CSC343 Assignment 1

Shanthosh Sivayogalingam, Bikramjit Narwal

February 9th, 2023

Part 1: Violating Our Constraints

For C8:

Trip

rID	tID	date	volume	eid1	eid2	fID
5	10	2023-03-04	100	1	2	7

Truck

tID	truckType	capacity
10	A	100

Driver

eID	truckType
1	C
2	D

Constraint C8 in English states that every trip mush have 1 of its two drivers as a driver who can drive that specific truck type. The tables above use the 3 relations used in the constraint which are Trip, Truck and Driver. The example shows an instance that violates this constraint because for the trip shown it has drivers with eids of 1 and 2 who can drive truckTypes C and D respectfully (shown in table for Driver). However, for the trip tuple shown the tID shows that for this trip the truckType is A so both driver's are not suitable to drive it.

For C9:

Maintenance

tID	eID	date
8	7	2022-01-01

Truck

tID	truckType	Capacity
8	Compost	80000

Technician

eID	truckType
7	Recycling

Constraint 9 suggests that every maintenance event must be carried out by a technician who is qualified to maintain the specific type of truck involved. Based on the information in the Technician table, a technician with eID 7 is only authorized to perform maintenance on Recycling trucks. However, the Maintenance table indicates that this technician worked on truck tID 8 on 2022-01-01. Upon investigation, it is determined that this truck is a Compost truck, for which the technician is not authorized to perform maintenance.

For C10:

Trip

rID	tID	date	volume	eID1	eID2	fID
1	1	2023-01-01	56777	34	35	45
4	1	2023-01-01	86333	34	35	32

This constraint specifies that the set of all pairs of trips, represented by the "TripPairs" relational algebra expression, only includes those where the two trips either have the same ID or share both common employee IDs. This set is then further filtered to remove any pairs that have identical trip ID, employee ID 1, and employee ID 2, meaning that the two trips in the pair are the exact same trip. The result will not contain any duplicates. Violating this constraint would mean including trips with the same tID, date, eID1, and eID2. In other words, two drivers in the same truck can’t be doing more than one trip a day.

Part 2: Queries

1. Find the trip(s) where the two drivers were hired at least 1000 days apart. Report the dates and both eIDs from those trips.

-- Cartesian Join employee twice to get two eID's and hireDates

   a. employeeInfo(eID1, eID2, hireDate1, hireDate2) := $\Pi_{T1.eID,\ T2.eID,\ T1.hireDate,\ T2.hireDate}(\rho_{T1}\text{Employee} \times \rho_{T2}\text{Employee})$

-- Find the driver hire dates for each trip and get the hire dates for each employee and their IDs

   b. tripAndEmployeeInfo(eID1, eID2, hireDate1, hireDate2, rID, tID, date) := $(\sigma_{((Trip.eID1 = R1.eID1)\ \wedge\ (Trip.eID2 = R1.eID2))}(\text{Trip} \times \text{employeeInfo}))$

-- Now select the trips where the difference between the two hire dates is at least 1000 days.

   c. eligibleTrips = $\sigma_{(((hireDate1 - hireDate2) >= 1000)\ \vee\ ((hireDate2 - hireDate1) >= 1000))}$ tripAndEmployeeInfo

2. Find the longest serving driver(s) from the ones that have a trip on every route. Report the eID of those drivers.

-- First, we will get the reality which is that every driver is not on every single route. This gives us the routes with actual eIDs

   a. Reality(rID, eID1, eID2) := $\Pi_{Route.rID,\ eID1,\ eID2}(\sigma_{(Route.rID = Trip.rID)}(\text{Route} \times \text{Trip}))$

-- In our dream world, we wish that every driver is on all routes

   b. dreamWorld(rID, eID) := $(\Pi_{rID}\ \text{Route} \times \Pi_{eID}\text{Driver})$

-- To get one column of all drivers eIDS, we take the union all eID1's and all eID2's. This eliminates any duplicates that are between the two tables.

   c. actualReality(rID, eID) := $(\Pi_{rID,\ eID1}\ \text{Reality}) \cup (\Pi_{rID,\ eID2}\ \text{Reality})$

-- The set difference between out dreamworld and reality will give us drivers that have been on all routes.

d. driverOnAllRoutes = Π_{eID} (dreamworld – Reality)

3. Find all drivers who have never had a trip on a route with stops requiring assistance. Report their eIDs.

-- Get the eIDs of drivers who have taken trips

a. driversWithTrips(eID1, eID2, rID) := ($\Pi_{eID1, eID2, rID}$ Trip)

-- Get the eIDs of drivers who have taken trips on routes with stops requiring assistance

b. driversRequiringAssistanceOnTrips(eID1, eID2, rID) := $\sigma_{(DriversWithTrips.rID=Stop.rID)}$(driversWithTrips × ($\sigma_{(assistance = 'True')}$ Stop))

-- Use set difference between driversWithTrips and driversRequiringAssistanceOnTrips to get drivers who never had a trip on a route with stops requiring assistance

c. noAssistance(eID1, eID2, rID) := driversWithTrips - driversRequiringAssistanceOnTrips

-- Only attain their eID's

d. $\Pi_{eID1, eID2}$ noAssistance

4. A Supported Route is a route where all stops on the route require assistance. Find all drivers who have been on at least two trips on all Supported Routes. Report the eIDs of the drivers and the rIDs of the routes.

-- First find all non-supported routes

a. allNonSupportedRoutes(rID) := Π_{rID} ($\sigma_{(assistance = 'False')}$ Stop)

-- Now to determine all Supported routes, subtract the non-supported routes from all possible routes.

b. allSupportedRoutes(rID) := Π_{rID} (Route) – allNonSupportedRoutes

-- Now find the trips related to all those supported routes

c. tripsOnSupportedRoutes(rID, date, eID1, eID2) :=
($\Pi_{rID,date,eID1,eID2}$ (Trip)) ⋈ allSupportedRoutes

-- Now create the relations for every possibility of which driver took at least two trips on all supported routes

d. firstAndFirstDriver(rID, eID) := $\Pi_{T1.rID,\ T1.eID1}$ ($\sigma_{T1.rID = T2.rID \wedge T1.date < T2.date \wedge T1.eID1 = T2.eID1}$ (ρ_{T1} tripsOnSupportedRoutes × ρ_{T2} tripsOnSupportedRoutes))
e. firstAndSecondDriver(rID, eID) := $\Pi_{T1.rID,\ T1.eID1}$ ($\sigma_{T1.rID = T2.rID \wedge T1.date < T2.date \wedge T1.eID1 = T2.eID2}$ (ρ_{T1} tripsOnSupportedRoutes × ρ_{T2} tripsOnSupportedRoutes)
f. secondAndFirstDriver(rID, eID) := $\Pi_{T1.rID,\ T1.eID1}$ ($\sigma_{T1.rID = T2.rID \wedge T1.date < T2.date \wedge T1.eID2 = T2.eID1}$ (ρ_{T1} tripsOnSupportedRoutes × ρ_{T2} tripsOnSupportedRoutes)
g. secondAndSecondDriver(rID, eID) := $\Pi_{T1.rID,\ T1.eID1}$ ($\sigma_{T1.rID = T2.rID \wedge T1.date < T2.date \wedge T1.eID2 = T2.eID2}$ (ρ_{T1} tripsOnSupportedRoutes × ρ_{T2} tripsOnSupportedRoutes)

-- Now to get the unique set of all drivers who have driven at least twice on all supported routes, take the union of all the driver relations to make sure you get all drivers

h. atLeastTwoTrips(rID, eID) := (firstAndFirstDriver) U (firstAndSecondDriver) U (secondAndFirstDriver) U (secondAndSecondDriver)

5. Find all drivers who have been on every trip collecting recycling at 40 St George Street. Report their eIDs and names.

-- Select trips to 40 St. George St. by filtering on “40 St. George St” and Recycling

a. trip40SGS(eID1, eID2) := $\sigma_{(wasteType = \text{‘Recycling’} \wedge address = \text{‘40 St. George St’})}$ (Trip ⋈ Stop ⋈ Route)

-- Project the eID1 and eID2 from the trips mentioned above

b. driver_eIDS(eID1, eID2) := $\Pi_{eID1,\ eID2}$ Trip40SGS

-- Join Drivers_eIDs with Employee on eID to get the names. Then project eID's and name attributes.

c. allRelevantDrivers(eID, name) = $\Pi_{(eID,\ name)}$ ($\sigma_{(eID1 = eID) \vee (eID2 = eID)}$ Driver_eIDS × Employee)

6. Find the route(s) that have the maximum number of truck trips on any single day. Report the rIDs of these routes.

a. This query is not possible using only relational algebra because to find the routes that have the maximum number of truck trips on any single day would require some sort of count function to determine the total number of trips.

7. Find all trucks that collected exactly one waste type in the past 7 days (i.e. (today – date) <= 7). Report the tIDs of those trucks.

-- Filter trips that only took place in the past 7 days.

a. tripsPast7Days(rID, tID, date, eID1, eID2) := $\sigma_{(today - date) <= 7)}$ Trip

-- Filter further to get wasteType and tIDs (join trip and route)

b. wasteTypes_tIDs(eID1, eID2, wasteType, tID) := $\sigma_{(Trip.rID = Route.rID)}$ (Trip × Route)

-- Self-join part b and check for at least 2 wasteTypes

c. atLeastTwoWasteTypes(eID1, eID2, wasteType, tID) := $\sigma_{((W1.wasteType \neq W2.wasteType) \wedge (W1.tID = W2.tID))}$($\rho_{W1}$ wasteTypes_tIDs × ρ_{W2} wasteTypes_tIDs)

-- Do a set difference between wasteTypes_tIDs and atLeastTwoWasteTypes to get all trucks that collected exactly one waste type

d. eligibleTrucks(tID) := Π_{tID} (wasteTypes_tIDs – atLeastTwoWasteTypes)

8. Find the average collection volume for each truck over all of its trips. Report the tID of the truck(s) whose average is closest to its capacity.

    a. The following query cannot be solved using only relational algebra because in order to find the average collection volume for each truck over all it's trip we would need to use aggregate functions (avg function).

9. Find the facility that collected the most waste total to date. Report the fID of that facility.

    a. The following query cannot be solved using only relational functions because it requires finding the sum of the waste collected by each facility, which can only be done through an aggregate function.

10. Find the opening date for each facility (we'll assume the first trip to that facility is when it opened). Report the address and opening date for each facility.

-- First to determine the first trip to a facility, compare the trip dates to all facilities and select the earliest date

    a. firstTripToFacility(fID, date) := $\Pi_{T1.fID,T1.date}$ ($\sigma_{(T1.fID = T2.fID \wedge T1.date < T2.date)}$($\rho_{T1}$ Trip × ρ_{T2} Trip))

-- Now to determine the facility address we can join with the facility relation to get that information.

    b. facilityInfo(address, date) := $\Pi_{address,date}$ (firstTripToFacility ⋈ Facility)

11. Find the trucks that are currently in need of maintenance (i.e. it has been > 100 days since their last maintenance). Report the eIDs of the technicians who can maintain them.

-- Find the last maintenance of every truck. Get all maintenance dates for every truck that are the largest (last maintenance)

a. lastMaintenanceDates (tID, eID, date) := $\sigma_{((T1.tID = T2.tID) \wedge (T1.date>T2.date))}$ (ρ_{T1}Maintenance × ρ_{T1}Maintenance)

-- Select the tuple from lastMaintenanceDates where the difference between the current date and the last maintenance date is greater than 100. Also project their tID

b. needMaintenance (tID) :=$\Pi_{tID}(\sigma_{((today - date) > 100)}$ lastMaintenanceDates)

-- Natural join needMaintenance and truck to get all the information about the trucks that need maintenance and then only project the eID of each tuple.

c. eligibleTrucks(tID, truckType) := $\Pi_{tID, truckType}$ (needMaintenance ⋈ Truck)
d. trucksCurrentlyNeedingMaintanence(tID) := Π_{tID} (eligibleTrucks ⋈ Technician)

12. Find all trucks that have been maintained by an employee who has driven that same truck on a trip. Report the tIDs and eID.

-- First create a relation for all trips of with the maintenance information of the trucks

a. maintainedTrucks(tID, eID, mdate) := ($\sigma_{(T1.tID = Trip.tID)}$ ($\rho_{T1(tID,eID,mdate)}$ (Maintenance) × Trip))

-- Create a relation for the maintenance info of the trucks for each trip respective of each driver

b. maintenanceFirstDriver(tID, eID) := $\Pi_{tID,eID}$ ($\sigma_{eID = eID1}$ (maintainedTrucks))
c. maintenanceSecondDriver(tID, eID) := $\Pi_{tID,eID}$ ($\sigma_{eID = eID2}$ (maintainedTrucks))

-- Now to find the relation with all the drivers and the tIDs of the trucks that they drove and maintained take the union of the previous two relations.

d. allTrucksMaintainedByDriver(tID, eID) := (maintanenceFirstDriver) ∪ (maintenanceSecondDriver)

13. Find the truck(s) that were on the most routes (but not necessarily the most trips) in the last 7 days. Report tIDs, rIDs, and waste types for those routes.

a. It is not possible to find the truck(s) that were on the most routes in the last 7 days without using aggregate functions in relational algebra. An aggregate function is necessary to count the number of routes for each truck and determine which truck had the highest count.

14. Find all trucks that have collected exactly 3 different waste types. Report the tIDs and truckTypes of these trucks.

-- To determine the trucks that have collected exactly 3 different waste types we used the self-join technique to determine at least 3 different and then at least 4 different and find the set difference. First, we determine the relation for every trip along with the wasteType info for that trip

a. tripWasteType(tID, wasteType) := $(\sigma_{\text{Trip.rID = Route.rID}}(\text{Trip} \times \text{Route}))$

-- Find the trucks that have collected at least 4 different waste types and then a relation for at least 3 different waste types.

b. atLeastFourTypes(tID) :=

$$\Pi_{\text{T1.tID}}(\sigma_{(\text{T1.tID=T2.tID=T3.tID=T4.tID}) \wedge (\text{T1.wasteType} \neq \text{T2.wasteType} \neq \text{T3.wasteType} \neq \text{T4.wasteType})}(\rho_{\text{T1}}\,\text{tripWasteType} \times \rho_{\text{T2}}\,\text{tripWasteType} \times \rho_{\text{T3}}\,\text{tripWasteType} \times \rho_{\text{T4}}\,\text{tripWasteType}))$$

c. atLeastThreeTypes(tID) :=

$$\Pi_{T1.tID}\,(\sigma_{(T1.tID=T2.tID=T3.tID)\,\wedge\,(T1.wasteType\,\neq\,T2.wasteType\,\neq\,T3.wasteType)}\,(\rho_{T1}\,tripWasteType\,\times\,\rho_{T2}\,tripWasteType\,\times\,\rho_{T3}\,tripWasteType))$$

-- To determine the relation with trucks that have collected exactly 3 different waste types.

d. exactlyThreeTypes(tID) := atLeastThreeTypes – atLeastFourTypes

-- To finally find the relation of those trucks and its truckType we natural join with the Truck relation

e. truckInfo(tID, truckType) := $\Pi_{tID,truckType}$ (exactlyThreeTypes ⋈ Truck)

15. Two routes are considered equivalent if they have the exact same stop addresses (ignore assistance for this) and collect the same type of waste. Report the rIDs, tIDs, and dates for all pairs of trips on equivalent routes. Do not report pseudo-duplicates

-- First, we need to determine a relation where we have the rID and respective stops for that route along with its waste type.

a. stopRouteInfo(rID, address, wasteType) := $\Pi_{Stop.rID,\ address,\ wasteType}\,(\sigma_{(Stop.rID\,=\,Route.rID)}$ (Stop × Route))

-- Any non-equivalent route is one where the route ids are not the same and either addresses for a stop are not the same or the waste type is not the same. So, we self-joined the previous relation (from a) to determine all the routes no are not equivalent.

b. noEquivalentRoutes(rID, address, wasteType) := $(\sigma_{(T1.rID\,\neq\,T2.rID\,\wedge\,T1.address\,\neq\,T2.address)\,\vee\,(T.rID\,\neq\,T2.rID\,\wedge\,T1.wasteType\,\neq\,T2.wasteType)}\,(\rho_{T1}\,stopRouteInfo\times\,\rho_{T2}\,stopRouteInfo))$

-- To find the relation of only routes that are equivalent we take the set difference to remove those routes that are not equivalent (determined in b above)

c. equivalentRoutes(rID, address, wasteType) := stopRouteInfo – notEquivalent

-- Now to determine those equivalent routes' trip information we crossed it with the Trip relation and selected only those trips of the equivalent routes and their info (route id, truck type and date)

d. equalRouteTrips(rID, tID, date) := $\Pi_{equivalent.rID,tID,date}$ ($\sigma_{(equivalent.rID = Trip.rID)}$(equivalentRoutes × Trip))

-- We want to eliminate any duplicates within this relation where we are dealing with the exact same trip so we find a relation for those duplicates.

e. Duplicates (rID, tID, date) : = $\Pi_{T1.rID,T1.tID,T1.date}$ ($\sigma_{(T1.rID = T2.rID \wedge T1.tID = T2.tID \wedge T1.date = T2.date)}$ (ρ_{T1} equalRouteTrips× ρ_{T2} equalRouteTrips))

-- Finally, we take the set difference and remove the duplicate entries and the final relation will have the pairs of equivalent trips and their route ID, truck ID and trip date.

f. finalTrips(rID,tID,date) := equalRouteTrips - Duplicates

Part 3: Your constraints

Please note, that for this section the query explanations are beneath each of them.

1. No route has more than 2 stops requiring assistance.

$\sigma_{((s1.address \neq s2.address) \wedge (s1.address \neq s3.address) \wedge (s1.assistance = s2.assistance = s3.assistance = \text{'True'}) \wedge (s1.rID = s2.rID = s3.rID))}(\rho_{S1}\ \text{Stop} \times \rho_{S2}\ \text{Stop} \times \rho_{S3}\ \text{Stop}) = \emptyset$

The above query checks for 3 stops that require assistance. Equating this constraint to the empty sets implies that a route has 3 stops requiring assistance which breaks the constraint that **no more** than two stops require assistance. Since we did the query for at least 3 stops, it holds for any number of stops greater than 3.

Affected Queries:

- Query 4: This would guarantee an empty result (As long as the route has more than 2 stops)

2. An employee is either a driver or a technician, and not both.

$\Pi_{eID}\ (\text{Driver}) \cap \Pi_{eID}\ (\text{Technician}) = \emptyset$

For this constraint, if there are any employee IDs of drivers that are common with employee IDS of technicians the constraint is violated. So, in the query we check the intersect of all driver IDs with technician IDs and set it equal to the null set.

Affected Queries:

- Query 12: Would result in an empty set since the person who maintained the truck cannot drive it (Technician Vs. Driver)

3. Every stop address is on at least one route for every waste type.

idealWorld(wasteType, address) := $\Pi_{(wasteType,\ address)}(\text{WasteType} \times \text{Stop})$

realWorld(wasteType, address) := $\Pi_{(wasteType,\ address)}(\sigma_{(Stop.rID = Route.rID)}(\text{Stop} \times \text{Route})$

idealWorld – realWorld = Ø

In this scenario we needed to take two cases into consideration. Case one included the ideal world where we assume every stop address has every waste type. However, in the real world (case two), we are not able to make this assumption therefore we are taking all the route addresses and the actual waste types associated with them. If the set difference between the two is not the empty set, that means that at least one stop address does not have every waste type. So, the set difference must be equal to the empty set.

Affected Queries:

- No queries affected, none would return an empty set

4. All employees who are drivers can drive at least two truck types.

atLeastTwo(eID) := $\Pi_{T1.eID}$ ($\sigma_{T1.eID=T2.eID \wedge T1.truckType \neq T2.truckType}$ (ρ_{T2} Driver × ρ_{T2} Driver))

Π_{eID} Driver – atLeastTwo = Ø

For this constraint, there must be no drivers only driving 1 truckType. So, we found the relation where we have all drivers who can drive at least two truck types and taking the set difference with all possible drivers leaves a relation with those who can drive only 1 truck type (which needs to be null).

Affected Queries:

- No queries affected, none would return an empty set

5. No route has a total volume of more than 1000 on any given date across all trips on that route.

This constraint cannot be expressed in relational algebra. The reason being that relational algebra does not support aggregation operations like SUM and COUNT, which are necessary to add up

the total volume of waste collected on a specific route and date. This information would need to be obtained from a separate trip relation, where the tID, rID, and date are used to match and accumulate the corresponding volume.

Affected Queries:

- No queries affected: Constraint not feasible

6. No facility has the same address as a stop.

allTripStops(rID, fID, address) :=($\Pi_{\text{Trip.rID, fID, address}}$ ($\sigma_{\text{(Trip.rID = Stop.rID)}}$ (Trip × Stop))

allTripStopsFacilityInfo(rID, fID, address, faddress):= allTripStops ⋈ ($\rho_{\text{(fID,faddress,wasteType)}}$ Facility)

$\sigma_{\text{address = faddress}}$ (allTripStopsFacilityInfo) = 0

For this constraint, we found all the stops related to every route and its respective facility with allTripStops. Then natural joined this relation with the Facility relation in order to add the information of the facility address to this relation as well. Finally, for this constraint to be true, no stop address should be equal to a facility address thus the relation of selecting all tuples with the same stop address and facility address should be null.

Affected Queries:

- No queries affected, none would return an empty set