**Instituto Tecnológico y de Estudios Superiores de Occidente**

**Bases de datos NO RELACIONALES**



**Proyecto integrador bases de datos no sql**

Presentan

**Miguel Mario Méndez Alcaraz, IS727312**

**Ángel David Cortes Pacheco, IS726442**

Profesor: Leobardo Ruiz Rountree

Fecha: 30/11/2023

Contenido

[Nombre del proyecto 3](#_Toc118282467)

[Introducción 3](#_Toc118282468)

[Alcance del Proyecto 3](#_Toc118282469)

[Requerimientos 3](#_Toc118282470)

[Diseño de la solución 3](#_Toc118282471)

[Modelos de datos utilizaditos 3](#_Toc118282472)

[Consultas 3](#_Toc118282473)

[Índices y Optimizaciones 3](#_Toc118282474)

[Conclusiones 3](#_Toc118282475)

[Fuentes de información 3](#_Toc118282476)

[Anexos 3](#_Toc118282477)

# Nombre del proyecto

Proyecto Integrador Bases de Datos No SQL

# Introducción

*<Problema que atiende o resuelve el proyecto>*

# Las bases de datos NoSQL, en contraste con las tradicionales bases de datos relacionales, han emergido como una alternativa flexible y escalable para abordar los desafíos de almacenamiento y procesamiento de datos en entornos contemporáneos. Entre las diversas opciones disponibles, tres destacadas son Cassandra, MongoDB y DGraph.

# Cassandra, diseñada para gestionar grandes volúmenes de datos distribuidos y altamente disponibles, pertenece a la categoría de bases de datos de tipo "column-family". Su arquitectura descentralizada y resistente a fallos la convierte en una elección popular para sistemas distribuidos a escala masiva, siendo especialmente adecuada para aplicaciones con requisitos de rendimiento y disponibilidad críticos. MongoDB, por otro lado, se clasifica como una base de datos de tipo "documento" y destaca por su capacidad para almacenar datos semi-estructurados en formato JSON BSON. Con un modelo de datos flexible y escalabilidad horizontal, MongoDB es favorecida en entornos donde la agilidad y la adaptabilidad son esenciales. DGraph, por último, se destaca en el ámbito de las bases de datos de grafos, proporcionando un modelo nativo de grafo y consultas altamente eficientes para relaciones complejas. Cada una de estas opciones ofrece características distintivas, lo que permite a los desarrolladores elegir la base de datos NoSQL más adecuada según las necesidades específicas de sus aplicaciones.

Con estas 3 tecnologías distintas, vamos a buscar combinarlas en un “sistema unificado”, es decir, utilizar las 3 en un mismo archivo para poder hacer consultas a cada una, dependiendo de la opción que el usuario elija.

# Alcance del Proyecto

El proyecto:

* Mostrará el resultado de los 3 modelos realizados, utilizando Cassandra, MongoDB y DGraph como mínimo una vez.
* Solo utilizara los datos generados por el archivo *data.py* y cualquier archivo que se genere de este.
* Se podrá realizar consulta a cualquiera de las 3 tecnologías desde un mismo archivo main.py, es decir, el usuario final no tendrá que correr diferentes archivos para acceder a los resultados de los modelos.

# Requerimientos

* El sistema deberá tener instalado Docker, con los contenedores de Cassandra, MongoDB y DGraph
* Los integrantes del proyecto deberán tener conocimiento sobre las 3 tecnologías (Cassandra, Mongo DB, Dgraph) desde la creación de una consulta hasta el manejo de los datos y el lenguaje que se use en cada uno.
* Los integrantes del proyecto deberán tener conocimiento de Python para poder hacer un código que acceda a los datos sin tener que entrar a los contenedores de Docker
* El sistema deberá tener un solo archivo main.py que se ejecutara y del cual se harán las consultas a los modelos.

# Diseño de la solución

## Modelos de datos utilizaditos

MongoDB – A diagram of a flight

Description automatically generated

Cassandra – A diagram of a flight

Description automatically generated

DGraph –A diagram of a diagram

Description automatically generated

## Consultas

MongoDB –

collection.aggregate([{"$group": {"\_id": {"month": "$month", "destination": "$destination"}, "total": { "$sum" : 1}}}, {"$match": {"total": {"$gt": 4}}}])

collection.aggregate([{"$group": {"\_id": {"month": "$month", "destination": "$destination"}, "total": { "$sum" : 1}}}])

collection.aggregate([{"$group": {"\_id": {"month": "$month", "destination": "$destination"}, "total": { "$sum" : 1}}}, {"$match": {"\_id.destination": ap}}])

Cassandra –

SELECT\_AIRPORTS\_FOOD\_SERVICE = """

SELECT destination

FROM destination\_counts

WHERE passengers > 3

ALLOW FILTERING;

"""

SELECT\_ALL\_AIRPORTS\_PASSENGERS = """

SELECT destination, COUNT(\*) as passengers

FROM flights\_data

WHERE connection = True and wait > 60 and wait < 360

GROUP BY destination

ALLOW FILTERING;

"""

SELECT\_PASSENGERS\_PER\_AIRPORT = """

SELECT destination, COUNT(\*) as passengers

FROM flights\_data

WHERE connection = True and wait > 60 and wait < 360 and destination = ?

GROUP BY destination

ALLOW FILTERING;

"""

DGraph –

def buscarAirline(client, name):

    query = """query search\_airline($a: string) {

        all(func: has(age)) {

            flight @filter(eq(airline, $a) AND (eq(month,"1") OR eq(month,"12"))){

                count(airline)

                airline

                origin

                destination

                month

                }

        }

    }"""

    variables = {'$a': name}

    res = client.txn(read\_only=True).query(query, variables=variables)

    ppl = json.loads(res.json)

    # *Print results.*

    print(f"Number of people in the airline {name}: {len(ppl['all'])}")

    print(f"Data associated with {name}:\n{json.dumps(ppl, indent=2)}")

def buscarTodos(client):

    query = """{

        all(func: has(age)) {

            flight @filter(eq(month, 12) OR eq(month, 1)) {

                count(airline)

                airline

                origin

                destination

                month

                }

            }

        }"""

    res = client.txn(read\_only=True).query(query)

    ppl = json.loads(res.json)

    # *Print results.*

    print(f"Data associated with:\n{json.dumps(ppl, indent=2)}")

## Índices y Optimizaciones

MongoDB – No utilizamos índices, ya que el dataset era muy pequeño, pero para las consultas que usamos se podrían indexar tanto destino como mes para agilizar las consultas ya que son nuestra principales comparaciones y llaves que se usan para realizar las consultas

Cassandra – Primary Key: Destination, Connection, Wait Clustering Keys: Connection, wait. Se utilizaron estas llaves para poder ejecutar las consultas necesarias, ya que para estas necesitábamos distinguir los vuelos que eran conexión y que el tiempo de espera fuera entre 1 y 6 horas.

DGraph –age: int @index(int) . airline: string @index(hash) . origin: string @index(exact) . flight: [uid] @reverse . Estas fueron las variables que utilizamos como índices y en su caso flight como reverse. El motivo de esto fue porque estas fueran las variables que utilizamos para hacer las querys.

# Conclusiones

Miguel: Para concluir con este proyecto, y de cierta manera con la clase, puedo decir que gracias a este proyecto agregue y recupere lo visto a lo largo del semestre en los tres motores de bases de datos (Cassandra, MongoDB, GraphQL). Esto debido a que a la hora de realizar las soluciones no solo aplicamos los conocimientos previos también debido al diseño que hicimos de tener todas las bases de datos corriendo con un solo main.py tuvimos que buscar soluciones distintas. Algunos de los problemas que se presentaron fueron que no se creaba la base de datos correctamente, como en Cassandra que no se definían bien las Clustering Keys y entonces no dejaba realizar las consultas de manera correcta. Otro importante que tuvimos que resolver fue en Mongo, ya que la manera que vimos en la clase fue usando uvicorn para correr un sistema donde pudiéramos realizar las consultas, el problema con esto es que tenemos que correr también los otros dos motores, por lo que tuvimos que buscar otra solución (pymongo)

En general, sirvió como un resumen de la clase y una nueva manera de ver las bases de datos. Estuvo interesante dejar de lado SQL y tener una nueva perspectiva que no utilice tablas relacionales como tal. Considero que aunque indagamos en los 3 motores, para poder conseguir el nivel en las industrias, tocaría seguir estudiando y mejorando en la base de datos de nuestra preferencia.

Angel: Me gusta la escalabilidad horizontal y el rendimiento de lectura/escritura distribuida de Cassandra, lo que la hace adecuada para aplicaciones que requieren una expansión fácil a medida que crece la carga de trabajo.

MongoDB me atrajo por su flexibilidad de esquema, lo que facilita la adaptación a cambios en los requisitos. Además, su eficiencia en operaciones de lectura es una ventaja, aunque puede requerir escalabilidad vertical.

Dgraph, con su modelo de datos de grafo y consulta GraphQL, proporciona una representación más natural de relaciones complejas. Aunque de los tres fue el que mas se me complico y el que menos me gusto. No porque este difícil, si no porque la poca documentación que hay no se me hace completa.

# 

# Fuentes de información

1. PyMongo (2023). *Mongo\_client – Tools for connecting to MongoDB.* Recuperado de: <https://pymongo.readthedocs.io/en/stable/api/pymongo/mongo_client.html>

# Anexos

*<Información adicional que no clasifica en ninguno de los puntos anteriores>*