic System R optimization algorithm, and coverage of nested subquery optimization. The chapter ends with a description of materialized views, their role in optimization and a description of incremental view-maintenance algorithms.

It should be emphasized that the estimates of query sizes and selectivities are approximate, even if the assumptions made, such as uniformity, hold. Further, the cost estimates for various algorithms presented in Chapter 12 assume only a minimal amount of memory, and are thus worst case estimates with respect to buffer space availability. As a result, cost estimates are never very accurate. However, practical experience has shown that such estimates tend to be reasonably accurate, and plans optimal with respect to estimated cost are rarely much worse than a truly optimal plan.

We do *not* expect students to memorize the size-estimates, and we stress only the process of arriving at the estimates, not the exact values. Precision in terms of estimated cost is not a worthwhile goal, so estimates off by a few I/O operations can be considered acceptable.

The theory in this chapter is ideally backed up by lab assignments where students study the query execution plans generated by one or more database systems. Most database products have an “explain plan” feature that lets the user find the evaluation plan used on a query. It is worthwhile asking students to explore the plans generated for different queries, with and without indices. Assignment may be designed in which students measure the performance speedup provided by indices. A more challenging assignment is to design tests to see how clever the query optimizer is, and to guess from these experiments which of the optimization techniques covered in the chapter are used in the system.

Exercises

13.15 Suppose that a B$^+$-tree index on (*dept_name*, *building*) is available on relation *department*. What would be the best way to handle the following selection?

$$\sigma_{(building\ <\ \text{“Watson”})\ \wedge\ (budget\ <\ 55000)\ \wedge\ (dept\_name\ =\ \text{“Music”})}(department)$$

Answer: Using the index on (*dept_name*, *building*), we locate the first tuple having (*building* “Watson” and *dept_name*“Music”). We then follow the pointers retrieving successive tuples as long as *building* is less than “Watson”. From the tuples retrieved, the ones not satisfying the condition ($budget < 55000$) are rejected.

13.16 Show how to derive the following equivalences by a sequence of transformations using the equivalence rules in Section 13.2.1.

a. $\sigma_{\theta_1 \wedge \theta_2 \wedge \theta_3}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(\sigma_{\theta_3}(E)))$

b. $\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_{\theta_3} E_2) = \sigma_{\theta_1}(E_1 \bowtie_{\theta_3} (\sigma_{\theta_2}(E_2)))$, where θ_2 involves only attributes from E_2

Answer:

a. Using rule 1, $\sigma_{\theta_1 \wedge \theta_2 \wedge \theta_3}(E)$ becomes $\sigma_{\theta_1}(\sigma_{\theta_2 \wedge \theta_3}(E))$. On applying rule 1 again, we get $\sigma_{\theta_1}(\sigma_{\theta_2}(\sigma_{\theta_3}(E)))$.

b. $\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_{\theta_3} E_2)$ on applying rule 1 becomes $\sigma_{\theta_1}(\sigma_{\theta_2}(E_1 \bowtie_{\theta_3} E_2))$. This on applying rule 7.a becomes $\sigma_{\theta_1}(E_1 \bowtie_{\theta_3} (\sigma_{\theta_2}(E_2)))$.

13.17 Consider the two expressions $\sigma_\theta(E_1 ⟕ E_2)$ and $\sigma_\theta(E_1 \bowtie E_2)$.

a. Show using an example that the two expressions are not equivalent in general.

b. Give a simple condition on the predicate θ, which if satisfied will ensure that the two expressions are equivalent.

Answer:

a. Consider relations *dept*(*id*, *deptname*) and *emp*(*id*,*name*,*dept_id*) with sample data as follows:
Sample data for *dept*:

501	Finance
502	Administration
503	Marketing
504	Sales

Sample data for *emp*:

1	John 501
2	Martin 503
3	Sarah 504

Now consider the expressions

$$\sigma_{deptname<'Z'}(dept \bowtie emp)$$

and $\sigma_{deptname<'Z'}(dept ⟕ emp)$.

The result of the first expression is:

501	Finance	1	John
503	Marketing	2	Martin
504	Sales	3	Sarah

whereas the result of the second expression is:

501	Finance	1	John
502	Administration	null	null
503	Marketing	2	Martin
504	Sales	3	Sarah

b. Considering the same example, if θ included the condition "*name* = 'Einstein'", the two expressions would be equivalent, that is they would always have the same result, since any tuple that is in in *dept* ⟕ *emp* but not in *dept* $\bowtie$ *emp* would not satisfy the condition since its *name* attribute would be null.

13.18 A set of equivalence rules is said to be *complete* if, whenever two expressions are equivalent, one can be derived from the other by a sequence of uses of the equivalence rules. Is the set of equivalence rules that we considered in Section 13.2.1 complete? Hint: Consider the equivalence $\sigma_{3=5}(r) = \{\ \}$.

Answer: Two relational expressions are defined to be *equivalent* when on all input relations, they give the same output. The set of equivalence rules considered in Section 13.2.1 is not complete. The expressions $\sigma_{3=5}(r)$ and $\{\ \}$ are equivalent, but this cannot be shown by using just these rules.

13.19 Explain how to use a histogram to estimate the size of a selection of the form $\sigma_{A\leq v}(r)$.

Answer: Suppose the histogram H storing the distribution of values in r is divided into ranges $r_1, \ldots, r_n$. For each range r_i with low value $r_{i:low}$ and high value $r_{i:high}$, if $r_{i:high}$ is less than v, we add the number of tuples given by

$$H(r_i)$$

to the estimated total. If $v < r_{i:high}$ and $v >= r_{i:low}$, we assume that values within r_i are uniformly distributed and we add

$$H(r_i) * \frac{v - r_{i:low}}{r_{i:high} - r_{i:low}}$$

to the estimated total.

13.20 Suppose two relations r and s have histograms on attributes $r.A$ and $s.A$, respectively, but with different ranges. Suggest how to use the histograms to estimate the size of $r \bowtie s$. Hint: Split the ranges of each histogram further.

Answer: Find the largest unit u that evenly divides the range size of both histograms. Divide each histogram into ranges of size u, assuming that values within a range are uniformly distributed. Then compute the estimated join size using the technique for histograms with equal range sizes.

13.21 Consider the query

```
select A, B
from   r
where r.B < some (select B
                  from   s
                  where s.A = r.A)
```

Show how to decorrelate the above query using the multiset version of the semijoin operation, defined in Exercise 13.14.

Answer: The solution can be written in relational algebra as follows: $\Pi_{A,B}(r \ltimes_\theta s)$ where $\theta = (r.B < s.B \wedge s.A = r.A)$.
The SQL query corresponding to this can be written if the database provides a semi join operator, and this varies across implementations.

13.22 Describe how to incrementally maintain the results of the following operations, on both insertions and deletions:

a. Union and set difference.

b. Left outer join.

Answer:

a. Given materialized view $v = r \cup s$, when a tuple is inserted in r, we check if it is present in v, and if not we add it to v. When a tuple is deleted from r, we check if it is there in s, and if not, we delete it from v. Inserts and deletes in s are handled in symmetric fashion.
For set difference, given view $v = r - s$, when a tuple is inserted in r, we check if it is present in s, and if not we add it to v. When a tuple is deleted from r, we delete it from v if present. When a tuple is inserted in s, we delete it from v if present. When a tuple is deleted from s, we check if it is present in r, and if so we add it to v.

b. Given materialized view $v = r ⟕ s$, when a set of tuples i_r is inserted in r, we add the tuples $i_r ⟕ s$ to the view. When i_r is deleted from r, we delete $i_r ⟕ s$ from the view. When a set of tuples i_s is inserted in s, we compute $r \bowtie i_s$. We find all the tuples of r which previously did not match any tuple from s(i.e. those padded with *null* in $r ⟕ s$) but which match i_s. We remove all those *null* padded entries from the view and add the tuples $r \bowtie s$ to the view. When i_s is deleted from s, we delete the tuples $r \bowtie i_s$ from the view. Also we find all the tuples in r which match i_s but which do not match any other tuples in s. We add all those to the view, after padding them with *null* values.

13.23 Give an example of an expression defining a materialized view and two situations (sets of statistics for the input relations and the differentials) such that incremental view maintenance is better than recomputation in one situation, and recomputation is better in the other situation.
Answer: Let r and s be two relations. Consider a materialized view on these defined by ($r \bowtie s$). Suppose 70% of the tuples in r are deleted. Then recompution is better that incremental view maintenance. This is because in incremental view maintenance, the 70% of the deleted tuples have to be joined with s while in recompution, just the remaining 30% of the tuples in r have to be joined with s.
However, if the number of tuples in r is increased by a small percentage, for example 2%, then incremental view maintenance is likely to be better than recomputation.

13.24 Suppose you want to get answers to $r \bowtie s$ sorted on an attribute of r, and want only the top K answers for some relatively small K. Give a good way of evaluating the query:

a. When the join is on a foreign key of r referencing s, where the foreign key attribute is declared to be not null.

b. When the join is not on a foreign key.

Answer:

a. Sort r and collect the top K tuples. These tuples are guaranteed to be contained in $r \bowtie s$ since the join is on a foreign key of r referencing s.

b. Execute $r \bowtie s$ using a standard join algorithm until the first K results have been found. After K tuples have been computed in the result set, continue executing the join but immediately discard any tuples from r that have attribue values less than all of the tuples in the result set. If a newly joined tuple t has an attribute value greater than at least one of the tuples in the result set, replace the lowest-valued tuple in the result set with t.

13.25 Consider a relation $r(A, B, C)$, with an index on attribute A. Give an example of a query that can be answered by using the index only, without looking at the tuples in the relation. (Query plans that use only the index, without accessing the actual relation, are called *index-only* plans.)
Answer: Any query that only involves the attribute A of r can be executed by only using the index. For example, the query

select sum(A)
from r

only needs to use the values of A, and thus does not need to look at r.

13.26 Suppose you have an update query U. Give a simple sufficient condition on U that will ensure that the Halloween problem cannot occur, regardless of the execution plan chosen, or the indices that exist.

Answer: The attributes referred in the WHERE clause of the update query U should not be a part of the SET clauses of U. This will ensure that the Halloween problem cannot occur.