Assignment: Extended Relational Algebra

Instructions

- This assignment is an **individual assignment**.
- Submit your assignment by **Sunday, 15 February 2026, 23:59** to Canvas.
- Download the following files from Canvas.
  - "`A01.py`" (template file).
  - "`Operations.py`" containing the actual relation algebra operations.
  - "`Parser.py`" containing the experimental parser.
  - "`Syntax.py`" containing the classes related to relational algebra operators.
- Download the "Database" directory from Canvas.
- Download the case study "`AppStore`" from Canvas.
- Submit the two functions and three expression in "`A01.py`" to Canvas.
- Indicate your matric number in the space provided.
- Do not modify other files besides "`A01.py`".
- Do not modify the template, only fill in the required component. This includes the comments in the template.
- Your solution should use only the default library. You are not allowed to add other imports.

Preliminary

Relational algebra is an *imperative language* to model query language introduced by Edgar F. Codd [1]. The basic operations in relational algebra are as follows.

Unary Operations

- Selection: $\sigma_{[\theta]}(R)$
- Projection: $\pi_{[L]}(R)$
- Renaming: $\rho(R_{\text{old}}, R_{\text{new}})$

Binary Operations

- Cross Product: $R_1 \times R_2$
- Union: $R_1 \cup R_2$
- Intersection: $R_1 \cap R_2$
- Difference: $R_1 - R_2$

To help you, we have made the following changes and/or additions.

- The algebra is using multi-set semantics.
- Addition of *deduplication* operation $\delta(R)$ called `dedup(rel)`. It removes duplicates from `rel` but the ordering is not guaranteed to be preserved. The operation uses hash (i.e., Python's dictionary).
- Addition of *grouping* operation $\gamma_{[\text{attrs}][\text{aggrs}]}(R)$ called `group(attrs, aggrs, rel)`. The operation is as follows.

  1. Group the relation `rel` such that rows with equal values in the list of attributes `attrs` are in the same group.
  2. Perform the aggregation functions in the list `aggrs`.
  3. The result is a table with attributes in `attrs` followed by the results from aggregation functions in `aggrs`.

  The operation $\gamma_{[\text{a, b, c}][\texttt{MAX}(\text{d}),\ \texttt{COUNT}(\text{e})]}(R)$ is equivalent to the SQL query below.

  Code: Group By

  ```
  SELECT a, b, c, MAX(d), COUNT(e)
  FROM R
  GROUP BY a, b, c;
  ```

- We have added the necessary parsing for the required operators to be implemented in this assignment. However, the parser is *experimental* and may have an error. Rest assured that any error will not impact your grading.

Please read the Python files to understand the available functions. The functions are similar to the functions shown in the lecture.

Recap that multi-set allows duplicates. So the deduplication operation has to be performed explicitly when needed. Additionally, order does not matter. We will accept your output even if the order of the rows in the resulting table is different.

Our aim in this assignment is to extend the basic relational algebra into an extended relational algebra. At the end, we will try to run an SQL query as relational algebra. However, you are responsible to be the query translator and optimizer.

Questions

For our running example, we will be using the following two tables named R and S respectively.

Table R

A	B	C
1	x	a
1	y	a
2	x	a
2	y	a
3	x	a
1	z	a
4	w	a

Table S

B
x
y

1. (4 points) **Extended Operations**.

   (a) (2 points) One operation our relational algebra is currently missing is *outer join*. We will focus on **left outer join**. Consider the following left outer join query.

Code: Left Outer Join

```
SELECT *
FROM R LEFT OUTER JOIN S
  ON Cond;
```

The equivalent relational algebra expression written in our convention is as follows.

$$R \text{⟕}_{[\text{Cond}]} S$$

The expected result of the expression $R \text{⟕}_{[R.B=S.B]} S$ is the following table.

Table $R \text{⟕}_{[R.B=S.B]} S$

R.A	R.B	R.C	S.B
1	x	a	x
1	y	a	y
2	x	a	x
2	y	a	y
3	x	a	x
1	z	a	*NULL*
4	w	a	*NULL*

If we look at the table closely, the first 4 rows comes from the *inner join* $R \text{⋈}_{[R.B=S.B]} S$ or equivalently $\sigma_{[R.B=S.B]}(R \times S)$. However, the last 3 rows come from the *dangling rows* (i.e., rows from R that do not match any rows from S based on the condition $R.B = S.B$).

These dangling rows are what we want to produce. If we can produce them, then we can construct left outer join. We will define the following operator ▷ such that $R \triangleright_{[R.B=S.B]} S$ produces exactly the last 3 rows above.

We call this operator as **left anti join**. Note that there may be a different definition of left anti join such that the *null padding* is not performed. We will refer to those as *left anti semi join* instead. In our left anti join, the null padding is added.

We can summarize the relationship between left outer join and left anti join as follows.

$$R \bowtie_{[\text{Cond}]} S \quad \equiv \quad \delta(\sigma_{[R.B=S.B]}(R \times S)) \cup (R \triangleright_{[R.B=S.B]} S)$$

Implement the function `anti(rel1, rel2, cond)` to compute `rel1` $\triangleright_{[\texttt{cond}]}$ `rel2`.

Code: Template

```
def anti(rel1, rel2, cond):
  data = []
  attr = rel1.attr + rel2.attr

  # rel1 (left-anti-join)[cond] rel2

  return Rel(f'({rel1.name} >[{cond}] {rel2.name})', attr, data)
```

(b) (2 points) Another useful operator to have is called the *division* operator. We can treat division as the *inverse* of multiplication. Since the analogue of multiplication in relational algebra is the cross product, it is an operator that hope to *undo* the effect of cross product.

For discrete data (e.g., integer), division often produce remainder. In integer division, we drop the remainder (i.e., $7 \div 2 = 3$ dropping the remainder of 1). Similarly, in relational algebra, we will drop the remainder.

Consider the tables R and S again. To see what $R \div S$ is doing, imagine that there is a table T such that $T = R \div S$. The main property we wish to obtain is $T \times S = R$. However, because our current relational algebra is using multi-set semantics, it is closer to $\delta(T \times S) = R$ (i.e., equivalent after deduplication).

Unfortunately, due to the remainder, $\delta(T \times S) \neq R$. But there is an interesting property here. Note that the table T contains the rows such that it is related to **all** elements of table S. This relationship between T and S are recorded as a row in R.

Table R

A	B	C
1	x	a
1	y	a
2	x	a
2	y	a
3	x	a
1	z	a
4	w	a

Table S

B
x
y

Table $T = R \div S$

A	C
1	a
2	a

You can check that both `(1,a)` and `(2,a)` are related to both `x` and `y` in the table R. In particular, those are the top 4 rows. Note that this also include row 6 after projection to attributes A and C and deduplication. As for row 5 and row 7, they are not included because of the following.

- Row 5: `(3,a)` is missing a relation with `y`. In other words, `(3, y, a)` is missing.
- Row 7: `(4,a)` is missing a relation with both `x` and `y`.

With a quick Google search, you can probably find the following definition. While you are allowed to use such definition, you should try implementing the division operation

directly. In particular, note that $\pi_{[R-S]}$ refers to removing the attributes of relation S from the attributes of relation R. Similarly, $\pi_{[R]}$ refers to projecting the result to only attributes in R.

$$R \div S \quad \equiv \quad \pi_{[R-S]}(R) - (\pi_{[R-S]}(\pi_{[R]}(\pi_{[R-S]}(R) \times S) - R))$$

Implement the function `div(rel1, rel2)` to compute `rel1` ÷ `rel2`.

Code: Template

```
def div(rel1, rel2):
  data = []
  attr = []

  # rel1 (div) rel2

  return Rel(f'({rel1.name} \ {rel2.name})', attr, data)
```

2. (6 points) **Query (Re)-Writing**.

   You are given the following query.

Code: Universal Quantification

```
SELECT c.first_name, c.last_name
FROM customers c
WHERE NOT EXISTS (
  SELECT *
  FROM games g
  WHERE g.name = 'Aerified'
    AND NOT EXISTS (
      SELECT *
      FROM downloads d
      WHERE c.customerid = d.customerid
        AND g.name = d.name
        AND g.version = d.version
    )
);
```

Your main task is to write 3 different relational algebra expression that produces the same result as below. We ignore the order of rows but the order and the number of columns must follow the sample output below.

first_name	last_name
Helen	Cook
Amy	Porter
Barbara	Tucker

(a) (2 points) Write a relational algebra expression equivalent to the SQL query above. You must use the **division** operator. You are not allowed to use the *set difference* and *left anti join* (or left outer join) operator.

(b) (2 points) Write a relational algebra expression equivalent to the SQL query above. You must use the **set difference** operator. You are not allowed to use the *division* and *left anti join* (or left outer join) operator.

(c) (2 points) Write a relational algebra expression equivalent to the SQL query above. You must use the **left anti join** operator. You are not allowed to use the *division* and *set difference* operator.

References

[1] E. F. Codd. "A relational model of data for large shared data banks". In: *Commun. ACM* 13.6 (June 1970), pp. 377–387. ISSN: 0001-0782. DOI: 10.1145/362384.362685. URL: https://doi.org/10.1145/362384.362685.