*Predicting Hotel Booking Cancellations*

Zeynep Demirci   
*Department of Applied Science Data Science  
San Jose State   
University*Istanbul, Turkey  
zeynep.demirci@sjsu.edu

Changgyun Kim  
*Department of Applied Science Data Science  
San Jose State   
University*Mountain View, USA  
changgyun.kim@sjsu.edu

Kranthi Raj Vellanki  
*Department of Applied Science Data Science  
San Jose State   
University*Berkeley, USA  
kranthiraj.vellanki@sjsu.edu

Huan Nguyen  
*Department of Applied Science Data Science*  
*San Jose State   
University*Ho Chi Minh, Vietnam  
huan.t.nguyen@sjsu.edu

**Abstract - With the freedom of using both NoSQL and SQL, we still keep the same objective of improving the hotel management process along with customer service, with data analysis. However, in this project, we will shift our focus toward choosing the right technology (NoSQL or SQL) for each table from the two previous projects. By working as a team and discussing the tradeoffs and figuring out what is the best outcome, we build our database depending on what is most convincing to each of the team members.**

***Keywords— NoSQL, SQL, cancellations, data, reservations, hotels***

# Introduction

With the popularity of NoSQL rising over recent years, the question of whether NoSQL or SQL is better has been a hot debate topic among database experts. However, the more fascinating question to ask is why not utilize both and utilize both of their benefits. For beginners who have very minimal exposure to the world of databases like us, the concept of using both SQL and NoSQL sounds really confusing at first. Throughout the project, we understood that it happens quite often nowadays, with companies using SQL and NoSQL together for different reasons and tasks. In general, it really depends a lot on some basic criteria for us to make the decision. For example, what does the majority structure of your database look like (e.g: structured or unstructured), do you want your database schema to be flexible or not, horizontal versus vertical scaling ability, etc. In order to make the ultimate choice, we, as data scientists, must first understand the pros and cons of both SQL and NoSQL.

# Requirements

- The system shall record the expected check-out date and time.

- The system must check in and check out the customer.

- The system allows you to modify a reservation without re-entering customer information.

- The system should send a reservation confirmation email to the customer.

- The system will charge an additional night's charge if the customer checks out after 11 am.

- The system must record customer feedback.

- The system records payments and payment types for meals.

- The system accepts reservations for restaurants and room service.

- The system will be appropriate to use both NoSQL and SQL databases.

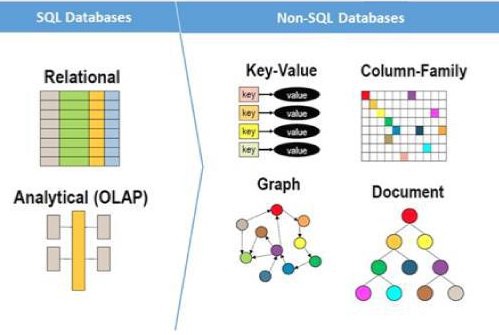
# Problem Statement

It is obvious that NoSQL databases have a rise in their popularity. Features of NoSQL like integrating big data, lowering cost, providing easier scalability make it an appealing option for companies. Even so, NoSQL happens to be a relatively young technology without the set of standards SQL databases like MySQL offers. Actually, both SQL and NoSQL have some advantages and disadvantages over the others. Mainly complex business needs and the volume and variety of data it consumes will dictate the choice between SQL and NoSQL. This project aims to explore the best way of database design for the chosen dataset using both SQL and NoSQL. We made some discussions among group members about our dataset and our business requirements to decide if we need to choose SQL or NoSQL database as our primary database or if we need both to meet these needs and we concluded to use both databases for our design.

# Tradeoffs

So what we have learned after doing research on how to choose between SQL and NoSQL for setting up databases is that it does happen, more often than we thought. We discussed and thought that combining the two techniques would be a hassle, hence not many companies would do it. However, in reality, a few companies do actually rely on both NoSQL and SQL to take advantage of both have to offer for different tasks. There are still many situations where highly structured SQL is preferable over NoSQL and vice versa.

As we learn in class, these are the types of NoSQL and SQL:



*Fig. 1. The different types of SQL databases and NoSQL databases.*

Some of the basic advantages of SQL would be:

* Create documents without carefully defining their structure upfront
* Add fields to your database without changing the fields of existing documents
* Store documents that have their own unique structure
* Have multiple databases with different structures and syntax

Some of the basic advantages of NoSQL would be:

* Basic Availability

This means that while the database guarantees the availability of the data, the database may fail to obtain the requested data, or the data may be in a changing or inconsistent state.

* Soft state

The state of the database can be changing over time.

* Eventual consistency

The database will eventually become consistent, and data will propagate everywhere at some point in the future.

|  | **SQL databases** | **NoSQL databases** |
| --- | --- | --- |
| **Data structure** | The SQL data structure is based on a relational model that normalizes data across strictly defined tables and standardizes the relationships between those tables, making SQL databases well suited to highly-structured data. | The NoSQL data structure does not require a normalized configuration or adhere to a relational model but is instead flexible enough to accommodate different models, including key-value, document, column-oriented, and graph. |
| **Language** | SQL databases are all about the SQL language. Some relational database products support pure SQL, but many include enhanced versions of the language—such as SQL Server’s Transact-SQL (T-SQL)—to accommodate product-specific features. However, all SQL databases support the core ANSI/ISO language elements. | NoSQL databases are not locked into one language. The language used depends on the type of NoSQL database, the individual implementation, and the specific operation. For example, MongoDB stores all documents in a JSON format, with queries based on the JavaScript programming language. |
| **Schemas** | An SQL database requires a predefined schema that determines how tables are configured and data is stored, resulting in a rigid structure that helps to optimize storage and ensure data integrity, but limits flexibility. | A NoSQL database uses a dynamic schema that requires no predefined data structure, resulting in a high degree of flexibility, such as being able to add documents with different fields to the same database. |
| **Data integrity** | SQL databases deliver a high degree of data integrity, adhering to the principles of atomicity, consistency, isolation, and durability (ACID), which are essential when supporting workloads such as financial transactions. | It can be difficult for NoSQL databases to deliver the same level of data integrity as SQL databases, with most adhering to BASE principles (basic availability, soft state, and eventual consistency), which means data in a distributed environment might be temporarily inconsistent. |
| **Scalability** | SQL databases primarily scale vertically, which means they can be easily scaled up by adding resources such as CPUs or memory, but SQL databases are not very efficient at scaling horizontally, making them ill-suited for large, distributed data sets. | NoSQL databases can scale horizontally very efficiently across systems and locations, making it possible to accommodate large stores of distributed data, while supporting increased levels of traffic. |
| **Querying** | SQL databases are efficient at processing queries and joining data across tables, making it easier to perform complex queries against structured data, including ad hoc requests. | NoSQL databases lack consistency across products and typically require more work to query data, particularly as query complexity increases. |
| **Maturity** | SQL databases are built on mature technologies that are well known and supported by large developer communities. | NoSQL products are not as mature and the technologies not as well supported as SQL products, but NoSQL technologies are making fast inroads into the industry, with developer communities constantly growing. |

*Table. 1. These are the main differences between SQL and NoSQL on data structure, language, schemas, data integrity, scalability, querying, and maturity. (Sheldon)*

# Implementation and Justification

Deciding which database is the best option for which table is really important. Actually, this is the first step of our project 3. When we discussed among group members, we ended up with the idea that we need to decide this according to what our data looks like, how we will query it, and our scalability needs.

In order to meet up these requirements, we ended up choosing MySQL as our main database and putting one of our tables which is customer history to MongoDB. We made the SQL vs. NoSQL decision based primarily on what the data looks like since the structure of the data is the most important factor in deciding whether to use a SQL or NoSQL database. This is because, if the data is primarily structured, a SQL database is likely the right choice. Since our data is transactional data and we do not expect to change the structure of our data frequently, we designed our database as relying mostly on SQL database that provides great benefits for transactional data.

On the other hand, using MongoDB for customer history data is better since we wanted to design this data as a document to keep information about previous bookings of the customers to use it for loyalty programs, etc. Also, this table does not need to join operations with other parts of the data. So, we decided that it is better to keep customer history data in MongoDB.

The other important factor to consider is how often we will query our data, how quickly we need to run queries, and who will be responsible for running these queries. Because our data is nicely structured and organized, it is very efficient to query our data with a SQL database.

On the other hand, when NoSQL database technology was being built, developers focused on scalability and flexibility, not query efficiency. It is known that NoSQL databases are appropriate for horizontal scaling but SQL databases can only be scaled vertically. This means that it is necessary to increase the capacity of a single server to scale the database which makes it not easy to scale. Scalability becomes important if the dataset will grow in the future. When we think about our case in terms of scalability we again prefer SQL databases since we do not expect a huge increase in the amount of data in the future.

When we look at our final design closely;

MySQL contains 5 different tables which have connections to each other. In this design, when there is a new booking, and for example, if it has a meal request, it will be easy to update the meal table since the meal table and booking table are in the same database.

**1- Mea**l: It contains meal details of the booking.

Categories are presented in standard hospitality meal packages:

Undefined/SC – no meal package;

BB – Bed & Breakfast;

HB – Half board (breakfast and one other meal – usually dinner);

FB – Full board (breakfast, lunch, and dinner).

**2- Customer Type**: It contains the type of booking and also whether it belongs to adults, children or babies.

Assuming customer type belongs to one of four categories:

Contract - when the booking has an allotment or other type of contract associated with it;

Group – when the booking is associated with a group;

Transient – when the booking is not part of a group or contract and is not associated with other transient bookings;

Transient-party – when the booking is transient but is associated with at least other transient bookings.

**3- Booking**: This table contains most of the information which relates to booking such as reserved room type, reservation status, required car parking, country, and so on.

**4- Hotel**: It contains hotel information as H1 stands for Resort Hotel or H2 stands for City Hotel.

**5- Room Traffic**: It includes details about when and how long the guest stayed at the hotel, and whether it is on the weekdays or on the weekends.

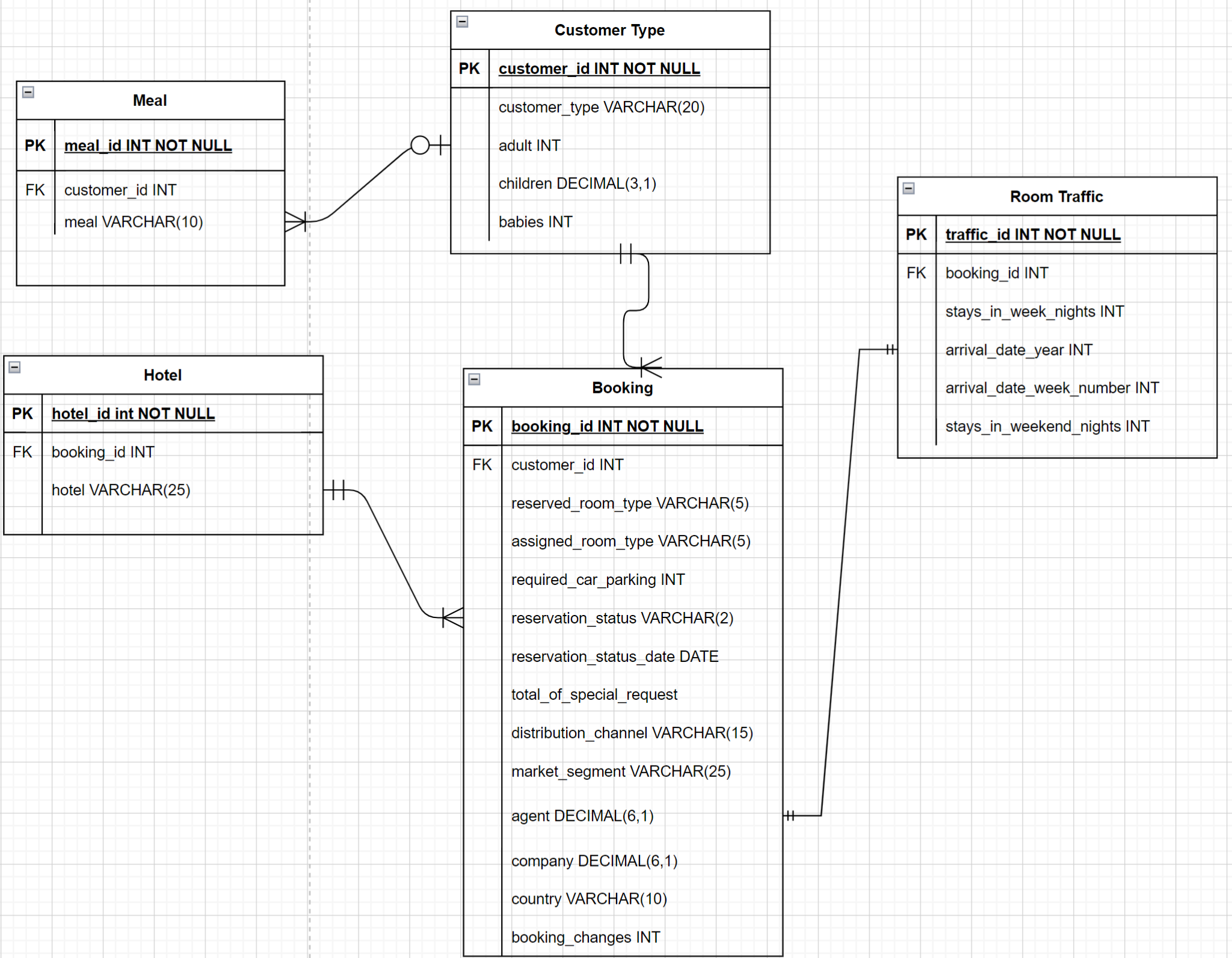
However, for the customer history table, we prefer to use MongoDB as a document to keep all necessary previous information of the customers and use this information to make an analysis on it. Actually, this model is designed to use previous behavior of the customer such as customer relationships management and loyalty programs, etc.

Our customer history table differs from the previous table which we used in Project 1, that’s because it is stored in a different database now, it includes some additional necessary information of the customer inside it.

# Final Database Design

The data came as a single CSV file which contains 32 columns in one place. After checking out the data values from the CSV file in python, we noticed that there were only two main data types which were integer and string. As mentioned above,

* These are the structure of the tables using MySQL



*Fig. 2. The ER diagram for SQL database.*

0.**hotel**: Either “Resort Hotel” or “City Hotel” only. **VARCHAR(25)**

1.**arrival\_date\_year**: The year of arrival date. **INT**

2.**arrival\_date\_week\_numbe**r: The week number for the arrival date. **INT**

3.**arrival\_date\_day\_of\_month**: The day of arrival. **INT**

4.**stays\_in\_weekend\_nights**: Number of weekend nights stayed. **INT**

5.**stays\_in\_week\_nights**: Number of weekday nights stayed or booked. **INT**

6.**adults**: Number of adults. **INT**

7.**children**: Number of children. **DECIMAL(3, 1)**

8.**babies**: Number of babies. **INT**

9.**meal**: Type of meal booked. Undefined/SC – no meal. **VARCHAR(10)**

10.**country**: Represented in the ISO 3155–3:2013 format. **VARCHAR(25)**

11.**market\_segment**: The term “TA” is“Travel Agents” and “TO” is “Tour Operators”. **VARCHAR(25)**

12.**distribution\_channel**: The term “TA” is “Travel Agents” and “TO” is “Tour Operators”. **VARCHAR(10)**

13.**reserved\_room\_type**: Code of room type reserved for security. **VARCHAR(15)**

14.**assigned\_room\_type**: Code for the type of room assigned to the booking. **VARCHAR(15)**

15.**booking\_changes**: Number of changes made to the booking from the moment the booking. **INT**

16.**agent ID**: Id of the travel agency made the booking. **INT**

17.**company ID**: Id of the company that made the booking. **INT**

18.**customer\_type**: Type of booking, four categories: Transient, Transient-Party, Contract, Group. **VARCHAR(20)**

19.**ADR**: Average Daily Rate. **DECIMAL(6, 2)**

20.**required\_car\_parking\_spaces**: Number of car parking spaces required. **INT**

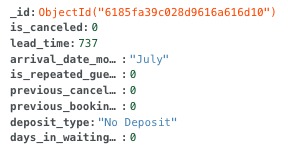
21. **total\_of\_special\_requests**: Number of special requests made. **INT**

22.**reservation\_status**: Three categories: Canceled, Check-Out. **VARCHAR(15)**

23.**reservation\_status\_date**: Date at which the last status was set. **DATE**

24.**total\_night**: Number of total nights stayed or booked (‘stays\_in\_week\_nights' + 'stays\_in\_weekend\_nights') . **INT**

* This is the collection using NoSQL



*Fig. 3. The collection for customer history in NoSQL.*

0.**\_id:** uniquely defined primary key field of the collection.

1.**is\_canceled**: If canceled (1) or not (0). **INT**

2.**lead\_time**: Days between the entering date of the booking and the arrival date. **INT**

3.**arrival\_date\_month**: The month of arrival date. **VARCHAR(15)**

4.i**s\_repeated\_guest**: repeated guest (1) or not (0). **INT**

5.**previous\_cancellations**: Number of previous bookings that were canceled. **INT**

6.**previous\_bookings\_not\_canceled**: Number of previous bookings not canceled. **INT**

7.**deposit\_type**: Whether deposit was made or not. **VARCHAR(15)**

8..**days\_in\_waiting\_list**: Number of days the booking was on the waiting list. **INT**

# SQL and NOSQL Performance Measurement

1. SQL Performance Measurement

To measure the performance of SQL query statements, we first imported the hotel booking data set. It took over an hour to upload the dataset because of very large data with over 100,000 rows. On the other hand, the query we wrote to analyze hotel reservation cancellation was very effective. These queries could be executed in less than .09 seconds. Additionally, we could also know the fetch time of the data, which means that it was instantaneous and the data was well-structured.

Furthermore, since MYSQL provides a measurement tool built into the program itself, this can be measured with accuracy. (figure 28)



*Fig. 4. This figure shows MySQL returning the duration and fetch time (SQL performance measurement) of some of the queries we ran for this project*

# NoSQL Performance Measurement

We tried various methods to measure the performance between SQL and NOSQL. To measure the performance of query statements in NOSQL, we did it using ".explain("executionstats")". The result was that NoSQL code was running faster than MySQL code. This command provides detailed performance statistics information on the selected query plan, and we can easily obtain the performance measurement result for the NOSQL query statement. The figures below show the performance measurement results for the NOSQL query statement.

To measure query performance, we looked up who canceled hotel reservations among people whose customer type was Transient-Party by using explain('executionState'). Also, in order to obtain more performance measurement results, we searched for those who canceled hotel reservations among those whose arrival month was June.

db.hotel.find({"is\_canceled":1,"customer\_type":"Transient-Party"}).explain("executionStats")



*Fig. 5. execution time for finding the customers who canceled their reservation as a Transient-Party.*

*db.hotel.find({"is\_canceled":1,"arrival\_month\_date":"June"}).explain("executionStats")*



*Fig. 6. execution time for finding the customers who canceled in the month of June.*

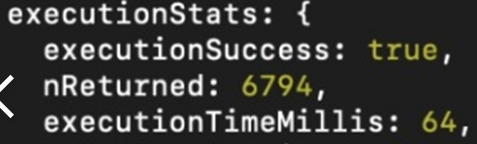
1. SQL & NoSQL Performance Measurement For Project 3

We used SQL & NoSQL to create table structures that we think are important and then measure the performance of the query again.

**Query 1**

Query 1 shown below is a query to find people who arrive in November. By using the .explain("executionStats") statement, we were able to get 0.064s performance measurement result in executing query statements in MongoDB and 0.073s in MYSQL. These result values can confirm that the performance in MongoDB is better than the performance in MYSQL.

*db.hotel.find({"arrival\_date\_month":”November”}).explain("executionStats")*



*Fig. 7. execution time for finding the customers who arrived in the month of November in MongoDB.*

*SELECT COUNT(is\_canceled) FROM hotel\_bookings WHERE arrival\_date\_month = ‘November’ LIMIT 0,1000*

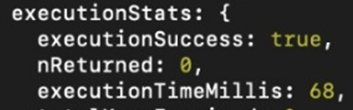


*Fig. 8. execution time for finding the customers who canceled in the month of November in MongoDB.*

**Query 2**

Query 2 shown below is a query to find people who canceled hotel reservations among those whose arrival month is June. By using the .explain("executionStats") statement to get more performance measurement result data, we got 0.068s performance measurement result in MongoDB and 0.336s in MYSQL.

*db.hotel.find({"is\_canceled":1,"arrival\_month\_date":"June"}).explain("executionStats")*

**

*Fig. 9. execution time for finding the customers who canceled in the month of June in MongoDB.*

*SELECT COUNT(b.is\_canceled), COUNT(b.is\_canceled)/10939 AS ‘Percentage’ FROM room\_traffic r, booking b WHERE b.booking\_id = r.booking\_id AND r.arrival\_date\_month = ‘Jane’ AND b.is\_canceled = 1;*

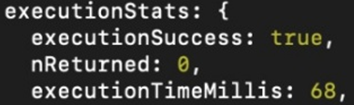
**

*Fig. 10. execution time for finding the customers who canceled in the month of June in MYSQL.*

**Query 3**

Query 3 shown below is a query to find people whose customer type is Transient-Party. We used the .explain("executionStats") statement to get performance measurement result data in the same way, and we got 0.068s performance measurement result in MongoDB and 0.089s in MYSQL. These results show that performance in MongoDB is better than in MYSQL.

*db.hotel.find({"customer\_type":”Transient-Party”}).explain("executionStats")*

**

*Fig. 11. execution time for finding the customers who canceled their reservation as a Transient-Party in MongoDB.*

*SELECT COUNT(customer\_type) FROM hotel\_bookings WHERE customer\_type = ”Transient-Party”;*

**

*Fig 12 execution time for finding the customers who canceled their reservation as a Transient-Party in MYSQL.*

| DB | *Query1 speed* | *Query2 speed* | *Query3 speed* |
| --- | --- | --- | --- |
| *NOSQL* | *0.064 s* | *0.068 s* | *0.068 s* |
| *SQL* | *0.073 s* | *0.336 s* | *0.089 s* |

*Table. 2. Comparing SQL and NoSQL performance Measurement result for queries 1, 2, and 3 in projects 1 and 2*

1. Compare the results of Project 1, 2, 3

Comparing projects 1, 2, and 3, we were able to obtain faster performance measurements for most of the same query statements that we consider important in SQL DB.

After finishing projects 1 and 2, we discussed whether to use SQL or NoSQL and decided to use SQL. We also found that there were some unnecessary relationships in our dataset, so we decided to split up the tables and remove the history table from the original data set to get more clean results we wanted.

In project 3, we split a table structure that we thought was important and got better values than the previous project performance measurement results.

| DB | *Query1 speed* | *Query2 speed* | *Query3 speed* |
| --- | --- | --- | --- |
| *Project1* |  | *0.080 s* | *0.075 s* |
| *project2* |  | *0.093 s* | *0.092 s* |
| *project3*  *(SQL)* | *0.073 s* | *0.336 s* | *0.089 s* |
| *project3 (NoSQL)* | *0.064 s* | *0.068 s* | *0.068 s* |

*Table 3. Comparing SQL and NoSQL performance Measurement results for queries 1,2 and 3 in projects 1, 2 and 3*

# Lessons Learned

Through this project, we have learned many things about databases and the use cases in which you can divide up the databases. We figured that MongoDB would be best if you want a scalable database, which is why we decided it would be best to host data about previous customers in MongoDB as it would be a growing database. On the other hand, we have our MySQL database which is best if you would like a more structured database, and we decided that this would be best to host our data about future reservations. This database will be growing and shrinking whereas the table we will be using for MongoDB will only be growing. Furthermore, we learned how the table structure can help to query more efficiently. When comparing our results from projects 1 and 2 we noticed that although MongoDB was slightly more efficient they were pretty much the same. Now that we have re-organized the structures of our tables, we have noticed that the queries we have running in MongoDB are executing significantly faster when compared to our project 2. Additionally, it is also important to note that the data in MySQL is also running faster as the relationships are closer making the queries execute faster. Overall, this project has taught us the importance of creating good ER-Diagrams based on the necessities of your requirements. A small change can result in a big difference in the querying of data and its efficiency. Our group has done an excellent job at figuring out how to divide up the data tables, and which database system to use for maximum efficiency.

| **SQL Use-Case** | **NoSQL Use-Case** |
| --- | --- |
| Data is highly structured and that structure does not change frequently | Working with a large amount of unstructured or semi-structured data |
| Support transaction-oriented | Require flexible or dynamic schema |
| The high degree of data integrity and security | Requires scaling horizontally |
| Need to perform complex queries | Want to streamline development |

*Table. 4. Summary of what we have learned how to choose SQL or NoSQL*

1. Conclusion

In conclusion, through our project, we have been able to figure out what the most efficient way to divide up our database is. When compared to previous versions of our project we have been able to execute our queries at a much faster rate, allowing commands to be executed faster resulting in more efficiency. We were able to accomplish this by breaking down our tables and splitting them into MySQL and MongoDB (also known as NoSQL). By doing this we were able to store old customer records to analyze in MongoDB as it is more efficient when dealing with large amounts of data that needs to be scalable. Additionally, MongoDB is more efficient when dealing with data that is this big which led us to this decision. We decided that we could use MySQL for the other tables as it will just host current bookings, which means the data size will be smaller and more organized which is better for the use case of storing peoples’ reservations. Overall, by creating our databases in both MongoDB and MySQL we will be able to maximize efficiency while keeping the integrity of the databases and users’ data.

# Reference

[1] M. Chan, M. C. on A. 21, and N. \*, “SQL vs. NoSQL - what's the best option for your database needs?,” Thorn Technologies, 15-Sep-2021. [Online]. Available: https://www.thorntech.com/sql-vs-nosql/. [Accessed: 02-Dec-2021].

[2] R. Sheldon, “How to choose between SQL and NoSQL databases,” Simple-Talk, 03-May-2021. [Online]. Available: https://www.red-gate.com/simple-talk/databases/nosql/how-to-choose-between-sql-and-nosql-databases/. [Accessed: 01-Dec-2021].

[3] J. Mostipak, “Hotel Booking Demand,” Kaggle, 13-Feb-2020. [Online]. Available: https://www.kaggle.com/jessemostipak/hotel-booking-demand. [Accessed: 01-Dec-2021].