

Relational Algebra: Sample Solutions

Note that the solutions given here are *samples*, i.e., there may be many more ways to express these queries in relational algebra.

1. Write queries in relational algebra

Write the following queries in relational algebra.

1. “Find the names of suppliers who supply some red part.”

$$\pi_{\text{sname}}((\sigma_{\text{colour}='red'}(\text{Part}) \bowtie \text{Catalog}) \bowtie \text{Supplier}))$$

Since there is not subscript under the joins, the joins are natural joins, i.e., the common attributes are equated.

2. “Find the IDs of suppliers who supply some red or green part.”

$$\pi_{\text{sid}}(\sigma_{\text{colour}='red' \vee \text{colour}='green'}(\text{Part}) \bowtie \text{Catalog})$$

An equivalent formulation uses the union operator

$$\pi_{\text{sid}}(\sigma_{\text{colour}='red'}(\text{Part}) \bowtie \text{Catalog} \cup \sigma_{\text{colour}='green'}(\text{Part}) \bowtie \text{Catalog}).$$

The latter version can be refined by pushing the projection through the union:

$$\pi_{\text{sid}}(\sigma_{\text{colour}='red'}(\text{Part}) \bowtie \text{Catalog}) \cup \pi_{\text{sid}}(\sigma_{\text{colour}='green'}(\text{Part}) \bowtie \text{Catalog}).$$

3. “Find the IDs of suppliers who supply some red part or are based at 21 George Street.”

$$\pi_{\text{sid}}(\sigma_{\text{colour}='red'}(\text{Part}) \bowtie \text{Catalog}) \cup \pi_{\text{sid}}(\sigma_{\text{address}='21G.S.'}(\text{Supplier})).$$

4. “Find the names of suppliers who supply some red part or are based at 21 George Street.”

$$\begin{aligned} &\pi_{\text{sname}}(\sigma_{\text{colour}='red'}(\text{Part}) \bowtie \text{Catalog} \bowtie \text{Supplier}) \\ &\cup \pi_{\text{sname}}(\sigma_{\text{address}='21G.S.'}(\text{Supplier})). \end{aligned}$$

Alternatively, we can pull the projection on sname to the top level.

$$\begin{aligned} &\pi_{\text{sname}}(\sigma_{\text{colour}='red'}(\text{Part}) \bowtie \text{Catalog} \bowtie \text{Supplier}) \\ &\cup \pi_{\text{sname}}(\sigma_{\text{address}='21G.S.'}(\text{Supplier})). \end{aligned}$$

5. “Find the IDs of suppliers who supply some red part and some green part.”

$$\pi_{\text{sid}}(\sigma_{\text{colour}='red'}(\text{Part}) \bowtie \text{Catalog}) \cap \pi_{\text{sid}}(\sigma_{\text{colour}='green'}(\text{Part}) \bowtie \text{Catalog})$$

Alternatively, we can replace the intersection with a join:

$$\pi_{\text{sid}}(\sigma_{\text{colour}='red'}(\text{Part}) \bowtie \text{Catalog}) \bowtie \pi_{\text{sid}}(\sigma_{\text{colour}='green'}(\text{Part}) \bowtie \text{Catalog}).$$

6. “Find pairs of sids such that the supplier with the first sid charges more for some part than the supplier with the second sid.”

First, by creating temporary copies we introduce two versions of Catalog:

$$\begin{aligned} \text{Cat1} &\leftarrow \text{Catalog} \\ \text{Cat2} &\leftarrow \text{Catalog}. \end{aligned}$$

Then we join the two versions and take the projection:

$$\pi_{\text{Cat1.sid}, \text{Cat2.sid}}(\text{Cat1} \bowtie_{\text{Cat1.pid}=\text{Cat2.pid} \wedge \text{Cat1.cost} > \text{Cat2.cost}} \text{Cat2})$$

Note that we have to qualify the attributes by the relation name because both relations have attributes with the same names.

7. “Find the IDs of suppliers who supply only red parts.”

$$\pi_{\text{sid}}(\text{Supplier}) \setminus \pi_{\text{sid}}(\text{Catalog} \bowtie_{\sigma_{\text{colour} \neq 'red'}}(\text{Part})).$$

8. “Find the IDs of suppliers who supply every part.”

First, we observe that the projection of Catalog on the attributes sid and pid,

$$\pi_{\text{sid}, \text{pid}}(\text{Catalog})$$

contains all pairs of suppliers and the parts they supply, expressed by sids and pids.

The Cartesian product

$$\pi_{\text{sid}}(\text{Catalog}) \times \pi_{\text{pid}}(\text{Part})$$

contains all possible combinations of (1) the suppliers that supply something with (2) all the products.

If we take the set-theoretic difference of the second and the first relation,

$$\pi_{\text{sid}}(\text{Catalog}) \times \pi_{\text{pid}}(\text{Part}) \setminus \pi_{\text{sid}, \text{pid}}(\text{Catalog}),$$

we obtain those combinations of suppliers and parts where the supplier does not deliver the part. Let us store the difference in the temporary relation Temp1:

$$\text{Temp1} \leftarrow \pi_{\text{sid}}(\text{Catalog}) \times \pi_{\text{pid}}(\text{Part}) \setminus \pi_{\text{sid}, \text{pid}}(\text{Catalog}).$$

Now, $\pi_{\text{sid}}(\text{Temp1})$, the projection of this relation onto the attribute sid, gives us the suppliers that do *not* supply some part, i.e., some part is missing in their range of products.

However, $\pi_{\text{sid}}(\text{Catalog})$, the projection of **Catalog** onto **sid** gives us the suppliers that *do* supply *at least* some part.

Therefore,

$$\pi_{\text{sid}}(\text{Catalog}) \setminus \pi_{\text{sid}}(\text{Temp1}),$$

the difference of the latter and the former, gives us the **sid**'s of suppliers that

- supply something, and
- have no part missing in their range of products.

That means, it gives us the **sid**'s of the suppliers that supply all parts.

2. Queries in relational algebra, what do they mean?

For each of the following relational algebra queries, say what they mean.

1. $\pi_{\text{sname}}(\sigma_{\text{colour}='red'}(\text{Part}) \bowtie \sigma_{\text{cost}<100}(\text{Catalog}) \bowtie \text{Supplier})$

“Find the names of suppliers supplying some red part for less than 100 Quid.”

2. $\pi_{\text{sname}}(\pi_{\text{sid}}(\sigma_{\text{colour}='red'}(\text{Part}) \bowtie \sigma_{\text{cost}<100}(\text{Catalog})) \bowtie \text{Supplier})$

“Find the names of suppliers supplying some red part for less than 100 Quid.”

This query is an optimized version of the previous one: from the join of **Part** and **Catalog** it retains only the projection onto **sid**. Then it uses **sid** to retrieve the **sname**s of the suppliers.

3. $\pi_{\text{sname}}(\sigma_{\text{colour}='red'}(\text{Part}) \bowtie \sigma_{\text{cost}<100}(\text{Catalog}) \bowtie \text{Supplier}) \cap \pi_{\text{sname}}(\sigma_{\text{colour}='green'}(\text{Part}) \bowtie \sigma_{\text{cost}<100}(\text{Catalog}) \bowtie \text{Supplier})$

“Find the names of suppliers such that there is a supplier with that name supplying some red part for less than 100 Quid and a supplier with that name supplying some green part for less than 100 Quid.”

The English rendering of the query sounds unnecessarily complicated. However, as the following example shows, the complicated formulation is needed to express the meaning of the query.

Suppose there are two distinct suppliers, both with the name 'Smith', such that the first Smith supplies some red part for less than 100, and the second Smith supplies some green part for less than 100. Then the name 'Smith' is returned by this query, even if the first Smith doesn't supply a green part, and the second doesn't supply a red part.

4. $\pi_{\text{sid}}(\sigma_{\text{colour}='red'}(\text{Part}) \bowtie \sigma_{\text{cost}<100}(\text{Catalog}) \bowtie \text{Supplier}) \cap \pi_{\text{sid}}(\sigma_{\text{colour}='green'}(\text{Part}) \bowtie \sigma_{\text{cost}<100}(\text{Catalog}) \bowtie \text{Supplier})$

“Find the sids of suppliers supplying some red part for less than 100 Quid and some green part for less than 100 Quid, and return only sid's of suppliers recorded in the table Supplier.”

If there is a foreign key constraint on `Catalog(sid)`, then the join with `Supplier` is superfluous. (Some query optimisers find out such redundant joins. For instance, Microsoft's SQL Server eliminates joins that are redundant because of foreign key constraints, and it eliminates parts of the query that are inconsistent.)

$$5. \pi_{\text{name}}(\pi_{\text{sid}, \text{name}}(\sigma_{\text{colour}=\text{'red'}}(\text{Part}) \bowtie \sigma_{\text{cost} < 100}(\text{Catalog}) \bowtie \text{Supplier}) \cap \pi_{\text{sid}, \text{name}}(\sigma_{\text{colour}=\text{'green'}}(\text{Part}) \bowtie \sigma_{\text{cost} < 100}(\text{Catalog}) \bowtie \text{Supplier}))$$

“Find the names of suppliers supplying some red part for less than 100 Quid and some green part for less than 100 Quid.”

This is the query that intuitively makes most sense.

Databases

DBDMG - Politecnico di Torino

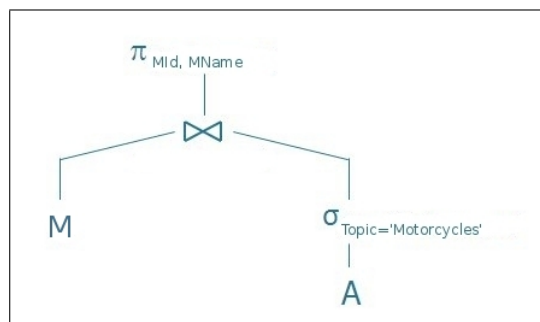
Relational Algebra (I) – Solutions

Exercise 1. Given the relational schema including the following tables (primary keys are underlined):

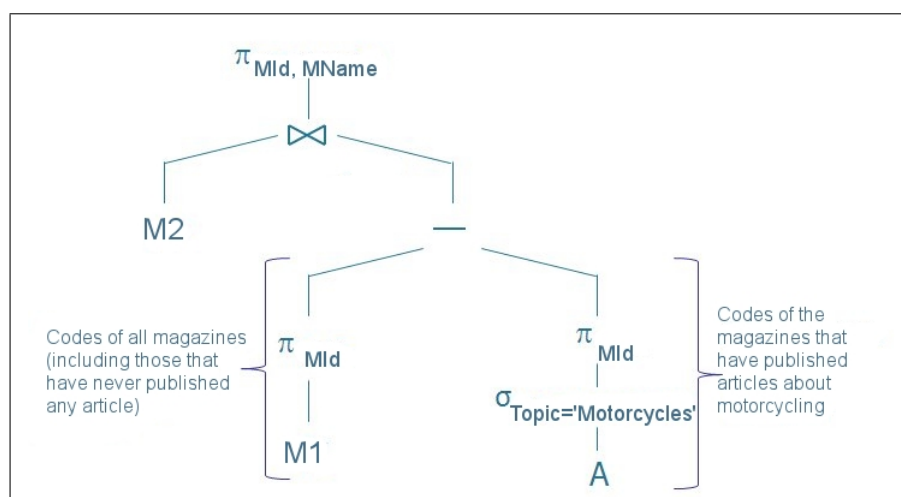
MAGAZINE (MId, MName, Publisher)
 ARTICLE (AId, Title, Topic, MId)

express the following queries in relational algebra:

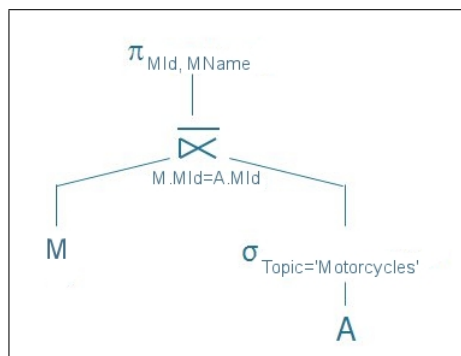
- (a) Find the names of the magazines that have published at least one article about motorcycles.



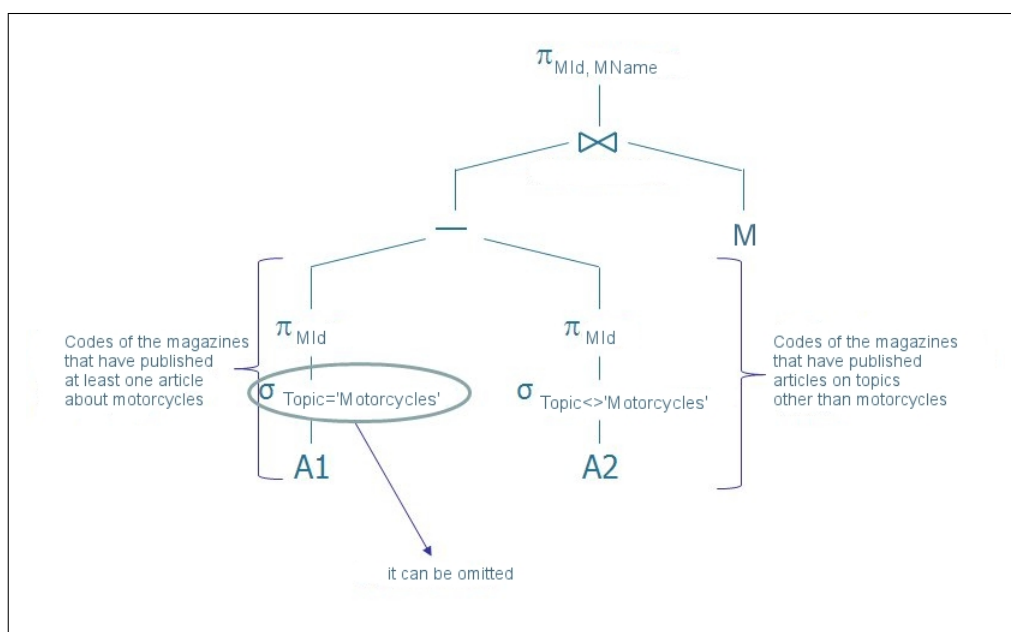
- (b) Find the names of the magazines that have never published any article about motorcycles.



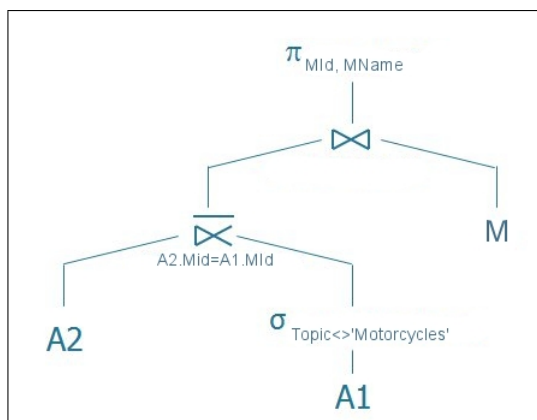
Alternative solution:



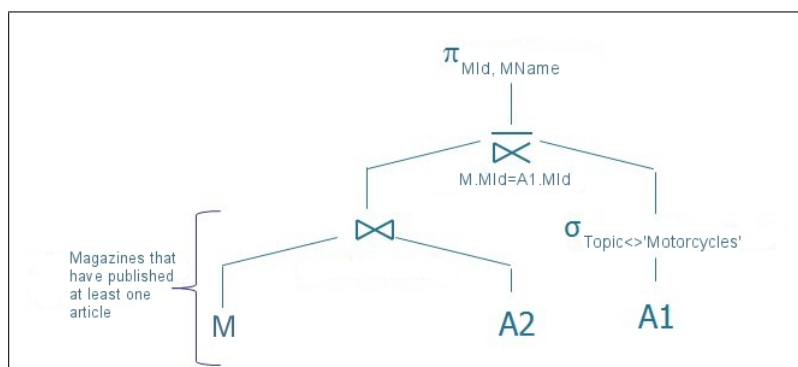
(c) Find the names of the magazines that have only ever published articles about motorcycles.



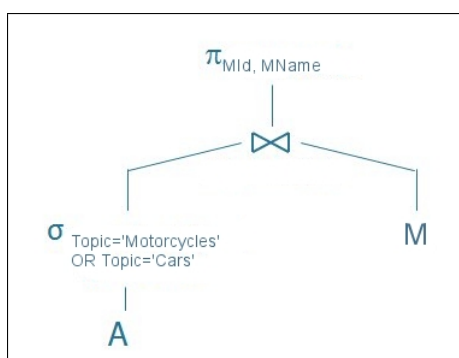
Alternative solution:



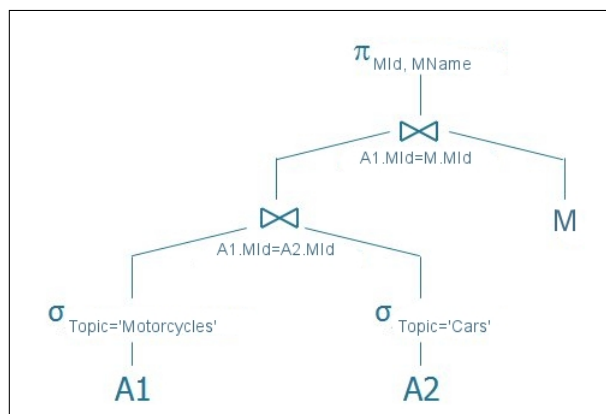
Alternative solution:



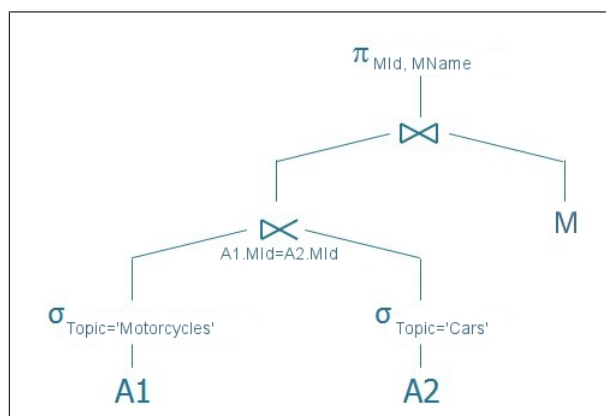
(d) Find the names of the magazines that publish articles about motorcycles or cars.



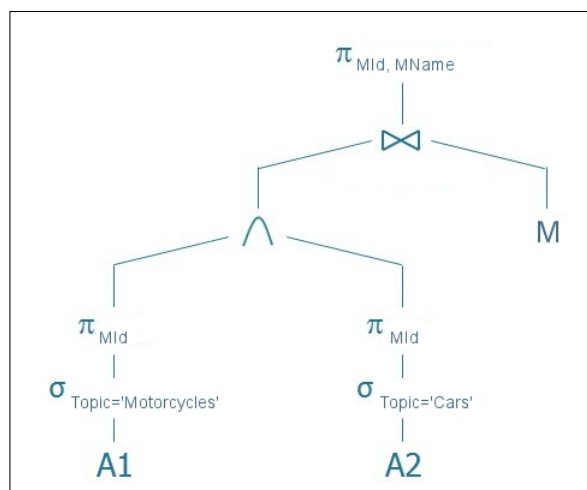
(e) Find the names of the magazines that publish both articles about motorcycles and articles about cars.



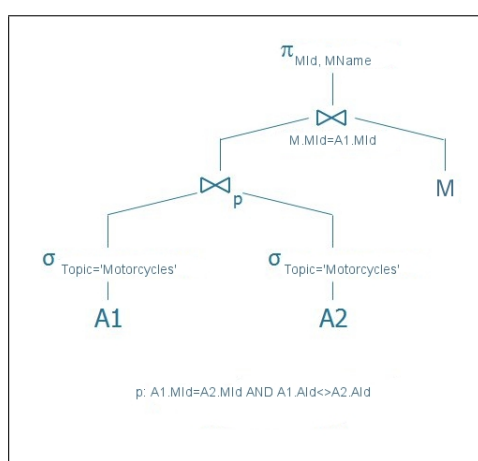
Alternative solution:



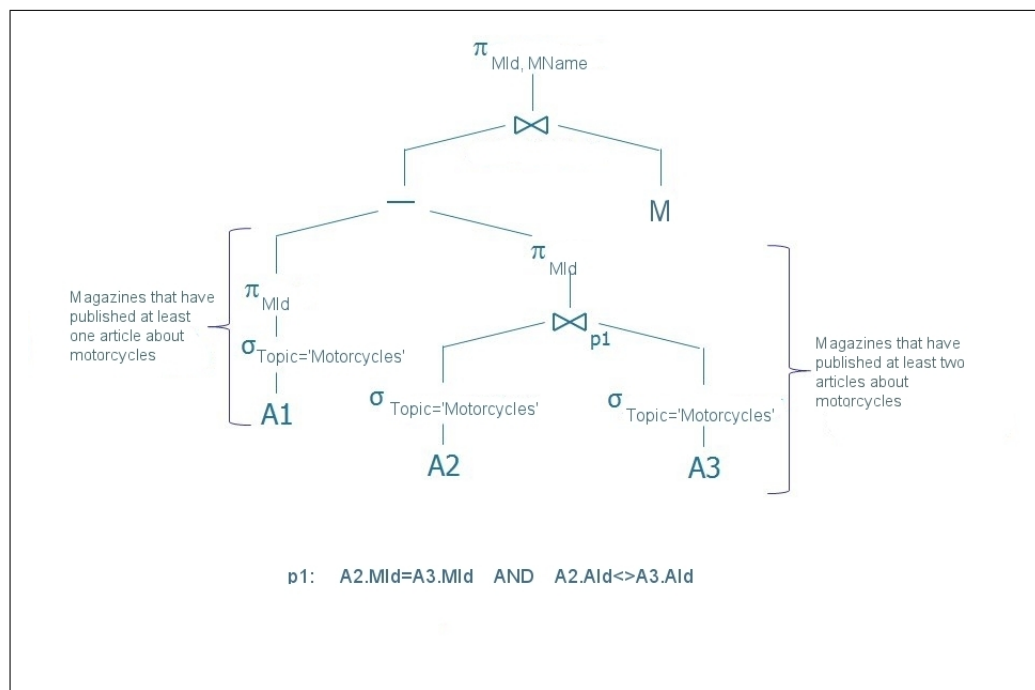
Alternative solution:



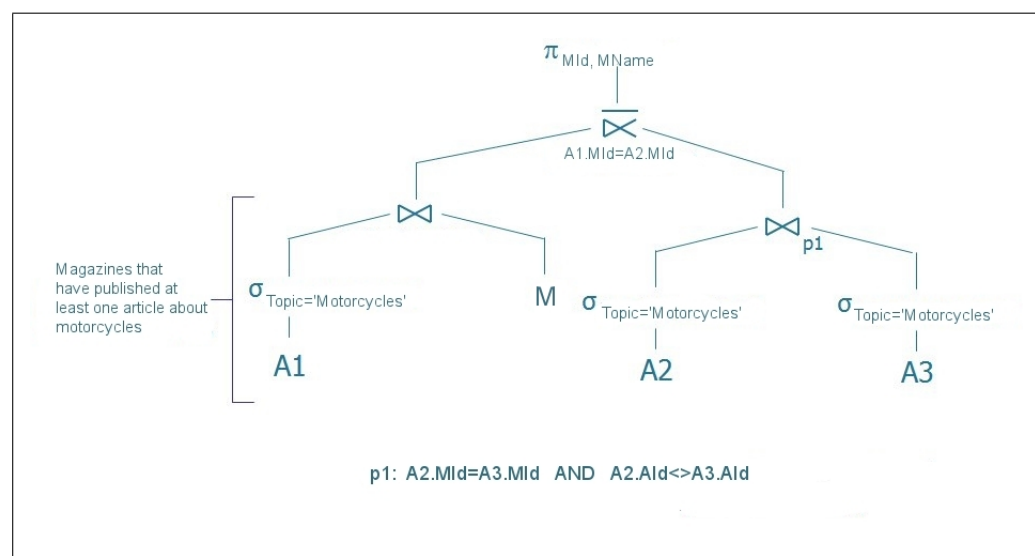
(f) Find the names of the magazines that have published at least two articles about motorcycles.



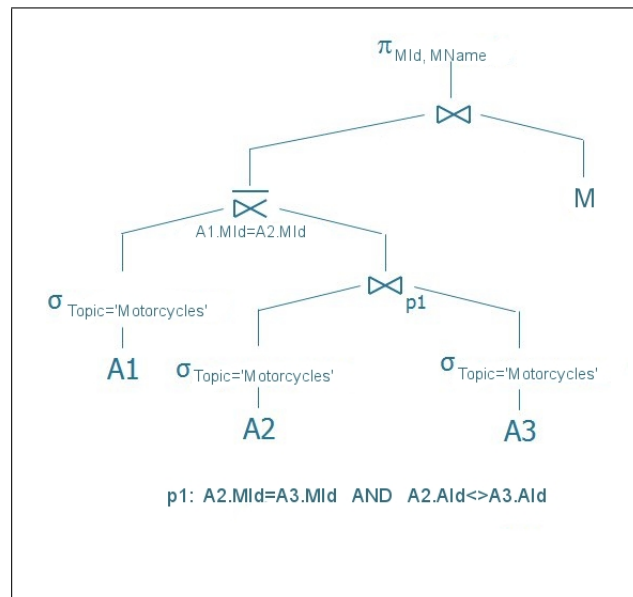
- (g) Find the names of the magazines that have published only one article about motorcycles (i.e., they may have published any number of articles about other topics).



Alternative solution:



Alternative solution:



Exercise 2. Given the relational schema including the following tables (primary keys are underlined):

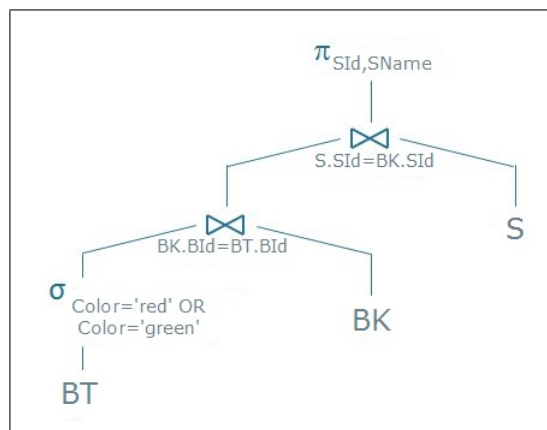
SAILOR(SId, SName, Expertise, DateOfBirth)

BOOKING(SId, BId, Date)

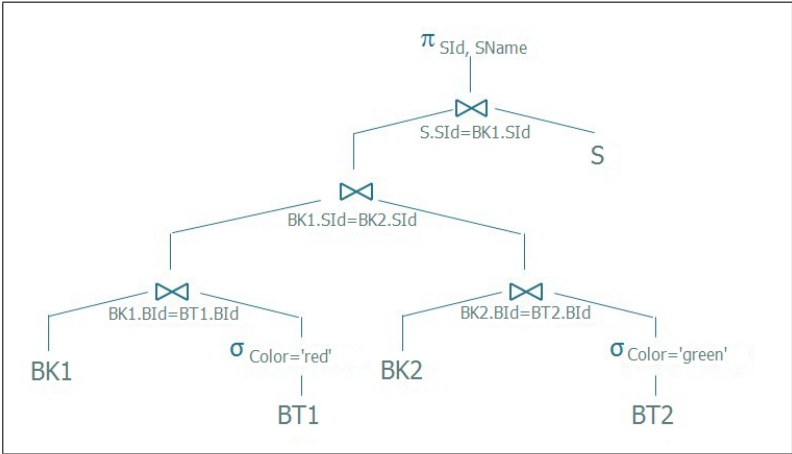
BOAT(BId, BName, Color)

express the following queries in relational algebra:

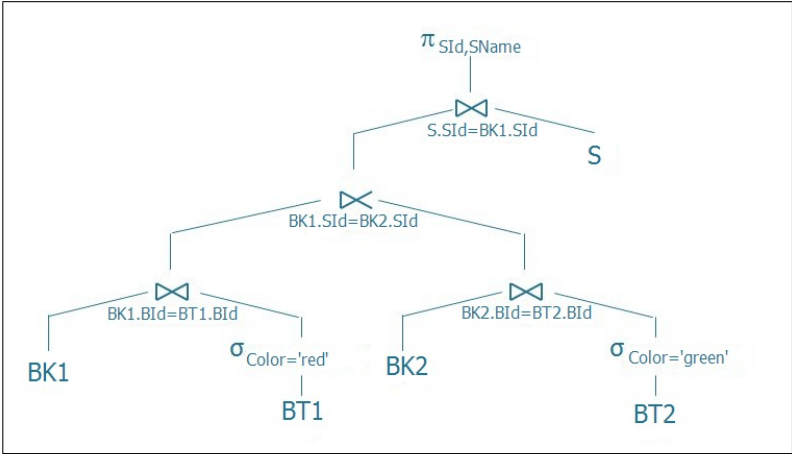
(a) Find the names of the sailors who have booked a red boat or a green boat.



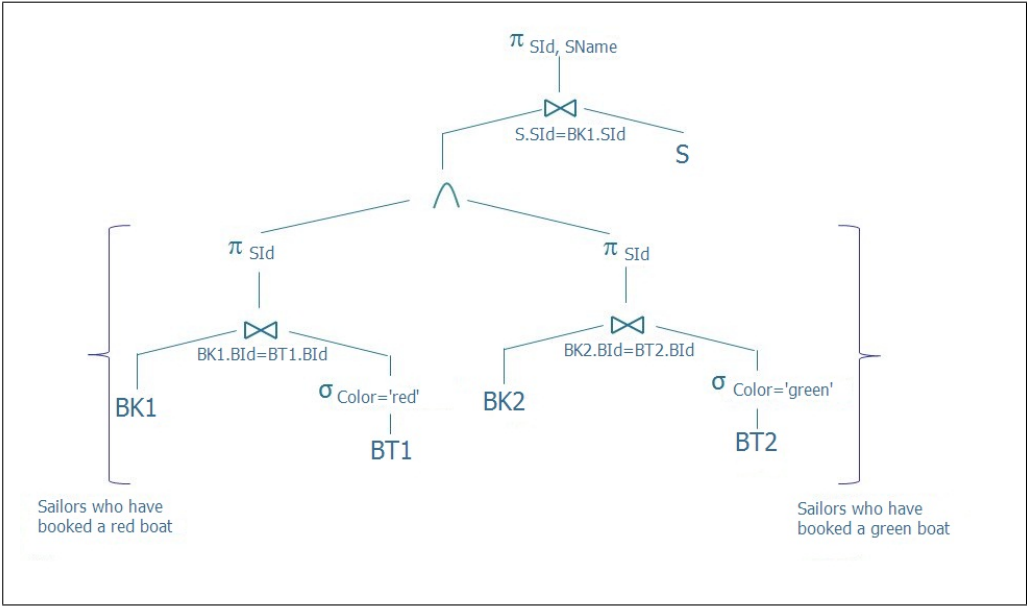
(b) Find the codes and the names of the sailors who have booked a red boat and a green boat.



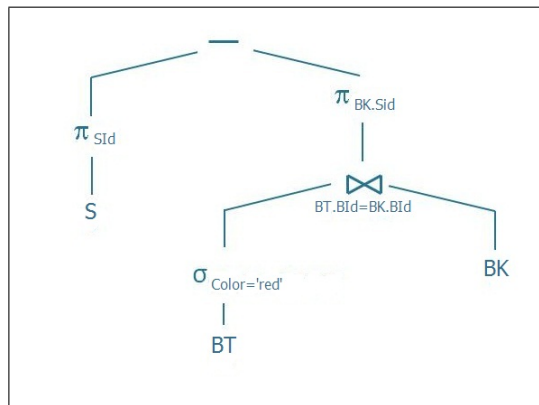
Alternative solution:



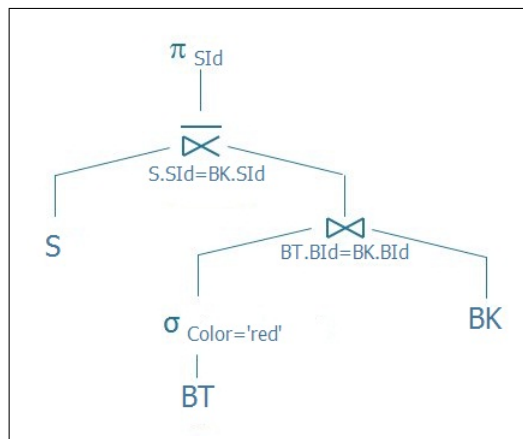
Alternative solution:



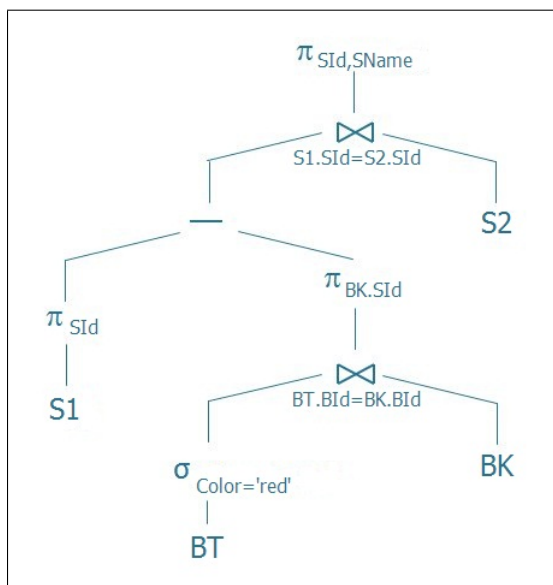
(c) Find the codes of the sailors who have never booked a red boat.



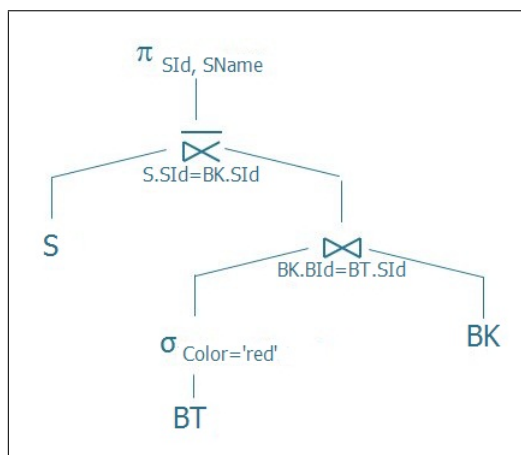
Alternative solution:



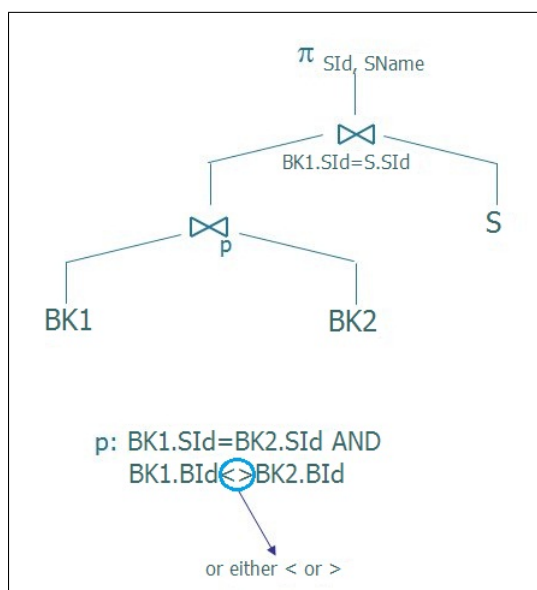
- (d) Find the codes and the names of the sailors who have never booked a red boat.



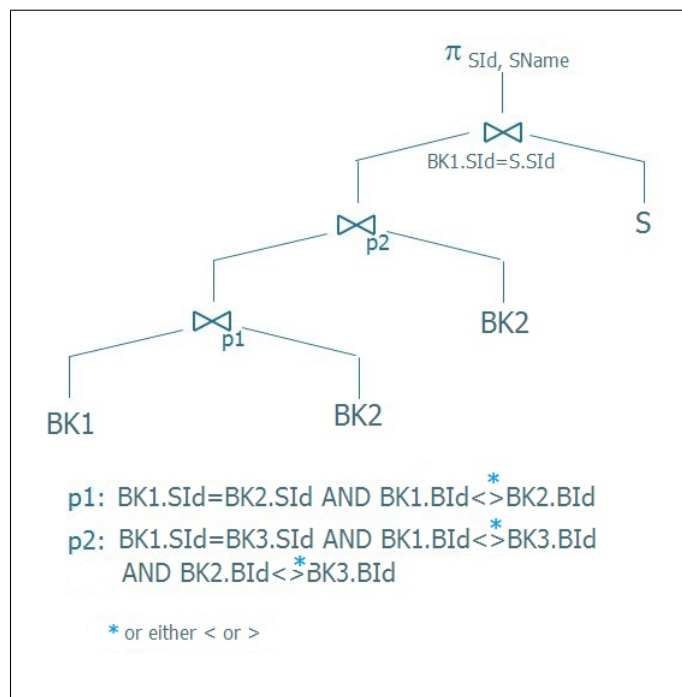
Alternative solution:



- (e) Find the codes and the names of the sailors who have booked at least two boats.



(f) Find the codes and the names of the sailors who have booked at least three boats.



Exercise 3. Given the relational schema including the following tables (primary keys are underlined):

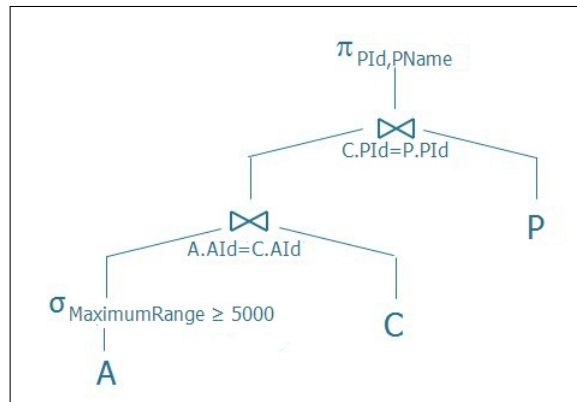
AIRCRAFT(AId, AName, MaximumRange)

CERTIFICATE(PIId, AId)

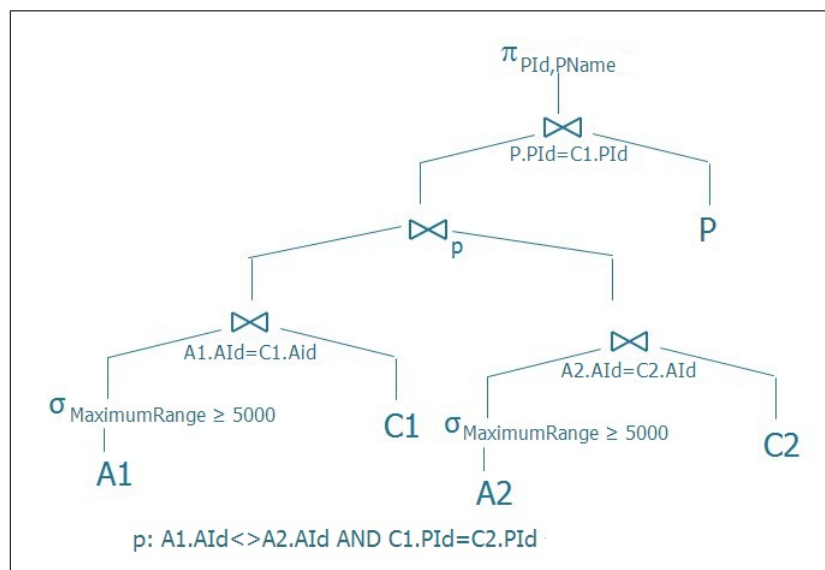
PILOT(PIId, PName, Salary)

express the following queries in relational algebra:

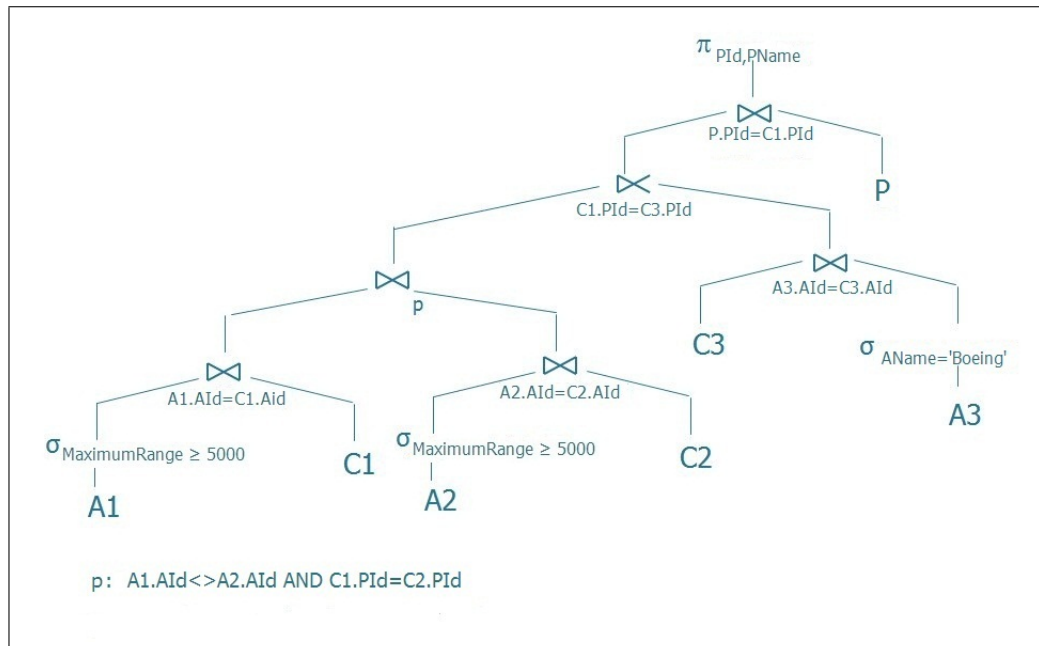
- (a) Find the codes and the names of the pilots who are qualified to fly on an aircraft that can cover distances greater than 5,000 km ($\text{MaximumRange} \geq 5,000$).



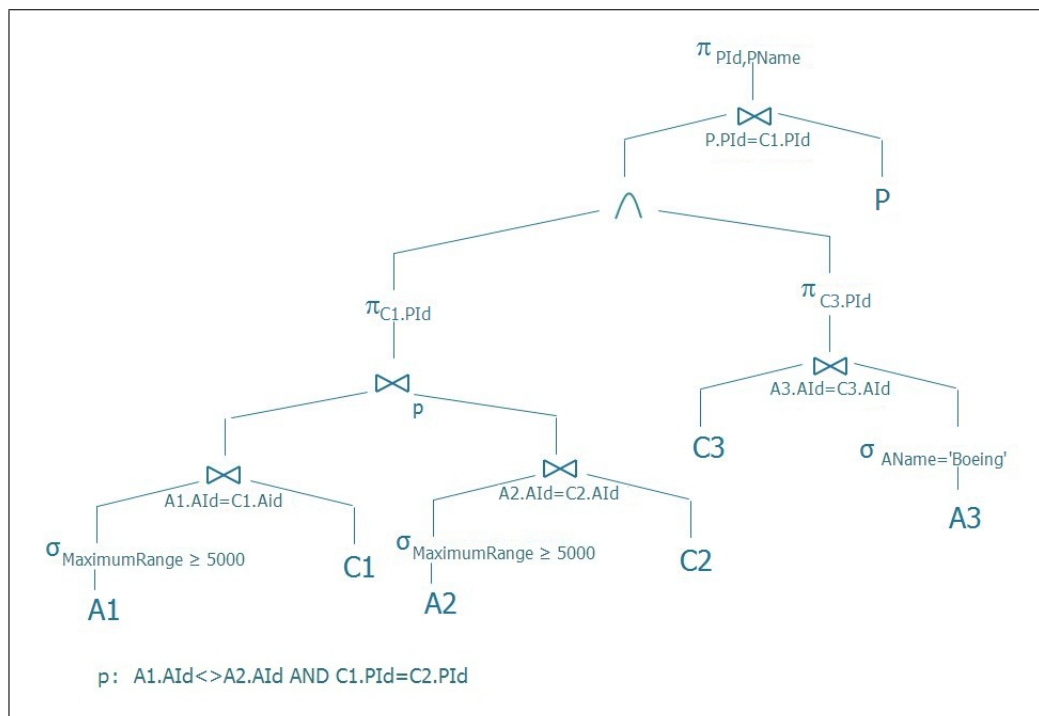
- (b) Find the codes and the names of the pilots who are qualified to fly on at least two aircrafts that can cover distances greater than 5,000 km.



- (c) Find the codes and the names of the pilots who are qualified to fly on at least two aircrafts that can cover distances greater than 5,000 km, and who are qualified to fly on a Boeing.



Alternative solution:



Databases - Exercise 3 : Relational Algebra

Due date: Thursday 5.11.15

- ★ The exercise should be submitted in the course box in Rothberg building, or via the submission link on the course homepage as pdf file, before 23:55.
- ★ Exercises handed in electronically must be typed (i.e., cannot be handwritten and then scanned).
- ★ Please write your name, id and login at the top of the submitted exercise.

Writing Relational Algebra Queries

You are given the following relations in a law firms database:

- ★ Client (Cid, Lid, Cname, birthYear, caseField)
- ★ Lawyer (Lid, Fname, Lname, Speciality, Salary, startingYear)
- ★ Firm (Fname, City, managerName)

Notes:

- ★ Cid and Cname refer to id and name of a client, respectively. Similarly, Lid and Lname refer to id and name of a lawyer, respectively. CaseField is a field of law which this client's case is about. StartingYear is the year when a lawyer started working for this law firm. Fname refers to a name of a law firm.
- ★ When we say that some client was presented by a law firm, we mean that this client was presented by some lawyer who works in this law firm.
- ★ The relation Lawyer contains information not only about the entities of type Lawyer, but also about the relationship between Lawyers and Firms (meaning, the law firms where lawyers are working).

Write the following queries in relational algebra:

1. Find names of clients that were born after 1980 and their case field was "Traffic".

solution:

$$\pi_{Cname} (\sigma_{birthYear > 1980 \wedge caseField = "Traffic"} (Client))$$

2. Find names of clients that in some case were presented by a lawyer that specializes in that case field.

solution:

$$\pi_{Cname} (\sigma_{Speciality = caseField} (Client \bowtie Lawyer))$$

3. Find case fields of clients that were presented at least once by a law firm that is managed by "Ally McBeal" or "Jessica Pearson".

solution:

$$\pi_{caseField} (Client \bowtie Lawyer \bowtie \pi_{Fname} (\sigma_{managerName = "Ally McBeal" \vee managerName = "Jessica Pearson"} (Firm)))$$

4. Find names of clients who were presented only by lawyers whose salary is at least 8000.

solution:

$$\pi_{Cname} (\pi_{Cid,Cname} (Client \bowtie Lawyer) - \pi_{Cid,Cname} (\sigma_{Salary < 8000} (Client \bowtie Lawyer)))$$

5. Find all pairs of client id and firm name such that no lawyer of that law firm presented this client, and this client had some case in a field other than "Divorce".

solution:

$$\pi_{Cid,Fname} (\sigma_{caseField \neq "Divorce"} (Client) \times \pi_{Fname} (Firm)) - \pi_{Cid,Fname} (Client \bowtie Lawyer)$$

6. Find all pairs of client id and lawyer name such that the client was presented by all lawyers with that name, and only by them.

solution: First, we find all pairs of client id and lawyer name, such that the client was presented by at least two lawyers with different names:

$$R = \pi_{Cid,Lname} (\pi_{Cid,Lname} (Client \bowtie Lawyer) \bowtie_{Cid=Cid2 \wedge Lname \neq Lname2} (\rho_{T(Cid2,Lname2)} (\pi_{Cid,Lname} (Client \bowtie Lawyer))))$$

Next, we find all pairs of client id and lawyer name, such that there is a lawyer with that name that did not present the client:

$$S = \pi_{Cid,Lname} (((\pi_{Cid} Client) \times (\pi_{Lid,Lname} Lawyer)) - \pi_{Cid,Lid,Lname} (Client \bowtie Lawyer))$$

So the final answer is:

$$(\pi_{Cid,Lname} (Client \bowtie Lawyer) - S) - R$$

Relational Algebra Equivalence

For each of the first seven parts, write if the two expressions are equivalent. Just answer **Yes/No**.

If there is no equivalence, is there containment between the two expression? Just write the **direction** of the containment. There is no need to prove your answer.

For each direction in which containment does not hold, give a **counterexample**.

X denotes a set of attributes (i.e., not just a single attribute), and all the expressions are well defined.

1. $R - \sigma_C (R)$ and $\sigma_{\neg C} (R)$ where $\neg C$ is the logical negation of C (it is assumed that $\neg C$ is transformed into the correct syntax)

solution: $R - \sigma_C (R) \equiv \sigma_{\neg C} (R)$

2. $(T - S) \cup (T - R)$ and $T - (S \cup R)$

solution: $(T - S) \cup (T - R) \supseteq T - (S \cup R)$. A counterexample for containment in the other direction is the following: T, R and S have 2 attributes: A, B. We take: $T = R = (1, 5)$, $S = (3, 4)$. So we get: $T - (S \cup R) = \emptyset$, but $(T - S) \cup (T - R) = (1, 5)$.

3. $\pi_X \sigma_C (R \cup S)$ and $\sigma_C (\pi_X (R) \cup \pi_X (S))$
(Assume that all the attributes mentioned in C appear also in X)

solution: $\pi_X \sigma_C (R \cup S) \equiv \sigma_C (\pi_X (R) \cup \pi_X (S))$

4. $\pi_X \sigma_C ((R \cup S) - S)$ and $\sigma_C (\pi_X (R \cup S) - \pi_X (S))$
(Assume that all the attributes mentioned in C appear also in X)

solution: $\sigma_C (\pi_X (R \cup S) - \pi_X (S)) \subseteq \pi_X \sigma_C ((R \cup S) - S)$. A counterexample for containment in the other direction is the following: R and S have 2 attributes: A, B. We take: $R = (10, 1), (20, 3), (30, 3)$, $S = (10, 1), (40, 3)$, $C = (B = 3)$, $X = B$. So we get: $\sigma_C (\pi_X (R \cup S) - \pi_X (S)) = \emptyset$, but $\pi_X \sigma_C ((R \cup S) - S) = (3)$.

5. $S \bowtie (R \cap T)$ and $(T \cap S) \bowtie R$
solution: $S \bowtie (R \cap T) \equiv (T \cap S) \bowtie R$
6. $S(A, B) \bowtie T(A, C)$ and $(\pi_B S(A, B)) \bowtie (S(A, B) \bowtie T(A, C))$
solution: $S(A, B) \bowtie T(A, C) \equiv (\pi_B S(A, B)) \bowtie (S(A, B) \bowtie T(A, C))$
7. $\pi_C (Q(A, B, C) \bowtie T(A, B, D))$ and $\pi_C (Q(A, B, C) \bowtie \pi_{B,D} T(A, B, D) \bowtie \pi_{A,D} T(A, B, D))$
solution: $\pi_C (Q(A, B, C) \bowtie T(A, B, D)) \subseteq \pi_C (Q(A, B, C) \bowtie \pi_{B,D} T(A, B, D) \bowtie \pi_{A,D} T(A, B, D))$.
A counterexample for containment in the other direction is the following: $Q = (1, 2, 3), T = (1, 4, 6), (5, 2, 6)$.
So we get: $\pi_C (Q(A, B, C) \bowtie T(A, B, D)) = \emptyset$, but $\pi_C (Q(A, B, C) \bowtie \pi_{B,D} T(A, B, D) \bowtie \pi_{A,D} T(A, B, D)) = (1, 2, 3, 6)$
8. Given the expression $\pi_{A,B,D} \sigma_{(A>3 \vee C=5) \wedge G \neq 7 \wedge D < 6 \wedge B > 4} (R(A, B, C) \bowtie S(A, B, D, E, F) \bowtie T(A, D, F, G))$
write an equivalent expression in which selection and projection are pushed inside as much as possible
(that is, are applied as early as possible).
solution: First, we push all selections as much as possible:

$$\pi_{A,B,D} (\sigma_{(A>3 \vee C=5) \wedge B > 4} R(A, B, C) \bowtie \sigma_{D < 6 \wedge B > 4} S(A, B, D, E, F) \bowtie \sigma_{G \neq 7 \wedge D < 6} T(A, D, F, G))$$

Then we push projections:

$$\pi_{A,B,D} ((\pi_{A,B} \sigma_{(A>3 \vee C=5) \wedge B > 4} R(A, B, C)) \bowtie (\pi_{A,B,D,F} \sigma_{D < 6 \wedge B > 4} S(A, B, D, E, F)) \bowtie (\pi_{A,D,F} \sigma_{G \neq 7 \wedge D < 6} T(A, D, F, G)))$$

Good luck !

Relational-algebra exercises

Appendix to Lecture 3

Running example: Movies database

Movie (title, year, length, inColor, studioName, producerC)

MovieStar (name, address, gender, birthdate)

StarsIn (movieTitle, movieYear, starName)

MovieExec (name, address, cert, netWorth)

Studio (studioName, presc);

Movies

SIMPLE QUERIES

Selections: Movies

1. Find titles of all black-and-white movies which were produced after 1970
2. Find titles of all movies produced by MGM studio after 1970 or with length less than 1.5 hours
3. Find producer of 'Star wars'

Projections: Movies

- 4. Info about all Disney movies produced in year 1990
- 5. Title and length of all Disney movies produced in year 1990
- 6. Title and length in hours of all Disney movies produced in year 1990

Joins: Movies

7. For each movie's title produce the name of this movie's producer
8. Find the names of producers of movies where Harrison Ford starred.

Movie (title, year, length, inColor, studioName, producerC)

MovieStar (name, address, gender, birthdate)

StarsIn (movieTitle, movieYear, starName)

MovieExec (name, address, cert, netWorth)

Studio (studioName, presc);

9. Find all name pairs in form (movie star, movie producer) that live at the same address.

Star = $\rho_{\text{star, staraddress}} (\pi_{\text{name, address}} (\text{MovieStar}))$

Prod = $\rho_{\text{prod, prodaddress}} (\pi_{\text{name, address}} (\text{MovieExec}))$

$\pi_{\text{star, prod}} ((\text{Star}) \bowtie_{\text{staraddress=prodaddress AND star != prod}} (\text{Prod}))$

Movies

MORE COMPLEX QUERIES

Movie (title, year, length, inColor, studioName, producerC)

MovieStar (name, address, gender, birthdate)

StarsIn (movieTitle, movieYear, starName)

MovieExec (name, address, cert, netWorth)

Studio (studioName, presc);

10. Find the names of all producers who did NOT produce 'Star wars'

➤ Simple:

$\pi_{\text{name}}(\text{MovieExec}) -$

$\pi_{\text{name}}((\text{Movie}) \bowtie_{\text{title='Star wars' AND producerC=cert}}(\text{MovieExec}))$


➤ More efficient (smaller Cartesian product)

$\pi_{\text{name}}((\sigma_{\text{title='Star wars'}}(\text{Movie})) \bowtie_{\text{producerC!=cert}}(\text{MovieExec}))$

****9B.** Find all name pairs in form (movie star, movie producer) that live at the same address. Now, try to eliminate palindrome pairs: leave (a,b) but not both (a,b) and (b,a).

1. $\text{Star} = \rho_{\text{name} \rightarrow \text{star}}(\text{MovieStar})$
 $\text{Prod} = \rho_{\text{name} \rightarrow \text{prod}}(\text{MovieExec})$
2. $\text{Pairs} = \pi_{\text{star,prod}}((\text{Star}) \bowtie_{\text{Star.address}=\text{Prod.address AND star} \neq \text{prod}}(\text{Prod}))$
3. $\text{PA} = \sigma_{\text{star} < \text{prod}}(\text{Pairs})$ // Pairs in **A**scending order
 $\text{PD} = \sigma_{\text{star} > \text{prod}}(\text{Pairs})$ // Pairs in **D**escending order
4. $\text{Palindrome} = (\text{PA}) \bowtie_{\text{PA.star}=\text{PD.prod AND PA.prod}=\text{PD.star}}(\text{PD})$
5. $\text{Pairs} - \pi_{\text{PD.star,PD.prod}}(\text{Palindrome})$

Example on
the next page



1. Renaming

Star	
star	addr
A	1
B	1
C	2
F	3

Prod	
prod	addr
A	1
B	1
D	2
E	3

1
Star= $\rho_{\text{name} \rightarrow \text{star}}$ (MovieStar)
Prod= $\rho_{\text{name} \rightarrow \text{prod}}$ (MovieExec)

Star	Addr	Prod	Addr
A	1	A	1
A	1	B	1
A	1	D	2
A	1	E	3
B	1	A	1
B	1	B	1
B	1	D	2
B	1	E	3
C	2	A	1
C	2	B	1
C	2	D	2
C	2	E	3
F	3	A	1
F	3	B	1
F	3	D	2
F	3	E	3

2. Cartesian product: Star x Prod

2. Pairs = $\pi_{\text{star,prod}}$

((Star)

$\bowtie_{\text{Star.address=Prod.address AND star!=prod}}$
(Prod))

Pairs	
Star	Prod
A	B
B	A
C	D
F	E

3. Sorted pairs

Pairs	
Star	Prod
A	B
B	A
C	D
F	E

3. $PA = \sigma_{\text{star} < \text{prod}}(\text{Pairs})$ // Pairs in **A**scending
 $PD = \sigma_{\text{star} > \text{prod}}(\text{Pairs})$ // Pairs in **D**escending

PA	
Star	Prod
A	B
C	D

PD	
Star	Prod
B	A
F	E

4. Cartesian product PA x PD

PA	
Star	Prod
A	B
C	D

x

PD	
Star	Prod
B	A
F	E

Palyndrome (only colored tuple qualify)			
PA.Star	PA.Prod	PD.Star	PD.Prod
A	B	B	A
A	B	F	E
C	D	B	A
C	D	F	E

4. Palindrome = (PA)

⋈_{PA.star=PD.prod AND PA.prod=PD.star}
(PD)

5. Remove palindrome tuples from pairs

5. Pairs – $\pi_{PD.star, PD.prod}(\text{Palindrome})$

Pairs	
Star	Prod
A	B
B	A
C	D
F	E

-

Palyndrome			
PA.Star	PA.Prod	PD.Star	PD.Prod
A	B	B	A

result	
Star	Prod
A	B
C	D
F	E

Movie (title, year, length, inColor, studioName, producerC)

MovieStar (name, address, gender, birthdate)

StarsIn (movieTitle, movieYear, starName)

MovieExec (name, address, cert, netWorth)

Studio (studioName, presc);

11. Find names of producers that produced at least one movie for each of different studios: Disney and MGM

$\pi_{\text{name}}[(\sigma_{\text{studioName}='Disney'}(\text{Movie})) \bowtie_{\text{producerC=cert}}(\text{MovieExec})]$

\wedge

$\pi_{\text{name}}[(\sigma_{\text{studioName}='MGM'}(\text{Movie})) \bowtie_{\text{producerC=cert}}(\text{MovieExec})]$

Movie (title, year, length, inColor, studioName, producerC)

MovieStar (name, address, gender, birthdate)

StarsIn (movieTitle, movieYear, starName)

MovieExec (name, address, cert, netWorth)

Studio (studioName, presc);

12. Find all movie titles for which there is no producer entry in MovieExec table

$\pi_{\text{title}}(\text{Movie}) - \pi_{\text{title}}((\text{Movie}) \bowtie_{\text{producerC=cert}} (\text{MovieExec}))$

Movie (title, year, length, inColor, studioName, producerC)

MovieStar (name, address, gender, birthdate)

StarsIn (movieTitle, movieYear, starName)

MovieExec (name, address, cert, netWorth)

Studio (studioname, presc);

13. Find the names of all stars which starred in at least 2 movies (according to our database)

1. $S1 = \rho_{\text{title1, year1, name1}}(\text{StarsIn})$

$S2 = \rho_{\text{title2, year2, name2}}(\text{StarsIn})$

2. $(S1) \bowtie_{\text{name1=name2 AND (title1 \neq title2 or year1 \neq year2)}} (S2)$

Lab database: Pizza

Person (name, age, gender)

Frequents (name, pizzeria)

Eats (name, pizza)

Serves (pizzeria, pizza, price)

Pizza

TEST YOURSELF ON SIMPLE QUERIES

Projections: Pizza

1. Find full information about all possible places and prices to get mushroom or pepperoni pizzas
2. Find name of pizzerias that serve mushroom or pepperoni pizzas
3. Compute the full list of pizza types, with the corresponding pizzerias and the price of pizza in cents

Selections: Pizza

4. Find names of all customers under 18
5. Find names of all female customers older than 25

Join: Pizza

6. Find all pizza types that both Amy and Dan eat
7. Find the names of all females who eat a mushroom pizza
8. Find the names of pizzerias where Hil can buy pizzas she eats for less than 10\$

Person (name, age, gender)

Frequents (name, pizzeria)

Eats (name, pizza)

Serves (pizzeria, pizza, price)

9. Find the names of all females who eat either mushroom or pepperoni pizza (or both).

$\pi_{\text{name}} (\sigma_{\text{gender}='female' \text{ AND } (\text{pizza}='mushroom' \text{ OR } \text{pizza}='pepperoni') } (\text{Person} \bowtie \text{Eats}))$

Person (name, age, gender)

Frequents (name, pizzeria)

Eats (name, pizza)

Serves (pizzeria, pizza, price)

10. Find the names of all females who eat both mushroom and pepperoni pizza.

$$\pi_{\text{name}}(\sigma_{\text{gender}='female' \text{ AND } \text{pizza}='mushroom'}(\text{Person} \bowtie \text{Eats}))$$
$$\cap$$
$$\pi_{\text{name}}(\sigma_{\text{gender}='female' \text{ AND } \text{pizza}='pepperoni'}(\text{Person} \bowtie \text{Eats}))$$

Person (name, age, gender)

Frequents (name, pizzeria)

Eats (name, pizza)

Serves (pizzeria, pizza, price)

11. Find all pizzerias that serve at least one pizza that Amy eats for less than \$10.00.

$\pi_{\text{pizzeria}}(\sigma_{\text{name}='Amy'}(\text{Eats}) \bowtie \sigma_{\text{price} < 10}(\text{Serves}))$

Person (name, age, gender)

Frequents (name, pizzeria)

Eats (name, pizza)

Serves (pizzeria, pizza, price)

12. Find all pizzerias frequented by at least one person under the age of 18.

$$\pi_{\text{pizzeria}}(\sigma_{\text{age} < 18}(\text{Person}) \bowtie \text{Frequents})$$

Person (name, age, gender)

Frequents (name, pizzeria)

Eats (name, pizza)

Serves (pizzeria, pizza, price)

13. Find all pizza types which are not eaten by anyone

$\pi_{\text{pizza}}(\text{Serves}) - \pi_{\text{pizza}}(\text{Eats})$

Person (name, age, gender)

Frequents (name, pizzeria)

Eats (name, pizza)

Serves (pizzeria, pizza, price)

14. Find all pizzerias that are frequented by only females or only males.

$$\left[\begin{array}{l} \pi_{\text{pizzeria}}(\sigma_{\text{gender}='female'}(\text{Person}) \bowtie \text{Frequents}) - \\ \pi_{\text{pizzeria}}(\sigma_{\text{gender}='male'}(\text{Person}) \bowtie \text{Frequents}) \end{array} \right] \cup$$

$$\left[\begin{array}{l} \pi_{\text{pizzeria}}(\sigma_{\text{gender}='male'}(\text{Person}) \bowtie \text{Frequents}) - \\ \pi_{\text{pizzeria}}(\sigma_{\text{gender}='female'}(\text{Person}) \bowtie \text{Frequents}) \end{array} \right]$$

Person (name, age, gender)

Frequents (name, pizzeria)

Eats (name, pizza)

Serves (pizzeria, pizza, price)

15. Find all pizzerias where Dan could buy pizzas that he eats, and where he has never bought a pizza yet

$\pi_{\text{pizzeria}} [(\sigma_{\text{name}='Dan'}(\text{Eats})) \bowtie (\text{Serves})]$

-

$\pi_{\text{pizzeria}} (\sigma_{\text{name}='Dan'}(\text{Frequents}))$

Person (name, age, gender)

Frequents (name, pizzeria)

Eats (name, pizza)

Serves (pizzeria, pizza, price)

16. For each person, find all pizzas the person eats that are not served by any pizzeria the person frequents. Return all such person (name) / pizza pairs.

$\text{Eats} - \pi_{\text{name, pizza}}(\text{Frequents} \bowtie \text{Serves})$

Person (name, age, gender)

Frequents (name, pizzeria)

Eats (name, pizza)

Serves (pizzeria, pizza, price)


17. Find the names of all people who frequent only pizzerias serving at least one pizza they eat.

$\pi_{\text{name}}(\text{Person})$

–

$\pi_{\text{name}}(\text{Frequents} - \pi_{\text{name,pizzeria}}(\text{Eats} \bowtie \text{Serves}))$

Explanation
on the next
page



17. Find the names of all people who frequent only pizzerias serving at least one pizza they eat.

- 1. List of all pizzerias which serve at least one of pizzas which particular person can eat:

$\pi_{\text{name,pizzeria}}(\text{Eats} \bowtie \text{Serves})$

- 2. List of all pizzerias which are frequented by this person but do not serve any pizza he can it

Frequents - $\pi_{\text{name,pizzeria}}(\text{Eats} \bowtie \text{Serves})$

- 3. Answer to the query

$\pi_{\text{name}}(\text{Person})$

–

$\pi_{\text{name}}(\text{Frequents} - \pi_{\text{name,pizzeria}}(\text{Eats} \bowtie \text{Serves}))$

Person (name, age, gender)

Frequents (name, pizzeria)

Eats (name, pizza)

Serves (pizzeria, pizza, price)


18. Find the names of all people who frequent every pizzeria serving at least one pizza they eat.

$\pi_{\text{name}}(\text{Person})$

–

$\pi_{\text{name}}(\pi_{\text{name,pizzeria}}(\text{Eats} \bowtie \text{Serves}) - \text{Frequents})$

Explanation
on the next
page



18. Find the names of all people who frequent every pizzeria serving at least one pizza they eat.

➤ 1. List of all pizzerias per person which serve at least one pizza this person can eat:

$\pi_{\text{name,pizzeria}}(\text{Eats} \bowtie \text{Serves})$

➤ 2. List of pizzerias which serve the desirable pizza but which person did not visit yet

$\pi_{\text{name,pizzeria}}(\text{Eats} \bowtie \text{Serves}) - \text{Frequents}$

➤ 3. All the people excluding those in p.2

$\pi_{\text{name}}(\text{Person})$

–

$\pi_{\text{name}}(\pi_{\text{name,pizzeria}}(\text{Eats} \bowtie \text{Serves}) - \text{Frequents})$

Person (name, age, gender)
 Frequents (name, pizzeria)
 Eats (name, pizza)
 Serves (pizzeria, pizza, price)

19. Find the pizzeria serving the cheapest pepperoni pizza. In the case of ties, return all of the cheapest-pepperoni pizzerias.

$\pi_{\text{pizzeria}}(\sigma_{\text{pizza}='pepperoni'} \text{Serves})$


—

$\pi_{\text{pizzeria}} [\sigma_{\text{price} > \text{price2}} ($
 $\pi_{\text{pizzeria}, \text{price}} (\sigma_{\text{pizza}='pepperoni'} \text{Serves})$

×

$\rho_{\text{pizzeria2}, \text{price2}} \pi_{\text{pizzeria}, \text{price}} (\sigma_{\text{pizza}='pepperoni'} \text{Serves}))]$

Explanation
on the next
page



19. Find the pizzeria serving the cheapest pepperoni pizza. In the case of ties, return all of the cheapest-pepperoni pizzerias.

- 1. Finds all pizzerias where price for pepperoni pizza is greater than in some other pizzeria

$$\sigma_{\text{price} > \text{price2}} (\pi_{\text{pizzeria}, \text{price}} (\sigma_{\text{pizza} = \text{'pepperoni'}} \text{Serves})$$

×

$$\rho_{\text{pizzeria2}, \text{price2}} [\pi_{\text{pizzeria}, \text{price}} (\sigma_{\text{pizza} = \text{'pepperoni'}} \text{Serves})]$$

- 2. Subtracts it from all other pizzerias serving pepperoni pizzas

$$\pi_{\text{pizzeria}} (\sigma_{\text{pizza} = \text{'pepperoni'}} \text{Serves})$$

−

$$\pi_{\text{pizzeria}} [\sigma_{\text{price} > \text{price2}} (\pi_{\text{pizzeria}, \text{price}} (\sigma_{\text{pizza} = \text{'pepperoni'}} \text{Serves})$$

×

$$\rho_{\text{pizzeria2}, \text{price2}} \pi_{\text{pizzeria}, \text{price}} (\sigma_{\text{pizza} = \text{'pepperoni'}} \text{Serves})]$$

Relational Algebra Exercises

Pubs Database Schema

author(author_id, first_name, last_name)

author_pub(author_id, pub_id, author_position)

book(book_id, book_title, month, year, editor)

pub(pub_id, title, book_id)

- *author_id* in *author_pub* is a foreign key referencing *author*
- *pub_id* in *author_pub* is a foreign key referencing *pub*
- *book_id* in *pub* is a foreign key referencing *book*
- *editor* in *book* is a foreign key referencing *author*(*author_id*)
- Primary keys are underlined

Pubs Database State

r(author)

author_id	first_name	last_name
1	John	McCarthy
2	Dennis	Ritchie
3	Ken	Thompson
4	Claude	Shannon
5	Alan	Turing
6	Alonzo	Church
7	Perry	White
8	Moshe	Vardi
9	Roy	Batty

r(author_pub)

author_id	pub_id	author_position
1	1	1
2	2	1
3	2	2
4	3	1
5	4	1
5	5	1
6	6	1

r(book)

book_id	book_title	month	year	editor
1	CACM	April	1960	8
2	CACM	July	1974	8
3	BST	July	1948	2
4	LMS	November	1936	7
5	Mind	October	1950	NULL
6	AMS	Month	1941	NULL
7	AAAI	July	2012	9
8	NIPS	July	2012	9

r(pub)

pub_id	title	book_id
1	LISP	1
2	Unix	2
3	Info Theory	3
4	Turing Machines	4
5	Turing Test	5
6	Lambda Calculus	6

Figure 1: Relational Database Schema

1. How many tuples will be returned by the following relational algebra query?

$$\pi_{book_title}(book)$$

Solution: 7

2. What question does the following expression answer?

$$|\pi_{author_id}(author) - \pi_{editor}(book)|$$

Solution: How many authors are not book editors.

3. Write a relational algebra expression that returns the names of all authors who are book editors.

Solution: $\pi_{first_name, last_name}(author \bowtie_{author_id=editor} book)$

4. Write a relational algebra expression that returns the names of all authors who are **not** book editors.

Solution: $\pi_{first_name, last_name}((\pi_{author_id}(author) - \pi_{editor}(book)) * author)$

5. Write a relational algebra expression that returns the names of all authors who have at least one publication in the database.

Solution: $\pi_{first_name, last_name}(author * author_pub)$

6. How many tuples are returned by the following relational algebra expression?

$$author \bowtie_{author_id=editor} book$$

Solution: 11

7. What question does the following relational algebra expression answer?

$$author * (author_pub * (\sigma_{month='July'}(book) * pub))$$

Solution: Which authors authored a pub that was published in July?

CS 2541: In-Class Exercises 2. Relational Algebra and Calculus Queries.

- **lives**(person-name,street,city)
- **works**(person-name, company-name,salary)
- **located-in**(company-name,city)
- **manages**(person-name,manager-name)

For the above schema (the primary key for each relation is denoted by the underlined attribute), provide relational algebra expressions for the following queries:

Note: For notational convenience, I am using pname instead of person-name, cname instead of company-name, and mname instead of manager-name. In addition, instead of $lives[pname]$ i am using the equivalent notation $lives.pname$ in an SQL like syntax in some queries – you can use either.

1. Find all tuples in works of all persons who work for the City Bank company (which is a specific company in the database).

- (a) $\sigma_{(cname='City Bank')}(works)$
- (b) $\{p | p \in works \wedge (p[cname] = 'City Bank')\}$

2. Find the name of persons working at City Bank who earn more than \$50,000.

- (a) $\pi_{pname}(\sigma_{(cname='City Bank') \wedge (salary > 50000)}(works))$
- (b) $\{p | \exists w \in works ((p[pname] = w[pname]) \wedge (w[cname] = 'City Bank'))\}$

Since p only has attribute of pname in the query, its 'type' is a single attribute (person-name).

Notation options: The solutions for relational calculus use the notation tuplevariable[attribute] (for example $w[pname]$); you could also use the easier to read notation tuplevariable.attribute (for example $w.pname$).

3. Find the name and city of all persons who work for City Bank and earn more than 50,000. Similar to previous query, except we have to access the lives table to extract the city of the employee. Note the join condition in the query.

- (a) $\pi_{lives.pname, lives.city}(\sigma_{((cname='City Bank') \wedge (salary > 50000) \wedge (lives.pname = works.pname))}(lives \times works))$

We can use the equijoin operator *bowtie* instead of the cross product and σ operators and write the query as: $\pi_{lives.pname, lives.city}(\sigma_{((cname='City Bank') \wedge (salary > 50000))}(lives \bowtie_{lives.pname = works.pname} works))$

(b) In the query below, the tuple p (the only free variable) refers only to attributes pname and city.

$$\{p | \exists w \in works, \exists l \in lives \\ ((w[cname] = 'City Bank') \wedge (w[salary] > 50000) \wedge (w[pname] = l[pname]) \\ \wedge (p[pname] = w[pname]) \wedge (p[city] = l[city]))\}$$

This could also be written as (i.e., the quantifiers are 'nested')

$$\{p \mid \exists w \in \text{works}((p[pname] = w[pname]) \wedge (w[cname] = 'City Bank') \wedge (w[salary] > 50000) \wedge (\exists l \in \text{lives}(l[pname] = w[pname]) \wedge (p[city] = l[city])))\}$$

4. Find names of all persons who live in the same city as the company they work for. For this query we need to access the lives table to get city of the employee and the located-in table to get city of the company; plus the works table to associate employee with their company. The selection condition is then that the two cities are the same.

(a)

$$\pi_{\text{lives.pname}} (\sigma_{((\text{located-in.city}=\text{works.cname}) \wedge (\text{located-in.city}=\text{lives.city}) \wedge (\text{lives.pname}=\text{works.pname}))} (\text{works} \times \text{lives} \times \text{located-in}))$$

Using the equijoin operator, we can write the above query as:

$$\pi_{\text{lives.pname}} (\sigma_{(\text{located-in.city}=\text{lives.city})} ((\text{works} \bowtie_{(\text{lives.pname}=\text{works.pname})} \text{lives}) \bowtie_{\text{works.cname}=\text{located-in.cname}} \text{located-in}))$$

(b)

$$\{p \mid \exists w \in \text{works}, \exists l \in \text{lives} \exists y \in \text{located-in} ((l.city = y.city) \wedge (w.cname = y.cname) \wedge (w.pname = l.pname) \wedge (p.pname = l.pname))\}$$

5. Find names of all persons who live in the same city and on the same street as their manager. This requires accessing lives table twice – once for finding city of employee and a second time for finding city of manager. Therefore we need two variables from lives with different names; one will refer to employee and the other will refer to the manager. So create a copy of lives, called mlives, which we use to find the address of the manager.

(a)

$$\pi_{\text{lives.pname}} (\sigma_{((\text{lives.city}=\text{mlives.city}) \wedge (\text{lives.street}=\text{mlives.street}) \wedge (\text{manages.pname}=\text{lives.pname}) \wedge (\text{mname}=\text{mlives.pname}))} (\text{lives} \times \text{manages} \times (\rho_{\text{mlives}}(\text{lives}))))$$

(b)

$$\{x \mid \exists y \in \text{lives}, \exists z \in \text{lives} \exists m \in \text{manages} \\ ((z.\text{city} = y.\text{city}) \wedge (y.\text{street} = z.\text{street}) \wedge \\ (y.\text{pname} = m.\text{pname}) \wedge (z.\text{pname} = m.\text{managername}) \wedge (x.\text{pname} = y.\text{pname}))\}$$

6. Find names of all persons who do not work for City Bank. Can write this in multiple ways
- one solution is to use set difference:

$$\begin{aligned} \text{(a)} & (\pi_{\text{pname}}(\text{works})) - (\pi_{\text{pname}}(\sigma_{\text{cname}='City Bank'}(\text{works}))) \\ \text{(b)} & \{x \mid \exists y \in \text{works} (x.\text{pname} = y.\text{pname} \wedge y.\text{cname} \neq 'City Bank')\} \end{aligned}$$

7. Find the name of all persons who work for City Bank and live in DC. Similar to query 3, but select only with tuples where person city is DC.

$$\begin{aligned} \text{(a)} & \pi_{\text{lives.pname}}(\sigma_{((\text{cname}='City Bank') \wedge (\text{lives.city}='DC') \wedge (\text{lives.pname}=\text{works.pname}))})(\text{lives} \times \text{works}) \\ \text{(b)} & \end{aligned}$$

$$\begin{aligned} \{p \mid \exists w \in \text{works}, \exists l \in \text{lives} \\ ((w[\text{cname}] = 'City Bank') \wedge (l[\text{city}] = 'DC') \wedge (w[\text{pname}] = l[\text{pname}]) \\ \wedge (p[\text{pname}] = w[\text{pname}]))\} \end{aligned}$$