# BEMM459J
# Database Technologies for Business Analytics

Group Number: 21

Word Count: 2702

# A1. Overview and Business model

A group of young graduates faced a problem in the UK market which was that there was no unified platform to book tickets for movies anywhere in the country. They saw this as an opportunity to develop their own platform, which they would later name BookMyTickets. Users would not have to go to a specific multiplex or theatre app or website, they could do this from the comfort of their home. This vendor-agnostic platform helps users to select whichever multiplex they like. The business model works in a way that a small percentage of the price of the ticket is the revenue for BookMyTickets, and the rest goes to the multiplex vendor.

Users will log in with the username and password and can select from the various movies running in the multiplex which will show them the number of seats and screen number. After the payment for the ticket, the user gets the booking id which will be the confirmation of their ticket booked in the respective auditorium.
A few years later, BookMyTickets decided to add options that help the users with nearby restaurants and taxi services. This mobile application will help users to find nearby restaurants where they can go out for dinner and what taxi services will be there in proximity.

# A2. Assumptions

- Multiplex movie theatres with bigger facilities may contain multiple auditoriums.
- Flexibility in the number of seats in each auditorium.
- Within a row, each seat has a row number and seat location.
- A film may be shown more than once at various times, or it may be shown concurrently in a separate auditorium.
- Only one seat may be reserved for each screening.
- Desire to keep track of who made each reservation.

# A3. Use case

Booking a movie on the BookMyTickets platform

- User goes on the app and log in with the details of username and password.
- App shows different movies on with various locations within the UK.
- App also shows how many screens and numbers of seats does that auditorium has.
- After making the payment, user is notified with the reservation id which will have information on auditorium id, screen id, seat id, movie id and screen start time.

# A4. Needs for RDBMS and NoSQL databases

Two phase commit features found in relational database will not always be a valid choice for system availability, which the primary reason for the necessity for both RDBMS AND NoSQL databases for BookMyTickets. RDBMS is crucial because it stick to a particular process and applies ACID (Atomic, Consistent, Isolated and Durable) characteristics while performing transaction modifications and queries regardless of any interruption including software malfunction and power outage.

**ACID**

**Atomicity:** Requires a 'all or nothing' behaviour. Either every instruction is correctly carried out or none of them are. The entire transaction takes place at once, or does not happen at all. For example, When one user is trying to book a ticket and he or she can select the seat and reach the payment gateway, but due to some bank server issues, the payment could not go through. Does that mean his or her seat has been booked? It is not, it does not mean the seat has been booked. The full transaction means receiving your ticket and making the payment as well. If any of the systems fail, the operation will be aborted.

**Consistency**: It is the status of the data both before and after the transaction is completed. All the data in the database is accurate after the transaction has been completed. In BookMyTicket, this will verify that the total number of seats booked plus a number of unreserved seats is equal to the total number of seats in the auditorium. After each transaction consistency is checked due to ensure nothing has gone wrong.

**Isolation**: Because of isolation, multiple transactions may be active at once. Any parallel transaction will give the impression that it will not occur at the same time. A locking mechanism is necessary to achieve isolation. It is defined as a state of separation where no data from one database. In BookMyTicket, let's say two people are trying to make a booking for the same seat. To understand this asset property, the two transactions are happening at the same time on the same database in isolation to ensure that one does not affect the other. They are serialized by the system and the one who clicks 1st and makes the payment. The seat gets reserved for him, and the available seats are updated and the user the other user gets an error that that site is not available anymore.

**Durability**: It is the effect of a failure on an ongoing transaction. If a durable transaction finishes unexpectedly, the status of the data will not be affected.

NoSQL databases use eventual consistency because of the difficulty of maintaining consistency. It ensures that copies of the data will eventually be consistent but allows for brief periods of inconsistent copies. CAP theorem states that it is impossible to ensure all the required properties like consistency, availability, and partition tolerance at the same time in a distributed system.

**CAP**

**Consistency**: it means the most recent write or an error is sent to all reads.

**Availability**: it means that even though it is not the latest information, all readings include data.
**Partition**: It means the system functions as intended despite network issues.

One of the cornerstones of object-oriented programming in NoSQL, polymorphism allows the BookMyTickets application to be extended to perform many roles inside a program. For example, a function in NoSQL can serve several purposes, it can have the same name and be utilised differently depending on the arguments it receives such as null or a variable with distinct data type or numerous variables.

There is a necessity of RDBMS and NoSQL to eliminate some challenging searches and joins that need data from various tables within a relational database. No SQL supports de-normalization, this requirement is readily replaced. This polymorphic pattern in NoSQL allows all data to be kept in a single collection for speedier searching. In NoSQL embedded documents can be used to prevent these joins. BookMyTickets needs to combine RDBMS and NoSQL databases due to the complexity, risk, and expense of database replication implementation.

# A5. The preferred NoSQL database – MongoDB

Document databases (MongoDB) are non-relational databases or NoSQL. Document databases prefer flexible documents in place of rigid rows and columns to store data. The most widely used alternative to a relational database is document database. The most significant advantages of MongoDB are:

- It is quick and simple for the developers to use.
- It is flexible and enables the data model to adapt as application requirements do.

MongoDB precisely satisfies Book my ticket needs because according to this project MongoDB takes precedence over No SQL database. The primary cause is that the CRUD (create, retrieve, update, and delete) code can be executed without first defining the collections since it is Schema less.
**Create**: Each document has a special identification number. In the database, documents can be created.
**Read**: The use of the database is to read the documents and by using the specific field values the developers can identify the document by utilizing query language.
**Update**: It is possible to update or modify the existing document partially or fully.
**Delete**: Deletion of documents can be done from the database.
A collection can include many documents because it lacks structure, and it may not have the same number of keys. That is because no relational database (NoSQL) can manage different

structured data. It can make modifications because is schema-less. While the relational database (NoSQL) can only manage the structured data. No modifications can be done here.

# A6. ER Diagram

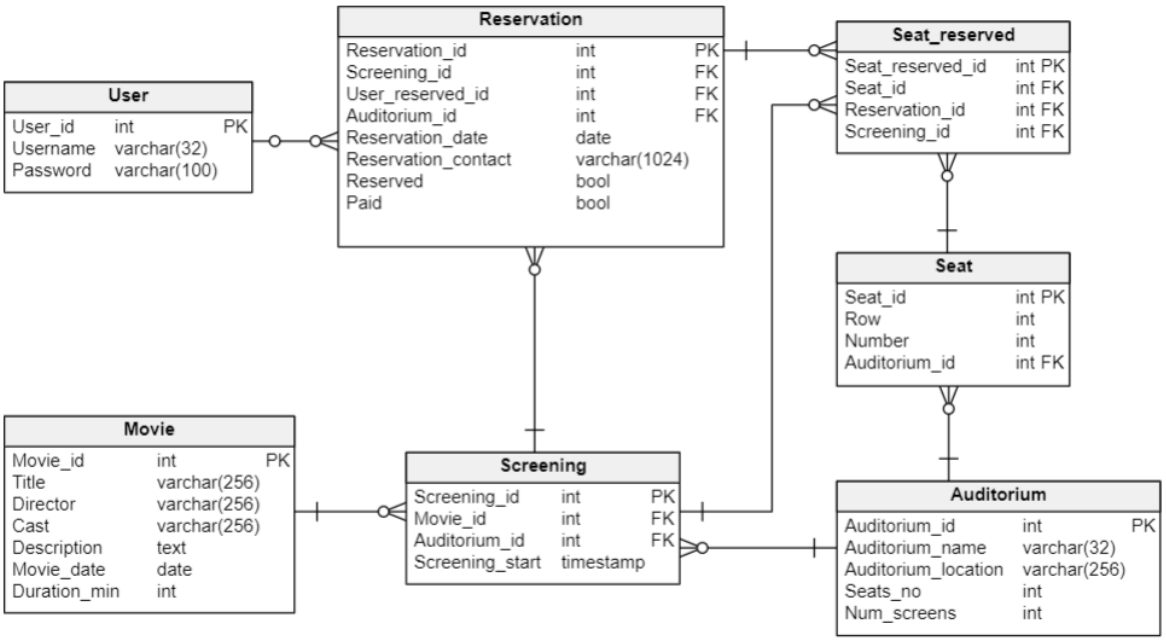



*Fig1: ER Diagram*

**Short table description:**

The movie table contains data about the movie released which will be shown in the theater. The primary key is the movie_id which will be auto incremented. It contains fields which are name explained. The column duration_min shows how long the movie is gives an alert message if someone wants to enter a screening in that theater where there is already a movie running.

The Auditorium table gives information about all the auditoriums with their location, number of screens and the seat numbers available.

The screening table gives information on all the screenings of movies. This must contain related movie and auditorium information and the start time. The restriction is that there cannot be two showings in the same auditorium at the same time. To resolve this, auditorium_id and screening_start is defined as unique key.

The seat table consists of lists of all the seats that are uniquely assigned to one auditoriums.

Main information is stored in the reservation and seat_reserved tables. If the reservation is made (and payment is done) the attributes Paid and reserved will change to True and the user will be given the reservation_id which confirms the booking.

- User and Reservation entities have one-to-many relationship. So, the user logged in can make one reservation or multiple reservations and one seat cannot be reserved by different user.
- Reservation and Seat_reserved entities have one-to-many relationship. So, one reservation can have one or many seats reserved.
- Seat and Seat_reserved entities have one-to-many relationship which means from the number of seats available user can reserve at least one seat. And the Seat_reserved entity connects the details of the seat in Seat entity to Reservation entity.
- Auditorium and Seat entities have one-to-many relationship which means the auditorium has many seats and all the details regarding seating number for different auditorium is stored in Seats entity.
- Auditorium and Screening entities have one-to-many relationship which means auditoriums can have one or multiple screens to show movies. Screening entity stores the number of screens available to each auditorium.
- Screening and Seat_reserved entities have one-to-many relationship which means one or more seats cane be reserved for the same screen, also it shows the seats reserved for each screens by the user.
- Screening and Reservation entities have one-to-many relationship which means a reservation made by the user can have one or multiple number of screens.
- The Movie entity has a one-to-many relationship with the Screening entity which means a movie can be played in multiple screens. The movie entity stores the details of each movie they are playing on screen.

# A7. Initial Domain Attributes

| Entity | Attribute name | Description | Datatype and length | Nulls | Multi-Valued |
|---|---|---|---|---|---|
| User | User_id(PK) | Uniquely identifies the user | Integer | No | No |
| | Username | Username of users | Maximum 32 characters | No | No |
| | Password | Password for authentication | Alphanumeric & must include special characters | No | No |
| Reservation | Reservation_id(PK) | Uniquely identifies the reservation | integer | No | No |

| | | | | | |
|---|---|---|---|---|---|
| | Screening_id | Uniquely identifies the screening | integer | No | No |
| | User reserved_id | Uniquely identifies the reservation | integer | No | No |
| | Auditorium_id | Uniquely identifies the auditorium | integer | No | No |
| | Reservation_date | Date of reservation | Date format | No | No |
| | Reservation_contact | Contact number of the user | Text(1024 characters) | No | No |
| | Reserved | Confirmation of reservation | Boolean | Yes | No |
| | Paid | Confirmation of Payment | Boolean | Yes | No |
| Seat Reserved | Seat_reserved_id(PK) | Uniquely identifies the seat reserved | Integer | No | No |
| | Seat_id | Uniquely identifies the seat | Integer | No | No |
| | Reservation_id | Uniquely identifies the reservation | Integer | No | No |
| | Screening_id | Uniquely identifies the screening | integer | No | No |
| Seat | Seat_id(PK) | Uniquely identifies the seat | integer | No | No |
| | Row | Number of rows | integer | No | No |
| | Number | Seat number | integer | No | No |
| | Auditorium_id | Uniquely identifies the auditorium | integer | No | No |
| Auditorium | Auditorium ID(PK) | Uniquely identifies the auditorium | integer | No | No |
| | Auditorium_name | Name of the auditorium | Text(32 characters) | No | No |
| | Auditorium_location | Location of the | Text(256 | No | No |

| | | | | | |
|---|---|---|---|---|---|
| | | auditorium | characters) | | |
| | Seat_no | Seat number | integer | No | No |
| | Num_screens | Screen Number | integer | No | No |
| Screening | Screening_id(PK) | Uniquely identifies the screening | Integer | No | No |
| | Movie_id | Uniquely identifies the movie | integer | No | No |
| | Auditorium_id | Uniquely identifies the auditorium | integer | No | No |
| | Screening_start | Starting time of the movie on the screen. | Timestamp | No | No |
| Movie | Movie_id(PK) | Uniquely identifies the movie | integer | No | No |
| | Title | Title of the movie | Text(256 characters) | No | No |
| | Director | Name of the director | Text(256 characters) | No | No |
| | Cast | Cast of the movie | Text(256 characters) | No | No |
| | Description | Description of the movie | Text | Yes | No |
| | Movie_date | Movie releasing date | Date format | No | No |
| | Duration_min | Minimum duration of the movie | integer | No | No |

# A8. Logical database design using DBDL

## User Table

**User** (User_id, Username, Password)
**Primary Key** User_id

## Reservation Table

**Reservation** (Reservation_id, Screening_id, User_reserved_id, Auditorium_id, Reservation_date, Reservation_contact, Reserved, Paid)

**Primary Key** Reservation_id
**Foreign Key** Screening_id, User_reserved_id, Auditorium_id

## Seat_reserved Table

**Seat_reserved** (Seat_reserve_id, Seat_id, Reservation_id, Screening_id)
**Primary Key** Seat_reserve_id
**Foreign Key** Seat_id, Reservation_id, Screening_id

## Seat

**Seat** (Seat_id, Row, Number, Auditorium_id)
**Primary Key** Seat_id
**Foreign Key** Auditorium_id

## Auditorium

**Auditorium** (Auditorium_id, Auditorium_name, Auditorium_location, seats_no, Num_Screens)
**Primary Key** Auditorium_id

## Screening

**Screening** (Screening_id, Movie_id, Auditorium_id, Screeing_start)
**Primary key** Screening_id
**Foreign Key** Movie_id, Auditorium_id

## Movie

**Movie** (Movie_id, Title, Director, Cast, Description, Movie_Date, Duration_Min)

**Primary Key** Movie_id

# A9. NoSQL – MongoDB

Data in MongoDB has a flexible schema. documents in the same collection. They do not need to have the same set of fields or structure Common fields in a collection's documents may hold different types of data. Data Model Design MongoDB provides two types of data models: Embedded data model and a Normalized data model. Based on our requirement, we also included Embedded data model where we can have (embed) all the related data in a single document, it is also known as a de-normalized data model.

Data is stored in a collection in MongoDB equivalent to a table in RDBMS. Each record is kept in a separate document within the collection, which consists of many documents. Data in MongoDB is stored in JSON format that is in key-value pairs. We created a subset within the

Embedded data model where we can have (embed) all the related data in a single document, it is also known as a de-normalized data model.

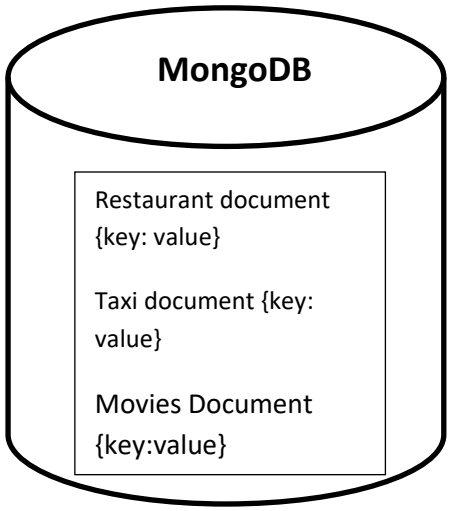
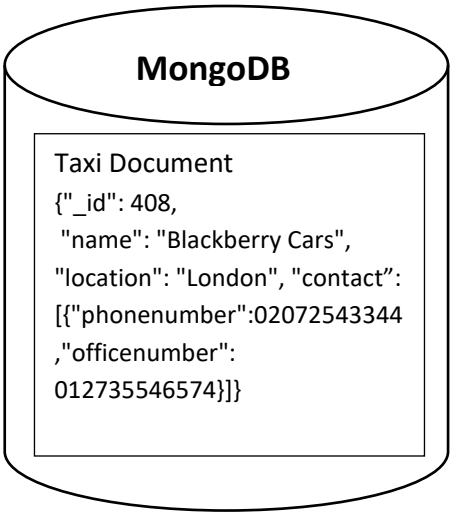
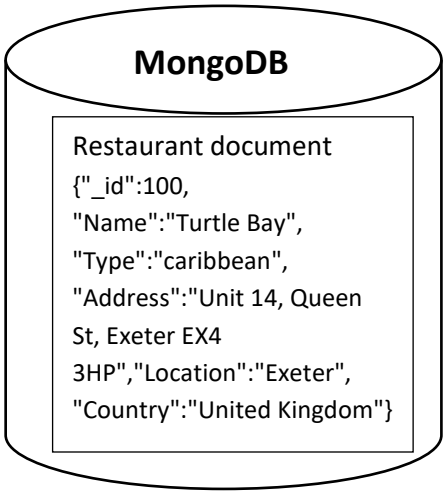



*Fig2: Key value for MongoDB (Dynamic Schema)*



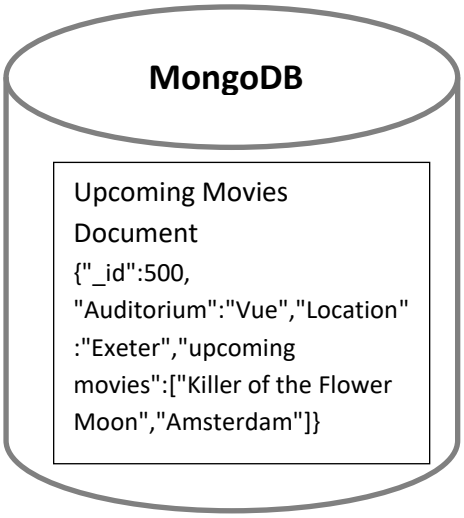



*Fig3: Collections and documents for MongoDB*

We created a subset within the main document, an embedded data model document that supports array data storage because the taxi document was expanded to include more information on the Taxi Service phone numbers. The restaurant document gives the recommendations near the auditorium where the user have reserved their tickets. Along with that we also created an upcoming movie list for the users to anticipate when they are logging into the application. The purpose of NoSQL storage is to enable extremely quick queries, flexibility, and developer friendliness.

# A10. Polyglot Persistence for BookMyTickets Application

Polyglot persistence refers to use a variety of data storage methods for storing data depending on how the data will be used by different applications. This procedure is predicated on the knowledge that some data stores will handle particular types of data more effectively than others. Benefits of polyglot persistence are as follows:

- It supports one or more database models which can choose the most appropriate data model.
- To reduce fragmentation polyglot persistence streamlines processes and aids in the selection of most appropriate component.
- Response time is faster
- Relevance oriented processing would be assigned naturally.
- It enhances flexibility between relational and no relational database.

BookMyTickets application uses RDBMS and NoSQL databases in designing its data architecture to benefit from a mixture of data sources i.e. SQLite and MongoDB.
The data flow within the application was designed to allow the data to be distributed in the system. The critical tables like User, Reservation, Movie, Seat_reserved, Seat, Auditorium, Screening were designed using relational database i.e. SQLite. The less critical ones such as taxi and restaurants were designed as collection on the document database i.e. MongoDB.

# A11. Recommendations

- Ticket cancelation service can be provided.
- A table to store the payment details can be included in the current relational database design.
- Include actual hotel and cab reservations in addition to merely making advice.
- Wallet, vouchers, and redeemable vouchers could be added as a service.

# A12. References

*NoSQL Vs SQL Databases*. (n.d.). MongoDB. Retrieved July 21, 2022, from https://www.mongodb.com/nosql-explained/nosql-vs-sql#:~:text=NoSQL%20databases%20offer%20many%20benefits