Activity for Distributed system

prepare the D.S for all business model that you discovered in previous Assignment

1) Prepare the document, where you describe the no. of node needed and what are your expansion plans for scalability (max 3-5 node)

2) which storage concept (RDBMS/NOSQL) will be suited for your business model and why?

3) use fragmentation techniques, like vertical/horizontal to divide entire data of the necessary table you describe in previous assignment.

4)which DS Architecture will you implement to your business model with justification

5) which factor application from CAP theorem to your business model dependency upon it justify the DB

---

1. *Business model: Food Delivery App*

Data model-

i) *User table:*

Field	***Data Type***
User_ID	*Integer*
Username	*String*
Address	*String*
Contact_Detail	*String*

ii) *Restaurant Table:*

Field	***Data Type***
Restaurant_ID	*Integer*
Restaurant_Name	*String*
Location	*String*
Cuisine_Type	*String*

iii) Order Table:

Field	***Data Type***
Order_Id	*Integer*
User_ID	*Integer*
Restaurant_ID	*Integer*
Delivery_ID	*Integer*
Total_Amount	*Float*
Order_Date	*Date*

iv) Payment Transaction Table:

Field	***Data Type***
TransactionID	Integer
UserID	Integer
Order_Id	Integer
Amount	Float
Payment_date	Date

v) Rating and Review table:

Field	***Data Type***
ReviewID	*Integer*
UserID	*Integer*
Restaurant_ID	*Integer*
Rating	*Integer*
Comment	*String*

vi) Order menu table:

Field	*Data Type*
ItemID	*Integer*
Restaurant_ID	*Integer*
Name	*String*
Price	*Float*

a) Node and expansion plan for scalability -

i) User Node:

Attributes: User_ID, Username, Address, Contact_Detail
Expansion Plan: Scale based on the number of registered users and their activities.

ii) Restaurant Node:

Attributes: Restaurant_ID, Restaurant_Name, Location, Cuisine_Type
Expansion Plan: Scale based on the number of registered restaurants and their offerings.

iii) Order Node:

Attributes: Order_Id, User_ID, Restaurant_ID, Delivery_ID, Total_Amount, Order_Date
Expansion Plan: Scale based on the volume of orders. Implement sharding or distribution based on geographic regions for optimized order processing.

b) Database system (RDBMS/NoSQL) -

I believe MongoDB (NoSQL database) is better suited for a food delivery application
Justification:

- *Flexible Schema: Accommodates evolving data structures and accommodates semi-structured data like user reviews.*
- *Scalability: Horizontally scalable, supporting increased data and user load.*
- *Document-Oriented: Suitable for storing and retrieving complex, nested data structures.*

c) fragmentation techniques -

Horizonal fragmentation: (based on region)

(user table)
Fragment 1: Jaipur
Fragment 2: Alwar

Few entity -

Fragment 1: Jaipur

Order_id	*User_id*	*Restaurant_ID*	*Delivery_ID*	*Total_Amount*	*Order_date*
001	*101*	*201*	*301*	*1200.00*	*19-01-2024*
007	*105*	*205*	*310*	*1500.00*	*25-01-2024*

Fragment 2: Alwar

Order_id	*User_id*	*Restaurant_ID*	*Delivery_ID*	*Total_Amount*	*Order_date*
002	*110*	*201*	*305*	*1000.00*	*20-01-2024*
004	*111*	*210*	*309*	*1200.00*	*22-01-2024*

Justification why I use horizontal fragmentation:

In database design technique where rows of a table are divided or distributed based on a specific criteria. Some main reason it's crucial to consider certain factors when deciding on the most appropriate fragmentation for a food delivery app:

i) *Improved Query Performance:*
ii) *Efficient Data Storage and Retrieval for Specific Use Cases:*
iii) *Geographical Distribution for Regional Operations:*
iv) *Data security (both user data security and company data)*

d) Distributed System Architecture:

Client-server architecture

I choose client - server architecture system because it divided into two main components – clients and servers. Clients request services or resources, and servers provide these services or resources. server is responsible for coordinating transactions, managing data storage, and providing access control, communication between clients and servers is typically facilitated through a network.

I think it will suitable choice for the Food Delivery App due to its scalability, centralized control, security benefits, and efficient resource utilization. It allows for effective management of order processing, payment transactions, and real-time updates, contributing to a reliable and responsive user experience.

e) CAP Theorem application

It's frequently used for big data and real-time applications running at multiple different locations.
Relative to the CAP theorem, MongoDB is a CP data store - it resolves network partitions by maintaining consistency, while compromising on availability

Consistency Partition Tolerance:

Consistency -

MongoDB uses a single-master model for each replica set. When a write is performed, it is acknowledged only after the primary member has committed the write operation.

Partition Tolerance-
MongoDB supports horizontal scaling through sharding. Sharding distributes data across multiple nodes (shards), allowing the system to handle increased data and traffic.

MongoDB, in the context of a distributed system like a Food Delivery App, the emphasis is on maintaining strong consistency within each shard or replica set while ensuring partition tolerance. This allows the system to provide continuous service, even in the face of network partitions or node failures, making it well-suited for scenarios where data consistency is a critical requirement.

2) Business model: Streaming music

a) Node and expansion plan for scalability

i) User table Node:

Attributes: User_ID, Name, Age, Gender, Location_id, Subscription_type
Expansion plan: Scale based on the number of registered users and their activities.

ii) Song library table node:

Attributes: Song_id, Artist_id, Genre, Language, Release_data

Expansion plan: Scale based on the song library expands and the number of play records increases.

iii)Location table:

Attributes: Location_id, City, Country, Region
Expansion plan: Add nodes to cover new regions and optimize for user location-based queries.

b) Database system (RDBMS/NoSQL)

NoSQL Database (e.g. MongoDB):

Justification:

- *Flexible Schema: Well-suited for handling the dynamic nature of music data and user-related information.*

- *Horizontal Scalability: NoSQL databases, like MongoDB, allow for easy horizontal scaling, crucial for a growing user base and varied data types.*

- *Query Performance: NoSQL databases excel in read and write operations, vital for a music streaming service with a large number of users and real-time interactions.*

c) fragmentation techniques

Tables: User Table

Criteria: Distribute user and play data across nodes based on user attributes, such as geographical location or user ID ranges.

Justification: Optimizes data retrieval for location-based queries and user-specific operations.

Horizonal fragmentation: (based on region)

(user table)
Fragment 1: Jaipur
Fragment 2: Alwar

Few entity -

Fragment 1: Jaipur

User_id	*Name*	*Age*	*Gender*	*Location_id*	*Subscription_type*
101	*Rishabh Gupta*	*19*	*Male*	*201*	*Premium*
102	*Raja Pandey*	*20*	*Male*	*201*	*Free*

Fragment 2: Alwar

User_id	*Name*	*Age*	*Gender*	*Location_id*	*Subscription_type*
103	*Prerna shah*	*25*	*Female*	*215*	*Free*

d) Distributed System Architecture

Distributed System Architecture: Client-Server Architecture

In a Client-Server framework, the system comprises two primary elements: clients, which request services or resources, and servers, which supply these services or resources. Typically, communication between clients and servers is facilitated over a network.

When a user streams a song through the app, the client sends a request to the server. The server handles song streaming, updates the play history, and notifies the relevant parties (if needed).

The Client-Server architecture is a suitable choice for the Streaming Music Service due to its scalability, centralized control, security benefits, and efficient resource utilization. It allows for effective management of song streaming, playlist operations, and real-time updates, contributing to a reliable and responsive user

e) CAP Theorem application

It's frequently used for big data and real-time applications running at multiple different locations.
Relative to the CAP theorem, MongoDB is a CP data store - it resolves network partitions by maintaining consistency, while compromising on availability

Justification:

Consistency: Critical for a music streaming service to provide accurate and up-to-date information about users, songs, and play history.

Partition Tolerance: Essential to maintain service availability, especially considering potential network partitions in a distributed environment.

Trade-off: Prioritizing consistency and partition tolerance is crucial for ensuring a reliable and responsive streaming experience, justifying the choice of a NoSQL database that aligns with these requirements.

Rishabh Gupta (BT21GCS020)