Relational algebra for databases

Álvaro Fernández Barrero

2025/09/29

When we are working with databases, all the methods we have are based on a field on mathematics called *relational algebra*. This field is quite similar to *set theory* but with some important specific operations, which means that, if you are already familiar with sets in mathematics, understanding relational algebra will be super simple.
In case you are not familiar with set theory whatsoever, I will give here some context before going straight to relational algebra.

1 Set theory context

First, we can define a set like a group of n items:

$$\mathcal{S} = \{5, 10, 15, 20, ...\}$$

This is how we can define the typical basic sets we have been learning since we were little kids making our first steps in mathematics.

$$\mathbb{N} := \{1, 2, 3, 4, 5, ...\}$$

$$\mathbb{Z} := \{..., -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, ...\}$$

$$\mathbb{Q} := \{\frac{a}{b} | a, b \in \mathbb{Z}, b \neq 0\}$$

$$\mathbb{I} := \{\pi, e, \varphi, \sqrt{2}, \sqrt{3}, \sqrt{5}, ...\}$$

It is important to bear in mind the fact that, in a set, there cannot be repeated elements.
Now, it turns out that it would be very powerful to create operations between sets. Two of the most basic and useful ones are *union* and *intersection*.

$$A \cup B \qquad\qquad A \cap B$$

The union operation obtains a new set with all elements in the set A and all elements in the set B, whereas the intersection obtains a new set with all elements that are both in the set A and in the set B.

For example, let us say we have a set A with all multiples of two and a set B with all multiples of five:

$$A := \{x | \frac{x}{2} \in \mathbb{N}\} \qquad B := \{x | \frac{x}{5} \in \mathbb{N}\}$$

Therefore, $A \cup B$ (union) will be a set with all multiples of two and all multiples of five, and $A \cap B$ (intersection) will be a set with all multiples of two and five:

$$A \cup B \ = \{x | \frac{x}{2} \in \mathbb{N} \vee \frac{x}{5} \in \mathbb{N}\} \qquad A \cap B \ = \{x | \frac{x}{2}, \frac{x}{5} \in \mathbb{N}\}$$

In a more rigorous way, we can define these operations as:

$$S \cup T \ := \{x | x \in S \vee x \in T\} \qquad S \cap T \ := \{x | x \in S \wedge x \in T\}$$

In case we need to make unions after unions or intersections after intersections, we can use this notation that is quite similar to summations or products:

$$\bigcup_{\mathcal{A} \in S} \mathcal{A} \qquad \bigcap_{\mathcal{A} \in S} \mathcal{A}$$

Given these operations, we can define now another essential set in mathematics which is the real numbers $\mathbb{R}$:

$$\mathbb{R} := \mathbb{Q} \cup \mathbb{I}$$

Now that we have actually a way to add elements in a set, let us also create an operation to remove them:

$$A \setminus \{2, 4\} = \{x | \frac{x}{2} \in \mathbb{N} \vee \frac{x}{5} \in \mathbb{N}, x > 4\}$$

$$S \setminus T := \{x | x \in S \wedge x \notin T\}$$

We can additionally define a new operation to check how many elements are within a set, which will receive the name *cardinality*.

$$\#A = |A| = n, \ \ n \in \mathbb{N} \cup \{0\}$$

As you might have noticed already, all elements in a set can also be in another set. For instance, all natural numbers $\mathbb{N}$ are in the integers set $\mathbb{Z}$ too. This is called a subset and it can be represented as:

$$\mathbb{N} \subset \mathbb{Z}$$

Nevertheless, a set can be a subset and also equal to another, which is represented as:

$$S \subseteq T$$

From here we could write a simple theorem which says:

Theorem 1 *Let a set S be a subset of another set T if and only if the union of both of them is T.*

Or, expressed more mathematically:

$$S \cup T = T \iff S \subseteq T$$

To finalize set theory, we can define the cartesian product as an operation that relates every element of a set S with all the elements with another set T:

$$S \times T := \{(s,t)|s \in S \land t \in T\}$$

2 Relational algebra

In mathematics, we have several types of algebra, such as linear algebra to study linear operations, abstract algebra to study algebraic structures, boolean algebra to study bit operations and, now, we came across relational algebra, the mathematical language of databases.
In order to study relational algebra, it is essential to understand the basic operations in set theory because when we are working with this algebra, we will come across some set operations. This is why I started with them.

2.1 Selection (σ)

Let us say we have a database named "PROGRAMMERS" with the following information of my programmers: name, ID, field_of_work (database, front-end, back-end, full-stack...) and years_in_business. Now, imagine we have a tough problem in the back-end part, which means that we need to obtain the programmers dedicated to the back-end or full-stack part with at least 5 years of experience in the business to ensure they have the level enough to solve this problem.
To select easily those programmers, we can use the selection operation which returns a tuple with the elements in my database that meet the given conditions:

$$\sigma_{\varphi}(\Sigma)$$

Here, φ are the conditions we have to apply to my database Σ.
For this example, it could look something like:

$$\varphi = years\_in\_business \geq 5 \land (field\_of\_work = "back-end" \lor field\_of\_work = "full-stack")$$

$$\therefore \sigma_{years\_in\_business \geq 5 \land (field\_of\_work = "back-end" \lor field\_of\_work = "full-stack")}(PROGRAMMERS)$$

A simpler example would be the table called "STUDENTS" for a high-school, where we need to save students data such as their names, id or grades. Students in 12th grade are about to graduate, so we need to obtain their information with that condition:

$$\sigma_{grade=12}(STUDENTS)$$

2.2 Projection (Π)

Now, I have an university table called "PROFESSORS" with the information of each professor such as name, id, department and experience. But I actually only care for a moment about the name and department. Here we can use this projection, an operation that is quite similar to selection, but whereas selection gives me the the information of each element that meet the given conditions, projection gives me the information (attributes) I need of every element.

$$\Pi_{a_1,\, a_2,\, a_3,\, ..,\, a_n}(\Sigma)$$

In the example, the result would look as:

$$\Pi_{name,\, department}(PROFESSORS)$$

2.3 Union ($\cup$)

Just like in *set theory*, in *relational algebra* there is also a union operation that works exactly like you might expect.
We still have the table "PROFESSORS" with the attributes name, id, subjects, department and experience, but additionally, now we have also a table "STUDENTS" whose attributes are name, id, grade and subjects. I could be interested in mixing both tables to have only one with this information.
This is a clear example of use for my union operation, which has the next form, exactly like in set theory:

$$\Sigma \cup \Lambda$$

The example looks like:

$$PROFESSORS\ \cup\ STUDENTS$$

But I might be only interested in relevant information for my case like the name, id or subjects. Thus, I can use the projection to obtain only those attributes:

$$\Pi_{name,\, id,\, subjects}(PROFESSORS)\ \cup\ \Pi_{name,\, id,\, subjects}(STUDENTS)$$

2.4 Set difference (-)

This operation removes the elements in a table Σ that are also in Λ.

$$\Sigma - \Lambda$$

For instance, imagine we have the table for the events done in each year. Each table has the attributes type, date and money_made and now I need to know which events I did in 2022 that I didn't in 2023 or 2024 because I ran out of ideas. It would look as:

$$E\_2022 - E\_2023 - E\_2024$$

Since I am only interest in the type of the event per se, I could use again a projection:

$$\Pi_{type}(E\_2022) - \Pi_{type}(E\_2023) - \Pi_{type}(E\_2024)$$

I am actually interest in how popular was the event, which means that I will set a minimum money the event has earned so I do not repeat events people did not like:

$$\Pi_{type}(\sigma_{money\_made \geq n}(E\_2022)) - \Pi_{type}(\sigma_{money\_made \geq n}(E\_2023)) - \Pi_{type}(\sigma_{money\_made \geq n}(E\_2024))$$

$$n \in \mathbb{R}^+$$

2.5 Cartesian product ($\times$)

Like the union operation, cartesian product in relational algebra does exactly what it does in set theory: pairs all the elements in the first table Σ with each element in the second table Λ.

$$\Sigma \times \Lambda$$

For instance, imagine you have two tables: one with the students in mathematics named "MATHEMATICS" and another with the students in physics named "PHYSICS". Both tables have the attributes name, id, age and grade. Now, you want to know which students are in both, so we can do:

$$\sigma_{MATHEMATICS.id=PHYSICS.id}(\Pi_{name,\, id}(MATHEMATICS)\ \times\ \Pi_{name,\, id}(PHYSICS))$$

As you may have noticed, we can reference a specific attribute in one of the tables in order to make this condition: the ids must be the same in MATHEMATICS and in PHYSICS. This same concept will show up again a bit later, keep it in mind.

2.6 Rename (ρ)

When we have an operation, the outcome of it does not really have a name, which makes it a bit harder to work with. It would be extremely useful to have the power to name the table result after an operation with their attributes. In order to solve this problem, we have this operator:

$$\rho_{\Phi(n_1,\, n_2,\, n_3,\, ...,\, n_k)}(\Sigma)$$

A example could be the following one:

$$\rho_{next\_events(type,\, money\_made)}(\Pi_{type}(\sigma_{money\_made > n}(E\_2022)) - \Pi_{type}(\sigma_{money\_made > n}(E\_2023)))$$

2.7 Intersection ($\cap$)

Just like union and cartesian product, intersection has a super stretch relationship with its version in set theory because it is actually the same operation: returns only the elements both tables share.

$$\Sigma \cap \Lambda$$

For example. let us say we have a library database, where we can find two tables: one table *"READER"* saves the people who stay to read there with the attributes name, id, stayed_time and book_read and, in the other table *"BORROWERS"*, we save the people who borrow books with the attributes name, id, books_borrowed. With this information, we require to know who did both, who stays to read in the library and borrows books. Here, we can apply this operation like this:

$$\Pi_{name,\, id}(READERS) \cap \Pi_{name,\, id}(BORROWERS)$$

2.8 Assignment operator ($=$ / $\leftarrow$)

This operation has nothing weird and we all are super used to use this operator. As you might expect, it just assign values like in mathematics by saying $x = n$ or in programming by saying "number = n;" where number is a variable. The only difference is that we actually have two symbols to represent this operation: the classical "=" or a left arrow "$\leftarrow$"

$$\Sigma = \Lambda \qquad\qquad \Sigma \leftarrow \Lambda$$

For instance, we could see:

$$\Gamma \leftarrow \Pi_{a_1,\, a_2}(\sigma_{a_1}(\Sigma)) \times \Lambda$$

2.9 Join operations

Join operations are a mix of cartesian products with a selection process for the result. We can find several types:

2.9.1 θ-join/conditional join ($\bowtie_\theta$)

This is the most general case of the join operations. This can be defined as:

$$\Sigma \bowtie_\theta \Lambda := \sigma_\theta(\Sigma \times \Lambda)$$

2.9.2 Equi-join ($\bowtie_\theta$)

This is a special case of θ-join, the difference is only the fact that you can only use a equivalence to make your conditions, but the rest remains the same and so does the definition. This is written exactly the same as θ-join:

$$\Sigma \bowtie_{\theta} \Lambda := \sigma_{\theta}(\Sigma \times \Lambda)$$

We have already used this operation before without even knowing it in the example for the *cartesian product*. The given result of it can be rewritten as:

$$\Pi_{name,\,id}(MATHEMATICS) \bowtie_{MATHEMATICS.id=PHYSICS.id} \Pi_{name,\,id}(PHYSICS)$$

2.9.3 Natural join ($\bowtie_*$)

This is a special case of equi-join. In order to use this operation, there must be at least one common attribute between Σ and Λ.
When we are using this operation, we do not have to specify any condition explicitly. Instead, it compares the values of the commons attributes and, if they are the same, they will stay in the result. It is basically a way to get a table with the rows in my tables that contains the same value.
This can be defined as:

$$\Sigma \bowtie_* \Lambda := \Pi_{\{x|x=\Sigma.a_i\}\cup(\{x|x=\Lambda.a_i\}-\{A|A\,\in\,\{x|x=\Sigma.a_i\}\cap\{x|x=\Lambda.a_i\}\})}(\sigma_{\bigwedge_{a\in\{x|x=\Sigma.a_i\}\cap\{x|x=\Lambda.a_i\}}\Sigma.a=\Lambda.a}(\Sigma\times\Lambda))$$

This might seem a bit confusing, so let us define a couple of variables:

$$R := \{x|x = \Sigma.a_i\}$$
$$S := \{x|x = \Lambda.a_i\}$$

Here, R represents all attributes in the Σ table and S all the attributes in the Λ table. Given these new variables, the definition will turn into:

$$\Sigma \bowtie_* \Lambda = \Pi_{R\,\cup\,(S-\{A|A\,\in\,R\,\cap\,S\})}(\sigma_{\bigwedge_{a\in R\,\cap\,S}\Sigma.a=\Lambda.a}(\Sigma \times \Lambda))$$

This looks a bit more readable.

2.10 Division operation (/)

The division operation is used when you want to find the elements in one table Σ that are related to all elements in another table Λ.

$$\Sigma/\Lambda$$

Theorem 2 *Given two tables Σ and Λ, we can define the operation $\Sigma/\Lambda \iff \Lambda \subset \Sigma$. If $\Lambda \not\subseteq \Sigma \implies \nexists \Sigma/\Lambda$.*

For instance, let us say we have a table STUDENTS with the attributes name, id, age and course_id and another one called COURSES with the attribute course_id and available. We need to check who has enrolled in all the courses we have. To solve this problem, we can use this division operation like:

$$\Gamma \leftarrow \rho_{ENROLLED\_ALL\_COURSES}(\Pi_{name,id,course\_id}(STUDENTS)/\sigma_{available="true"}(COURSES))$$