1 From SQL to the Logical Plan (5 Points)

In this exercise, we want to optimise a SQL query based on the following schemas using the rules presented in the lecture.

[persons] : {[pid: int, name: varchar, birth_year: int]}
[libraries] : {[lid: int, city: varchar]}
[members] : {[mid: (persons), favorite_author: (authors), late_fees: int]}
[membership] : {[member: (members), library: (libraries)]}
[authors] : {[aid: (persons), salary: int]}
[books] : {[bid: int, author: (authors), title: varchar, year: int, genre: varchar]}
[borrow] : {[member: (members), library: (libraries), book: (books), borrow_date: date, due_date: date]}
[reserve] : {[member: (members), library: (libraries), book: (books), reservation_date: date]}

```
SELECT favorite_author, reservation_date, year
FROM   books, reserve, members
WHERE  book = bid
       AND member = mid
       AND late_fees > 20
       AND title = 'The Da Vinci Code'
       AND reservation_date = '24.02.2024';
```

(a) Translate the SQL query canonically into a relational algebra expression. Use only projections, selections and Cartesian products. (1 Point)

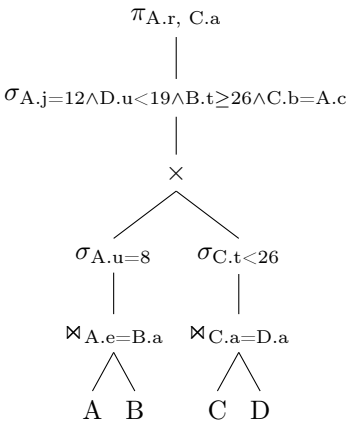
(b) Draw the logical plan of the query as a tree. (1 Point)

(c) Apply the rules for heuristic query optimisation known from the lecture and the notebook Rule-based Optimization.ipynb and draw the optimised logical plan of the query as a tree. (3 Points)

2 Heuristic Query Optimisation (7 Points)

In this exercise, you will again optimise logical plans based on the heuristic rules from the lecture and the notebook Rule-based Optimization.ipynb.

(a) The following logical plan has already been partially optimised. Completely optimise the plan using the known rules and draw it as a query tree. Assume that the following plan does not necessarily contain all attributes of the underlying relational schemas. (2 Points)



(b) The following relational algebra expression is to be translated step by step into an optimised logical plan. Assume that the following expression does not necessarily contain all attributes of the underlying relational schemas.

$$\pi_{\text{Q.m,R.o,S.a}}\left(\sigma_{\text{R.t=S.b}\wedge\text{S.t}\leq\text{33}\wedge\text{S.u=T.s}\wedge\text{S.u}\geq\text{27}\wedge\text{R.e=18}\wedge\text{Q.s=R.e}\wedge\text{T.b}\geq\text{9}}\left((\text{Q}\times\text{R})\times(\text{S}\times\text{T})\right)\right)$$

1. Convert the expression into a logical plan. (1 Point)
2. Optimise the logical plan with the known rules. (2 Points)
3. The logical plan can be further optimised. To do this, analyse the predicates occurring in the query. Explain the optimisation and draw the resulting logical plan. (2 Points)

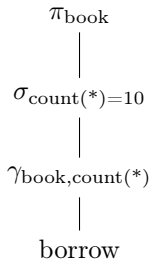
3 SQL vs Relational Algebra Semantics (3 Points)

Consider the following SQL query based on the schemas below.

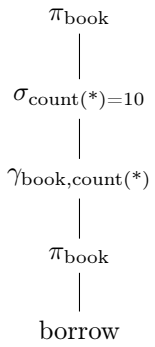
[persons] : {[pid: int, name: varchar, birth_year: int]}
[libraries] : {[lid: int, city: varchar]}
[members] : {[mid: (persons), favorite_author: (authors), late_fees: int]}
[membership] : {[member: (members), library: (libraries)]}
[authors] : {[aid: (persons), salary: int]}
[books] : {[bid: int, author: (authors), title: varchar, year: int, genre: varchar]}
[borrow] : {[member: (members), library: (libraries), book: (books), borrow_date: date, due_date: date]}
[reserve] : {[member: (members), library: (libraries), book: (books), reservation_date: date]}

```
SELECT   book
FROM     borrow
GROUP BY book
HAVING   COUNT(*) = 10
```

Figure 1a shows the canonical translation from the SQL query into a logical plan, while Figure 1b shows the logical plan after applying heuristic optimisation. Here, only projection pushdown was applied.



(a) Logical plan after canonical translation.



(b) Logical plan after applying heuristic optimisation.

Figure 1: Logical plans for the given SQL query.

1. Assuming the set semantics from relational algebra, what problem arises from the optimisation above? (2 Points)
2. How could you fix this problem? (1 Point)

4 Recursive Inverted Index (5 Points)

In this exercise, you will implement an inverted index. An inverted index is a mapping from the content (e.g. words) of a document to the document itself. For example, consider the following two documents containing one sentence each:

1. I learned a lot in the course BDE.
2. I learned even more in the course DBSys.

A possible inverted index for these two documents looks as follows:
{'I': {1, 2}, 'learned': {1, 2}, 'a': {1}, 'lot': {1}, 'in': {1, 2}, 'the': {1, 2}, 'course': {1, 2}, 'BDE': {1}, 'even': {2}, 'more': {2}, 'DBSys': {2}}

An inverted index maps each unique word to a list of documents containing the word. In addition, in this exercise we radix-partition the words, i.e. we *recursively* partition the words characterwise similar to the notebook Indexing by Recursive External Partitioning.ipynb.

This structure allows us to easily search for documents that contain a word that exactly matches a given search word, but also words that start with the given search word (prefix search). Implement the following functions of the class `RecursiveInvertedIndex` in the attached notebook:

- `__init__`: This is the constructor of the class, which takes a list of document names and creates the recursive, inverted index based on these documents. Your words must be radix-partitioned. You can use the helper function `get_words_from_document(document_name)`, which takes a document name as input and then returns a set of words contained in the document. Note, that all words are converted to lowercase.
- `search(word)`: This function uses the inverted index to return a list of all matching documents given the key `word`. Matching here means that the search word either exactly matches one of the words in a document, or at least one of the words in a document starts with the search word. If no documents exist for a search word, an empty set is returned. Otherwise, the set of all matching documents is returned. Your implementation should be case-insensitive.

Your implementation must pass all provided unit tests without hardcoding!

Submission

Solutions must be submitted in teams of 3 to 4 students by June, 13 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.