# Relational Algebra Exercises for Tutorial

Solve all queries below using only select, project, Cartesian product, and natural join. Do not use theta-join, set operations, renaming or assignment.




## First Schema

Suppliers(sID, sName, address)

Parts(pID, pName, colour)

Catalog(sID, pID, price)

Catalog[sID] $\subseteq$ Suppliers[sID]

Catalog[pID] $\subseteq$ Parts[pID]

Notice:

- In this schema, everywhere we want values to match across relations, the attributes have matching names. And everywhere the attributes have matching names, we want values to match across relations.
- This means that natural join will do exactly what we want in all cases.

Questions:

1. If sID is a key for the Suppliers relation, could it be a key for the Catalog relation?

   Answer: Just because it is a key in one relation doesn't mean it is in another; being a key is relative to the relation. But *is* it a key for Catalog? No. We almost surely want to be able to list multiple parts by one supplier in our catalog.

2. Find the names of all red parts. → look for partsName whose colour is red

   Answer:
   → columns → row
   $\Pi_{pName}(\sigma_{colour=\text{“}red\text{”}} Parts)$

3. Find all prices for parts that are red or green. (A part may have different prices from different manufacturers.)

   (i) select row of colour in Parts
   (ii) select column of price

   Answer:

   $\Pi_{price}((\sigma_{colour=\text{“}red\text{”} \vee colour=\text{“}green\text{”}} Parts) \bowtie Catalog)$

   ↳ or ↳ natural join

4. Find the sIDs of all suppliers who supply a part that is red or green.

   Answer:

   $\Pi_{sID}((\sigma_{colour=\text{“}red\text{”} \vee colour=\text{“}green\text{”}} Parts) \bowtie Catalog)$

   $\Pi_{sID}((\sigma_{color='green' \vee color='red'} Parts) \bowtie Catalog)$

5. Find the sIDs of all suppliers who supply a part that is red *and* green.

   Answer: Trick question. Each tuple has only one colour, and each part has only one tuple (since pID is a key), so no part can be recorded as both red and green.

6. Find the names of all suppliers who supply a part that is red or green.

   Answer:

   $\Pi_{sName}((\Pi_{sID}((\sigma_{colour=\text{“}red\text{”} \vee colour=\text{“}green\text{”}} Parts) \bowtie Catalog)) \bowtie Suppliers)$

   $\Pi_{sName}((\Pi_{sID}((\sigma_{color='green' \vee color='red'} Parts) \bowtie Catalog)) \bowtie Suppliers)$

## Second Schema

Employees(number, name, age, salary)

Supervises(boss, employee)

Supervises[boss] $\subseteq$ Employees[number]

Supervises[employee] $\subseteq$ Employees[number]

Notice:

- In this schema, wherever we want values to match across relations, the attributes **do not** have matching names. This means that natural join will not force things to match up as we'd like.
- In fact, since there are no attribute names in common across the two relations, natural join is no different from Cartesian product.
- We are forced to use selection to enforce the necessary matching.

Questions:

1. What does it say about our domain that employee is a key for Supervises?

   Answer: Every employee has one boss.

2. Does the schema allow for an employee with no boss?

   Answer: Yes.

3. How would the world have to be different if boss were a key for Supervises?

   Answer: It would mean that every boss could have at most one employee. Not very sensible!

4. How would the world have to be different if both boss and employee together were a key for Supervises?

   Answer: This would imply that neither alone is a key, since keys are minimal. Thus, bosses could have multliple employees (sensible) and employees could have multiple bosses (possibly sensible).

5. Find the names and salaries of all bosses who have an employee earning more than 100.
   **Hint:** Below each subexpression, write the names of the attributes in the resulting relation.

   Answer:

   $\Pi_{name,salary}\sigma boss = number((\Pi_{boss}\sigma_{number=employee}((\Pi_{number}\sigma_{salary>100}Employee)\times Supervises))\times Employee)$

## Third Schema

This schema is for a salon. Services could be things like "haircut" or "manicure".

Clients(CID, name, phone)

Staff(SID, name)

Appointments(CID, date, time, service, SID)

Appointments[CID] $\subseteq$ Clients[CID]

Appointments[SID] $\subseteq$ Staff[SID]

Notice:

- In this schema, everywhere we want values to match across relations, the attributes have matching names. But there are also attributes with matching names whose values we do not want to match across relations.
- In those cases, that natural join will get rid of many tuples that we need, so we must use Cartesian product and make any necessary matching happen using select. (Unless we can remove the problem attributes first.)

Questions:

1. Find the appointment time and client name of all appointments for staff member Giuliano on Feb14. (Assume that you can compare a date value to "Feb 14" using "="). At each step, use projection to pare down to only the attributes you need.

   Answer: $\Pi_{name,time}((\Pi_{CID,SID,time}\sigma_{date="Feb14"}Appointments) \bowtie (\Pi_{SID}\sigma_{name="Giuliano"}Staff) \bowtie Clients)$

2. Now solve the same problem but begin by putting all three relations together in full — with all of their attributes.

   Answer:

   This time, we mustn't use natural join or we'll force the client name and staff names to match, which would be very inappropriate! So we use Cartesian product and are stuck enforcing all the things that do need to match, like SID when we combine Staff and Appointments.

   I'm not going to write out this version of the query.

3. Which answer is better?

   Answer:

   The first is more "efficient" because it produces smaller intermediate relations. But since this is all just math, it doesn't matter! (And in a DMBS, where queries are actually executed and can therefore be more or less efficient, the DBMS optimizes our queries.)