Relational Algebra Problems

Question:

Consider the following relational database schema consisting of the four relation schemas:

passenger (pid, pname, pgender, pcity)

agency (aid, aname, acity)

flight (fid, fdate, time, src, dest)

booking (pid, aid, fid, fdate)

Answer the following questions using relational algebra queries;

a) Get the complete details of all flights to New Delhi.

b) Get the details about all flights from Chennai to New Delhi.

c) Find only the flight numbers for passenger with pid 123 for flights to Chennai before 06/11/2020.

d) Find the passenger names for passengers who have bookings on at least one flight.

e) Find the passenger names for those who do not have any bookings in any flights.

f) Find the agency names for agencies that located in the same city as passenger with passenger id 123.

g) Get the details of flights that are scheduled on both dates 01/12/2020 and 02/12/2020 at 16:00 hours.

h) Get the details of flights that are scheduled on either of the dates 01/12/2020 or 02/12/2020 or both at 16:00 hours.

i) Find the agency names for agencies who do not have any bookings for passenger with id 123.

j) Find the details of all male passengers who are associated with Jet agency.

Solution:

Relational algebra operators:

σ – selection with conditions (It selects all tuples that satisfies the conditions. Shows entire table with respect to the structure)

Π – projection operator (It selects the attributes which are listed here)

⋈ - natural join operator (Binary operator that join two relations on common attributes' values)

-, ∪, and ∩ - set operators (difference, union and intersection)

Most of the following queries can be written in many different ways.

a) Get the complete details of all flights to New Delhi.

$$\sigma_{destination = \text{“New Delhi”}}(\text{flight})$$

b) Get the details about all flights from Chennai to New Delhi.

$$\sigma_{src = \text{“Chennai”} \wedge dest = \text{“New Delhi”}}(\text{flight})$$

c) Find only the flight numbers for passenger with pid 123 for flights to Chennai before 06/11/2020.

$$\Pi_{fid}(\sigma_{pid = 123}(\text{booking}) \bowtie \sigma_{dest = \text{“Chennai”} \wedge fdate < 06/11/2020}(\text{flight}))$$

[**Hint:** ***Given conditions are pid, dest, and fdate. To get the flight id for a passenger given a pid, we have two tables flight and booking to be joined with necessary conditions. From the result, the flight id can be projected***]

d) Find the passenger names for passengers who have bookings on at least one flight.

$$\Pi_{pname}(\text{passenger} \bowtie \text{booking})$$

e) Find the passenger names for those who do not have any bookings in any flights.

$$\Pi_{pname}((\Pi_{pid}(\text{passenger}) - \Pi_{pid}(\text{booking})) \bowtie \text{passenger})$$

[**Hint:** ***here applied a set difference operation. The set difference operation returns only pids that have no booking. The result is joined with passenger table to get the passenger names.***]

f) Find the agency names for agencies that located in the same city as passenger with passenger id 123.

$$\Pi_{aname}(\text{agency} \bowtie_{acity = pcity}(\sigma_{pid = 123}(\text{passenger})))$$

[**Hint:** ***we performed a theta join on equality conditions (equi join) here. This is done between details of passenger 123 and the agency table to get the valid records where the city values are same. From the results, aname is projected.***]

g) Get the details of flights that are scheduled on both dates 01/12/2020 and 02/12/2020 at 16:00 hours.

$$(\sigma_{fdate = 01/12/2020 \wedge time = 16:00}(\text{flight})) \cap (\sigma_{fdate = 02/12/2020 \wedge time = 16:00}(\text{flight}))$$

[**Hint:** ***the requirement is for flight details for both dates in common. Hence, set intersection is used between the temporary relations generated from application of various conditions.***]

h) Get the details of flights that are scheduled on either of the dates 01/12/2020 or 02/12/2020 or both at 16:00 hours.

$$(\sigma_{fdate = 01/12/2020 \wedge time = 16:00}(\text{flight})) \cup (\sigma_{fdate = 02/12/2020 \wedge time = 16:00}(\text{flight}))$$

i) Find the agency names for agencies who do not have any bookings for passenger with id 123.

$$\Pi_{aname}(\text{agency} \bowtie (\Pi_{\text{aid}}(\text{agency}) - \Pi_{\text{aid}}(\sigma_{pid = 123}(\text{booking}))))$$

j) Find the details of all male passengers who are associated with Jet agency.

$$\Pi_{passengers.pid,\ pname,\ pcity}(\sigma_{pgender = \text{“Male”} \wedge aname = \text{‘Jet’}}(\text{passengers} \bowtie \text{booking} \bowtie \text{agency}))$$

[**Hint:** ***To get the link between passengers and agency, we need to join all three tables passengers, booking, and agency with necessary condition. Here, agency links both passengers and agency. As we have performed natural join operation between all three tables, the degree of the result will consist of all attributes from all the three tables. Hence, we project only passengers details as these are mentioned as required.***]

****

Player relation

Player Id	Team Id	Country	Age	Runs	Wickets
1001	101	India	25	10000	300
1004	101	India	28	20000	200
1006	101	India	22	15000	150
1005	101	India	21	12000	400
1008	101	India	22	15000	150
1009	103	England	24	6000	90
1010	104	Australia	35	1300	0
1011	104	Australia	29	3530	10
1012	105	Pakistan	28	1421	166
1014	105	Pakistan	21	3599	205

Deposit relation

Acc. No.	Cust-name
A 231	Rahul

A 432	Omkar
R 321	Sachin
S 231	Raj
T 239	Sumit

Borrower relation

Loan No.	Cust-name
P-3261	Sachin
Q-6934	Raj
S-4321	Ramesh
T-6281	Anil

R-Schema(id, name)

R – Relation

Id	Name
101	Raj
102	Rahul
103	Sachin
104	Anil
105	Prasad

S-Schema(id, name)

S – Relation

Id	Name
101	Raj
104	Anil
106	Kapil
107	Sumit

(1) Select Operation (σ)

The select operation selects the tuples (rows) that satisfy the given predicate (condition). The lower case Greek letter Sigma (σ) is used to represent the select operation.

The predicate appears as a subscript to σ and argument relation is given in parenthesis following σ. Predicates can be defined using the operators =, !=, <=, <, >, >= etc. and they may be connected by using the connectives.

Notation of Selection Operation

σp(r)

Where,
σ is predicate,
r stands for relation (name of the table).
p is the prepositional logic.

Exercises –

Question A. Find all tuples from player relation for which country is India.

Solution – Use Query:

σ“country” = “India”(Player)

Result –

Player Id	Team Id	Country	Age	Runs	Wickets
1001	101	India	25	10000	300
1004	101	India	28	20000	200
1006	101	India	22	15000	150
1005	101	India	21	12000	400
1008	101	India	22	15000	150

Question B. Select all the tuples for which runs are greater than or equal to 15000.

Solution – Use Query:

σ“runs” >= “15000”(Player)

Result –

Player Id	Team Id	Country	Age	Runs	Wickets
1004	101	India	28	20000	200
1006	101	India	22	15000	150
1008	101	India	22	15000	150

Question C. Select all the players whose runs are greater than or equal to 6000 and age is less than 25

Solution – Use Query:

σ“runs” > “6000” ^ age < 25(Player)

Result –

Player Id	Team Id	Country	Age	Runs	Wickets
1006	101	India	22	15000	150
1005	101	India	21	12000	400
1008	101	India	22	15000	150
1009	103	England	24	6000	90

(2) Project Operation (π)

Projection of a relation P (P-Schema) on the set of attributes Y is the projection of each tuple of the relation P on the set of attributes Y.

The projection operation is a unary operation and it returns its argument relation with certain attributes left out. It is denoted by a Greek letter pi (π). The attributes, which appear in the result, are listed as a subscript to π.

Notation of Project Operation

πA1, A2, An(r)

Where,
A1, A2, An are attribute name of the relation r.

Exercises –

a. List all the countries in Player relation.

Solution – Use Query:

π“country”(Player)

Result –

Country
India
England

Australia

Pakistan

b. List all the team ids and countries in Player Relation

Solution – Use Query

πTeam Id, Country (Player)

Result –

Team Id	Country
101	India
103	England
104	Australia
105	Pakistan

(3) Union Operation (∪)

Compatible relations: Two relations R and S are said to be compatible relations if they satisfy following two conditions –

1. The relations R and S are of same entity i.e. the number of attributes are same.
2. The domains of the ith attribute of R and ith attribute of S must be same for all i.

The union of R and S is set theoretic union of R and S, if R and S are compatible relations.

It is denoted by ∪, the resultant relation P(P=R ∪ S) has tuples drawn from R and S such that a tuple in P is either in R or S or in both of them.

Notation of Union Operation

P = R ∪ S

Where,
R and S are relations.

Example – 1: P = R ∪ S is given by relation

Id	Name
101	Raj
102	Rahul

103 Sachin

104 Anil

105 Prasad

106 Kapil

107 Sumit

(4) Set Difference Operation (-)

The set difference operation removes common tuples from the first relation. It is denoted by '-' sign. The expression R-S results in a relation containing those tuples in R but not in S. For set difference operation, relations must be compatible relations.

Notation of Set Difference Operation

P = R – S

Where,
R and S are relations.

Example –

A. P = R – S is given by

Id	Name
106	Kapil
107	Sumit

Exercises –

A. Find all the customers having an account but not the loan.

Solution – Use Query

πcust-name(Depositor) – πcust-name(Borrower)

Result –

Cust-name
Rahul
Omkar
Sumit

B. Find all the customers having a load but not the account.

Solution – Use Query

πcust-name(Borrower) – πcust-name(Depositor)

Result –

Cust-name
Ramesh
Anil

(5) Cartesian Product Operation (X)

Cartesian product of two relations is the concatenation of tuples belonging to the two relations. It is denoted by 'x' sign. If R and S are two relations, (R X S) results in a new relation P, which contains all possible combination of tuples in R and S. For Cartesian product operation, compatible relations are not required.

Notation of Cartesian Product Operation

P = R X S

Where,
P, R and S are relations.

X represents concatenations. The degree/arity of the resultant relation is given by

|P|=|R|=|S|

Example

Employee-Schema = { Emp-id, Name }

Emp-Id	**Name**
101	Sachin
103	Rahul
104	Omkar
106	Sumit
107	Ashish

Project-Schema = { Proj-name }

Proj-name
DBMS 1
DBMS 2

Find R = Employee X Project

Solution –

R-Schema = {Emp-id, Name, Proj-name}

Emp-Id	Name	Proj-name
101	Sachin	DBMS 1
101	Sachin	DBMS 2
103	Rahul	DBMS 1
103	Rahul	DBMS 2
104	Omkar	DBMS 1
104	Omkar	DBMS 2
106	Sumit	DBMS 1
106	Sumit	DBMS 2
107	Amit	DBMS 1
107	Amit	DBMS 2

If the attribute name is same in both argument relations, then that is distinguished by attaching the name of the relation from which the attribute originally came.

Exercise

Given
Customer schema = {cust-id, name}
Customer

Cust-Id	Name
101	Sachin
102	Rahul

103	Ramesh

Employees Schema = {emp-id, name}
Employee

Emp-Id	Name
201	Omkar
202	Sumit
203	Ashish

Find R = Customer X Employee

Solution
R-Schema = { cust-id, customer.name, emp-id, employee.name}

Customer X Employee

Cust-Id	Customer.name	Emp-id	Employee.name
101	Sachin	201	Omkar
101	Sachin	202	Sumit
101	Sachin	203	Ashish
102	Rahul	201	Omkar
102	Rahul	202	Sumit
102	Rahul	203	Ashish
103	Ramesh	201	Omkar
103	Ramesh	202	Sumit
103	Ramesh	203	Ashish

6.Rename Operation

This is a unary operation. Any relational algebra expression returns a new relation, but this relation is not having a name associated with it. Using Rename operation, we can rename such result relations or if we want to change the name of a given relation, it can be changed using rename operation.

It is denoted by rho (ρ)

Notation of Rename Operation

ρ(NewName, OldName)

Where,

NewName – New name of the relation.
OldName – Old name of the relation.

Example –

Question – Rename Player relation to PlayerList.

Solution – ρ(PlayerList, Player).

Exercise

Question 1. Rename Customer relation to CustomerList.

Solution
ρ(CustomerList, Customer).

****

Q) Consider the following schema:
Suppliers (sid : integer, sname : string, address : string)
Parts (pid : integer, pname : string, color : string)
Catalog (sid : integer, pid : integer, cost : real)

The key fields are underlined and domain of each field is listed after the field name
1) Find the name of suppliers who supply some red parts
We first find the pids of parts that are red in color and then we compute the natural join of this with catalog from this we project sid which gives ids of the supplier who supply some red part, then we take the natural join of this with supplier and project names which gives us the names of suppliers who supply some red part

Step 1 : R1 = πpid(σcolor = 'red'parts)
Step 2 : R2 = πsid(R1 $ Catalog)
Step 3 : R3=πname(R2 $ Sppliers)
Required answer is R3

2) Find the sids of suppliers who supply some red or green parts

Step1: R1 = πpid(σcolor = 'red' V 'green'parts)
Step 2 : R2 = πsid(R1 $ Catalog)
Same as above one but here we have to choose red or green parts and we have

to have sids of suppliers so we can stop after step 2 after choosing parts either in red color or green color

3) Find the sids of suppliers who supply some red part or are at 221 packer Ave
Sids of suppliers who supply some red part
Step 1 : R1 = πpid(σcolor = 'red'parts)
Step 2 : R2 = πsid(R1 $ Catalog)
Sids of suppiers who are at 221 packer Ave
Step 1 : R3 = πsidσaddress = '221 packer Ave(Suppliers)
Therefore sids of suppliers who supply some red part or are at 221 packer Ave
Is R2 U R3

4) Find the sids of suppliers who supply some red part and some green part
A) R1 = πsid(πpid(σcolor = 'red' parts) $ Catalog)
R2 = πsid(πpid(σcolor = 'green'parts) $ Catalog)
From question one we get the sids of suppliers who supply some red part (R1)
Similarly R2 is the sids of suppliers who supply some green part
Required list of sids who supply some red and some green part is R1
Intersection R2

5) Find the sids of suppliers who supply every part
A) R1=πsid,pid Catalog
R2=πpidParts
R1/R2 give us the required list of sids of suppliers who supply every part

6)Find the sids of suppliers who supply every red part
A)This is same as previous one but in R2 we consider only red parts
R1= πsid,pid Catalog
R2=πpidσcolr='red'parts
So required answer is R1/R2

7)Find the sids of suppliers who supply every red or green part
A) R1= πsid,pid Catalog
R2=πpidσcolr='red' v 'green' parts
R1/R2 gives the sids of suppliers who supply every part which is either red in Color or green in color

****

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] ⊆ Suppliers[sID]
Catalog[pID] ⊆ 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.
Answer:
ΠpN ame(σcolour="red"P arts)

3. Find all prices for parts that are red or green. (A part may have different prices from different manufacturers.)
Answer:
Πprice((σcolour="red"∨colour="green"P arts) ./ Catalog)

4. Find the sIDs of all suppliers who supply a part that is red or green.
Answer:
ΠsID((σcolour="red"∨colour="green"P arts) ./ 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:
ΠsN ame((ΠsID((σcolour="red"∨colour="green"P arts) ./ Catalog)) ./ Suppliers)
1

Second Schema
Employees(number, name, age, salary)
Supervises(boss, employee)
Supervises[boss] ⊆ Employees[number]
Supervises[employee] ⊆ 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:
Πname,salaryσboss =
number((Πbossσnumber=employee((ΠnumberσsalaryЈ100Employee)×Supervises))×
Employee)
2

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] ⊆ Clients[CID]
Appointments[SID] ⊆ 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:
Πname,time((ΠCID,SID,timeσdate=“F eb14”Appointments) ./ (ΠSIDσname=“Giuliano”Staf f) ./ 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

****

Relational Algebra and SQL Practice Questions
User
Id Name Age Gender OccupationId CityId
1 John 25 Male 1 3
2 Sara 20 Female 3 4
3 Victor 31 Male 2 5
4 Jane 27 Female 1 3
Occupation
OccupationId OccupationName
1 Software Engineer
2 Accountant
3 Pharmacist
4 Library Assistant
City
CityId CityName
1 Halifax
2 Calgary
3 Boston
4 New York
5 Toronto
1. Solve the following relational expressions for above relations.
a. PName(RAge>25(User))
b. RId>2∨Age!=31(User)
c. RUser.OccupationId=Occupation.OccupationId(User X Occupation)
d. User ⋈ Occupation ⋈ City
e. PName,Gender(RCityName=”Boston”(User ⋈ City))
2. Write SQL statements for relational expressions in question 1.

Answers
a. PName(RAge>25(User))
Name
Victor
Jane

SELECT Name
FROM User
WHERE Age > 25;
b. RId>2∨Age!=31(User)
Id Name Age Gender OccupationId CityId
1 John 25 Male 1 3
2 Sara 20 Female 3 4
3 Victor 31 Male 2 5
4 Jane 27 Female 1 3
SELECT *
FROM User
WHERE id>2 OR Age != 31;
c. RUser.OccupationId=Occupation.OccupationId(User X Occupation)
Id Name Age Gender OccupationId CityId OccupationId OccupationName
1 John 25 Male 1 3 1 Software Engineer
2 Sara 20 Female 3 4 3 Pharmacist
3 Victor 31 Male 2 5 2 Accountant
4 Jane 27 Female 1 3 1 Software Engineer
SELECT *
FROM User u, Occupation o
WHERE u.OccupationId = o.OccupationId;

d. User ⋈ Occupation ⋈ City
CityId OccupationId Id Name Age Gender OccupationName CityName
3 1 1 John 25 Male Software Engineer Boston
4 3 2 Sara 20 Female Pharmacist New York
5 2 3 Victor 31 Male Accountant Toronto
3 1 4 Jane 27 Female Software Engineer Boston
SELECT *
FROM User NATURAL JOIN Occupation NATURAL JOIN City;
e. PName,Gender(RCityName=”Boston”(User ⋈ City))
Name Gender
John Male
Jane Female
SELECT Name, Gender
FROM User NATURAL JOIN City
WHERE CityName = ”Boston”;

****

Consider a database with the following schema:

Person (<u>name</u>, age, gender)	name is a key
Frequents (<u>name</u>, <u>pizzeria</u>)	(name, pizzeria) is a key
Eats (<u>name</u>, <u>pizza</u>)	(name, pizza) is a key

Serves (pizzeria, pizza, price) (pizzeria, pizza) is a key

Write relational algebra expressions for the following nine queries. (Warning: some of the later queries are a bit challenging.)

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

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

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

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

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

f. 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.

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

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

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

2. Consider a schema with two relations, R(A, B) and S(B, C), where all values are integers. Make no assumptions about keys. Consider the following three relational algebra expressions:

a. $\pi_{A,C}(R \bowtie \sigma_{B=1} S)$

b. $\pi_{A}(\sigma_{B=1} R) \times \pi_{C}(\sigma_{B=1} S)$

c. $\pi_{A,C}(\pi_{A} R \times \sigma_{B=1} S)$

Two of the three expressions are equivalent (i.e., produce the same answer on all databases), while one of them can produce a different answer. Which query can produce a different answer? Give the simplest database instance you can think of where a different answer is produced.

3. Consider a relation R(A, B) that contains r tuples, and a relation S(B, C) that contains s tuples; assume $r > 0$ and
$s > 0$. Make no assumptions about keys. For each of the following relational algebra experssions, state in

terms of r and s the minimum and maximum number of tuples that could be in the result of the expression.

a. $R \cup \rho_{S(A,B)} S$

b. $\pi_{A,C}(R \bowtie S)$

c. $\pi_B R - (\pi_B R - \pi_B S)$

d. $(R \bowtie R) \bowtie R$

e. $\sigma_{A>B} R \cup \sigma_{A<B} R$

4. Two more exotic relational algebra operators we didn't cover are the *semijoin* and *antijoin*. Semijoin is the same as natural join, except only attributes of the first relation are returned in the result. For example, if we have relations Student(ID, name) and Enrolled(ID, course), and not all students are enrolled in courses, then the query
"Student ⋉Enrolled" returns the ID and name of all students who are enrolled in at least one course. In the general case, $E_1 \ltimes E_2$ returns all tuples in the result of expression E_1 such that there is at least one tuple in the result of E_2 with matching values for the shared attributes. antijoin is the converse: $E_1 \triangleright E_2$ retuns all tuples in the result of expression E_1 such that there are no tuples in the result of E_2 with matching values for the shared attributes. For example, the query "Student ▷Enrolled" returns the ID and name of all students who are not enrolled in any courses.

Like some other relational operators (e.g., intersection, natural join), semijoin and antijoin are abbreviations - they can be defined in terms of other relational operators. Define $E_1 \ltimes E_2$ in terms of other relational operators. That is, give an equation "$E_1 \ltimes E_2$ = ??", where ?? on the right-hand side is a relational algebra expression that doesn't use semijoin. Similarly, give an equation "$E_1 \triangleright E_2$ = ??", where ?? on the right-hand side is a relational algebra expression that doesn't use antijoin.

5. Consider a relation Temp(regionID, name, high, low) that records historical high and low temperatures for various regions. Regions have names, but they are identified by regionID, which is a key. Consider the following query, which uses the linear notation introduced at the end of the relational algebra videos.

$$T1(rID, h) = \pi_{regionID,\, high}\, Temp$$
$$T2(rID, l) = \pi_{regionID,\, low}\, Temp$$
$$T3(regionID) = \pi_{rID}(T1 \bowtie_{h<high} Temp)$$
$$T4(regionID) = \pi_{rID}(T1 \bowtie_{l>low} Temp)$$
$$T5(regionID) = \pi_{regionID}\, Temp - T3$$
$$T6(regionID) = \pi_{regionID}\, Temp - T4$$
$$Result(n) = \pi_{name}(Temp \bowtie (T5 \cup T6))$$

1. Sample solutions; in general there are many correct expressions for each query.

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

b. $\pi_{\text{name}}\left(\sigma_{\text{gender}='\text{female}' \wedge (\text{pizza}='\text{mushroom}' \vee \text{pizza}='\text{pepperoni}')}(\text{Person} \bowtie \text{Eats})\right)$

c. $\pi_{\text{name}}\left(\sigma_{\text{gender}='\text{female}' \wedge \text{pizza}='\text{mushroom}'}(\text{Person} \bowtie \text{Eats})\right) \cap$
$\pi_{\text{name}}\left(\sigma_{\text{gender}='\text{female}' \wedge \text{pizza}='\text{pepperoni}'}(\text{Person} \bowtie \text{Eats})\right)$

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

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

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

g. $\pi_{\text{name}}(\text{Person}) - \pi_{\text{name}}\left(\text{Frequents} - \pi_{\text{name},\text{pizzeria}}(\text{Eats} \bowtie \text{Serves})\right)$

h. $\pi_{\text{name}}(\text{Person}) - \pi_{\text{name}}\left(\pi_{\text{name},\text{pizzeria}}(\text{Eats} \bowtie \text{Serves}) - \text{Frequents}\right)$

i. $\pi_{\text{pizzeria}}(\sigma_{\text{pizza}='\text{pepperoni}'}\text{Serves}) -$
$\pi_{\text{pizzeria}}\left(\sigma_{\text{price}>\text{price2}}\left(\begin{array}{l}\pi_{\text{pizzeria},\text{price}}(\sigma_{\text{pizza}='\text{pepperoni}'}\text{Serves}) \\ \times \\ \rho_{\text{pizzeria2},\text{price2}}\left(\pi_{\text{pizzeria},\text{price}}(\sigma_{\text{pizza}='\text{pepperoni}'}\text{Serves})\right)\end{array}\right)\right)$

Query results for SQL data:

a. Straw Hat, New York Pizza, Pizza Hut

b. Amy, Fay

c. Amy

d. Little Caesars, Straw Hat, New York Pizza

e. Little Caesars, Chicago Pizza, New York Pizza

f. Amy: mushroom, Dan: mushroom, Gus: mushroom

g. Amy, Ben, Dan, Eli, Fay, Gus, Hil

h. Fay

i. Straw Hat, New York Pizza

2. Query (c) is different. Let R = {(3, 4)} and S = {(1, 2)}. Then query (a) and (b) produce an empty result while (c) produces {(3, 2)}.

3. a. Minimum = max(r, s) (if one relation is a subset of the other)
Maximum = r + s (if the relations are disjoint)

b. Minimum = 0 (if there are no shared B values)
Maximum = r x s (if all of the B values are the same)

c. Minimum = 0 (if there are no shared B values)
Maximum = min(r, s) (if one relation's B values are a subset of the other's, and all B values are distinct)

d. (equivalent to R)
Minimum = r, Maximum = r

e. Minimum = 0 (if A = B in all tuples of R)
Maximum = r (if A <> B in all tuples of R)

4.

$$E_1 \ltimes E_2 = \pi_{\text{schema}(E_1)}(E_1 \bowtie E_2)$$

$$E_1 \triangleright E_2 = E_1 - \pi_{\text{schema}(E_1)}(E_1 \bowtie E_2)$$

or

$$E_1 \triangleright E_2 = E_1 - (E_1 \ltimes E_2)$$

5. Names of regions with the highest high temperature and/or lowest low temperature

Type-1: Given a relational algebra expression, find the result.
Suppose you have a relation Order(Prod_Id, Agent_Id, Order_Month) and you have to find out what will the following algebra expression return.

$$\prod_{\text{Order1.Prod\_Id}} (\rho(\text{Order1},\text{Order})_{\substack{\text{Order1.Prod\_Id=Order2.Prod\_Id} \\ \text{and Order1.Agent\_Id}\neq\text{Order2.Agent\_Id} \\ \text{and Order1.Order\_Month=Order2.Order\_Month}}} \rho(\text{Order2},\text{Order}))$$

Process the expression starting from innermost brackets.

In this example, we have renamed order to Order1 and Order2 (Both represent the same relation order). Then we have applied the conditional join between Order1 and Order2. It will return those rows where Product_Id and Order_Month of Order1 and Order2 are the same but Agent_Id of Order1 and Order2 is different. It implies the rows where the same product is ordered by two different agents in the same month. Then we are projecting the Prod_Id.

So the final output will return the Prod_Id of products that are ordered by different agents in the same month. We can do this by taking sample data. Let Order relation consists of the following data.

Table – Order

Prod_Id	Agent_Id	Order_Month
P001	A001	JAN
P002	A002	FEB
P002	A001	FEB
P001	A002	FEB

When we apply the following expression, the rows which are highlighted in blue will be selected.

$$(\rho(\text{Order1},\text{Order})_{\substack{\text{Order1.Prod\_Id=Order2.Prod\_Id} \\ \text{and Order1.Agent\_Id}\neq\text{Order2.Agent\_Id} \\ \text{and Order1.Order\_Month=Order2.Order\_Month}}} \rho(\text{Order2},\text{Order}))$$

Order1.Prod_Id	Order1.Agent_Id	Order1.Order_Month	Order2.Prod_Id	Order2.Agent_Id	Order2.Order_Month
P001	A001	JAN	P001	A001	JAN
P002	A002	FEB	P001	A001	JAN
P002	A001	FEB	P001	A001	JAN
P001	A002	FEB	P001	A001	JAN

Order1.Prod_Id	Order1.Agent_Id	Order1.Order_Month	Order2.Prod_Id	Order2.Agent_Id	Order2.Order_Month
P001	A001	JAN	P002	A002	FEB
P002	A002	FEB	P002	A002	FEB
P002	A001	FEB	P002	A002	FEB
P001	A002	FEB	P002	A002	FEB
P001	A001	JAN	P002	A001	FEB
P002	A002	FEB	P002	A001	FEB
P002	A001	FEB	P002	A001	FEB
P001	A002	FEB	P002	A001	FEB
P001	A001	JAN	P001	A002	FEB
P002	A002	FEB	P001	A002	FEB
P002	A001	FEB	P001	A002	FEB
P001	A002	FEB	P001	A002	FEB

After projecting Order1.Prod_Id, the output will be **P002** which is Prod_Id of products that are ordered by at least two different agents in the same month.

Note – If we want to find Prod_Id which are ordered by at least three different agents in same month, it can be done as:

$$\prod_{\text{Order1.Prod\_Id}} (\sigma_{\text{Order1.Prod\_Id=Order2.Prod\_Id}} \text{ and Order1.Prod\_Id=Order3.Prod\_Id and Order1.Agent\_Id}\neq\text{Order2.Agent\_Id and Order1.Agent\_Id}\neq\text{Order3.Agent\_Id and Order2.Agent\_Id}\neq\text{Order3.Agent\_Id and Order1.Order\_Month=Order2.Order\_Month and Order1.Order\_Month=Order3.Order\_Month}(\rho(\text{Order1,Order})X\ \rho(\text{Order2,Order})X\ \rho(\text{Order3,Order})))$$

Type-2: Given two relations, what will be the maximum and minimum number of tuples after natural join?

Consider the following relation R(A,B,C) and S(B,D,E) with underlined primary key. The relation R contains 200 tuples and the relation S contains 100 tuples. What is the maximum number of tuples possible in the natural Join R and S?

To solve this type of question, first, we will see on which attribute natural join will take place. Natural join selects those rows which have equal values for common attribute. In this case, the expression would be like:

$\sigma_{R.B=S.B}$ (RX S)

In relation R, attribute B is the primary key. So Relation R will have 200 distinct values of B. On the other hand, Relation S has BD as the primary key. So attribute B can have 100 distinct values or 1 value for all rows.

Case-1: S.B has 100 distinct values and each of these values match to R.B

R		S	
A	Other Attributes	B	Other Attributes
1		1	
2		2	
.		.	
.		.	
.		.	
200		100	

In this case, every value of B in S will match to a value of B in R. So natural join will have 100 tuples.

Case-2: S.B has 1 values and this values match to R.B

R		S	
A	Other Attributes	B	Other Attributes
1		1	
2		1	
.		.	
.		.	
.		.	
200		1	

In this case, every value of B in S will match to a value of B in R. So natural join will have 100 tuples.

Case-3: S.B has 100 distinct values and none of these values matches to R.B

R		S	
A	Other Attributes	B	Other Attributes
1		201	
2		202	
.		.	
.		.	
.		.	
200		300	

In this case, no value of B in S will match to a value of B in R. So natural join will have 0 tuple.

Case-4: S.B has 1 value and it does not match with R.B

R		S	
A	Other Attributes	B	Other Attributes
1		300	
2		300	
.		.	
.		.	
.		.	
200		300	

In this case, no value of B in S will match to a value of B in R. So natural join will have 0 tuples.

So the maximum number of tuples will be 100 and min will be 0.

Note – If it is explicitly mentioned that S.B is a foreign key to R.B, then Case-3 and Case-4 discussed above are not possible because the value of S.B will be from the values of R.B. So, the minimum and maximum number of tuples in natural join will be 100.

Given below are a few examples of a database and a few queries based on that.

(1). Suppose there is a banking database which comprises following tables :

Customer(Cust_name, Cust_street, Cust_city)

Branch(Branch_name, Branch_city, Assets)

Account (Branch_name, Account_number, Balance)

Loan(Branch_name, Loan_number, Amount)

Depositor(Cust_name, Account_number)

Borrower(Cust_name, Loan_number)

Query : Find the names of all the customers who have taken a loan from the bank and also have an account at the bank.

Solution:

Step 1 : Identify the relations that would be required to frame the resultant query.

First half of the query(i.e. names of customers who have taken loan) indicates "borrowers" information.

So Relation 1 ——> Borrower.

Second half of the query needs Customer Name and Account number which can be obtained from Depositor relation.

Hence, Relation 2——> Depositor.

Step 2 : Identify the columns which you require from the relations obtained in Step 1.

Column 1 : Cust_name from Borrower

$$\Pi_{customer\_name}\ (borrower)$$

Column 2 : Cust_name from Depositor

$$\Pi_{customer\_name}\ (depositor)$$

Step 3 : Identify the operator to be used. We need to find out the **names of customers** who are present in **both Borrower** table and **Depositor** table.

Hence, operator to be used—-> Intersection.

Final Query will be

$$\Pi_{customer\_name}(borrower) \cap \Pi_{customer\_name}(depositor)$$