Database Systems WS 2023/24
Exercise 5: Distributed 20.11.2023, Due 27.11.2023 12:00 MEZ
Submitted by Erik Schwede, Suma Keerthi

1 Join-Size Estimation (1 P.)

1. Estimate the size of the join $R(a,b) \bowtie S(b,c)$ using histograms for $R.b$ and $S.b$. Assume $V(R,b) = V(S,b) = 30$ and the histograms for both attributes give the frequencies of the four most common values, as below, and further assume that every value appearing in the relation with the smaller set of values (R in this case) will also appear in the set of values of the other relation.

	0	1	2	3	others
$R.b$	10	8	7	11	39

	0	1	2	4	others
$S.b$	12	8	10	8	52

How does this estimate compare with the simpler estimate, assuming that all 30 values are equally likely to occur, with $T(R) = 75$ and $T(S) = 90$?

Solution: 0,1,2 appear in both the histograms. For b=0 10 tuples of R with b=0 join with 12 tuples of S having the same b-value leading to 120 tuples in the result For b=1 8 tuples of R with b=1 join with 8 tuples of S having the same b-value leading to 64 tuples in the result For b=2 7 tuples of R with b=2 join with 10 tuples of S having the same b-value leading to 70 tuples in the result

Was not able to solve the below two parts of 1a, on how to estimate values to calculate. The value of b=3 of R is not present in S and the value of b=4 of S is not present in R. Hence we have to estimate - Need explanation from this part onwards until the end of 1a.

How does this estimate compare with the simpler estimate, assuming that all 30 values are equally likely to occur, with $T(R) = 75$ and $T(S) = 90$?

2. Estimate the size of the natural join $R(a,b) \bowtie S(b,c)$ if we have the following histogram information. Give a lower and upper bound for the join size and explain under which circumstances they appear.

	$b < 0$	$b = 0$	$b > 0$
R	300	100	400
S	300	200	600

Solution: The 100 tuples of R with $b = 0$ join with the 200 tuples of S with same b-value as b=0 is appearing in both the histogram, leading to the result of 20000 tuples For the values b¡0 and b¿0, we don't know how many times b would appear in both the histograms. For b¡0 and b¿0, we might have distinct values.If both have distinct values, then we consider as 0 for both relations for the lower bound. Lower bound is: $20000 + 0 + 0 = 20000$
If b has same values in both R and S, we consider joining them with the respective values of R and S with $b < 0$ and $b > 0$ $b < 0$: 300 tuples of R with $b < 0$ join with 300 tuples of S with same b-value leading to 90000 tuples $b > 0$: 400 tuples of R with $b > 0$ join with 600 tuples of S with same b-value leading to 240000 tuples
Upper bound is : $20000 + 90000 + 240000 = 350000$

2 Join-Ordering: Dynamic Programming (1 P.)

1. Manually create the DP-table for the relations A,B,C with cardinalities $|A| = 100$, $|B| = 25$, $|C| = 80$ and selectivities $f_{A,C} = 0.05$, $f_{B,C} = 0.3$ with C_{out} as cost function. Cross products are allowed this time. Please keep the replaced entries in the table and highlight the final ones.

Solutions:

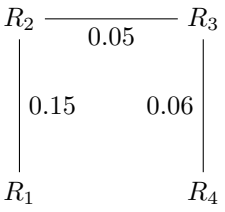
Submitted by Erik Schwede, Suma Keerthi

Relations	T	$\|T\|$	$C_{out}(T)$
$\{A\}$	A	100	0
$\{B\}$	B	25	0
$\{C\}$	C	80	80
$\{A, B\}$	$(A \times B)$	2500	2500
$\{A, C\}$	$(A \bowtie C)$	400	400
$\{B, C\}$	$(B \bowtie C)$	600	600
$\{A, B, C\}$	$(A \times B) \bowtie C$	3000	5500
$\{A, B, C\}$	$(A \bowtie C) \bowtie B$	3000	3400
$\{A, B, C\}$	$(B \bowtie C) \bowtie A$	3000	3600

2. Given the following DP-table with intermediate results and the query graph (with selectivities):

Relations	T	$\|T\|$	$C_{out}(T)$
$\{R_1\}$	R_1	40	0
$\{R_2\}$	R_2	10	0
$\{R_3\}$	R_3	20	0
$\{R_4\}$	R_4	30	0
$\{R_1, R_2\}$	$(R_1 \bowtie R_2)$	60	60
$\{R_1, R_3\}$	$(R_1 \times R_3)$	800	800
$\{R_1, R_4\}$	$(R_1 \times R_4)$	1200	1200
$\{R_2, R_3\}$	$(R_2 \bowtie R_3)$	10	10
$\{R_2, R_4\}$	$(R_2 \times R_4)$	300	300
$\{R_3, R_4\}$	$(R_3 \bowtie R_4)$	36	36
$\{R_1, R_2, R_3\}$	$((R_2 \bowtie R_3) \bowtie R_1)$	60	70
$\{R_1, R_2, R_4\}$	$((R_1 \bowtie R_2) \times R_4)$	1800	1860
$\{R_1, R_3, R_4\}$	$((R_3 \bowtie R_4) \times R_1)$	1440	1476
$\{R_2, R_3, R_4\}$	$((R_2 \bowtie R_3) \bowtie R_4)$	18	28

- $|R_1| = 40$
- $|R_2| = 10$
- $|R_3| = 20$
- $|R_4| = 30$



Calculate the optimal bushy join tree for the relations $\{R_1, R_2, R_3, R_4\}$ with the DP-algorithm shown in the lecture.

Solutions:

If all relations are joined, the $|T|$ will be equal for all possible join trees. Because the result will always be the same, only the way of computation changes when using a different join tree. That means, for all four relations, $|T|$ only needs to be calculated once.

Calculating $|T|$ with the relation $\{R_1, R_2, R_3\} \bowtie \{R_4\}$:
$|T| = 60 \cdot 30 \cdot 0.06$
$|T| = 108$

Cost calculations for all posible join trees:

$C_{out}(\{R_1, R_2, R_3\} \bowtie \{R_4\}) = 70 + 0 + 108 = 178$
$C_{out}(\{R_1, R_2, R_4\} \bowtie \{R_3\}) = 1860 + 0 + 108 = 1968$
$C_{out}(\{R_1, R_3, R_4\} \bowtie \{R_2\}) = 1476 + 0 + 108 = 1584$
$C_{out}(\{R_2, R_3, R_4\} \bowtie \{R_1\}) = 28 + 0 + 108 = 136$
$C_{out}(\{R_1, R_2\} \bowtie \{R_3, R_4\}) = 60 + 36 + 108 = 204$
$C_{out}(\{R_1, R_3\} \bowtie \{R_2, R_4\}) = 800 + 300 + 108 = 1208$
$C_{out}(\{R_1, R_4\} \bowtie \{R_2, R_3\}) = 1200 + 10 + 108 = 1318$

There are double the amount of possible join trees available, but because a join is commutative, they would result in the same cost. As visible from the calculations, the best solution for R_2, R_3, R_4 joined with the best solution of R_1 results in the join tree with the lowest cost. That can be written as:

$$(((R_2 \bowtie R_3) \bowtie R_4) \bowtie R_1)$$

$$C_{out}(((R_2 \bowtie R_3) \bowtie R_4) \bowtie R_1) = 136$$

3 Simplifying queries (1 P.)

Given the following queries from the uni_db schema from:

1.
```
1  SELECT DISTINCT p.name,
2   (SELECT MAX(position) FROM professors p2
3    WHERE p2.PID=p.PID)
4  FROM professors p, lectures l
5  WHERE
6   EXISTS
7     (SELECT *
8      FROM
9        (SELECT
10          e.matrnr,
11          (SELECT s.name FROM students s WHERE e.matrnr=s.matrnr),
12          LID
13        FROM exam e
14        ORDER BY e.grade LIMIT 2
15        ) t,
16       lectures l2
17     WHERE t.LID = l2.LID
18       AND l2.LID = l.LID
19    )
20  AND p.PID = l.heldby;
```

Solutions:

From line 2 to 3 the nested query is not needed. It looks for the maximum position for only one position value for each professors. Just adding the position value to the SELECT statement will result in the same query.

Before:

```
1   SELECT DISTINCT p.name,
2    (SELECT MAX(position) FROM professors p2
3     WHERE p2.PID=p.PID)
4   FROM professors p, lectures l
```

Optimised:

```
1   SELECT DISTINCT p.name, p.position
2   FROM professors p, lectures l
```

The following WHERE statement checks two conditions. If the professors holdes the lecture and there is a nested EXISTS check. Because EXISTS only checks if there is at least one element in the table it is not important what the elements of the table contain. Therefor the most inner The most inner sub query is not nesasary and also not the matrnr. Only LID is needed for the following WHERE check.
Before:

```
      (SELECT
         e.matrnr,
         (SELECT s.name FROM students s WHERE e.matrnr=s.matrnr),
         LID
       FROM exam e
       ORDER BY e.grade LIMIT 2
       )
```

Optimised:

```
      (SELECT e.LID
       FROM exam e
       ORDER BY e.grade LIMIT 2)
```

Also the check of the following WHERE clause can be simplified as the following.
Before:

```
  WHERE t.LID = l2.LID
      AND l2.LID = l.LID
```

Optimised:

```
  WHERE t.LID = l.LID
```

Also, the nested SELECT * query is completely unnecessary because it just takes the exact value of the inner nested query.
Taking all these factors into account, our query looks like the following:

```
  SELECT DISTINCT p.name, p.position
  FROM professors p, lectures l
  WHERE
   EXISTS
      (SELECT e.LID
      FROM exam e
      WHERE e.LID = l.LID
      ORDER BY e.grade LIMIT 2)
   AND p.PID = l.heldby;
```

The only nested query that still remains is only dependent on the lecture relation and not on the professors relation. Instead of EXISTS, IN can be used in this case.
The final result looks like the following:

```
1   SELECT DISTINCT p.name, p.position
2   FROM professors p, lectures l
3   WHERE
4     l.LID IN
5       (SELECT LID
6       FROM exam
7       ORDER BY grade LIMIT 2)
8    AND p.PID = l.heldby;
```

When looking at the query plans, the cost is noticeably lower than the cost of the original query. The main reason for this is that the optimized query has to execute one join less than the original query.

2.

```
1 SELECT
2  s2.name,
3 (WITH RECURSIVE t(LID) AS (
4    SELECT exam.LID FROM students, exam
5    WHERE students.matrnr = exam.matrnr AND students.matrnr = s2.
     matrnr
6      AND exam.grade = (SELECT min(grade) FROM exam)
7  UNION ALL
8    SELECT prerequisites.required FROM t, prerequisites WHERE t.LID =
      prerequisites.lecture
9 )
10 SELECT count(*) FROM t) x
11 FROM
12 students s2;
```

Solution: In order to optimize the query, we have to make the inner WITH RECURSIVE query independent from the outer query. So, we only have to execute this inner query once. So far, it is dependent on the outer table s2. It is only comparing the matrnr of the table s2 with the exam matrnr. If the WITH RECURSIVE table t also has the matrnr as a parameter, we can perform a join later to avoid a nested query.
The optimized query looks like the following:

```
WITH RECURSIVE t(matrnr, LID) AS
(
    SELECT exam.matrnr, exam.LID FROM exam
    WHERE exam.grade = (SELECT min(grade) FROM exam)
  UNION ALL
    SELECT prerequisites.required FROM t, prerequisites
    WHERE t.LID = prerequisites.lecture
)

SELECT
 s2.name,
 x.amount
FROM
students s2 LEFT JOIN (SELECT count(*) amount, matrnr FROM t) x ON
  (s.matrnr = x.matrnr);
```

It is important to mention that the optimized query is returning null instead of 0 for the count column. When analyzing the query plan, the calculated cost for the optimized plan is lower than the cost for the original plan because the table t is not executed for each element of students but only once.

4 Correctness of Unnesting (1 P.)

Provide unnested queries for the given nested queries. Show through an example, by specifying contents of tables and corresponding results, why the type of join (e.g,. INNER, LEFT OUTER) is important when unnesting a certain query. Considering the given queries, will the change in the type of the join impact the correctness of the query?

1.
```
SELECT DISTINCT P.playerId
FROM Player P
WHERE (
    SELECT COUNT(G.id)
    FROM Game G
    WHERE G.playerId = P.playerId
  ) >10
```

Solution:
The query defines us to return the players ids of the players who play more than 10 games. Hence we use a regular join to un-nest the query. If we consider the inner query, it would return the count of all the game ids played by all players, when it has to compare, it has to go through every record of the inner query output to search for the values greater than 10 which is costly.

It is a dependent join as the inner query depends on the outer query.

Hence we create a regular join, which counts the game ids from the game table, the output of the common key (playerID) between the players and game table would be the number of games played by a particular player.

If we group them with the player id, we get the games played by each player and then we filter the output of the group by column using aggregate condition of count ¿ 10.

Group by P.playerID returns the unique player id details, hence we do not need DISTINCT in the select query.

```
SELECT P.playerId
FROM Player P, Game G,
WHERE P.playerID = G.playerId
GROUP by P.playerId
HAVING count(G.ID)>10
```

2.
```
SELECT DISTINCT P.name, (SELECT COUNT(*)
FROM Game G
WHERE P.playerId = G.playerId)
FROM Player P
```

Solution:
This query defines us to return all the players who play games. It is also a dependent join as the inner query is dependent on the outer query.

If we just join the column with the regular join, it might miss out on the count of the players who did not play any game as it would not find any match in the where condition. Hence we use LEFT OUTER JOIN here to join both tables here.

Group by P.name returns the unique player id details, hence we do not need DISTINCT in the select query.

```
SELECT P.name, Count(G.Id)
FROM Player P LEFT OUTER JOIN Game G,
WHERE P.playerID = G.playerId
GROUP by P.name
```