CSC 370

Activity Worksheet:
Logical Query Plans

Dr. Sean Chester

Fall 2022

Notes

In this worksheet, you will practice generating logical query plans from SQL code. The questions are split into two sections: one in which you convert SQL queries into relational algebra parse trees and one in which you generate alternative query plans from a give one. The first question of each section has been answered already as an exemplar.

Questions

1. Converting SQL to Logical Query Plans: Generate a logical query plan as a relational algebra parse tree that corresponds to the SQL query below

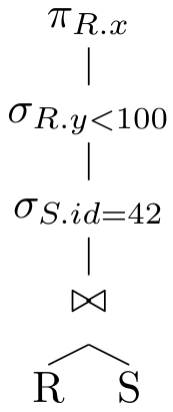
```
SELECT 'R'.'x'
FROM 'R'
  NATURAL JOIN 'S'
WHERE 'R'.'y' < 100
  AND 'S'.'id' = 42;
```

Solution:

We first need to convert this query to relational algebra. For that, we can use the systematic heuristic of nesting joins innermost, followed by selection predicates, and finally using projection in the outermost level of nesting:

$$\pi_{R.x}(\sigma_{R.y<100}(\sigma_{S.id=42}(R \bowtie S)))$$

Then we can parse this into a tree in which the relations are the leaves, the operators are the interior nodes, and the operands are the children of each node (just like a typical algebraic parse tree):



2. Converting SQL to Logical Query Plans: Generate a logical query plan as a relational algebra parse tree that corresponds to the SQL query below

```
SELECT `Person`.`name`, COUNT(*) AS `NameConflicts`
FROM `Student`
  NATURAL JOIN `Person`
GROUP BY `Person`.`name`
HAVING `NameConflicts` > 1;
```

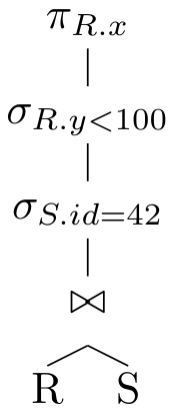
Solution:

3. Converting SQL to Logical Query Plans: Generate a logical query plan as a relational algebra parse tree that corresponds to the SQL query below

```
SELECT `Student`.`name`
FROM `Student`
   JOIN `Department`
     ON `Student`.`major` = `Department`.`code`
WHERE `Department`.`Faculty` <> 'Continuing Studies'
   AND `Student`.`gpa` > (
     SELECT 2 * AVERAGE(`S2`.`gpa`)
     FROM `Student`
```

Solution:

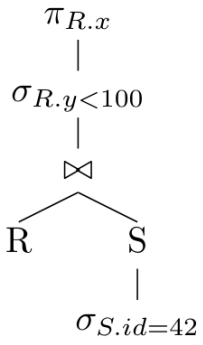
4. Generating Alternative Plans: You are given the logical query plan below. List all equivalent logical query plans in which a node has a non-redundant effect.

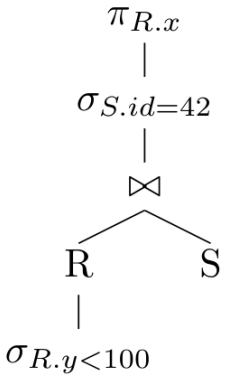


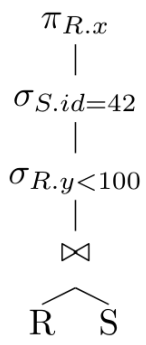
Solution: We can:

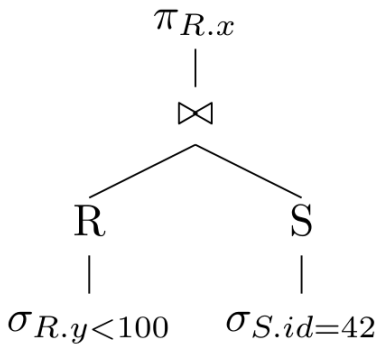
- rearrange the selections
- combine the selections
- push the selections through the join
- add the selections as a condition on the join (if we knew here the attributes used in the natural join)

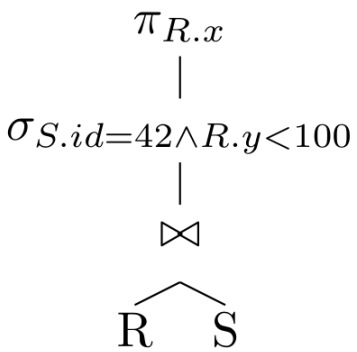
This produces five different possibilities.



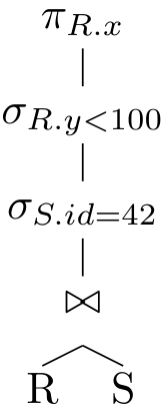








5. Generating Alternative Plans: You are given the logical query plan below. List all equivalent logical query plans in which a node has a non-redundant effect.



Solution:

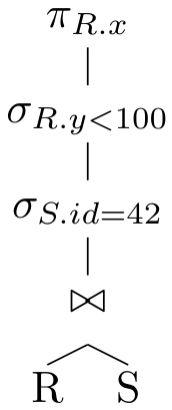
Solutions

Question 1

We first need to convert this query to relational algebra. For that, we can use the systematic heuristic of nesting joins innermost, followed by selection predicates, and finally using projection in the outermost level of nesting:

$$\pi_{R.x}(\sigma_{R.y<100}(\sigma_{S.id=42}(R \bowtie S)))$$

Then we can parse this into a tree in which the relations are the leaves, the operators are the interior nodes, and the operands are the children of each node (just like a typical algebraic parse tree):

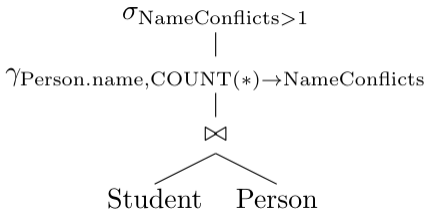


Question 2

We first need to convert this query to relational algebra. For that, we can use the systematic heuristic of nesting grouping and aggregation operators after the tuple-oriented operators but before the projection:

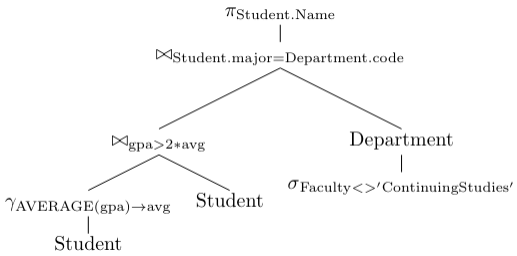
$$\sigma_{NameConflicts>1}(\gamma_{Person.name,COUNT(*)\rightarrow NameConflicts}(Student \bowtie Person))$$

This produces the following tree:



Question 3

For simplicity, we will jump straight to the parse tree. Here, we note that a sub-query is simply a sub-tree:

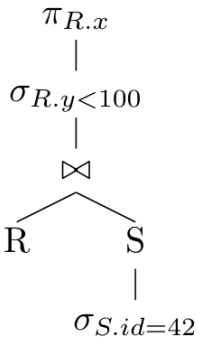


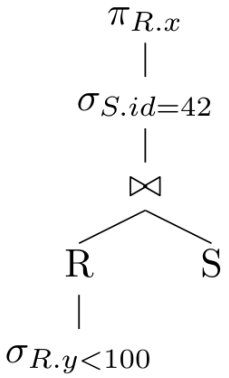
Question 4

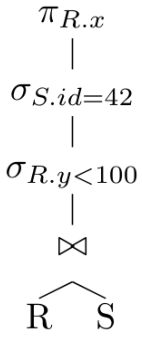
We can:

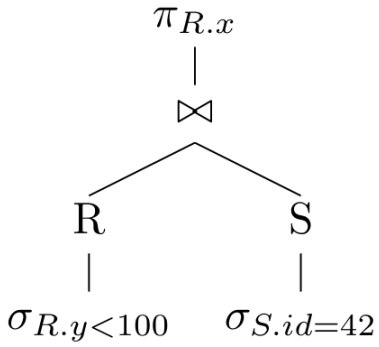
- rearrange the selections
- combine the selections
- push the selections through the join
- add the selections as a condition on the join (if we knew here the attributes used in the natural join)

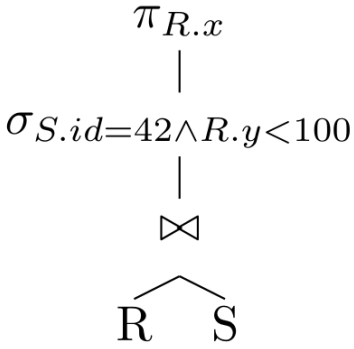
This produces five different possibilities.











Question 5

We can:

- push the projection through the selection, since the selection only needs attribute x
- push the projection through the join, since the join predicate only requires attribute x from R
- push the selection through the join
- push the duplicate elimination operator, but not the projection nor the join

This produces seven different possibilities. They are not drawn/typeset in this solution deck.