Assignment 1

Due: June 3rd, at 11:59 pm sharp!

Introduction

For this assignment, you will be writing queries in relational algebra on a database. We are providing the schema for you. Later in the course, you will learn about how to develop your own schema based on knowledge of the domain. Even though developing a schema is really the first step in building a database, it is a more advanced task than querying an existing database; this is why we will be learning about it and practising it later.

About the domain

The schema below represents data for a company that operates taxicabs in Toronto. The data consists of details about vehicles, drivers, customers and travel bookings. A customer can make a booking for a cab indicating a pickup location, time and the total number of passengers. Once a booking is made, a cab and driver are assigned to pick up the customer as part of a travel. The customer may not be picked up at the same time as indicated in the booking. However, he/she is always picked up at the same location as indicated in the booking. All other relevant details are laid out in the schema.

Remember that all your queries will be written with respect to the schema below. Your goal is to produce answers that are correct with respect to this schema, even if they aren't quite correct in the real world.

Schema

Relations

- Driver(driverID, name, age, gender, joinDate, salary)
  A tuple in this relation represents a driver employed by the taxicab company.
  driverID is the driver ID, *joinDate* is the date on which the driver joined the company, and *salary* is the monthly salary of the driver paid by the company (in Canadian dollars).

- Cab(cabID, manufacturer, model, deploymentDate, type, capacity)
  A tuple in this relation represents a cab operated by the company.
  cabID is the cab ID, *manufacturer* is the manufacturer name (e.g. 'Tesla', 'Mercedes'), *model* is the model name (e.g. 'Model S', 'E Class'), *deploymentDate* is the date on which the cab was first deployed for service , *type* is the type of cab (e.g. 'sedan', 'limousine', 'bus'), and *capacity* is the maximum number of passengers that the cab can carry.

- Customer(customerID, name, gender, registrationDate)
  A tuple in this relation represents a customer of the taxicab company.
  customerID is the customer ID, *registrationDate* is the date on which the customer registered with the company.

- Location(locationID, name, zipcode)
  A tuple in this relation represents a location in the city where the company has operated in the past or will operate in future.

locationID is the location ID, *name* is the name of the location (e.g. '225 Bloor St', '60 Harbord St'), and *zipcode* is the zipcode of the location (e.g. 'M5S1G2')

- Booking(bookingID, customerID, noOfPassengers, bookingTime, pickupLocationID, pickupTime, status)
  A tuple in this relation represents a booking made by a customer.
  bookingID is the booking ID, *noOfPassengers* is the number of passengers, *bookingTime* is the timestamp at which the customer made the booking, *pickupLocationID* is the ID of the location from which the customer wants to be picked up, *pickupTime* is the timestamp at which the customer wants to be picked up, and *status* is the status of the booking (one of 'confirmed', 'cancelled' or 'waiting')

- Travel(travelID, bookingID, cabID, driverID, pickupTime, dropTime, dropLocationID, distance, status, cost)
  A tuple in this relation represents a single instance of travel involving a customer, driver and cab.
  travelID is the travel ID, *pickupTime* and *dropTime* are the timestamps at which the customer was picked up and dropped respectively, *dropLocationID* is the ID of the location at which the customer was dropped, *distance* is the distance travelled by the cab during the travel (from the time of pickup to drop), *status* is the status of the travel (one of 'in progress', 'completed' or 'terminated'), and *cost* is the cost of the travel (in Canadian dollars)

Integrity constraints

- Booking[customerID] $\subseteq$ Customer[customerID]
- Booking[pickupLocationID] $\subseteq$ Location[locationID]
- Booking[status] $\subseteq$ {'confirmed', 'cancelled','waiting'}
- Travel[bookingID] $\subseteq$ Booking[bookingID]
- Travel[cabID] $\subseteq$ Cab[cabID]
- Travel[driverID] $\subseteq$ Driver[driverID]
- Travel[pickupLocationID] $\subseteq$ Location[locationID]
- Travel[dropLocationID] $\subseteq$ Location[locationID]
- Travel[status] $\subseteq$ {'in progress', 'completed', 'terminated'}

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 class and on the slides. 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.

Additional points to keep in mind:

- Do not make any assumptions about the data that are not enforced by the original constraints given in the schema above. 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), and that it can use comparison operators (such as $\leq$ and $\neq$) and boolean operators ($\vee$, $\wedge$ and $\neg$).
- Some of these queries and questions involve working with dates and timestamps. In your queries, assume that you may do arithmetic and comparisons on dates and timestamps. Also, you can use hour, day, month and year for comparing dates and timestamps. For example,

  date1 > date2 (i.e., date1 is later than date2)
  time1 - time2 < 10 hours (i.e., time1 is later than time2 by less than 10 hours)
  date1 - date2 > 1 year (date 1 is later than date2 by more than a year)

- You are encouraged to use assignment to define intermediate results, and 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.
- When asked for a maximum or minimum or other such cases, if there are ties, report all of them.

Write queries for each of the following. If a question cannot be expressed, simply write "cannot be expressed":

1. Report the names of the drivers who have never driven for more than 50 kms each in two different travels.
2. Report the zipcode of the location at which the most number of passengers were dropped in a single travel. Assume that all passengers travelling in a cab are dropped together at the same location.
3. Report the names of the customers who have been picked up from at least two different locations in travels that were 'completed'.
4. Report the manufacturer and model names of cabs which were always driven by the same driver. Also report the name of the driver.
5. Report the type of cab which was filled (with passengers) to capacity exactly twice. Also report its capacity.
6. Report the name and age of all drivers who are youngest and have driven a cab with the oldest date of deployment.
7. Report the ID of the customer who took the longest time to book a cab after registering with the company. You may use comparison operators to compare dates and timestamps with each other.
8. Report the name and salary of the driver who is paid the second highest salary among all female drivers.
9. Report the cost of the shortest distance travel for which the booking was 'confirmed' or the travel was 'terminated'.
10. A driver is considered "flexible" if he/she has driven every *type* of cab. Report the names of all "flexible" drivers.

Part 2: Additional Integrity Constraints

Express the following integrity constraints with the notation $R = or \neq \emptyset$, where R is an expression of relational algebra. If a constraint cannot be expressed using this notation, simply write "cannot be expressed".

You are welcome to define intermediate results with assignment and then use them in an integrity constraint.

1. No driver can have a monthly salary less than \$2000.
2. A customer can not make a booking before his/her registration date.
3. Every cab must be of type 'sedan', 'limousine' or 'bus'.
4. There exist drivers who have never driven a 'limousine' as part of some travel.
5. A driver can pick up customers only after at least an hour has passed since another driver dropped customers using the same cab.

Submission 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. Whatever you choose to use, you need to produce a final document in pdf format, and you must call it "a1.pdf" (with no capital letters).

You are allowed, and in fact encouraged, to work with a partner for this assignment. You must declare your team (whether it is a team of one or of 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.

Once you have submitted, be sure to check that you have submitted the correct version; new or missing files will not be accepted after the due date.