Selinger Query Optimization

Overview

Consider the relations $R(a, b)$, $S(b, c)$, and $T(c, d)$ with clustered indexes on $R.a$, $S.b$, and $T.c$ and with unclustered indexes on $S.c$ and $T.d$. All columns have type `real`, and all indexes have index keys in the range $[1, 100]$. We want to optimize the following query:

```
SELECT *
 FROM  R, S, T
WHERE  R.b = S.b AND S.c = T.c
  AND  R.a <= 50
  AND  (T.c <= 50 AND T.d <= 20)
```

Pass 1

In the first pass of the Selinger query optimization algorithm, we find the minimum (estimated) cost query plan for every (relation, interesting order) pair. To do so, we estimate the cost of every possible single-relation query plan. We can draw each query plans as a tree where FTS represents a full table scan, IS_x represents a full index scan on a column x, and $\text{IS}_{x \leq k}$ represents an index scan on column x which also applies the filter $x \leq k$. We also make sure to push down as many selections as possible. Here are all eight of the single-relation query plans (which we've also given shorter names q_R, q_{Ra}, etc.):

q_R	q_{Ra}	q_S	q_{Sb}	q_{Sc}	q_T	q_{Tc}	q_{Td}
$\sigma_{R.a \leq 50}$ | FTS | R	$\text{IS}_{a \leq 50}$ | R	FTS | S	IS_b | S	IS_c | S	$\sigma_{T.d \leq 20}$ | $\sigma_{T.c \leq 50}$ | FTS | R	$\sigma_{R.d \leq 20}$ | $\text{IS}_{c \leq 50}$ | R	$\sigma_{R.c \leq 50}$ | $\text{IS}_{d \leq 20}$ | R

Now, let's compute the interesting order, I/O cost, and output size of each of these query plans. Let $[R]$, $[S]$, and $[T]$ be the number of *pages* in R, S, and T. Similarly, let $|R|$, $|S|$, and $|T|$ be the number of *tuples* in R, S, and T. Also assume it takes 2 I/Os to traverse a B+ tree index.

Query Plan	Interesting Order	I/O Cost	Output Size
q_R	None	$[R]$	$0.5[R]$
q_{Ra}	None	$2 + 0.5[R]$	$0.5[R]$
q_S	None	$[S]$	$[S]$
q_{Sb}	(b)	$2 + [S]$	$[S]$
q_{Sc}	(c)	$2 + \|S\|$	$[S]$
q_T	None	$[T]$	$0.1[T]$
q_{Tc}	(c)	$2 + 0.5[T]$	$0.1[T]$
q_{Td}	None	$2 + 0.2\|T\|$	$0.1[T]$

Some notes on interesting order:

- q_R, q_S, and q_T have no interesting order because their output is not sorted.
- q_{Ra} produces tuples sorted by a, so you might think that the interesting order of q_{Ra} is (a). However, the query never joins on $R.a$, so we do not consider this sort order interesting. Similarly, q_{Td} has no interesting order because the query does not join on d.
- q_{Sb}, q_{Sc}, and q_{Tc} have interesting orders (b), (c), and (c) because their output is sorted on a column which is later joined.

Some notes on I/O cost:

- The I/O cost of q_R, q_S, and q_T is $[R]$, $[S]$, and $[T]$ because a full table scan requires us to read every page in a relation.
- The I/O cost of q_{Ra} is $2 + 0.5[R]$. It takes 2 I/Os to read the B+ tree index on $R.a$. Then, we have to read every tuple with $R.a \leq 50$. Because the values of a fall in the range $[0, 100]$, the selectivity of $R.a \leq 50$ is 0.5. Thus, we estimate that only half of the tuples satisfy $R.a \leq 50$. Because the index is clustered, we only have to read half the pages: $0.5[R]$. A similar line of reasoning can be used to compute the I/O cost of $q_S b$ and $q_T c$.
- The I/O cost of q_{Td} is $2 + 0.2|T|$. It takes 2 I/Os to read the B+ tree index on $T.d$ Then, we have to read every tuple with $T.d \leq 20$. Because the values of d fall in the range $[0, 100]$, the selectivity of $T.d \leq 50$ is 0.2. Thus, we estimate that only a fifth of the tuples satisfy $T.d \leq 20$. Because the index is unclustered, it takes us 1 I/O to retrieve each of these tuples. Thus, it takes $0.2|T|$ I/Os. A similar line of reasoning can be used to compute the I/O cost of q_{Sc}.

Some notes on output size:

- Note that the estimated output size of q_R and q_{Ra} is the same; the estimated output size of q_S, q_{Sb}, and q_{Sc} is the same; and the estimated output size of q_T, q_{Tc}, and q_{Td} is the same. This is no coincidence. The estimated output size of a relational algebra expression will be the same no matter which query plan we choose to implement it.
- The estimated output size of q_R and q_{Ra} is $0.5[R]$ because the selectivity of $R.a \leq 50$ is 0.5.
- The estimated output size of q_S, q_{Sb}, and q_{Sc} is $[S]$ because we read every single tuple of S.
- The estimated output size of q_T, q_{Tc}, and q_{Td} is $0.2[T]$ because the selectivity of $T.c \leq 50 \wedge T.d \leq 20$ is $0.5 \times 0.2 = 0.1$.

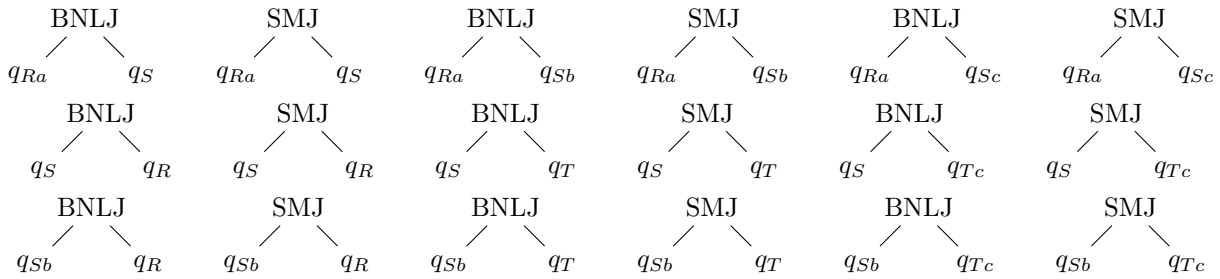
Now that we've computed the I/O cost for every single-relation query plan, we retain the lowest cost plan for each (relation, interesting order pair). We'll use this information in Pass 2:

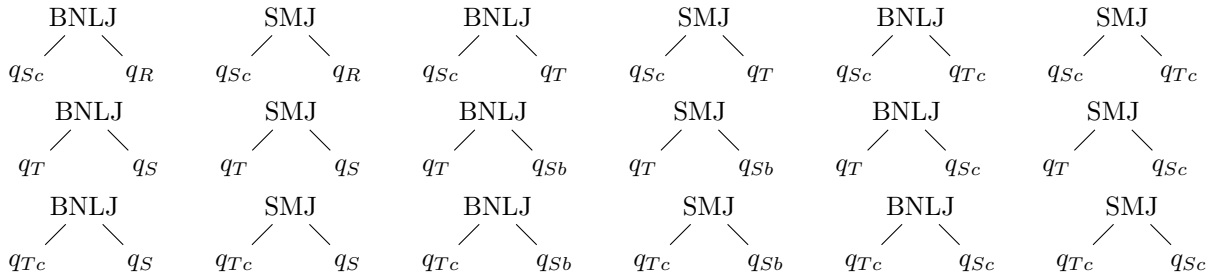
Relation	Interesting Order	Lowest Cost Query Plan
R	None	q_{Ra}
S	None	q_S
S	(b)	q_{Sb}
S	(c)	q_{Sc}
T	None	q_T
T	(c)	q_{Tc}

Notice that we did not retain q_R because q_{Ra} had lower cost. Similarly, we did not retain q_{Td} because q_T had lower cost. Also note that even though q_{Sc} has a very high cost, there is no other single-relation query plan over S with the interesting order (c), we retain q_{Sc}.

Pass 2

In the second pass of the Selinger query optimization algorithm, we find the minimum (estimated) cost query plan for every (set of 2 relation, interesting) order pair. To do so, we estimate the cost of every possible two-relation query plan in which the left and right subqueries are two of the minimum cost single-relation query plans we computed during Pass 1. Considering only block nested loop joins (BNLJ) and sort-merge joins (SMJ), we can list all the two-relation query plans we will consider:





Notice that this list doesn't include *every* pair of single-relation query plans from Part 1. Notice that BNLJ(q_{Ra}, q_T) is missing for example. This is because R does not join with T in our query, so we ignore it. Also notice that BNLJ(q_{Ra}, q_S) is considered but BNLJ(q_R, q_S) is not. This is because we q_{Ra} has a lower cost than q_R and both have the same interesting order.

As with Pass 1, we now compute the interesting order, I/O cost, and output size of these query plans. There's a lot of query plans, so let's only look at a handful. Let B be the number of buffer pages, let $SC(x) = 2x\lceil 1 + \log_{B-1}(\lceil \frac{x}{B} \rceil)\rceil$ be the cost of sorting x pages, and assume the merge phase of a sort merge join on two relations of size n and m can be done in $n + m$ I/Os.

Query Plan	Interesting Order	I/O Cost	Output Size
BNLJ(q_{Ra}, q_S)	None	$0.5[R] + \left\lceil \frac{0.5[R]}{B-2} \right\rceil [S]$	$\frac{0.5[R][S]}{100}$
SMJ(q_{Ra}, q_S)	None	$SC(0.5[R]) + SC([S]) + 0.5[R] + [S]$	$\frac{0.5[R][S]}{100}$
BNLJ(q_{Ra}, q_{Sb})	None	$0.5[R] + \left\lceil \frac{0.5[R]}{B-2} \right\rceil [S]$	$\frac{0.5[R][S]}{100}$
SMJ(q_{Ra}, q_{Sb})	None	$SC(0.5[R]) + 0.5[R] + [S]$	$\frac{0.5[R][S]}{100}$

Some notes on interesting order:

- BNLJ(q_{Ra}, q_S) and BNLJ(q_{Ra}, q_{Sb}) have no interesting order because a BNLJ does not output tuples in any particular order, even if one or both of its inputs are sorted.
- SMJ(q_{Ra}, q_S) and SMJ(q_{Ra}, q_{Sb}) output tuples sorted on (b), but the (b) column is no longer interesting since it will no longer be used as part of a join.

Some notes on I/O cost:

- Estimating the IO cost of BNLJ(q_{Ra}, q_S) and BNLJ(q_{Ra}, q_{Sb}) is a straightforward use of the BNLJ cost formula where q_{Ra} has $0.5[R]$ pages and q_{Sb} has $[S]$ pages. These values were computed in Pass 1.
- The cost of SMJ(q_{Ra}, q_S) is $SC(0.5[R]) + SC([S]) + 0.5[R] + [S]$. Neither q_{Ra} nor q_S output tuples sorted by (b), so we have to sort both. Sorting q_{Ra} takes $SC(0.5[R])$ I/Os and sorting q_S takes $SC([S])$ I/Os. Merging the two takes $0.5[R] + [S]$ I/Os.
- The cost of SMJ(q_{Ra}, q_{Sb}) is $SC(0.5[R]) + 0.5[R] + [S]$. q_{Ra} is not sorted by (b), but q_{Sb} is. Thus, we have to sort q_{Ra}, which takes $SC(0.5[R])$ I/Os, but we do not have to sort q_S. Merging the two takes $0.5[R] + [S]$ I/Os.

Some notes on output size:

- As we noted in Pass 1, the output size of a relational algebra expression is the same no matter which plan we choose to implement it. Thus, BNLJ(q_{Ra}, q_S), SMJ(q_{Ra}, q_S), BNLJ(q_{Ra}, q_{Sb}), and SMJ(q_{Ra}, q_{Sb}) all have the same output size. The maximum size of $R \bowtie_{R.b=S.b} S$ is $[R][S]$. The selectivity of $R.a \leq 50 \wedge R.b = S.b$ is $0.5 \times (\frac{1}{\max(50,100)})$ which is $0.5 \times \frac{1}{100}$. Thus, the output size is $\frac{0.5[R][S]}{100}$.