La Salle Air
2020-2021

Databases: Second Semester Project

laSalle
UNIVERSITAT RAMON LLULL

TABLE OF CONTENTS

INTRODUCTION

Background

La Salle AIR (LSAIR) is system developed by La Salle for analysing how people is travelling around the world. LSAIR includes data from airports, flights, airplanes, people who is travelling, and people who works for making the travels possible such as pilots, flight attendees, mechanics, and shop keepers, among others.

LSAIR has been implemented as a MySQL database comprising more than forty tables highly interrelated. LSAIR can answer questions like how many people have travelled to a particular destination, which is the airport which has more flights, and which airplanes haven ‘t been maintained for the last month. However, the database requires some updates to answer complex queries and to figure-it out the relations between passengers.

A major update is to find relations between people, to do so, part of the information of the LSAIR database will be migrated to a graph database which excels in social network analysis.

Database students will take care of updating the database following the requirements described in the next sections of this document.

Project objectives

The goals of the project are the following:

- To update LSAIR relational database by implementing: new queries, tables, triggers and events.
- To design and implement a LSAIR graph-based database with data from LSAIR relational database.
- To provide documentation that describes the whole project: physical model updates, triggers/events, queries, graph model, and queries on graph-based database.

Document objectives

The goals of this document are:

- To list the requirements of the new LSAIR relational and LSAIR graph-based databases.
- To explain how to deliver the outputs of the project.

Considerations

LSAIR database is divided in four modules; therefore, it is recommended to carry out the project by a team of four students. Each student can lead one of the modules although communication between the team members is so important because the four modules are highly interconnected.

LSAIR RELATIONAL DATABASE REQUIREMENTS

The current LSAIR database has been implemented as a MySQL database. It contains four modules that are highly interrelated. The modules and their main tables are:

- Flight (Green): Flight, Pilot, Flight Attendee, Passenger, Flight Ticket, and Check-in.
- Airport (Red): Airport, Route, Airline, Plane, Maintenance and Airplane Pieces.
- Shopping (Blue): Company, Restaurant, Waiting Area, Store, and Product.
- Luggage (Yellow): Luggage, Hand Luggage, Checked Luggage Claim, Lost Object, and Refund.

Answer the requirements (queries, triggers, events, and stores procedures) of all four modules.

Requirement 1 Flights

To solve the requirements of this module, you should use only the tables that belongs to this module.

Requirement (Query) 1.1 Anticipating countries

We would like to know if it is true that buying tickets in advance allows you to get a better price. Show the average price of tickets bought by passengers from the most anticipating country and the average price of tickets bought by passengers from the least anticipating country. Only considers countries with more than 300 people according to the data in the database. The data to retrieve is:

Type of country *(most anticipating country or least anticipating country)*	country name	difference in hours *(between buying the ticket and the flight)*	Average price

Requirement (Query) 1.2 Passengers and turbulences

We want passengers who after a flight with 'Strong Turbulences' have not flown again. The data to retrieve is:

person id	name	surname	born date

Requirement (Query) 1.3 Pilots and co-pilots

We want to identify pilots who have been co-pilots more times than pilots as long as their grade is above two points above the average grade of all pilots. Return the list by the number of times they have been a pilot in descending order and then by their grade also in descending order. The data to retrieve is:

flying license	# times she/he was pilot	grade

Requirement (Query) 1.4 Very old passengers cannot communicate

We want to identify passengers who, of the languages they speak, none are spoken by their flight attendants. Consider only passengers who were born more than 100 years ago. The data to retrieve is:

passenger name	surname	born date

Requirement (Query) 1.5 Window passengers

We want to know which passengers who have ever travelled in business class, had a window seat (A or F) on all their flights. The passenger must have checked in at least twice. The data to retrieve is:

passenger name	surname

Requirement (Trigger) 1.6 Invalid tickets

We have been asked to carry out a security check on every flight ticket purchase. In this way, we have been asked to detect all those ticket purchases that are no longer valid (i.e. tickets for a flight that has already departed). To do this, we have to check when inserting data in the Flight tickets table and make sure that the date of purchase is not higher than the date of the purchased flight. Likewise:

- In case there is no problem, it must be inserted in the Flight tickets table.
- If there is a problem, delete it from the Flighttickets table and insert it in a table called TicketError (its columns will be: *personId, name, surname, flightId, dateOfFlight* and *dateOfTheTicketPurchase*).

Requirement (Trigger) 1.7 Credit card criminals

We want to control possible crimes due to credit card theft. Check the case when inserting a passenger, his credit card is already registered with another passenger. In case this happens, store in a *CrimeSuspect* table the following information of the possible criminal (passagerId, name, surname, passport, and phone).

Requirement (Trigger) 1.8 Cancelled flights

In case a flight is cancelled, emails will be sent to the affected persons in order to inform them about what happened. In this way, every time a flight is deleted from the database, we have been asked to insert in a table called *CancelledFlightsMails* the *flightId*, *personId*, *namePerson*, *emailPerson*, *priceOfTicket*, *isBusinessTicket*, and *commission*.

In addition, you need the cancellation cost table where you store the *flightId* and the *cost of cancelling* the flight taking into account that it will be refunded using the following formula:

$$refund = (price\_of\_the\_ticket)^2 * (commission + 1)$$

Note: The parameter *commission* in the table *CancelledFlightsMails* and in the refund formula, is the difference in days between the day of the flight and the day the flight is removed.

Note 2: Consider that each passenger will have a maximum of one ticket.

Requirement (Event) 1. 9 Flight statistics

Create a table called "daily flights" where each day you will insert how many flights will depart on that particular day. This table will only have as columns the date and the number of flights departing on that day. In addition, create a table called "monthly flights" where the average number of daily flights will be stored for each month.

Requirement 2 Airports

To solve the requirements of this module, you should use only the tables that belongs to this module.

Requirement (Query) 2.1 Airlines and petrol capacity

We want to find the airlines and the number of routes they operate where the minimum petrol to fly the route is greater than the capacity of the aircraft assigned to the route. We only need to consider routes between airports in different countries. The data to retrieve is:

airline name	# routes

Requirement (Query) 2.2 Mechanic grades

We want to make a study to relate the grade a mechanic has and the duration of the maintenance s/he usually does. Therefore, list the ranges of marks (i.e., 0-1, 1-2... 9-10) and the average duration of the maintenance carried out by the mechanics who fall into each of the ranges. Only maintenance where less than 10 pieces have been replaced should be taken into account. The data to retrieve is:

grade range	duration average

Requirement (Query) 2.3 Airports and mean distance routes

We want to know which airports have the average distance of routes departing from them greater than the average distance of routes departing from airports in the same country. The data to retrieve is:

airport id	country id	average distance

Requirement (Query) 2.4 Airlines and routes

We want to find active airlines that do not have any routes departing from or entering your home country sorted by longest route duration. The data to retrieve is:

airline name	airline id	country name	longest route duration

Requirement (Query) 2.5 Pieces replaced

We want to list planes that have had to replace the same piece more than once, and that the cost of changing those pieces was more than half of the cost of all other parts changed on those planes. The data to retrieve is:

plane id	piece name *(the ones that have been replaced more than 1 time)*	# pieces replaced

Requirement (Trigger) 2.6 Routes cancelled

As we all know, this last year has been hard, not only on a psychological level, but also on an economic level, many companies have been affected by the measures applied by governments. For example, airlines are constantly affected by the closing of borders in some countries, which causes them to cancel routes between specific countries. To consider all these facts we have been asked to store in a table called *RoutesCancelled* (which will contain the name of the destination, the name of the origin, the number of airlines that managed this map and the date on which the route was deleted) the routes that are being cancelled. In addition, when a route is deleted, we have also been asked to delete the route from the *RouteAirline* table.

Requirement (Trigger) 2.7 Mechanics firings

From the mechanics unions, we have been asked to have a history of the reason for each of the firings of mechanics that have the different airports. The reasons for the dismissals will be stored in the table *MechanicsFirings* (id of the mechanic, name, surname, date of birth of the person and reason for firing). You have to take into account that there can be 3 types of reasons for dismissal:

- Retirement: If the date when the person is deleted from the table is 65 years old or older.
- Not completing the evaluation period: When the sum of repairs that the mechanic has done does not add up to more than 10 hours.
- Firing without reason: When it does not belong to any of the previous types.

You have also to remove the maintenance and the pieces (*PieceMaintenance* table) replaced by them.

Requirement (Trigger) 2.8 Petrol updates

Every year, technological improvements are made in planes, which makes us a more and more environmentally sustainable society. For this reason, we want to have a history of the times that the amount of petrol is updated to carry out a route. Create a historical table called *EnvironmentalReductions* in which the route is stored (VARCHAR where the origin + destination are included in a single value), the difference with respect to the minimum value prior to the update and the date on which the update occurred.

Requirement (Event) 2.9 Yearly maintenance costs

We know that the cost of maintaining a plane is high, which is why the airlines, motivated by all the economic losses they have suffered during the pandemic, want to carry out a study on those planes that cost them the most to maintain. In order to carry out this study, they have asked us to write down each year in a table called *MaintenanceCost*, which will have

the name of the plane and the economic costs of maintenance that it has meant during the year.

Note: To find out how much it costs to maintain a plane you only have to look at the price of the pieces used for the maintenance of the plane. It must be considered that there may be some cases where the maintenance cost may be 0.

Requirement 3 Shopping

To solve the requirements of this module, you should use only the tables that belongs to this module.

Requirement (Query) 3.1 Local commerce

We want to know which companies are committed to local trade. That is to say, which have at least 1 out of 40 food products coming from the same country as the company's country. The data to retrieve is:

company name	country name

Requirement (Query) 3.2 Best trade relations

We want to know which countries have the best trade relations with each other. Considering that VIP rooms can have an associated restaurant. List pairs of countries (one from the companies that have the VIP rooms and the other from the companies that have the restaurants) as long as they are different. To ensure that trade relations can be good, the two countries (VIP and restaurant) must not have more than 80 prohibited products each one of them. The data to retrieve is:

vip rooms' country	restaurants' country	#vip rooms/restaurant

Requirement (Query) 3.3 The best restaurant

From the company that has the most waiting areas, we want to know which of its restaurants has the best score. The data to retrieve is:

company name	score

Requirement (Query) 3.4 Stores and restaurants

We want to identify companies (name and value) that have a) shops that sell at least 20% of the total products of the same company, and b) restaurants of at least two different types. The data to retrieve is:

company name	company value

Requirement (Query) 3.5 Waiting areas without shop keepers

We want to find out which waiting areas do not have enough shop keepers to cover the hours they are open. The areas are open every day of the week and only those areas that close before midnight are taken into account. The data to retrieve is:

company name	opening hour	close hour	airport ID	waiting area ID

Requirement (Trigger) 3.6 Waiting areas shutdown

Many companies, due to the current economic situation, are being forced to consider very tough strategies due to the reduction in profits they have suffered. That is why this time we have been asked every time a waiting area is eliminated, to introduce in a table called *EconomicReductions* a) the name of the company, b) the name of the closed waiting area, c) the annual savings that the company will have when closing the waiting area, and d) the annual expenses that the company will have without taking into account the waiting area that has now been closed. Please note that both the expenses and the savings are calculated only on the basis of the salary of the employees, without taking into account commissions.

Note: The salary of an employee is 10 € per hour and a year has 52 weeks.

Finally, in case the company has no more expenses, delete it from the database. Remember that in case of deleting a *waitingArea* or a company you must delete everything related to it, otherwise MySql will not let you perform the deletion operation.

Requirement (Trigger) 3.7 Updated products

On this occasion, we want to take into account the strategies followed by the different stores. That is why this time we have been asked to insert in a table called *PriceUpdates*, every time the price of a product is updated, the following information: name of the product, company in charge of the product, previous price, later price of the product, date of the change and a comment.

In case the product is already in the table of price updates on another date and its value has decreased, enter the following comment: "*This product has been changing over time, it is possible that it is a strategy of the company*".

Requirement (Trigger) 3.8 Product values per m2

Generate a table named *AverageSquareMetreValue* (with attributes *storeId* and *valueM2*) that has the average price of the products per square metre of the different shops found in the airports. This table will be updated each time a product is inserted in a shop. Check in each insertion that all the shops are represented in this table, if any shop is not represented enter its value per m2, even if it is 0.

Requirement (Event) 3.9 Expired food

We want to make a daily control of those food products (food) that are already expired so that the shop assistants can immediately remove them from sale. In this way, create a table called *ExpiredProducts* which includes the product id, the expiry date, and the day on which this warning occurs. In this table, all those products that are out of date and are still on sale will be inserted.

Requirement 4 Luggage

To solve the requirements of this module, you should use only the tables that belongs to this module.

Requirement (Query) 4.1 Luggage handlers' rewards

We want to reward the heavy workers, in particular the luggage handlers who have loaded a plane by themselves and are paid less than the average wage of the luggage handlers. The data to retrieve is:

person id	name	surname	salary

Requirement (Query) 4.2 Luggage details

We want to retrieve some luggage details. Select the color, brand, weight, volume, extra cost, and fragility of the luggage that belong to those passengers whose first four letters of their name are the same as the first four letters of their country. If they have more than one luggage, give the details of the one that have less weight. If they do not have luggage, return the name, email, and country of the passenger. The data to retrieve is:

passenger name	email	country	color	brand	weight	volume	extra cost	fragility

Requirement (Query) 4.3 Lost objects

We want to make a psychological study of the influence of the colour or brand of luggage. Specifically, we want to know the number of lost objects according to their colour and brand. Therefore, we have to make a list of colours and brands in which for each one of them we indicate how many objects have been lost and how many passengers have lost them. If a passenger has lost more than one different piece of luggage, it will have to be included in all possible groups. The list should be sorted according to the ratio of lost items relative to the number of passengers. The data to retrieve is:

type (color or brand)	# lost objects	# passengers	ratio (lost object/passengers)

Requirement (Query) 4.4 Special objects

We want to list special objects according to the number (i.e., 0, 1, 2, or 3) of hazardous characteristics (i.e., fragility, corrosive, flammable) they have. For each item in the list, indicate their average extra cost. The data to retrieve is:

hazardous level	average extra cost

Requirement (Query) 4.5 Claims

We want to find passengers who meet these two criteria: 1) they have made more claims than they have travelled, and 2) they have never had a refund accepted. The data to retrieve is:

person ID	name	surname	# claims	# flights

Requirement (Trigger) 4.6 Refunded tickets

We want to prevent clever people from trying to find vulnerabilities in the database. Therefore, we want that for each refund that is made, we check if there is already another refund accepted for the same ticket. In case there is, we will insert in the table *RefundsAlterations* the Id of the person, the ticket id and the comment "*Refund of a ticket already processed correctly*".

We also want to control the people who make excessive attempts to refund the same ticket. In case the refund of a ticket is rejected 3 times or more, the comment "*Excessive Attempts*" will be entered in the above table.

Requirement (Trigger) 4.7 Lost object days

It controls each time a lost object goes from not found to found, in this case, it introduces in the table *LostObjectsDays* the id of the lost object, how many days it has taken to find it, and what is the average number of days it takes to find objects of the same type as the lost object. For the calculation of the average, you have to take into account only the objects that are in the table *LostObjectsDays*.

Note: Two objects are of the same type if they have the same description.

Requirement (Event) 4.8 Transported statistics

Calculate for each day, month, and year how many kilograms have been transported, how many dangerous goods, and how many favourable claim returns have been produced. Therefore, you will need three tables:

- *DailyLuggageStatistics*: date, number of kilograms of the transported objects, number of danger objects, and accepted returned claims.
- *MonthlyLuggageStatistics*: year, month, number of kilograms of the transported objects, number of danger objects, and accepted returned claims.
- *YearlyLuggageStatistics*: year, number of kilograms of the transported objects, number of danger objects, and accepted returned claims.

Requirement 5 Cross-module (optional)

To solve cross-module queries you can use any the tables of the database.

Requirement (Query) 5.1 Overbooked airlines

We want to know which airlines have had the most overbooked flights, that is, at least one passenger who has bought a ticket but has not checked in. Consider only airlines that have had this problem with more than 10% of their passengers. The data to retrieve is:

airline name	# overbooked flights

Requirement (Query) 5.2 Lost objects due to jet lag

We want to know how jet lag affects passengers, especially when it comes to losing their luggage. We want to know the number of lost items according to the time difference between the origin and destination of the flights. We will assume that the lost property belongs to a flight if the claim has been made within three months after the flight. The data to retrieve is:

time zone difference in hours *(i.e., 0, 1, 2… 23h)*	# lost objects

Requirement (Query) 5.3 Flight attendant vacancy

The airline Ryanair has a vacancy for a flight attendant and wants to search in the passenger database. Specifically, it is looking for passengers who speak at least two languages (one of them being "*Chavacano*") and who have made a flight where the time difference between origin and destination is 3 hours or more. The data to retrieve is:

passenger name	surname	Phone number	# languages spoken

Requirement (Query) 5.4 Smuggling airports

We want to know at which airports smuggling is taking place. This means that there are passengers who embark products that are prohibited in the country of their destination. List the airports and the total number of passengers passing through the airport. The data to retrieve is:

Airport name	total number of passengers

Requirement (Query) 5.5 Reliable plane types

We are looking for the most reliable types of planes. We want to list plane types that at least 53% of their flights have a perfect status, have made more than 500 flights, the distance flown is more than 1.000.000 km, and more than 70 airlines have used them. The data to retrieve is:

type name

Requirement (Query) 5.6 Employees languages

We want to know which languages are spoken most by flight attendants and other employees. For each type of employee (flight attendants and other employees), list the languages they speak and how many people speak them. The data to retrieve is:

employee type *(flight attendee or other employee)*	language	# people who speaks the language

LSAIR GRAPH DATABASE REQUIREMENTS

For carrying out social network analyses, LSAIR database has to be migrated to a graph database such as Neo4j. To validate the feasibility of moving all relational data to a graph-based data model, you have to develop 2 case studies.

For each of them, extract data from the relational database and import it to Neo4J. Before importing the data, it is interesting to think about the data model you are going to need: which nodes, properties, and relations.

Case Study 1: Airplane, airport, city, and country

The goal of this case study is to relations between airplanes, airports and countries based on the routes and flights. We want to migrate from the relational database the following:

- **Dataset 1**: Retired **planes** on the last 3 years that belongs to airlines from countries whose name starts by S. For each plane, we need the *plane id*, *retirement year*, *plane type name*, *airline name*, *number of times that has been maintained*, *number of different pieces that have been replaced*, and *the total cost of the replaced pieces*. Notice that there may be planes that have not been maintained or have not had parts replaced.

- **Dataset 2**: Considering only the planes selected in the previous point, we want to select the **airports** that those planes have a route. For the airports, we want to know the *airport id*, the *name*, *altitude*, *city id*, *city name*, *city time zone*, and *country name*.

- For connecting both datasets, we want to know if a **plane** of the first dataset has as *destination* or as *departure* an **airport** of the second dataset.

Data model

Once you have data selected in the relational database, you can store it in CSV files that you can import in Neo4J using "LOAD CSV WITH HEADERS FROM…" command.

The data model for this case study should consider only the following entities (nodes): **plane**, **airport**, **city** and **country**. You have to decide which properties will have each entity and how are they related.

Queries

Once you have the data imported into Neo4J, you can answer the following queries.

1. Select the planes that has not flown to any airport and has less than 8 pieces replacements.
2. Make a list planes and the number of airports that have been. Order the list by the number of airports in a descendant order.

3. Make a list of planes and the number of countries that have been if they have been in an airport with an altitude value more than 100.
4. Display the shortest route between “Greece” and “Singapore”.
5. Display the routes between “Laos” and “South Korea”.

Case Study 2: Pilots, flight attendee, language, flight and airport

The goal of this case study is to relations between flight crew based on language they speak and airports they have been. We want to migrate from the relational database the following:

- **Dataset 1**: Non retired **pilots** that speak more than 3 different **languages** and their salary is over 100.000. For each pilot, we want to know the *pilot id*, *name*, *surname*, *email*, *sex*, *salary*, *years working* and *the languages she/he speaks*.
- **Dataset 2**: Flights attendants who have been on the same flight as the pilots. Consider only the pilots of the first dataset. For the flight attendants, we need their *id*, *name*, *surname*, *email*, *sex*, *salary*, *years working* and *the languages she/he speaks.*
- For connecting both datasets, we want to know the **flights** that pilots and flight attendants have been. For each flight we want to know the *id*, *date*, *destination airport*, and *departure airport*.

Data model

Once you have data selected in the relational database, you can store it in CSV files that you can import in Neo4J using “LOAD CSV WITH HEADERS FROM…” command.

The data model for this case study should consider only the following entities (nodes): **pilot**, **flight attendee**, **language**, **flight** and **airport**. You have to decide which properties will have each entity and how are they related.

Queries

Once you have the data imported into Ne4J, you can answer the following queries.

1. We want to discover new relations between crew mates. In the case that two different flight attendees have been in the same flight, they can be considered colleagues. Therefore, create “colleague” relation between those flight attendees. Thus, display all the colleagues.
2. Suppose two flight attendees have been in the same airport (regardless the date), they speak the same language, and are not colleagues, they will become acquaintances. Therefore, create “acquaintance” relation between those flight attendees. Thus, display all the acquaintances.
3. It is well known that pilots and assistants have adventures. Create “affair” relation between pilots and flight attendees that have been in the same flight, speak the same

language, and the difference in the number of years worked is less than 10. Thus, display all the affaires.

4. Display a list of the languages and the number of affaires due to that language. Order the list by the number of affaires in a descendant order.
5. Select the pilots that have affaires with two or more than two flight attendees that are colleagues among them.
6. Show the relations between pilots that have more than one affaire. Consider only relations such as affair, colleague, and acquaintance.

DELIVERY INSTRUCTIONS

The project is meant to be developed by a team of four members. There are around 10 sessions at the end of the semester in which professors will mentor and guide the work carried by the team.

The project documentation should be submitted **before the May 30th, 2021 at 23:59.**

The name of the file submitted will follow this pattern: **2021P2Group#.zip** where **#** is the number of the group that will be assigned by the professors. The zip will contain these folders:

- **Report**. This folder will include the report written by the team following the given template.
- **SQL scripts**. This folder will include the scripts for the requested queries, triggers, events.
- **Neo4J scripts**. This folder will include the scripts for generating CSV files from the relational database (.sql) and scripts for the requested queries in Cypher (.txt).

Considerations

- **If there is a missing file, the submission will not be accepted.**
- **If the report template is not strictly followed, the submission will not be accepted.**
- **If a copy is detected, all the members of the implied teams will fail the semester, those who have copied as those who have been copied. Moreover, they will have to check with the university normative to take into consideration other consequences.**