1 Plan Enumeration (6 Points)

Consider the following SQL query.

```
SELECT A.f, B.x, C.w, D.e
FROM   A, B, C, D
WHERE  A.g = C.d AND C.n = D.d AND A.s = B.q AND B.o = C.r;
```

Assume the following result sizes.

subproblem	result size
{A}	50
{B}	95
{C}	20
{D}	65
{A,B}	40
{A,C}	70
{A,D}	3,250
{B,C}	100
{B,D}	6,175
{C,D}	180
{A,B,C}	40
{A,B,D}	2,600
{A,C,D}	300
{B,C,D}	543
{A,B,C,D}	8

In the table above, each subproblem represents all possible join orders given by the corresponding relations. For example, subproblem {A,B,C} describes all join orders for the three relations A, B, and C, regardless of whether a join predicate exists between the individual tables.

In this task, you have only a simple hash-based join with the following cost function available.

$$C_{\text{HashJoin}}(R \bowtie S) = |R| + |S|$$

Note that this cost function only applies to joins. In case of a Cartesian product, the costs are as follows.

$$C_{\text{CP}}(R \times S) = |R| \cdot |S|$$

(a) Determine the optimal join order using only "left-deep" plans. Complete and extend the following table with subplans of size two, three and four:

subplan	costs	result size
A	0	50
B	0	95
C	0	20
D	0	65
A $\bowtie$ B		
A $\bowtie$ C		
A $\times$ D		
B $\bowtie$ C		
B $\times$ D		
C $\bowtie$ D		

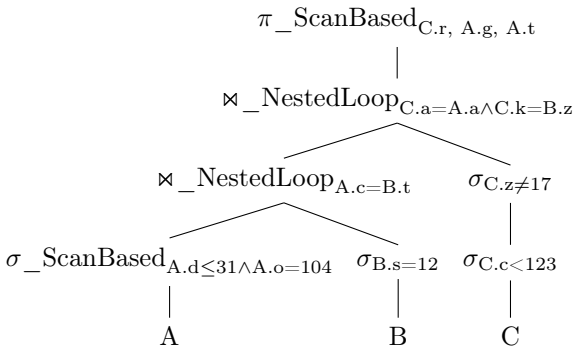
Make sure that the costs of a join are made up of the costs of calculating the inputs and of executing the join. Since the cost functions are both symmetric, the order of the inputs is irrelevant. For example, A $\bowtie$ B and B $\bowtie$ A do not have to be listed separately above. (4 Points)

(b) Draw the join graph for the given query. Label the edges with the respective join predicates. (1 Point)

(c) How could you have used the join graph to exclude certain entries in the table from Task (a)? (1 Point)

2 Cost-based Optimisation (5 Points)

Given is the following "hybrid" query tree, which has been optimised heuristically (additional projections are omitted here for the sake of simplicity). Some logical operators have already been replaced by physical ones. Your task is now to translate the remaining logical operators into physical ones as well.



Indexes exist on the attributes B.s, C.c, and C.z. For a predicate p with selectivity $sel(p)$ on table T, the following costs apply for a full scan or index access.

$$scan(T) = |\ T\ |$$
$$index(T,p) = log_2(|\ T\ |) + 42 \cdot sel(p) \cdot |\ T\ |$$

(a) Compute the most cost-effective physical plan for the following table sizes and selectivities and draw the corresponding query tree (with physical operators).

1. $|\ \text{B}\ | = 300,000$ $\quad sel(\text{B.s=12}) = 0.02$
   $|\ \text{C}\ | = 50,000$ $\quad sel(\text{C.z}\neq 17) = 0.1$ $\quad sel(\text{C.c<123}) = 0.05$
2. $|\ \text{B}\ | = 70,000$ $\quad sel(\text{B.s=12}) = 0.04$
   $|\ \text{C}\ | = 150,000$ $\quad sel(\text{C.z}\neq 17) = 0.015$ $\quad sel(\text{C.c<123}) = 0.075$

Proceed as follows.

(i) For each of the three selections for which an index exists, use the cost functions to determine whether to use a scan or an index access.

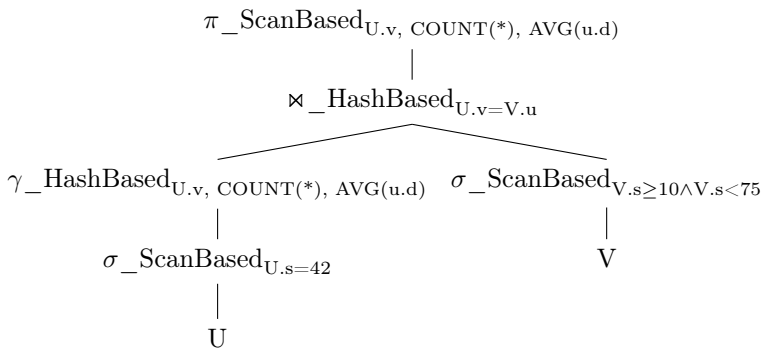
(ii) Replace the remaining logical operators with physical operators. You do not have to redraw the complete query tree, but make it clear which logical operator is replaced by which physical operator.

Note that an index only exists on tables, not on intermediate results. For example, if an index access to C.z≠17 is used, the order of the two predicates on M must be swapped. In addition, there can only be one index access per table. Furthermore, multiple predicates can be evaluated in one scan without additional costs. In particular, this means that C.z≠17 ∧ C.c<123 can be evaluated with one scan. Since we have not discussed any physical data layouts, we assume that scans on an intermediate result created by an index access do not incur any additional costs. (4 Points)

(b) At what selectivity is it worth using the index? State the selectivity as a function of the table size and interpret your result, i.e. look at how the function behaves for arbitrarily large table sizes. (1 Point)

3 Code Compilation (4 Points)

Consider the following physical plan.



Translate this plan into pseudocode. Use slide 38 from the lecture Query Optimisation (Part 2) as a reference. Produce as few intermediate results as possible.

4 Plan Enumeration Without Cartesian Products (5 Points)

In the notebook Plan Enumeration.ipynb we have already seen how we can, for a given cost function, determine the cheapest plan by enumerating all plan alternatives for a query.

(a) Implement the `find_cheapest_plan_with_pruning()` function, which, unlike `find_cheapest_plan_exhaustive()`, ignores plans with a Cartesian product. To do this, use the join graph passed as a parameter to decide whether a join between two relations or partial results exists. Your function should only consider "left-deep" plans. (4 Points)

Note: Our reference implementation takes less than a second for the complex example of the IMDb database. Your implementation should be in the same order of magnitude.

(b) How can you extend `find_cheapest_plan_with_pruning()` to narrow the search space of possible plans even further, under the condition *not to calculate a worse plan*? (1 Point)

Submission

Solutions must be submitted in teams of 3 to 4 students by June, 20 2024, 10:00 a.m. via your personal status page in CMS using the Team Groupings functionality. Late submissions will not be graded!

Please note that there are two submissions, under **Theoratical** you submit your solutions to exercises 1, 2 and 3 as a PDF file.

Your solution to exercise 4 must be handed in under **Practical** as txt file.

Make sure that you only copy the complete Jupyter cells you want to add and that indentation and formatting are correct.