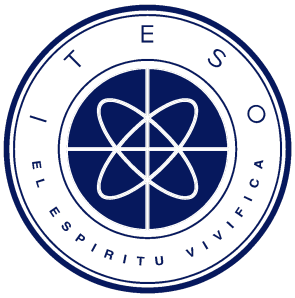
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

BASES DE DATOS NO RELACIONALES



PROYECTO INTEGRADOR BASES DE DATOS NO RELACIONALES || TRAVEL PREDICTOR

Presentan

Cordero Hernández, Marco Ricardo - IS727272

Rodríguez Castro, Carlos Eduardo - IS727366

Díaz Aguayo, Adriana - IS727550

Profesor: Ruiz Rountree, Leobardo

Fecha: 10/05/2023

Contenido

Nombre del proyecto

Travel Predictor

Introducción

El mundo moderno se ha convertido en algo soportado por las interconexiones hechas posibles por los avances tecnológicos. Las aplicaciones de actualidad tienen un único propósito: el movimiento de datos. Ya sea la actualización del saldo bancario de una cuenta al realizar una transacción, hasta la transmisión de cadenas de caracteres entre dos entidades para entablar alguna conversación, la esencia del todo es el movimiento de datos para transformarla en información. La información es el vector que da el sustento a las interacciones en la sociedad actual, puesto que esta representa decisiones, acciones, repercusiones, y muchos otros elementos críticos para el desarrollo de la vida contemporánea.

Hay información que resulta útil de un momento a otro y no vuelve a ser requerida, sin embargo, hay otra que se desea que cuente con persistencia y durabilidad, que se desea que se almacene de alguna u otra forma para posteriormente ser manipulada e interpretada. Es aquí en donde entran las bases de datos, las cuales, dependiendo de las estructuras que pueden almacenar y su enfoque primordial, dan amplio soporte a fracciones de herramientas inimaginables.

A continuación, se presenta una propuesta de proyecto que ha sido desarrollado bajo la premisa de la exploración fuera de los modelos relacionales, y, atendiendo al propósito del curso actual, haciendo uso de tecnologías conocidas como NoSQL.
Con respecto a lo último mencionado, se ha hecho uso de las siguientes bases de datos:

- **Cassandra**: Se utilizaron las ventajas que ofrece como herramienta de almacenamiento columnar para la creación de estructuras flexibles, al mismo tiempo que se hacen uso de las bondades que ofrecen las tablas tradicionales. También, la rapidez que proporciona un buen diseño de tablas en conjunto de una correcta definición de llaves tanto de partición como de agrupamiento, hizo la retribución de datos un proceso sumamente eficiente.

- **MongoDB**: Su modelo en base a documentos ha sido el de mayor utilidad (más no el único) para interconectar los resultados de las consultas hacía el lenguaje sobre el cual se trabajó. Esto ha permitido hacer un mejor manejo de los resultados y proveer una predicción más precisa, además de brindar una espectro extendido de posibilidades en cuanto a construcción y personalización refiere. Adicionalmente, la capacidad de operaciones de agregación y la idealización de relaciones entre colecciones, hicieron

de esta herramienta algo sumamente parecido a lo que se podrían encontrar en una base de datos relacional, lo cual potencialmente sería de suma ayuda para reportes numéricos que se recaban de los datos. Esto aplica para cualquier modelo.

- **Neo4j**: Su modelo de almacenamiento por medio de grafos hizo posible encontrar relaciones dentro de la información utilizando el lenguaje de consultas nativo llamado Cypher en tiempos y complejidades de consulta menores. Las capacidades visuales de la herramienta fueron de gran ayuda para corroborar que los datos que se están extrapolando y su interpretación son correctos, lo cual posteriormente podría ser utilizado para cualquier modelo y posteriormente ser adaptado para soportar fuertemente a uno solo.

Las bases mencionadas han sido manejadas a través de interfaces desarrolladas en el lenguaje de programación *Python* a manera de aplicaciones individuales, las cuales atienden a problemas específicos que se describen en la siguiente sección.

Alcance del Proyecto

Este desarrollo pretende atender un problema en común: análisis de afluencia en aeropuertos desde el punto de vista de una compañía de marketing y consultoría. La manera en que se maneja lo mencionado es mediante la disección de viajes en relación a los siguientes datos:

- Aerolínea (Airline)
- Origen (From)
- Destino (To)
- Día (Day)
- Mes (Month)
- Año (Year)
- Edad (Age)
- Género (Gender) [Hombre, Mujer, Sin especificar, Confidencial)
- Motivo (Reason) [Vacaciones/Placer, Negocios/Trabajo, Regreso a casa]
- Alojamiento (Stay) [Hotel, Estadía en casa foránea de corto plazo, Casa, Amigo/Familia]
- Transporte (Transit) [Taxi de aeropuerto, Renta de automóvil. Servicio de transporte, Transporte público, Aventón, Transporte propio]
- Conexión (Connection) [¿El vuelo es escala/conexión?]
- Espera (Wait) [Si es conexión ¿Cuánto tiempo hay de por medio entre vuelos?]

Del problema previamente mencionado, surgen tres vertientes analíticas, para las cuales se proponen los siguiente modelos:

1. Modelo que sugiere cuáles son los meses en los que es recomendable introducir campañas de publicidad para empresas de renta de carros para cada aeropuerto.

2. Modelo que recomienda en qué aeropuertos es recomendable abrir servicios de alimentos/bebidas.

3. Modelo que determine picos en afluencia de pasajeros en los aeropuertos para valorar la opción de introducir descuentos o distintas estrategias comerciales (propuesto por el equipo).

Una vez que los datos han sido comprendidos y los modelos han sido propuestos, ambos deben ser transformados hacía aplicaciones funcionales. Cabe mencionar que el enfoque de desarrollo que se ha tomado ha sido el de segregar cada modelo en su propia aplicación, de forma que se cuenta con 3 nuevas herramientas que atienden un propósito específico, lo cual, por supuesto cuenta con una posibilidad de mejora y migración hacía un sistema completo con todas sus funciones contenidas dentro de un mismo espacio.

Requerimientos

Como se menciona en la sección anterior, no se cuenta con un sistema completamente integrado con tecnologías interconectadas para dar funciones extendidas, no obstante, esto no hace de las herramientas construidas menos capaces o inútiles.

Para el propósito descriptivo de la sección, se verá a los desarrollos como componentes de un gran sistema, del predictor de viajes.

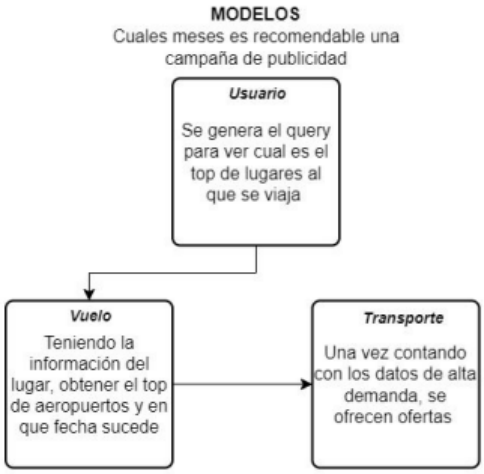
- El sistema deberá ser capaz de atender los requerimientos descritos previamente
  - El sistema deberá ser capaz de recomendar temporadas óptimas para la introducción de campañas publicitarias enfocadas a la renta de vehículos de transporte personal.
  - El sistema deberá ser capaz de realizar observaciones oportunas para la apertura de nuevos establecimientos de comida/autoservicio en distintos aeropuertos.
  - El sistema deberá ser capaz de valorar la posibilidad de implantación de distintos tipos de descuentos basados en el análisis de flujo de pasajeros.
- El sistema deberá contar con parámetros de entrada para cada modelo propuesto, brindando mayor flexibilidad de consulta.
- El sistema deberá ser resiliente a fallos en su operación, de forma que no se muestren errores fatales que terminarían con su ejecución relacionados al funcionamiento interno.
- El sistema deberá estar protegido contra inyecciones de consultas que podrían filtrar información confidencial.
- El sistema deberá manejar la autenticación de credenciales internamente, es decir, sin requerir de la intervención del usuario para su acceso (al menos para la propuesta actual).

Diseño de la solución

Modelos de datos utilizados

Modelo 1:

Este modelo sugiere cuáles son los meses en los que es recomendable introducir campañas de publicidad para empresas de renta de carros para cada aeropuerto, fue implementado en CassandraDB. A partir de la cantidad de pasajeros registrados por aeropuerto de destino final se cuentan y agrupan la suma para poder mostrar al personal de publicidad cuáles son los meses indicados para lanzar campañas, para esto asumimos que los meses con mayor afluencia de pasajeros serían los meses de interés.

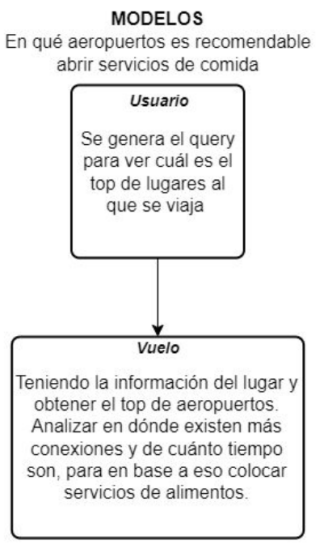


Modelo 2:

El siguiente modelo analiza el tipo de viaje y sus conexiones directas para determinar si sería óptimo instaurar nuevos establecimientos de consumibles en los aeropuertos. Se pensó este modelo como algo fuertemente conectado, por lo tanto, la herramienta predilecta ha sido Neo4j, por la posibilidad de ofrecer consultas eficientes para el análisis de relaciones, siendo esto un concepto elemental de los grafos, es decir, el fundamento sobre el cual está construida esta base.

Como detalla el diagrama siguiente, primero se obtienen los aeropuertos con mayor tráfico de pasajeros, luego, a partir de la determinación de estos datos, se aplica una segunda tanda analítica y de conteo para finalmente determinar la conveniencia de aperturas en los lugares previamente recabados.

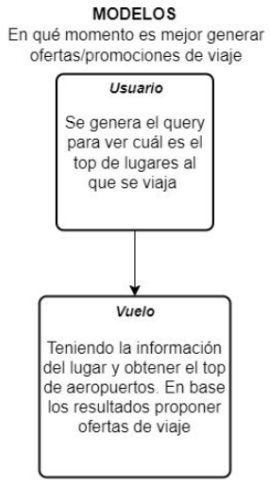
La forma en la que los datos son procesados hacen posible la omisión de algunas columnas en el conjunto de datos inicial, tales como género, edad, etc.



Modelo 3:

El último modelo alberga un funcionamiento un tanto sencillo, pero no por ello lo hace menos útil para el propósito planeado. La forma en que se manipulan los datos es mediante la obtención de dos extremos distintos de flujo de gente: puntos de mayor y menor cantidad de viajes. Según estas interpretaciones, se proponen distintas estrategias de descuentos a implantar a conveniencia de la situación.

Al requerir varias etapas de procesamiento para datos con relaciones no tan fuertes, MongoDB fue la herramienta de preferencia para concretar este requerimiento del proyecto. Su sencilla integración con Python han hecho de su uso algo bastante intuitivo y de suma utilidad.



Consultas

Modelo 1:

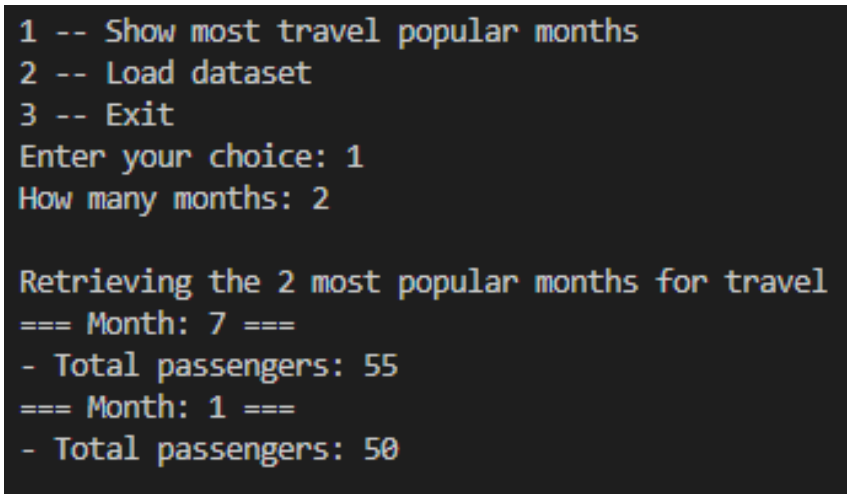
Las consultas más frecuentes realizadas al modelo serán para insertar nuevos registros de pasajeros y vuelos. También se realizará una consulta para obtener los meses más populares de vuelos para poder satisfacer el propósito del modelo. Pero debido a que los datos no cambian inmediatamente, ésta consulta no se realiza tan frecuentemente y probablemente solo se ejecute para actualizar los registros una vez al año. Las consultas trabajan con la siguiente estructura de la tabla *"passengers"*:

Todo se ejecuta de forma programática, existe la solución de correr consultas dentro de cqlsh, pero para esta solución se realiza a través de consultas hechas desde Python donde se valida el esquema y el keyspace. Se creó una función para cargar el dataset que genera el script flight_data.py.

El script de Python (encontrado en anexos) se compone de model.py, donde se define el modelo de datos, y app.py donde se define la interacción que tendrá el usuario con el modelo. App.py le muestra al usuario 2 opciones, la primera es obtener los meses con mayor cantidad de pasajeros y la segunda es cargar información de un dataset. La segunda opción se crea a partir de la necesidad de cargar los datos del script que el profesor proporcionó.

Una vez que el dataset y el esquema han sido cargados al keyspace de CassandraDB se habilita la opción de obtener los resultados (primera opción).

Este es el resultado de pedir los 2 meses más populares de viaje, el mes 7 (julio) y el mes 1 (enero). Podemos ver que hubo 55 pasajeros en el mes de julio y 50 en enero a partir de un dataset arbitrario con 500 registros. Estos serían los meses a recomendar para introducir una campaña de publicidad para la renta de autos dentro de los aeropuertos.

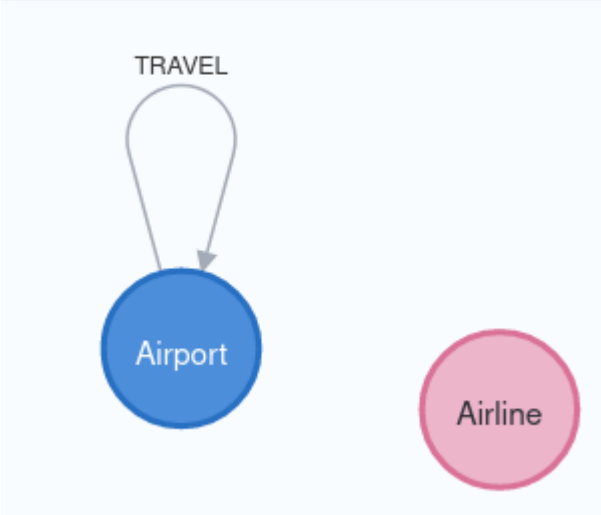


Modelo 2:

Se construyeron varias consultas para el trabajo con Neo4j, tanto para la inserción de datos como para su análisis. El código puede ser encontrado en la sección de anexos.

El modelo generado es sumamente sencillo, pero funcional.

```
CALL db.schema.visualization
```



Nodos:
- Airline
  - name:str
- Airport
  - id:str

Relaciones:
- TRAVEL
  - date:date
  - wait:int
  - connection:bool

Para el caso, no se hace uso de la información relacionada a las aerolíneas, sin embargo, se ha dejado dentro de la base para futuras implementaciones posibles.

Para conveniencia de su análisis, todas las consultas y sentencias utilizadas son las siguientes:

```
// Creación de constraints
CREATE CONSTRAINT airline_constraint IF NOT EXISTS FOR (al:Airline) REQUIRE al.name IS UNIQUE
CREATE CONSTRAINT airport_constraint IF NOT EXISTS FOR (ap:Airport) REQUIRE ap.id IS UNIQUE

// Limpieza inicial de la base
MATCH (n) DETACH DELETE n

// Creación de nodo tipo aerolínea
CREATE (al:Airline {name: $name})

// Creación de nodo tipo aeropuerto
CREATE (ar:Airport {id: $id})

// Creación de relación de tipo viaje entre aeropuertos
MATCH (org:Airport {id: $id_from}), (dest:Airport {id: $id_to})
MERGE (org)-[:TRAVEL {date: date($date), connection: $connection, wait: $wait}]->(dest)

// Búsqueda de resultados para predicción
MATCH (org:Airport)-[t:TRAVEL]->(dest:Airport) WHERE t.connection = "True"
[AND date >= date($date)]
[AND date <= date($date)]
RETURN org.id as ORIGIN, COUNT(dest) AS CONNECTIONS, avg(t.wait) AS WAIT_AVG
ORDER BY [WAIT_AVG, CONNECTIONS | CONNECTIONS, WAIT_AVG] DESC
```

```
[LIMIT N]

//* --- GDS --- *//
// Page Rank
CALL gds.graph.project(
  'mod2_page_rank',
  'Airport',
  'TRAVEL',
  {
    relationshipProperties: 'wait'
  }
)

CALL gds.pageRank.stream('myGraph')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).id AS name, score
ORDER BY score DESC, name ASC

// Closeness Centrality
CALL gds.graph.project('mod2_centrality', 'Airport', 'TRAVEL')

CALL gds.beta.closeness.stream('mod2_centrality')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).id AS id, score
ORDER BY score DESC
```

Como se puede observar, las consultas tienen un grado considerable de personalización en cuanto a flexibilidad y entrega del resultado se refiere, no obstante, su construcción no es tan compleja como parece. La forma de obtener las recomendaciones se sustenta puramente en el análisis estadístico de la mayor cantidad de pasajeros que incurren en los aeropuertos en un carácter de vinculación entre vuelos, esto, en conjunto del análisis del tiempo de espera para estas instancias, determinan la manera de obtener las predicciones deseadas. Los cálculos recaen puramente en la relación *"TRAVEL"*, la cual almacena los detalles de viajes entre aeropuertos, indicando su fecha, si se trata de una conexión entre vuelos, y, en caso de ser lo último, su tiempo de espera entre ellos.

Para utilizar el script, existen las siguientes instrucciones:

```
python3 travel_predictor_mod2.py
usage: travel_predictor_mod2.py [-h] [-f FILE] [-t TOP] [-r REVERSE] [-s START] [-e END] [-l LINKS] [-c CENTRALITY] {fill,predict,stats}

positional arguments:
  {fill,predict,stats}  Available application actions

options:
  -h, --help            show this help message and exit
  -f FILE, --file FILE  Dataset for database filling (csv format)
  -t TOP, --top TOP     Limit results to [top] records
  -r REVERSE, --reverse REVERSE
                        Reverse sorting order of (Waiting avg, Connection amount) factor
  -s START, --start START
                        Starting date in format (dd-mm-yyy)
  -e END, --end END     Ending date in format (dd-mm-yyy)
  -l LINKS, --links LINKS
                        Execute Page Rank algorithm for link between aiports
  -c CENTRALITY, --centrality CENTRALITY
                        Execute Closeness Centrality algorithm for node inspection
```

Y algunas muestras de su utilización (con un dataset de 2500 registros obtenido de fligh_data.py):

```
(venv) marcordero@mrch-ubuntu:~/.../NoSQL_P2023/Proyecto/Neo4j$ python3 travel_predictor_mod2.py predict
The top 5 most suitable airports to open more establishments are the following
1 - Airport ID: DEN
        - Connections: 64
        - Waiting time average per connection: 313.66 minutes

2 - Airport ID: LAX
        - Connections: 73
        - Waiting time average per connection: 334.25 minutes

3 - Airport ID: SJC
        - Connections: 72
        - Waiting time average per connection: 349.85 minutes

4 - Airport ID: IZT
        - Connections: 82
        - Waiting time average per connection: 354.99 minutes

5 - Airport ID: MTY
        - Connections: 89
        - Waiting time average per connection: 360.93 minutes

(venv) marcordero@mrch-ubuntu:~/.../NoSQL_P2023/Proyecto/Neo4j$ python3 travel_predictor_mod2.py predict -s 08/06/2020 -e 01/01/2023 -t 3 -r True
The top 3 most suitable airports to open more establishments starting from 08-06-2020 up to 01-01-2023 are the following
1 - Airport ID: DEN
        - Connections: 12
        - Waiting time average per connection: 293.58 minutes

2 - Airport ID: PDX
        - Connections: 16
        - Waiting time average per connection: 299.00 minutes

3 - Airport ID: GDL
        - Connections: 17
        - Waiting time average per connection: 333.24 minutes
```

```
(venv) marcordero@mrch-ubuntu:~/.../NoSQL_P2023/Proyecto/Neo4j$ python3 travel_predictor_mod2.py stats -l 1 -c 1
-- PAGE RANK --
Aeropuerto: TPQ - Puntuación: 1.0149778466995731
Aeropuerto: JFK - Puntuación: 0.9569508379589623
Aeropuerto: PVR - Puntuación: 0.9628429593865205
Aeropuerto: GDL - Puntuación: 1.0225198006291152
Aeropuerto: PDX - Puntuación: 1.0087654978014295
Aeropuerto: DEN - Puntuación: 0.9358625858591483
Aeropuerto: SJC - Puntuación: 0.921809675058331
Aeropuerto: IZT - Puntuación: 0.9207295041544812
Aeropuerto: TLC - Puntuación: 0.9729850408760293
Aeropuerto: MTY - Puntuación: 1.0022126930572688
Aeropuerto: LAX - Puntuación: 0.8539887165894766

-- CENTRALITY --
Aeropuerto: TPQ - Puntuación: 1.0
Aeropuerto: JFK - Puntuación: 1.0
Aeropuerto: PVR - Puntuación: 1.0
Aeropuerto: GDL - Puntuación: 1.0
Aeropuerto: PDX - Puntuación: 1.0
Aeropuerto: DEN - Puntuación: 1.0
Aeropuerto: SJC - Puntuación: 1.0
Aeropuerto: IZT - Puntuación: 1.0
Aeropuerto: TLC - Puntuación: 1.0
Aeropuerto: MTY - Puntuación: 1.0
Aeropuerto: LAX - Puntuación: 1.0
```

Modelo 3:

Se construyeron consultas de obtención de datos específicos a tablas auxiliares y una consulta general que atienden al propósito del modelo a través de MongoDB. El código puede ser encontrado en la sección de anexos **.**

La estructura de los documentos nuevamente resulta sencilla, pero útil, siendo la siguiente.

Colecciones:

Airline
- name:str

Airport
- id:str

Travel
- airline:str
- from:str
- to:str
- connection:bool
- date:date
- reason:str
- stay:str

Las consultas y sentencias utilizadas son las siguientes.

```
// Limpieza inicial
db.[coleccion].insertMany({})
db.[coleccion].deleteMany({})

// Índices
db.travel.createIndex({'date':1})
db.travel.createIndex({'date':1, 'from':1})
db.travel.createIndex({'date':1, 'airline':1})
db.travel.createIndex({'date':1, 'from': 1, 'airline':1})

db.travel.createIndex({'from':1})
db.travel.createIndex({'airline':1})
db.travel.createIndex({'from': 1, 'airline':1})

// Obtener todos los aeropuertos
db.airport.aggregate({'$project': {'_id':0, 'id':'$id'}})

// Obtener todas las aerolíneas
db.airline.find({}, {'_id':0})

// Consulta principal
db.travel.aggregate([
    {
        $match:
            {
                'connection': {'$ne': 'true'},
                'stay': {'$in': ['Hotel','Short-term homestay']},
                'reason': {'$in': ['On vacation/Pleasure','Back Home']},
                'airline': '$QUERY_AIRLINE$',
                'from': '$QUERY_ORIGIN_AIRPORT$',
                'date': {'$gte': ISODate('$QUERY_START_DATE$'), '$lt': ISODate('$QUERY_END_DATE$')}
            }
    },
    {
        $group: {
            '_id': {'$month': "$date"},
            'qty': {'$sum': 1}
        }
    },
```

```
    {
        $project:
            {
                '_id': 0,
                'month': '$_id',
                'qty': '$qty'
            }
    },
    {
        $sort:
            { 'qty' : -1 } }
])
```

Para el presente modelo se construyó una consulta compleja con alto grado de customización, permitiendo definición de parámetros tales como aeropuerto, aerolínea y rangos de fecha (pudiendo omitir ambos extremos). Si se observa un poco la estructura, se podrá percatar que nuevamente se han manejado las predicciones a través de la colección ***TRAVEL***, sobre la cual recaen las conexiones pertinentes entre entidades menores. Se podría ver como una entidad débil de los modelos relaciones tradicionales; irónicamente, es la de mayor utilidad aún siendo un vínculo “sencillo”.

Para utilizar el script, existen las siguientes instrucciones:

```
python3 travel_predictor_mod3.py
usage: travel_predictor_mod3.py [-h] [-f FILE] [-s START] [-e END] {fill,predict}

positional arguments:
  {fill,predict}        Acciones disponibles

options:
  -h, --help            show this help message and exit
  -f FILE, --file FILE  Set de datos para el llenado de la base (formato csv)
  -s START, --start START
                        Año de inicio para rango de búsqueda
  -e END, --end END     Año de término para rango de búsqueda
```

La selección de aeropuerto y aerolínea es manejada internamente por el programa, esto debido a que los datos idealmente estarían cambiando con el tiempo, por lo tanto, las opciones siempre tendrían que ser dinámicas.

```
Ejecución sin parámetros
python3 travel_predictor_mod3.py predict

¿Deseas un aeropuerto en específico [S/n]: n

¿Deseas una aerolínea específica? [S/n]: n

Mostrando posibles ofertas


Resultados:
Aeropuerto "TPQ"
        Posibilidad de introducción de paquetes grupales o similar
        -> Junio, Noviembre, Octubre

        Posibilidad de descuentos por baja demanda
        -> Septiembre, Febrero, Julio

Aeropuerto "JFK"
        Posibilidad de introducción de paquetes grupales o similar
        -> Febrero, Enero, Octubre

        Posibilidad de descuentos por baja demanda
        -> Abril, Junio, Marzo

Aeropuerto "PVR"
        Posibilidad de introducción de paquetes grupales o similar
        -> Marzo, Junio, Enero

        Posibilidad de descuentos por baja demanda
        -> Mayo, Octubre, Diciembre

Aeropuerto "GDL"
        Posibilidad de introducción de paquetes grupales o similar
        -> Junio, Enero, Abril

        Posibilidad de descuentos por baja demanda
```

```
        -> Febrero, Agosto, Junio

Aeropuerto "SJC"
        Posibilidad de introducción de paquetes grupales o similar
        -> Junio, Enero, Mayo

        Posibilidad de descuentos por baja demanda
        -> Abril, Noviembre, Febrero

Aeropuerto "IZT"
        Posibilidad de introducción de paquetes grupales o similar
        -> Julio, Noviembre, Enero

        Posibilidad de descuentos por baja demanda
        -> Diciembre, Abril, Mayo

Aeropuerto "TLC"
        Posibilidad de introducción de paquetes grupales o similar
        -> Febrero, Julio, Octubre

        Posibilidad de descuentos por baja demanda
        -> Noviembre, Marzo, Agosto

Aeropuerto "MTY"
        Posibilidad de introducción de paquetes grupales o similar
        -> Mayo, Marzo, Abril

        Posibilidad de descuentos por baja demanda
        -> Diciembre, Agosto, Noviembre

Aeropuerto "LAX"
        Posibilidad de introducción de paquetes grupales o similar
        -> Septiembre, Mayo, Enero

        Posibilidad de descuentos por baja demanda
        -> Febrero, Noviembre, Junio

Mes con mejor posibilidad de introducción de promociones: Febrero
```

```
Ejecución con parámetros
python3 travel_predictor_mod3.py predict -s 2019 -e 2022

¿Deseas un aeropuerto en específico [S/n]:
Selecciona un aeropuerto para realizar la predicción
1 - TPQ
2 - JFK
3 - PVR
4 - GDL
5 - PDX
6 - DEN
7 - SJC
8 - IZT
9 - TLC
10 - MTY
11 - LAX
Selección: TPQ

¿Deseas una aerolínea específica? [S/n]: s
Selecciona una aerolínea
1 - Aeromar
2 - Alaska
3 - Aeromexico
4 - Lufthansa
5 - Volaris
6 - Viva Aerobus
7 - Delta Airlines
8 - American Airlines
Selección: 7
No hay recomendaciones disponibles para el conjunto de parámetros actual.
```

```
Ejecución con parámetros
python3 travel_predictor_mod3.py predict -s 2023

¿Deseas un aeropuerto en específico [S/n]: s
Selecciona un aeropuerto para realizar la predicción
1 - TPQ
2 - JFK
3 - PVR
4 - GDL
5 - PDX
6 - DEN
7 - SJC
8 - IZT
9 - TLC
10 - MTY
11 - LAX
Selección: 10

¿Deseas una aerolínea específica? [S/n]: s
Selecciona una aerolínea
1 - Aeromar
2 - Alaska
3 - Aeromexico
4 - Lufthansa
5 - Volaris
6 - Viva Aerobus
7 - Delta Airlines
8 - American Airlines
Selección: 5

Mostrando posibles ofertas
        Parámetros:
                - Desde 2023
                - Aerolínea "Volaris"
                - Aeropuerto "MTY"

Resultados:
        Posibilidad de introducción de paquetes grupales o similar
        -> Enero, Febrero

        Posibilidad de descuentos por baja demanda
        -> Enero, Febrero
```

```
Ejecución con parámetros
python3 travel_predictor_mod3.py predict -e 2021

¿Deseas un aeropuerto en específico [S/n]: n

¿Deseas una aerolínea específica? [S/n]: s
Selecciona una aerolínea
1 - Aeromar
2 - Alaska
3 - Aeromexico
4 - Lufthansa
5 - Volaris
6 - Viva Aerobus
7 - Delta Airlines
8 - American Airlines
Selección: 3

Mostrando posibles ofertas
        Parámetros:
                - Hasta 2021
                - Aerolínea "Aeromexico"

Resultados:
Aeropuerto "TPQ"
        Posibilidad de introducción de paquetes grupales o similar
        -> Noviembre, Mayo, Junio

        Posibilidad de descuentos por baja demanda
        -> Mayo, Junio, Septiembre

Aeropuerto "JFK"
        Posibilidad de introducción de paquetes grupales o similar
        -> Febrero, Diciembre

        Posibilidad de descuentos por baja demanda
        -> Febrero, Diciembre
```

```
Aeropuerto "SJC"
        Posibilidad de introducción de paquetes grupales o similar
        -> Mayo

        Posibilidad de descuentos por baja demanda
        -> Mayo

Aeropuerto "IZT"
        Posibilidad de introducción de paquetes grupales o similar
        -> Junio

        Posibilidad de descuentos por baja demanda
        -> Junio

Aeropuerto "TLC"
        Posibilidad de introducción de paquetes grupales o similar
        -> Abril, Diciembre, Mayo

        Posibilidad de descuentos por baja demanda
        -> Mayo, Octubre, Enero

Aeropuerto "MTY"
        Posibilidad de introducción de paquetes grupales o similar
        -> Junio, Octubre, Mayo

        Posibilidad de descuentos por baja demanda
        -> Octubre, Mayo, Marzo

Aeropuerto "LAX"
        Posibilidad de introducción de paquetes grupales o similar
        -> Septiembre, Enero

        Posibilidad de descuentos por baja demanda
        -> Septiembre, Enero

Mes con mejor posibilidad de introducción de promociones: Mayo
```

Índices y Optimizaciones

Modelo 1:

Por diseño, CassandraDB es una base de datos columnar que no soporta unión de tablas (JOIN) ni sentencias de agrupamiento de diferentes tablas en una misma consulta esto nos lleva a duplicar información en diferentes tablas. La ventaja de CassandraDB es que estos duplicados permiten tener una mayor velocidad de consulta. Para este modelo se duplicaron todos los datos del vuelo dentro de la tabla de pasajeros para facilitar las consultas dirigidas a contar la cantidad de pasajeros por mes. También se hizo uso del driver de CassandraDB para Python para mostrar el resultado de forma más sencilla y también hacer un procesamiento de las cadenas de caracteres que arroja la consulta ya que CQL es muy restrictivo en este aspecto.

Modelo 2:

A diferencia de otras bases utilizadas, la ventaja que presenta Neo4j como base orientada a grafos es que no hace uso de operaciones complejas basadas en teorías de conjuntos como uniones, intersecciones, etc., si no que almacena las relaciones entre los datos nativamente, lo cual permite una rápida obtención de resultados en consultas hechas hacía la propia base.

Adicional a lo anterior, la omisión de campos de datos irrelevantes hicieron posible una mejora tanto en la parte de inserción de nuevos datos y su posterior manejo para el reporte de los resultados.

También, se ha de mencionar que se crearon un par de índices de manera nativa en la base, esto para evitar duplicidad y asegurar una operación idónea dentro de los grafos.

Modelo 3:

Similar al modelo anterior, gran parte de la rapidez atribuida a este último modelo es el correcto tratamiento de los datos al momento de su ingreso en la base utilizada, lo cual se traduce directamente a omisión de datos irrelevantes. En este punto de la exposición del detallado vale la pena indicar que en cierta medida u otra, todos los modelos omitieron piezas de información del set de datos original, algo que en realidad no es incorrecto, ya que, si se realiza un análisis ligeramente más reflexivo, se podrá dar cuenta que la comprensión de las demográficas no siempre involucran todas las variables en un sistema, en este caso, no toda la información resulta relevante para un correcto resultado. Quizás con la aplicación de algoritmos de análisis más especializados se podría encontrar una correlación crítica, pero eso va más allá del alcance actual propuesto.

La mayor ventaja de la base sobre la cual se base este modelo ha sido la de la posibilidad de manejar el mismo conjunto de datos a través de diferentes etapas de post proceso, evitando la creación de rutinas complejas de transformación de los datos, lo cual, por consecuencia, acelera significativamente el despliegue de resultados.

Por si lo anterior fuera poco, como se puede observar en las sentencias anteriores, se crearon índices para la colección sobre la cual se obtienen los datos para las predicciones.

```
proj_mod3> db.travel.createIndex({'date':1})
date_1
proj_mod3> db.travel.createIndex({'date':1, 'from':1})
date_1_from_1
proj_mod3> db.travel.createIndex({'date':1, 'airline':1})
date_1_airline_1
proj_mod3> db.travel.createIndex({'date':1, 'from': 1, 'airline':1})
date_1_from_1_airline_1
proj_mod3>

proj_mod3> db.travel.createIndex({'from':1})
from_1
proj_mod3> db.travel.createIndex({'airline':1})
airline_1
proj_mod3> db.travel.createIndex({'from': 1, 'airline':1})
from_1_airline_1
```

Conclusiones

Rodríguez Castro:

El tratar con estos 3 motores de bases de datos no relacionales me hizo identificar los casos de uso donde podemos favorecer el uso de uno sobre otro debido a las ventajas que ofrece. El trabajar con CassandraDB fue complicado ya que no soporta muchas operaciones a los que estaba acostumbrado con motores como MySQL (base de datos relacional) y no era tan flexible con las consultas como MongoDB. Con Neo4j también tuve problemas, pero estos no fueron tan serios como los anteriores, sino que eran errores de sintaxis con Cypher. En general me llevo muchos aprendizajes de todos los motores utilizados para realizar este proyecto, espero que en un futuro pueda aplicar y expandir este conocimiento de forma profesional una vez que me gradúe.

Díaz Aguayo:

La realización de este proyecto ayudó a aterrizar los conocimientos adquiridos en el curso de las distintas bases no relacionales que existen y dónde emplear cada una de ellas.

Cordero Hernández:

Llegado a este punto del proyecto, e incluso, del curso, se ha de admitir que el trayecto no ha sido algo fácil. Puede que fuese intuitivo, pero eso no implica la calidad de sencillez. Los aprendizajes logrados y adoptados con cada clase, actividad, tarea y laboratorio asignado han sido de gran utilidad, puesto que, como se espera que se haya demostrado en este desarrollo, se han aplicado satisfactoriamente para dejar de lado la teoría y práctica (en el sentido más puro de la palabra) para atender a lo que podría ser una problemática o área de mejora real.

Colaborar en una trina complicó un poco las cosas, ya que se tuvo que hacer uso de todas las bases revisadas a lo largo de la materia, no obstante, finalmente se trabajó de forma productiva y fructífera, puesto que el cometido del proyecto llegó a concretarse como se había planeado desde su inicio.

Cabe destacar que las bases de Cassandra y MongoDB fueron las que mayor impacto provocaron en cuanto a nuevos aprendizajes refiere, ya que en materias anteriores, ya se había aprendido acerca de Neo4j, y por consecuencia, también se había utilizado. A pesar de este hecho, cada momento en que el conocimiento se presenta ha sido bien recibido y repleto de reflexión, principalmente para la posibilidad de aplicar los aprendizajes actuales en un ámbito laboral en un futuro no tan distante.

Fuentes de información

Apache Cassandra. (s.f.). *The Cassandra Query Language (CQL)*. Recuperado de https://cassandra.apache.org/doc/latest/cassandra/cql/.

DataStax Documentation. (s.f.). *DataStax Python Driver 3.27 (Latest version)*. Recuperado de *https://docs.datastax.com/en/developer/python-driver/3.27/api/cassandra/*.

Neo4j. (s.f.). *API Documentation*. Recuperado de https://neo4j.com/docs/api/python-driver/current/api.html.

Neo4j. (2022). *The Neo4j Graph Data Science Client Manual v1.6*. Recuperado de https://neo4j.com/docs/graph-data-science-client/current/.

MongoDB. (s.f.). *How to Use Python with MongoDB*. Recuperado de https://www.mongodb.com/languages/python.

MongoDB. (2023). *MongoDB Documentation*. Recuperado de https://www.mongodb.com/docs/.

Anexos

Repositorio con códigos utilizados:

- https://github.com/Marcox385/NoSQL_P2023/tree/main/Proyecto

Código para modelo 1 (Cassandra):

```
CREATE_KEYSPACE = """
        CREATE KEYSPACE IF NOT EXISTS {}
        WITH replication = {{ 'class': 'SimpleStrategy', 'replication_factor': {} }}
"""


CREATE TABLE IF NOT EXISTS passengers (
        passenger_id UUID,
        airline TEXT,
        airport_from TEXT,
        airport_to TEXT,
        flight_day INT,
        flight_month INT,
        flight_year INT,
        age INT,
        gender TEXT,
        travel_reason TEXT,
```

```
        stay TEXT,
        transit TEXT,
        connection BOOLEAN,
        wait INT,
        PRIMARY KEY((flight_month, passenger_id), flight_year)
    )

def create_schema(session):
    log.info("Creating model schema")
    session.execute(CREATE_PASSENGERS_TABLE)

def load_dataset(session, dataset_path):
    with open(dataset_path, "r") as file:
        csvreader = csv.reader(file)
        for row in csvreader:
            if row != [] and row[0] != "airline":
                tmp = row
                try:
                    # Integer conversion
                    tmp[3] = int(tmp[3])
                    tmp[4] = int(tmp[4])
                    tmp[5] = int(tmp[5])
                    tmp[6] = int(tmp[6])
                    tmp[-1] = int(tmp[-1])
                    tmp[-2] = bool(tmp[-2])
                except Exception as e:
                    pass
                else:
                    print(tmp)
                    model.insert_passenger(session, tmp)
INSERT_PASSENGER = """
    INSERT INTO passengers(
        passenger_id,
        airline,
        airport_from,
        airport_to,
        flight_day,
        flight_month,
        flight_year,
        age,
        gender,
```

```
        travel_reason,

        stay,

        transit,

        connection,

        wait) VALUES(

            uuid(), ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?);

"""


def get_most_popular_travel_months(session, months_qty):

    log.info(f"Retrieving the {months_qty} most popular months for travel")

    stmt = session.prepare(SELECT_POPULAR_MONTHS)

    rows = session.execute(stmt)
```

Código para modelo 2 (Neo4j):

```
#!/usr/bin/env python3
''' Modelo 2: Modelo que recomienda en qué aeropuertos es
recomendable abrir servicios de alimentos/bebidas '''
import argparse, os, sys

from neo4j import GraphDatabase
from graphdatascience import GraphDataScience
from neo4j.exceptions import ClientError, ConstraintError

class TravelPredictor(object):

    def __init__(self, uri, user, password):
        self._uri = uri
        self._AUTH = (user, password)
        self.driver = GraphDatabase.driver(self._uri, auth=self._AUTH)
        self._create_constraints()

        self.transactions = {
            'DELETE_ALL': 'MATCH (n) DETACH DELETE n',
            'Airline': 'CREATE (al:Airline {name: $name})',
            'Airport': 'CREATE (ar:Airport {id: $id})',
            'Travel': '''MATCH (org:Airport {id: $id_from}), (dest:Airport {id: $id_to}) '''
                                    '''MERGE (org)-[:TRAVEL {date: date($date), connection: $connection, wait:
$wait}]->(dest)'''
        }

    def __enter__(self):
        return self

    def _create_constraints(self):
        ''' Create constraints'''
        def inner_tx(tx):
                tx.run('CREATE CONSTRAINT airline_constraint IF NOT EXISTS FOR (al:Airline) REQUIRE al.name IS
UNIQUE')
            tx.run('CREATE CONSTRAINT airport_constraint IF NOT EXISTS FOR (ap:Airport) REQUIRE ap.id IS UNIQUE')

        with self.driver.session() as session:
            session.execute_write(inner_tx)

    def _generic_write_tx(self, exec_str, **kwargs):
        def inner_tx(tx):
            tx.run(exec_str, **kwargs)

        with self.driver.session() as session:
            try:
                session.execute_write(inner_tx)
            except (ClientError, ConstraintError) as e:
                print(f'Error in transaction: {exec_str}\nParameters -> {kwargs}\nFull trace -> {e}')
```

```
                exit(1)

    def fill(self, source):
        ''' Populate database given input dataset '''
        airlines = set()
        airports = set()
        op_count = 1

        # Reset database contents
        self._generic_write_tx(self.transactions['DELETE_ALL'])

        with open(source, newline='') as csv:
            headers = csv.readline().rstrip('\r\n').split(',')
            curr_data = {}
            lines = csv.readlines()

            for line in lines:
                curr_data = {k:v for k, v in zip(headers, line.rstrip('\r\n').split(','))}

                if (curr_data['airline'] not in airlines):
                    self._generic_write_tx(self.transactions['Airline'], name=curr_data['airline'])
                    airlines.add(curr_data['airline'])

                if (curr_data['from'] not in airports):
                    self._generic_write_tx(self.transactions['Airport'], id=curr_data['from'])
                    airports.add(curr_data['from'])

                if (curr_data['to'] not in airports):
                    self._generic_write_tx(self.transactions['Airport'], id=curr_data['to'])
                    airports.add(curr_data['to'])

                date = '-'.join([curr_data['year'], curr_data['month'], curr_data['day']])

                self._generic_write_tx(self.transactions['Travel'], id_from=curr_data['from'],
                                                                    id_to=curr_data['to'],    date=date,
connection=curr_data['connection'],wait=int(curr_data['wait']))

                op_count += 1
                print(f"Progreso: {op_count/len(lines):.0%}", end='\r')
        sys.stdout.write("\033[K")
        print('Finalizado')

    def search_range(self, top:int=5, reverse:bool=False, start:str='', end:str=''):
        '''
            Search nodes between a start and end date to predict viability
            of new stores in airports
        '''
        def inner_tx(tx, limit, order, start_date, end_date):
            limit = f' LIMIT {limit}' if limit else ''
            order = ' ORDER BY ' +  ('WAIT_AVG, CONNECTIONS' if not order else 'CONNECTIONS, WAIT_AVG') + ' DESC'

            result = tx.run('MATCH (org:Airport)-[t:TRAVEL]->(dest:Airport) WHERE t.connection = "True"'
                   + start_date + end_date + ' RETURN org.id as ORIGIN, COUNT(dest) AS CONNECTIONS, avg(t.wait) AS
WAIT_AVG'
                + order + limit)
            return [record.values() for record in result]

        with self.driver.session() as session:
            start_date = ''
            end_date = ''

            if (start):
                if ('/' in start): start = start.replace('/', '-')
                start_date = list(map(lambda x: x.lstrip('0'), start.split('-')))
                start_date = '{' + f'year: {start_date[2]}, month: {start_date[1]}, day: {start_date[0]}' + '}'
                start_date = f' AND t.date >= date({start_date})'
                start = f'desde {start} '

            if (end):
                if ('/' in end): end = end.replace('/', '-')
                end_date = list(map(lambda x: x.lstrip('0'), end.split('-')))
                end_date = '{' + f'year: {end_date[2]}, month: {end_date[1]}, day: {end_date[0]}' + '}'
                end_date = f' AND t.date <= date({end_date})'
                end = f'hasta {end} '

            try:
                top = int(top)
```

```
                if (top <= 0):
                    raise ValueError
            except ValueError:
                top = 5

            try:
                reverse = bool(reverse)
            except ValueError:
                reverse = False

            results = session.execute_read(inner_tx, top, reverse, start_date, end_date)

            print(f'El top {top} de aeropuertos con posibilidad de apertura de establecimientos '
                + (start if start else '') + (end if end else '') + 'son los siguientes')
            for i, r in enumerate(results, 1):
                  print(f'{i} - ID de Aeropuerto: {r[0]}\n\t- Conexiones: {r[1]}\n\t- Promedio de tiempo de espera
por conexión: {r[2]:.2f} minutes\n')

    def stats(self, links:bool=False, centrality:bool=False):
        ''' Call GDS Algorithm for node inspection '''
        with self.driver.session() as session:
            gds = GraphDataScience(self._uri, auth=self._AUTH)

            try:
                gds.run_cypher("CALL gds.graph.drop('mod2_page_rank')")
                gds.run_cypher("CALL gds.graph.drop('mod2_centrality')")
            except ClientError:
                pass

            if (links):
                G_mod2_pr, _ = gds.graph.project(
                    'mod2_page_rank',
                    'Airport',
                    {'TRAVEL': {'properties': ['wait']}}
                )

                pr_results = gds.pageRank.stream(G = G_mod2_pr)
                print('-- PAGE RANK --')
                for node_id, score in zip(pr_results.nodeId, pr_results.score):
                    print(f'Aeropuerto: {gds.util.asNode(node_id)._properties["id"]} - Puntuación: {score}')

            if (links and centrality): print()

            if (centrality):
                G_mod2_cent, _ = gds.graph.project(
                    'mod2_centrality',
                    'Airport',
                    'TRAVEL'
                )

                pr_results = gds.beta.closeness.stream(G = G_mod2_cent)
                print('-- CENTRALITY --')
                for node_id, score in zip(pr_results.nodeId, pr_results.score):
                    print(f'Aeropuerto: {gds.util.asNode(node_id)._properties["id"]} - Puntuación: {score}')

    def close(self):
        ''' Close Neo4j instance '''
        self.driver.close()

    def __exit__(self, exc_type, exc_value, traceback):
        ''' Object cleanup '''
        self.close()

if __name__ == "__main__":
    neo4j_uri = os.getenv('NEO4J_URI', 'bolt://localhost:7687')
    neo4j_user = os.getenv('NEO4J_USER', 'neo4j')
    neo4j_password = os.getenv('NEO4J_PASSWORD', 'neo4j_nosql')

    parser = argparse.ArgumentParser()

    actions = ['fill', 'predict', 'stats']
    parser.add_argument('action', choices=actions,
            help='Acciones disponibles')
    parser.add_argument('-f', '--file',
            help='Set de datos para el llenado de la base (formato csv)', default=None)
    parser.add_argument('-t', '--top',
```

```
            help='Limitar resultados a la cota superior [top]', default=5)
    parser.add_argument('-r', '--reverse',
            help='Invertir factores de ordenamiento (Waiting avg, Connection amount)', default=False)
    parser.add_argument('-s', '--start',
            help='Fecha inicio en el formato (dd-mm-aaaa)', default=None)
    parser.add_argument('-e', '--end',
            help='Fecha final en el formato (dd-mm-aaaa)', default=None)
    parser.add_argument('-l', '--links',
            help='Ejecutar algoritmo Page Rank para vínculos entre aeropuertos', default=False)
    parser.add_argument('-c', '--centrality',
            help='Ejecutar algoritmo Closeness Centrality para inspección de nodos', default=False)

    args = parser.parse_args()

    with TravelPredictor(neo4j_uri, neo4j_user, neo4j_password) as tp:
        if args.action == 'fill':
            if (not args.file):
                print('Archivo de entrada faltante. Intenta de nuevo.')
                exit(1)

            if ('.csv' not in args.file):
                print('Formato de archivo incorrecto.')
                exit(1)

            tp.fill(args.file)
        elif args.action == 'predict':
            tp.search_range(top=args.top, reverse=args.reverse, start=args.start,end=args.end)
        elif args.action == 'stats':
            tp.stats(args.links, args.centrality)
```

Código para modelo 3 (Mongodb):

```
#!/usr/bin/env python3

''' Modelo 3: Modelo que determine picos en afluencia de pasajeros

en los aeropuertos para valorar la opción de introducir descuentos

o distintas estrategias comerciales '''

import argparse, os, sys

from datetime import datetime


from pymongo import MongoClient


class TravelPredictor(object):

    ''' Modelo 3 '''

    def __init__(self, uri:str, target_db:str):

        self._client = MongoClient(uri)

        self.db = self._client[target_db]


        months = ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio', 'Julio',

                  'Agosto', 'Septiembre','Octubre', 'Noviembre', 'Diciembre']

        self._months = {i:m for i, m in enumerate(months, 1)}


    def __enter__(self):

        ''' Context manager helper '''
```

```
        return self

    def _cleanup(self, collections:list):
        ''' Database cleanup '''
        for collection in collections:
            self.db[collection].delete_many({})

    def fill(self, source:str):
        ''' Populate database given input dataset '''
        airlines = set()
        airports = set()
        op_count = 1

        collections = ['airline', 'airport', 'travel']

        # Reset database contents
        self._cleanup(collections)

        with open(source, newline='') as csv:
            headers = csv.readline().rstrip('\r\n').split(',')
            curr_data = {}
            lines = csv.readlines()

            for line in lines:
                curr_data = {k:v for k, v in zip(headers, line.rstrip('\r\n').split(','))}

                if (curr_data['airline'] not in airlines):
                    # Airline insertion
                    self.db['airline'].insert_one({'name': curr_data['airline']})
                    airlines.add(curr_data['airline'])

                if (curr_data['from'] not in airports):
                    # Airport insertion
                    self.db['airport'].insert_one({'id': curr_data['from']})
                    airports.add(curr_data['from'])

                if (curr_data['to'] not in airports):
                    # Airport insertion
                    self.db['airport'].insert_one({'id': curr_data['to']})
                    airports.add(curr_data['to'])
```

```
            date = datetime(year=int(curr_data['year']),
                            month=int(curr_data['month']),
                            day=int(curr_data['day']))

            # Travel insertion
            self.db['travel'].insert_one({
                'airline': curr_data['airline'],
                'from': curr_data['from'],
                'to': curr_data['to'],
                'connection': True if curr_data['to'] == 'True' else False,
                'date': date,
                'reason': curr_data['reason'],
                'stay': curr_data['stay']
            })

            op_count += 1
            print(f"Progreso: {op_count/len(lines):.0%}", end='\r')
    sys.stdout.write("\033[K")
    print('Finalizado')

def _show_airports(self):
    ''' Return all airports in the database '''
    return [a['id'] for a in self.db['airport'].aggregate([{'$project': {'_id':0, 'id':'$id'}}])]

def _show_airlines(self):
    ''' Return all airlines in the database '''
    return [a['name'] for a in self.db['airline'].find({}, {'_id':0})]

def _choice_menu(self, choices:list, dec_msg:str = '', prev_msg:str = ''):
    ''' Menu display helper '''
    try:
        dec = input(f'\n{dec_msg} [S/n]: ').upper()
    except KeyboardInterrupt:
        print('\nSelección abortada')
        return None

    if (not(not dec or dec in ('S', 'Y'))):
        return None
```

```
        selection = ''
        choices = {str(i):elem for i, elem in enumerate(choices, 1)}

        while (not selection):
            print(prev_msg)
            for i, elem in choices.items():
                print(f'{i} - {elem}')

            try:
                selection = input('Selección: ')
            except KeyboardInterrupt:
                print('\nSelección abortada')
                return None

            if (selection not in choices.values()):
                if (selection in choices.keys()):
                    selection = choices[selection]
                else:
                    print('Selección incorrecta. Intenta de nuevo.\n')
                    selection = ''

        return selection

    def promotions(self, start = None, end = None):
        ''' Predict promotions based on parameters '''
        def query(airport, airline, start, end):
            ''' Actual query to be applied against database '''
            match_dict = {
                '$match':
                    {
                        'connection': {'$ne': 'true'},
                        'stay': {'$in': ['Hotel','Short-term homestay']},
                        'reason': {'$in': ['On vacation/Pleasure','Back Home']}
                    }
            }

            group_dict = {
                '$group':
                    {
                        '_id': {'$month': '$date'},
```

```
                'qty': {'$sum': 1}
            }
        }

        project_dict = {
            '$project':
                {
                    '_id': 0,
                    'month': '$_id',
                    'qty': '$qty',
                }
        }

        sort_dict = {
            '$sort': {'qty': -1}
        }

        if (airport):
            match_dict['$match']['from'] = airport

        if (airline):
            match_dict['$match']['airline'] = airline

        if (start):
            match_dict['$match']['date'] = {'$gte': datetime(start, 1, 1)}

        if (end):
            if ('date' in match_dict['$match'].keys()):
                match_dict['$match']['date']['$lt'] = datetime(end, 1, 1)
            else:
                match_dict['$match']['date'] = {'$lt': datetime(end, 1, 1)}

        return list(self.db['travel'].aggregate([
            match_dict, group_dict, project_dict, sort_dict
        ]))

    def top_month(*args):
        ''' Get month with most occurrence in given lists '''
        dict_holder = {}
        res_month = ''
```

```
            max_holder = 0

            for months in args:
                for month in months:
                    if (month not in dict_holder.keys()):
                        dict_holder[month] = 0

                    dict_holder[month] += 1

                    if (dict_holder[month] > max_holder):
                        res_month = month
                        max_holder = dict_holder[month]

            return res_month

        res_str = '\nMostrando posibles ofertas\n\tParámetros:'

        if (start):
            try:
                start = int(start)
                res_str += f'\n\t\t- Desde {start}'
            except ValueError:
                print(f'Parámetro de búsqueda incorrecto ({start=}). Revisa tus entradas.')
                exit(1)

        if (end):
            try:
                end = int(end)
                res_str += f'\n\t\t- Hasta {end}'
            except ValueError:
                print(f'Parámetro de búsqueda incorrecto ({end=}). Revisa tus entradas.')
                exit(1)

        airports = self._show_airports()
        airport = self._choice_menu(airports,
                '¿Deseas un aeropuerto en específico',
                'Selecciona un aeropuerto para realizar la predicción')

        airline = self._choice_menu(self._show_airlines(),
                '¿Deseas una aerolínea específica?',
```

```
            'Selecciona una aerolínea')

    if (airline):
        res_str += f'\n\t\t- Aerolínea "{airline}"'

    if (airport):
        query_res = query(airport, airline, start, end)

        if (not query_res):
            print('No hay recomendaciones disponibles para el conjunto de parámetros actual.')
            return

        res_str += f'\n\t\t- Aeropuerto "{airport}"'
        top = [self._months[r['month']] for r in query_res[:3]]
        bottom = [self._months[r['month']] for r in query_res[-3:]]

        print(res_str + '\n\nResultados:')
        print('\tPosibilidad de introducción de paquetes grupales o similar\n\t->', ', '.join(top))
        print('\n\tPosibilidad de descuentos por baja demanda\n\t->', ', '.join(bottom))

        if (not top == bottom):
            print(f'\nMes con mejor posibilidad de introducción de promociones: {top_month(top, bottom)}')
    else:
        if (res_str.endswith('Parámetros:')): res_str = res_str.rstrip('Parámetros:')
        print(res_str + '\n\nResultados:', end='')
        holder = []

        for airport_i in airports:
            print(f'\nAeropuerto "{airport_i}"')

            res = query(airport_i, airline, start, end)

            if (not res):
                print('\tSin recomendaciones posibles')
                continue

            top = [self._months[r['month']] for r in res[:3]]
            bottom = [self._months[r['month']] for r in res[-3:]]
            holder.append(top)
            holder.append(bottom)
```

```
                print('\tPosibilidad de introducción de paquetes grupales o similar\n\t->', ', '.join(top))
                print('\n\tPosibilidad de descuentos por baja demanda\n\t->', ', '.join(bottom))

            print(f'\nMes con mejor posibilidad de introducción de promociones: {top_month(*holder)}')

    def close(self):
        ''' Close database connection '''
        self._client.close()

    def __exit__(self, exc_type, exc_value, traceback):
        ''' Object cleanup '''
        self.close()

if __name__ == '__main__':
    mdb_uri = os.getenv('MONGODB_URI', 'mongodb://localhost:27017')
    db_name = os.getenv('MONGODB_DB_NAME', 'proj_mod3')

    parser = argparse.ArgumentParser()

    actions = ['fill', 'predict']
    parser.add_argument('action', choices=actions,
            help='Acciones disponibles')
    parser.add_argument('-f', '--file',
            help='Set de datos para el llenado de la base (formato csv)', default=None)
    parser.add_argument('-s', '--start',
            help='Año de inicio para rango de búsqueda', default=None)
    parser.add_argument('-e', '--end',
            help='Año de término para rango de búsqueda', default=None)

    args = parser.parse_args()

    with TravelPredictor(mdb_uri, db_name) as db:
        if args.action == 'fill':
            if (not args.file):
                print('Archivo de entrada faltante. Intenta de nuevo.')
                exit(1)

            if (not args.file.endswith('.csv')):
                print('Formato de archivo incorrecto.')
```

```
            exit(1)


        db.fill(args.file)
    elif args.action == 'predict':
        db.promotions(start=args.start, end=args.end)
```