University of Toronto
CSC343, Winter 2024
Hshmat Sahak – 1005903710
Nabil Mohamed – 1005948575

Assignment 1

Part 1: Create an instance (10%)

We provide an instance of the database below, which matches the given description. We provide only tuples that are needed to represent the described trip. We assume that optional trips need to be included in the itinerary, since they are still part of the planned route for the day.

City		
cityID	name	country
1	Toronto	Canada
2	Montreal	Canada

Trip		
tripID	startCity	length
1	1	4

TransportActivity		
aID	destCity	mode
4	2	train

TourActivity			
aID	name	duration	extra
1	ROM	3	False
2	CN Tower	3	False
3	Zoo	2	False
5	Included	1	False
6	Optional	2	True

Itinerary				
tripID	day	step	cityID	activityID
1	1	1	1	1
1	1	2	1	2
1	2	1	1	3
1	3	1	1	4
1	4	1	2	5
1	4	2	2	6

Staff		
staffID	name	hireDate
1	staff1	Jan 1, 2023

StaffRequired	
activityID	role

StaffAssignment			
staffID	tripID	startDate	role

Traveller			
tID	name	email	citizenship
1	A	a@a.com	Canada
2	B	b@b.com	Canada
3	C	c@c.com	Canada

Visa		
traveller	country	expiry

Booking					
bID	tripID	startDate	traveller	bookedBy	price
1	1	Jan 26, 2023	1	1	100
2	1	Jan 26, 2023	2	1	100
3	1	Jan 26, 2023	3	1	100

ExtraBooking		
bID	activity	price
3	6	100

Part 2: Give the output of a query (5%)

$\Pi_{\text{activityID}}$Itinerary gives the IDs of all activities in the itinerary:

activityID
1
2
3
4
5
6

$\Pi_{\text{activityID}}(\sigma_{\text{name = 'Montreal'}}$ (Itinerary $\bowtie$ City))) gives the IDs of all activities in the itinerary that happen in Montreal:

activityID
5
6

So, Temp(aID), the set difference between the above two, is simply the IDs of all activities that occur in Toronto:

activityID
1
2
3
4

Π_{duration} (Temp $\bowtie$ TourActivity) gives the duration of all tour activities in Toronto. So, our final result is the duration of activities 1, 2, and 3:

duration
3
3
2

Part 3: Violating a constraint (10%)

In order to give an instance of the database that violates the constraint ($\Pi_{\text{tripID, startDate, role}}$ (Booking $\bowtie$ Itinerary $\bowtie$ StaffRequired)) - $\Pi_{\text{tripID, startDate, role}}$StaffAssignment $= \emptyset$, we simply add a tuple to the StaffRequired instance from Part 1. Logically, the constraint is violated if there are staff required for a trip but no staff actually assigned to it. Since the staffID attribute in StaffAssignment is just a *subset* of the staffID attribute in StaffRequired, we do not violate any of the original constraints.

Putting arbitrary values, we get that the instance from Part 1, with StaffRequired replaced with the instance below violates the constraint above.

StaffRequired	
activityID	role
1	Guide

Now, Booking $\bowtie$ Itinerary $\bowtie$ StaffRequired is shown below:

bID	tripID	startDate	traveller	bookedBy	price	day	step	cityID	activityID	role
1	1	Jan 26, 2023	1	1	100	1	1	1	1	Guide
2	1	Jan 26, 2023	2	1	100	1	1	1	1	Guide
3	1	Jan 26, 2023	3	1	100	1	1	1	1	Guide

$\Pi_{\text{tripID, startDate, role}}$ (Booking $\bowtie$ Itinerary $\bowtie$ StaffRequired) is thus:

tripID	startDate	role
1	Jan 26, 2023	Guide

However, since StaffAssignment is empty, $\Pi_{\text{tripID, startDate, role}}$StaffAssignment is also empty. Thus, the difference constraint is violated, as the final result is not an empty set, it is the table shown above.

Part 4: Queries (50%)

We write the queries below in relational algebra, using the notations and operators discussed in lecture.

1. Find all trips that visit exactly three different countries. Report the tripID, start city and country, and end city and country for all such trips.

   We will first find all trips that visit at least 3 different countries. Then we will find all trips that visit at least 4 countries. The set difference will give the tripIDs of trips that visit exactly 3 countries.

- - Start Countries for each activity. We say start country because the last step of last day could be a transport activity, so the expression below does not capture all countries (hence the suffix base).

$TripCountriesBase(tripID, country) := \Pi_{tripID,country}(Itinerary \bowtie City)$

- - Rename TransportActivity for join operation to match aID and activityID

$TransportActivityRenamed(activityID, destCity, mode) := \Pi_{aID,destCity,mode} TransportActivity$

- - Cities visited by transport activities, to account for moving to new country in last day, last step

$TripCitesTransport(tripID, destCity) := \Pi_{tripID,destCity}(Itinerary \bowtie TransportActivityRenamed)$

- - Get transport destination countries from city

$TripCountriesTransport(tripID, country) :=$
$\quad \Pi_{tripID,country}\sigma_{destCity=cityID}(TripCitiesTransport \times City)$

- - Combine the base and transport destination to get all countries visited by each trip

$TripCountries(tripID, country) := TripCountriesBase \cup TripCountriesTransport$

- - TripIDs that visit at least 3 different countries

$$Atleast3(tripID) := \Pi_{T1.tripID}(\sigma_{\substack{T1.tripID=T2.tripID \\ \wedge \\ T2.tripID=T3.tripID \\ \wedge \\ T1.country\neq T2.country \\ \wedge \\ T2.country\neq T3.country \\ \wedge \\ T3.country\neq T1.country}} [(\rho_{T1}TripCountries)\times(\rho_{T2}TripCountries)\times(\rho_{T3}TripCountries)])$$

- - TripIDs that visit at least 4 different countries

$$Atleast4(tripID) := \Pi_{T1.tripID}\ \sigma_{\substack{T1.tripID=T2.tripID \\ \wedge \\ T2.tripID=T3.tripID \\ \wedge \\ T3.tripID=T4.tripID \\ \wedge \\ T1.country\neq T2.country \\ \wedge \\ T2.country\neq T3.country \\ \wedge \\ T3.country\neq T4.country \\ \wedge \\ T4.country\neq T1.country}} [(\rho_{T1}TripCountries)\times(\rho_{T2}TripCountries)\times(\rho_{T3}TripCountries)\times(\rho_{T4}TripCountries)]$$

- - TripIDs that visit exactly 3 countries

$Exactly3(tripID) := Atleast3 - Atleast4$

Now, we will find the tripID, startCity and startCountry of each trip.

- - tripID, city and country of day 1, step1 for each trip

$$FirstStepLocation(tripID, startCity, startCountry) := \Pi_{tripID,name,country}\sigma_{\substack{day=1 \\ \wedge \\ step=1}}(Itinerary \bowtie City)$$

Now, we will find the tripID, endCity, and endCountry of each trip.

- - Rows that are not last step of last day

$\mathit{NotLastStep}(\mathit{tripID}, \mathit{day}, \mathit{step}, \mathit{cityID}, \mathit{activityID}) :=$
$\quad \Pi_{T1.tripID, T1.day, T1.step, T1.cityID, T1.activityID}(\sigma_{\substack{T1.tripID = T2.tripID \\ \wedge \\ (T1.day < T2.day \vee T1.step < T2.step)}} [(\rho_{T1}\mathit{Itinerary}) \times (\rho_{T2}\mathit{Itinerary})])$

- - Rows that are last step of last day

$\mathit{LastStep}(\mathit{tripID}, \mathit{day}, \mathit{step}, \mathit{cityID}, \mathit{activityID}) := \mathit{Itinerary} - \mathit{NotLastStep}$

- - Rename Tour Activity so we can join aId and activityID

$\mathit{TourActivityRenamed}(\mathit{activityID}, \mathit{tourname}, \mathit{duration}, \mathit{extra}) := \Pi_{aID, name, duration, extra}\mathit{TourActivity}$

- - LastStep rows that are tour activities

$\mathit{LastStepTourLocation}(\mathit{tripID}, \mathit{endCity}, \mathit{endCountry}) :=$
$\quad \Pi_{tripID, name, country}(\mathit{LastStep} \bowtie \mathit{City} \bowtie \mathit{TourActivityRenamed})$

- - TripId and destCity of LastStep rows that are transport activities

$\mathit{LastStepTransportCities}(\mathit{tripID}, \mathit{destCity}) := \Pi_{tripID, destCity}(\mathit{LastStep} \bowtie \mathit{TransportActivityRenamed})$

- - TripId and destination country of LastStep rows that are transport activities

$\mathit{LastStepTransportLocation}(\mathit{tripID}, \mathit{endCity}, \mathit{endCountry}) :=$
$\quad \Pi_{tripID, name, country}(\sigma_{destCity = cityID}(\mathit{LastStepTransportCities} \times \mathit{City}))$

- - Combine Tour and Transport activities

$\mathit{LastStepLocation}(\mathit{tripID}, \mathit{endCity}, \mathit{endCountry}) :=$
$\quad \mathit{LastStepTourLocation} \cup \mathit{LastStepTransportLocation}$

We now report final answer.

- - Combine first and last locations of each trip, then select trips of interest

$\mathit{FirstLastLocationExactly3}(\mathit{tripID}, \mathit{startCity}, \mathit{startCountry}, \mathit{endCity}, \mathit{endCountry}) :=$
$\quad (\mathit{FirstStepLocation} \bowtie \mathit{LastStepLocation}) \bowtie \mathit{Exactly3}$

2. Find all the cities where no trip is ever offered to. Report the city name and country.

- - CityIDs of all starting cities in each step

$\mathit{StepStartCity}(\mathit{cityID}) := \Pi_{cityID}\mathit{Itinerary}$

- - Rename TransportActivity for join operation to match aID and activityID

$\mathit{TransportActivityRenamed}(\mathit{activityID}, \mathit{destCity}, \mathit{mode}) := \Pi_{aID, destCity, mode}\mathit{TransportActivity}$

- - Cities visited by transport activities, to account for moving to new city in last day, last step

$\mathit{DestinationCities}(\mathit{cityID}) := \Pi_{destCity}(\mathit{Itinerary} \bowtie \mathit{TransportActivityRenamed})$

- - All cities visited; combine start cities and destination cities

$\mathit{AllCitiesVisited}(\mathit{cityID}) := \mathit{StepStartCity} \cup \mathit{DestinationCities}$

- - Cities not visited

$\mathit{CityNoTrip}(\mathit{name}, \mathit{country}) := \Pi_{name, country}((\Pi_{cityID}\mathit{City} - \mathit{AllCitiesVisited}) \bowtie \mathit{City})$

3. Find the trip(s) that requires the most staff (not how many different staff are actually assigned). Report the tripIDs and roles required.

   Cannot be expressed.

   We can obtain tripIDs which occur a predefined k times, and via set operations get those which occur more or less than that. If we know beforehand that the max number of roles is 11 for example, we could find the tripId(s) and corresponding roles with exactly 11 required roles. However, we do not know beforehand what the max number of roles are.

4. Find any days on any trips that have a total duration of tour activities that exceeds 12 hours. Report the tripIDs and the days.

   Cannot be expressed.

   Relational algebra, unlike SQL, does not support aggregation. So, while we can join the Itinerary and TourActivity tables, we do not have a tool that supports making sure the duration of tour activities exceeds 12 hours.

5. For each trip, find the most and second-most expensive extra booking(s) made. Report the trip IDs, activity IDs, and prices of these bookings. If there are ties, report them all.

   - - TripID, activity and price for all extra bookings

   $ExtraBookings(tripID, activity, price) := \Pi_{tripID, activity, price}((\Pi_{bID, tripID} Booking) \bowtie ExtraBooking)$

   - - ExtraBookings that are strictly cheaper than at least two other extra bookings

   $$NotTop2(tripID, activity, price) := \Pi_{T1.tripID, T1.activity, T1.price} \sigma_{\substack{T1.tripID = T2.tripID \\ \wedge \\ T2.tripID = T3.tripID \\ \wedge \\ T1.price < T2.price \\ \wedge \\ T2.price < T3.price}} [(\rho_{T1} ExtraBookings) \times (\rho_{T2} ExtraBookings) \times (\rho_{T3} ExtraBookings)]$$

   - - tripID, activity, and price of most and second most expensive extra bookings

   $Top2Extra(tripID, activity, price) := ExtraBookings - NotTop2$

6. A staff member is said to have worked as a guide on a trip if they were assigned a role 'guide' on a trip for any start date. Find all staff members who were hired before 2020 who have worked as a guide on every trip. Report the staff IDs.

   - - staffID and tripID where staff with ID staffID is hired before 2020 and role on trip with ID tripID is 'guide'

   $$StaffTrips(staffID, tripID) := \Pi_{staffID, tripID}(\sigma_{\substack{hiredate.year < 2020 \\ \wedge \\ role = 'guide'}}(StaffAssignment \bowtie Staff))$$

   - - staffID and tripID where staff was either not hired before 2020 or was not a guide on that trip $MissingPairs(staffID, tripID) := (\Pi_{staffID} Staff \times \Pi_{tripID} Trip) - StaffTrips$

   - - staffID, where staff is hired before 2020 and is guide on every trip

   $GuideEveryTrip(staffID) := \Pi_{staffID} Staff - \Pi_{staffID} MissingPairs$

7. Find all trips that visit at least one new city every day. Report the tripIDs.

   - - days in itinerary for each trip

   $ItineraryDays(tripID, day) := \Pi_{tripID, day} Itinerary$

- - days where transport is required to meet the criteria.

$TransportRequired(tripID, day) := \sigma_{day \neq 1} ItineraryDays$

- - Days in each trip where we have a transport activity

$TransportDays(tripID, day) := \Pi_{tripID, day}(\sigma_{activityID = aID}(Itinerary \times TransportActivity))$

- - tripID, day pairs that are missing a transport activity

$MissingTransportDays(tripID, day) := TransportRequired - TransportDays$

- - tripIDs that visit a new city each day

$DesiredTripIDs(tripID) := \Pi_{tripID} Itinerary - \Pi_{tripID} MissingTransportDays$

8. Travellers are considered "influencers" if they are booked on the earliest booking for every trip they are booked on, and every trip they are booked on has at least two people booked on a later date. An influencer must be booked on at least one trip. Find all travellers who are influencers. Report their names and email addresses.

- - tripID, startDate pairs in booking table

$BookingDays(tripID, startDate) := \Pi_{tripID, startDate} Booking$

- - tripID, startDate pairs that do not correspond to earliest date for a trip

$NotEarliestDay(tripID, startDate) :=$
$\Pi_{T1.tripID, T1.startDate} \sigma_{\substack{T1.tripID = T2.tripID \\ \wedge \\ T1.startDate < T2.startDate}} [(\rho_{T1} BookingDays) \times (\rho_{T2} BookingDays)]$

- - earliest date for each trip

$EarliestDay(tripID, earliestDay) := BookingDays - NotEarliestDay$

- - trip, startDate, traveller combinations

$Travels(tripID, startDate, traveller) := \Pi_{tripID, startDate, traveller} Booking$

- - tripID, startDate, traveller that correspond to travel date being before atleast 2 people

$PossibleInfluencers(tripID, startDate, traveller) :=$
$\Pi_{T1.tripID, T1.startDate, T1.traveller}(\sigma_{\substack{T1.tripID = T2.tripID \\ \wedge \\ T2.tripID = T3.tripID \\ \wedge \\ T1.startDate < T2.startDate \\ \wedge \\ T1.startDate < T3.startDate \\ \wedge \\ T1.traveller \neq T2.traveller \\ \wedge \\ T2.traveller \neq T3.traveller \\ \wedge \\ T3.traveller \neq T1.traveller}} [(\rho_{T1} Travels) \times (\rho_{T2} Travels) \times (\rho_{T3} Travels)])$

- - TripID, traveller where traveller is booked for tripID but is not influencer for that trip (earliest and atleast 2 people after)

$Missing(tripID, traveller) := \Pi_{tripID, traveller}(Booking) - \Pi_{tripID, traveller}(\sigma_{startDate = earliestDay}(PossibleInfluencers \bowtie EarliestDay))$

- - Traveller IDs for all influencers

$InfluencerIDs(tID) := \Pi_{traveller} Booking - \Pi_{traveller} Missing$

- - Name and email address of all influencers

$Influencers(name, email) := \Pi_{name,email}(Traveller \bowtie InfluencerIDs)$

9. Find all travellers who have spent more than \$10000 in total on trips with this travel company. Report their names and email addresses.

   Cannot be expressed.

   Since relational algebra does not support aggregation, we cannot find travellers who spent more than 10000 in total, as "in total" suggests aggregation.

10. Find all pairs of staff who have worked together on a booked trip (i.e. same tripID and start date). Report the IDs of those staff, and do not report pseudoduplicates.

    - - staff assigned to each trip offering

    $StaffTrips(tripID, startDate, staffID) :=$
    $\Pi_{tripID,startDate}Booking \bowtie \Pi_{tripID,startDate,staffID}StaffAssignment$

    - - staff that worked together

    $WorkedTogether(staff1, staff2) :=$
    $$\Pi_{T1.staffID,T2.staffID}\sigma_{\substack{T1.tripID=T2.tripID \\ \wedge \\ T1.startDate=T2.startDate \\ \wedge \\ T1.staffID<T2.staffID}}[(\rho_{T1}StaffTrips) \times (\rho_{T2}StaffTrips)]$$

Part 5: Additional Integrity Constraints (25%)

Express the following integrity constraints with the notation $R = \emptyset$, where R is an expression of relational algebra. You are welcome to define intermediate results with assignment. The last step for each question must be a single assertion of the form expression $= \emptyset$.

Remember that at least one of the queries and/or integrity constraints in this assignment cannot be expressed in the language that you are using. In those cases, simply write "cannot be expressed".

1. All trips must have a Day 1 in the itinerary, and all days following Day 1 (if any) must be consecutively numbered with no gaps.

   This constraint can be decomposed as the intersection of the two following "sub-constraints":

   A: All trips must have a day 1 in the itinerary.

   B: For all trips, all days following day 1 must be consecutive with no gaps.

   A:

   - - All tripIDs in Itinerary which have a day 1.

   $TripsWithDay1(tripID) := \Pi_{tripID}(\sigma_{day=1}(Itinerary))$

   - - All tripIDs. Since all trips must have a day 1, use Trip and not Itinerary here.

   $Trips(tripID) := \Pi_{tripID}Trip$

   - - The difference between all trips and trips in the itinerary with a day 1.

   $A(tripID) := Trips - TripsWithDay1$

   B:

   - - All (tripID, day) tuples in the itinerary where day is not equal to 1.

   $NotDay1Trips(tripID, day) := \Pi_{tripID,day}(\sigma_{day\neq1}(Itinerary))$

- - All pairs of (tripID, day) tuples from the itinerary which have the same trip ID and consecutive days (where none of the days are day 1).

$Pairs(B1.tripID, B1.day, B2.tripID, B2.day) := \sigma_{\substack{B1.tripID=B2.tripID, \\ B2.day-B1.day=1}}(\rho_{B1}NotDay1Trips \times \rho_{B2}NotDay1Trips)$

- - (tripID, day) tuples which make up the **first partner** in the pair of (tripID, day) tuples in the Pairs relation.

$ConsecutiveDays1(tripID, day) := \Pi_{B1.tripID,B1.day}Pairs$

- - (tripID, day) tuples which make up the **second partner** in the pair of (tripID, day) tuples in the Pairs relation.

$ConsecutiveDays2(tripID, day) := \Pi_{B2.tripID,B2.day}Pairs$

- - All (tripID, day) tuples where day is greater than 1 and the days are consecutive.

$B(tripID, day) := NotDay1Trips - (ConsecutiveDays1 \cup ConsecutiveDays2) = \emptyset$

- - **Final constraint:** We want both A and B to be null.

$A \cup \Pi_{tripID}B = \emptyset$

2. All activities that are extra bookings must be on the itinerary for the trip.

   Two relations are relevant to this question: Itinerary and ExtraBooking.

   - - All activityIDs in the Itinerary which correspond to activities in ExtraBooking.

   $ExtraBookingActivitiesInItinerary(activityID) := \Pi_{activityID}(Itinerary \bowtie \rho_{(bID,activityID,price)}ExtraBooking)$

   - - All activities in ExtraBooking.

   $ExtraBookingActivities(activityID) := \Pi_{activity}ExtraBooking$

   - - **Final constraint:** The difference between activites in ExtraBooking and the ExtraBooking activites in Itinerary must be 0.

   $ExtraBookingActivities - ExtraBookingActivitiesInItinerary = \emptyset$

3. All travellers require a valid visa to visit any country other than the one they are a citizen of. A visa is valid if it exists and expires after the start date of the trip. (We won't worry about it being valid for the whole duration of the trip, and we will make the assumption everyone needs a visa for every country except their own.)

   - - For all travellers, the startDate of the trip and beginning country of activity for all activities they have booked.

   $BookedTravels(traveller, startDate, country) := \Pi_{traveller,startDate,country}(Booking \bowtie Itinerary \bowtie City)$

   - - The country in which a transportation activity ends in (for all transportation activities).

   $TransportActivityDestCountry(activityID, destCountry) :=$
   $\Pi_{aID,country}(\sigma_{destCity=cityID}(TransportActivity \times City))$

   - - The start date of trip and destination country of every transport activity a traveller books.

   $BookedTransportTravels(traveller, startDate, country) :=$
   $\Pi_{traveller,startDate,destCountry}(Booking \bowtie Itinerary \bowtie TransportActivityDestCountry)$

   - - The start date of trip and all countries visited for all travellers.

$AllTravels(traveller, startDate, country) := BookedTravels \cup BookedTransportTravels$

- - Every country that a traveller has citizenship for (for all travellers).

$Citizenship(traveller, citizenship) := \Pi_{tID, citizenship} Traveller$

- - The start date, and destination country for all countries that a traveller does not have citizenship for but intends to visit (based on their bookings).

$NeedVisaBooked(traveller, startDate, country) := \Pi_{traveller, startDate, country}(\sigma_{country \neq citizenship}(AllTravels \bowtie Citizenship))$

- - All the (trip, country visited) pairs the traveller has a valid visa for.

$HaveVisaBooked(traveller, startDate, country) := \Pi_{traveller, startDate, country}(\sigma_{expiry > startDate}(NeedVisaBooked \bowtie Visa))$

- - The trips that a traveller intends to go on, but does not have the required visa(s) for.

$IllegalBooked(traveller, startDate, country) := NeedVisaBooked - HaveVisaBooked$

- - **Final Constraint**:

IllegalBooked = ∅

4. No trip can have a day that visits more than two countries.
   - - The tripID, day, and start country for every activity in the Itinerary.

   $ItineraryStartCountries(tripID, day, country) := \Pi_{tripID, day, country}(Itinerary \bowtie City)$

   - - The tripID, day, and destination city for every transport activity in the Itinerary

   $ItineraryDestCities(tripID, day, cityID) := \Pi_{tripID, day, destCity} \sigma_{activityID = aID}(Itinerary \times TransportActivity)$

   - - The tripID, day, and destination country for every transport activity in the Itinerary

   $ItineraryDestCountries(tripID, day, country) := \Pi_{tripID, day, country}(ItineraryDestCities \bowtie City)$

   - - The overall tripID, day and country visited by all activities

   $ItineraryCountries(tripID, day, country) := ItineraryStartCountries \cup ItineraryDestCountries$

   - - The tripID, day, and country for every trip that has visited at least three countries in a day.

   $$Atleast3Countries(tripID, day, country) := \Pi_{T1.tripID, T1.day, T1.country} \sigma_{\substack{T1.tripID = T2.tripID \\ \wedge \\ T2.tripID = T3.tripID \\ \wedge \\ T1.day = T2.day \\ \wedge \\ T2.day = T3.day \\ \wedge \\ T1.country \neq T2.country \\ \wedge \\ T2.country \neq T3.country \\ \wedge \\ T3.country \neq T1.country}} [\rho_{T1} ItineraryCountries \times \rho_{T2} ItineraryCountries \times \rho_{T3} ItineraryCountries]$$

   - - **Final Constraint**: There should be no trip that visits at least 3 countries in a day.

   Atleast3Countries = ∅

5. No trip can have two transportation activities back to back, including the last activity on one day and the first activity on the next.
   - - Get the tripID, day, and step for all transport activities in Itinerary.

$TransportActivities(tripID, day, step) :=$
$\Pi_{tripID,day,step}\sigma_{activityID=aID}(Itinerary \times TransportActivity)$

- - The tripID, day, and step of all transport activities that occur on the same day, and within one step of each other.

$BacktoBackSameDay(tripID, day, step) :=$
$\Pi_{T1.tripID,T1.day,T1.step}(\sigma_{\substack{T1.tripID=T2.tripID\\ \wedge\\ T1.day=T2.day\\ \wedge\\ T2.step=T1.step+1}}[\rho_{T1}TransportActivities \times \rho_{T2}TransportActivities])$

- - All unique (tripID, day, step) tuples in Itinerary.

$AllSteps(tripID, day, step) := \Pi_{tripID,day,step}Itinerary$

- - All (tripID, day, step) tuples in Itinerary where the step is not the last step of a day in a trip.

$NotLastStep(tripID, day, step) :=$
$\Pi_{T1.tripID,T1.day,T1.step}\sigma_{\substack{T1.tripID=T2.tripID\\ \wedge\\ T1.day=T2.day\\ \wedge\\ T1.step<T2.step}}[AllSteps \times AllSteps]$

- - The number of steps per day for each trip.

$StepsPerDay(tripID, day, length) := \Pi_{tripID,day,step}(AllSteps - NotLastStep)$

- - All pairs of transport steps that occur within a particular trip (for all trips).

$TransportStepPairs(tripID, day, step, day2, step2) :=$
$\Pi_{T1.tripID,T1.day,T1.step,T2.day,T2.step}(\sigma_{T1.tripID=T2.tripID}[\rho_{T1}TransportActivities \times \rho_{T2}TransportActivities])$

- - Find consecutive transportation activities on different days.

$BacktoBackDiffDays(tripID, day, step) :=$
$\Pi_{tripID,day,step}\sigma_{\substack{day2=day+1\\ \wedge\\ step2=1\\ \wedge\\ step=length}}(TransportStepPairs \bowtie StepsPerDay)$

- - **Final constraint**: we wont both back to back on same day and different days to be null.

$BacktoBackSameDay \cup BacktoBackDiffDays = \emptyset$

6. A trip booking (tripID, startDate) where the price is above the average price of all bookings for that trip (not including extras) must have at least two different staff assigned to that trip.

   Cannot be expressed.

   Average price requires aggregation, which is not supported by relational algebra.