U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Base de Dados

Carpooling App

26 de Maio de 2019

2MIEIC01 – Grupo 104

Francisco Batista up201604320@fe.up.pt

João Rocha up201708566@fe.up.pt

Tiago Alves up201603820@fe.up.pt

Index

Context

In this project we pretend to create a database for an application that manages rides so different users can share them. The database should be capable of storing and manage all the necessary information for the complete operation.

Our idea is to create a platform where a user with a car creates a trip with some specifications, like starting/ending time and initial/final address, and other users can join the trip, with the intention of splitting the final costs. The path chosen will be calculated by a map provider (google maps etc.) and all the users with cars will have to register its model so the trip price can be calculated using its average consumption and the trip distance.

Specifications

Any person using the *app* will be a **User**. The users will have a unique UID, username, password, rating, photo and email address. Users will be able to exchange many messages between themselves. They will be stored as **Message** with a date and text.

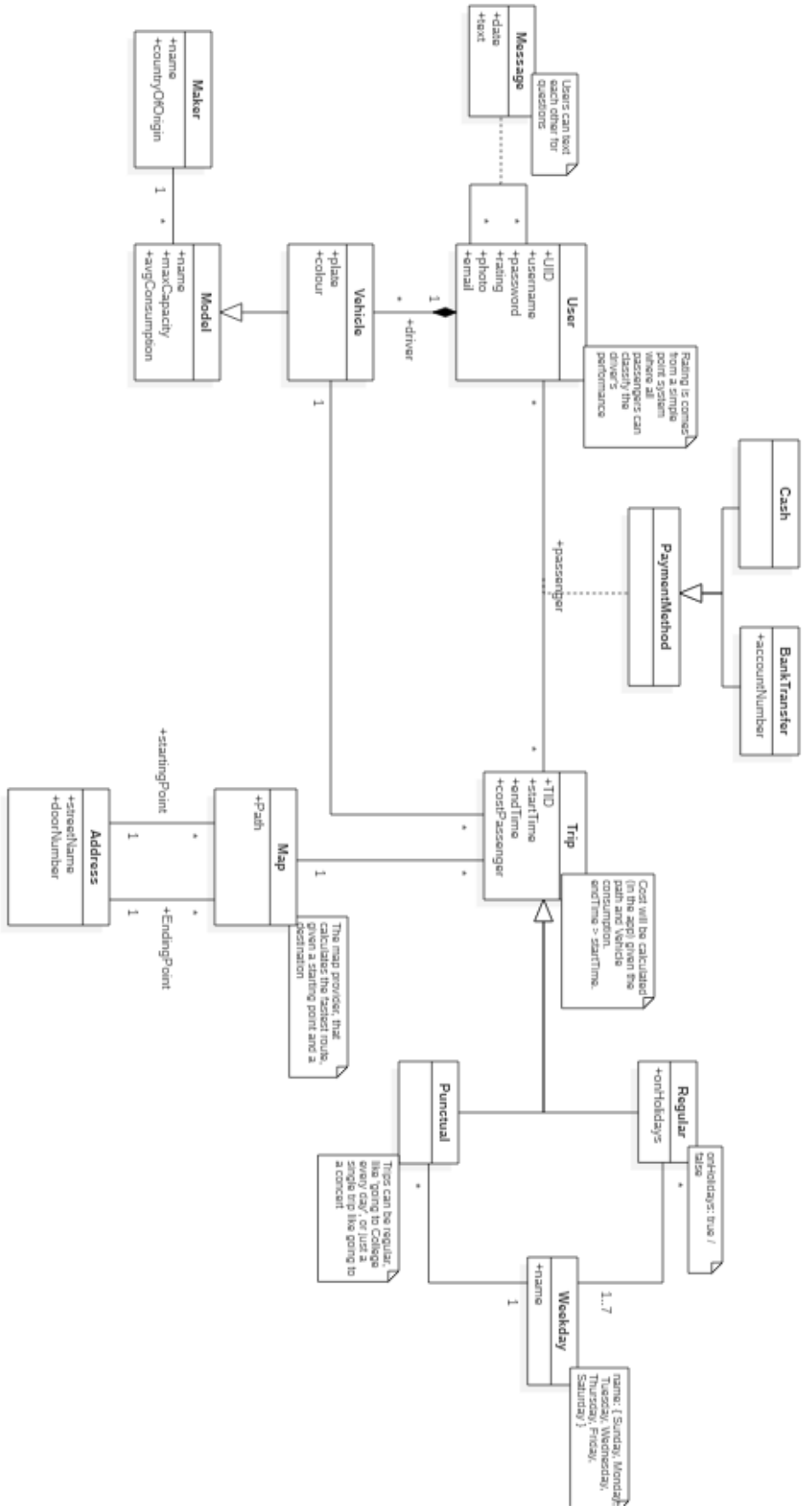
One of the Users will have a **Vehicle** which will be identified by their plate and color. Each Vehicle belongs to a **Model** that is composed by name, maximum capacity, average consumption and that belongs to a **Maker** with its name and country of origin. One user can own and drive as many vehicles as they wish.

The app also has a map provider that creates **Maps,** which creates a Path automatically using the *Order* of the **Addresses** that are associated (Like google maps). These Addresses, with streetName and doorNumber can be rearranged in different orders, creating different maps and paths depending of the **ListPassengers** that keeps record of the order of the stops on each map.

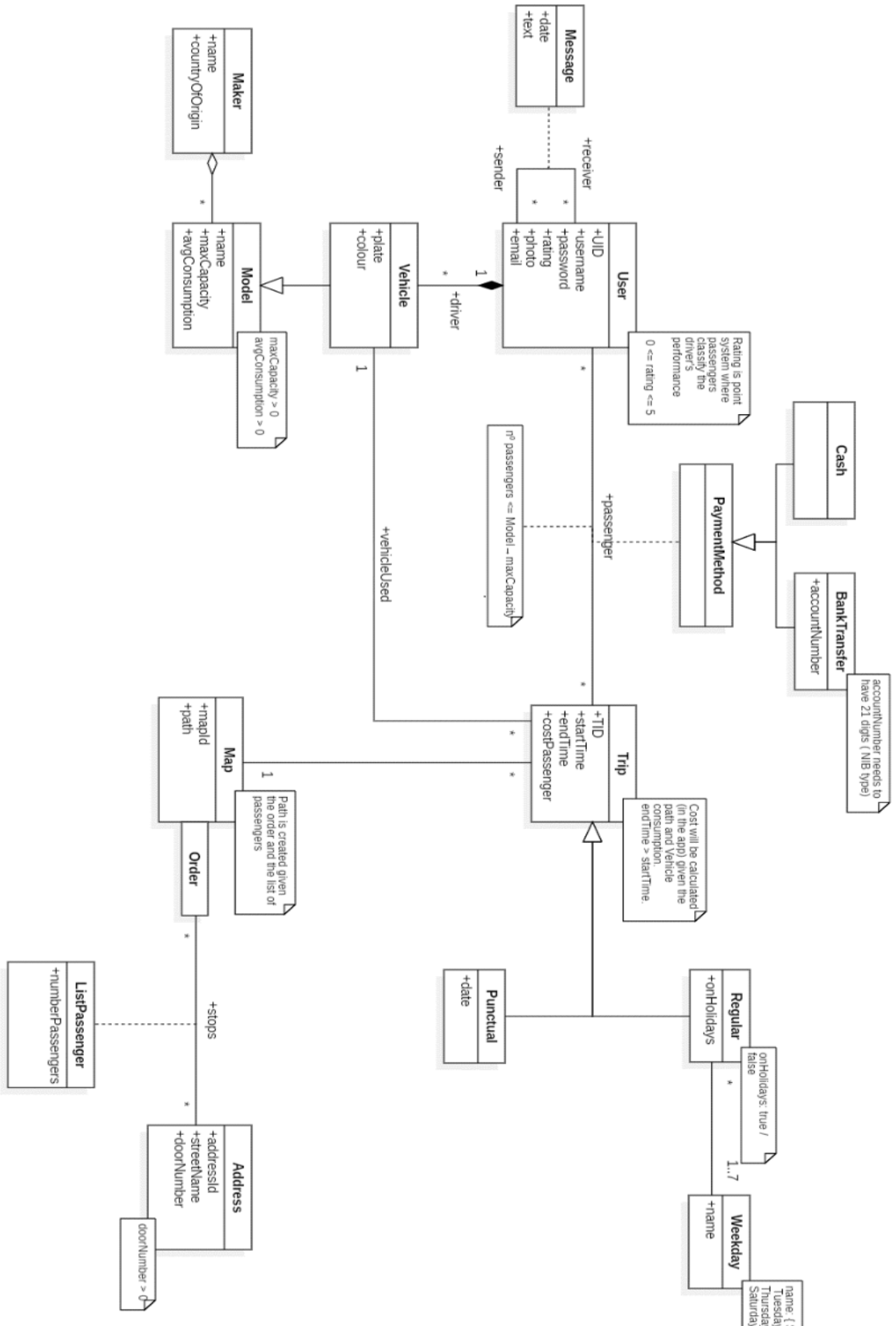
The vehicle, passenger (users) and the map are all connected to the **Trip** which will have an ID (TID), startTime, endTime, and cost per passenger (costPassenger) that will be calculated based on the path and number of passengers in each part of the trip. Each trip can only have a Map, a Vehicle, where the driver associated will be its driver, and it can have passengers as long as they don't surpass the Vehicle's maximum capacity. There will also be a **Payment Method** that will consist of either **Cash** or **Bank Transfer**. An account Number is needed if the User decides to opt for the Bank Transfer.

Finally, the Trip can be subdivided in two: The **Punctual** and **Regular** trip. The Regular trips are associated with a **Weekday**. The difference between them is that the Punctual must be associated with just one date and the regular will be associated with at least one day of the week. The regular trip also has an attribute called is onHolidays, which informs if there will be a "break" of the regular trip. The deleted tables like **DeletedPassengers**, **DeletedVehicles**, **DeletedTrips** and **DeletedMessages** will only be considered on the triggers as they are a copy of classes referred previously and will not be accessible to the users in the application.

Initial Conceptual Model



Reviewed Conceptual Model



Relational Model

User (uID, username, password, rating, photo, email)

- UID → username, password, rating, photo, email
- UID is the primary key
- rating is one *derived attribute*

Message (idMessage, date, text, sender →User, receiver →User)

- idMessage → date, text, sender, receiver
- idMessage is the primary key
- sender and receiver are *foreign keys*

Maker (name, countryOfOrigin)

- name is the *primary key*.
- name → countryOfOrigin.

Vehicle (plate, name →Model, color)

- plate is the primary key.
- plate → name, color.
- name is a *foreign key*

Model (name, maxCapacity, avgConsumption, maker →Maker)

- name is the *primary key*
- name → maxCapacity, avgConsumption, maker.
- maker is a *foreign key*

Map (mapId, path)

- mapId is the *primary key*
- mapId → path

Passenger (UID → User, TID → Trip, PID → PaymentMethod)

- UID, TID → PaymentMethod
- UID and TID are the *composite primary key*
- UID, TID and PaymentMethod are *foreign keys*

Address (addressId, streetName, doorNumber)

- addresId is the *primary key*
- addressId → streetName, doorNumber

Trip (TID, startTime, endTime, costPassenger, map → Map, plate → Vehicle)

- TID is the *primary key*
- TID→ startTime, endTime, costPassenger, idMapProvider, plate
- map and plate are *foreign keys*
- costPassenger is a *derived attribute*.

ListPassenger (LID, address → Address)

- LID is the *primary key*
- LID → address
- address is a *foreign key*

Stop (address → Address, map -> Map, listPassengers -→ ListPassengers, order)

- map and order are the *composite primary key.*
- Address, order → listPassengers, address
- Address, map and listPassengers are *foreign keys*

Cash (pid→ PaymentMethod)

- pid is the *primary key*
- pid is a *foreign key*

PaymentMethod (pid, user → User, trip → Trip)

- pid → user, trip
- pid is the *primary key*
- user and trip a *foreign key*

BankTransfer (pid →PaymentMethod, accountNumber)

- pid → accountNumber
- pid is the *primary key*
- pid is a *foreign key*

Regular (TID →Trip, onHolidays, weekday → Weekday)

- TID → onHolidays, Weekday
- TID is the *primary key*
- TID and weekday are *foreign keys*

Punctual (TID →Trip, date)

- TID → date
- TID is the *primary key*
- TID is a *foreign key*

Weekday (name)

- Name is the *primary key*

Functional Dependencies and Normal Form Analysis

In all the previous examples, it is possible to see that in each relation, the left side of the functional dependencies is a key for that relation. Therefore, the relational model is in the **Boyce-Codd Normal Form** and consequently, in the **3rd Normal Form**.

In the following paragraphs it is shown that the closure of the attributes in the left side is all the attributes in that relation:

User: $\{UID\}^+ = \{UID, username, password, rating, photo, e\text{-}mail\}$

Message: $\{idMessage\}^+ = \{idMessage, date, text, sender, receiver\}$

Vehicle: $\{plate\}^+ = \{plate, idModel, color\}$

Model: $\{name\}^+ = \{name, maxCapacity, avgConsumption, Maker\}$

Maker: $\{name\}^+ = \{name, countryOfOrigin\}$

Map: $\{MapId\}^+ = \{MapId, path\}$

Address: $\{AddressId\}^+ = \{AddressId, streetName, doorNumber\}$

Stop: $\{AddressId, MapId\}^+ = \{ AddressId, MapId, listPassenger, order\}$

Trip: $\{TID\}^+ = \{TID, startTime, endTime, costPassenger, idMapProvider, plate\}$

Passenger: $\{UID, TID\}^+ = \{UID, TID, paymentMethod\}$

PaymentMethod $\{\underline{PID}\}^+ \rightarrow \{ user, trip \}$

Cash: $\{PID\}^+ = \{PID\}$;

BankTransfer: $\{PID\}^+ = \{PID, accountNumber\}$

Regular: $\{TID\}^+ = \{TID, isOnHolidays\}$

Punctual: $\{TID\}^+ = \{TID, date\}$

Weekday: $\{dayNumber\}^+ = \{dayNumber\}$

ListPassenger $\{\underline{LID}\}^+ = \{LID, address\}$

Restrictions

	Restriction	Implementation
User	There cannot exist two users with the same UID	UID PRIMARY KEY
	There cannot exist two users with the same username and all users must have a username	username UNIQUE NOT NULL
	The rating must be between 1 and 5. Default value is 0	rating CHECK (rating >= 1 AND rating <= 5) DEFAULT (0.00)
	Users must have a password associated	password NOT NULL
	Users must have an email associated, the users cannot share emails.	Email UNIQUE NOT NULL
Message	There cannot exist two messages with the same messageId	messageID PRIMARY KEY
	The sender and receiver are foreign keys.	sender REFERENCES User(uID), receiver REFERENCES User(uID)
	Date and text can't be NULL.	date NOT NULL, text NOT NULL
Vehicle	There cannot exist two vehicles with the same plate, vehicles have a unique plate.	plate PRIMARY KEY
	driver is a foreign key	driver REFERENCES Model(name)

	Restriction	Implementation
Model	There cannot exist two models with the same name	name PRIMARY KEY
	maxCapacity must be greater than one	maxCapacity CHECK (maxCapacity >= 1) NOT NULL
	avgConsumption default value is 6.0. It cannot be a negative value	avgConsumption CHECK (avgConsumption >= 0) DEFAULT (6.00)
	maker is a foreign key. It cannot be NULL	maker REFERENCES Maker(name) NOT NULL
Maker	There cannot exist two makers with the same name	Name PRIMARY KEY
	countryOfOrigin cannot be null	countryOfOrigin NOT NULL
Map	There cannot exist two maps with the same mapId	mapId PRIMARY KEY
	All map must have a path	path NOT NULL
	All maps must have an order	order NOT NULL
Address	There cannot exist two addresses with the same addressID	addressId PRIMARY KEY
	All addresses must have a streetName	streetName NOT NULL

	Restriction	Implementation
Address	All addresses must have a doorNumber and must be greater than zero	doorNumber NOT NULL CHECK (doorNumber > 0)
Trip	There cannot exist two trips with the same TID	TID PRIMARY KEY
	mapId and plate are foreign keys	mapId REFERENCES Map(mapId), vehicle REFERENCES Vehicle(plate)
	endTime must be bigger than startTime and neither of them can be null	endTime CHECK (endTime > startTime) startTime NOT NULL endTime NOT NULL
	All trips must have a costPassenger associated and it must be greater than zero	costPassenger CHECK (costPassenger >= 0) NOT NULL
stop	There cannot be two orders with the same addressID and order	PRIMARY KEY (addressID, order
ListPassengers	addressId and orderId are the composite primary keys	orderId REFERENCES Order(orderId), addressId REFERENCES Address(addressId), PRIMARY KEY (orderId , addressId)
	numberPassengers cannot be NULL	numberPassenger NOT NULL
PaymentMethod	user and trip are the composite Primary Key	user REFERENCES User(UID), trip REFERENCES Trip(TID), PRIMARY KEY (user, trip)

	Restrictions	Implementation
Stop	address is a foreign key	address INTEGER REFERENCES Address (addressId)
	map is a foreign key	map INTEGER REFERENCES Map(mapID)
	Listpassenger is a foreign key	listpassenger INTEGER REFERENCES listPassenger (lId)
	All stops must have a order	Order INTEGER NOT NULL
	Address and order are the composite primary key	PRIMARY KEY (address , "order"));
Cash	Pid is a foreign key, it is the Primary Key	Pid REFERENCES PaymentMethod(PID) PRIMARY KEY NOT NULL
BankTransfer	Pid is a foreign key, it is the Primary Key	Pid REFERENCES PaymentMethod(PID) PRIMARY KEY NOT NULL
	account number must be 21 digits long. All BankTransfers must have an accountNumber associated	accountNumber CHECK (accountNumber >= 100000000000000000000 and accountNumber < 999999999999999999999) accountNumber NOT NULL
Regular	TID is a foreign key	TID REFERENCES Trip(TID)
	There cannot exist two regular trips with the same TID	TID PRIMARY KEY
	oHolidays has the default value of true	isOnHolidays DEFAULT(1)

	Restriction	Implementation
Punctual	There cannot exist two punctual trips with the same TID	TID PRIMARY KEY
	TID is a foreign key	TID REFERENCES Trip(TID)
	date cannot be null	date NOT NULL
Weekday	There cannot exist two weekdays with the same day	day PRIMARY KEY
	day can only be one valid day of the week	(CHECK name==’Monday’ or name==’Tuesday’ or name==’Wednesday’ or name==’Thursday’ or name==’Friday’ or name==’Saturday’ or name==’Sunday’)

Queries

The queries for this project were chosen based on usefulness, complexity and statistic usability. The top 10 queries are the ones described below.

Query 1: This query returns information that is useful to know how many people are using the application for statics purposes. It shows the total number of users using the application, how many trips were done, the total amount of money involved since the beginning of the app, the cheapest and most expensive trip, and finally, the average cost of all the trips realized.

Query 2: Shows how many cars each country has. This applies for all the countries present in the database and can be useful for marketing, to know which countries need more publicity and the ones who don't. It is also useful for statistical reasons.

Query 3: Obtains for each user all the money spent on the app, the amount of money it would have spent if the user didn't share the rides and the total profit. The table also includes the amount of trips each user has made and the data is ordered by the amount spent. This Query is a way to motivate the user to share more rides and consequently save more money.

Query 4: Obtain the application top 5 Users podium, ordered by rating in a descending order and for the tied ones, ordered by the number of trips the user has made. This is useful because it incentives users to share more rides to be in the podium and maybe to reward them.

Query 5: Gets all the Regular trips that include one address (Street Name and Door Number indicated in the query). It shows the TID so the user can easily identify the trip, the day of the week it occurs and finally both the begging time and ending time. This way the user can easily find which regular trips are the best option for a certain address considering this/her schedule. The current value for the address is "Rua das Flores" and "100".

Query 6: Obtain for a specific user, indicated in the Query, all the trips that he's been part of, with the respective cost and duration. In addition to these, each trip has associated the vehicle's plate and model along with the respective driver and mapID. This way the user can keep track of all the trips realized and can even contact the driver for any reason. It is also useful for statistical reasons.

Query 7: Shows in a descending order the most visited addresses that were visited since the beginning of the application. It returns the address ID and the corresponding street name, door number and how many times it was visited. Using this querie the clients know which are the places that most probably will have a trips available soon and for tourists for example, to know the most visited places in the city.

Query 8: Obtain all the vehicle makers present in the database, alongside it's average consumption so the user is more informed on which vehicle is cheaper in case there are two trips with the same stops. It is also useful for statics purposes.

Query 9: Get The vehicles and the respective drivers ordered by the time they have spent on trips. This query is useful not only to discover who is the driver of a certain vehicle, but for statistical reasons to know how much each driver has spent on the application. This can be used to reward the drivers depending on how much they used the app.

Query 10: Obtains all the drivers that have trips between two certain hours, so the user can talk to them and see if they can stop in the wanted destination. This is useful in case none of the trips available between those hours has the stop the user wants and it is necessary to talk to the drivers to see if any of them can add the destinations to its trip. The data is organized by the start time of the trips.

Note: In the above queries, when it is referred that something is already indicated in the query, it means that the query already has an example to demonstrate its purpoe, but can be modified by the user for different information.

Triggers

We also created 4 triggers which will help keep the database with correct and useful information. One of the files has two triggers.

Trigger 1: deleteUser

Before the delete command on a User has been executed this trigger will ensure that the data from the User and all data depending on it in the passenger, Vehicle and Message tables is registered into new tables for deleted values. Immediately after doing so, it will delete all this data from the original tables. This way all the unnecessary information that would stay in the database about the User who was deleted also gets deleted from the main tables but is still accessible from the deleted tables.

Trigger 2: validateEmailInsert and validateEmailUpdate

These two triggers will serve as a restriction for the input of email addresses making sure that at any point, whenever you insert a new User or update an old one's email, the email will have to be compliant with the standard format of an email address, otherwise, an error will be printed out.

Trigger 3: updateCostPassenger

This trigger will update the cost per passenger of any trip whenever a new passenger is added to that trip. To do this it calculates the total cost of the trip by by multiplying the number of passengers before the insertion by the initial cost per passenger. After that it divides the total cost by the new number of passengers in the trip which provides the new cost per passenger without any need for manual change.

To run these triggers, the script ‘createDB.sql’ should be read, followed by the ‘populate.sql’ scripts and the ‘.mode columns’, ‘.headers on’ and ‘.nullvalue NULL’ commands to format. After that, the triggers ‘gatilhoN_adiciona.sql should be read as well. Testing the triggers is done reading the ‘gatilhoN_verifica’ scripts and removing by using the ‘gatilhoN_remove.sql’ scripts.