1.

a. Explain THREE (3) possible benefits of denormalization and THREE (3) possible disadvantages of denormalization.

Denormalization is the process of adding redundancy to a database schema, typically to improve query performance. Here are three potential benefits of denormalization:

1. Improved query performance: By denormalizing a database schema, you can eliminate the need for expensive join operations, which can improve the performance of queries that retrieve data from multiple tables.
2. Simplified schema design: Denormalization can help to simplify the schema design of a database, making it easier to understand and work with.
3. Enhanced data consistency: In some cases, denormalization can help to improve data consistency by reducing the risk of data inconsistencies that can occur when data is stored in multiple tables.

However, there are also potential disadvantages to denormalization:

1. Increased data redundancy: Denormalization typically involves adding redundancy to a database schema, which can increase the amount of storage space required to store the data.
2. Complexity of updates: Denormalized schemas can be more complex to update, as changes to data in one table may need to be propagated to multiple tables.
3. Increased risk of data inconsistencies: By adding redundancy to a database schema, there is an increased risk of data inconsistencies occurring if the redundant data is not kept in sync with the primary data.

b. Describe THREE (3) advantages and THREE (3) disadvantages of NoSQL databases.

NoSQL databases are a type of database that are designed to handle large amounts of data that is distributed across a wide range of servers. Here are three advantages of NoSQL databases:

1. Horizontal scaling: NoSQL databases are designed to scale horizontally, which means that they can easily handle increases in data volume and traffic by adding more servers to the database cluster. This makes them well-suited for use in distributed systems and for handling large amounts of data.
2. Flexibility: NoSQL databases are typically more flexible than traditional relational databases, as they do not require a fixed schema and can handle unstructured data. This makes them well-suited for storing data with complex and evolving relationships.

3. High performance: NoSQL databases are designed to handle high levels of read and write throughput, making them well-suited for use in high-performance systems that require fast data access.

However, there are also some potential disadvantages to using NoSQL databases:

1. Limited query capabilities: NoSQL databases generally do not support the same range of sophisticated query capabilities as traditional relational databases, which can make it more difficult to perform complex data analysis tasks.
2. Lack of ACID transactions: NoSQL databases typically do not support ACID (Atomicity, Consistency, Isolation, Durability) transactions, which can make it more difficult to ensure data consistency and integrity in certain situations.
3. Lack of support for joins: NoSQL databases typically do not support joins, which can make it more difficult to combine data from different sources or to perform certain types of data analysis tasks.

C. Suppose that you are tasked with implementing a distributed data solution for a retail
enterprise who wishes to achieve the following:
a. Store details of customers, their accounts, transactions
against these
accounts ensuring that all data is secure and consistent. The enterprise operates in several global regions. Customers are associated with a particular global region.
b. Implement a chat utility for customer support which is available 24 x 7. Chat
participants need to be able to view a full thread of chat conversations.
c. Explore expansion into new regional markets, predicting expected profit
levels, potential challenges to stock management using not only data owned
by the retail enterprise but external sources such as regional regulatory information, taxation etc.
For each of the above, state whether you would implement a SQL or NoSQL solution.
Justify your answer.

Here are my recommendations for implementing a distributed data solution for the given requirements:

1. For storing customer, account, and transaction details in a secure and consistent manner, I would recommend using a SQL solution. A relational

database management system (RDBMS) such as MySQL or PostgreSQL would be well-suited for this task, as it would allow you to define a schema for the data and enforce data integrity through the use of foreign keys and other constraints. Additionally, RDBMSs support ACID transactions, which can help to ensure the consistency and integrity of the data even in the face of concurrent updates.

2. For implementing a chat utility that is available 24x7, I would recommend using a NoSQL solution. A document-oriented database such as MongoDB or Couchbase would be well-suited for this task, as it would allow you to store chat conversations as documents and retrieve them quickly and efficiently. Additionally, NoSQL databases are generally more scalable than RDBMSs, which would be beneficial for handling high levels of traffic.
3. For exploring expansion into new regional markets and predicting expected profit levels, I would recommend using a combination of SQL and NoSQL solutions. A SQL solution such as an RDBMS could be used to store and manage the internal data owned by the retail enterprise, while a NoSQL solution such as a graph database (such as Neo4j) could be used to store and analyze external data sources such as regional regulatory information and taxation data. This would allow you to perform complex data analysis tasks and explore relationships between different data sources.

2.

a.

i. Explain FOUR (4) key characteristics of a data warehouse

A data warehouse is a centralized repository of data that is used for reporting, analysis, and decision-making. Here are four key characteristics of a data warehouse:

1. Subject-oriented: A data warehouse is designed to support analysis of data from a particular subject area, such as sales, finance, or marketing. The data in a data warehouse is typically organized around specific business subjects and is designed to support the needs of decision-makers in that subject area.
2. Integrated: A data warehouse integrates data from a wide range of sources, including transactional databases, log files, and external data sources. The data is cleaned, transformed, and integrated into a consistent format to support analysis and reporting.
3. Time-variant: A data warehouse stores historical data, allowing users to analyze trends and changes over time. The data in a data warehouse is typically organized by time period, such as by day, week, month, or year.
4. Non-volatile: A data warehouse is a stable and permanent store of data. Once data is loaded into a data warehouse, it is typically not updated or deleted, allowing users to rely on the data for accurate analysis and reporting.

ii. Briefly compare a data warehouse and relational DBMS considering data
design, data structure and access pattern.

A data warehouse and a relational database management system (RDBMS) are both systems for storing and managing data, but they have some key differences in terms of data design, data structure, and access patterns.

1. Data design: A data warehouse is typically designed to support reporting and analysis, while an RDBMS is designed to support transactional processing. As a result, the data design of a data warehouse is typically more focused on supporting business intelligence and decision-making, while the data design of an RDBMS is focused on ensuring data integrity and consistency.
2. Data structure: A data warehouse typically uses a dimensional data model, which organizes data into fact tables (which contain measurements or metrics) and dimension tables (which describe the context of the measurements). An RDBMS typically uses a relational data model, which organizes data into tables with rows and columns and uses foreign keys to establish relationships between tables.
3. Access patterns: A data warehouse is typically accessed for reporting and analysis purposes, which means that the access patterns are more predictable and the queries are typically more complex and resource-intensive. An RDBMS is accessed for transactional processing, which means that the access patterns are more varied and the queries are typically simpler and less resource-intensive.

b.

i. Explain the ACID and BASE transaction models and when you would use
each.

ACID and BASE are two different models for managing transactions in a database.

ACID stands for Atomicity, Consistency, Isolation, and Durability, and it is a set of properties that describe the behavior of transactions in a database. In the ACID model, transactions are atomic, meaning that they are either fully completed or fully rolled back, and they do not leave the database in an intermediate state. Transactions are also consistent, meaning that they maintain the integrity of the database by only allowing valid data to be written to the database. Transactions are isolated, meaning that they do not interfere with each other, and they are durable, meaning that they are persisted to the database and are not lost in the event of a failure.

BASE stands for Basically Available, Soft state, Eventual consistency, and it is a model for managing transactions in a database that is designed to support distributed systems and high levels of concurrency. In the BASE model, transactions are designed to be available most of the time, even if the database is not fully consistent. Transactions are soft, meaning that they can be overwritten or rolled back, and the database may be in a temporarily inconsistent state. Transactions are eventually consistent, meaning that the database will eventually converge to a consistent state, although it may take some time.

You would typically use the ACID model when you need to ensure strong consistency and data integrity, such as when you are storing financial transactions or other critical data. You would use the BASE model when you need to support high levels of concurrency and availability, such as when you are building a distributed system or a system with high write throughput.

ii. Consider the following scenarios:
a. Departmental managers in a retail company want to identify buying patterns
of individual customers and different types of customers, analyse the impact
of special sales promotions and determining future pricing policy for different
products.
b. A small marketing company wants to store data from social networks and
conduct sentiment analysis on this data to explore the impact of its marketing
campaigns, in particular involving TV advertising during prime time. Analysis
will be differentiated between weekday and weekend sentiment.
For each, state whether you consider the ACID or BASE transaction model most
suitable.

For the following scenarios, I would recommend the following transaction models:

1. For the retail company wanting to identify buying patterns and analyze the impact of special sales promotions, I would recommend using the ACID transaction model. This is because the analysis of customer buying patterns

and the determination of pricing policy are critical tasks that require strong consistency and data integrity. The ACID model would ensure that the data is accurately captured and that the analysis is based on reliable data.

2. For the small marketing company wanting to store data from social networks and conduct sentiment analysis, I would recommend using the BASE transaction model. This is because the analysis of sentiment is not a critical task that requires strong consistency and data integrity, and the BASE model would allow the marketing company to scale the system to handle high levels of concurrency and write throughput. Additionally, the BASE model would allow the marketing company to perform sentiment analysis in real-time, which would be useful for analyzing the impact of marketing campaigns in real-time.

3.

a.

i. Explain what a secondary index is and its purpose in a data store

A secondary index is an index that is created on a database table or other data structure in addition to the primary index. The primary index is typically used to look up rows in the table based on their primary key value, while the secondary index can be used to look up rows based on other values in the table.

The purpose of a secondary index is to improve the performance of queries that filter or sort data based on values other than the primary key. For example, if you have a table of customer records and you want to find all customers in a particular city, you could use a secondary index on the city column to speed up the query. Similarly, if you want to sort the customer records by last name, you could use a secondary index on the last name column to speed up the sorting process.

Secondary indexes can be created on single columns or on multiple columns, and they can be either unique or non-unique. They can also be created using different indexing algorithms, such as B-trees or hash tables, depending on the needs of the application.

ii. Suppose you are trying to improve query performance in a SQL data store
and in a NonSQL data store. Should you add multiple indexes to the tables
involved? Justify your answer.

In general, adding multiple indexes to a table can improve query performance, but it can also have some drawbacks. Here are some considerations for adding multiple indexes to a SQL data store and a NoSQL data store:

1. SQL data store: In a SQL data store, adding multiple indexes can improve query performance by providing additional ways to look up rows in the table. However, adding too many indexes can have a negative impact on performance, as it can increase the size of the database and make it more expensive to update the data. Additionally, adding multiple indexes can make it more difficult to optimize queries, as the database management system (DBMS) has to choose which index to use for each query.
2. NoSQL data store: In a NoSQL data store, adding multiple indexes can also improve query performance by providing additional ways to look up data. However, the impact on performance will depend on the specific NoSQL database being used and the type of index being created. Some NoSQL databases are designed to support a large number of indexes, while others are more limited in the number of indexes that can be created. As with a SQL data store, adding too many indexes can have a negative impact on performance and make it more difficult to optimize queries.

In general, it is important to carefully consider the trade-offs between query performance and the overhead of maintaining multiple indexes before adding them to a data store. It may be necessary to perform testing to determine the optimal number of indexes for a given workload.

b. Suppose you are implementing a chat system and are designing the data store for a
collection of messages. Each message has an author name, recipient name, content,
sequence number and timestamp.
Explain how you would implement a secondary index to facilitate pattern matching
on chat content in each of the following:
• PostgreSQL
• MongoDB
• Apache Cassandra
You are not required to write any code.
In your answer you must explain:
• The most appropriate type of index
• The potential disadvantages

Here is how I would implement a secondary index to facilitate pattern matching on chat content in each of the following data stores:

1. PostgreSQL: In PostgreSQL, I would create a full-text search index using the `to_tsvector()` function and the `tsvector` data type. This index would allow me to perform fast pattern matching on the chat content using the `@@` operator. For example, I could create the index as follows:

   ```
   CREATE INDEX chat_content_idx ON messages USING gin (to_tsvector(content));
   ```

   Then, I could use the following query to search for messages containing the pattern "hello world":

   ```
   SELECT * FROM messages WHERE to_tsvector(content) @@ 'hello & world';
   ```

   The disadvantage of using a full-text search index in PostgreSQL is that it can be slower to update than other types of indexes, as it requires the index to be rebuilt whenever the data changes.
2. MongoDB: In MongoDB, I would create a text index using the `$text` operator. This index would allow me to perform fast pattern matching on the chat content using the `$search` operator. For example, I could create the index as follows:

   ```
   db.messages.createIndex({ content: "text" });
   ```

   Then, I could use the following query to search for messages containing the pattern "hello world":

   ```
   db.messages.find({ $text: { $search: "\"hello world\"" } });
   ```

   The disadvantage of using a text index in MongoDB is that it can be slower to update than other types of indexes, as it requires the index to be rebuilt whenever the data changes.
3. Apache Cassandra: In Apache Cassandra, I would create a secondary index using the `CREATE INDEX` statement. This index would allow me to perform fast pattern matching on the chat content using the `LIKE` operator. For example, I could create the index as follows:

   ```
   CREATE INDEX chat_content_idx ON messages (content);
   ```

   Then, I could use the following query to search for messages containing the pattern "hello world":

SELECT * FROM messages WHERE content LIKE '%hello world%';

The disadvantage of using a secondary index in Apache Cassandra is that it can increase the amount of storage space required to store the data, as it requires additional data to be written to the index. Additionally, secondary indexes in Cassandra are not automatically updated when the data changes, so they may not always reflect the current state of the data.

c. Suppose you are implementing a chat system and are designing the data store for a
collection of messages. Each message has an author name, recipient name, content,
sequence number and timestamp.

i. Other than using indexes, explain TWO (2) approaches which could be used
to improve performance in a document NoSQL database and TWO (2)
approaches which could be used in a distributed NoSQL wide column
database

Here are two approaches that could be used to improve performance in a document NoSQL database, such as MongoDB:

1. Sharding: Sharding is a technique for distributing data across multiple servers in a database cluster. By distributing the data across multiple servers, it is possible to scale the database horizontally to handle increases in traffic and data volume. Sharding can improve performance by allowing the database to spread the load across multiple servers and by allowing queries to be executed in parallel on different servers.
2. Caching: Caching is a technique for storing frequently accessed data in memory to improve access times. By storing data in memory, it is possible to reduce the number of disk I/O operations required to retrieve the data, which can significantly improve performance. Caching can be particularly useful for read-heavy workloads, as it can reduce the number of times that the database needs to access the disk to retrieve data.

Here are two approaches that could be used to improve performance in a distributed NoSQL wide column database, such as Apache Cassandra:

1. Replication: Replication is a technique for storing multiple copies of data across different servers in a database cluster. By storing multiple copies of data, it is possible to improve the availability and reliability of the database, as

well as the performance of read queries. Replication can improve performance by allowing read queries to be executed on any of the replica servers, which can reduce the load on the primary server and improve read performance.
2. Data partitioning: Data partitioning is a technique for dividing data across different servers in a database cluster based on a partition key. By dividing the data in this way, it is possible to distribute the load across multiple servers and improve the performance of both read and write queries. Data partitioning can improve performance by allowing the database to scale horizontally and by allowing queries to be executed in parallel on different servers.

ii. Provide examples of situations in which you would use each approach.

Sharding:
When the data volume or traffic levels are too high for a single server to handle effectively

When you need to scale the database to handle increases in traffic or data volume

When you need to improve the performance of read queries by allowing them to be executed in parallel on different servers

Caching

- When the data is accessed frequently and the performance of read queries is critical

Data partitioning

For example, if the chat system is being used by a large number of users and the database is struggling to keep up with the volume of messages being generated, data partitioning could be used to distribute the data across multiple servers and improve performance.

Data replication:

For example, if the chat system is being used by a large number of users and the database is struggling to keep up with the volume of read queries being generated, data replication could be used to store multiple copies of the data and improve read performance. Additionally, data replication could be used to ensure that the data is available in the event of a server failure or other issue.

4.

a. What is partitioning?
How can vertical and horizontal partitioning be used during the physical

data design
phase to improve performance?

Partitioning is a technique for dividing data across multiple servers or storage devices in a database or other data store. The goal of partitioning is to improve the performance, scalability, and availability of the data store by distributing the load and data across multiple servers or devices.

There are two main types of partitioning: vertical partitioning and horizontal partitioning.

1. Vertical partitioning: Vertical partitioning is a technique for dividing a table into multiple tables based on the columns in the table. For example, if a table contains customer data with columns for name, address, and purchase history, vertical partitioning could be used to create separate tables for each of these columns. This can improve performance by reducing the amount of data that needs to be accessed for each query and by allowing queries to be executed in parallel on different servers or devices.
2. Horizontal partitioning: Horizontal partitioning is a technique for dividing a table into multiple tables based on the rows in the table. For example, if a table contains customer data with millions of rows, horizontal partitioning could be used to create separate tables for different ranges of rows. This can improve performance by reducing the amount of data that needs to be accessed for each query and by allowing queries to be executed in parallel on different servers or devices.

Both vertical and horizontal partitioning can be used during the physical data design phase to improve performance by reducing the amount of data that needs to be accessed for each query and by allowing queries to be executed in parallel on different servers or devices. However, it is important to carefully consider the trade-offs between the benefits of partitioning and the overhead of maintaining and querying multiple partitions.

b. Explain the difference between partition and replication.

Partitioning is used to divide data across multiple servers or devices, while replication is used to store multiple copies of data across different servers or devices. Both techniques can improve performance, scalability, and availability, but they are used for different purposes and have different trade-offs. Partitioning can improve the performance of both read and write queries by allowing them to be executed in parallel on different servers or devices, but it can also increase the complexity of the data store and require additional overhead to maintain and query the partitions. Replication can improve the availability and reliability of the data store by storing multiple copies of the

data, but it can also increase the storage requirements and the complexity of the data store.

c.

i. Explain the CAP theorem.

The CAP theorem, also known as Brewer's theorem, is a theoretical result in computer science that states that it is impossible for a distributed data store to simultaneously provide more than two of the following three guarantees:

1. Consistency: Consistency refers to the property of a distributed data store that ensures that all nodes in the system see the same data at the same time. This means that if a write operation is performed on one node, it should be immediately visible to all other nodes in the system.
2. Availability: Availability refers to the property of a distributed data store that ensures that all nodes in the system are able to respond to read and write requests at all times. This means that the data store should be able to handle read and write requests even if some nodes in the system are unavailable or experiencing failures.
3. Partition tolerance: Partition tolerance refers to the ability of a distributed data store to continue functioning even if some nodes in the system are unable to communicate with each other. This can occur due to network failures, hardware failures, or other issues.

According to the CAP theorem, it is impossible for a distributed data store to provide all three of these guarantees simultaneously. This means that designers of distributed data stores must choose which two of these guarantees they want to prioritize and design their systems accordingly.

ii. Why is the PACELC extension important?

The PACELC theorem is an extension of the CAP theorem that adds the consideration of the network latency between nodes in a distributed data store. The acronym PACELC stands for "Partition Availability Consistency Elasticity Latency Capability."

The PACELC theorem states that in a distributed data store, the choice between consistency and availability depends on the relative values of partition tolerance (P), availability (A), consistency (C), elasticity (E), latency (L), and capability (C).

The importance of the PACELC extension is that it recognizes that the trade-offs between consistency and availability in a distributed data store are not fixed, but rather depend on the specific characteristics of the system. In

particular, the PACELC theorem acknowledges that the relative importance of consistency and availability can depend on the network latency between nodes in the system, as well as other factors such as the elasticity of the system (i.e., its ability to handle changes in load) and its capability (i.e., its processing power and other resources).

By considering these factors, the PACELC theorem provides a more nuanced view of the trade-offs between consistency and availability in a distributed data store, and can help designers make more informed decisions about how to design their systems to meet their specific needs and requirements.

d. Suppose that you are tasked with implementing a distributed data solution for a retail
enterprise who wishes to achieve the following:
a. Store details of customers, their accounts, transactions against these accounts ensuring that all data is secure and consistent. Online retail applications must be available 24 x 7.
b. Implement a chat utility for customer support which is available 24 x 7. Chat
participants need to be able to view a full thread of chat conversations.
c. Explore expansion into new regional markets, predicting expected profit
levels, potential challenges to stock management using not only data owned
by the retail enterprise but external sources such as regional regulatory information, taxation etc.

For the first requirement, it would be important to prioritize consistency in the distributed data solution. Ensuring that all data is secure and consistent is critical for a retail enterprise, as it ensures that customers can trust the accuracy and reliability of the data being presented to them. To achieve this, a distributed data solution that is designed to prioritize consistency, such as a database that uses a strong consistency model, would be appropriate.

For the second requirement, it would be important to prioritize availability in the distributed data solution. Ensuring that the chat utility is available 24x7 is critical for customer support, as it allows customers to communicate with the retail enterprise at any time. To achieve this, a distributed data solution that is designed to prioritize availability, such as a database that uses an eventually consistent model, would be appropriate.

For the third requirement, it may be necessary to prioritize both consistency and availability in the distributed data solution. Predicting expected profit levels and identifying potential challenges to stock management requires accurate and reliable data, which would be best achieved with a strong consistency model. However, it may also be necessary to ensure that the data is available 24x7 in order to support the expansion into new regional markets. In this case, it may be necessary to find a balance between consistency and availability in the distributed data solution. One possibility might be to use a database that uses a hybrid consistency model, which allows for a trade-off between consistency and availability.

iii. Discuss the implications of each property of the CAP theorem for each scenario above and for each identify which properties are most important. Justify your answer.

For the first requirement of storing customer data, consistency is the most important property of the CAP theorem. Ensuring that all data is secure and consistent is critical for a retail enterprise, as it ensures that customers can trust the accuracy and reliability of the data being presented to them. In this case, it would be important for the distributed data solution to prioritize consistency over availability and partition tolerance.

For the second requirement of implementing a chat utility, availability is the most important property of the CAP theorem. Ensuring that the chat utility is available 24x7 is critical for customer support, as it allows customers to communicate with the retail enterprise at any time. In this case, it would be important for the distributed data solution to prioritize availability over consistency and partition tolerance.

For the third requirement of expanding into new regional markets, both consistency and availability are important properties of the CAP theorem. Predicting expected profit levels and identifying potential challenges to stock management requires accurate and reliable data, which would be best achieved with a strong consistency model. However, it may also be necessary to ensure that the data is available 24x7 in order to support the expansion into new regional markets. In this case, it would be important for the distributed data solution to find a balance between consistency and availability, and possibly to prioritize partition tolerance as well in order to ensure that the data store can continue functioning even if some nodes are unable to communicate with each other.

i.