University of Toronto
CSC343, Fall 2018

Assignment 1

Juliano Garcia de Oliveira

UTORid: olive207

Part 1: Queries

Question 1

Even thought it doesn't make sense given the schema, this query can still be expressed in relational algebra.

$EmpExpertise := (\Pi_{eID} Employee) \bowtie PilotExpertise$ *// To get the airline of the employee*

// Do a self join using eID, then get every entry where the aircraftType is different

$Experts(eID) := \Pi_{E1.eID}[(\rho_{E1} EmpExpertise) \bowtie_{E1.eID=E2.eID \wedge E1.aircraftType \neq E2.aircraftType} (\rho_{E2} EmpExpertise)]$

$Answer := Experts \bowtie Employee$

Question 2

Main idea is to construct the airports that one can get by using just One flight, then Two flights, and finally, Three flights. With the simplifying assumption, if the arrival time of the first flight + 120 is lesser than the departure time of the second flight, we know for sure that both flights can be done in the same day giving 120 minutes between them.

$FlInfo(cID, fnum, o, d, dt, at) := \Pi_{cID, flightNumber, origin, destination, departureTime, arrivalTime} Flight$

$One := \sigma_{o='YYZ'} FlInfo$ *// Can do all these flights*

// For two flights, get all the rows where the second flight have the same origin as the destination of the first flight which is not 'YYZ' because we don't want to go back to YYZ

$Two := One \bowtie_{((One.cID \neq FlInfo.cID) \vee (One.fnum \neq FlInfo.fnum)) \wedge (One.d = FlInfo.o) \wedge (FlInfo.d \neq 'YYZ')} FlightInfo$

// Filter the ones that have 120 minutes in between, and rename attributes

$Two(cID, fnum, o, d, dt, at) := \Pi_{FlInfo.cID, FlInfo.fnum, FlInfo.o, FlInfo.d, FlInfo.dt, FlInfo.at} \sigma_{One.at+120 \leq FlInfo.dt} Two$

// Basically the same thing for Three, but using flights from Two

$Three := Two \bowtie_{((Two.cID \neq FlInfo.cID) \vee (Two.fnum \neq FlInfo.fnum)) \wedge (Two.d = FlInfo.o)} FlightInfo$

$Three(d) := \Pi_{FlInfo.d} \sigma_{Three.at+120 \leq FlInfo.dt} Three$

// Get all the airports code and name from the ones you can go, then use union to complete the answer

$OneFlight(apID, name) := \Pi_{apID, name}(Airport \bowtie_{apID=d} (\Pi_d One))$

$TwoFlight(apID, name) := \Pi_{apID, name}(Airport \bowtie_{apID=d} (\Pi_d Two))$

$ThreeFlight(apID, name) := \Pi_{apID, name}(Airport \bowtie_{apID=d} (\Pi_d Three))$

$Answer(apID, name) := OneFlight \cup TwoFlight \cup ThreeFlight$

Question 3

Main idea is to construct the instances where at least one speak french, then do set difference from all instances.

$FrenchSpeakers := (\Pi_{eID}\sigma_{primary\_language='French' \vee secondary\_language='French'}Employee)$

$AtLeastOneFrench(fID) := \Pi_{fID}(CrewAssignment \bowtie FrenchSpeakers)$

$AllFlightsInstances(fID) := \Pi_{fID}FlightInstance$

$NoFrenchFlights(fID) := AllFlightsInstances - AtLeastOneFrench$

$Answer(cID, flightNumber, date) := \Pi_{cID,flightNumber,date}(FlightInstance \bowtie NoFrenchFlights)$

Question 4

Idea is to get all possible 4 tuples of flights, such that every flight is different from each other. Then, get the airport in which all agree, i.e. if all flights have the same 'origin', it means this airport has MORE than 3 departing flights (that is, at LEAST 4 flights departed a day, since a flight starts and ends at the same day). The whole ***Temp*** *relations are just to make sure every flight is paired only with different flights, and is broken down for readability.*

$FlightSimplified(cid, flightNumber, origin) := \Pi_{cid,flightNumber,origin}Flight$

$QuadriCross := (\rho_{F1}FlightSimplified)\times(\rho_{F2}FlightSimplified)\times(\rho_{F3}FlightSimplified)\times(\rho_{F4}FlightSimplified)$

$Temp1a := \sigma_{\neg(F1.cid=F2.cid \wedge F1.flightNumber=F2.flightNumber)}\ QuadriCross$

$Temp1b := \sigma_{\neg(F2.cid=F3.cid \wedge F2.flightNumber=F3.flightNumber)}\ Temp1a$

$Temp1c := \sigma_{\neg(F1.cid=F3.cid \wedge F1.flightNumber=F3.flightNumber)}\ Temp1b$

$Temp2 := \sigma_{\neg(F1.cid=F4.cid \wedge F1.flightNumber=F4.flightNumber) \wedge \neg(F2.cid=F4.cid \wedge F2.flightNumber=F4.flightNumber)}\ Temp1b$

$AllDiffFlights := \sigma_{\neg(F3.cid=F4.cid \wedge F3.flightNumber=F4.flightNumber)}\ Temp2$

// It was asked in Piaza if we should return just apID or everything for an airport, instructor didn't answered by the time I made this, so I'm returning just the apID. If necessary to return all other attributes I would just use a join.

$Answer(airport) := \Pi_{F1.origin}\sigma_{F1.origin=F2.origin \wedge F2.origin=F3.origin \wedge F3.origin=F4.origin}\ AllDiffFlights$

Question 5

cannot be expressed

Question 6

Bind all eID with flightInstances, then group them by two. Get all flightsInstances groups which happen on the same date, so we can compare both. But, when we do this, there will be (pseudo)duplicates in our relation. We want first to get 'GOOD' cases, which is cases of two flights that do not overlap. I'll get the good cases by checking if, for two FlightInstances F1 and F2, $F1.arrivalTime < F2.departureTime$, then they DON'T OVERLAP, since all flights starts and ends in the same day only. This will eliminate the (pseudo)duplicates + overlapping flights, and we then filter only the IDs of the flight + eID. This will gives us a tuple with (fA, fB, eID). Then, to get the flights that DO overlap, first we need to remove the (pseudo) duplicates from the table, filter them the same way, and subtract all good flights from all flights, which will get us the flights that DO overlap.

$CAFI := CrewAssignment \bowtie FlightInstance$

$SameDay := (\rho_{C1} CAFI) \bowtie_{C1.date=C2.date \wedge C1.fID \neq C2.fID \wedge C1.eID=C2.eID} (\rho_{C2} CAFI)$

// If $C1.arrivalTime < C2.departureTime$, this implies that they do not overlap

$DontOverlap(fA, fB, eID) := \Pi_{C1.fID,C2.fID,C1.eID}[\sigma_{C1.arrivalTime<C2.departureTime} SameDay]$

// Here we generate all pseudo duplicates to filter correctly, that is, if (X, Y) is in the relation, we construct a new relation with [(X, Y), (Y, X)]

$Cprod(iB, iA) := (\Pi_{fB} DontOverlap) \times (\Pi_{fA} DontOverlap)$

// Make sure we are with only valid pairs of fA, fB

$Temp1 := \sigma_{(fA=iB \wedge fB=iA) \cup (fA=iA \wedge fB=iB)}[DontOverlap \times Cprod]$

$AllPseudodupDontOverlap(fA, fB, eID) := \Pi_{iB,iA,eID}$

// Here we remove (pseudo)duplicates by applying the selection $C1.fID < C2.fID$

$AllCases(fA, fB, eID) = \Pi_{C1.fID,C2.fID,C1.eID} \sigma_{C1.fID<C2.fID} SameDay$

$GoodFilteredCases := \sigma_{fA<fB} AllPseudodupDontOverlap$

// First case of scheduling conflict: Employee works in at least one pair of flights with overlap

$Violation1(eID) := \Pi_{eID}[AllCases - GoodFilteredCases]$

// If same date and don't overlap, check if $fA.destination \neq fB.origin$

$F1 := \Pi_{cID,flightNumber,destination,arriveTime} Flight$

$Temp1 := (\sigma_{C1.arrivalTime<C2.departureTime} SameDay) \bowtie_{C1.cID=F1.cID \wedge C1.flightNumber=F1.flightNumber} F1$

$Temp2 := Temp1 \bowtie_{C2.cID=F2.cID \wedge C2.flightNumber=F2.flightNumber} (\rho_{F2}(\Pi_{cID,flightNumber,origin,departureTime} Flight))$

// Second case of scheduling conflict: Same day, don't overlap, but destination of first flight is different than origin of second flight

$Violation2a := \Pi_{C1.eID}[\sigma_{F1.destination \neq F2.origin} Temp2]$

// Third case of scheduling conflict: Same day, don't overlap, $F1.destination = F2.origin$ but the time between scheduled arrival and departure is less than one hour

$Violation2b := \Pi_{C1.eID} \sigma_{F1.destination=F2.origin \wedge F2.departureTime-F1.arriveTime<60} Temp2$

// Unite all eID whom have schedule conflicts

$Answer(eID) := Violation1 \cup Violation2a \cup Violation2b$

Question 7

Giving our schema, only makes sense if we have only one type of aircraftType registered... But this doesn't stop us from writing the query in relational algebra.

// Basically re-create division operator

$AllAircraftTypes(aircraftType) := \Pi_{aircraftType} Aircraft$

$ExpertEmployee(eID) := \Pi_{eID} PilotExpertise - \Pi_{eID}[((\Pi_{eID} PilotExpertise) \times AllAircraftTypes) - PilotExpertise]$

$Answer(eID, rank) := \Pi_{eID,rank}(ExpertEmployee \bowtie Employee)$

Question 8

cannot be expressed

Question 9

Use set difference to get the newer one(s)

$NotBiggest(acID) := \Pi_{AC1.acID} \sigma_{AC1.firstFlightDate < AC2.firstFlightDate}[(\rho_{AC1} Aicraft) \times (\rho_{AC2} Aircraft)]$

$NewestAC := (\Pi_{acID} Aircraft) - NotBiggest$

$NewestACdates(acID, fdate) := \Pi_{acID, firstFlightDate}(NewestAC \bowtie Aircraft)$

$Answer(cID) := \Pi_{cID}(FlightInstance \bowtie_{date=fdate} NewestACdates)$

Question 10

Separate in two different violations

// Get all the assigned captains

$FlightCap(fID, eID) := \sigma_{role='Captain'} CrewAssignment$

$CaptainFlights(fID, eID, aircraftType, acID) := ((FlightCap \bowtie PilotExpertise) \bowtie (\Pi_{fID,acID} FlightInstances))$

$HasExpertise(fID, eID) := \Pi_{fID,eID} \sigma_{aircraftType=type}(CaptainFlights \bowtie (\Pi_{acID,type} Aircraft))$

// We need to keep (fID, eID) tuples for the set difference to make sense, because in the end we want pairs where eID doesn't have expertise in aircraft from flight fID.

$NoExpertise(fID, eID) := (\Pi_{fID,eID} CaptainFlights) - HasExpertise$

$Violations1(cID, fID) := \Pi_{cID,fID}(NoExpertise \bowtie FlightInstance)$

// The second possible violation is distance and range of aircraft

$Temp1 := (\rho_{FI} FlightInstances) \bowtie_{FI.cID=F.cID \wedge FI.flightNumber=F.flightNumber} (\rho_F Flight)$

$FlightDetailed(fID, cID, fnum, origin, dest, acID) := \Pi_{FI.fID,FI.cID,FI.flightNumber,F.origin,F.destination,FI.acID} Temp1$

$FlightACrange(fID, cID, fnum, origin, dest, acID, range) := FlightDetailed \bowtie (\Pi_{acID,range} Aircraft)$

$Flight2Distance1 := (\rho_{FC} FlightACrange) \bowtie_{FC.origin=AD.apID1 \wedge FC.dest=AD.apID2} (\rho_{AD} AirportDistance)$

$Violations2a(fID, cID) := \Pi_{fID,cID} \sigma_{distance>range} Flight2Distance$

// Maybe the entry (A, B) of origin-destination is not in AirportDistance, only (B, A), so we check again with the reverse order.

$Flight2Distance2 := (\rho_{FC} FlightACrange) \bowtie_{FC.origin=AD.apID2 \wedge FC.dest=AD.apID1} (\rho_{AD} AirportDistance)$

$Violations2b(fID, cID) := \Pi_{fID,cID} \sigma_{distance>range} Flight2Distance2$

// Answer is union of every possible case

$Answer(fID, cID) := Violations1 \cup Violations2a \cup Violations2b$

Question 11

Seems there's a typo in the question ("who with")... I considered this question is asking for all employees who speak french as a second language or (logical or) have expertise in an aircraft with maximum speed over 700.)

$FSpeakers(eID) := \Pi_{eID}\sigma_{secondary\_language='French'}Employee$

// Have expertise in an arcraft with maxspeed $> 700km/h$

$FAircraft := \Pi_{acID,type,maxSpeed}\sigma_{maxSpeed>700}Aircraft$

$HaveExp(eID) := \Pi_{eID}[PilotExpertise \bowtie_{PilotExpertise.aircraftType=FAircraft.type} FAircraft]$

$Answer(eID) := FSpeakers \cup HaveExp$

Question 12

What is 'registered fleet', and how can we determine it? There's nothing in the schema that points that, it was asked (multiple times) on piazza but no instructor answer, so I had to assume things here.

// I'm assuming every aircraft in the registered fleet has flown once, or better, have an entry in the flightInstance relation.

$Dreamliner(acID) := \Pi_{acID}\sigma_{manufacturer='Boeing' \wedge type='787-9'}Aircraft$

$HaveDreamliner(cID) := \Pi_{cID}[FlightInstance \bowtie Dreamliner]$

$Answer(cID) := (\Pi_{cID}Airline) - HaveDreamliner$

Part 2: Integrity constraints

Question 1

$\sigma_{origin=destination}Flight = \emptyset$

Question 2

$\sigma_{C1.fID=C2.fID \wedge C1.eID=C1.eID \wedge C1.role \neq C2.role}[(\rho_{C1}CrewAssignment) \times (\rho_{C2}CrewAssignment)] = \emptyset$