Introduction to Databases
Tutorial 1

Dr Paolo Guagliardo

Fall 2020 (week 3)

Database schema. Consider the following schema:

Customer : *ID*, *Name*, *City*

where *ID* is unique (that is, no two rows in Customer can have the same value for *ID*);

Account : *Number*, *Branch*, *CustID*, *Balance*

where *Number* is unique (that is, no two rows in Account can have the same value for *Number*).

Problem 1 (mandatory). Write query the following queries in SQL:

(1) Return the (value of attribute) *Number* of all accounts owned by customers called "John Doe".

(2) Return the *Number* and *Branch* of all accounts owned by a customer with *ID* "xyz123", only if there is such a customer in the Customer table.

(3) Return the *Number* and *Balance* of all overdrawn accounts in the "London" branch.

(4) Return all pairs (*Name*, *Number*) where *Name* is the name of a customer and *Number* is the number of an account owned by that customer, such that the branch of the account is in a different city than the one where the customer lives.

Solution.

(1)
```
SELECT A.number
FROM   Customer C, Account A
WHERE  C.id = A.custid AND C.name = 'John Doe' ;
```

or, using the explicit join syntax for readability:

```
SELECT A.number
FROM   Customer C JOIN Account A ON C.id = A.custid
WHERE  C.name = 'John Doe' ;
```

(2) To make sure that customers owning the account do indeed exist in the Customer table, we need to join between Account and Customer:

```
SELECT A.number, A.branch
FROM   Customer C JOIN Account A ON C.id = A.custid
WHERE  C.id = 'xyz123' ;
```

Observe that it is *not* enough to select only from Account. In other words, the following query

```
SELECT A.number, A.branch
FROM   Account A
WHERE  A.custid = 'xyz123' ;
```

does not comply with the requirements, because it does not guarantee the existence of a customer with ID "xyz123" in the Customer table.

(3) This is just an easy filter on Account:

```
SELECT A.number, A.balance
FROM   Account A
WHERE  A.branch = 'London' AND A.balance < 0 ;
```

(4)
```
SELECT C.name, A.number
FROM   Account A JOIN Customer C ON A.custid = C.id
WHERE  A.branch <> C.city ;
```

Problem 2 (mandatory). 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) We know from the lectures how to join customers with accounts they own (via the customer's ID):

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

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:

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

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

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

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

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

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

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

This is precisely the *semijoin* $\textsc{Customer} \ltimes_{\text{ID}=\text{CustID}} \textsc{Account}$. 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:

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

This is precisely the *antijoin* $\textsc{Customer}\ \overline{\ltimes}_{\mathrm{ID}=\mathrm{CustID}}\ \textsc{Account}$.[1] We are only interested in *ID* and *Name* of customers (who do not own an account), so we project over these attributes:

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

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

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

But the above query does not work when $\textsc{Account}$ is empty, because in such a case it gives an empty answer while, 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:

$$Q \cup \big(\pi_{\mathrm{ID},\,\mathrm{Name}}(\textsc{Customer})\ \overline{\ltimes}_{1=1}\ \pi_{\varnothing}(\textsc{Account})\big)$$

which will give us $\textsc{Customer}$ when $\textsc{Account}$ 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:

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

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 $\textsc{Account}$ with a renamed version of itself as follows:

$$\textsc{Account}\ \overline{\ltimes}_{\mathrm{Balance}\,<\,\mathrm{Balance}'}\ \textsc{Account}'$$

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

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

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

Problem 3 (optional). Can query (4) of Problem 2 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

[1] This returns all the rows r in $\textsc{Customer}$ for which there is **no** row s in $\textsc{Account}$ such that the concatenation $t = rs$ of r and s does not satisfy the condition $t(\mathrm{ID}) = t(\mathrm{CustID})$. Recall that in RA rows are *functions* from attributes to values.

Problem 4 (optional). 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 query

$$\textsc{Customer} \bowtie \big(\pi_{\mathrm{ID,City}}(\textsc{Customer}) \cap \rho_{\mathrm{CustID}\to\mathrm{ID},\,\mathrm{Branch}\to\mathrm{City}}\big(\pi_{\mathrm{Branch,CustID}}(\textsc{Account})\big)\big)$$

Solution. The projections and renaming are straightforward:

$$\pi_{\mathrm{Branch,CustID}}\left(\begin{array}{llcr} \textbf{Number} & \textbf{Branch} & \textbf{CustID} & \textbf{Balance} \\ \hline 111 & \text{London} & 1 & 120 \\ 222 & \text{Edinburgh} & 1 & 62 \\ 333 & \text{London} & 3 & 76 \\ 444 & \text{London} & 2 & 200 \end{array}\right) = \begin{array}{lc} \textbf{Branch} & \textbf{CustID} \\ \hline \text{London} & 1 \\ \text{Edinburgh} & 1 \\ \text{London} & 3 \\ \text{London} & 2 \end{array}$$

$$\rho_{\mathrm{CustID}\to\mathrm{ID},\,\mathrm{Branch}\to\mathrm{City}}\left(\begin{array}{lc} \textbf{Branch} & \textbf{CustID} \\ \hline \text{London} & 1 \\ \text{Edinburgh} & 1 \\ \text{London} & 3 \\ \text{London} & 2 \end{array}\right) = \begin{array}{lc} \textbf{City} & \textbf{ID} \\ \hline \text{London} & 1 \\ \text{Edinburgh} & 1 \\ \text{London} & 3 \\ \text{London} & 2 \end{array}$$

$$\pi_{\mathrm{ID,City}}\left(\begin{array}{cll} \textbf{ID} & \textbf{Name} & \textbf{City} \\ \hline 1 & \text{John} & \text{London} \\ 2 & \text{Mary} & \text{Edinburgh} \\ 3 & \text{Jeff} & \text{London} \\ 4 & \text{Jane} & \text{Cardiff} \end{array}\right) = \begin{array}{cl} \textbf{ID} & \textbf{City} \\ \hline 1 & \text{London} \\ 2 & \text{Edinburgh} \\ 3 & \text{London} \\ 4 & \text{Cardiff} \end{array}$$

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:

$$\begin{array}{lc} \textbf{City} & \textbf{ID} \\ \hline \text{London} & 1 \\ \text{Edinburgh} & 1 \\ \text{London} & 3 \\ \text{London} & 2 \end{array} \cap \begin{array}{cl} \textbf{ID} & \textbf{City} \\ \hline 1 & \text{London} \\ 2 & \text{Edinburgh} \\ 3 & \text{London} \\ 4 & \text{Cardiff} \end{array} = \begin{array}{lc} \textbf{City} & \textbf{ID} \\ \hline \text{London} & 1 \\ \text{London} & 3 \\ & \\ & \end{array}$$

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

$$\begin{array}{cll} \textbf{ID} & \textbf{Name} & \textbf{City} \\ \hline 1 & \text{John} & \text{London} \\ 2 & \text{Mary} & \text{Edinburgh} \\ 3 & \text{Jeff} & \text{London} \\ 4 & \text{Jane} & \text{Cardiff} \end{array} \bowtie \begin{array}{lc} \textbf{City} & \textbf{ID} \\ \hline \text{London} & 1 \\ \text{London} & 3 \\ & \\ & \end{array} = \begin{array}{cll} \textbf{ID} & \textbf{Name} & \textbf{City} \\ \hline 1 & \text{John} & \text{London} \\ 3 & \text{Jeff} & \text{London} \\ & & \\ & & \end{array}$$

The answer to the query on the given database is:

ID	**Name**	**City**
1	John	London
3	Jeff	London