**MOVIE TICKET RESERVATION SYSTEM  
TEAM 7**

|  |  |
| --- | --- |
| **Team Members** | **NUID** |
| Krutik Kanakia | 002787847 |
| Kumar Mehul | 002761391 |
| Mansi Sanjeev Upadhyay | 002766397 |
| Tanuj Verma | 002726506 |

**Data Model:** Document (NoSQL)

**Target Platform:** Arango DB

**Objective/Scope:**

* Create a Arango Database System to store movie reservation system information
* Implement Data Validation to ensure that the data entered in the database is accurate and consistent
* Use indexing to improve the performance and scalability of the database
* Use complex queries to extract maximum information from our database
* Use visualizations to discover the trends and movie popularity among the customers

**Visualization Tool:** Tableau

**CONTENTS**

1. **Architecture Diagram**
   1. Overview of system components
   2. Data flow between components
2. **Entity Relationship Diagram**
   1. Collections definitions
   2. Edges definitions
3. **Graph Diagram** a. Edge definitions
4. **Data Previews**
   1. Sample data for each Collections
   2. Data formats
5. **ETL Process**
   1. Data extraction techniques
   2. Data transformation methods
   3. Data loading procedures
6. **Importing Data into Arango DB**
   1. Collection and edge definitions with indexes
   2. Data import process
   3. Creating indexes in database collections
7. **Data Refresh**
   1. Data update frequency
   2. Processes for updating the database and automating it
8. **Queries Executed on the Implemented Database-Reports**
   1. Query explanations and results
9. **Visualizations**
   1. Queries for data visualization
   2. Dashboard design and layout
   3. Interpretation of visualization results
10. **Conclusion**

**ARCHITECTURE DIAGRAM**

**Diagram

Description automatically generated**

**Key takeaways:**

* **Data Sources:** We used Mockaroo as our primary data source for our project.
* **Data Ingestion:** For pre-processing the data we used python script and “arangoimport” command-line tool as our source to push data into Arango DB.
* **Data Storage:** We used Arango DB as our source for data Storage.
* **Visualization:** Tableau was used for visualizing different parameters of our Database.
* **Data Analysis:** Complex DB queries was used for data analysis

**ENTITY RELATIONSHIP DIAGRAM**

**Diagram

Description automatically generated**

**Following are the collections used in our ER Diagram:**

**Collection: Customer**

Attributes:‘Customer\_ID’, ‘Name’, ‘Email’, ‘Phone’, ‘Address’, ‘State’, ‘Password’

Description: Customer Entity contains the information of the Customers that book the movie ticket.   
Relation: Customer is related to Reservation as the customer makes a reservation after looking at the showtimes.

**Collection: Reservation**

Attributes:’Reservation\_ID’,’Customer\_ID’,’Show\_ID’,’Seats’,’DateTime’ Description: Reservations entity contains the details of the reservation made by the customers  
Relation: Reservation is related to Customer as Customer is the one making the reservation. It is also related to the Showtime collection from which it derives the details of the showtime for which the reservation has been made.

**Collection: ShowTime**

Attributes:’Show\_ID’,’Movie\_ID’,’Theater\_ID’,’Start\_Time’,’Price’Description: Showtimes entity contains the details of the different showtimes of the movie according to the theaters in which the movie is being shown.  
Relation: Showtimes is related to Movie and Theaters entities from which it will derive the details of the movie and the theater in which the movie is being shown.

**Collection: Movie**

Attributes:’Movie\_ID’,’Title’,’Release\_Date’,’Genre’’,Rating’,’Duration\_hr’,’Language’ Description: The Movie entity stores the different details about the movie like the movie name, movie duration, genre,rating,etc.  
Relation: Movie entity is related to the Theaters entity in which means the theaters in which the movie is being shown, and to the entity Showtimes which has the details about the showtimes of the different movies.

**Collection: Theater**

Attributes:’Theater\_ID’,’Movie\_ID’,’Name’,’Location’,’Capacity’,’Amenities’ Description: Theaters has the details of different theaters like the theater name, location, capacity, etc.  
Relation: Theater is related to Movies from which it derives the details of the movies being shown there, and to Showtimes, showing the showtimes at the different theaters.

**GRAPH DIAGRAM**

**Diagram

Description automatically generated**

**EDGE COLLECTIONS**

Customer and Reservation (**cust\_resv**): This edge collection represents the relationship between customers and their reservations. Each edge connects a customer to their reservation, indicating which customer made which reservation.

Reservation and ShowTime (**res\_show**): This edge collection represents the relationship between reservations and showtimes. Each edge connects a reservation to a showtime, indicating for which showtime the reservation has been made.

ShowTime and Movie (**show\_movie**): This edge collection represents the relationship between showtimes and movies. Each edge connects a showtime to a movie, indicating which movie is being played at a specific showtime.

ShowTime and Theater (**show\_theater**): This edge collection represents the relationship between showtimes and theaters. Each edge connects a showtime to a theater, indicating in which theater a specific movie is being played at a particular showtime.

Movie and Theater (**movie\_theater**): This edge collection represents the relationship between movies and theaters. Each edge connects a movie to a theater, indicating which theaters are playing a specific movie.

**DATA PREVIEWS**

Below screenshots tell us the nature of data we got from our data source for all collections:

**Movie Theater**

**Table

Description automatically generated**Table

Description automatically generated

**ShowTime Reservations**

**Table

Description automatically generated with medium confidenceTable

Description automatically generated**

**Customer**

**Table

Description automatically generated**

* In above Table we observe that there are multiple columns with null values and duplicate data.
* We will clean that data using python. After cleaning data, we will be updating our original Data files with processed data.

.

**ETL PROCESS**

Below code reads data from five CSV files into pandas Data Frames, removes null and duplicate rows, and then saves the cleaned and transformed data back to the same CSV files

**Python Script**

import pandas as pd

import csv

#Read all the CSV files

df\_Movie = pd.read\_csv("/Users/tanujverma/Desktop/NEU/ADBMS/ADBMS\_project/Movie.csv")

df\_Theater = pd.read\_csv("/Users/tanujverma/Desktop/NEU/ADBMS/ADBMS\_project/Theater.csv")

df\_ShowTime = pd.read\_csv("/Users/tanujverma/Desktop/NEU/ADBMS/ADBMS\_project/ShowTime.csv")

df\_Reservation = pd.read\_csv("/Users/tanujverma/Desktop/NEU/ADBMS/ADBMS\_project/Reservation.csv")

df\_Customer = pd.read\_csv("/Users/tanujverma/Desktop/NEU/ADBMS/ADBMS\_project/Customer.csv")

# Print row and column counts before dropping null and duplicate values

print("Row and column counts before dropping null and duplicate values:")

print("df\_Movie:", df\_Movie.shape)

print("df\_Theater:", df\_Theater.shape)

print("df\_ShowTime:", df\_ShowTime.shape)

print("df\_Reservation:", df\_Reservation.shape)

print("df\_Customer:", df\_Customer.shape)

# Drop null and duplicate values from each data frame

df\_Movie.dropna(inplace=True)

df\_Movie.drop\_duplicates(inplace=True)

df\_Theater.dropna(inplace=True)å

df\_Theater.drop\_duplicates(inplace=True)

df\_ShowTime.dropna(inplace=True)

df\_ShowTime.drop\_duplicates(inplace=True)

df\_Reservation.dropna(inplace=True)

df\_Reservation.drop\_duplicates(inplace=True)

df\_Customer.dropna(inplace=True)

df\_Customer.drop\_duplicates(inplace=True)

# Print row and column counts after dropping null and duplicate values

print("\nRow and column counts after dropping null and duplicate values:")

print("df\_Movie:", df\_Movie.shape)

print("df\_Theater:", df\_Theater.shape)

print("df\_ShowTime:", df\_ShowTime.shape)

print("df\_Reservation:", df\_Reservation.shape)

print("df\_Customer:", df\_Customer.shape)

* **Output:**

Table

Description automatically generated with low confidence

We can see here all columns with null and duplicate values are remove in latter output (reduced row counts)

* **Updated our original CSV files:**

df\_Movie.to\_csv('/Users/tanujverma/Desktop/NEU/ADBMS/ADBMS\_project/Movie.csv', index=False)

df\_Theater.to\_csv('/Users/tanujverma/Desktop/NEU/ADBMS/ADBMS\_project/Theater.csv', index=False)

df\_ShowTime.to\_csv('/Users/tanujverma/Desktop/NEU/ADBMS/ADBMS\_project/ShowTime.csv', index=False)

df\_Reservation.to\_csv('/Users/tanujverma/Desktop/NEU/ADBMS/ADBMS\_project/Reservation.csv', index=False)

df\_Customer.to\_csv('/Users/tanujverma/Desktop/NEU/ADBMS/ADBMS\_project/Customer.csv', index=False)

Now our data is clean and can be imported to Arango DB WebUI.

**WebUI before Importing the Data**

Text

Description automatically generated with medium confidence

**IMPORTING DATA IN ARANGO-DB: Collection and Edges**

Now we will import all our **collection** Files using Arangoimport.

**Queries:**

We will write 5 queries for 5 collection documents which we cleaned earlier  
  
**1.**arangoimport –file /Users/tanujverma/Desktop/NEU/ADBMS/ADBMS\_project/Movie.csv –type csv –collection **Movie** –create-collection

Text

Description automatically generated

Similarly, we will use below commands to import data to our DB for remaining all collections.

**2.**arangoimport –file /Users/tanujverma/Desktop/NEU/ADBMS/ADBMS\_project/Reservation.csv –type csv –collection **Reservation** –create-collection

**3.**arangoimport –file /Users/tanujverma/Desktop/NEU/ADBMS/ADBMS\_project/Theater.csv –type csv –collection **Theater** –create-collection

**4.**arangoimport –file /Users/tanujverma/Desktop/NEU/ADBMS/ADBMS\_project/Customer.csv –type csv –collection **Customer** –create-collection

**5.**arangoimport –file /Users/tanujverma/Desktop/NEU/ADBMS/ADBMS\_project/ShowTime.csv –type csv –collection **ShowTime** –create-collection

**After importing data out UI will look like:**

Graphical user interface, application

Description automatically generated

**Customer\_ID :1 data**

Graphical user interface, text, application

Description automatically generated

**Now we will write AQL queries for all our Edges(total:5) in Web UI**

**1.Customer and Reservation:**cust\_resv

FOR c IN Customer

FOR r IN Reservation

FILTER r.Customer\_ID == c.Customer\_ID

INSERT {

“\_from”: c.\_id,

“\_to”: r.\_id,

“type”: “booking”

} INTO cust\_resv

Graphical user interface, application

Description automatically generated

**2.Reservation and ShowTime:** res\_show

FOR r IN Reservation

FOR s IN ShowTime

FILTER r.Show\_ID == s.Show\_ID

INSERT {

“\_from”: r.\_id,

“\_to”: s.\_id,

“type”: “forShowtime”

} INTO res\_show

Table

Description automatically generated

**3.ShowTime and Movie :** show\_movie

FOR s IN ShowTime

FOR m IN Movie

FILTER s.Movie\_ID == m.Movie\_ID

INSERT {

“\_from”: s.\_id,

“\_to”: m.\_id,

“type”: “forMovie”

} INTO show\_movie

Graphical user interface, application, Teams

Description automatically generated

**4.ShowTime and Theater:** show\_theater

FOR s IN ShowTime

FOR t IN Theater

FILTER s.Movie\_ID == t.Movie\_ID

INSERT {

“\_from”: s.\_id,

“\_to”: t.\_id,

“type”: “forshowtheater”

} INTO show\_theater

**Table

Description automatically generated with medium confidence**

**5.Movie and Theater :** movie\_theater

FOR m IN Movie

FOR t IN Theater

FILTER m.Movie\_ID == t.Movie\_ID

INSERT {

“\_from”: m.\_id,

“\_to”: t.\_id,

“type”: “formovietheaters”

} INTO movie\_theater

Table

Description automatically generated with medium confidence

**Now our database is fully implemented.** Screenshot of all Edge and collection files are below:

Graphical user interface, application, table

Description automatically generated

We now create a few **indexes** which will be used for query optimization.

**Customer**: Persistent index on Email

Graphical user interface, application

Description automatically generated

**Movie**: Persistent Index on Movie\_ID

Graphical user interface, application

Description automatically generated

**Reservation**: Geoindex on Location

Graphical user interface, application

Description automatically generated

**Now we converted our csv files with same data to Json files for future implementations  
(same data in now in json format)**

**DATA REFRESH**

**Python-based Data Refresh Implementation**

* Utilizing Python to monitor and update data in the Arango DB web UI when changes are made to the node files.
* JSON Files Storage: Storing JSON files for all collections in the following directory: "/Users/tanujverma/Desktop/NEU/ADBMS/ArangoDB".
* File Monitoring: Actively monitoring JSON files in the specified directory for any modifications or updates.

**Implementation Example**

We have created below: **ONGOING DATA REFRESH**(we can create a script of below code and run it via terminal(cronjob) for constant monitoring or at intervals)

import json

import time

from arango import ArangoClient

from watchdog.observers import Observer

from watchdog.events import FileSystemEventHandler

class MyHandler(FileSystemEventHandler):

def on\_modified(self, event):

if event.src\_path.endswith('.json'):

update\_arango\_db(event.src\_path)

def update\_arango\_db(json\_file\_path):

file\_name = json\_file\_path.split('/')[-1].split('.')[0]

if file\_name in ['Movie', 'Theater', 'ShowTime', 'Reservation', 'Customer']:

update\_collection(json\_file\_path)

print(f"Collection updated: {file\_name}")

def update\_collection(json\_file\_path):

collection\_name = json\_file\_path.split('/')[-1].split('.')[0]

# Connect to the ArangoDB server

client = ArangoClient(hosts='http://localhost:8529')

db = client.db('\_system', username='root', password='')

# Clear the existing collection

if db.has\_collection(collection\_name):

collection = db.collection(collection\_name)

collection.truncate()

else:

# Create the collection if it doesn't exist

db.create\_collection(collection\_name)

collection = db.collection(collection\_name)

# Import data from the JSON file

with open(json\_file\_path, 'r') as f:

data = json.load(f)

for document in data:

collection.insert(document)

def main():

path = "/Users/tanujverma/Desktop/NEU/ADBMS/ArangoDB" # Set your path to the JSON files

event\_handler = MyHandler()

observer = Observer()

observer.schedule(event\_handler, path, recursive=False)

observer.start()

try:

while True:

time.sleep(1)

except KeyboardInterrupt:

observer.stop()

observer.join()

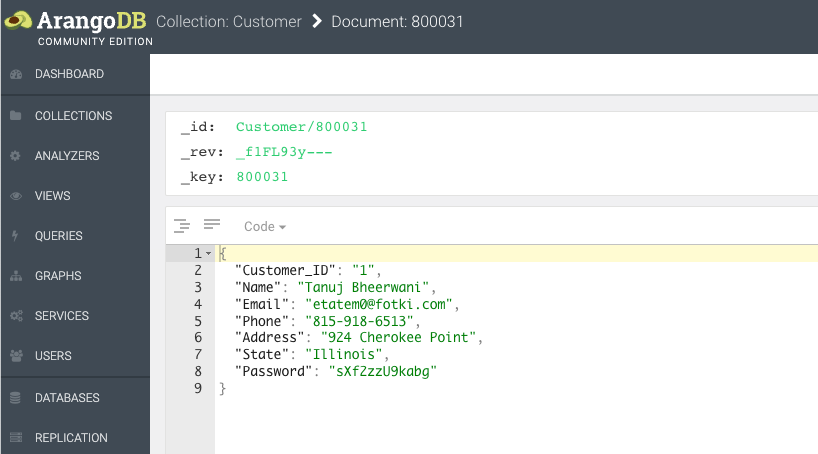
if \_\_name\_\_ == "\_\_main\_\_":

main()

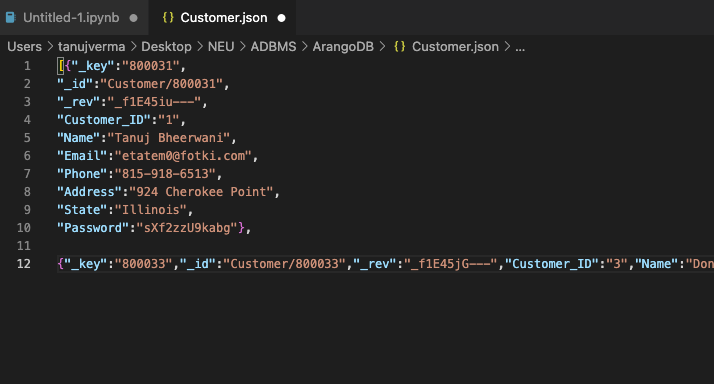
Now whenever we change any attributes in the code that collection will be reflected.

**Example of Implementation**

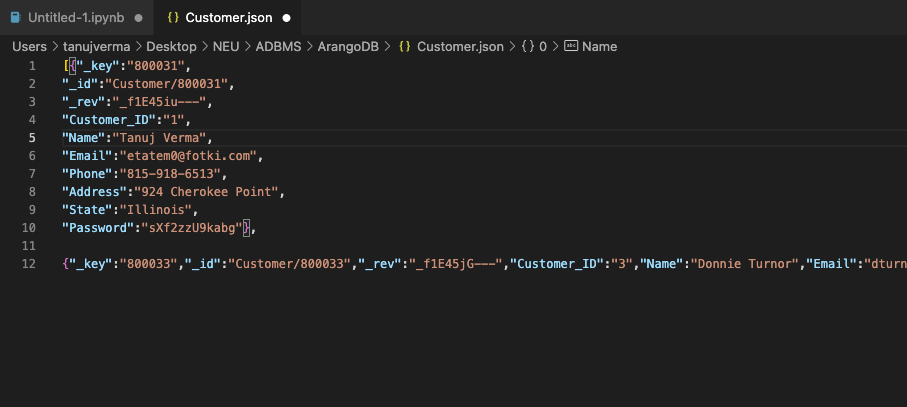
Below are the customer details for Customer\_ID 1 :



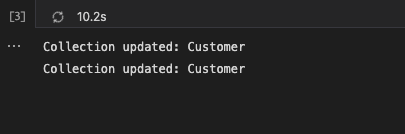
We are going to change the Customer Name. Now we make change into the JSON file of Customer.  
Originally in our json file, we have the name as “Tanuj Bheerwani”



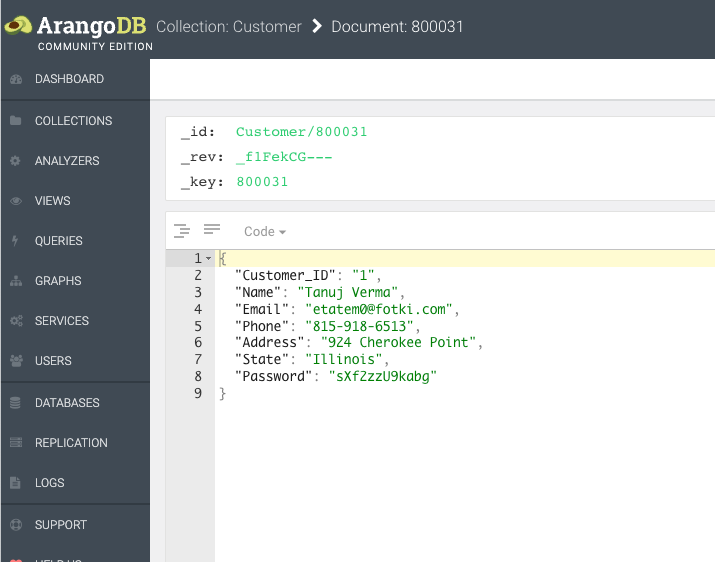
We change it to “Tanuj Verma” and save it.



Our Python script immediately found which collection was updated and pushed that change into the web UI.



Now we check our web UI. We can see that Name is automatically updated. That’s how we implemented ongoing data refresh.



CronJob automation steps we can use for automation.

* Opened a terminal and found the path to the Python interpreter on my system
* Then opened the crontab file for the current user by typing crontab -e in the terminal.
* Set set the script to run every hour by adding the following line to the crontab file:

0 \* \* \* \* /usr/bin/python3 /Users/tanujverma/Desktop/data\_refresh.py

* Saved the crontab file and exited the editor. The cron daemon is now set to execute the Python script at the specified intervals.

**QUERIES WHICH WE EXECUTED ON OUR IMPLEMENTED DATABASE-REPORTS**

**1. Find all movies in a specific language: “Bulgarian” with a minimum rating:**

**Query:**

LET language = "Bulgarian"   
LET minRating = 4.0  
FOR m IN Movie

FILTER m.Language == language AND m.Rating >= minRating RETURN m.Title

**Output:**



**2. Find customers who have made reservations for a specific movie: “Hill, The”**

**Query:**

FOR c IN Customer

FOR cr IN cust\_resv

FILTER cr.\_from == c.\_id

FOR r IN Reservation

FILTER r.\_id == cr.\_to

FOR s IN ShowTime

FILTER s.Show\_ID == r.Show\_ID

FOR sm IN show\_movie

FILTER sm.\_from == s.\_id

FOR m IN Movie

FILTER m.\_id == sm.\_to AND m.Title == "Hill, The"

RETURN c.Name

**Output:**

Graphical user interface, text, application, email

Description automatically generated

**3.Show count of reservations made by each customer :**

**Query:**

FOR c IN Customer  
LET reservationCount = (

FOR cr IN cust\_resv FILTER cr.\_from == c.\_id FOR r IN Reservation

FILTER r.\_id == cr.\_to

RETURN 1 )

RETURN {"CustomerName": c.Name, "Reservations": SUM(reservationCount)}

**Output:**

Graphical user interface, text, application, email

Description automatically generated

**4. Show count of showtimes for all the movies**

**Query:**

FOR m IN Movie  
LET showtimeCount = (

FOR sm IN show\_movie FILTER sm.\_to == m.\_id FOR s IN ShowTime

FILTER s.\_id == sm.\_from

RETURN 1  
)RETURN {"MovieTitle": m.Title, "Showtimes": SUM(showtimeCount)}

**Output:**

Graphical user interface, application

Description automatically generated

**5. Show all the customers who have booked movies**

**Query :**

FOR c IN Customer FOR cr IN cust\_resv

FILTER cr.\_from == c.\_id FOR r IN Reservation

FILTER r.\_id == cr.\_to FOR s IN ShowTime

FILTER s.Show\_ID == r.Show\_ID FOR sm IN show\_movie

FILTER sm.\_from == s.\_id FOR m IN Movie

FILTER m.\_id == sm.\_to  
RETURN {"CustomerName": c.Name, "MovieTitle": m.Title}

**Output:**

**Table

Description automatically generated with medium confidence**

**6. Show all the movies running in a theater with the movie name, theater name, and the count of shows in that theater**

**Query:**

FOR t IN Theater  
FOR mt IN movie\_theater

FILTER mt.\_to == t.\_id FOR m IN Movie

FILTER m.\_id == mt.\_from LET showCount = (

FOR st IN show\_theater FILTER st.\_to == t.\_id FOR s IN ShowTime

FILTER s.\_id == st.\_from AND s.Movie\_ID == m.Movie\_ID

RETURN 1 )

RETURN {"MovieTitle": m.Title, "TheaterName": t.Name, "ShowCount": SUM(showCount)}›

**Output:**

Graphical user interface

Description automatically generated

**7. Show the customer’s name, watched genres, and count of times they watched those genres**

**Query:**

FOR c IN Customer FOR cr IN cust\_resv

FILTER cr.\_from == c.\_id FOR r IN Reservation

FILTER r.\_id == cr.\_to FOR s IN ShowTime

FILTER s.Show\_ID == r.Show\_ID FOR sm IN show\_movie

FILTER sm.\_from == s.\_id FOR m IN Movie

FILTER m.\_id == sm.\_to  
COLLECT customerName = c.Name, genre = m.Genre WITH COUNT INTO genreCount RETURN {"CustomerName": customerName, "Genre": genre, "Count": genreCount}

**Output:**

Graphical user interface, text, application

Description automatically generated

**8. Show all the movies watched by the customer whose duration is more that 1 hour**

**Query:**

FOR c IN Customer FOR cr IN cust\_resv

FILTER cr.\_from == c.\_id FOR r IN Reservation

FILTER r.\_id == cr.\_to FOR s IN ShowTime

FILTER s.Show\_ID == r.Show\_ID FOR sm IN show\_movie

FILTER sm.\_from == s.\_id FOR m IN Movie

FILTER m.\_id == sm.\_to AND m.Duration\_hr > 1  
RETURN {"CustomerName": c.Name, "WatchedMovie": m.Title, "Duration\_hr": m.Duration\_hr}

**Output:**

Graphical user interface, application

Description automatically generated

**9.Show movies running in all theaters**

**Query:**

FOR m IN Movie  
COLLECT movieID = m.Movie\_ID INTO moviesInAnyTheater = m LET theaters = (

FOR mt IN movie\_theater  
FILTER mt.\_from == moviesInAnyTheater[0].\_id FOR t IN Theater

FILTER t.\_id == mt.\_to

RETURN t.Name )

FILTER LENGTH(theaters) > 0  
RETURN {"MovieTitle": moviesInAnyTheater[0].Title, "Theaters": theaters}

**Output:**

Graphical user interface, application

Description automatically generated

**10. Show movies not running in any theater**

**Query:**

FOR m IN Movie  
FILTER m.Movie\_ID NOT IN (

FOR mt IN movie\_theater RETURN mt.\_from )  
RETURN {"Title": m.Title, "Genre": m.Genre}

**Output:**

Background pattern

Description automatically generated with medium confidence

**VISUALIZATIONS**

We have used tableau for visualization. In total we have created 6 visualizations and 2 dashboards.  
**Steps to Connect to tableau:**

* Install tableau services from web UI

Graphical user interface

Description automatically generated with low confidence

* After installation is completed, we open Tableau and go to Connections from web Server and put in the URL : <http://localhost:8529/_db/_system/tableau/index.html>.This will make sure that we are connecting directly our Arango DB with Tableau and can run queries from tableau directly.
* After that we will put all our queries in TABLES show below. In total we have used 6 complex queries for visualizations

Graphical user interface, table

Description automatically generated

After writing queries we click Extract. We then connect all tables logically and our data is ready for visualization.

Graphical user interface

Description automatically generated

**QUERIES FOR VISUALIZATION**

**1.** **CUSTOMERS WITH RESERVATION COUNTS**

**Query:**

FOR c IN Customer

LET reservationCount = (

FOR cr IN cust\_resv

FILTER cr.\_from == c.\_id

FOR r IN Reservation

FILTER r.\_id == cr.\_to

RETURN 1)

LET reservationSum = SUM(reservationCount)

SORT reservationSum desc

RETURN {"CustomerName": c.Name, "Reservations": reservationSum}

**Output in Arango DB:**

**Background pattern

Description automatically generated**

**Visualization (bar plot):**

Chart, bar chart

Description automatically generated

**2. NUMBER OF THEATERS SCREENING EACH MOVIE**

**Query:**

FOR m IN Movie

FOR t IN Theater

FILTER m.Movie\_ID == t.Movie\_ID

COLLECT movie\_title = m.Title INTO movie\_group

RETURN { movie\_title: movie\_title, theater\_count: LENGTH(movie\_group) }

**Output in Arango DB:**

Background pattern

Description automatically generated

**Visualization (packed bubbles):**



**3.CUSTOMERS' MOVIE WATCHLIST AND DURATION**

**Query:**

FOR c IN Customer FOR cr IN cust\_resv

FILTER cr.\_from == c.\_id FOR r IN Reservation

FILTER r.\_id == cr.\_to FOR s IN ShowTime

FILTER s.Show\_ID == r.Show\_ID FOR sm IN show\_movie

FILTER sm.\_from == s.\_id FOR m IN Movie

FILTER m.\_id == sm.\_to AND m.Duration\_hr > 1

RETURN {"CustomerName": c.Name, "WatchedMovie": m.Title, "Duration\_hr": m.Duration\_hr}

**Output in Arango DB:**

Background pattern

Description automatically generated

**Visualization (Side-by-Side Bar plot with 3 entities):**

Chart, bar chart

Description automatically generated

**4. MOVIES WITH COUNT OF SHOWTIMES**

**Query :**

FOR m IN Movie

LET showtimeCount = (

FOR sm IN show\_movie FILTER sm.\_to == m.\_id FOR s IN ShowTime

FILTER s.\_id == sm.\_from

RETURN 1

)RETURN {"MovieTitle": m.Title, "Showtimes\_Count": SUM(showtimeCount)}

**Output in Arango DB:**

Background pattern

Description automatically generated

**Visualization (Square Plot):**

Chart, treemap chart

Description automatically generated

**5. GENRE SUMMARY: RESERVATIONS, MOVIE COUNT, AND AVERAGE RATING**

**Query:**

LET genre\_summary = (

FOR m IN Movie

FOR r IN Reservation

FOR s IN ShowTime

FILTER r.Show\_ID == s.Show\_ID

FILTER m.Movie\_ID == s.Movie\_ID

COLLECT genre = m.Genre INTO genre\_group

RETURN {genre: genre,movie\_count: LENGTH(UNIQUE(genre\_group[\*].m.Movie\_ID)),avg\_rating: AVG(genre\_group[\*].m.Rating),

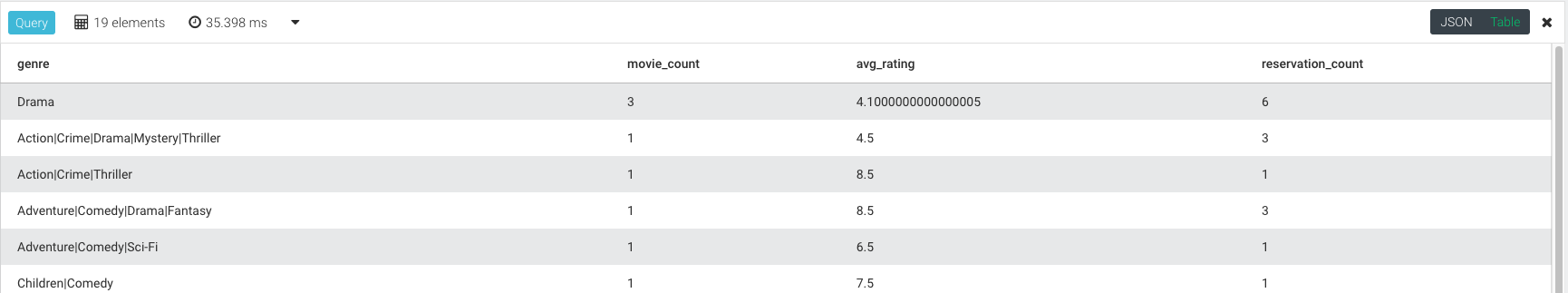
reservation\_count: LENGTH(genre\_group)})

FOR summary IN genre\_summary

SORT summary.movie\_count DESC

RETURN summary

**Output in Arango DB:**



**Visualization (horizontal Side-by-Side bar plot):**

**Chart, bar chart

Description automatically generated**

**6.** **THEATER COUNT BY LOCATION**

**Query:**

FOR t IN Theater

COLLECT location = t.Location INTO theatersGroup

LET theaterCount = LENGTH(theatersGroup)

RETURN {"Location": location, "TheaterCount": theaterCount}

**Output in Arango DB:**

**Background pattern

Description automatically generated**

**Visualization (Maps):**

**Map

Description automatically generated**

**DASHBOARDS**

**1.CUSTOMER DASHBOARD**

This dashboard contains Customer Analysis for our Arango DB data.

**Chart, timeline, bar chart

Description automatically generated**

**Key takeaways:**

1. By analyzing the duration of the movies watched by the customers, we can identify what type of duration preference the customer has, and we can target those customers with the advertisements of their preferred duration movies, which will help increase the return rate of the customer and help increase revenue.
2. By analyzing the data of customers with high reservation count, we can identify the high revenue customers, and we can target the customers with discounts and offers based on the reservation counts, i.e., customers with higher reservation count will get more offers and discounts to keep them encouraged to keep coming back.

**2.THEATER AND GENRE DASHBOARD**

This dashboard contains analysis of Theater count by location and Genre Summary

**A picture containing website

Description automatically generated**

**Key takeaways:**

1. Analyzing the theater count by location can help identify potential market expansion opportunities. States with lower or no theater counts may present untapped markets where the demand for movies might be high.
2. The genre summary visualization offers insights into moviegoers' preferences based on genre. This data can help identify which genres are the most popular and have the highest number of reservations.
3. By comparing the number of movie reservations per genre with the average rating, it is possible to identify which genres generate the most revenue while also maintaining high customer satisfaction. These genres can be targeted for future movie releases and marketing campaigns.

**CONCLUSION**

In this report, we present an in-depth overview of the ArangoDB project, including its architectural design, data modeling, data preprocessing, and data analysis. The project demonstrated the versatility and strength of ArangoDB as a multi-model database, which provides a robust and versatile solution for managing complex data structures and relationships.

We began by outlining the system's architecture and data flow between components, which enabled a clear comprehension of the structure of the project. Following this, we defined the collections, edges, and relationships between them using the Entity Relationship Diagram and Graph Diagram. These diagrams served as a firm basis for designing the database schema and comprehending the underlying data structure.

The Data Previews section provided insight into the dataset by providing a preview of the sample data, its format, and its arrangement within the collections. Then, we discussed the ETL procedure, which consisted of data extraction, transformation, and loading to ensure that the data was efficiently prepared and imported into ArangoDB. The sections that followed described the procedures involved in creating, defining, and importing data into the database, as well as administering data updates and refreshes. Furthermore, to automate the data refresh process and minimize manual intervention, we implemented a cron job that schedules and triggers the Python script at regular intervals.

Execution of queries and reports on the implemented database was a crucial aspect of this endeavor. We demonstrated the robust querying capabilities of ArangoDB by showcasing various query types and explaining their purpose and results. In addition, we investigated the visualizations produced by these queries, creating dashboards and interpreting the results to gain valuable insights.

This ArangoDB initiative effectively demonstrated the benefits of utilizing a multi-model database for managing and analyzing complex data. Through a well-structured design, efficient data preprocessing, and thorough analysis, we were able to extricate valuable insights and demonstrate the potential of ArangoDB for a variety of applications. To expand the project's scope, future work may involve further database schema optimization, investigation of advanced query techniques, and the incorporation of additional data sources.