Advanced Databases
Project Report: Group 21

Ana Moreira 54514
Benjamim Ventura 61032
Miguel Duarte 54941
Tomás Fonseca 62908

Introduction

This project was made with the goal to compare two types of databases, namely MySQL and NoSQL databases. In order to do this, we evaluated the performance of both types of databases through different data modeling structures, queries and varied optimization measures.

About the dataset

The dataset chosen by our group consists of 32 columns and 119391 lines, about reservations made in city and resort hotels from 2015 to 2017. This dataset contains extensive information about the reservations from the number of adults to if there were any requirements about parking spaces. For this project we split this dataset into 3 separate files, “connecting” them through a column we created named “id_reservation” (this column is unique in the Table/Document).
Link to the dataset:
https://www.kaggle.com/datasets/jessemostipak/hotel-booking-demand/data

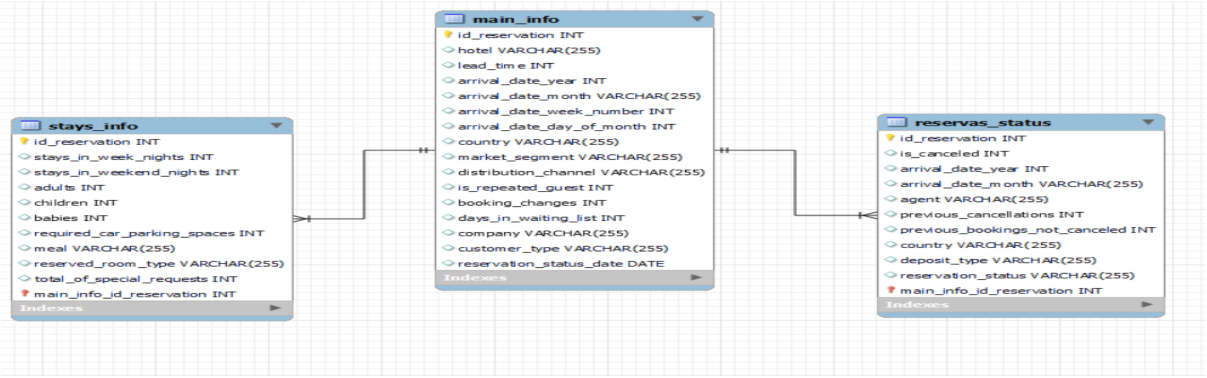
How did we divide it

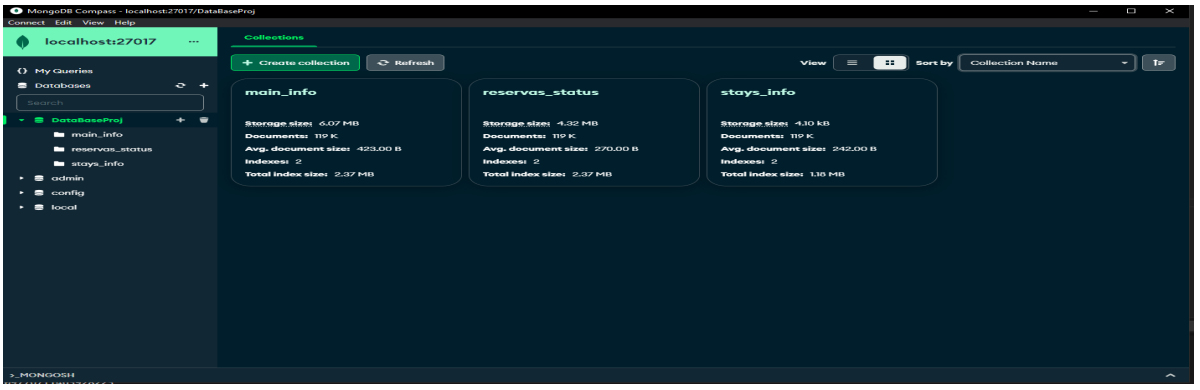
Name: “main_Info”:
Selected columns:
hotel, lead_time, arrival_date_year, arrival_date_month, arrival_date_week_number, arrival_date_day_of_month, country, market_segment, distribution_channel, is_repeated_guest, booking_changes, days_in_waiting_list, company, customer_type, reservation_status_date

Name: “reservas_status”
Selected columns:
is_cancelled, arrival_date_year, arrival_date_month, agent, previous_cancellations, previous_bookings_not_cancelled, country, deposit_type, reservation_status

Name: “stays_info”
Selected columns:
stays_in_week_nights, stays_in_weekends_nights, adult, children, babies, required_car_parking_spaces, meal, reserved_room_type, total_of_special_requests

Scheme for both databases (Mysql and MongoDB)





Discussion of point done/not done in the project

Besides deep query analysis and improvement of query performance (changing query structure), every point of the proposed problem was addressed and implemented.

Creation of the databases, and running the queries

```
import mysql.connector
import pandas as pd
from sqlalchemy import create_engine, text
from sqlalchemy.types import Integer, Float, String

# Creates the database
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password = '1234'
  )
mycursor = mydb.cursor()
try:
mycursor.execute('DROP DATABASE IF EXISTS hotel_bookings')
except:
    pass
mycursor.execute("CREATE DATABASE hotel_bookings")
mydb.commit()
print("Database 'hotel_bookings' created successfully!")
# Close the connection to the database
mydb.close()
```

```
Database 'hotel_bookings' created successfully!
```

Creating tables

```
mycursor = mydb.cursor()
mycursor.execute("USE hotel_bookings")
# Create the 'main_info' table
mycursor.execute("""
CREATE TABLE main_info (
    id_reservation INT AUTO_INCREMENT PRIMARY KEY,
    hotel VARCHAR(255),lead_time INT,arrival_date_year INT,arrival_date_month VARCHAR(255),
    arrival_date_week_number INT,arrival_date_day_of_month INT,country VARCHAR(255),
    market_segment VARCHAR(255), distribution_channel VARCHAR(255), is_repeated_guest INT,
    booking_changes INT, days_in_waiting_list INT,company VARCHAR(255),
    customer_type VARCHAR(255),reservation_status_date DATE
);""")
# Create the 'reservas_status' table
mycursor.execute("""
CREATE TABLE reservas_status (
    id_reservation INT, is_canceled INT,
    arrival_date_year INT,arrival_date_month VARCHAR(255),
    agent VARCHAR(255),previous_cancellations INT,
    previous_bookings_not_canceled INT,country VARCHAR(255),
    deposit_type VARCHAR(255),reservation_status VARCHAR(255),
    PRIMARY KEY (id_reservation),
    FOREIGN KEY (id_reservation) REFERENCES Main_Info(id_reservation)
);""")
# Create the 'stays_info' table
mycursor.execute("""
CREATE TABLE stays_info (id_reservation INT,stays_in_week_nights INT,stays_in_weekend_nights INT,adults INT,
children INT,babies INT,required_car_parking_spaces INT,meal VARCHAR(255),
reserved_room_type VARCHAR(255),total_of_special_requests INT,
PRIMARY KEY (id_reservation),FOREIGN KEY (id_reservation) REFERENCES Main_Info(id_reservation)
);""")
# Commit the changes to the database
mydb.commit()
print("Database 'Table main_rnfo, reservas_status, stays_info ' created successfully!")
# Close the connection to the database
mydb.close()
```

Inserting data into the house_bookings mysql database using csv file

```python
#Insert de dados no mysql usando o ficheiro csv
import pandas as pd
import time
from sqlalchemy import create_engine
# Carrega os dados CSV em DataFrames
df1 = pd.read_csv('Tabela1.csv')
df2 = pd.read_csv('Tabela2.csv')
df3 = pd.read_csv('Tabela3.csv')

# Estabeleça uma conexão com o banco de dados MySQL usando SQLAlchemy
engine = create_engine('mysql+mysqlconnector://root:cussueca2018@localhost/hotel_bookings')

# Medir o tempo de execução
start_time = time.time()
# Inserir dados na base de dados
try:
    with engine.begin() as connection:
        df1.to_sql('main_info', con=connection, if_exists='append', index=False)
        df2.to_sql('reservas_status', con=connection, if_exists='append', index=False)
        df3.to_sql('stays_info', con=connection, if_exists='append', index=False)

    # if_exists: Este parâmetro determina o que fazer se a tabela já existir no banco de dados.
    # index=False: Não inclui a coluna de índices do DataFrame como uma coluna
    # 'append': Adiciona os novos dados à tabela existente.

    end_time = time.time()
    execution_time = end_time - start_time

    print(f"Dados inseridos com exito. Tempo de execução: {execution_time:.2f} segundos.")
# Exibir o erro caso a inserção falhar
except Exception as e:
    print(f"Error: {e}")
```

How to run our project

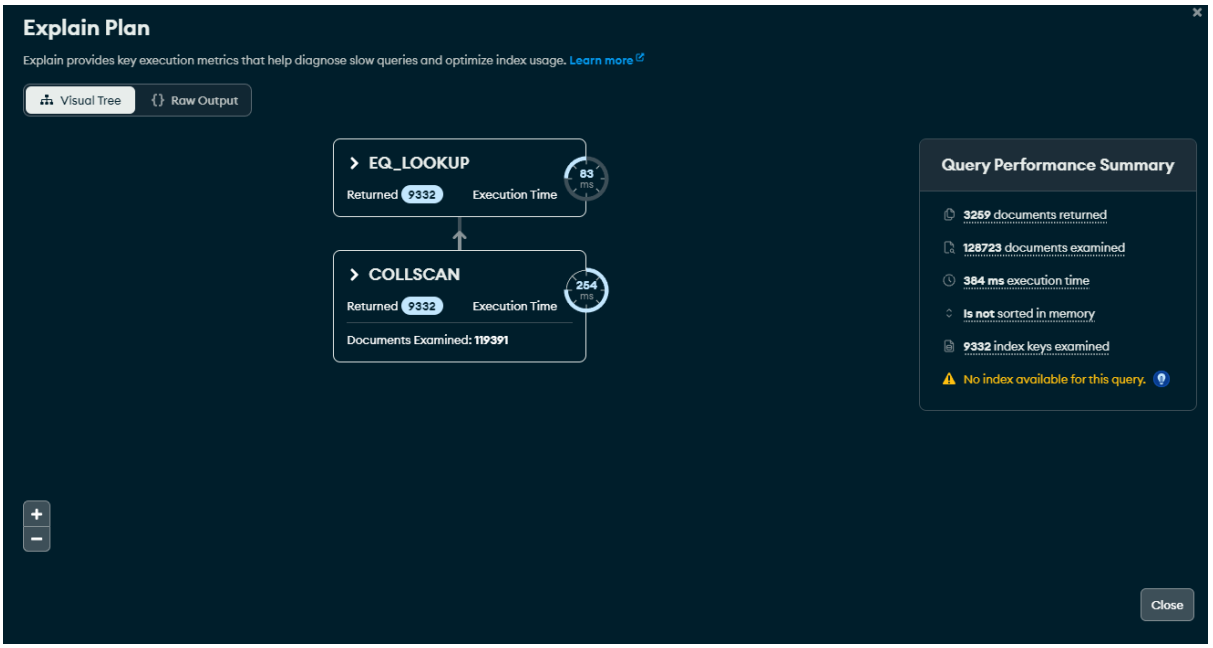
In order to replicate our project, first the script.py file should be executed (file: “hotel_bookings.csv” needs to be in the same directory as script.py). This will create every file needed to create the databases and its tables/documents, i.e the .csv and .json files that were used in our project. Next, both the mysql and mongodb local servers should be started. After that, the mongoDB.py and mysqlDB.py files can be executed, keeping in mind to change the mysql user password. In these files, we first created the three collections in mongodb or tables in mysql with the data from the created files, called “main_info”, “reservas_status” and “stays_info”.

To test the performance of each of the databases, we implemented different queries that allowed us to answer several questions that are shown below.

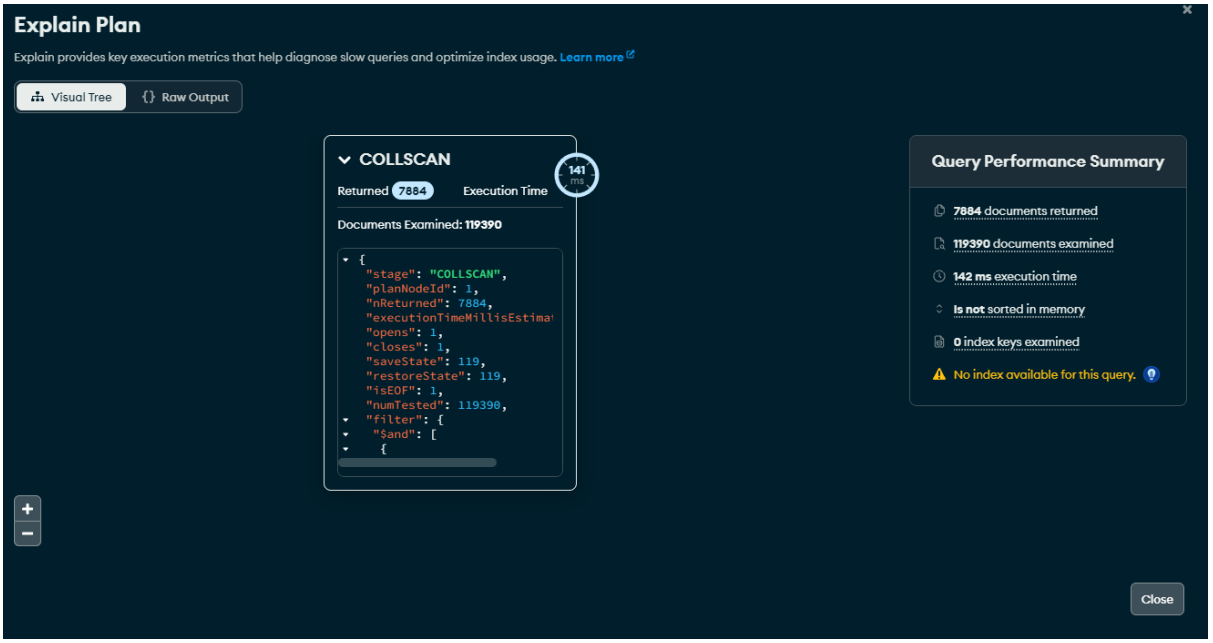
Simple Query 1 - What are the countries that had reserves changed by more than 90, and in which year did this occur?

Simple Query 2 - What are the countries that had a number of reserves changed by more than 500, and in which year did this occur?

Complex Query 1 - How many families with kids canceled their reservations



Complex Query 2 - How many families with kids have parking spaces or meals at the hotel.



After writing all the queries in mongodb and mysql, we tested them and obtained the time it took to run them in both databases.

To improve the performance of the queries, we introduced indexes to the collections created in mongodb. This produced a significant improvement in the speed of the queries (more noticeable in the complex ones).

Updating a record in the Main_info table

```
novo_hotel = "City Hotel"
start_time = time.time()
sql_update = text(f""" UPDATE main_info SET hotel = :novo_hotel WHERE hotel = "7 Hotel" """)
with engine.begin() as connection:
connection.execute(sql_update, {"novo_hotel": novo_hotel})
```

Insertion of new records in the main_info, reservation status, stay info tables

```
    #Inserção de dados na tabela Main
    data = {'hotel': ['BDA Hotel'],'lead_time': [30],'arrival_date_year': [2023],'arrival_date_month': ['December'],'arrival_date_week_number': [48],
        'arrival_date_day_of_month': ['1'],'country': ['PRT'],'market_segment': ['Direct'],'distribution_channel': ['Direct'],'is_repeated_guest': [0],
        'booking_changes': [2],'days_in_waiting_list': [0],'company': ['BDA'],'customer_type': ['Transient'],'reservation_status_date': ['2023-12-01']
    }
    df = pd.DataFrame(data)
```

```
#Inserção de dados nas tabelas com chave estrangeira
data1 = {
    'id_reservation': [119399],'is_canceled': [0],'arrival_date_year': [2023],'arrival_date_month': ['December'],
    'agent': [48],'previous_cancellations': [1],'previous_bookings_not_canceled': [0],'country': ['PRT'],
    'deposit_type': ['No Deposit'],'reservation_status': ['Check-Out']
}
data2 = {
    'id_reservation': [119399], 'stays_in_week_nights': [3],'stays_in_weekend_nights': [2],'adults': [2],
    'children': [1],'babies': [1],'required_car_parking_spaces': [1],'meal': ['BB'],'reserved_room_type': ['B'],
    'total_of_special_requests': [2]
}
# Inserção de dados nas tabelas 'reservas_status' e 'stays_info' usando SQLAlchemy
df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)
```

Runtime description

In both mysqlDB.py and mongoDB.py, at the beginning of the runtime, we delete the databases if they exist in order to allow multiple runs without the need to delete inserts and updates. In mongoDB.py all the six queries are made without indexes and in the end of the file both complex queries are executed again with indexes created in order to see the drastic increase in performance.

Conclusion

In our group’s opinion, we can conclude that there was no need to separate the original dataset into 3 separate tables, since this made the queries take longer due to joins and lookups. This change was only made due to the project requirements, although the original dataset contained all the information in only one single table/document. This type of query operations in MySQL were especially harder than in mongoDB, because of the primary keys there was the need to always do join for all the three tables in every complex query.