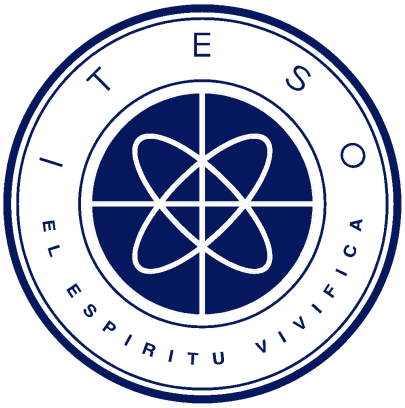
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

BASES DE DATOS NO RELACIONALES



PROYECTO INTEGRADOR BASES DE DATOS NO RELACIONALES

Presentan

Miguel Alberto Torres Dueñas 739674
Yael Alejandro Rodriguez Barreto

Profesor: **LEOBARDO RUIZ ROUNTREE**

Fecha: **25/11/2023**

Contenido

Nombre del proyecto

Aero Calafia Airlines

Introducción

El objetivo de este proyecto es atender algunas problemáticas que enfrentan las empresas relacionadas a los aeropuertos, mediante el diseño de distintas bases de datos que nos ayudarán a resolver estas oportunidades de negocio. Dichas problemáticas son obtener el mejor mes para hacer una campaña de renta de carros para una empresa aeroportuaria, obtener en base a los datos los aeropuertos en los que implementar más servicios de alimentos y bebidas, y por último un modelo que ayude a conocer a las aerolíneas sus meses de mayor demanda, de cara a que estas puedan generar más ofertas convenientes para los vuelos y su compañía. Para estos modelos, utilizaremos tanto Python como JavaScript para manejar las distintas bases de datos no relacionales que utilizaremos para cada modelo, siendo dichas bases Cassandra, Dgraph y MongoDB. Además, proporcionaremos un menú para el cliente en los cuales pueda interactuar y consultar la información, donde dichas consultas proporcionarán la información necesaria para que los clientes se encarguen de realizar lo necesario para resolver las problemáticas.

Alcance del Proyecto

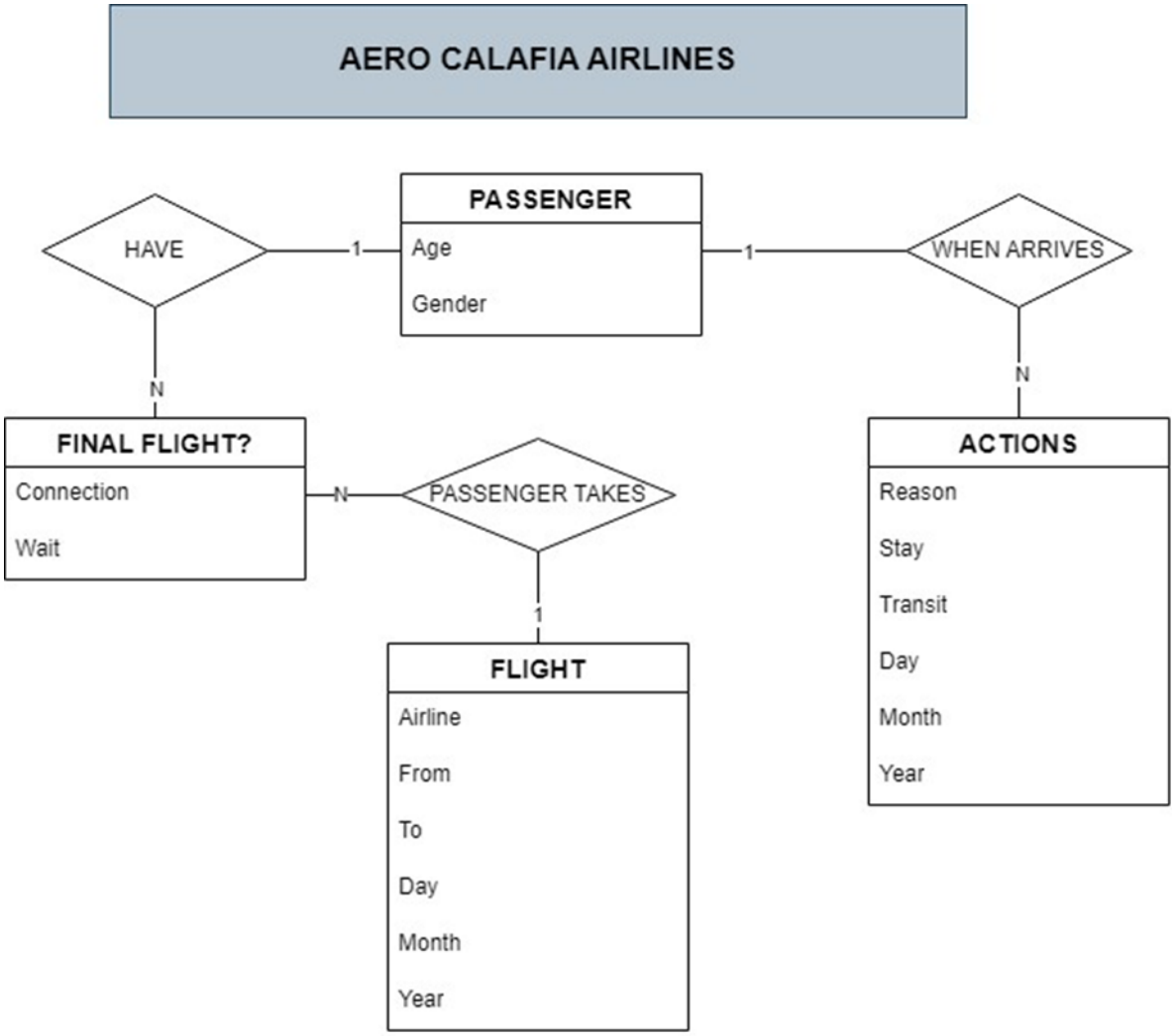
En nuestro proyecto, se incluye el manejo de 3 distintas bases de datos no relacionales, donde cada una se ejecutará desde la terminal (no incluye una interfaz gráfica), en donde los modelos para los problemas de el mejor mes para la renta de carros y las épocas del año para hacer descuentos cuentan con un menú en el que el cliente puede interactuar entre varias opciones, mientras que el modelo del problema de implementación de servicios y bebidas no cuenta con uno, por lo que se mostrarán los datos de manera automática.

Requerimientos

El sistema deberá ser capaz de ejecutar consultas de datos masivos de forma rápida y eficiente a través de los distintos sistemas de gestión de bases de datos mencionados anteriormente; esto en Python y JavaScript (cada modelo es en uno de los dos, no en

ambos). Dichas consultas mostrarán la información necesaria para los clientes con la cual podrán resolver la problemática planteada.

Diseño de la solución

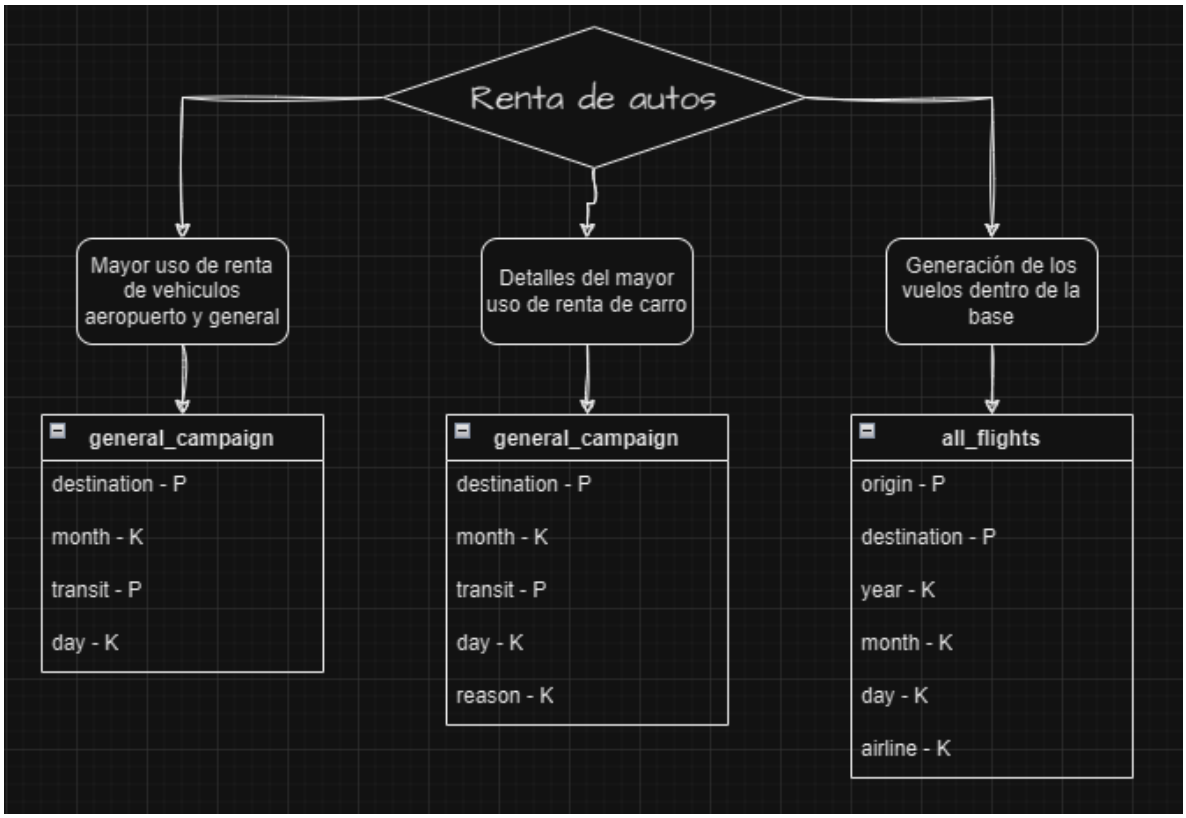


Cassandra

Modelos de datos utilizados

Para resolver el problema que sugiera cuáles son los meses que es recomendable introducir campaña de publicidad para empresas de renta de carros para cada aeropuerto utilizamos Cassandra. Los datos que decidimos utilizar para resolver la problemática fueron “to” (Airport Destination), “month” (del flight) y transit. En el caso de to fue debido a que los servicios de renta de carros se utilizan una vez aterrizas en dicho aeropuerto, en month fue

para poder ver por mes la frecuencia en la que esto se repitey en transit fue para ver las veces en las que los pasajeros utilizaron como transporte “Car rental”. Además, dependiendo de la consulta se utilizaban o mostraban de diferente manera, tal y como mostraremos a continuación.

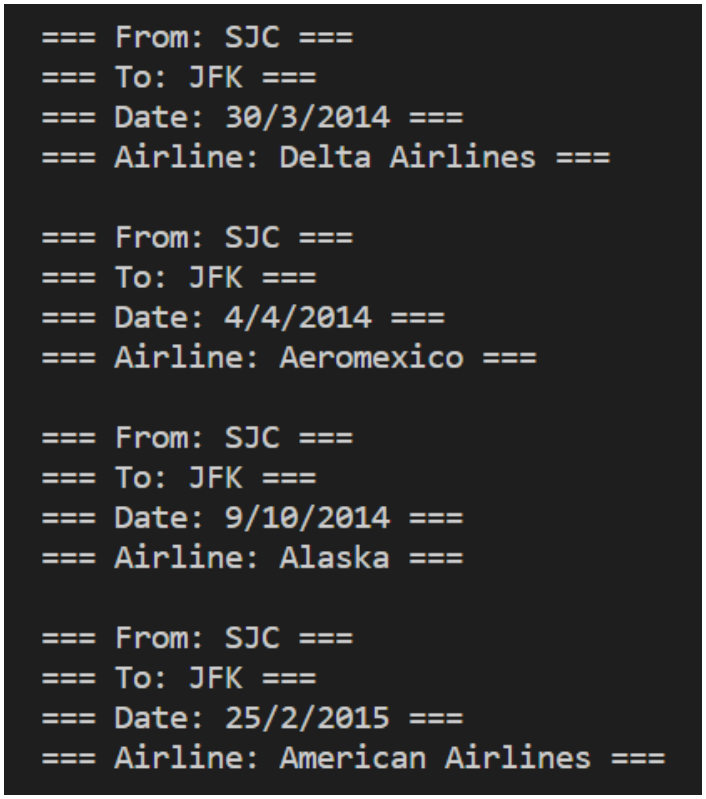


Consultas

Las diferentes consultas que el usuario puede hacer desde el menú del cliente son las siguientes:

```
1 -- Show All Flights
2 -- Show data related to the Airport
3 -- Show data related to the Airport (more details)
4 -- Summary of months by Airport
5 -- Exit
Enter your choice:
```

1. Show All Flights: Esta consulta muestra todos los vuelos registrados, mediante la obtención de los datos from, to, day, month, year y airline. Es importante recalcar que esta consulta es un extra, no resuelve la problemática.



2. Show data related to the airport: Esta consulta requiere que el cliente ingrese el aeropuerto a buscar, para después mostrar todos los vuelos en los que el aeropuerto está involucrado como destino, junto al mes en el que se realizó. Esta consulta muestra todo sin agrupar.

```
=== Airport: LAX ===
=== Month: 4 ===

=== Airport: LAX ===
=== Month: 4 ===

=== Airport: LAX ===
=== Month: 5 ===

=== Airport: LAX ===
=== Month: 5 ===

=== Airport: LAX ===
=== Month: 6 ===

=== Airport: LAX ===
=== Month: 6 ===
```

3. Show data related to the airport (more details): Esta consulta hace lo mismo que la consulta anterior, solo que esta vez agrega los motivos por los que el pasajero realizó el viaje (esto utilizando el dato reason). Esta consulta fue hecha por si el cliente considera dicha información útil de cara a resolver la problemática.

```
=== Airport: LAX ===
=== Month: 12 ===
=== Reason: Business/Work ===

=== Airport: LAX ===
=== Month: 12 ===
=== Reason: Business/Work ===

=== Airport: LAX ===
=== Month: 1 ===
=== Reason: On vacation/Pleasure ===

=== Airport: LAX ===
=== Month: 3 ===
=== Reason: On vacation/Pleasure ===

=== Airport: LAX ===
=== Month: 4 ===
=== Reason: On vacation/Pleasure ===

=== Airport: LAX ===
=== Month: 7 ===
=== Reason: On vacation/Pleasure ===
```

4. Summary of months by airport: Esta consulta es la que resuelve por completo la problemática, ya que esta le pide al cliente que ingrese el aeropuerto, para así mostrar la frecuencia de viajes en las que el aeropuerto es destino pero por mes.

```
=== Airport: LAX ===
=== Month: 6 ===
=== Count: 4 ===

=== Airport: LAX ===
=== Month: 7 ===
=== Count: 5 ===

=== Airport: LAX ===
=== Month: 8 ===
=== Count: 6 ===

=== Airport: LAX ===
=== Month: 9 ===
=== Count: 5 ===

=== Airport: LAX ===
=== Month: 10 ===
=== Count: 6 ===

=== Airport: LAX ===
=== Month: 11 ===
=== Count: 8 ===
```

Optimizaciones

En este caso la mejor forma de optimizar el código es quitarle el rango de clustering a todos lo campos sobre los cuales no sería necesario trabajar de forma directa, además de usar únicamente aquellos que fueran necesarios en la consulta, esto disminuiría no solo el espacio necesario para el guardado de información, si no que también agilizaría el proceso de su búsqueda.

Dgraph

Modelos de datos utilizados

Para resolver el problema que nos permita conocer las épocas de mayor demanda de vuelos por aerolínea para la generación de ofertas y descuentos para clientes utilizamos Dgraph. Los datos que decidimos utilizar para resolver la problemática fueron “airline” y “month” (del flight). En el caso de airline fue debido a que nos interesa saber la frecuencia por aerolínea de dicha información, y en el caso de month fue para poder ver por mes la frecuencia de los vuelos con dicha aerolínea.

Consultas

Las diferentes consultas que el usuario puede hacer desde el menú del cliente son las siguientes:

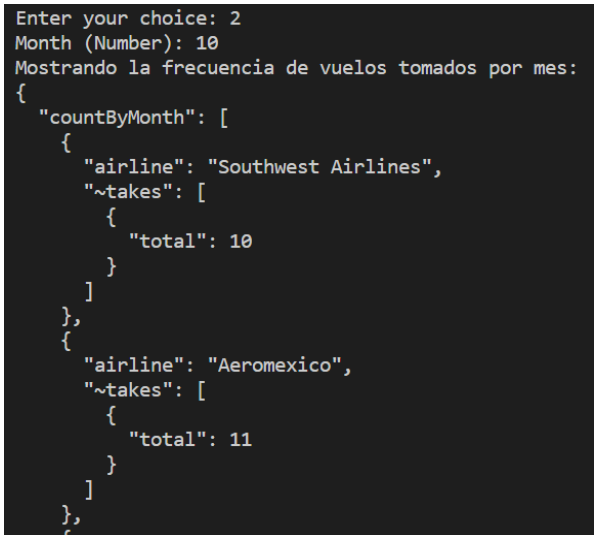
```
1 -- Create data
2 -- Count Airline by Months
3 -- Count Months by Airline
4 -- Drop All
5 -- Exit
Enter your choice:
```

1. Create Data: Esta opción es necesaria para poder ejecutar las demás consultas, ya que esta crea los datos en la base de Dgraph (hace una mutación de todos los datos necesarios).

```
Enter your choice: 1
Commit Response: start_ts: 130005
commit_ts: 130006

UIDs: {'53male': '0x1395e', '86male': '0x13a32', 'MagniCharters10': '0x138e7', 'Alaska6': '0x138ee', '45male': '0x138b8', '67female': '0x13
a05', '62female': '0x139e1', '24unspecified': '0x138a0', '18unspecified': '0x13a14', '84unspecified': '0x1395f', '17male': '0x139ce', '4fem
ale': '0x1395a', 'American Airlines5': '0x13935', '60undisclosed': '0x13975', 'Volaris11': '0x139d2', '4unspecified': '0x13911', '42undiscl
osed': '0x139e4', 'Viva Aerobus3': '0x13933', 'Volaris6': '0x13920', 'American Airlines3': '0x139f0', '16female': '0x13a28', '25female': '0
x13926', 'Viva Aerobus1': '0x13988', '1female': '0x138f2', '57unspecified': '0x1388a', '77male': '0x1399a', 'Alaska12': '0x139c8', 'Southwe
st Airlines2': '0x13a2d', 'MagniCharters1': '0x138cc', '53unspecified': '0x138e4', '5undisclosed': '0x139c6', 'Aeromexico7': '0x1396d', 'Vo
laris9': '0x138e2', '1male': '0x13956', '18female': '0x139bf', '35unspecified': '0x139c4', '51unspecified': '0x13918', '84female': '0x13a0d
', '32male': '0x138ed', '35undisclosed': '0x13a01', '41male': '0x138ba', '72female': '0x138db', '66unspecified': '0x139fe', '71male': '0x13
9d8', '8undisclosed': '0x13892', 'Delta Airlines12': '0x138f3', '77female': '0x139d0', '64undisclosed': '0x138bc', '12female': '0x139b8', '
79unspecified': '0x13999', 'Southwest Airlines12': '0x138d8', 'American Airlines7': '0x1392f', '83female': '0x138a2', 'Volaris5': '0x13915'
, 'Alaska3': '0x13957', '46unspecified': '0x13934', '52unspecified': '0x139fc', '14male': '0x13930', '60unspecified': '0x13902', '29female'
: '0x13a2f', '87female': '0x138c8', '4undisclosed': '0x139bc', '27unspecified': '0x13950', '88unspecified': '0x1388d', '84male': '0x1391a',
 'Viva Aerobus8': '0x138c9', '87male': '0x13990', 'Alaska11': '0x139e7', '34female': '0x139a0', '67male': '0x139ad', 'Southwest Airlines7':
```

2. Count Airline by Months: Esta consulta requiere que el usuario ingrese el mes del cual quiere ver la frecuencia de vuelos para cada aerolínea. Donde se muestra la aerolínea y luego la cantidad de vuelos hechos durante ese mes.



3. Count Months by Airline: Esta consulta requiere que el cliente ingrese la aerolínea por la cual hace la búsqueda, en donde se mostrará para dicha aerolínea la frecuencia con la que viajan con esa aerolínea en cada uno de los meses.

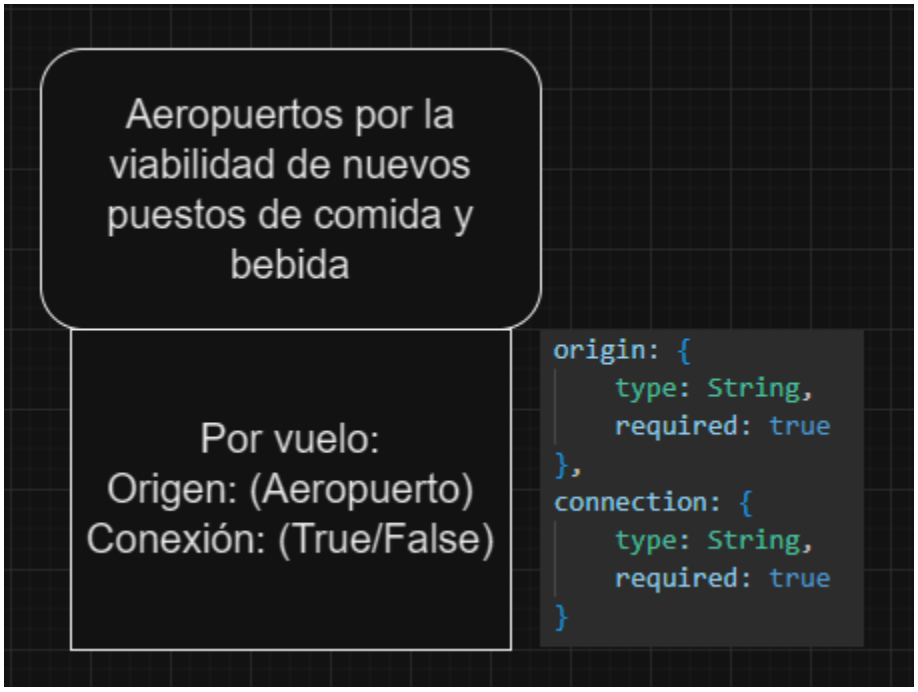
Índices

En este caso se generaron índices en airline y month para mejorar la eficiencia en la búsqueda y obtención de los datos, en donde justamente estos dos datos fueron los utilizados durante las consultas.

MongoDB

Modelos de datos utilizados

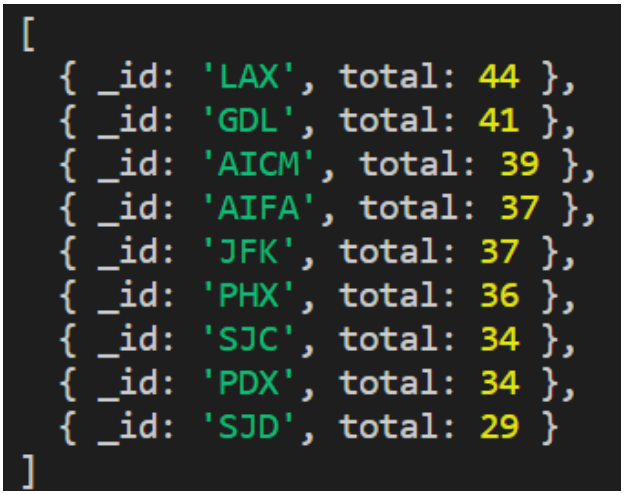
Para resolver el problema que recomiende en qué aeropuertos es recomendable abrir servicios de alimentos/bebidas utilizamos MongoDB. Los datos que decidimos utilizar para resolver la problemática fueron “from” (Airport Origin) y “connection”. En el caso de from fue debido a que los servicios de alimentos y bebidas del aeropuerto son utilizados principalmente desde donde tomas tu vuelo, y el dato de connections lo utilizamos debido a que si el pasajero toma una conexión esperaría en dicho aeropuerto, por lo que saber con qué frecuencia pasa en cada aeropuerto nos ayudará a saber justamente donde implementar más servicios de alimentos y bebidas.



Consultas

En este caso el usuario no tiene un menú con el cual interactuar, por lo que la única consulta se hace al ejecutar el programa, la cual nos muestra cada aeropuerto y la

frecuencia con la que estos son parte de un vuelo conexión, lo cual dictaría que los pasajeros deben esperar un tiempo para poder tomar el vuelo, y en dicho tiempo podrían aprovechar los servicios de alimentos y bebidas.



Optimizaciones

Para evitar la consulta o búsqueda excesiva de datos se optó por desde un inicio descartar todos aquellos vuelos que no poseen una conexión como tal, haciendo que las agrupaciones fueran más veloces de lo que serían si se tomaran todos los datos a la vez.

Conclusiones

Miguel: Este proyectó fue interesante de realizar ya que juntó las 3 bases de datos no relacionales vistas durante el semestre, donde en su mayoría me gustó trabajar con 2 de las 3 bases. Considero que este proyecto fue de provecho ya que considero que algunas de estas bases las podré utilizar en situaciones de la vida real en la vida profesional, tal como Mongo (la cual ya he utilizado en la materia de desarrollo de aplicaciones web). Además, considero que la práctica fue realizada de manera exitosa, a pesar de las dificultades que enfrentamos como extraer los datos del csv o incluso la configuración con docker y el ambiente virtual (cosas que medio se nos había olvidado), pero que al final pudimos resolverlas y entregar un proyecto completo.

Yael: Gracias a esta práctica como es común hubo un repaso sobre gran parte de los temas trabajados durante el semestre, si bien claramente existieron complicaciones debido al tiempo que pudo pasar entre la revisión de los primeros temas como lo son cassandra y

mongodb, estos pudieron resolverse con relativa facilidad tras un tiempo de analizar los temas que iban surgiendo.

Cabe recalcar que si bien usamos la estructura base planteada en un inicio existieron casos donde encontramos información fácilmente despreciable que en ciertas ocasiones podría incluso considerarse un estorbo, por lo que sobre la marcha se debieron hacer pequeñas modificaciones a la estructura hecha en un principio.

Finalmente podemos resumir que todo lo visto es de gran ayuda para lograr tener un mayor número de conocimientos en cuanto a las maneras modernas de guardado de información y teniendo a nuestra disposición un mayor repertorio de herramientas en nuestro futuro.

Fuentes de información

Bonthu, H. (2023, 17 octubre). *How to read and write with CSV files in Python?* Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/08/python-tutorial-working-with-csv-file-for-data-science/

Count - query language. (s. f.). https://dgraph.io/docs/query-language/count/

Anexos

Anexos Cassandra

Anexo 1

```
CREATE_RENTAL_CAMPAIGN_TABLE = """
    CREATE TABLE IF NOT EXISTS rental_campaign (
        destination TEXT,
        month INT,
        transit TEXT,
        day INT,
        reason TEXT,
        PRIMARY KEY ((transit, destination), reason, month, day)
    )
"""
```

Anexo 2

```
CREATE_ALL_FLIGHTS = """
    CREATE TABLE IF NOT EXISTS all_flights(
        origin TEXT,
        destination TEXT,
        year INT,
        month INT,
        day INT,
        airline TEXT,
        PRIMARY KEY((origin, destination), year, month, day, airline )
    )
```

Anexo 3

```
CREATE_ALL_FLIGHTS = """
    CREATE TABLE IF NOT EXISTS all_flights(
        origin TEXT,
```

```
        destination TEXT,
        year INT,
        month INT,
        day INT,
        airline TEXT,
        PRIMARY KEY((origin, destination), year, month, day, airline )
    )


"""
```

Anexo 4

```
SELECT_BEST_MONTH_GENERAL_CAMPAIGN = """
    SELECT destination, month
    FROM general_campaign
    WHERE transit = ? AND destination = ?
"""
```

Anexo 5

```
SELECT_BEST_MONTH = """
    SELECT destination, month, count(*)
    FROM general_campaign
    WHERE transit = ? AND destination = ?
    GROUP BY month
"""
```

Anexo 6

```
SELECT_BEST_MONTH_RENTAL_CAMPAIGN = """
    SELECT destination, month, reason
    FROM rental_campaign
    WHERE transit = ? AND destination = ?
"""
```

Anexo 7

```
SELECT_ALL_FLIGHTS = """
    SELECT origin, destination, year, month, day, airline
    FROM all_flights
"""
```

Anexo 8

```
def get_all_flights(session):
    log.info(f"Retrieving all flights")
    stmt = session.prepare(SELECT_ALL_FLIGHTS)
        rows  =  session.execute(stmt)  #rows  =  session.execute(stmt,
      [username])
    for row in rows:
        print(f"=== From: {row.origin} ===")
        print(f"=== To: {row.destination} ===")
        print(f"=== Date: {row.day}/{row.month}/{row.year} ===")
        print(f"=== Airline: {row.airline} === \n")

def get_general_campaign(session, airport):
    log.info(f"Showing the best options for the campaign")
    stmt = session.prepare(SELECT_BEST_MONTH_GENERAL_CAMPAIGN)
    rows = session.execute(stmt, ["Car rental", airport])
    for row in rows:
        print(f"=== Airport: {row.destination} ===")
        print(f"=== Month: {row.month} === \n")

def get_best_month(session, airport):
    log.info(f"Showing the best options for the campaign")
    stmt = session.prepare(SELECT_BEST_MONTH)
    rows = session.execute(stmt, ["Car rental", airport])
    for row in rows:
        print(f"=== Airport: {row.destination} ===")
        print(f"=== Month: {row.month} === ")
        print(f"=== Count: {row.count} === \n")

def get_detailed_campaign(session, airport):
       log.info(f"Showing  the  best  options  for  the  campaign  (more
     detailed)")
    stmt = session.prepare(SELECT_BEST_MONTH_RENTAL_CAMPAIGN)
    rows = session.execute(stmt, ["Car rental", airport])
```

```
    for row in rows:
        print(f"=== Airport: {row.destination} ===")
        print(f"=== Month: {row.month} ===")
        print(f"=== Reason: {row.reason} === \n")
```

Anexo 9

```
def main():
    log.info("Connecting to Cluster")
    cluster = Cluster(CLUSTER_IPS.split(','))
    session = cluster.connect()

    model.create_keyspace(session, KEYSPACE, REPLICATION_FACTOR)
    session.set_keyspace(KEYSPACE)

    model.create_schema(session)

    while(True):
        print_menu()
        option = int(input('Enter your choice: '))
        if option == 1:
            model.get_all_flights(session)
        if option == 2:
            account = input('Enter the airport to see all data: ')
            model.get_general_campaign(session, account)
        if option == 3:
             account = input('Enter the airport to see all data (more
detailed): ')
            model.get_detailed_campaign(session, account)
        if option == 4:
            account = input('Enter the airport for the Summary: ')
            model.get_best_month(session, account)
        if option == 5:
            exit(0)
```

Anexos Dgraph

Anexo 10

```
schema = """
    type Passenger {
        age
        gender
        takes
    }

    age: int .
    gender: string .
    takes: [uid] @reverse .

    type Flight {
        airline
        month
    }

    airline: string @index(exact) .
    month: int @count @index(int) .

    """
```

Anexo 11

```
def create_data(client):
    client_stub = create_client_stub()
    client = create_client(client_stub)

    model.set_schema(client)

    todo = []

    with open('flight_passengers.csv', newline='') as csvfile:
        datos = csv.reader(csvfile)
        count = True

        for fila in datos:
            if(count):
                count = False
                continue
            obj1 = {}
```

```
            obj2 = {}
                airline, origin , destination, day, month, year, age,
      gender, reason, stay, transit, connection, wait = fila
            obj1['age'] = age
            obj1['gender'] = gender
            obj1['uid'] = "_:"+age+gender
            obj1['dgraph.type'] = 'Passenger'
            obj2['airline'] = airline
            obj2['month'] = month
            obj2['uid'] = "_:"+airline+month
            obj2['dgraph.type'] = 'Flight'
            obj1['takes'] = [obj2]
            todo.append(obj1)

    txn = client.txn()
    try:
        p = todo

        response = txn.mutate(set_obj=p)

        commit_response = txn.commit()
        print(f"Commit Response: {commit_response}")

        print(f"UIDs: {response.uids}")
    finally:
        txn.discard()
```

Anexo 12

```
query = """query count_months($a: int) {
        countByMonth(func: eq(month, $a)) {
            airline
            ~takes{
            total: count(uid)
            }
        }
    }"""
```

Anexo 13

```
query = """query count_months_by_airline($a: string) {
        countByMonth(func: eq(airline, $a)) {
            month
            ~takes{
            total: count(uid)
            }
        }
    }"""
```

Anexo 14

```
def main():
    client_stub = create_client_stub()
    client = create_client(client_stub)

    model.set_schema(client)

    while(True):
        print_menu()
        option = int(input('Enter your choice: '))
        if option == 1:
            model.create_data(client)
        if option == 2:
            person = input("Month (Number): ")
            model.count_months_general(client, person)
        if option == 3:
            person = input("Airline: ")
            model.count_months_by_airline(client, person)
        if option == 4:
            model.drop_all(client)
        if option == 5:
            model.drop_all(client)
            close_client_stub(client_stub)
            exit(0)
```

Anexos Mongo

Anexo 15

```
mongoose.connect(mongoConnection, {useNewUrlParser: true});
let userSchema = mongoose.Schema({
    origin: {
        type: String,
        required: true
    },
    connection: {
        type: String,
        required: true
    }
});
```

Anexo 16

```
if(!bandera)
        return;

    const filePath = 'flight_passengers.csv'; // Reemplaza con la ruta
de tu archivo CSV
    const results = [];

    fs.createReadStream(filePath)
        .pipe(csv())
            .on('data', (data) => results.push({origin: data.origin,
connection: data.connection}))
           .on('end', () => { console.log('Datos leídos del archivo
CSV:'); empezamos(results)});
```

Anexo 17

```
function empezamos(results){
    for(let i of results){
        let user = User(i);
        user.save().then((doc) => console.log(("Usuario creado")));
    }
}

findData();
```

```
function findData(){
    User.aggregate([
        {$match: { connection: "True" }},
        {$group: { _id: '$origin', total: {$sum: 1}}},
        {$sort: { total: -1}}
    ]).then((docs => {
        console.log(docs);
    })).catch((err) => console.log(err))

}
```