2014-2015

Q2. (a) Present a case where programmatic optimization is preferred over systematic optimization.

Programmatic optimization: Programmatic optimization is a technique in which the programmer writes code to optimize the performance of a program.

Systematic optimization: while systematic optimization refers to the process of identifying and addressing bottlenecks or inefficiencies in a system as a whole.

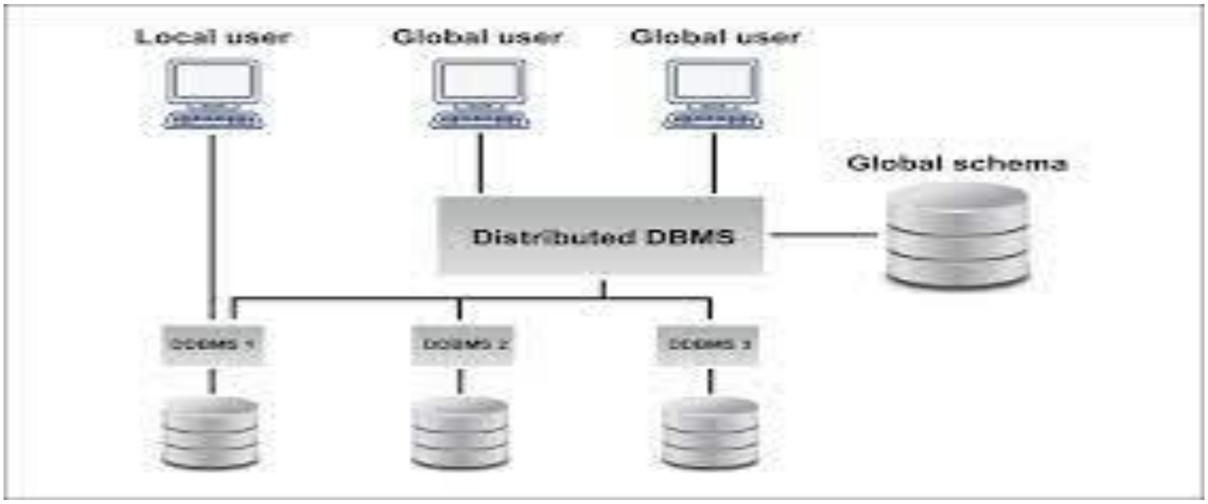
There are several situations where programmatic optimization may be preferred over systematic optimization:

- **Time constraints:** Programmatic optimization can often be completed more quickly than systematic optimization, especially if the programmer is familiar with the codebase and the specific performance issues that need to be addressed. This can be particularly useful in situations where time is of the essence, such as when the program is needed for a time-sensitive project
- **Complex systems:** Systematic optimization can be more challenging in complex systems that involve many interconnected components and subsystems. In such cases, programmatic optimization may be a more efficient way to identify and address performance issues, as it allows the programmer to focus on specific areas of the code rather than trying to analyze the entire system.
- **Customization:** Programmatic optimization allows for more flexibility and customization, as the programmer has complete control over the specific optimizations that are implemented. This can be particularly useful in cases where the program needs to be tailored to the specific needs of the user or application.
- **Limited resources:** Systematic optimization often requires additional resources, such as specialized tools or personnel, which may not be available in all situations. Programmatic optimization, on the other hand, can be performed with just the code and a text editor, making it a more accessible option for resource-constrained environments.

Use case: One example of a use case where programmatic optimization may be preferred over systematic optimization is in the development of a custom application for a small business.
Imagine that a small business owner has hired a developer to create a custom application to manage their inventory and order processing. The application needs to be completed in a short amount of time, as the business owner is eager to start using it as soon as possible.
In this case, the developer may choose to focus on programmatic optimization in order to get the application up and running as quickly as possible. They may write code to optimize specific areas of the application, such as the database queries or the algorithms used to process orders, in order to improve the overall performance of the application.
This approach allows the developer to complete the project quickly and efficiently, while still ensuring that the application performs well for the small business owner. It may not be feasible to conduct a full-scale systematic optimization of the application in the limited time available, so programmatic optimization is the best option in this case.

Q2. (b) Discuss the advantages and disadvantages of distributing a database? (see notes too for this, week 2)

Distributed Database: A distributed database is a database that is spread across multiple servers, as opposed to a traditional, centralized database that is stored on a single server. Distributed databases are designed to improve performance, scalability, and availability by allowing data to be stored and accessed in multiple locations. This can also make it more resistant to failures and outages, as data can still be accessed even if one server or location goes down.



Advantages of Distributed Database:

Distributing a database refers to the practice of storing and managing data across multiple servers, rather than on a single machine. There are several advantages to distributing a database:

- **Increased scalability**: By distributing the database across multiple servers, it is possible to handle larger volumes of data and more concurrent users without performance degradation. This makes it easier to scale the database as the needs of the application grow.
- **Improved availability:** Distributing the database across multiple servers can also improve the availability of the data, as it reduces the risk of a single point of failure. If one server goes down, the data can still be accessed from the other servers.
- **Enhanced security:** Storing data on multiple servers can also increase security, as it makes it more difficult for a hacker to gain access to all of the data at once.

Disadvantages of Distributed Database:

However, there are also some disadvantages to distributing a database:

- **Increased complexity:** Managing a distributed database can be more complex than managing a single server database, as it requires coordinating updates and queries across multiple machines.
- **Higher costs:** Distributing a database typically requires additional hardware and software, which can increase the overall cost of the system.
- **Reduced performance:** In some cases, distributing a database can result in reduced performance, as data may need to be transmitted between servers in order to complete certain queries. This can be mitigated through the use of techniques such as database sharding and indexing, however.

Against decentralized database Advantages
Increased reliability/availability, Local control over data

Software cost and complexity • Processing overhead • Slower response for certain queries

[redacted]Present a scenario of an oranganization where distribution of data can be beneficial for a company and then analyze through components of distributed query optimization.

Scenario: An organization that operates multiple retail stores in different locations can benefit from distributing data across all of its stores. By having access to real-time data on sales, inventory levels, and customer demographics, store managers can make more informed decisions on everything from product ordering and stocking to staffing and promotions. For example, if a store in a certain location is consistently selling more of a certain product than other stores, that store's manager can order more of that product to keep it in stock, and other store managers can do the same if they see similar trends in their own store's data. Additionally, having access to data on customer demographics can help managers tailor their marketing and promotions to better target their specific customer base.

(handmade diagram for this scenario)

Components of distributed query optimization:

There are three components of distributed query optimization:

- **Access Method:**
  In order to access and distribute data across all retail stores, the organization can use a centralized database system. The system can be accessed by each store through the internet using a secure connection (such as a virtual private network, or VPN). This will allow the store managers to access the data in real-time, from any location.
- **Transmission Cost:**
  The cost of transmitting data across the network will depend on several factors, including the size of the data, the number of stores that need to access the data, and the speed of the network connection. To reduce transmission costs, the organization can consider using efficient data compression techniques and optimize the data transmission protocols.
- **Join Criteria:**
  Join criteria are the conditions that must be met in order to combine data from multiple sources into a single result. In the case of the retail stores, the join criteria can be based on the location of the stores, the type of product being sold, or the demographic data of the customers. For example, the data from all stores can be joined based on the sales data for a specific product, allowing the managers to compare sales trends across different locations and make informed decisions about ordering and stocking.

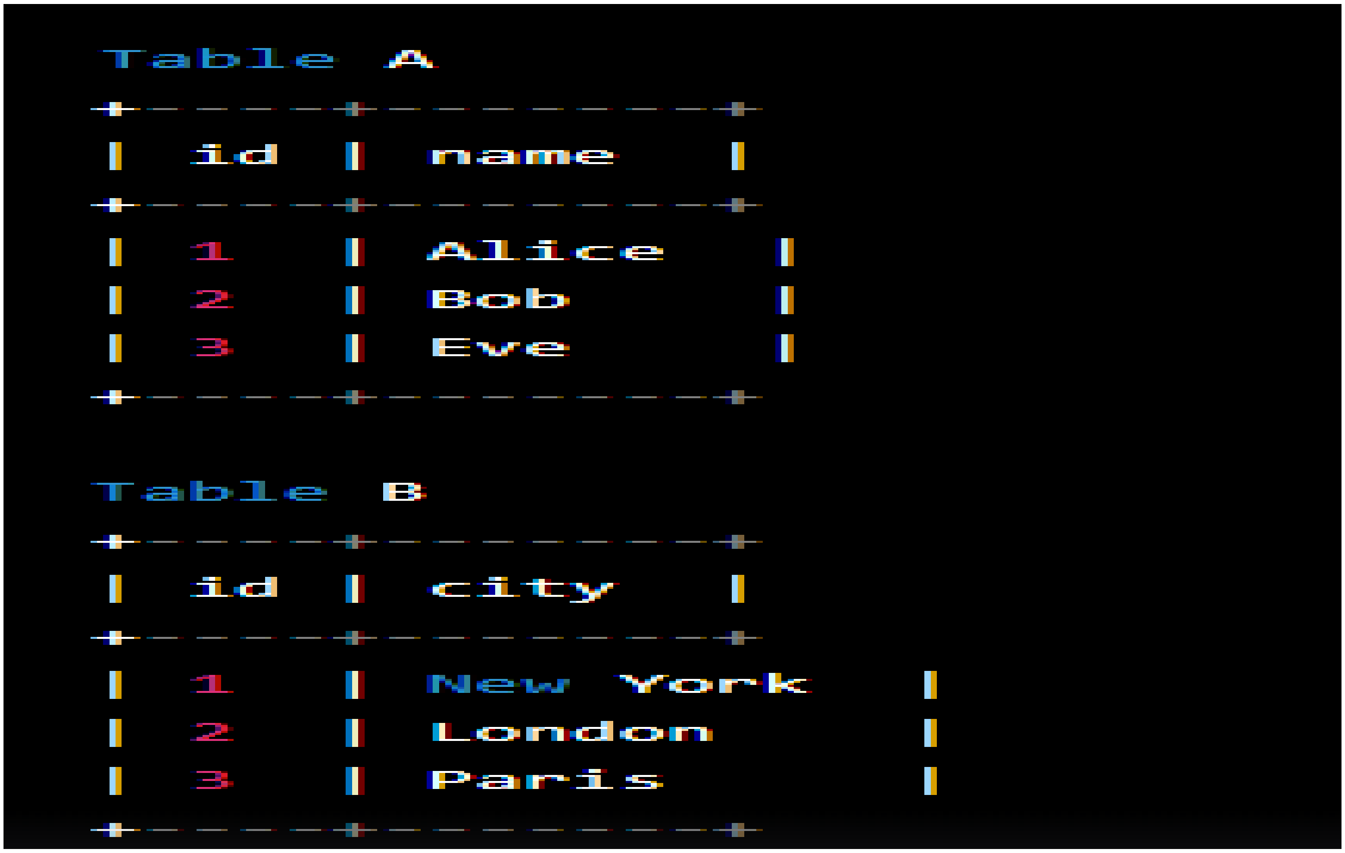
Overall, by distributing the data to multiple locations and using efficient join criteria, access methods, and minimizing the transmission cost, the retail company can improve its inventory management, customer service and improve their overall efficiency.

Q4. Compare merge scan and nested loop join with appropriate example? (tho sir excluded this topic)

A merge scan and a nested loop join are both algorithms that are used to perform a join operation in a database. A join operation combines rows from two or more tables based on a related column or set of columns.

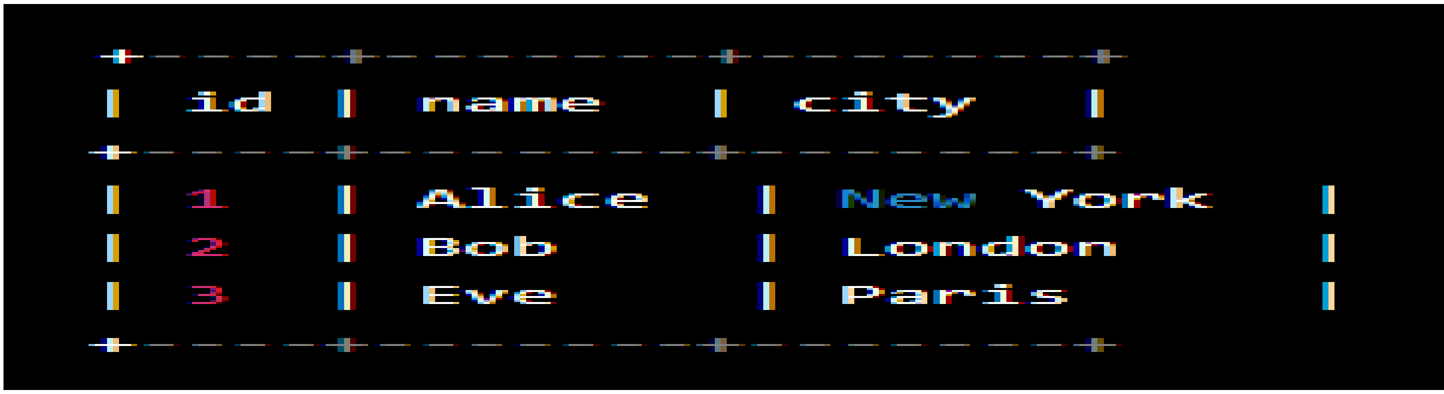
A merge scan is a type of join that processes the two tables being joined in sorted order. It works by reading the rows from each table and comparing them based on the join condition. When a match is found, the rows are combined and returned as the result of the join.

For example, consider the following two tables:



```
 Table A
+------+----------+
|  id  |  name    |
+------+----------+
|  1   |  Alice    |
|  2   |  Bob      |
|  3   |  Eve      |
+------+----------+

Table B
+------+----------+
|  id  |  city    |
+------+----------+
|  1   |  New York      |
|  2   |  London        |
|  3   |  Paris         |
+------+----------+
```

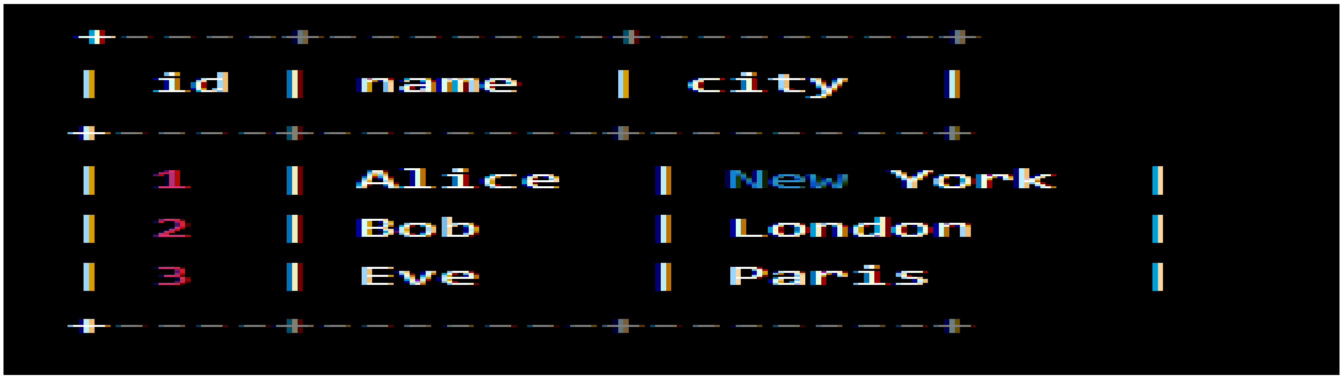
A merge scan join of these two tables might produce the following result:



```
+----+-------------+-------------+
| id | name        | city        |
+----+-------------+-------------+
| 1  | Alice       | New York    |
| 2  | Bob         | London      |
| 3  | Eve         | Paris       |
+----+-------------+-------------+
```

A nested loop join, on the other hand, processes the tables being joined by iterating through each row in one table and searching for a match in the other table. It works by reading each row from the outer table and then searching for a matching row in the inner table. If a match is found, the rows are combined and returned as the result of the join.

For example, a nested loop join of the tables above might produce the same result as the merge scan:



```
+----+-------------+-------------+
| id | name        | city        |
+----+-------------+-------------+
| 1  | Alice       | New York    |
| 2  | Bob         | London      |
| 3  | Eve         | Paris       |
+----+-------------+-------------+
```

Both merge scan and nested loop join have their own advantages and disadvantages. Merge scan is generally faster when both tables are already sorted and there are a large number of matching rows, as it can process the tables in a single pass. Nested loop join is more flexible, as it can handle a wider variety of join conditions and can be more efficient when there are relatively few matching rows.

Homogeneous	Heterogeneous
Every node in the distributed network has the same dbms	Different nodes has diff dbms in the network
e.g every node has an oracle database	e.g nodes have different dbms, hierarchical dbms relational dbms or oo dbms
Autonomous: each dbms works independently passing messages	Full: All nodes has the distributed functionality partial: Some nodes have the distributed functionality

Non-autonomous: a central or master node coordinates the messages across distributed database	
Level of synchronous is greater in homogenous	Level of synchronization are less
Easy to design and implement	Involves complex design and planning
It is used where there is a lot of sharing and common distribution	It is used where there is uncommon distribution and less sharing of data
Easier to manage	Difficult to manage
Difficult to enforce	Preferred by individual organizations

Q5. (a) Differentiate between homogenous and non-homogenous distributed

A homogenous distributed database is a database system in which all of the database servers are running the same software and use the same database management system (DBMS). This means that the database servers are able to communicate with each other and share data seamlessly, as they are all using the same database technology.

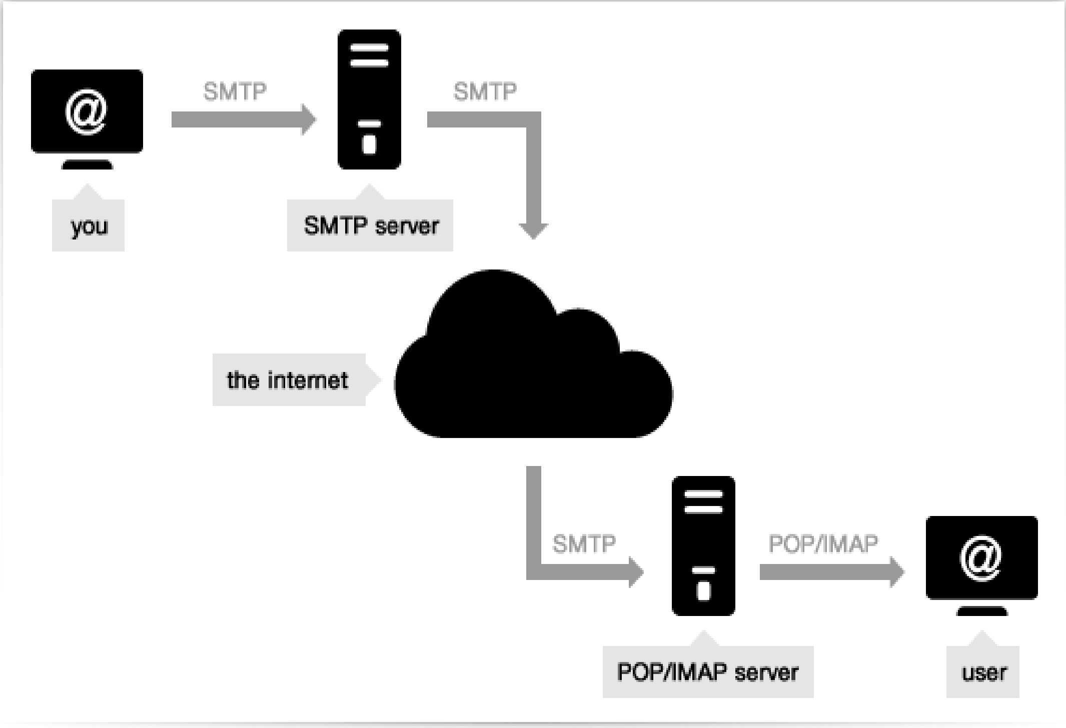
In contrast, a non-homogenous distributed database is a database system in which the database servers are running different software and/or using different DBMSs. This can make it more challenging to share data between the servers, as they may not be able to communicate with each other directly.

One example of a homogenous distributed database is a cluster of database servers all running the same version of MySQL. All of the servers in the cluster are able to communicate with each other and share data using the MySQL database management system.

On the other hand, a non-homogenous distributed database might consist of a mix of MySQL and PostgreSQL servers, for example. In this case, the servers running MySQL and PostgreSQL would need to use some sort of middleware or connector in order to share data, as they do not use the same database technology.

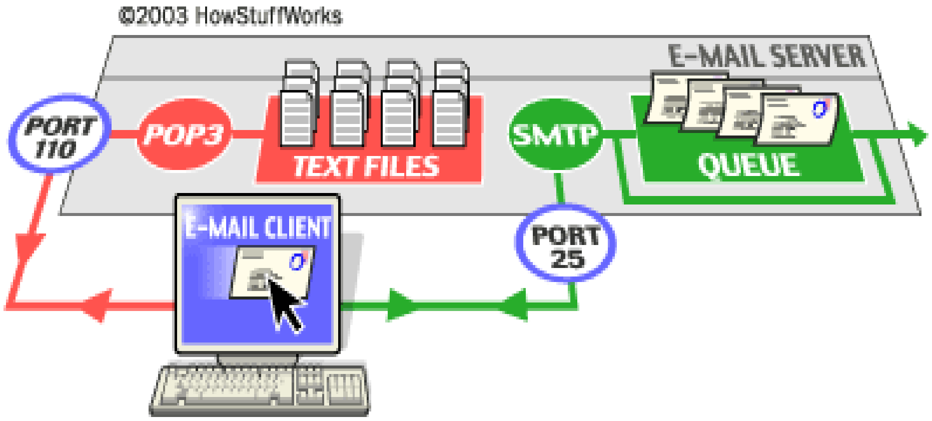
Homogenous distributed databases are generally easier to manage and maintain, as all of the servers are using the same technology and can communicate with each other directly. Non-homogenous distributed databases, on the other hand, can be more complex and may require additional tools or infrastructure in order to share data between the servers.

Q5. (b) Discuss the working of SMTP server with example.



SMTP Server

An e-mail server that uses the Simple Mail Transfer Protocol (SMTP). SMTP is a member of the TCP/IP suite of protocols that governs the exchange of electronic mail between message transfer agents.



Whenever you send a piece of e-mail, your e-mail client interacts with the SMTP server to handle the sending. The SMTP server on your host may have conversations with other SMTP servers to deliver the e-mail.

Let's assume that I want to send a piece of e-mail. My e-mail ID is brain, and I have my account on howstuffworks.com. I want to send e-mail to jsmith@mindspring.com. I am using a stand-alone e-mail client like Outlook Express.

When I set up my account at howstuffworks, I told Outlook Express the name of the mail server -- mail.howstuffworks.com. When I compose a message and press the Send button, here's what happens:

1. Outlook Express connects to the SMTP server at mail.howstuffworks.com using port 25.
2. Outlook Express has a conversation with the SMTP server, telling the SMTP server the address of the sender and the address of the recipient, as well as the body of the message.
3. The SMTP server takes the "to" address (jsmith@mindspring.com) and breaks it into two parts: the recipient name (jsmith) and the domain name (mindspring.com). If the "to" address had been another user at howstuffworks.com, the SMTP server would simply hand the message to the POP3 server for howstuffworks.com (using a little program called the **delivery agent**). Since the recipient is at another domain, SMTP needs to communicate with that domain.
4. The SMTP server has a conversation with a **Domain Name Server**, or **DNS**. It says, "Can you give me the IP address of the SMTP server for mindspring.com?" The DNS replies with the one or more IP addresses for the SMTP server(s) that Mindspring operates.
5. The SMTP server at howstuffworks.com connects with the SMTP server at Mindspring using port 25. It has the same simple text conversation that my e-mail client had with the SMTP server for HowStuffWorks, and gives the message to the Mindspring server. The Mindspring server recognizes that the domain name for jsmith is at Mindspring, so it hands the message to Mindspring's POP3 server, which puts the message in jsmith's mailbox.

If, for some reason, the SMTP server at HowStuffWorks cannot connect with the SMTP server at Mindspring, then the message goes into a queue. The SMTP server on most machines uses a program called **sendmail** to do the actual sending, so this queue is called the **sendmail queue**. Sendmail will periodically try to resend the messages in its queue. For example, it might retry every 15 minutes. After four hours, it will usually send you a piece of mail that tells you there is some sort of problem. After five days, most sendmail configurations give up and return the mail to you undelivered.

POP3 Server

POP stands for Post Office Protocol. This is used to describe how e-mail clients interact with mail servers. The POP3 Server is a type of mail server used for incoming mail. In simple terms, POP servers provide a mail-drop service (a temporary mailbox to leave messages so they can be picked up at the recipient's convenience.) When users connect to their ISP POP servers, their e-mail software interface with the server and download any messages for them. POP is only used to receive messages, it is not used to send them.

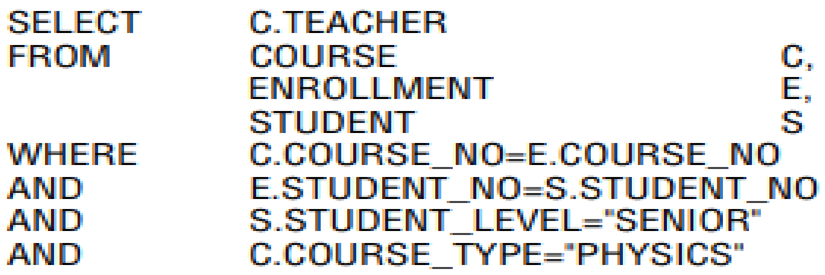
Q6. Discuss optimization example with different usecases

Optimization refers to the process of improving the performance of a system, program, or process. There are many different ways to optimize a system, and the specific approach will depend on the nature of the system and the goals of the optimization. Here are a few examples of optimization in different use cases:

In order to understand distributed query optimization more fully, let's take a look at an example of a query accessing tables in multiple locations. Consider the ramifications of coding a program to simply retrieve a list of all teachers who have taught physics to seniors. Furthermore, assume that the COURSE table and the ENROLLMENT table exist at Site 1; the STUDENT table exists at Site 2. If either all of the tables existed at a single site, or the DBMS sup- ported distributed multi-site requests, the SQL shown in Figure 1 would satisfy the requirements. However, if the DMBS can not per- form (or optimize) distributed multi-site requests, programmatic opti- mization must be performed. There are at least six different ways to go about optimizing this three-table join.
Option 1: Start with Site 1 and join COURSE and ENROLLMENT, selecting only physics courses. For each qualifying row, move it to Site 2 to be joined with STUDENT to see if any are seniors.
Option 2: Start with Site 1 and join COURSE and ENROLLMENT, selecting only physics courses, and move the entire result set to Site 2 to be joined with STUDENT, checking for senior students only.
Option 3: Start with Site 2 and select only seniors from STUDENT. For each of these examine the join of COURSE and ENROLLMENT at Site 1 for physics classes.
Option 4: Start with Site 2 and select only seniors from STUDENT at Site 2, and move the entire result set to Site 1 to be joined with COURSE and ENROLLMENT, checking for physics classes only.
Option 5: Move the COURSE and ENROLLMENT tables to Site 2 and proceed with a local three-table join.
Option 6: Move the STUDENT to Site 1 and proceed with a local three-table join.

Which of these six options will perform the best? Unfortunately, the
only correct answer is "It depends." The optimal choice will depend upon:
■ the size of the tables;
■ the size of the result sets — that is, the number of qualifying rows
and their length in bytes; and
■ the efficiency of the network.
Try different combinations at your site to optimize distributed
queries. But remember, network traffic is usually the cause of most
performance problems in a distributed environment. So devoting most
of your energy to options involving the least amount of network traffic is
a wise approach. In addition, bad design can also be the cause of many
distributed performance problems .

Figure 1: SQL to Satisfy Single Site or Multi-Site Requests



```
SELECT      C.TEACHER
FROM        COURSE                      C,
            ENROLLMENT                  E,
            STUDENT                     S
WHERE       C.COURSE_NO=E.COURSE_NO
AND         E.STUDENT_NO=S.STUDENT_NO
AND         S.STUDENT_LEVEL="SENIOR"
AND         C.COURSE_TYPE="PHYSICS"
```

Q7. Draw and explain the architecture of linked server and its creation through stored procedure.

A linked server configuration enables SQL Server to execute commands against OLE DB data sources on
remote servers. Linked servers offer the following advantages:

- Remote server access.
- The ability to issue distributed queries, updates, commands, and transactions on heterogeneous data sources across the enterprise.
- The ability to address diverse data sources similarly.

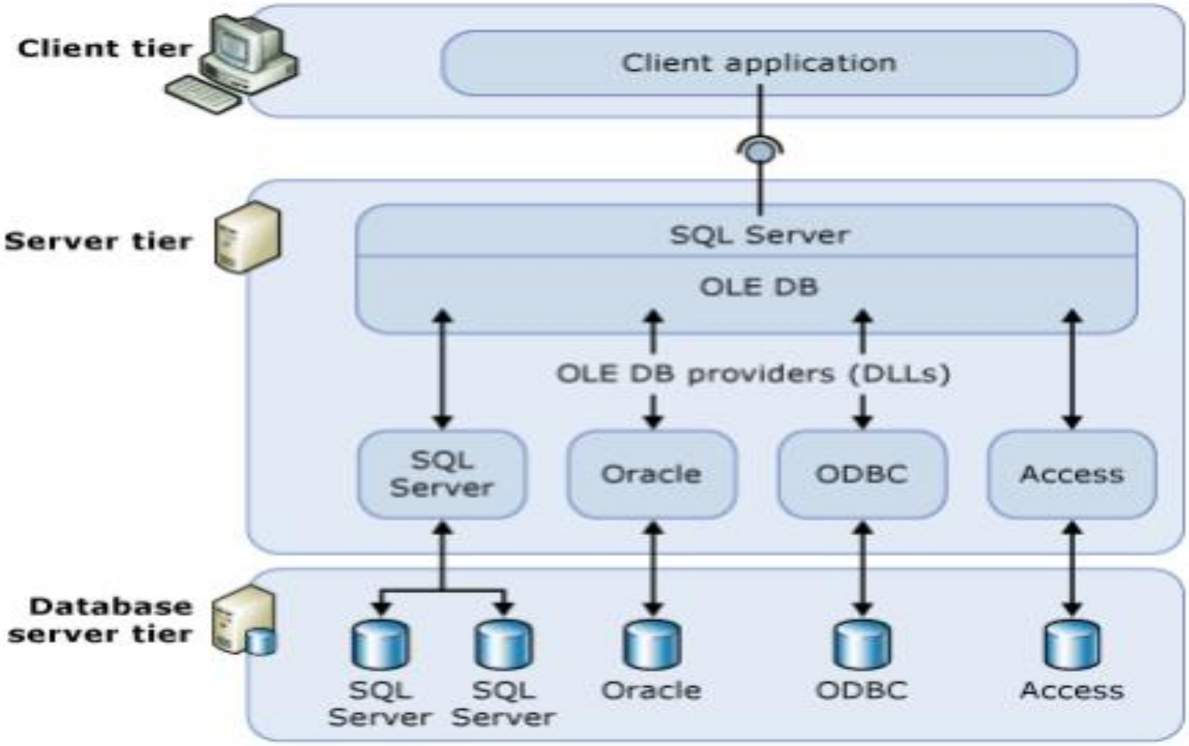
Linked Server Components

A linked server definition specifies the following objects:

- An OLE DB provider
- An OLE DB data source

An OLE DB provider is a DLL that manages and interacts with a specific data source. An OLE DB data source identifies the specific database that can be accessed through OLE DB. Although data sources queried through linked server definitions are ordinarily databases, OLE DB providers exist for a variety of files and file formats. These include text files, spreadsheet data, and the results of full-text content searches. The Microsoft SQL Native Client OLE DB Provider (PROGID: SQLNCLI) is the official OLE DB provider for SQL Server. Linked Server Details

The following illustration shows the basics of a linked server configuration.



Typically, linked servers are used to handle distributed queries. When a client application executes a distributed query through a linked server, SQL Server parses the command and sends requests to OLE DB. The rowset request may be in the form of executing a query against the provider or opening a base table from the provider.

For a data source to return data through a linked server, the OLE DB provider (DLL) for that data source must be present on the same server as the instance of SQL Server.

When a third-party OLE DB provider is used, the account under which the SQL Server service runs must have read and execute permissions for the directory, and all subdirectories, in which the provider is installed.

Managing Providers

There is a set of options that control how SQL Server loads and uses OLE DB providers that are specified in the registry. For more information, see Configuring OLE DB Providers for Distributed Queries.

Managing Linked Server Definitions

When you are setting up a linked server, register the connection information and data source information with SQL Server. After registered, that data source can be referred to with a single logical name.

You can use stored procedures and catalog views to manage linked server definitions:

- Create a linked server definition by running sp_addlinkedserver.
- View information about the linked servers defined in a specific instance of SQL Server by running a query against the sys.servers system catalog views.
- Delete a linked server definition by running sp_dropserver. You can also use this stored procedure to remove a remote server.

You can also define linked servers by using SQL Server Management Studio. In the Object Explorer, right click Server Objects, select New, and select Linked Server. You can delete a linked server definition by right-clicking the linked server name and selecting Delete.

Q8. Explain socket programming and its key components?
Socket programming is a method of creating networked applications that can communicate with each other using network sockets. A socket is a endpoint for sending or receiving data across a computer network, and it is the fundamental building block for network communication.

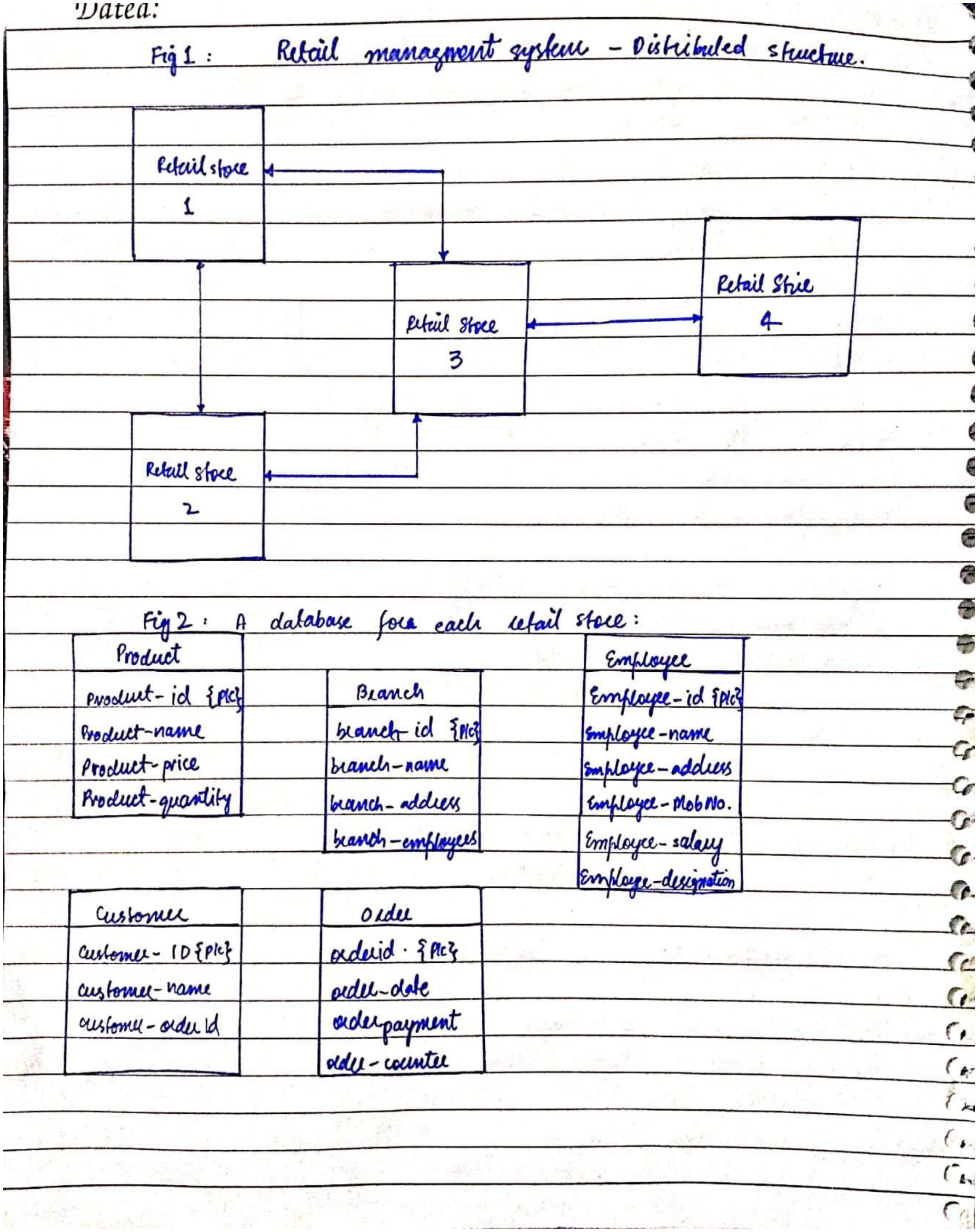
There are several key components to socket programming:

- **Protocols:** A protocol is a set of rules that define how data is exchanged between two systems. There are many different protocols used in socket programming, including TCP, UDP, and SCTP.
- **Ports:** A port is a logical connection point for sending or receiving data on a computer. Each port has a unique number that is used to identify it, and different protocols use different port numbers to differentiate between different types of data.
- **IP addresses:** An IP address is a numerical label assigned to each device connected to a computer network that uses the Internet Protocol for communication. IP addresses are used to identify devices on the network and to route data between them.
- **Sockets:** A socket is a software abstraction that represents a connection between two systems. It is created using a combination of the IP address of the host and the port number of the process that is using the socket.
- **Socket API:** The socket API (Application Programming Interface) is a set of functions that a programmer can use to create and manage sockets in their application. The socket API allows for the creation of both client and server sockets, and it provides functions for sending and receiving data over the network.

Socket programming is a powerful tool for building networked applications, and it is used in a wide variety of contexts, including web servers, chat applications, and online games.

Q1. Present a case of oranganization where distribution of data may increase their organizational performance. Also apply the components of distributed query optimization in your presented case. Also use the optimization technique which you would prefer in accordance with your scenario.

Scenario: An organization that operates multiple retail stores in different locations can benefit from distributing data across all of its stores. By having access to real-time data on sales, inventory levels, and customer demographics, store managers can make more informed decisions on everything from product ordering and stocking to staffing and promotions. For example, if a store in a certain location is consistently selling more of a certain product than other stores, that store's manager can order more of that product to keep it in stock, and other store managers can do the same if they see similar trends in their own store's data. Additionally, having access to data on customer demographics can help managers tailor their marketing and promotions to better target their specific customer base.



Dated:

Fig 1 : Retail managment system - Distributed structure.

Fig 2 : A database for each retail store:

Product
Product-id {PK}
Product-name
Product-price
Product-quantity

Branch
branch-id {PK}
branch-name
branch-address
branch-employees

Employee
Employee-id {PK}
Employee-name
Employee-address
Employee-MobNo.
Employee-salary
Employee-designation

Customer
Customer-ID{PK}
customer-name
customer-order Id

Order
orderid · {PK}
order-date
orderpayment
order-counter

Components of distributed query optimization:

There are three components of distributed query optimization:

- **Access Method:** In order to access and distribute data across all retail stores, the organization can use a centralized database system. The system can be accessed by each store through the internet using a secure connection (such as a virtual private network, or VPN). This will allow the store managers to access the data in real-time, from any location.

- **Transmission Cost:** The cost of transmitting data across the network will depend on several factors, including the size of the data, the number of stores that need to access the data, and the speed of the network connection. To reduce transmission costs, the organization can consider using efficient data compression techniques and optimize the data transmission protocols.
- **Join Criteria:** Join criteria are the conditions that must be met in order to combine data from multiple sources into a single result. In the case of the retail stores, the join criteria can be based on the location of the stores, the type of product being sold, or the demographic data of the customers. For example, the data from all stores can be joined based on the sales data for a specific product, allowing the managers to compare sales trends across different locations and make informed decisions about ordering and stocking

Overall, by distributing the data to multiple locations and using efficient join criteria, access methods, and minimizing the transmission cost, the retail company can improve its inventory management, customer service and improve their overall efficiency.

Programmatic optimization VS systematic optimization:

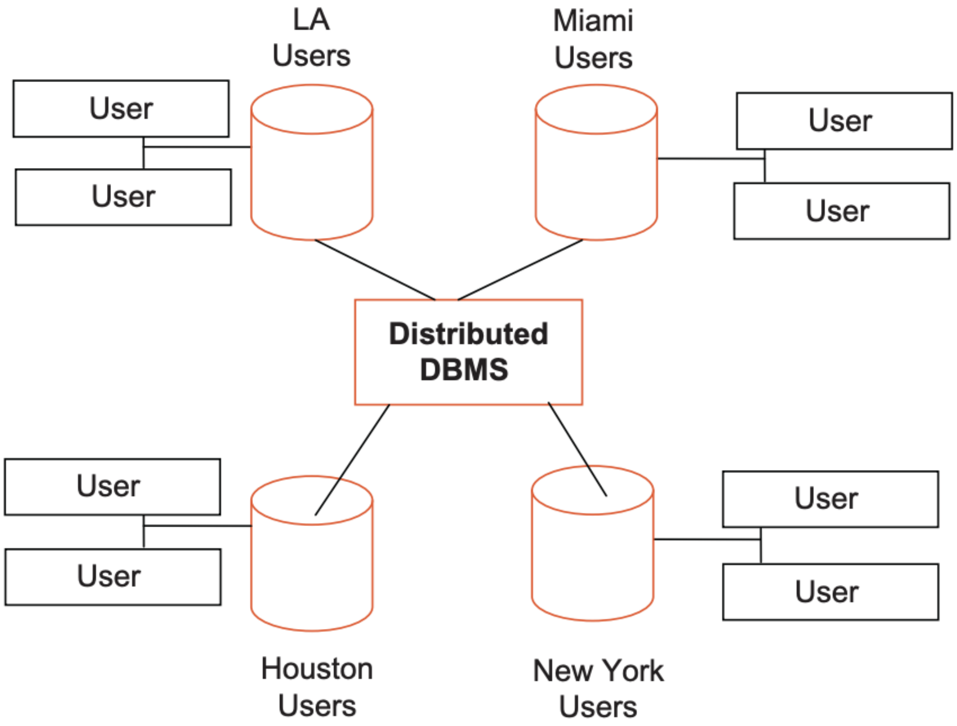
Programmatic optimization is preferable over systematic optimization in the case of an organization that operates multiple retail stores in different locations because:

- **Flexibility**: Programmatic optimization allows for more flexibility as it can adapt to changes in the data and the business environment on the fly, whereas systematic optimization is based on pre-determined rules and can be less adaptable to change.
- **Real-time decisions**: Programmatic optimization allows for real-time decision making, as it can process and analyze data in real-time, whereas systematic optimization requires data to be processed and analyzed before decisions can be made.
- **Data-driven decisions:** Programmatic optimization is based on data-driven decisions, as it uses algorithms to analyze and make decisions based on the data, whereas systematic optimization is based on pre-determined rules.
- **Scalability:** Programmatic optimization is more scalable, as it can handle large amounts of data and multiple sources of data, whereas systematic optimization is more limited in its ability to handle large amounts of data.
- **Personalization:** Programmatic optimization can be used to personalize the retail experience for customers based on their demographics, browsing history, and purchase history, which systematic optimization cannot do.
- **Targeting:** Programmatic optimization can be used to target specific customer demographics with tailored marketing and promotions, which systematic optimization cannot do.

Overall, programmatic optimization is more suitable for this retail organization as it can handle the complexity of data distribution and allows for real-time decision making and personalization which is important for retail industry.

Q2. What are the reasons for fragmenting a database? Discuss the types of fragmentation with an example?

Fragmenting a database refers to the practice of dividing a database into smaller pieces, or fragments, that can be stored and managed separately.



As in above example we fragmented our database into 4 fragments according to location of users. Whoever user belongs to what country we put that particular user in that particular database.

There are several reasons why a database might be fragmented:

- **Performance:** Fragmenting a database can improve the performance of certain types of queries, especially when the data is distributed across multiple servers or storage devices. This is because the query can be processed in parallel, with each fragment being processed on a different server or device.
- **Scalability**: Fragmenting a database can also make it easier to scale the database as the amount of data grows. This is because new fragments can be added as needed, without having to migrate the entire database to a new server or storage system.
- **Availability:** Fragmenting a database can increase the availability of the data, as it reduces the risk of a single point of failure. If one fragment becomes unavailable, the other fragments can still be accessed, allowing the database as a whole to remain operational.
- **Data distribution:** In some cases, fragmenting a database may be necessary in order to distribute the data across multiple geographic locations or to meet regulatory requirements for data storage. There are also some potential drawbacks to fragmenting a database, such as increased complexity and the need for additional infrastructure to manage the fragments. Therefore, it is important to carefully consider the benefits and drawbacks of fragmenting a database before implementing this approach.

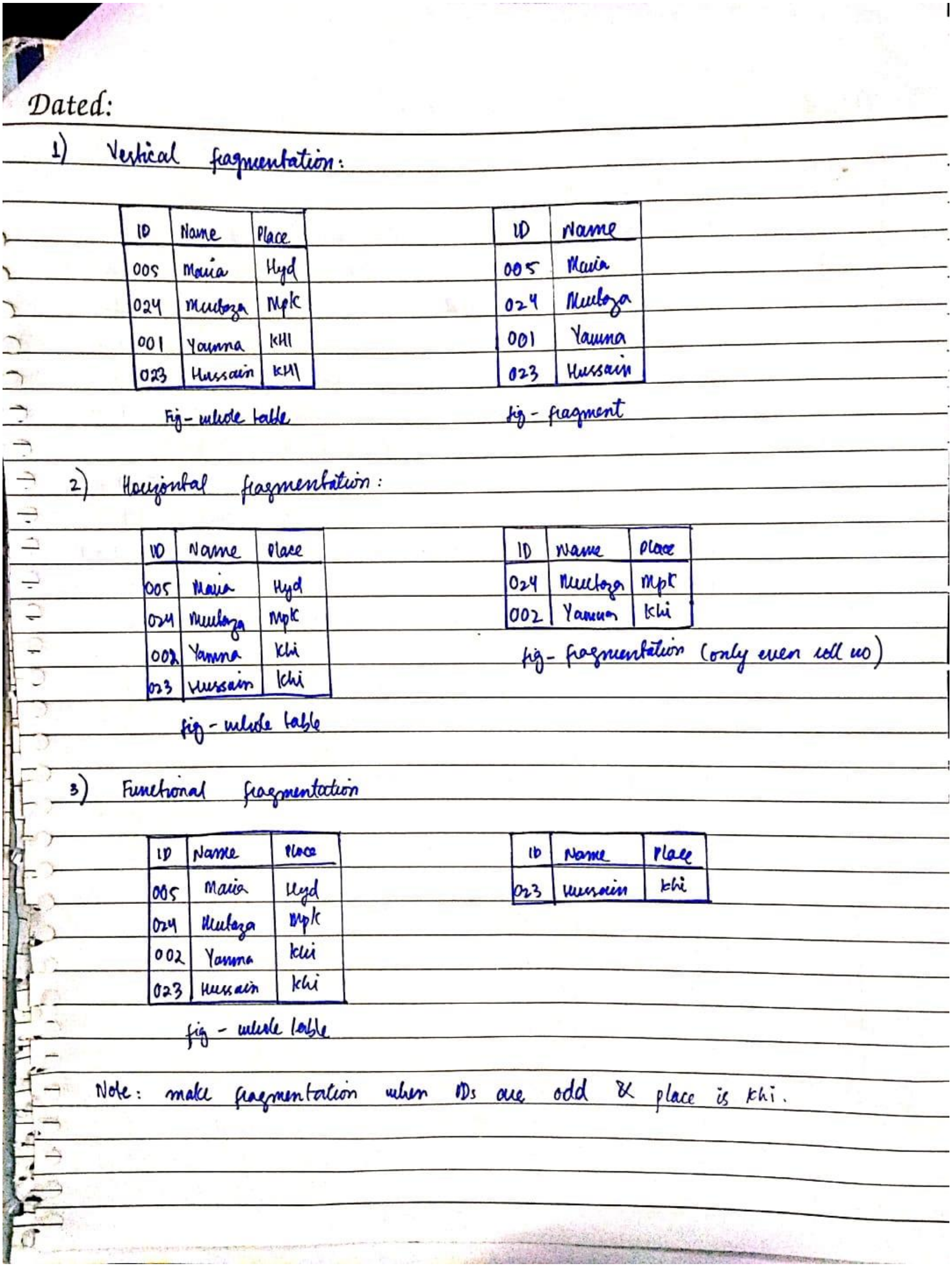
-

Types of Fragmentation:

There are several types of fragmentation that can be applied to a database, including horizontal fragmentation, vertical fragmentation, and functional fragmentation.

1. **Horizontal fragmentation:** This type of fragmentation involves dividing a table into smaller tables that contain a subset of the original data. For example, a table containing customer data might be horizontally fragmented by dividing it into smaller tables based on geographic region, with each table containing only the data for customers in a particular region.
2. **Vertical fragmentation:** This type of fragmentation involves dividing a table into smaller tables based on the columns in the original table. For example, a table containing customer data might be vertically fragmented by dividing it into tables based on personal data, contact data, and order data, with each table containing a subset of the original columns.
3. **Functional fragmentation:** This type of fragmentation involves dividing a database into smaller databases based on the functions or processes that the data is used for. For example, a database containing customer data might be functionally fragmented by dividing it into separate databases for sales, marketing, and customer service, with each database containing only the data needed for the specific function.

4.



Dated:

1) Vertical fragmentation:

ID	Name	Place
005	Maria	Hyd
024	Murtaza	Mpk
001	Yamna	KHI
023	Hussain	KHI

Fig- whole table

ID	Name
005	Maria
024	Murtaza
001	Yamna
023	Hussain

fig - fragment

2) Horizontal fragmentation:

ID	Name	Place
005	Maria	Hyd
024	Murtaza	Mpk
002	Yamna	Khi
023	Hussain	Khi

fig - whole table

ID	Name	Place
024	Murtaza	Mpk
002	Yamna	Khi

fig- fragmentation (only even roll no)

3) Functional fragmentation

ID	Name	Place
005	Maria	Hyd
024	Murtaza	Mpk
002	Yamna	Khi
023	Hussain	Khi

fig - whole table

ID	Name	Place
023	Hussain	Khi

Note: make fragmentation when IDs are odd & place is Khi.

5. **Hybrid fragmentation:** It is combination of all about discuss fragmentations.

<add diagram>

Q3. How blockchain can support distributed database? Discuss any blockchain platform to support your answer.

Blockchain technology can support a distributed database by providing a decentralized and tamper-proof system for storing and sharing data. In a traditional centralized database, all data is stored on a single server or a group of servers that are controlled by a single entity. This centralization can lead to issues such as single points of failure, lack of transparency, and the potential for data to be tampered with or manipulated.

In contrast, a distributed database is one where data is stored across multiple nodes or devices in a network. Each node in the network has a copy of the entire database, and changes to the database are made simultaneously across all nodes. This allows for multiple parties to access and update the database without the need for a central authority to manage and approve changes.

Blockchain technology can be used to create a distributed database because it utilizes a decentralized network of nodes to validate and record transactions. Each block in the blockchain contains a set of records, and once a block is added to the chain, its data cannot be altered. This creates a tamper-proof system for storing and sharing data. Additionally, the use of consensus mechanisms such as Proof of Work (PoW) or Delegated Proof of Stake (DPoS) ensures that all nodes in the network have the same version of the database and that any changes made to the database are validated by the network.

This allows for the creation of decentralized applications (dApps) and smart contracts that can be used to create decentralized databases that can be accessed and updated by multiple parties in a secure and transparent way. Additionally, different blockchain platforms such as Ethereum, EOS, IOTA, Holochain and others provide different features and consensus mechanisms that can be used to optimize the performance, security and scalability of the distributed databases.

Blockchain technology can help create a type of database that is spread out across multiple computers, instead of being stored in one central location. One example of a blockchain platform that supports this is Ethereum. Ethereum is a decentralized, open-source blockchain platform that includes a built-in programming language called Solidity that can be used to create decentralized applications (dApps). These dApps can use Ethereum's decentralized virtual machine (EVM) to store and manage data in a distributed way. Smart contracts, which are self-executing contracts with the terms of the agreement between buyer and seller being directly written into lines of code, can be used to create and manage distributed databases on the Ethereum platform. The data is stored in the smart contract and is replicated across all of the nodes in the Ethereum network, ensuring that the data is secure and not controlled by a single entity.

Q4. (a) Differentiate between synchronous and Asynchronous distributed database.

Synchronous and asynchronous are two different modes of communication that can be used in a distributed database system.

A synchronous distributed database is one in which all of the nodes in the database system communicate with each other in real-time. In a synchronous system, a node that wants to send data to another node will wait until the receiving node is ready to receive the data. This ensures that all of the nodes in the system are always in sync and that any updates to the database are propagated immediately.

An asynchronous distributed database is one in which the nodes do not communicate with each other in real-time. In an asynchronous system, a node that wants to send data to another node will send the data and then continue processing without waiting for a response. The receiving node will process the data at a later time, when it is ready.

Synchronous Distributed Database
– All copies of the same data are always identical
– Data updates are immediately applied to all copies
throughout network
– Good for data integrity
– High overhead ➔ slow response times
• Asynchronous Distributed Database
– Some data inconsistency is tolerated
– Data update propagation is delayed
– Lower data integrity
– Less overhead ➔ faster response time

There are several advantages and disadvantages to using synchronous and asynchronous communication in a distributed database. Synchronous communication ensures that the nodes are always in sync and can be more reliable, but it can also be slower and may introduce delays in the system. Asynchronous communication, on the other hand, can be faster and more efficient, but it can also be less reliable, as there is a risk of data being lost or delayed if a node is not available to receive it.

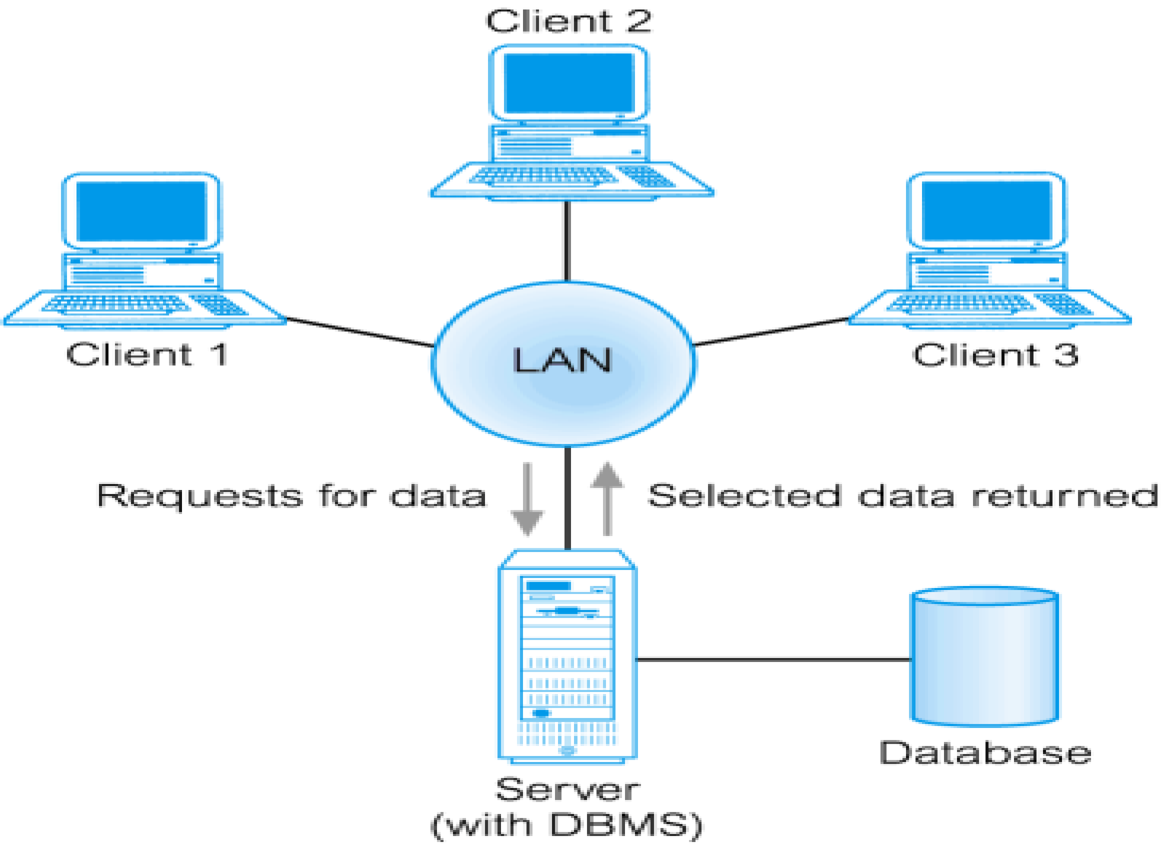
The choice between synchronous and asynchronous communication in a distributed database will depend on the specific requirements and goals of the system. Both approaches have their own strengths and weaknesses, and the right choice will depend on the needs of the application and the environment in which it is deployed.

Q4. (b) Define physical level security, O.S. security, SQL authorizations and identify theft.

1. **Physical level security:** Physical level security refers to measures taken to protect the physical assets of a system, such as servers, storage devices, and networks, from unauthorized access or damage. This might include measures such as locks on server rooms, access control systems, and security cameras.
2. **O.S. security:** O.S. security (operating system security) refers to measures taken to secure the operating system of a computer or server from threats such as viruses, malware, and unauthorized access. This might include measures such as firewalls, antivirus software, and user authentication systems.
3. **SQL authorizations:** SQL authorizations refer to the permissions and privileges that are granted to users or groups of users in a database management system (DBMS) that uses the SQL language. These permissions determine what actions a user is allowed to perform in the database, such as creating, reading, updating, or deleting data.
4. **Identity theft:** Identity theft is the unauthorized use of another person's personal information, such as their name, social security number, or credit card details, in order to commit fraud or other crimes. Identity theft can be a serious threat to both individuals and organizations, and it is important to take steps to protect personal information and to be aware of the signs of identity theft.

Q5. How does client server interaction work? Discuss and explain the algorithm to implement socket programming?

In a client-server interaction, a client application sends a request to a server application, and the server processes the request and returns a response. This interaction is typically facilitated using network sockets, which are endpoint connections for sending and receiving data over a network.



Here is an overview of the algorithm for implementing socket programming:

1. The client creates a socket and specifies the address and port number of the server it wants to connect to.
2. The client sends a request to the server by writing to the socket.
3. The server receives the request by reading from the socket.
4. The server processes the request and generates a response.
5. The server sends the response to the client by writing to the socket.
6. The client receives the response by reading from the socket.
7. The client processes the response and takes any necessary actions.
8. The client closes the socket.

This algorithm can be implemented using the socket API (Application Programming Interface), which provides a set of functions for creating and managing sockets in a program. The specific steps for implementing the algorithm will depend on the programming language and operating system being used.

Q2. Is proof of work a better choice in a public blockchain or in a private blockchain? Justify your answer logically using an example? Why do we need consensus in blockchain? What is the impact of increasing or decreasing block generation rate in blockchain?

PoW a better choice in a public or private blockchain:
Proof of work (PoW) is a consensus algorithm that is used to validate transactions and secure a blockchain network. In a PoW system, network participants (called "miners") compete to solve complex mathematical problems in order to validate transactions and create new blocks in the chain. The first miner to solve the problem is rewarded with a reward (usually in the form of cryptocurrency).

In a public blockchain, PoW is generally considered a good choice because it allows for a decentralized and secure network with no single point of control. This is important in a public blockchain, as it allows anyone to participate in the network and helps to ensure that the network is not controlled by a single entity.

In a private blockchain, on the other hand, PoW may not be the best choice because it requires a significant amount of computational power and may not be necessary in a closed system with fewer participants. In a private blockchain, it may be more efficient to use a different consensus algorithm, such as proof of stake (PoS), which does not require miners to solve complex mathematical problems.

For example, a private blockchain network for supply chain management within a company may not require PoW as the number of trusted participants is known and limited. Instead, a consensus mechanism such as "Proof of Authority" (PoA) could be used, where only authorized nodes are allowed to validate transactions and create new blocks. This would be more efficient as it would not require a large amount of computational work, and still able to maintain the integrity and security of the network.

In summary, PoW is a suitable choice for public blockchain networks because it allows for a decentralized, trustless consensus mechanism. However, in private blockchain networks, where the number of trusted participants is known and limited, alternative consensus mechanisms such as PoA may be more efficient and suitable.

Need of consensus in blockchain:
Consensus is an important concept in blockchain technology because it allows network participants to agree on the state of the blockchain and to validate transactions. Without consensus, it would be difficult to ensure the integrity and security of the blockchain, as there would be no way to determine which transactions are valid and which are not.

There are several reasons why consensus is important in blockchain:

- It ensures that the blockchain is secure and tamper-proof.
- It allows network participants to trust the blockchain and to rely on it for accurate and reliable data.
- It helps to prevent double spending, in which a single unit of cryptocurrency is spent more than once.
- It allows for decentralized decision-making, as all network participants have an equal say in the validation of transactions.

Impact of increasing or decreasing block generation rate in blockchain:
The block generation rate refers to the rate at which new blocks are added to the blockchain. In a blockchain system, the block generation rate is typically determined by the consensus algorithm being used.

Increasing the block generation rate can have several effects on a blockchain:

- It can increase the speed at which transactions are processed, as more blocks can be created to hold new transactions.
- It can reduce the time it takes for a transaction to be included in the blockchain, as blocks are created more frequently.
- It can increase the security of the blockchain, as more blocks means more work is required to alter the chain and more miners are needed to validate transactions.

Decreasing the block generation rate can have the opposite effects:

- It can decrease the speed at which transactions are processed.
- It will result in slower confirmation times and lower throughput, but it may also reduce the risk of forks and conflicts.
- It may be more suitable for networks that prioritize security and stability over high transaction speeds.

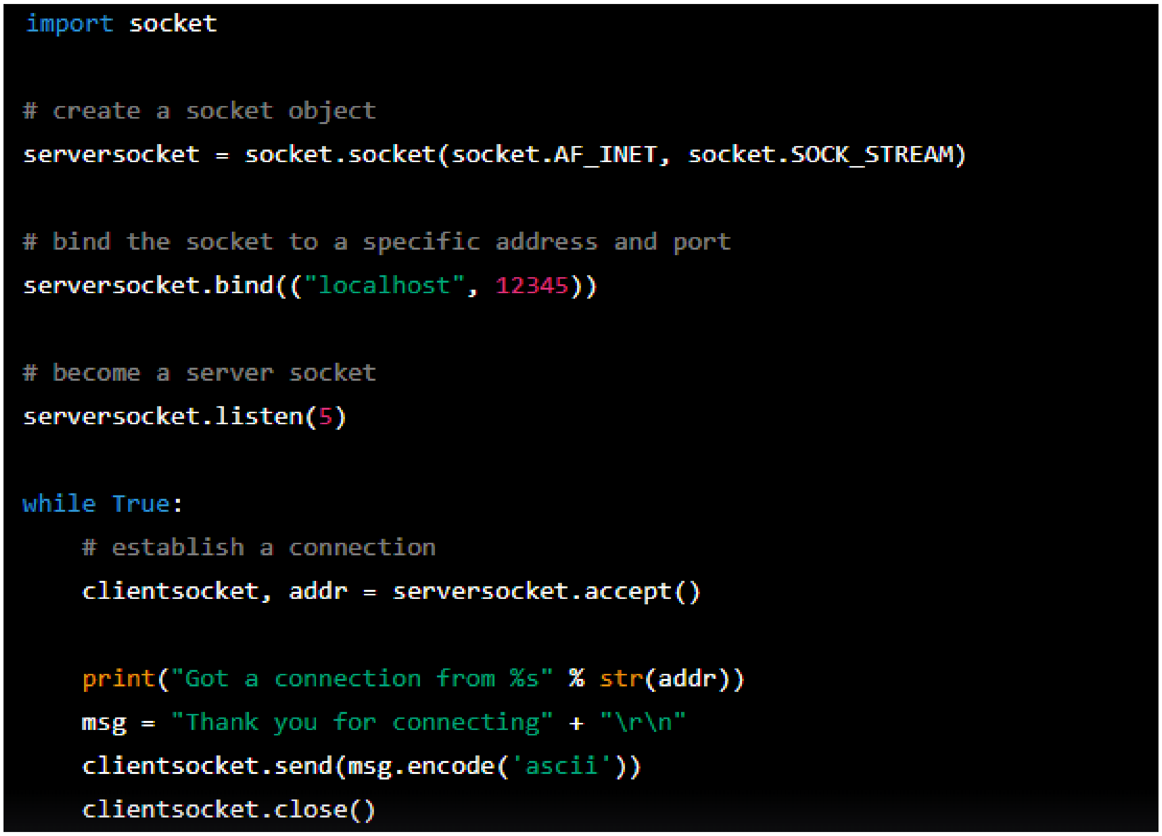
Q3. Explain the basics of client/Server programming? Write down the program to support your answer.

Client/server programming refers to the architecture in which a program, called the client, sends a request to another program, called the server, which then responds to the request. The client and server can be on the same machine or on different machines connected over a network.

In client/server programming, the client is responsible for presenting the user interface, handling user input, and formatting requests to the server. The server, on the other hand, is responsible for processing requests, performing tasks, and returning data to the client.

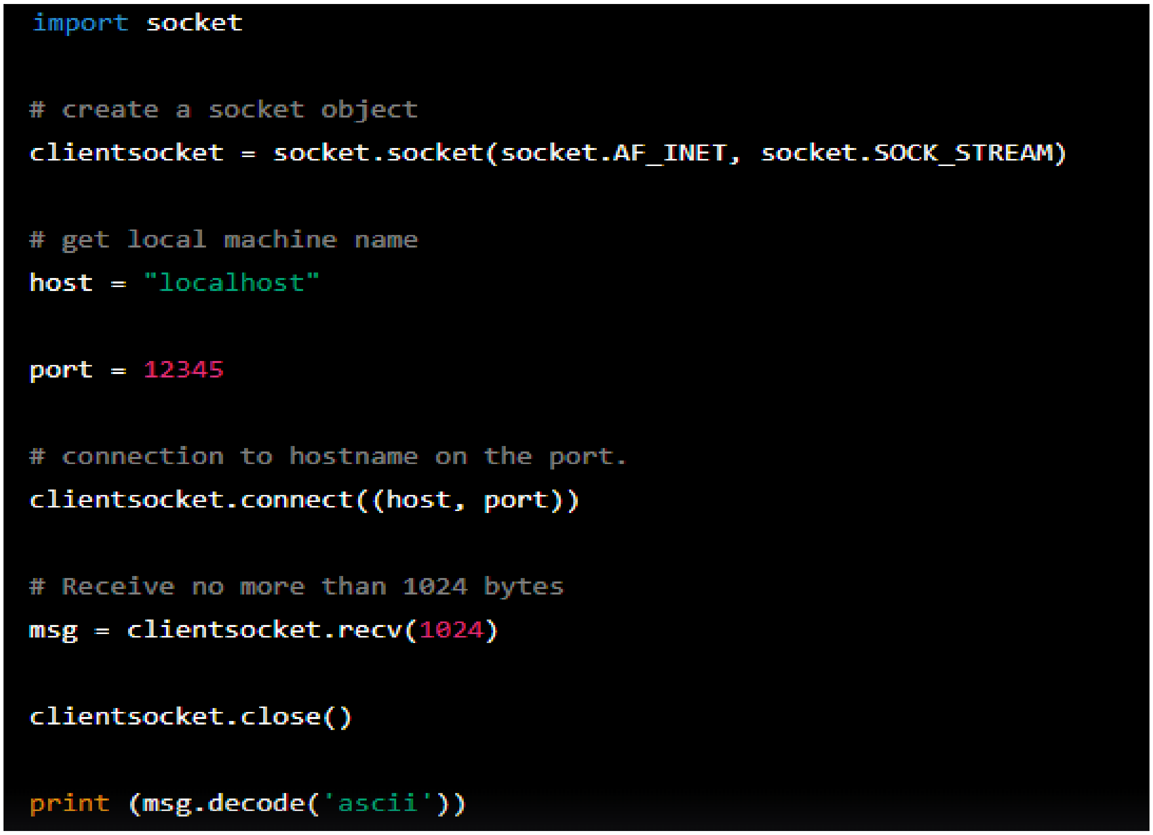
Here is a simple example of a client/server program in Python:

Server:



```python
import socket

# create a socket object
serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# bind the socket to a specific address and port
serversocket.bind(("localhost", 12345))

# become a server socket
serversocket.listen(5)

while True:
    # establish a connection
    clientsocket, addr = serversocket.accept()

    print("Got a connection from %s" % str(addr))
    msg = "Thank you for connecting" + "\r\n"
    clientsocket.send(msg.encode('ascii'))
    clientsocket.close()
```

Client:



```python
import socket

# create a socket object
clientsocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# get local machine name
host = "localhost"

port = 12345

# connection to hostname on the port.
clientsocket.connect((host, port))

# Receive no more than 1024 bytes
msg = clientsocket.recv(1024)

clientsocket.close()

print (msg.decode('ascii'))
```

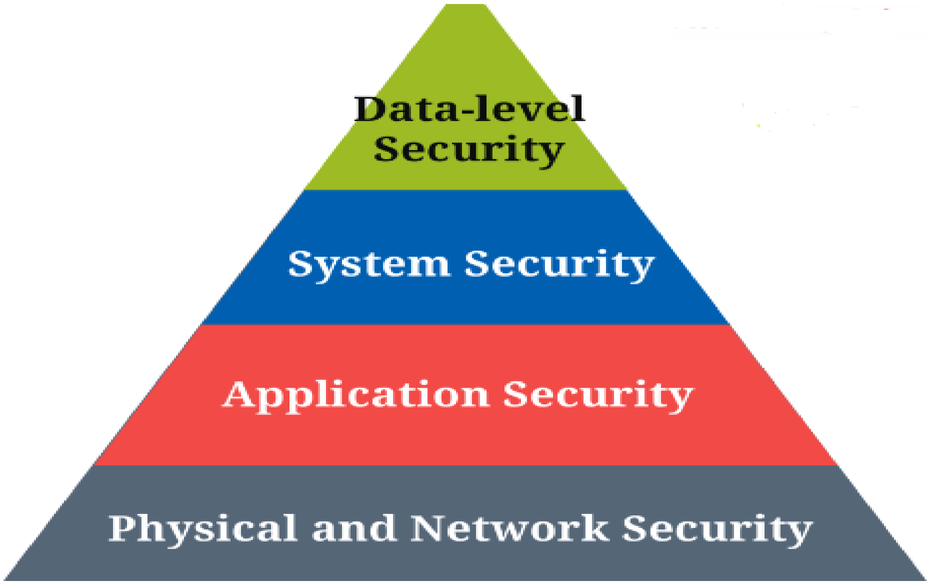
In this example, the server creates a socket and binds it to the address "localhost" and port 12345. It then listens for incoming connections and upon receiving a connection, sends a message "Thank you for connecting" to the client and closes the connection. The client creates a socket and connects to the server at the same address and port. It receives the message sent by the server and prints it out.

This is a basic example of client-server programming, which demonstrates how a client and a server communicate with each other and exchange data over a network.

Q4. (b) Define levels of data security.

There are several levels of data security that can be implemented in order to protect sensitive information from unauthorized access or manipulation. These levels may include:

- **Physical security:** Physical security measures are designed to protect the physical assets of a system, such as servers, storage devices, and networks, from unauthorized access or damage. This might include measures such as locks on server rooms, access control systems, and security cameras.
- **Network security:** Network security measures are designed to protect data as it is transmitted over a network. This might include measures such as firewalls, encryption, and secure protocols like SSL and TLS.
- **Access control:** Access control measures are designed to restrict access to data or systems to authorized users only. This might include measures such as user authentication systems, role-based access control, and two-factor authentication.
- **Data protection:** Data protection measures are designed to protect the data itself from unauthorized access or manipulation. This might include measures such as data encryption, data masking, and data backup and recovery systems.
- **Security monitoring:** Security monitoring involves monitoring the system for security breaches or potential threats and taking appropriate actions to mitigate them. This might include measures such as intrusion detection systems, security incident and event management (SIEM) systems, and vulnerability assessment tools.



Implementing these levels of security can help to protect data and systems from a wide range of threats, including hackers, viruses, and insider threats. It is important to assess the specific security needs of an organization and implement appropriate measures to meet those needs.

Q3. Propose and explain a framework to counter OS level attack.
Answer: 1
One framework to counter OS level attacks is the Defense-in-Depth strategy, which involves implementing multiple layers of security controls to protect a system. This approach can be broken down into several components:

- Access controls: This includes implementing authentication mechanisms, such as password policies and two-factor authentication, to ensure that only authorized users can access the system.
- Network security: This involves implementing firewalls, intrusion detection/prevention systems, and virtual private networks (VPNs) to protect the system from network-based attacks.
- Host-based security: This includes implementing anti-virus/anti-malware software, security software updates, and intrusion detection/prevention systems on individual hosts to protect them from malicious software and other attacks.
- Application security: This involves implementing security controls at the application level, such as input validation and sanitization, to protect against attacks that target specific applications.
- Monitoring and incident response: This includes implementing monitoring and logging systems to detect and respond to security incidents in a timely manner.

By implementing multiple layers of security controls, the Defense-in-Depth strategy makes it more difficult for attackers to penetrate a system. Even if one layer of security is breached, the other layers can help to contain and mitigate the attack.

Answer: 2

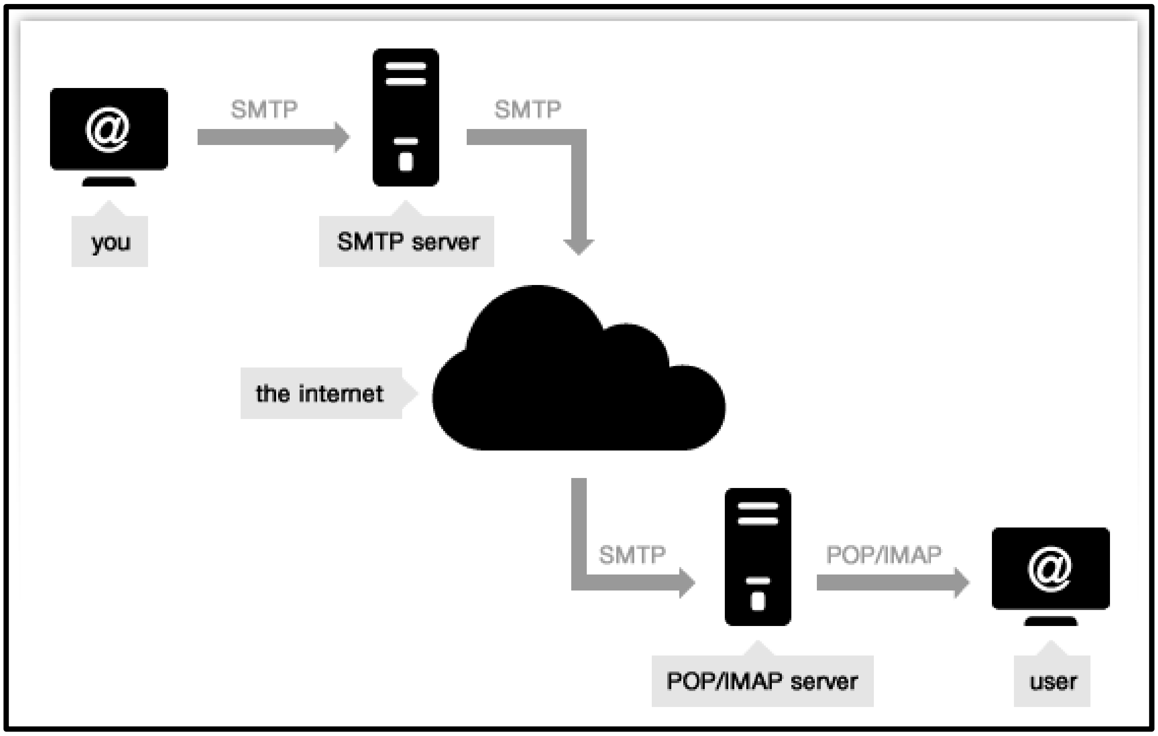
Another framework to counter OS level attacks is the Zero Trust model. The Zero Trust model assumes that all network entities and users, whether inside or outside the network perimeter, are untrusted until proven otherwise. This framework involves implementing several key components:

- Micro-segmentation: This involves dividing the network into smaller segments and implementing strict access controls between them. This makes it more difficult for an attacker to move laterally within the network.
- Multi-factor authentication: This involves implementing multiple forms of authentication, such as something you know (e.g. password), something you have (e.g. security token), and something you are (e.g. biometrics), to ensure that only authorized users can access the system.
- Least privilege access: This involves providing users with the least amount of access necessary to perform their job functions. This limits the damage that can be done by a compromised user account.
- Continuous monitoring: This involves implementing monitoring and logging systems to detect and respond to security incidents in a timely manner.
- Security automation: This involves automating security tasks such as security assessments, incident response, and patch management to ensure that security controls are up-to-date.

By following the Zero Trust model, you can create a more secure OS environment by always being vigilant to the risk of an attack and implementing multiple layers of security checks to ensure that only trusted entities have access to the system.

Q3. Discuss the working of SMTP server with example and explain its architecture.

SMTP (Simple Mail Transfer Protocol) is a protocol for sending email messages between servers. Most email systems that send mail over the Internet use SMTP to send messages from one server to another, and to deliver messages to local mail clients like Microsoft Outlook or Apple Mail.



Here is an example of how an SMTP server works:

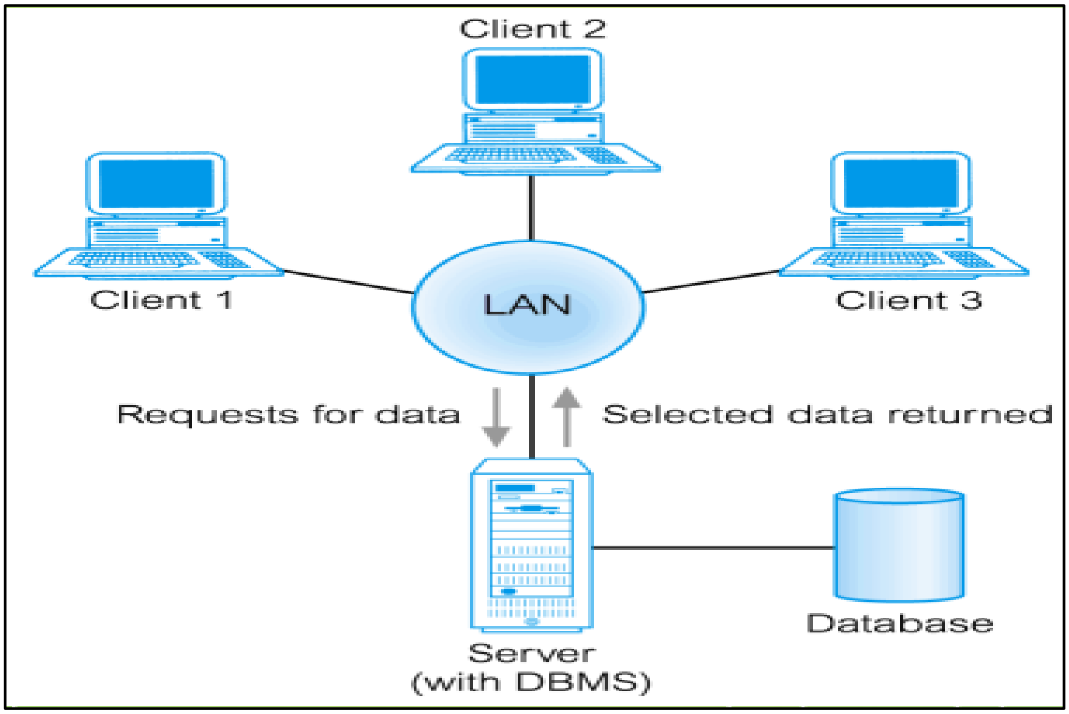
- A user composes and sends an email message using a mail client, such as Microsoft Outlook or Apple Mail.
- The mail client connects to the user's outgoing mail server (SMTP server) and sends the message.
- The SMTP server receives the message and checks the destination email address to determine the correct mail server to deliver the message to.
- The SMTP server establishes a connection to the destination mail server and sends the message.
- The destination mail server receives the message and stores it in a queue to be delivered to the recipient's mailbox.
- The destination mail server delivers the message to the recipient's mailbox, either by sending it to the recipient's local mail client or by storing it in an online mailbox that the recipient can access via the web.

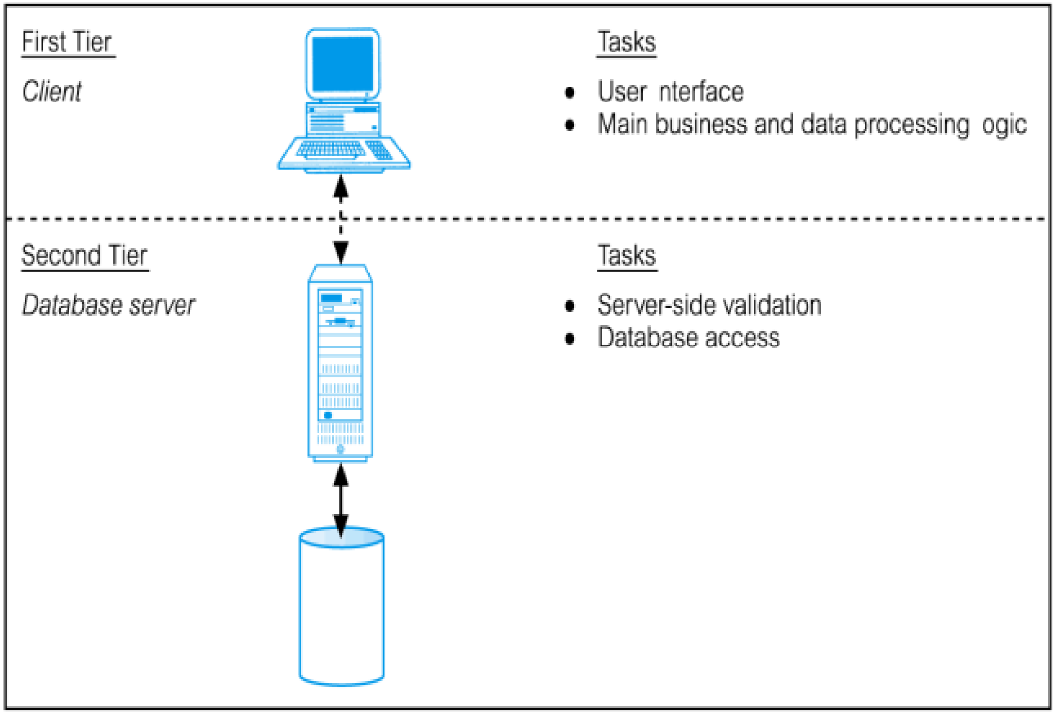
In this example, the SMTP server acts as a relay, forwarding the message from the sender's mail server to the recipient's mail server. The SMTP server does not store the message itself, but simply passes it along to the next server in the chain.

SMTP servers are an important part of the email infrastructure, as they enable users to send and receive messages across the Internet. They also play a role in helping to ensure the security and integrity of email messages, as they can be configured to authenticate users and check for spam or other malicious content.

Q5. Explain client sever architecture with an example.

Client-server architecture is a common network architecture in which one or more client computers request and receive services or resources from one or more server computers. The client and server communicate with each other using a specific protocol, such as TCP/IP.





For example, consider a web application where users access a website through their web browsers (clients) to request information from a web server. The web server retrieves the requested information from a database server and sends it back to the client through the web browser. In this scenario, the web browser acts as a client, the web server acts as a server, and the database server acts as a back-end server.

Another example is an email system, where the client computer (such as Microsoft Outlook or Gmail) connects to an email server (such as a Microsoft Exchange server or an IMAP server) to send and receive emails. The client computer requests the email server to send, receive, or retrieve emails, and the email server responds by fulfilling the request and sending the emails back to the client computer.

In summary, client-server architecture is a way to organize and distribute the workload across multiple devices and computers, it allows to separate the function of providing and consuming services or resources, and it allows for centralization of certain processes and data, making it more efficient and manageable.

Question no 4b in 2020

A distributed database is a complex system that requires careful planning and design to ensure its proper functioning. In order to understand the importance of this, consider the example of a bank with multiple branches in different cities of Pakistan, such as Karachi, Lahore, Peshawar, and Islamabad. The headquarters of the bank has several branches and the master copy of the database, which is maintained at the headquarters, is different from the copies of all respective branches. Hybrid fragmentation is the most optimal solution for distributing the database, as it offers a balance between performance and reliability. However, if not executed properly, it can result in decreased data reliability and security, which is of utmost importance in a banking system. Fragmentation issues must be addressed in order to avoid transmission and communication problems that can lead to longer access times for branches and increased vulnerability to security breaches. As security is a critical factor for the bank data, frequent transmission and wastage of time must be minimized to protect the bank's reputation and maintain customer trust.

Question ICO, IPO, STO

An Initial Public Offering (IPO) is a milestone event in the life cycle of a company, where it transforms from a privately-held entity to a publicly-traded one by issuing shares of stock to the public. The goal of an IPO is to raise capital by selling ownership shares to investors, providing the company with access to larger pools of funding, increasing its visibility and brand recognition, and offering liquidity options to its founders and early investors. The process typically involves the company working with an investment bank to underwrite the offering, set the initial stock price, and sell shares through a stock exchange to the public.

ICO: ICO stands for Initial Coin Offering. It's a form of fundraising where a new cryptocurrency project sells a portion of its token supply to early backers in exchange for funding. ICOs are often used by blockchain-based startups to raise capital and build a community of users and investors.

Initial Public Offering (IPO)	Initial Coin Offering (ICO)	Security Token Offering (STO)
Regulatory compliance is a prerequisite	No regulatory compliance	Regulatory compliance is a prerequisite
Min. earnings threshold & stable business record	Whitepaper	Asset-backed Might be backed by utility
Prospectus - Legal	No standard format for the whitepaper - Might not be legal	Purpose - Investment Usage on the platform
Lawyers & banks	Computer programmers & Internet	Barrier to entry – High (ico much lower)
Restricted to a country; Requires brokers for international dealings	International	Accredited investors Open to all
Ownership stake	Might entitle one to a future service	raise funds for crypto projects, no equity
Annual dividends based on performance	Benefits are listed in the whitepaper	Offered, mint native digital token

STO: STO stands for Security Token Offering. It's similar to an ICO, but instead of selling tokens that represent a future product or service, STOs offer tokens that represent ownership in a tangible asset, such as real estate or a company's stock. STOs are considered securities and are subject to government regulations, whereas ICOs are often unregulated.

Web 3.0 is the third iteration to the internet, that combines a bouquet of disruptive technologies such as blockchain, AR, VR, Cloud, Edge, IoT, Geolocation tech, 5G, digital twin and runs on an AI-based analytics layer for data-driven insights. The underlying blockchain technology enables trust through mathematical proof and smart contracts. This facilitates creation of a network with minimal managerial requisites, thereby helping drive the new internet economy as a group or individuals, rather than giving into the centralized controls of large tech giants. Web 1.0 introduced a new platform for digital content, and Web 2.0 gave us user-driven feedback, Web 3.0 represents a new phase in the internet’s evolution i.e., a visually dynamic, semantic and spatial Web

The goal of Web 3.0 is to create an intelligent, autonomous, connected, and open internet. Tim Berners Lee labelled Web 3.0 as the semantic web (Semantic Web technologies enable people to create data stores on the Web, build vocabularies, and write rules for handling data). The impressive capabilities of Web 3.0 such as understanding information like a human, through hyper-intelligent networks, coupled with underlying secure digital ledger layer to track financial interactions, and convergence of AI, 5G telecommunications will open new avenues of growth and exploration for the digital ecosystem. It is forecasted to revolutionize the modus operandi of today’s network and enhance it manifold. It has the potential to blur the lines between the physical and digital world and provide a seamless interaction

Area	Web 2.0	Web 3.0	Future prospect
Type of web	Social	Semantic/spatial	Web 3.0 will offer a more immersive experience that will offer information and services in XR
Structure	centralized	Decentralized	Web 3.0 has a blockchain enabled architecture that enables necessary decentralization. This is further augmented by a balanced economy, where creators and users are compensated for their time, effort, and information
Target audience	2D	Mix of 2D & 3D	Web 3.0 is focused on reaching individual users, and offers a custom tailored experiences based on their online behavior
Viewability	Communities	Individuals	Web 3.0 combines 2D and 3D by using immersive realities to render an enriched UX.
Application type	Apps	DApps	Using smart contracts dApps are self-executing and require minimum governance
App development	Open code plus proprietary	100% open source	This will provide complete transparency and will be available to everyone for inspection and development

Decentralized Data Management

Monolithic architectures often use a shared database, which can be a single data store for the whole application or many applications. This leads to complexities in changing schemas, upgrades, downtime, and dealing with backward compatibility risks. A service-based approach mandates that each service get its own data storage and doesn’t share that data directly with anybody else.

All data-bound communication should be enabled via services that encompass the data. As a result, each service team chooses the most optimal data store type and schema for their application. The choice of the database type is the responsibility of the service teams. It is an example of decentralized decision-making with no central group enforcing standards apart from minimal guidance on connectivity. AWS offers many fully managed storage services, such as object store, key-value store, file store, block store, or traditional database. Options include, Amazon $3, Amazon DynamoDB, Amazon Relational Database Service (Amazon RDS), and Amazon Elastic Block Store (Amazon EBS).

Decentralized data management enhances application design by allowing the best data store for the job to be used. This also removes the arduous task of a shared database upgrade, which could be weekends-worth of downtime and work, if all goes well. Since each service team owns its own data, its decision making becomes more independent. The teams can be self-composed and follow their own development paradigm.

A secondary benefit of decentralized data management is the disposability and fault tolerance of the stack. If a particular data store is unavailable, the complete application stack does not become unresponsive. Instead, the application goes into a degraded state, losing some capabilities while still servicing requests. This enables the application to be fault tolerant by design.

The following are the key factors from the twelve-factor app pattern methodology that play a role in organizing around capabilities:

Disposability (maximize robustness with fast startup and graceful shutdown) — The services should be robust and not dependent on externalities. This principle further allows for the services to run ina limited capacity if one or more components fail.

Backing services (treat backing services as attached resources) — A backing service is any service that the app consumes over the network such as data stores, messaging systems, etc. Typically, backing services are managed by operations. The app should make no distinction between a local and an external service.

Admin processes (run admin/management tasks as one-off processes) — The processes required to do the app’s regular business, for example, running database migrations. Admin processes should be run in a similar manner, irrespective of environments.

To achieve a microservices architecture with decoupled data management, the following software design
patterns can be used:

Controller — Helps direct the request to the appropriate data store using the appropriate mechanism.
Proxy — Helps provide a surrogate or placeholder for another object to control access to it.

Visitor — Helps represent an operation to be performed on the elements of an object structure.

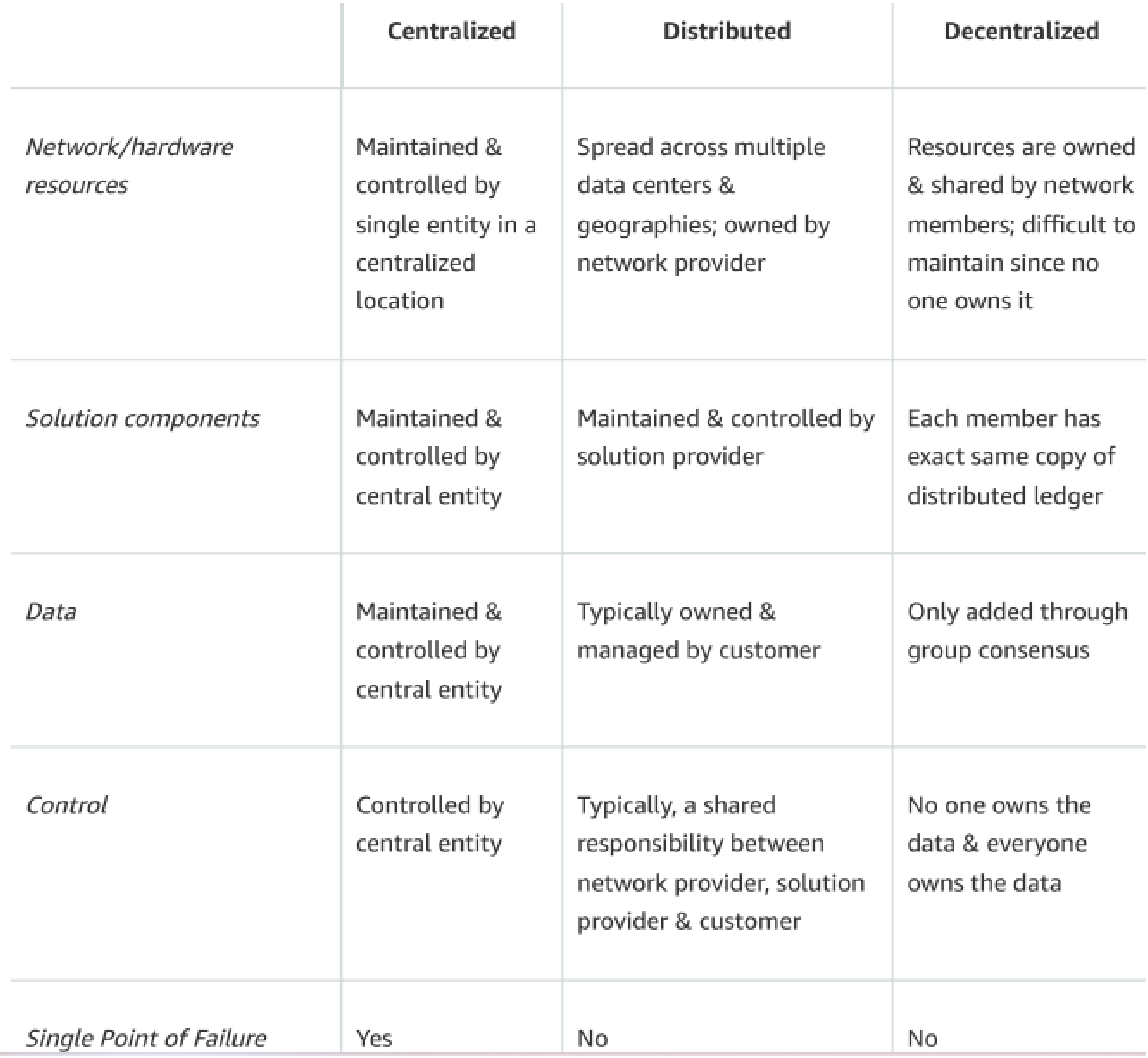
Interpreter — Helps map a service to data store semantics.

Observer - Helps define a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically.

Decorator — Helps attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to sub-classing for extending functionality.

Memento - Helps capture and externalize an object's internal state so that the object can be returned to this state later.

Differences:



	Centralized	**Distributed**	**Decentralized**
Network/hardware resources	Maintained & controlled by single entity in a centralized location	Spread across multiple data centers & geographies; owned by network provider	Resources are owned & shared by network members; difficult to maintain since no one owns it
Solution components	Maintained & controlled by central entity	Maintained & controlled by solution provider	Each member has exact same copy of distributed ledger
Data	Maintained & controlled by central entity	Typically owned & managed by customer	Only added through group consensus
Control	Controlled by central entity	Typically, a shared responsibility between network provider, solution provider & customer	No one owns the data & everyone owns the data
Single Point of Failure	Yes	No	No

Fault tolerance	Low	High	Extremely high
Security	Maintained & controlled by central entity	Typically, a shared responsibility between network provider, solution provider & customer	Increases as # of network members increase
Performance	Maintained & controlled by central entity	Increases as network/hardware resources scale up and out	Decreases as # of network members increase
Example	ERP system	Cloud computing	Blockchain