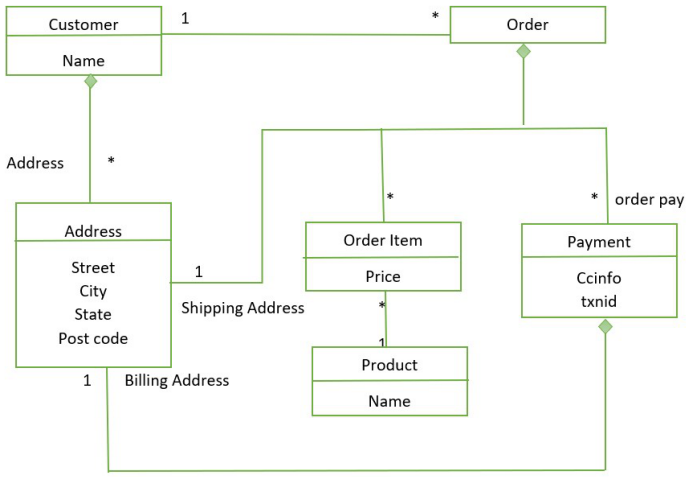
Q1. a. What is NoSQL? Provide a brief explanation of aggregate data models along with a clear diagram, using the concepts of relations and aggregates as examples.

Ans. NoSQL is a class of database management systems designed to handle large volumes of unstructured or semi-structured data that don't necessarily fit into the rigid schema of traditional relational databases. NoSQL databases are optimized for horizontal scaling, distributed architecture, and are particularly suited to large-scale, high-traffic applications.

Key Characteristics of NoSQL Databases

- Schema flexibility: Data doesn’t need to conform to a strict schema.
- Scalability: Optimized for horizontal scaling across distributed systems.
- Varied Data Models: NoSQL databases use a range of models to store data, such
- as document, key-value, column-family, and graph.
- Eventual Consistency: Some NoSQL databases offer tunable consistency levels,
- trading off strict consistency for performance and availability in distributed
- systems.

Aggregate data models are a key concept in NoSQL databases. Unlike relational databases that treat data as rows in tables with fixed relationships, aggregate models bundle related data together into aggregates.



Here in the diagram have two Aggregate: Customer and Orders link between them represent an aggregate. The diamond shows how data fit into the aggregate structure. Customer contains a list of billing address Payment also contains the

billing address The address appears three times and it is copied each time The domain is fit where we don’t want to change shipping and billing address.

1b. Explain impedance mismatch.

Ans. Impedance mismatch, also known as "object-relational impedance mismatch," refers to the differences and inconsistencies between the way data is represented in an object-oriented programming (OOP) language and how it is stored in a relational database. In object-oriented programming, data is typically represented as objects, encapsulating both data and behaviour, while relational databases store data in tables with rows and columns.

Q2. a. Provide a brief explanation of the advantages of relational databases.

Ans. Relational databases offer several advantages that make them a popular choice for managing data across various applications and industries. Here are some key benefits:

1. **Data Integrity**: Relational databases enforce data integrity through constraints such as primary keys, foreign keys, and unique constraints. These ensure that the data remains accurate and consistent.
2. **Structured Query Language (SQL)**: Relational databases use SQL, a powerful and widely-adopted language for querying and managing data, making it accessible for developers and analysts.
3. **Normalization**: The process of normalization reduces redundancy and dependency by organizing data into tables. This leads to efficient storage and easier data management.
4. **Data Relationships**: Relational databases excel at handling complex data relationships. They allow users to easily join tables, enabling sophisticated queries that extract insights across related data sets.
5. **Security**: Relational databases offer robust security features, such as user authentication, role-based access control, and encryption options, ensuring that data is protected from unauthorized access.
6. **Concurrency Control**: They support multiple users accessing the database simultaneously, employing mechanisms like locking and transactions to maintain data consistency.
7. **Backup and Recovery**: Most relational database systems come with built-in solutions for data backup and recovery, ensuring data can be restored in case of failure.

8. **Scalability**: While traditionally better for smaller to medium-sized datasets, many relational database systems can scale effectively to handle larger datasets and more complex applications.
9. **ACID Compliance**: Most relational databases adhere to ACID (Atomicity, Consistency, Isolation, Durability) properties, ensuring reliable transactions.

2b. Explain schema less databases.

Ans. Schemaless databases refer to databases that do not require a fixed schema, meaning data structures can vary dynamically. Most NoSQL databases, like document and key-value stores, are schemaless. This approach offers flexibility, as data can evolve without the need to alter a rigid schema, which is beneficial for applications with frequently changing data structures.
Characteristics:

- Flexible Data Structure: Supports data with varying formats and attributes.
- No Schema Migration: Data model changes are straightforward since there's no fixed schema.
- Data Evolution: Suited for applications that evolve rapidly, enabling easy addition or removal of attributes.

Q3. Write a short note on

i. Key – value data model
The key-value data model is the simplest and most basic NoSQL data model, where data is stored as a collection of key-value pairs. Each unique key points to a single value, which could be a simple data type (string, number) or complex objects like JSON, BLOBs, or serialized objects. Key-value stores are highly performant for simple read and write operations and are often used in caching and session management.
Characteristics:
    - Data Structure: Stores data as a collection of key-value pairs.
    - Keys: Unique identifiers used to retrieve the associated value.
    - Values: The data associated with the key, which can be any data type or serialized
    - structure.

ii. Column family stores
Column family databases (e.g., Apache Cassandra, HBase) organize data in rows and columns but differ from relational databases by grouping columns into column families. Each column family can have a unique set of columns for each row, allowing a flexible, sparse data

structure. This model is especially effective for storing time-series data and for applications requiring fast data access across a wide range of data points.

Characteristics:

- Data Structure: Organizes data into rows and column families. A column family is akin to a table, but rows within the column family can contain different columns.
- Flexible Columns: Each row can contain different columns within the same family.
- Sparse Data: Efficiently handles sparse data, where many columns might be empty.
- High Scalability: Designed for high scalability and distributed data storage, making it suitable for handling large datasets.

iii. Quorums

Quorums are a consistency mechanism in distributed databases used to ensure that a majority of nodes agree on a given operation (read or write) before it is considered complete. This approach is common in systems aiming for high availability and consistency despite network failures.

Types of Quorum Configurations:

- Read Quorum: A read operation is considered successful if it retrieves data from a majority of nodes.
- Write Quorum: A write operation is successful if it is confirmed by a majority of nodes.

Quorum Formula:

To ensure consistency, a typical quorum formula requires that: Read Quorum +

Write Quorum>Total Nodes.

Q4 a. Explain the following

i. Graph databases

A graph database is designed to store and manage data represented as nodes (entities), edges (relationships), and properties (attributes of nodes/edges). This model excels in applications where the relationships between data points are as important as the data itself. Graph databases are highly effective for querying complex and interconnected data.

Characteristics:

- Node-Edge-Property Model: Data is stored as nodes (similar to entities), edges (relationships between nodes), and properties (attributes of nodes and edges).

- Efficient Relationship Management: Quickly traverses relationships between nodes, enabling efficient querying of deeply connected data.
- Schema Flexibility: Schema is not strictly defined, allowing for dynamic data relationships.

ii. Document data models

The document data model organizes data as collections of documents, where each document is a set of key-value pairs similar to JSON or BSON format. Unlike key-value stores, document databases can support complex structures, allowing nested objects and arrays. This model provides greater flexibility in data representation and is ideal for storing semi-structured data.

Characteristics:

- Data Structure: Stores data as documents, typically JSON, BSON, or XML.
- Schema Flexibility: No strict schema; documents within the same collection can have varying structures.
- Embedded Documents: Documents can contain nested documents or arrays, allowing a more hierarchical data structure.
- Indexing and Querying: Allows indexing on specific fields and querying of individual document attributes, not just primary keys.

4 b. How does master slave and peer to peer data distribution model differentiated?

Ans. The master-slave model is a type of replicated database model where one server (the master) is designated to handle all data updates, while one or more slave servers replicate data from the master. The slaves provide read-only access to data, which helps balance the read load. Updates are directed only to the master, which then propagates changes to the slaves.

Characteristics:

- Centralized Write Operations: Only the master can handle write operations.
- Read Scalability: Allows multiple read-only replicas, providing read scalability.
- Consistency Challenges: Ensuring that all slaves are synchronized with the master requires consistent replication mechanisms.

In a peer-to-peer (P2P) model, each node in the network is both a server and a client, meaning all nodes can read and write data, and there is no single master server. Each node communicates and synchronizes directly with others, distributing the data management and avoiding a central point of control.

Characteristics:

- Decentralized: No master node; each node is both a client and a server.
- High Fault Tolerance: If one node fails, others can continue operations without dependency on a central authority.
- Complex Consistency Management: Managing data consistency and coordination across multiple writable nodes can be challenging.

Q5 a. Write a short note on.

i. Read consistency.

Read consistency determines the consistency of data read from a distributed database. In distributed systems, different nodes may have slightly different versions of the data due to replication delays. Read consistency ensures that users are aware of the potential consistency levels when accessing data.

Types of Read Consistency:

- Strong Read Consistency: Guarantees that a read will always return the most recent data after an update.
- Eventual Read Consistency: Allows for temporary discrepancies, as reads may access stale data, with the system eventually achieving consistency.
- Read-Your-Writes Consistency: Guarantees that after a user writes data, any subsequent reads from that same user will see their updates.

ii. CAP theorem

The CAP theorem states that in a distributed data system, it is impossible to simultaneously provide all three of the following guarantees:

- Consistency (C): All nodes see the same data at the same time.
- Availability (A): Every request receives a response (either success or failure), even if parts of the system are down.
- Partition Tolerance (P): The system continues to operate despite network partitions, where nodes cannot communicate with each other.

According to the CAP theorem, a distributed system can provide only two out of these three guarantees at the same time: CP Systems: Prioritize consistency and partition tolerance but may sacrifice availability during network issues. AP Systems: Prioritize availability and partition tolerance but may have weaker consistency guarantees. CA Systems: Prioritize consistency and availability but lack partition tolerance, which limits scalability.

5b. What is peer-to-peer replication, and how is it different from master-slave replication? Give the diagram.

Ans. The master-slave model is a type of replicated database model where one server (the master) is designated to handle all data updates, while one or more slave servers replicate data from the master. The slaves provide read-only access to data, which helps balance the read load. Updates are directed only to the master, which then propagates changes to the slaves.

Characteristics:

- Centralized Write Operations: Only the master can handle write operations.
- Read Scalability: Allows multiple read-only replicas, providing read scalability.
- Consistency Challenges: Ensuring that all slaves are synchronized with the master requires consistent replication mechanisms.

In a peer-to-peer (P2P) model, each node in the network is both a server and a client, meaning all nodes can read and write data, and there is no single master server. Each node communicates and synchronizes directly with others, distributing the data management and avoiding a central point of control.

Characteristics:

- Decentralized: No master node; each node is both a client and a server.
- High Fault Tolerance: If one node fails, others can continue operations without dependency on a central authority.
- Complex Consistency Management: Managing data consistency and coordination across multiple writable nodes can be challenging.

Q6. Define version stamps. Explain briefly about various approaches of constructing version stamp.

Ans. Version stamps (or vector clocks) are a method used in distributed databases to track changes to data across different replicas or nodes. They help manage conflicting updates by recording the version of each change, allowing the system to resolve conflicts intelligently. When data is updated on a node, the version stamp is incremented, typically using a vector clock. Each replica tracks the latest version of the data it has seen.

1. Timestamps

Timestamps are a straightforward approach where each update is associated with a specific time of the update. The system uses timestamps to resolve conflicts by accepting the most recent update as the correct version. Each node attaches a timestamp to each update. When a conflict is detected, the system compares the timestamps and accepts the version with the latest timestamp.

Advantages: Simple to implement and requires minimal storage.

Disadvantages: Prone to conflicts if clocks aren't synchronized. Requires strict clock synchronization across nodes to avoid issues.

2. Counters

Counters use a numeric value to represent the order of updates, where each successive update increments the counter. Each version of the data has a counter, which is incremented on each update. When nodes replicate, the update with the highest counter is considered the latest.

Advantages: Simple to implement; avoids clock synchronization issues.

Disadvantages: Limited in peer-to-peer systems without a master node. If multiple nodes write simultaneously, conflicts can still arise.

3. Vector Clocks

Vector clocks are a more complex approach that tracks the causal history of updates across multiple nodes by maintaining a list of counters, one for each node. Each node maintains a separate counter for each other node, incrementing its counter whenever it makes an update. When conflicts arise, the system compares the vectors to determine causality. Each node maintains a vector of counters. Every time a node makes an update, it increments its own counter. When updates are replicated to other nodes, they compare vector clocks to determine which updates are more recent or causally related.

Advantages: Captures causal relationships between updates, allowing for sophisticated conflict resolution.

Disadvantages: Vector clocks grow with the number of nodes, adding overhead. Comparisons can become computationally complex in large clusters.

4. Hybrid Timestamps (Hybrid Logical Clocks)

Hybrid timestamps combine physical time (from clocks) and logical time (Lamport or vector clocks) to provide high accuracy without requiring perfect clock synchronization. This combines a physical timestamp and a logical counter, ensuring that causally related events are ordered correctly, even in the presence of slight clock drifts. Hybrid clocks use a physical timestamp for rough ordering, and a logical component (like a counter or vector clock) to adjust for causality. If events occur closely in time, the logical component ensures that causally related events are still ordered correctly.

Advantages: Provides accurate ordering without strict clock synchronization, reducing the chances of conflicts.

Disadvantages: More complex to implement; hybrid clocks add computational and storage overhead.