CS 186 Spring 2024 | Introduction to Database Systems | Lakshya Jain | DIS 7

1 Selectivity Estimation

Consider two relations R(a, b, c) and S(a), with 1000 tuples and 500 tuples respectively.
We have:

- An index on R.a with 50 unique integer values uniformly distributed in the range [1, 50]
- An index on R.b with 100 unique float values uniformly distributed in the range [1, 100]
- An index on S.a with 25 unique integer values uniformly distributed in the range [1, 25].

Assume that columns are independent (R.a, R.b, and R.c are independent).
Use selectivity estimation to estimate the number of tuples produced by the following queries.

1. SELECT * FROM R
   1000 (no predicates, so select all tuples)
2. SELECT * FROM R WHERE a = 42
   Sel = 1 / unique values of a = 1/50
   1000 * (1/50) = 20
3. SELECT * FROM R WHERE c = 42
   Sel = 1/10 because we do not have any information on the relation we use 1/10 as default.
   1000 * (1/10) = 100
4. SELECT * FROM R WHERE a <= 25
   Sel = (v – low) / (high – low + 1) + 1 / distinct values = (25 – 1) / (50 – 1 + 1) + 1/ 50 = 1/2
   1000 * (1/2) = 500
5. SELECT * FROM R WHERE b <= 25
   Sel = (v - low) / (high - low) = (25 - 1)/(100 - 1) = 24/99 = 0.2424...
   floor(1000 * (0.2424...)) = 242
6. SELECT * FROM R WHERE c <= 25
   Sel = 1/10
   1000 * (1/10) = 100
7. SELECT * FROM R WHERE a <= 25 AND b <= 25
   Sel = Sel(a <= 25) * Sel(b <= 25) = 1/2 * 24/99 = 0.1212...
   floor(1000 * (0.1212...)) = 121
8. SELECT * FROM R WHERE a <= 25 AND c <= 25
   Sel = Sel(a <= 25) * Sel(c <= 25) = 1/2 * 1/10 = 1/20

1000 * (1/20) = 50

9. SELECT * FROM R WHERE a <= 25 AND a > 10
   Sel(10 < a <= 25) = (25 - 10) / 50 = 0.3
   1000 * 0.3 = 300
10. SELECT * FROM R WHERE a <= 25 OR b <= 25
    Sel = Sel(a <= 25) + Sel(b <= 25) – Sel(a <= 25) * Sel(b <= 25)
    = 1/2 + 24/99 – 1/2 * 24/99 = 0.62121...
    floor(1000 * (0.62121...)) = 621
11. SELECT * FROM R WHERE a = c
    We don't have information on c, so we just use the number of distinct values in a: Sel = 1/50
    1000 * (1/50) = 20
12. SELECT * FROM R, S WHERE R.a = S.a
    Sel = 1/max(50, 25) = 1/50
    (1000 * 500) * (1/50) = 10,000

For the rest of this worksheet we will try to optimize the following query:

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

We have 3 relations: R(a,b), S(b,c), and T(c,d).

- R has 1,000 data pages and 10,000 records
- S has 2,000 data pages and 40,000 records
- T has 3,000 data pages and 30,000 records

2 Single Table Access Plans

Assume we have the following indexes:

- Alt 2 unclustered index on R.a with 50 leaf pages
- Alt 2 clustered index on R.b with 100 leaf pages
- Alt 2 clustered indexes on S.b, T.c, and T.d (leaf page counts aren't relevant)

Also assume that it takes 2 IOs to reach the level above a leaf node and that no index or data pages are ever cached. All indexes have keys in the range [1, 100] with 100 distinct values.

1. How many IOs does a full scan on R take?
   1000 IOs; need to read every data page.

2. How many IOs does an index scan on R.a take?
   2 + 0.5(50) + 0.5(10000) = **5,027 IOs**. The selectivity of the R.a $\leq$ 50 condition is 0.5 so we multiply the leaf/data pages by 0.5. The index is unclustered so you have to do an IO for every record.

3. How many IOs does an index scan on R.b take?
   2 + 100 + 1000 = **1102 IOs**. This index is clustered so we only have to do an IO for every page. We can't apply the selectivity of the R.a $\leq$ 50 condition to the leaf/data pages because the index is built on a different column, so we're going to need to read all the pages.

4. How many pages from R will advance to the next stage for all of these access plans?
   500. The selectivity is 1/2 so only half the records will advance. (1/2) * (10000 records) = 5000 records. 5000 records / (10 records / page) = 500 pages.

Now assume that the other potential single table access plans have the following IO costs:

- Full scan on S: 2000 IOs
- Index scan on S.b: 2500 IOs
- Full scan on T: 3000 IOs
- Index scan on T.c: 3500 IOs
- Index scan on T.d: 3500 IOs

5. Which single table access plans advance to the next stage?
   Full scan on R (best overall plan for R so it must advance)
   Index scan on R.b (interesting order because R.b is used in a join)
   Full scan on S (best overall plan for S so it must advance)
   Index scan on S.b (interesting order because S.b is used in a join)
   Full scan on T (best overall plan for T so it must advance)
   Index scan on T.c (produces an interesting order on T.c so it must advance)

3 Multi-Table Plans

1. Assume we have 52 buffer pages. Remember, when the query optimizer calculates the cost of joining 2 tables for a given query, it must take into account the single table access plan for each table respectively. Consider the following joins:

a. R *BNLJ* $_{R.b=S.b}$ S

   i. Which single table access plans for R and S will minimize the cost of this join?
   Since this is a (block) nested loop join, the order of the tuples does not reduce the I/O cost. Therefore, the cheapest single table access plans will be chosen. This means a full scan will be used for both R and S, since those are the best overall plans that advanced from the last stage.

   ii. What is the I/O cost for performing this join?
   $1000 + \lceil \frac{500}{50} \rceil * 2000 = 21{,}000$
   We have an initial cost of 1000 I/Os to do a full scan on R, but since we push down the selection on R.a, only 500 pages of R will be passed along to the BNLJ operator. This means that the rest of the join will cost (500/50)(2000), since we scan all of S for each chunk from R of size 500/50 = 10.

b. R *SMJ* $_{R.b=S.b}$ S

   i. Which single table access plans for R and S will minimize the cost of this join?
   If the Sort Merge Join operator is passed the pages of R and S already sorted on the columns R.b and S.b respectively, then the sort part of SMJ can be avoided. Therefore, the query optimizer will perform index scans on R.b and S.b before performing the SMJ.

ii. What is the I/O cost for performing this join?

1102 + 2500 = 3,602

Since the cost of sorting can be avoided by using index scans on R.b and S.b, the only cost is to merge the records. The cost of merging will involve just scanning the records using the index scan, which gives a cost of 1102 I/Os for an index scan on R.b and 2500 I/Os for an index scan on S.b.

Assume all of the joins our database could do are as follows:

1. R BNLJ S: 3.a
2. R SMJ S: 3.b
3. S BNLJ R: 18,000 IOs
4. S SMJ R: 3,000 IOs
5. R BNLJ T: 30,000 IOs
6. R SMJ T: 40,000 IOs
7. T BNLJ R: 35,000 IOs
8. T SMJ R: 20,000 IOs
9. S BNLJ T: 15,000 IOs
10. S SMJ T: 10,000 IOs
11. T BNLJ S: 25,000 IOs
12. T SMJ S: 30,000 IOs

2. Which of these joins will actually be considered by the query optimizer on pass 2?

1, 2, 3, 4, 9, 10, 11, 12. We don't consider joins requiring a cross join, but every other join will be considered. R and T require a cross join because there is no join condition, so we will not consider any join involving R and T.

3. Which of these joins will advance to the next pass of the query optimizer?

4, 10. None of these joins produce an interesting order. This is because a Sort Merge Join is the only join that produces data in sorted order and it is sorted on the columns in the join condition. If we SMJ S and R the data will be sorted on b, but b isn't used in another join condition, ORDER BY, or GROUP BY clause. The same idea applies for SMJ S and T which produces the data in sorted order on c. Therefore only the best join for each considered set of tables is advanced which is 4 and 10.

4. Will any of these joins produce an interesting order?

No (see explanation above)

5. How could we modify the query so that the S SMJ R produces an interesting order?

S SMJ R will be sorted on column b so we need b to be interesting. We could add ORDER BY b, GROUP BY b, or another join condition involving R.b or S.b to the query.

6. Will the query plan: T BNLJ (S SMJ R) be considered by the final pass of the query optimizer?

No, this query plan is not left deep (all joined tables must be on the left side of all joins for it to be left deep) so it is not considered.