University of Toronto
CSC343, Fall 2020

Assignment 1

Due: Thursday 15 October, before 8:00 p.m.

Learning Goals

By the end of this assignment you should be able to:

- Read a new relational schema and determine whether or not a particular instance is valid with respect to that schema.
- Apply the individual techniques for writing relational algebra queries and integrity constraints that we learned in class.
- Combine the individual techniques to solve complex problems.
- Identify problems that cannot be solved using relational algebra.

These skills will leave you well prepared to be a strong SQL programmer.

Schema

For this assignment, you will write queries and integrity constraints on a database for a travel company.

Relations

- City($\underline{\text{cityID}}$, name, region, country)
  A tuple in this relation represents a city in which this company does business. *name* is the name of the city, *country* is the country that the city is in, and *region* is the part of the country it is in.
- Trip($\underline{\text{tripID}}$, name, description, startCity)
  A tuple in this relation represents a trip offered by our tour company. *name* is the name of the trip, such as "Northern Lights Explorer", *description* is a longer description of the trip, and *startCity* is the city in which the trip begins. We consider the first day of the trip to be day 1 (as opposed to day 0).
- Itinerary($\underline{\text{tripID, cityID, day}}$, transportationType)
  A tuple in this relation represents the fact that this trip arrives in this city on this day of the trip. *day* is an integer (rather than a date), and *transportationType* is the means of transportation used to arrive at this city.
- BaseActivity($\underline{\text{tripID, cityID, name}}$)
  A tuple in this relation represents a "base activity", that is one included in the cost of any instance of this trip. *tripID* is the trip that includes this activity, *cityID* is the city on the trip in which this activity occurs, and *name* is the name of the activity, such as "snorkelling" or "Tower of London tour".
- AddOnActivity($\underline{\text{addID}}$, $\underline{\text{tripID, cityID, name}}$, cost)
  A tuple in this relation represents an activity that is an "add on", in other words, is not included in the cost of any instance of this trip. *tripID* is the trip that offers this add on activity, *cityID* is the city on the trip in which this activity occurs, *name* is the name of the activity, and *cost* is the cost a customer must pay to participate in this activity.

- Staff(staffID, fname, lname, employDate, phone, email, address)
  A tuple in this relation represents a staff member at the company. *fname* and *lname* record the staff member's name, *employDate* is the date on which this staff member began their employment at the company, *phone* is their phone number, *email* is their email address, and *address* is the address of their place of residence.

- TripInstance(instID, tripID, tripManager, startDate, price)
  A tuple in this relation represents a particular instance of a trip. Instances of a trip can differ from each other in various ways. For example, the "Northern Lights Explorer" trip could be offered starting on the first Saturday of each month, with prices varying by season. *tripID* is the trip of which this is an instance, *tripManager* is the staff member who is managing this instance, *startDate* is the day on which the trip begins, and *price* is the advertised price for this instance of this trip.

- TripInstanceTeam(instID, staffID, position)
  A tuple in this relation records the fact that this staff member is working on this trip instance. *position* is their role on the trip.

- Traveller(travID, fname, lname, email, dob, address, phone)
  A tuple in this relation represents a customer. *fname* and *lname* record the traveller's name, *email* is their email address, *dob* is their date of birth, *address* is the address of their place of residence, and *phone* is their phone number.

- Booking(instID, travID, roomType, creditCard, pricePaid)
  A tuple in this relation records the fact that this traveller booked this trip instance. *roomType* is an integer indicating the type of room this traveller requested, (*e.g.*, 2 for a double room), *creditCard* is their credit card number, and *pricePaid* is how much they paid for this booking. A traveller may pay less than the advertised price if, for example, they have a discount code. (Our database does not keep track of discount codes.)

- Review(tripID, travID, stars, text)
  A tuple in this relation represents a review by this traveller for this trip. *stars* is an integer representing the number of stars they gave the trip, and *text* is the written content of their review.

- AddOnBooking(instID, travID, addID)
  A tuple in this relation records the fact that, on this trip instance, this traveller booked this add-on activity.

- Accommodation(accID, name, accType, phone, cityID)
  A tuple in this relation represents a hotel or other place of accommodation that the company does business with. *name* is the name of the accommodation, such as "Hotel de Sevigne", *accType* is the type of accommodation it is, *phone* is its phone number, and *cityID* is the city where it is.

- Room(accID, roomSize)
  A tuple in this relation represents the fact that this accommodation has rooms of this size. *roomSize* is an integer indicating the number of people that the room can sleep.

- TripAccommodation(instID, accID)
  A tuple in this relation represents the fact that this trip instance offers accommodation at this place. The database currently does not record which travellers get booked into which accommodation or which room within it.

Integrity constraints

- Trip[startCity] ⊆ City[cityID]
- Itinerary[tripID] ⊆ Trip[tripID]
- Itinerary[cityID] ⊆ City[cityID]
- Itinerary[transportationType] ∈ {“bus”, “train”, “ferry”}
- BaseActivity[tripID] ⊆ Trip[tripID]
- BaseActivity[cityID] ⊆ City[cityID]
- AddOnActivity[tripID] ⊆ Trip[tripID]
- AddOnActivity[cityID] ⊆ City[cityID]
- TripInstance[tripID] ⊆ Trip[tripID]
- TripInstance[tripManager] ⊆ Staff[staffID]
- TripInstanceTeam[instID] ⊆ TripInstance[instID]
- TripInstanceTeam[staffID] ⊆ Staff[staffID]
- TripInstanceTeam[position] ∈ {“driver”, “guide”}
- Booking[instID] ⊆ TripInstance[instID]
- Booking[travID] ⊆ Traveller[travID]
- Review[tripID] ⊆ Trip[tripID]
- Review[travID] ⊆ Traveller[travID]
- AddOnBooking[instID, travID] ⊆ Booking[instID, travID]
- AddOnBooking[addID] ⊆ AddOnActivity[addID]
- Accommodation[cityID] ⊆ City[cityID]
- Accommodation[accType] ∈ {“hotel”, “premium hotel”, “hostel”, “overnight coach”, “cruise ship”, “apartment”, “chalet”}
- Room[accID] ⊆ Accommodation[accID]
- TripAccommodation[instID] ⊆ TripInstance[instID]
- TripAccommodation[accID] ⊆ Accommodation[accID]
- $\sigma_{\substack{It1.tripID=It2.tripID \\ \wedge \\ It1.day=It2.day \\ \wedge \\ It1.city\neq It2.city}}[\rho_{It1}Itinerary \times \rho_{It2}Itinerary] = \emptyset$

Warmup: Getting to know the schema

To get familiar with the schema, ask yourself questions like these (but don't hand in your answers):

1. Imagine a trip with the following itinerary:
   - The trip begins in Edinburgh.
   - On day 4, travellers go by train to Inverness.
   - On day 7, they go by coach to Fort William.
   - On day 8, they go by train to Mallaig. (This last leg of the journey takes the same route that the Harry Potter train takes to Hogwart's!)

   How would this information be recorded in the database?
2. Why is day part of the key for relation Itinerary?
3. A trip instance can have different staff associated with it as trip manager, driver, or guide. Why do you think the trip manager information was included in the TripInstance relation, while the other roles are represented in the TripInstanceTeam relation?
4. How many places of accommodation can a trip instance have in a city?
5. Can travellers rate a specific instance of the trip they took, or just the trip in general? (They can include words that talk about whatever they want, but what does the database associate their review with?)
6. Why does AddOnBooking have one constraint involving both instID and travID rather than a separate constraint for each (as with addID)?
7. Can you think of possibly important constraints that have not been expressed? Think about instances that the company might not want to allow but that the schema does allow.

Part 1: Queries

Write the queries below in relational algebra. There are a number of variations on relational algebra, and different notations for the operations. You must use the same notation as we have used in this course. You may use assignment, and the operators we have used in class: $\Pi, \sigma, \bowtie, \bowtie_{condition}, \times, \cap, \cup, -, \rho$. Assume that all relations are sets (not bags), as we have done in class, and do not use any of the extended relational algebra operations from Chapter 5 (for example, do not use the division operator).

Some additional points to keep in mind:

- Do not make any assumptions about the data that are not enforced by the original constraints given above, including the ones written in English. Your queries should work for any database that satisfies those constraints.
- Assume that every tuple has a value for every attribute. For those of you who know some SQL, in other words, there are no null values.
- Remember that the condition on a select operation may only examine the values of the attributes in one tuple, not whole columns. In other words, to use a value (other than a literal value such as 100 or "Adele"), you must get that value into the tuples that your select will examine.

- The condition on a select operation can use comparison operators (such as $\leq$ and $\neq$) and boolean operators ($\vee$, $\wedge$ and $\neg$). Simple arithmetic is also okay, *e.g.*, attribute1 $\leq$ attribute2 + 5000.
- In your select conditions, you can compare dates using comparison operators such as $<$.
- Some relations in our schema have a date-time attribute. You may use comparison operators on such values. You may refer to the year component of a date-time attribute d using the notation d.year.
- You are encouraged to use assignment to define intermediate results.
- It's a good idea to add commentary explaining what you're doing. This way, even if your final answer is not completely correct, you may receive part marks.
- The order of the columns in the result doesn't matter.
- If there are ties, report all of them.

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". Note: The queries are not in order according to difficulty.

1. Context: Bus trips have been falling in popularity. We are reviewing our pricing for these, as well as frequent traveller discounts, and arrangements with resellers like TravelZoo and Groupon.

   Query: For each instance of a trip that includes transportation by bus, find the highest price paid by any traveller for that instance, and find the lowest price paid by any traveller for that instance. Report the trip instance ID and advertised price, the highest price paid, and the lowest price paid.

2. Context: We are reviewing trips that haven't been offered recently, with a plan to rethink whether they might be more appealing if they visited more cities.

   Query: For each trip that has had at least 2 instances but none in 2019 or since, report the trip id and name, and the number of stops on its itinerary.

3. Context: The company is planning a special promotion for people who have taken extensive trips to Paris.

   Query: Find all travellers who have booked at least one instance of each trip that starts in Paris and includes 3 or more cities (the start city counts towards this total). Report the traveller's first and last name, and email address.

4. Context: We are looking for travellers who like a lot of perqs. We might design a special trip aimed at this market.

   Query: Let's define a plush trip as one with the most base activities. (There could be exactly one plush trip, or perhaps there are several that are tied.) Report the name and email address of all travellers who have taken one or more of these plush trips, have never taken a non-plush trip, and have never booked an add-on activity.

5. Context: The company is planning a staff retreat. Some of the activities will be done in pairs, and we'd like to group people who have experiences in common.

   Query: Find all pairs of staff who've both been trip manager for different instances of the same trip. For each pair, report both people's email address, and the start date of the very first trip instance they were manager for. (It might not be an instance of the/a trip they have in common with the other person.) Use attribute names staff1, start1, staff2, start2 and make the person represented by the first two attributes be the one from the pair with the most seniority (that is, the one whose employment date comes first). If there is a tie, it doesn't matter whose data is represented by the first two attributes.

6. Context: We are designing a new trip for very adventurous travellers and are targeting it towards people who are willing to spend money on add-on activities.

   Query: Find travellers who have booked more than one trip instance and on every trip instance they've booked, they booked all of the add on activities available. Report simply the email address of each such traveller.

7. Context: We think some of our reviews may be fake.

   Query: Let's say a low-ball rating for a trip is one that (a) is at most 2 stars, (b) is the lowest rating (lowest number of stars) for that trip, and (c) is unique for that trip: no one else gave that trip the same number of stars. Find travellers who've given at least one review and whose every review either gives a low-ball rating or has text consisting of just the word "terrible". For each such traveller, report just their email address.

8. Context: We are looking for some social media influencers to expand our sales of trips to South America, so we want to hire a young traveller who has had a positive outlook on our trips to Brazil.

   Query: Find all travellers who've rated a trip that includes Rio de Janeiro and Sao Paulo, and have given that trip a rating that is above the average rating for that trip. Report the traveller email, and first and last name. Include only travellers whose birth year is between 1995 and 2002 inclusive.

Part 2: Additional Integrity Constraints

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. Context: We don't want to book a traveller on an instance of a trip if it's clear that they will not have their accommodation needs met.

   Constraint: A traveller cannot book an instance of a trip unless there is, in every city on its itinerary (including the start city), an accommodation for that trip instance that offers either a room of the size they requested in their booking or a larger room.

2. Context: The company is concerned about the legitimacy of reviews.

   Constraint: You can only review a trip if you've booked at least one instance of it. (However, nothing stops you from reviewing a trip before you go on that trip.)

3. Context: The company needs to recoup the lost income from discounts by making money on add-on activities.

   Constraint: If a traveller does not book either the most expensive or second-most expensive add-on activity on a trip instance, they can't get a discount price for that trip instance (that is, they can't pay less than the advertised price). This constraint applies only to instances of trips that have two or more add-on activities.

Formatting instructions

Your assignment must be typed; handwritten assignments will not be marked. You may use any word-processing software you like. Many academics use LaTeX. It produces beautifully typeset text and handles mathematical notation well. If you would like to learn LaTeX, there are helpful resources online. Whatever you choose to use, you need to produce a final document in pdf format.

If you use software that lets you choose a font size, it must be at least 10. If you use LaTeX, the default font size (or larger) is acceptable.

Submission instructions

You must declare your team (whether it is a team of one or two students) and hand in your work electronically using the MarkUs online system. Instructions for doing so are posted on the Assignments page of the course website. Well before the due date, you should declare your team and try submitting with MarkUs. You can submit an empty file as a placeholder, and then submit a new version of the file later (before the deadline, of course); look in the "Replace" column.

For this assignment, hand in just one file: A1.pdf. If you are working in a pair, only one of you should hand it in.

Check that you have submitted the correct version of your file by downloading it from MarkUs; new files will not be accepted after the due date.

How your assignment will be marked

Most of the marks will be for the correctness of your answers. However, there will be additional marks allocated to each of these:

- Comments:
  Does every assignment statement have a comment above it specifying clearly exactly what rows get to be in this relation? Comments should describe the data (*e.g.*, "The student number of every student who has passed at least 4 courses.") not how to find it (*e.g.*, "Find the student numbers by self-joining . . ."). Put comments *before* the assignment, and two dashes on each line of your comment.
- Attribute names given on the LHS:
  Does every assignment statement name the attributes on the LHS? This should be done even if the names are not being changed. Together with the comments, it allows you to understand what a "subquery" is supposed to do without reading it. Think of this as analogous to good parameter names and good comments on a function.
- Relation and attribute names:
  Does every relation and every attribute have a name that will assist the reader in understanding the query quickly? Apply the same high standards you would when writing code.
- Formatting:
  Is the algebra formatted with appropriate line breaks and indentation to reveal the structure of the algebra for ease of understanding?