Introduction to Databases

Relational Algebra

under set semantics

Dr Paolo Guagliardo

(v26.0)

CHANGELOG

v26.0 Initial version.

Problem 1. Given the database below:

CUSTOMER

ID	Name	City
1	John	London
2	Mary	Edinburgh
3	Jeff	London
4	Jane	Cardiff

ACCOUNT

Number	Branch	CustID	Balance
111	London	1	120
222	Edinburgh	1	62
333	London	3	76
444	London	2	200

compute the answer to the following query (without rewriting it):

$$\text{CUSTOMER} \bowtie \left(\pi_{\text{ID,City}}(\text{CUSTOMER}) \cap \rho_{\text{CustID}\to\text{ID},\ \text{Branch}\to\text{City}}\left(\pi_{\text{Branch,CustID}}(\text{ACCOUNT})\right)\right)$$

To this end, first work out all intermediate results on paper and by hand; then verify them, and the final answer, using the Relational Algebra interpreter `real`[1] (see usage instructions) by appropriately rewriting the natural join ($\bowtie$) operation.

Solution. The projections and renaming are straightforward:

$\pi_{\text{Branch,CustID}}$

Number	Branch	CustID	Balance
111	London	1	120
222	Edinburgh	1	62
333	London	3	76
444	London	2	200

$=$

Branch	CustID
London	1
Edinburgh	1
London	3
London	2

$\rho_{\text{CustID}\to\text{ID},\ \text{Branch}\to\text{City}}$

Branch	CustID
London	1
Edinburgh	1
London	3
London	2

$=$

City	ID
London	1
Edinburgh	1
London	3
London	2

[1] https://git.ecdf.ed.ac.uk/pguaglia/real

$\pi_{\text{ID,City}}$

ID	Name	City
1	John	London
2	Mary	Edinburgh
3	Jeff	London
4	Jane	Cardiff

$=$

ID	City
1	London
2	Edinburgh
3	London
4	Cardiff

Intersection takes the rows that appear in *both* input relations; recall that columns/attributes are not ordered in the data model we use for RA:

City	ID
London	1
Edinburgh	1
London	3
London	2

$\cap$

ID	City
1	London
2	Edinburgh
3	London
4	Cardiff

$=$

City	ID
London	1
London	3

Natural join $\bowtie$ joins the input tables on their common attributes:

ID	Name	City
1	John	London
2	Mary	Edinburgh
3	Jeff	London
4	Jane	Cardiff

$\bowtie$

City	ID
London	1
London	3

$=$

ID	Name	City
1	John	London
3	Jeff	London

The answer to the query on the given database is:

ID	Name	City
1	John	London
3	Jeff	London

In `real`, the query would be written as:

```
<P>[ID,Name,City](
  <S>[ID=ID1 & City=City1](
    <R>[ID->ID1,City->City1](Customer)
    <X>
    ( <P>[ID,City](Customer)
      <I>
      <R>[CustID->ID,Branch->City]( <P>[Branch,CustID](Account) ))));
```

Problem 2. Using the definition of natural join and intersection, rewrite the query of Problem 1 into an equivalent expression that uses only primitive operations.

Solution. Let $E_1 = \pi_{\text{ID,City}}(\textsc{Customer})$ and $E_2 = \rho_{\text{CustID}\to\text{ID, Branch}\to\text{City}}\left(\pi_{\text{Branch,CustID}}(\textsc{Account})\right)$. We have already seen at the end of Problem 1, using `real`'s syntax, how to rewrite $\bowtie$. In mathematical syntax, we have:

$$\pi_{\text{ID,Name,City}}\left(\sigma_{\text{ID=ID1}\wedge\text{City=City1}}\left(\rho_{\text{ID}\to\text{ID1, City}\to\text{City1}}(\textsc{Customer}) \times (E_1 \cap E_2)\right)\right)$$

Now, the only thing left to do is to rewrite $\cap$ in terms of difference:

$$\pi_{\text{ID,Name,City}}\left(\sigma_{\text{ID=ID1}\wedge\text{City=City1}}\left(\rho_{\text{ID}\to\text{ID1, City}\to\text{City1}}(\textsc{Customer}) \times (E_1 - (E_1 - E_2))\right)\right)$$

We can achieve the same in `real` by defining the following "views" (i.e., *named* expressions):

```
E1 := <P>[ID,City]( Customer );
E2 := <R>[CustID->ID,Branch->City]( <P>[Branch,CustID]( Account ));
```

and then writing the expression below:

```
<P>[ID,Name,City](
  <S>[ID=ID1 & City=City1](
    <R>[ID->ID1,City->City1](Customer)
    <X>
    ( E1 <D> (E1 <D> E2) )));
```

Replacing E_1 (resp., `E1`) and E_2 (resp., `E2`) with the corresponding expressions, we obtain the final query.

Problem 3. Write the following queries in relational algebra:

(1) "ID and name of customers who own an account in a branch in their city."

(2) "ID and name of customers who do **not** own any account."

(3) "ID and name of customers who own an account in **every** branch."

(4) "ID and name of customers who own an account with a balance which is no less than the balance of any other account."

Solution.

(1) The expression of Problem 1 (and, in turn, also the one of Problem 2) does exactly what is required if add a final projection over *ID* and *Name* on top of it. But let us see how to build a much simpler query from scratch.

We know from the lectures how to join customers with accounts they own (via the customer's ID):

$$\textsc{Customer} \bowtie_{\text{ID=CustID}} \textsc{Account}$$

Additionally, we want the Customer's *City* to be the same as the corresponding Account's *Branch*, so we add this requirement to the join condition:

$$\textsc{Customer} \bowtie_{\text{ID=CustID} \wedge \text{City=Branch}} \textsc{Account}$$

Finally, we only want the *ID* and *Name* of customers, so we project over these two attributes:

$$\pi_{\text{ID, Name}}(\textsc{Customer} \bowtie_{\text{ID=CustID} \wedge \text{City=Branch}} \textsc{Account})$$

Of course, if needed, we can express the theta-join in terms of primitive operations only:

$$\pi_{\text{ID, Name}}\left(\sigma_{\text{ID=CustID} \wedge \text{City=Branch}}(\textsc{Customer} \times \textsc{Account})\right)$$

(2) We know how to get customers who own an account:

$$\pi_{\text{ID, Name, City}}(\textsc{Customer} \bowtie_{\text{ID=CustID}} \textsc{Account})$$

This is precisely the *semijoin* $\textsc{Customer} \ltimes_{\text{ID=CustID}} \textsc{Account}$. Now, if we remove the answers to this query (customers who own an account) from all customers, we get the customers who do **not** own an account:

$$\textsc{Customer} - \underbrace{\textsc{Customer} \ltimes_{\text{ID=CustID}} \textsc{Account}}_{\equiv\ \pi_{\text{ID,Name,City}}(\textsc{Customer} \bowtie_{\text{ID=CustID}} \textsc{Account})}$$

This is precisely the *antijoin* $\textsc{Customer} \mathbin{\overline{\ltimes}}_{\text{ID}=\text{CustID}} \textsc{Account}$.[2] Since we are only interested in *ID* and *Name* of customers (who do not own an account), we project over those attributes:

$$\pi_{\text{ID, Name}}(\underbrace{\textsc{Customer} \mathbin{\overline{\ltimes}}_{\text{ID}=\text{CustID}} \textsc{Account}}) \equiv \textsc{Customer} - \textsc{Customer} \ltimes_{\text{ID}=\text{CustID}} \textsc{Account}$$

(3) The first instinct would be to use division as follows:

$$\pi_{\text{ID, Name, Branch}}(\textsc{Customer} \bowtie_{\text{CustID}=\text{ID}} \textsc{Account}) \div \pi_{\text{Branch}}(\textsc{Account})$$

However, the above query does not work when Account is empty, because in such a case it gives an empty answer, whereas, in accordance with the logical notion of universal quantification, all customers should trivially be customers who own an account in **every** branch, simply because there are no branches! If we call the above query Q, this can be achieved with the following expression:

$$Q \cup \left(\pi_{\text{ID, Name}}(\textsc{Customer}) \mathbin{\overline{\ltimes}}_{1=1} \pi_{\varnothing}(\textsc{Account})\right)$$

which will give us all rows in Customer when Account is empty, and Q otherwise.

There is also another (equivalent) way to express this query, which does not use a division, but it is very similar to one:

$$\pi_{\text{ID, Name}}(\textsc{Customer}) - \pi_{\text{ID, Name}}\big(\pi_{\text{ID, Name}}(\textsc{Customer}) \times \pi_{\text{Branch}}(\textsc{Account}) - \pi_{\text{ID, Name, Branch}}(\textsc{Customer} \bowtie_{\text{CustID}=\text{ID}} \textsc{Account})\big)$$

The above expression is actually simpler than the previous one and matches the way of expressing the query in relational calculus (as we shall see).

(4) We first need to find accounts with maximal balance, that is, *accounts for which there is no (other) account with a higher balance*. This is obtained by (anti-)joining Account with a renamed version of itself as follows:

$$\textsc{Account} \mathbin{\overline{\ltimes}}_{\text{Balance} < \text{Balance}'} \textsc{Account}'$$

where $\textsc{Account}' = \rho_{\text{Number}\to\text{Number}', \text{Branch}\to\text{Branch}', \text{CustID}\to\text{CustID}', \text{Balance}\to\text{Balance}'}(\textsc{Account})$.

Then, we just need to join these accounts (with maximal balance) with their corresponding owners, as usual, and project on *ID* and *Name*:

$$\pi_{\text{ID, Name}}\big(\textsc{Customer} \bowtie_{\text{ID}=\text{CustID}} (\textsc{Account} \mathbin{\overline{\ltimes}}_{\text{Balance} < \text{Balance}'} \textsc{Account}')\big)$$

Problem 4. Can query (4) of Problem 3 ever return more than one tuple? If yes, show a database (over the given schema) on which that happens; otherwise, explain why it cannot happen.

Solution. Yes, because there may be more than one account with the same maximal balance, as in the following database:

Customer

ID	Name	City
1	John	London
2	Mary	Edinburgh

Account

Number	Branch	CustID	Balance
111	London	1	100
222	Edinburgh	2	100

[2] This returns all the rows r in Customer for which there is **no** row s in Account such that the concatenation $t = rs$ of r and s satisfies the condition $t(\text{ID}) = t(\text{CustID})$. Recall that in RA rows are *functions* from attributes to values.