Distributed Database Management Systems

Course Title: **Distributed Database Management Systems**
Course Code: **CS712**
Instructor: **Dr. Nayyer Masood (nayyerm@yahoo.com)**
Lecture No. **1**

Introduction:

This is an advanced course of the previous that you must have previously studied and that is the "Database Management Systems". This course enhances the concepts learnt earlier, moreover, the applications where you will be applying the concepts and the techniques learnt in this course are also more advanced and complex by nature. The Distributed Database Management Systems (DDBMS) uses the concepts of:

1) Database Management Systems
2) Networking

The key is to identify the environments in which we have to use the distributed databases. You may realize that using distributed databases in some situations may not prove to be fruitful. The implementation may develop drawbacks or may become in-efficient.

As a computer or database expert, you will always be having the basic assignment that you are given a system for which you have to design/develop a solution or a database system. You will have several options available and interesting thing is that every one of them would work. Now the challenge lies in selecting the most feasible solution. For this, the merits and demerits of every approach have to be analyzed. If this evaluation is not properly done in the beginning then it is very difficult to apply adjustments in the later stages. To be more precise, Distributed Database System (DDBS) will be one of many options available to you for any system that you will be asked to develop. It is your job to analyze whether the environment needs a DDBS solution or any other one. A wrong decision in this regard may introduce inefficiency rather than any advantage.

Note:
Lack of experience or/and homework may lead to the selection of a wrong option

Outline of Course

There are going to be four major component in this course:

Introductory Stuff
Architectures and Design Issues Technological Treatment
Related Topics

Introduction:

This part will cover the introduction to the basic concepts of databases and networks. Then we will also realize the nature of application that need a DDBS, so that we are convinced that this is the environment and it requires a DDBS and then we think and plan the later issues of the implementation.

Architecture and Design:

There are different architectures available for designing distributed systems and we have to identify the right one as different architectures are suitable for different environments. Moreover, there are different approaches to implement the DDBS designs, we will study those approaches since each suits different environment.

Note:
Selection of wrong architecture in an environment results in an in-efficient system

Technological Treatment:

Different design approaches of DDBS will be implemented using the prevailing DDBMSs, like SQL Server and Oracle. That will give you the idea of how a real DDBS will look like.

Theoretical Aspects:

We will discuss the theoretical aspects related to the DDBS. The study of these issues will help you administering a DDBS on one side and on the other side it will help you in the further studies/research in the DDBS. The database management systems available today do most of the administration automatically but it is important for the database designer to know the background procedures so that the overall efficiency of the distributed database management systems may be enhanced.

Recommended Books:

1- *Distributed Database Systems (2nd Edition) by T.M., Ozsu, P. Valdusiez*
2- *Distributed Database Systems. By D. Bell, J. Grimson, Addison-Wesley, 1992*
3- *Distributed Systems: Concepts and Design, 4th Edition, by G. Coulouris, J. Dollimore, T. Kindberg, Addison-Wesley*

The book mentioned at No. 1 is the main book for this course. It is a famous and one of the rare books written on the topic. Course is mainly based on this book. So you will holding this piece very frequent in coming days, make it a bedside item.

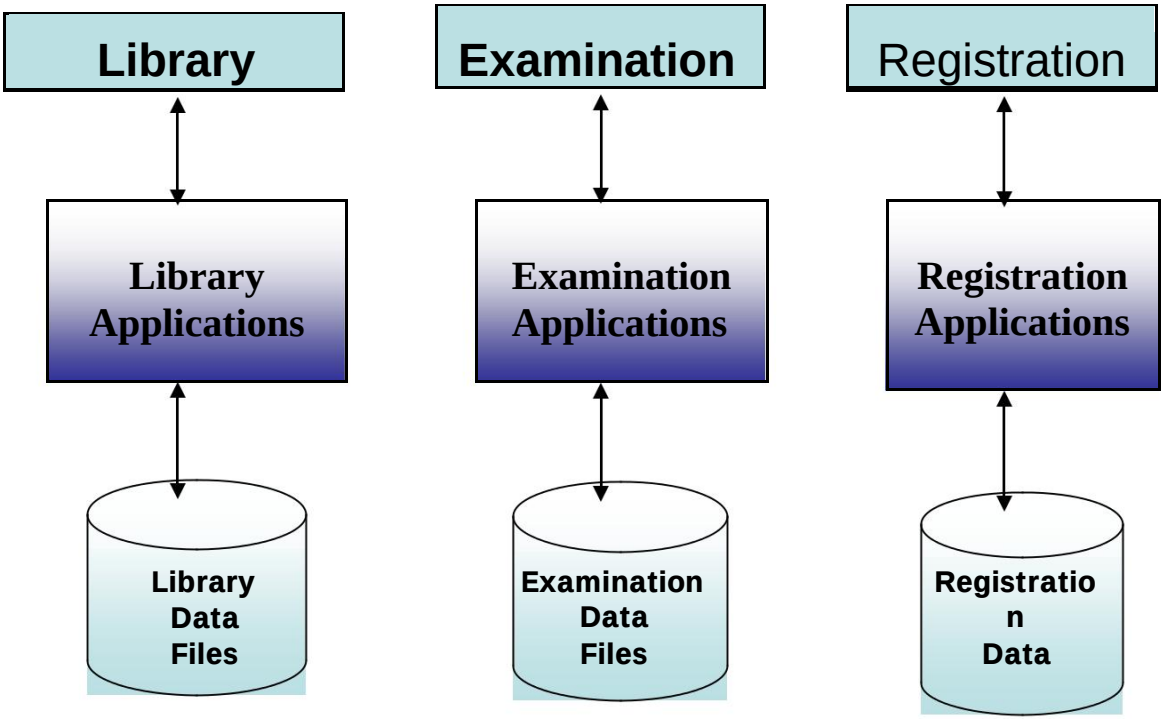
You are already familiar with the marks split, that consists of mid-term exam, assignments and a final exam. Good luck with your course and now lets start reading.

History:

This part is from the first course of Databases. Computer applications are divided into two types:

1) Data Processing Applications
2) Scientific / Engineering Applications

Data Processing Applications in computer terminology are referred to as “File Processing Systems”. In those applications the data was processed with the help of different programming languages. The program, which was used to process the data, had the data defined in it, therefore they were dependant on each other. If we had to make any change in the program it was difficult, as we had to take care of the dependant data as well and vice versa. You are already familiar with the following diagram



Program and Data Interdependence

Typical File Processing Environment

For example in the above example we can see three systems i.e. Examination, Library and the Registration system. Each of them is having its own data for processing, however there might be some information which is common to all the three systems but still being stored separately. This results in:

1) Data Redundancy
2) Expensive Changes/Modifications due to redundancy of the data

Database Approach:

To remove the defects from the file processing systems the database was approach was used which eliminated the interdependency of the program and the data. The changes/modifications can be brought about easily whether they were related to the programs or the data itself.

Database is a shared collection of logically related data

Distributed Computing Systems:

Distributed Computing System can be defined as "*A system consisting of a number of autonomous processing elements that are connected through a computer network and that cooperate in performing their assigned task*".

Three things are important here:

1) Multiple systems are involved.
2) These multiple systems are linked together through some network
3) These multiple systems perform common tasks in which they cooperate with each other.

Distributed Computing:

We can elaborate the concept of the Distributed Computing with the following example:

A Computer has different components for example RAM, Hard disk, Processor etc working together to perform a single task. So is this the example of Distributed Computing? The answer is **No.** This is because according to the definition there are "different computing systems" involved therefore we cannot say that the distributed activities involved in a single computer is an example of distributed computing.

The second thing is that what is being distributed? A few examples are given below:

1) Processing Logic can be distributed
2) We can divide our goal/task into different functions and get them distributed among various systems
3) Data
4) Control

All these things can be divided to make our system run efficiently.

Classification of Distributed Computing Systems:

Following factors are to be addressed:

1) **Degree of Coupling:**

   Here we have to see that how closely the systems are connected. We can measure this closeness in terms of messages exchanged between two systems and the duration for which they work independently.

> ***Note:***
> ***If two systems are connected through a network the coupling may be weak however if they are sharing some hardware the coupling is strong.***

2) **Interconnection Structure:**

   We have to see how the two systems are connected to each other. The connection can be point-to-point or sharing a common channel etc.

3) **Interdependence:**

   The interdependency doesn't base totally on the architecture, it is also based on the task and how it is distributed.

4) **Not Totally Independent:**

Why Distributed Computing Systems:

Some organization structures are suitable for Distributed Computing

1) Organization is expanded on a large geographic area.
2) Organization has different functioning units located in different areas Technological Push:
1) Hardware has become cheap.
2) Internet i.e. communication connectivity is easily available and cheap.

Distributed Computing Alerts:

1) Poor management leads to in-efficiency.
2) We create information islands and due to lack of standards the system gets in-efficient
3) Improper Designing e.g. we have to travel by air to some destination. We go to the an airline's booking office and get our booking done to some destination. The overall process takes a considerable delay. Our booking is done but with a delay this might occur as a result of improper design.

> ***Note:***
> ***Multiple options are available for designing distributed systems.***

Course Title: Distributed Database Management Systems
Course Code: CS712
Instructor: Dr. Nayyer Masood (nayyerm@yahoo.com)
Lecture No: 2

In this lecture:

- **Definition of a Distributed Database System (DDBS)**
- **The candidate applications for a DDBS**
- **The definition of a Distributed Database Management System (DDBMS)**

Distributed Database System:

A collection of logically inter related databases that are spread physically across multiple locations connected by a data communication link.

Main characteristics:

The DDBS, in its general meaning, is a solution that has been adopted for a single organization. That is, an organization wants to develop a database system for its requirements, now the analyst(s) analyze the requirements and out of many possible solutions, like, client-server, desktop based, or distributed database, this particular architecture is proposed. Following are the major characteristics of a DDBS highlighted in the definition above:

Data management at multiple sites: Although it belongs to the same organization but data in a DDBS is stored at geographically multiple sites. Data is not only stored at multiple sites but it is managed there, that is, tables are created, data entered by different users and multiple applications running. A complete DBMS environment at each local site.

Local requirements: Each of the sites storing data in a DDBS is called a local site. Every site is mainly concerned with the requirements and data management for the users directly associated with that site, called local users.

Global perspective: Apart from catering the local requirements, the DDBS also fulfils the global requirements. These are the requirements that are generated by the central management who want to make overall decisions about the organization and want to have the overall picture of the organization data. The DDBS fulfils the global requirements in a transparent way, that is, the data for these requirements is fetched from all the local sites, merged together and is presented to the global users. All this activity of fetching and merging is hidden from the global user who gets the feeling as if the data is being fetched from a single place.

In a DDBS environment, three types of accesses are involved:

Local access: the access by the users connected to a site and accessing the data from the same site.

Remote access: a user connected to a site, lets say site 1, and accessing the data from site 2.

Global access: no matter from where ever the access is made, data will be displayed after being collected from all locations.

A user does not know from where he is getting the data. To the user it appears that the data is present on the machine on which he is working.

Distributed databases; where to apply:

As mentioned before, the DDBS is one of the possible solutions for a database application. We need to analyze the environment to decide whether it requires a DDBS or not. The candidate applications for a DDBS have following two main characteristics:

1- Large number of users

2- Users are physically spread across large geographical area

Following are some of the Database applications that are strong candidates for a DDBS.

Banking Applications: Take the example of any Pakistani Bank. A bank has large number of customers and its branches are spread across all Pakistan (obviously, many of them have branches around the world, their candidature is even stronger). Now, in the modern banking, the customers not only access/use their accounts from within the branch rather they access data outside the branch. Like, from ATMs/branches spread across the city or country. Every time, when a user operates his account from anywhere in the country/world, his account/data is being accessed.

Air ticketing: We now have the facility to book a seat in any airline from any location to any destination. e.g. we can book return ticket from Lahore to Karachi and from Karachi to Lahore from the airline's Lahore office. This system too, has a large number of users spread across a large area. Whenever a booking is made, the data of the flights is accessed.

Business at multiple locations: A company having offices at multiple locations, or different units at different locations, like production, warehouses, sales operating from different locations, each site storing data locally however, these units need to access each other's data and data from all the sites is required for the global access.

Distributed Database Management System:

A software system that permits the management of distributed database and makes the distribution transparent to the users.

Like we need a DBMS for a centralized or client-server database, we do need a DDBMS for a DDBS. A DDBMS will behave like a normal DBMS on the local site, however, the additional facility that it provides is the creation and maintenance of the global access where data across multiple sites is accessed against a single query. The approach that most of the current commercial DBMS vendors (like Oracle, SQL Server, DB2, Sybase) have adopted is that they provide different versions for different situations. If the user needs a desktop database for the single computer usage, then a smaller version is available that does not support the remote access or data distribution. For client-server database there is another version, and for the DDBS environment the Enterprise Edition of the DBMS is provided that of course supports data distribution among multiple sites, the establishing of link between these sites and finally joining/combining data from multiple sites against a single query.

Decentralized database:

A collection of Independent databases on non-networked computers. In this environment the data at multiple computers is related but these computers are not linked, so whenever data has to be accessed from multiple computers, we need to apply some means of transferring data across these computers.

Summary: In today's lecture we have discussed the definition of the DDBS, the common applications where the DDBS can be applied and the reasons why the DDBS is recommended for these sites. This is extremely important to have a clear idea about what precisely is a DDBS if we want to implement a DDBS properly.

Course Title: Distributed Database Management Systems
Course Code: CS712
Instructor: Dr. Nayyer Masood (nayyerm@yahoo.com)
Lecture No: 3

In previous lecture:

- **Definition of a Distributed Database System (DDBS)**
- **The candidate applications for a DDBS**
- **The definition of a Distributed Database Management System (DDBMS)**

In this lecture:

- **Resembling Setups**
- **Why to have a DDBS**
- **Advantages/Promises of DDBS**

Resembling setups:

Distributed files:

A collection of files stored on differed computers of a network, not a DDBS; Why?

This is not enough for DDBS, as the data should be logically related.

> ***Note:***
> ***DDBS is logically related, has common structure among files, and accessed via the same interface.***

Multiprocessor system:

Multiple processors that share some common memory.

RAM Sharing → Tight coupling.
HDISK Sharing → Loose coupling.
Systems simply connected → Share Nothing.

Following diagrams explain these architectures:

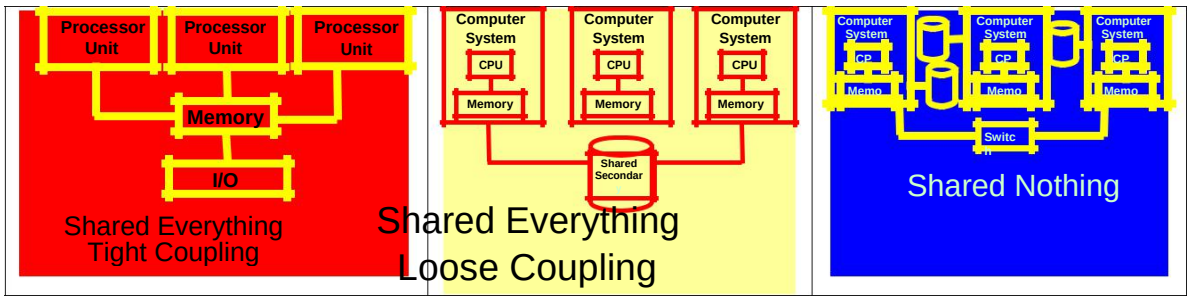


Fig. 1: Different architectures of the Multiprocessor Systems

Centralized C/S System

Data management is carried on a single centralized system. However this data is accessed from different machine (clients). All machines are connected with each other through a communication link (network). This is a very common architecture. The major characteristic of this architecture is that data storage and management is mainly done on the server. As the diagram at the next page shows, the data is associated with a single site, this site is basically the Server, rest of the machines are accessing data from the Server.

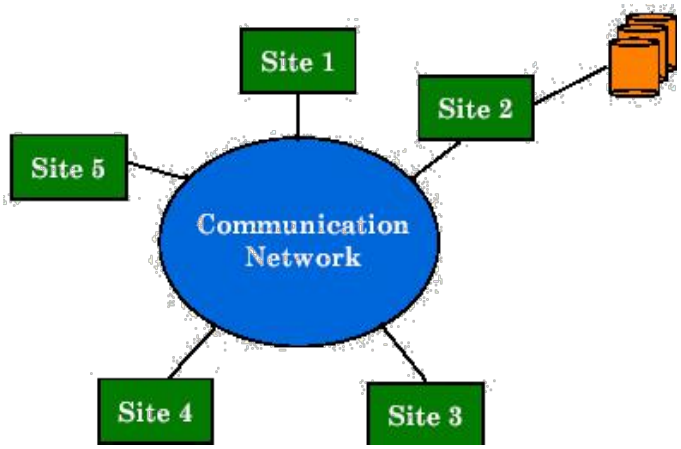


Fig. 2: Centralized Client/Server Architecture

The Distributed Database System

As has been discussed in the previous lecture, the data is managed/manipulated at multiple sites in a DDBS. There are many different architectures of a DDBS; a very general one is given below; this general architecture also establishes a picture of the DDBS in the mind that further helps to understand the working of a DDBS.

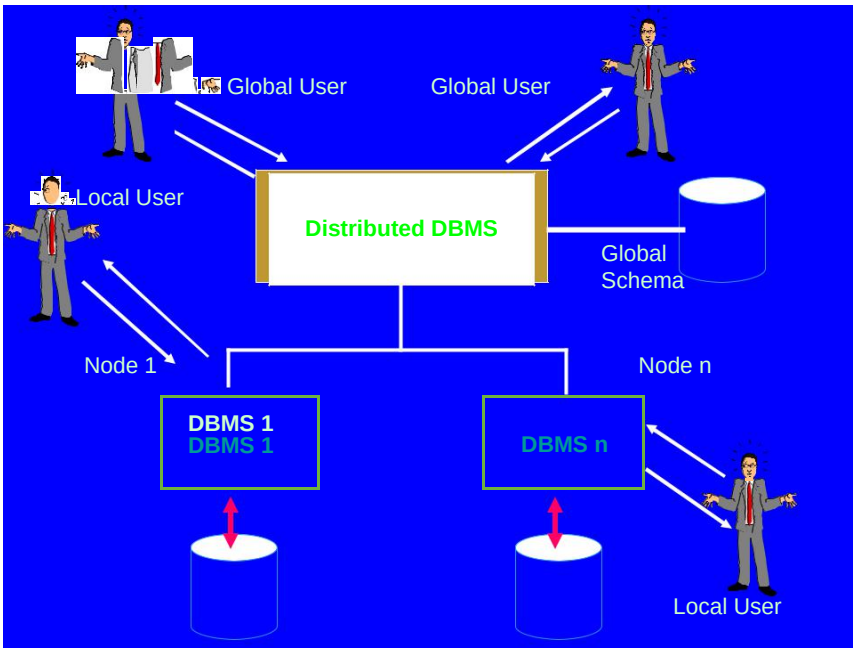


Fig. 3: General Architecture of DDBS

As is clear from the diagram, there are a number of local DBMSs called local nodes. Each local node works independently serving multiple users that are connected to it, these users are called local nodes. At the same time, the local nodes are connected with each other. A layer is superimposed on top of all these connected local DBMSs and that layer is the DDBMS. The DDBMS contains the global schema, that is basically the merger of all local schemas. There is no data underlying the global schema rather the data is

contained with the local DBMSs. The user connected to the DDBMS layer are called global users, as their queries are replied by collecting data from all the local nodes, and this activity of distribution is transparent from the global users.

Reasons for DDBS:

Local units want control over data.

Note:
A person who maintains the data on a local site is called local DBA.

A user on a local site is called the local user. Data is generated locally and accessed locally but there are situations where you require certain reports for which the data must be collected from all sites e.g. A bank wants to know how many customers it has having a balance of more than one core. Local control may be desirable in many situations, as it improves the efficiency in handling/managing the database. That is, local control is helpful in maintenance, backup activities, managing local users accounts etc.

Note:
We may require global access to the data.

There are two basic reasons for the DDBS Environment. To better understand these reasons, we need to see the other (than DDBS) alternative, and that is the centralized database or a client-server environment. Taking the example of our Bank database, if it is a centralized one, it means that the database is stored at a single place, lets suppose, in Pakistan they select a geographically central place, let it be Multan, then the database is stored in Multan, now users from all over Pakistan, whenever they want to use their account, the data will be accessed from the central database (in Multan). If it is a distributed environment, then the Bank can pick two, three, four or more places and each database will be storing the data of its surrounding areas. So the load now is distributed over multiple places. With this scenario in mind, lets discuss the reasons for DDBS:

Reduce telecom cost:

With a centralized database, all the users from all over the country will access it remotely and therefore there would be an increased telecommunication cost. However, if it is distributed then for all users the data is closer and it is local access most of the time. So it reduces the overall telecommunication cost.

Reduce the risk of telecom failures:

With a centralized database, all the working throughout the country depends on the link with the central site. If, due to any reason, link fails then working at all the places will be discontinued. So failure at a single site caused damage at all the places. On the other side, if it is distributed, then failure at a single site will disturb only the users of that site, remaining sites will be working just normal. Moreover, one form of data distribution is

replication where the data is duplicated at multiple places. This particular form of data distribution, further reduces the cost of telecommunication failure.

Schema contains:

What has to be shown to the global user.

How we are going to set data for a thing on each site.
The type of the data stored on each site.

How we are going to merge the data present on different sites.

Note:
Global users are attached to the Distributed DBMS layer.

Promises of DDBS:

If we adopt a DDBS as a solution for a particular application, what features we are going to get:

Transparency:

A transparent system hides the implementation details from the user. There are different types of transparencies, but at this point the focus is on the distribution transparence, that means that the global user will not have any idea that the data that is being provided to him is actually coming from multiple sites, rather he will get the feeling, as if the data is coming just from the machine that he is using. It is a very useful feature, as it saves the global user from the complexity and details of the distributed access.

Data Independence

Major advantage of the database approach is the data independence as the program and data are not dependent on each other i.e. we can change the program with very little or no changes made to the data and vice versa

In a 3-layer architecture the changes on lower level has little or no affect on higher level.

Logical data independence:

If we change the conceptual schema there is little or no effect on the External level.

Physical data independence:

If we change the physical or lower level then there is little or no effect on the conceptual level.

Network transparency:

This is another form of transparency. The user is unaware of even the existence of the network, that frees him from the problems and complexities of network.

Replication transparency:

Replication and fragmentation are the two ways to implement a DDBS. In replication same data is stored on multiple sites example e.g. In case of a bank every branch is holding the data of every other branch. The replication increases the availability of data and reduces the risk of telecom failure. In case of replication, the DDBS hides the replication from the end user, advantage is that user simply gets the benefits of the system and does not need to know the details or to understand the technical details.

Summary

In today's lecture we continued the discussion on distributed systems. We discussed the setups that resemble a DDBS and there we studied distributed file system and multiprocessor systems. In the later type, we have share everything and share nothing systems. We then discussed a centralized C/S system that is also a very popular architecture for the databases. Then we saw different reasons to have a DDBS, the situations where it suits, we compared it with its alternative and studied why a DDBS is useful for certain type of applications. Finally, we saw what advantages we are going to have if we adopt a DDBS solution.

Course Title: Distributed Database Management Systems
Course Code: CS712
Instructor: Dr. Nayyer Masood (nayyerm@yahoo.com)
Lecture No: 4

In previous lecture:

- **Resembling Setups**
- **Why to have a DDBS**
- **Advantages/Promises of DDBS**

In this lecture:

- **Promises of DDBS**
- **Reliability through DDBS**
- **Complicating factors**
- **Problem areas**

Note :
Full replication is when all the data is stored on every site and therefore every access will be local.

Fragmentation transparency:

A file or a table is broken down into smaller parts/sections called fragments and those fragments are stored at different locations. The fragmentation will be discussed in detail in the later lectures. However, briefly, a table can be fragmented horizontally (row-wise) or vertically (column-wise). Hence we have two major types of fragmentations,

horizontal and vertical. Different fragmentations of a table are placed at different locations. The basic objective of fragmentation and placement at different places is to maximize the local access and to reduce the remote access since the later causes cost and delay.

Fragmentation transparency is that a user should not know that the database is fragmented. The concept of fragmentation should be kept hidden from the user.

Note:
DBA designs the architecture of fragments where as once implemented it is managed by DDBMS.

Responsibility of transparency:-

Transparency is very much desirable since it hides all the technical details from the users, and makes the use/access very easy. However, providing transparency involves cost, the cost that has to be bear by someone. More transparency the DDBS environment provides most cost has to be paid. In this section, we are discussing that who is going to pay the cost of transparency, that is, who is responsible of providing transparency. There are three major players in this regard: the Language/Compiler, Operating System and the DDBMS.

Language/compiler: The language or compiler used to develop the front end application programs in a database system is one of the agents that can provide transparency, like VB, VB.Net, JSP, Perl etc. This front end tool is used to perform three major tasks, one, it is used to create the user interface (now a days it is generally a graphical user interface (GUI)), secondly, it is used to perform calculations or any other form of processing, and thirdly, it is used to interact with the DBMS, that is to store and retrieve the data. These three functions are performed by the application programmer(s) by using the tool. Now, from the end-user point of view it does not matter at all that which particular tool has been used to establish this front end GUI. It means the language/compiler component is providing certain form of transparency. It is making end-user free of the mechanism to create the GUI, the mechanism to establish the link with the DBMS for data manipulation, and accessing the data from the database. Rather, it can be said that since the users' interaction with the DBMS or DDBMS is through the application program that has been developed a particular tool (language/compiler), so in a sense we can say that all types of transparencies are provided to the end-user from this tool. Although practically it is not the case, but still apparently it can be said.

Operating system: This layer is not so visible to the end-users most of the time, since it resides below the DDBMS or the language used to develop the application. However, both of them heavily depend on the features supported by the operating system (OS). For example, some of the OSes provide network transparence, so the DBMS or the language uses the features of the OS to provide this facility to the end-users.

DDBMS: This is the third component that provides certain forms of transparencies, for example, the distribution transparency is supported by the DDBMS. Although DDBMS uses the feature of the OS, still the facility is finally presented by the DDBMS. It also provides certain level of transparency.

Practical Situation: Although we have studied three different components that may be responsible for providing transparencies, however, practically the features of all three components are used together to provide a very cohesive and easy to use environment for the users. For example, distributed OS provides the network transparency, the DDBMS provides the fragmentation or replication transparency, the front-end tool provides the transparency of the linking and manipulation of data.

All these features combined, make the DDBMS a viable and user friendly option to use.

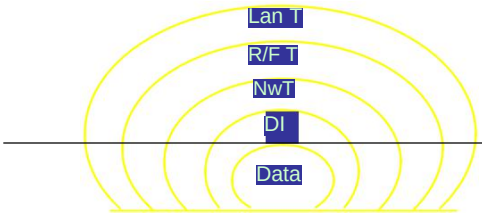


Fig. 1: Different layers of transparencies

Note:
higher will be the cost.

The more the

Reliability of DDBS:

Reliability through distributed transaction: The distributed nature in a DDBS environment reduces the chances of single point of failure and that increases the reliability of the entire system. It means that the entire system does not go down with the failure of a single system as is the case with centralized database systems. It definitely means, however, that in case of DDBS, the site that goes down, the users of that site will definitely suffer but not the entire system.

Concurrency issues: the concurrent access means the access of data by multiple users at the same time. Concurrency issues rise even in simple (client-server) databases, however these issues become more critical in case of a DDBS. Specially, in case of replication, when same data is duplicated at multiple site, then the consistency of data across multiple sites is a serious issue that needs extra care.

Performance Improvement:

The DDBS provides improved performance; the major factors causing this improved performance are data localization and query parallelism.

Data Localization: One of the basic principles of data distribution in a DDBS is that the data should reside at the closest site where it is most frequently accessed. This reduces the communication cost and the delay. However, the DDBS also involves the remote accesses as well and in that case delay is unavoidable, but through maximized data localization we get overall improved performance.

Query Parallelism: This is the second major factor that is the basis of improved performance in a DDBS. Since the DDBS involves multiple systems, a query in certain situations can be executed in parallel, that improves performance. There are two types of query parallelism, that is, the inter-query and intra-query parallelism. The former means that multiple queries can be executed at the same time, whereas the later means that the same query is split across multiple sites and this split components are executed in parallel that increases the throughput. These topics will be discussed in detail in the later lectures.

Complicating factors

There are certain aspects that complicate a DDBS environment. Following are some of those factors.

Selection of the Copy: In case of replication, the selection of the right copy is a complicating factor. That is, as the same data resides at multiple places, which particular site should be accessed against a particular query is an important factor to resolve. One simple solution is to decide on the basis of distance or load. However, the same question arises in a different situation when a particular site goes down. In this case the queries that were originally routed to this particular sites now have to be re-routed. Thus selection of the appropriate copy is an issue that needs extra attention in a DDBS environment.

Failure recovery: Likewise, in case of replication the synchronization of copies after failure has to be dealt with carefully.

Complexity: Since the data is stored at multiple sites and has to be managed the overall system is more complex as compared to a centralized database system.

Cost: A DDBS involves more cost, as the hardware and the trained manpower has to be deployed at multiple sites.

Distribution of Control: The access to the data should be allowed carefully. Rights to access data should be well defined for local sites.

The Problem Areas:

The following areas in DDBS still need more work and are considered problem areas

Database design: All the issues of a centralized database system are applicable in a DDBS but it introduces additional aspect related to data placement, that is, where our sites should be located

Query processing: problem arises in queries executed at multiple sites .e g. what should be done when data from one site is not collected.

Other critical issues include Concurrency Control, Operating System and Heterogeneity. These issues will be discussed in the later lectures.

The diagram shows the interlink between these problem areas.

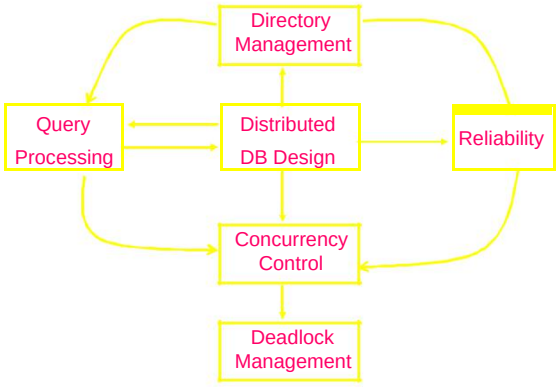


Fig. 2: Relationship among various issues of DDBS

The diagram shows that the DDBS design lies at the heart of all issues. It is linked with most of the issues like Directory Management, Reliability etc. It means that overall

performance of a DDBS mainly depends on the database design. If we could do it efficiently then most of the issues will be working effieciently.

Summary: This lecture continues the discussion on different forms of transparencies including fragmentation transparency. Then the issue of the responsibility for providing the transparency is discussed. Three different components may be considered as the transparency providers, however, practically all three components are used to provide different forms of transparencies and to provide the end-user a user-friendly environment to work with. After this, different issues that complicate the DDBS environment are discussed and finally some problem areas are discussed.

Course Title: Distributed Database Management Systems
Course Code: CS712
Instructor: Dr. Nayyer Masood (nayyerm@yahoo.com)
Lecture No: 5

In previous lecture:

- **Promises of DDBS**
- **Reliability through DDBS**
- **Complicating factors**
- **Problem areas**

In this lecture:

- Background topics
- Data Models
- Relational Data Model
- Normalization

So far we have discussed the introduction to distributed systems and distributed databases in particular. We should have some idea in mind about what a DDBS is and the environment where it suits and pros and cons of the DDBSs. Before moving to topics related to DDBS in details, let us step a little back and discuss some background topics. These topics include the Relational Data Model and the Networks. These are two important underlying topics that will help to have a better understanding of the DDBSs. We start with the Relational Data Model, we will discuss the Networking concepts later.

Data Model: a set of tools or constructs that are used to design a database. There are two major categories of data models. **Record based** data models that have relatively less constructs and the identification of records (defining key) is the responsibility of the user. This type of data models are Network, Hierarchical and Relational Data Models. The record based data models are also called the **Legacy data models.** Whereas the **Semantic** data models are the ones that have more constructs, so they are semantically rich, moreover the identification of records is managed by the system. Examples of such data models are Entity-Relationship and Object-Oriented data models.

The legacy data models have been and are commercially successful. That is, the DBMSs that are based on these data models have been mostly used for the commercial puposes. There are different reasons behind this. Like, the network and hierarchical data models based DBMSs were used because these data models are the initial ones. They were the ones who replaced the traditional file processing systems. So success of the initial DBMSs means the acceptance of database approach against then existing file system approach. This is the era from late 1960s to early 1980s. The relational data model was proposed in 1970 and by late 1980s it started gaining popularity and dominance. Today it is the unchallenged most popular and most widely (perhaps only) used data model. We have so many different DBMSs based on the relational data model (RDM), like, Oracle, Sybase, DB2, SQL Server and many more. The two major strengths of RDM that gave it so success are :

It is very simple as it has only on structure i.e a relation or a table.
It has a strong mathematical foundation.

The semantic data model, like Object-Oriented data models could not get popularity as commercial choice as they lack the same two features. So on one side OO data model was a bit difficult to understand due to its complexity and secondly it is not that well defined due to lack of mathematical support. However, semantic data models are heavily used as the design tool, that is, for the database design, specially for conceptual and external schemas, the semantic data models are used. The reason for this choice is that they provide more constructs and a database design in semantic data models is more expressive and hence is easy to understand.

Since the RDM is the dominant in the market, so our background discussion focuses only on this data model.

Relational Data Model

A data model that is based on the concept of relation or table. The concept of relation has been taken from the mathematical relation. In the databases, the same relation represented in a two dimensional way is called table, so relation or table represent the same thing. The RDM has got three components:

1. Structure → support for storage of data and RDM supports only a single structure and that is a relation or a table
2. Manipulation language → The language to access and manipulate data from the database. The SQL (Structured Query Language) has been accepted as the standard language for RDM. The SQL is based on relational algebra and relational calculus
3. Support for integrity constraints: The RDM support two major integrity constraints such that they have become a part of the data model itself. It means that any DBMS that is base on the RDM must support these two constraints and they are
   > Entity integrity constraint
   > Referential integrity constraint

These two constraints help enforce database consistency. The relational DBMSs provide support for all these three components.

Note:
Business rules are enforced through integrity constraints.

A relation R defined over domain D1, D2……Dn is a set of n- tuples < d1,d2 …….dn> such that < d1 D1, d2 D2, ……dn Dn>. The structure of a relation is described through a relation scheme which is of the form

R(A1:D1, A2:D2, ……, An:Dn) or simply R(A1, A2, ….., An) where A1, A2, …., An are the names of the attributes and D1, D2,….., Dn are their respective domains. Two example relation schemes are given below

- EMP (ENo:Integer, EName:Text, EAdd:Text, ESal:Number) or it can be simply written as EMP (ENo, EName, EAdd, ESal)
- Project (PNo: Char(8), PName:Text, bud:Number, stDate:Date) or simply Project (PNo, PName, bud, stDate)

Each of the attribute in these two relations has a domain and the domains need not to be distinct. A relation is collection of tuples based on the relation scheme. An example relation for the above EMP relation scheme is given below

ENo	EName	EAdd	ESal
1	Mansoor Shiraz	Lahore	25000
2	Ali Munib	Lahore	30000
3	Mian Javed	Lahore	50000
4	Salman Shahid	Karachi	34000

Keys: Key is a status that is assigned to a single or collection of attributes in a relation. There are different types of keys for different purposes, however most important role performed by keys is the unique identification. Different types of keys are mentioned below:

Super Key: An attribute or set of attributes whose value can be used to uniquely identify each record/tuple in a relation is super key. For example, in the EMP table ENo is a super key since its value identifies each tuple uniquely. However, ENo, EName both jointly are also the super key, likewise other combinations of attributes with ENo are all super keys.

Candidate Key: is the minimal super key, that is a super key whose no proper subset itself is a super key. Like in the above example ENo is a super key and a candidate as well. However, the ENo, EName jointly is super key not the candidate key as it has a proper subset (ENo) that is a super key. **Primary key:** The successful/selected candidate key is called the primary key, remaining candidate keys are called **alternate keys**. If a table has got a single candidate key then the same will be used as the primary key, however, if there are multiple candidate keys then we have to choose one of them as the primary key and then the others will be called the alternate keys. In our EMP table, we have got only one candidate key, so the same is the primary key and there is no alternate key. **Secondary key** is the attribute(s) that are used to access data from the relation but whose value is not necessarily unique, like EName. **Foreign key** is the attribute(s) in one table that is/are primary key in the other table. The purpose of defining foreign key is the enforcement of referential integrity constraint that has been mentioned above.

Table:

- Relation represented in a two dimensional form.
- After defining schema it is populated and we get the table.
- Tuples are rows and attributes are columns.
- Attributes have domain
- Attributes can contain NULL values. (NULL does not mean zero)
- If a primary key is based on more than one attributes then none of the attributes can have null value.

Normalization: is a step by step process to produce efficient and smart database design that makes is easier to maintain the consistency of the data base. Following are some main points about normalization:

- Strongly recommended not a must. Performed after the logical database design. That is, once we have finalized our design then we fine tune it through normalization process
- Deals with anomalies, that is, helps to convert the database design into an anomaly free design
- Anomalies are conditions which may make database incorrect or inconsistent. These anomalies are:
  - Duplication
  - Insertion anomaly
  - Update anomaly
  - Deletion anomaly

Normalization removes the anomalies and helps the maintenance of the database a lot easier. The normalization is mainly based on the decomposition of relations, that is, a relation is analyzed for the existence of anomalies if some are found then it is decomposed into smaller relations that do not have those anomalies. The concept of universal relation is also used in this regard, that is, the design is considered to be consisting of a single large table called the universal table, then this table is decomposed step by step following the normalization process. Finally, we get our database design which is fully normalized.

Normal Forms: There are different normal forms

First Normal Form

Second Normal Form

Third Normal Form

BCNF (Boyce Code Normal Form)
Fourth Normal Form

Fifth Normal Form

The two concepts regarding normalization are:

Lossless decomposition: When a relation is decomposed into two relations there should be no loss of information. i.e when we combine the decomposed relations together we get the original relation. This loss of information is concerned both with the relational schema as well as data.

Dependency Preservation: The same dependency should be maintained after the decomposition of a relation.

Dependency structure: Normalization is based on dependencies. Up to BCNF on FDs, further normal forms are based on multi-valued dependency and project-join dependency.

Note:
Dependencies are identified not designed.

A relation R defined on attributes R(A1, A2, …., An), if X A, Y A, if for each value of X there is a unique value in Y, then X functionally determines Y, X → Y and X → is called a functional dependence or FD. This concludes are discussion today, we will continue later inshAllah.

Summary: From today's lecture we are starting discussion on the background topics of the DDBSs and that are the relational data model and the networking. We started from the former one. The data model is important as it is the basis for every DBMS. There are record based and semantic data models. Most popular among the data models is the relational data model that is based on the mathematical relations. Relations are defined through relation schemes that are later populated to form relations. Relational database design is finalized through normalization which is based on the dependencies. Functional dependencies are the basis up to BCNF which is written in the form of X → Y.

Course Title: Distributed Database Management Systems
Course Code: CS712
Instructor: Dr. Nayyer Masood (nayyerm@yahoo.com)

Lecture No: 6

In previous lecture:

- Background topics
- Data Models
- Relational Data Model
- Normalization

In this lecture:

- Normalization
- Dependency and its types
- Different Normal forms

Dependencies:

Functional dependency: A functional dependency exists when the value of one of more attribute(s) can be determined from the value(s) of one or more other attribute(s). So, if we can get a unique value of an attribute B against a value of attribute(s) A, then we say that B is functionally dependent of A or A functionally determines B. Unique value means that if we have a value of A then there is definitely a particular/single/unique value of B. There is no ambiguity or no question of getting two or different values of B against one value of A.

For example, given the relation EMP(empNo, empName, sal), attribute empName is functionally dependant on attribute empNo. If we take any value of empNo, then we are going to have a unique or exactly one value of empName if it exisits. Lets say we have an empNo 'E00142', if this empNo exists then definitely we are going to get exactly one name against this empNo, lets say it is 'Murad'. According to this FD, it is not possible that against 'E00142', we get 'Murad' and also 'Waheed'. To make it more clear, lets describe the same thing inversely, if we have an empName 'Murad' and want to know the empNo, will we get a single empNo. No. Why? Because there could be a 'Murad' with empNo as 'E00142' and another with 'E00065' and yet some others. So this is what is meant by an FD, like empNo → empName. Same is the case with other attribute 'sal', if it is included in FD, that is, empNo → empName, sal.

Note:
Determinant is defined as the attribute on left hand side of the arrow in a functional dependency (FD) e.g. A is determinant in the following dependency A → B

Full Functional dependency: An FD in which the dependent attributes are determined by all of the determinant not by any subset of it (determinant). Obviously, if the determinant in an FD (A → B) consists of a single attribute, then it is definitely a full functional dependency, or we can say that B is fully functionally dependent on A. If, however, A is a set of attributes, then we have to see whether it is full functional dependency or not. How are going to know whether an FD is FFD or not? By seeing the other FDs of the same relation. If, for example, we have a relation R(a, b, c, d, e, f) and the FDs

a, b → c, d, e and e → f, in this case first FD has two attributes in the determinant, and there is no other FD in which either a or b alone determines any of the attributes c, d, or e, so it is an FFD. As we can see that there is another FD on this relation but this is a separate independent FD, it does not make the first one as a non-FFD.

Partial Functional dependency: An FD in which one or more non dependent attributes are also functionally dependent on part of the determinant. For example, consider the table Staff (StaddID, Name, BranchID) and the FDs:

StaffID, Name → BranchID
StaffID → BranchID

First FD shows that BranchID is functionally dependent on StaffID and Name, jointly. However, from the second FD, it is clear that BranchID is functionally dependent on a

StaffID alone as well, which is subset of StaffID, Name. So we will say that BranchID is partially dependent on StaffID, Name, that is, the first FD is a partial FD.

Transitive Dependency: If for a relation R, we have FDs a → b and b → c then it means that a → c, where b and c both are non-key attributes.

Normal forms

Let us now discuss the normal forms. Brief definitions of the Normal forms:

First Normal Form

Table must contain atomic/single values in the cells

Second Normal Form

- Relation must be in 1NF
- Each non-key attribute should be fully functionally dependent on key, that is, there exists no partial dependency

Third Normal Form

- Relation must be in 2NF
- There should be no transitive dependency

BCNF

For every FD X → Y in R, X is a super key or a relation in which every determinant is a candidate key.

Normalization Example

Table With Repeating Groups:

Project ID	Project Name	Employee ID	Employee Name	Rate Category	Hourly Rate
1023	Pakistan Travel Guide	12 13 14	Ali Mansoor Javed	A B C	60 50 40
1024	Virtual University Database	12 15	Ali Faisal	A B	60 50

This table is not in the first normal form as the cells contain multiple values or repeating groups. To convert it into first normal form, we need to expand it as in the following table:

First Normal Form:

Employee-Project Table:

ProjectID	ProjectName	EmployeeID	EmployeeName	RateCategory	HourlyRate
1023	Pakistan Travel Guide	12	Ali	A	60
1023	Pakistan Travel Guide	13	Mansoor	B	50
1023	Pakistan Travel Guide	14	Javed	C	40
1024	Virtual University Database	12	Ali	A	60

1024	Virtual University Database	15	Faisal	B	50

The table not is in first normal form and the FDs on this table are

1. ProjectID → ProjectName
2. EmployeeID → EmpoyeeName, RateCategory, HourlyRate
3. RateCategory → HourlyRate

None of the determinants can work as primary key, so out of given FDs we establish the FD

4. ProjectID, EmployeeID → ProjectName, EmployeeName, RateCategory, HourlyRate

Now the determinant of last FD is the key of the table.

Second Normal Form:

Cleary the above table is not in the second normal form since there exists the partial dependency through the FDs 1, 2 and 4. To bring it into second normal form, we decompose the table into the following tables:
Employee Table:

Employee ID	Employee Name	Rate Category	Hourly Rate
12	Ali	A	60
13	Mansoor	B	50
14	Javed	C	40
15	Faisal	B	50

Project Table:

Project ID	Project Name
1023	Pakistan Travel Guide
1024	Virtual University Database

Employee-Project Table:

Project ID	Employee ID
1023	12
1023	13
1023	14
1024	12
1024	15

All of the above three tables are in second normal form.

Third Normal Form:

Continuing the normalization process, we have to examine each of the three tables for the third normal form based on the four FDs mentioned above. According to those FDs the Project and Employee-Project tables are in third normal form, as they are in second

normal form and there is no transitive dependency. However, the Employee table is not in third normal form as there is a transitive dependence, that is,

- EmployeeID → EmpoyeeName, RateCategory, HourlyRate
- RateCategory → HourlyRate

To bring it in third normal form, we decompose it once again and get the tables:

Employee Table:

Employee ID	Employee Name	Rate Category
12	Ali	A
13	Mansoor	B
14	Javed	C
15	Faisal	B

Rate Table:

Rate Category	Hourly Rate
A	60
B	50
C	40

Project Table:

Project ID	Project Name
1023	Pakistan Travel Guide
1024	Virtual University Database

Employee-Project Table:

Project ID	Employee ID
1023	12
1023	13
1023	14
1024	12
1024	15

All of the above four tables are in third normal form.

Higher Normal Forms

Multi-valued Dependency: A relation R defined on attributes A(A1, A2, ……. An) if X A, Y A, Z A, if for each value of X, there is a unique value of (Y, Z) and Z depends only on X then X →→ Y and X →→ Y, that is X multi-determines Y and X multi-determines Z.

Fourth Normal Form: A relation is in 4 NF if it is in BCNF and there is no multi valued dependency. Consider the EMP table below:

EMP (empNo, projId, skill)

E001	P1	Welder
E001	P2	Electrician
E001	P1	Electrician
E001	P2	Welder

We have the MVDs empNo $\rightarrow\rightarrow$ projId
empNo $\rightarrow\rightarrow$ skill

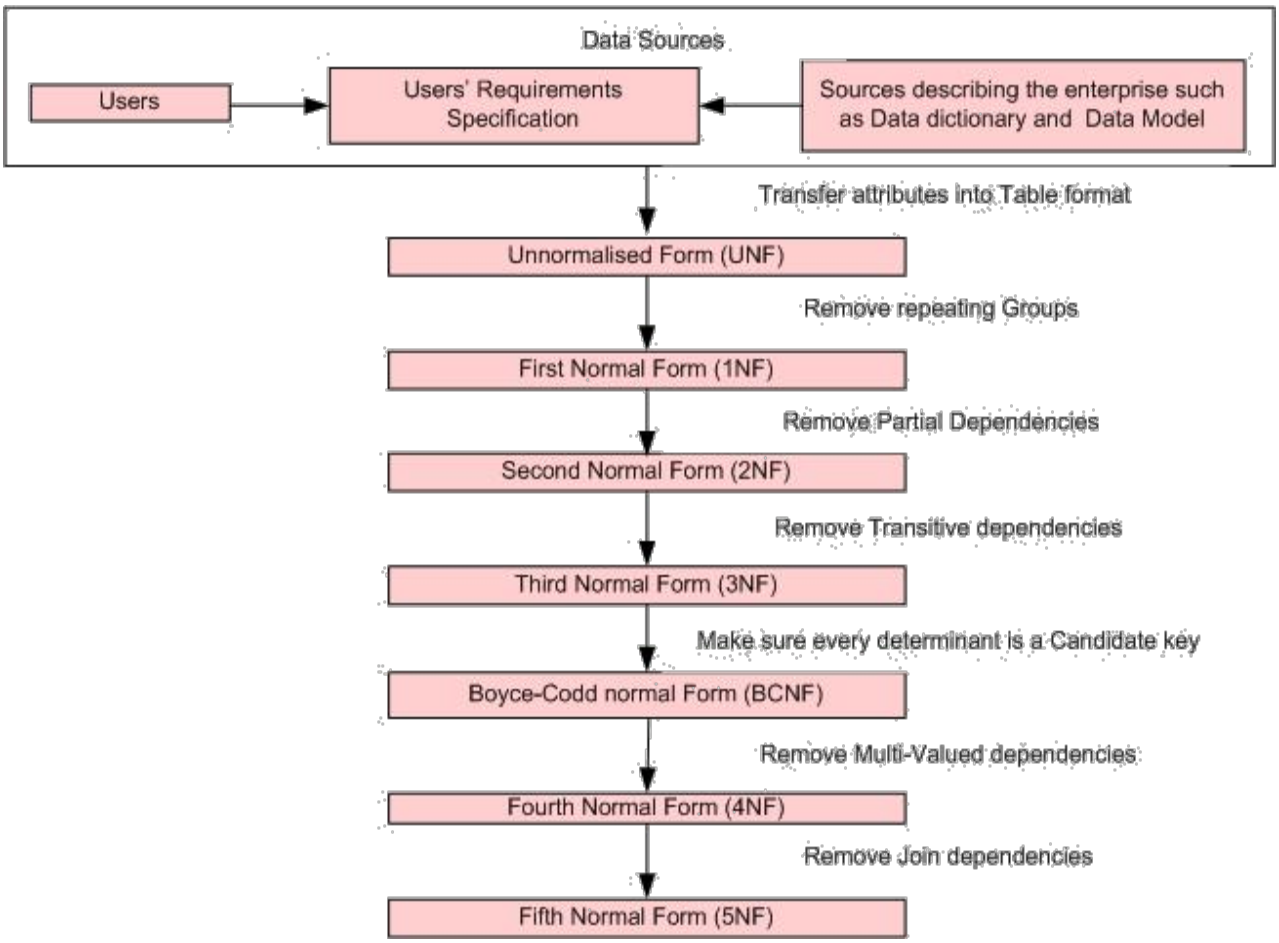
First MVD says that employee works on multiple projects and second MVD says that an employee has multiple skills. Moreover, projId and skill are independent of each other, that is, if an employee works on certain projects, it is independent of the fact that which skills he possesses and other way round. These MVDs introduce certain anomalies and to get rid of them we will have to split this table into two:

EMP(empNo, projId)
EMPSKL(empNo, skill)

Now both these tables are in fourth normal form.

Projection Join Dependency: A relation that has a join dependency cannot be divided into two or more relation such that the results table can be recombined to form the original table. A relation R is in 5 NF of every join dependency implied by the keys of R. We need to have a common attributes in the two decomposition of a relation. Fifth normal form basically sees the possibility of decomposing a table further to have more efficient tables, however, this decomposition should not be based on the keys since that will not bring any efficiency. If however, there exists the possibility of decomposing a table based on non-key attribute such that the join of the decomposed tables brings back the original table without any loss of information, then we should go for that decomposition. There is no formal way of normalizing a table in fifth normal form. There can be one hit and trial method though, we can split a table keeping any non-key attribute common and then we can check whether this split is lossless or not, if it is lossless then we have transformed the table into fifth normal form, if it is not lossless then we can not decompose it.

Normalization Guidelines



Integrity Rules:

Integrity rules are applied on data to maintain the consistency of the database, or to make it sure that the data in the database follows the business rules. There are two types of integrity rules

- Structural Integrity Constraints

- Depends on data storage
- Declared as part of data model
- Stored as part of schema
- Enforced by DBMS itself

- **Behavioral Integrity Constraint:**

- These are the rules/ semantics of the business
- Depends on business
- Enforced through application program

Summary: In this lecture we continued our discussion on the relational data model, and discussed the normalization. Its purpose, methodology and timing in the development process. Normalization is based on the dependencies that exist in the system. The analyst has to identify those dependencies and then use them upon his database design to make a better design. The normal forms upto BCNF are based on FDs and higher forms are based on other forms of dependencies. Finally we started discussing integrity constraints part of the relational data model that will be continues in the next lecture.

Course Title: **Distributed Database Management Systems**
Course Code: **CS712**
Instructor: **Dr. Nayyer Masood (nayyerm@yahoo.com)**
Lecture No: **7**

In previous lecture:

- Normalization
- Dependency and its types
- Different Normal forms

In this lecture:

- Integrity Constraints
- Relational Algebra
- Selection, Projection Operations
- Cartesian Product, Theta Join, Equi-Join, Natural Join

Structural Integrity Constraint:

As has been discussed in the previous lecture, these constraints are imposed on the structure of data. These constraints are very basic; in the sense that they are applicable to almost all real-world applications. That is why they have been included in the definition of the relational data model. As a result all relational DBMSs support these constraints. There are two types:

Entity Integrity Constraint: The primary key cannot be NULL even if the primary key consists of more than one attributes, none of the attributes can have NULL. By its definition, the primary key cannot have the duplicate values, the entity integrity constraint enforces this additional constraint on it.

Referential Integrity Constraint: A rule that addresses the validity of reference by one object in database to some other object in the database. It involves the concept of **foreign key**, which is an attribute or set of attributes that appear as a non key attribute in one relation and as a primary key attribute in another relation. The **referential integrity constraint** states that the value of foreign key can either be NULL or it must match with the value of primary key in its home relation. This constraint is of very important to maintain the consistency and correctness of the database, as the tables are decomposed during the normalization process and the decomposed parts most of the time need to maintain a link in order to retain the information in the original table. For this purpose, it is a must that the link between the split tables is a valid one. The referential integrity constraint helps to maintain this valid link

Relational Data Languages

This is the third component of relational data model. We have studied structure, which is the relation, integrity constraints both referential and entity integrity constraint. Data manipulation languages are used to carry out different operations like insertion, deletion or creation of database. Following are the two types of languages:

Procedural Languages: These are those languages in which what to do and how to do on the database is required. It means whatever operation is to be done on the database that has to be told that how to perform.

Non -Procedural Languages: These are those languages in which only what to do is required, rest how to do is done by the manipulation language itself.

Structured query language (SQL) is the most widely language used for manipulation of data. But we will first study Relational Algebra and Relational Calculus, which are procedural and non – procedural respectively.

Relational Algebra: Following are few major properties of relational algebra:

Relational algebra operations work on one or more relations to define another relation leaving the original intact. It means that the input for relational algebra can be one or more relations and the output would be another relation, but the original participating relations will remain unchanged and intact.Both operands and results are relations, so output from one operation can become input to another operation. It means that the input and output both are relations so they can be used iteratively in different requirements.

Allows expressions to be nested, just as in arithmetic. This property is called closure.

There are five basic operations in relational algebra: Selection, Projection, Cartesian product, Union, and Set Difference.

These perform most of the data retrieval operations needed.

It also has Join, Intersection, and Division operations, which can be expressed in terms of 5 basic operations.

The Select Operation: The select operation is performed to select certain rows or tuples of a table, so it performs its action on the table horizontally. The tuples are selected through this operation using a predicate or condition. This command works on a single table and takes rows that meet a specified condition, copying them into a new table. Lower Greek letter sigma (····) is used to denote the selection. The predicate appears as subscript to ····. The argument relation is given in parenthesis following the ····. As a result of this operation a new table is formed, without changing the original table. As a result of this operation all the attributes of the resulting table are same, which means that degree of the new and old tables are same. Only selected rows / tuples are picked up by the given condition. While processing a selection all the tuples of a table are looked up and those tuples, which match a particular condition, are picked up for the new table. The degree of the resulting relation will be the same as of the relation itself.

$$| \sigma | = | r(R) |$$

The select operation is commutative, which is as under: -

$$\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R))$$

If a condition 2 (c2) is applied on a relation R and then c1 is applied, the resulting table would be equivalent even if this condition is reversed that is first c1 is applied and then c2 is applied.

For example there is a table STUDENT with five attributes.

STUDENT

stId	stName	stAdr	prName	curSem
S1020	Sohail Dar	H#14, F/8-4,Islamabad.	MCS	4
S1038	Shoaib Ali	76 Jalilabad Multan	BCS	3
S1015	Tahira Ejaz	268 Justice Hameed Multan	MCS	5
S1018	Arif Zia	H#10, E-8, Islamabad.	BIT	5

Fig. 1: An example STDUDENT table

The following is an example of select operation on the table STUDENT:

$$\sigma_{\text{Curr\_Sem} > 3}(\text{STUDENT})$$

The components of the select operations are clear from the above example; σ is the symbol being used (operato), "curr_sem > 3" written in the subscript is the predicate and STUDENT given in parentheses is the table name. The resulting relation of this command would contain record of those students whose semester is greater than three as under:

$$\sigma_{\text{Curr\_Sem} > 3}(\text{STUDENT})$$

stId	stName	stAdr	prName	curSem
S1020	Sohail Dar	H#14, F/8-4,Islamabad.	MCS	4
S1015	Tahira Ejaz	H#99, Lala Rukh Wah.	MCS	5
S1018	Arif Zia	H#10, E-8, Islamabad.	BIT	5

Fig. 2: Output relation of a select operation

In selection operation the comparison operators like <, >, =, <=, >=, <> can be used in the predicate. Similarly, we can also combine several simple predicates into a larger predicate using the connectives *and* ( ) and *or* ( ).

The Project Operator The Select operation works horizontally on the table on the other hand the **Project operator** operates on a single table vertically, that is, it produces a vertical subset of the table, extracting the values of specified columns, eliminating duplicates, and placing the values in a new table. It is unary operation that returns its argument relation, with certain attributes left out. Since relation is a set any duplicate rows are eliminated. Projection is denoted by a Greek letter ( ). While using this operator all the rows of selected attributes of a relation are part of new relation. For example consider a relation FACULTY with five attributes and certain number of rows.

FACULTY

FacId	facName	Dept	Salary	Rank
F2345	Usman	CSE	21000	lecturer
F3456	Tahir	CSE	23000	Asst Prof
F4567	Ayesha	ENG	27000	Asst Prof
F5678	Samad	MATH	32000	Professor

Fig. 3: An example FACULY table

If we apply the projection operator on the table for the following commands all the rows of selected attributes will be shown, for example:

(FACULTY)

FacId, Salary

FacId	Salary
F2345	21000
F3456	23000
F4567	27000
F5678	32000

Fig. 4: Output relation of a project operation on table of figure 3

Some other examples of project operation on the same table can be:

Fname, Rank (Faculty)

Facid, Salary,Rank (Faculty)

Composition of Relational Operators: The relational operators like select and project can also be used in nested forms iteratively. As the result of an operation is a relation so this result can be used as an input for other operation. However, we have to be careful about the nested command sequence. Although the sequence can be changed, but the required attributes should be there either for selection or projection.

The Union Operation: **We will now study the binary operations, which are also called as set operations. The first requirement for union operator is that the both the relations should be union compatible. It means that relations must meet the following two conditions:**

Both the relations should be of same degree, which means that the number of attributes in both relations should be exactly same

The domains of corresponding attributes in both the relations should be same. Corresponding attributes means first attributes of both relations, then second and so on.

It is denoted by U. If R and S are two relations, which are union compatible, if we take union of these two relations then the resulting relation would be the set of tuples either in R or S or both. Since it is a set, so there are no duplicate tuples. The union operator is commutative which means: -

R U S = S U R

For Example there are two relations COURSE1 and COURSE2 denoting the two tables storing the courses being offered at different campuses of an institute. Now if we want to know exactly what courses are being offered at both the campuses then we will take the union of two tables:

COURSE1

crId	progId	credHrs	courseTitle
C2345	P1245	3	Operating Sytems
C3456	P1245	4	Database Systems
C4567	P9873	4	Financial Management
C5678	P9873	3	Money & Capital Market

COURSE2

crId	progId	credHrs	courseTitle
C4567	P9873	4	Financial Management
C8944	P4567	4	Electronics

COURSE1 U COURSE2

crId	progId	credHrs	courseTitle
C2345	P1245	3	Operating Sytems
C3456	P1245	4	Database Systems
C4567	P9873	4	Financial Management
C5678	P9873	3	Money & Capital Market
C8944	P4567	4	Electronics

Fig. 5: Two tables and output of union operation on those table

So in the union of above two courses there are no repeated tuples and they are union compatible as well

The Intersection Operation: The intersection operation also has the requirement that both the relations should be union compatible, which means they are of same degree and same domains. It is represented by . If R and S are two relations and we take intersection of these two relations then the resulting relation would be the set of tuples, which are in both R and S. Just like union intersection is also commutative.

R S = S R

For Example, if we take intersection of COURSE1 and COURSE2 of figure 5 then the resulting relation would be set of tuples, which are common in both.

COURSE1 COURSE2

crId	progId	credHrs	courseTitle
C4567	P9873	4	Financial Management

Fig. 6: Output of intersection operation on COURSE1 and COURSE 2 tables of figure 5

The union and intersection operators are used less as compared to selection and projection operators.

The Set Difference Operator

If R and S are two relations which are union compatible then difference of these two relations will be set of tuples that appear in R but do not appear in S. It is denoted by (-) For example if we apply difference operator on Course1 and Course2 then the resulting relation would be as under:

COURSE1 – COURSE2

CID	ProgID	Cred_Hrs	CourseTitle
C2345	P1245	3	Operating Sytems
C3456	P1245	4	Database Systems
C5678	P9873	3	Money & Capital Market

Fig. 7: Output of difference operation on COURSE1 and COURSE 2 tables of figure 5

Cartesian Product: The Cartesian product needs not to be union compatible. It means they can be of different degree. It is denoted by X. suppose there is a relation R with attributes (A1, A2,...An) and S with attributes (B1, B2……Bn). The Cartesian product will be:

R X S

The resulting relation will be containing all the attributes of R and all of S. Moreover, all the rows of R will be merged with all the rows of S. So if there are m attributes and C rows in R and n attributes and D rows in S then the relations R x S will contain m+n columns and C * D rows. It is also called as cross product. The Cartesian product is also commutative and associative. For Example there are two relations COURSE and STUEDNT

COURSE	
crId	**courseTitle**
C3456	Database Systems
C4567	Financial Management
C5678	Money & Capital Market

STUDENT	
stId	*stName*
S101	Ali Tahir
S103	Farah Hasan

COURSE X STUDENT

crId	courseTitle	stId	stName
C3456	Database Systems	S101	Ali Tahir
C4567	Financial Management	S101	AliTahr
C5678	Money & Capital Market	S101	Ali Tahir

C3456	Database Systems	S103	Farah Hasan
C4567	Financial Management	S103	Farah Hasan
C5678	Money & Capital Market	S103	Farah Hasan

Fig. 8: Input tables and output of Cartesian product

Join Operation: Join is a special form of cross product of two tables. In Cartesian product we join a tuple of one table with the tuples of the second table. But in join there is a special requirement of relationship between tuples. For example if there is a relation STUDENT and a relation BOOK then it may be required to know that how many books have been issued to any particular student. Now in this case the primary key of STUDENT that is stId is a foreign key in BOOK table through which the join can be made. We will discuss in detail the different types of joins in our next lecture. Following are the different types of joins: -

Theta Join
Equi Join
Semi Join
Natural Join
Outer Joins

We will now discuss them one by one

Theta Join

In theta join we apply the condition through the select operation on relation and then only those selected rows are used in the cross product with second relation. It means that in normal cross product all the rows of one relation are crossed with all the rows of second relation, but here only selected rows of a relation are made cross product with second relation. It is denoted as under: -

$$R \; X_{\theta} \; S$$

If R and S are two relations then θ is the condition, which is applied for select operation on one relation and then only selected rows are cross product with all the rows of second relation. For Example there are two relations of faculty and course, now we will first apply select operation on the faculty relation for selection certain specific rows then these rows will have across product with Course relation, so this is the difference in between cross product and theta join. We will now see first both the relation their different attributes and then finally the cross product after carrying out select operation on relation. From this example the difference in between cross product and theta join becomes clear.

FACULTY

facId	facName	Dept	Salary	Rank
F234	Usman	CSE	21000	lecturer
F235	Tahir	CSE	23000	Asso Prof
F236	Ayesha	ENG	27000	Asso Prof
F237	Samad	ENG	32000	Professor

COURSE

crCode	crTitle	fID
C3456	Database Systems	F234
C3457	Financial Management	
C3458	Money & Capital Market	F236
C3459	Introduction to Accounting	F237

(σ rank = 'Asso Prof'(FACULTY)) X COURSE

facId	facName	Dept	Salary	Rank	crCode	crTitle	fID
F235	Tahir	CSE	23000	Asso Prof	C3456	Database Systems	F234
F235	Tahir	CSE	23000	Asso Prof	C3457	Financial Management	
F235	Tahir	CSE	23000	Asso Prof	C3458	Money & Capital Market	F236
F235	Tahir	CSE	23000	Asso Prof	C3459	Introduction to Accounting	F237
F236	Ayesha	ENG	27000	Asso Prof	C3456	Database Systems	F234
F236	Ayesha	ENG	27000	Asso Prof	C3457	Financial Management	
F236	Ayesha	ENG	27000	Asso Prof	C3458	Money & Capital Market	F236
F236	Ayesha	ENG	27000	Asso Prof	C3459	Introduction to Accounting	F237

Fig. 9: Output of Theta Join

Equi-Join: ***In this example after fulfilling the select condition of Associate professor on faculty relation then it is cross product with course relation Equi – Join***

This is the most used type of join. In equi – join rows are joined on the basis of values of a common attribute between the two relations. It means relations are joined on the basis of common attributes between the relations which are meaningful. This means on the basis of primary key, which is a foreign key in another relation. Rows having the same value in the common attributes are joined. Common attributes appear twice in the output. It means that the attributes, which are common in both relations, appear twice, but only those rows, which are selected. Common attribute with the same name is qualified with the relation name in the output. It means that if primary and foreign keys of two relations are having the same names and if we take the equi – join of both then in the output relation the relation name will precede the attribute name. For Example, if we take the equi – join of faculty and course relations then the output would be as under: -

FACULTY ⋈ FACULTY.facID=COURSE.fID COURSE

facId	facName	Dept	Salary	Rank	crCode	crTitle	fID
F234	Usman	CSE	21000	lecturer	C3456	Database Systems	F234
F236	Ayesha	ENG	27000	Asso Prof	C3458	Money & Capital Market	F236
F237	Samad	ENG	32000	Professor	C3459	Introduction to Accounting	F237

Now in this example after taking equi – join only those tuples are selected in the output whose values are common in both the relations.

Natural Join: This is the most common and general form of join. If we simply say join, it means the natural join. It is same as equi – join but the difference is that in natural join, the common attribute appears only once. Now, it does not matter which common attribute should be part of the output relation as the values in both are same. For Example if we take the natural join of faculty and course the output would be as under: -

FACULTY ⋈ **FACULTY.facID=COURSE.fID** **COURSE**

facId	facName	Dept	Salary	Rank	crCode	crTitle
F234	Usman	CSE	21000	lecturer	C3456	Database Systems
F236	Ayesha	ENG	27000	Asso Prof	C3458	Money & Capital Market
F237	Samad	ENG	32000	Professor	C3459	Introduction to Accounting

Summary: Focus of this lecture was relational data languages and we discussed one of them and that is the relational algebra. It consists of five basic operations, two of them are unary, that is, they operate on one relation and three of them are binary. Further operations are also derived from these basic operations. We have discussed most of them, a couple of operations are remaining and that we will discuss in next lecture inshAllah.

Course Title: **Distributed Database Management Systems**
Course Code: **CS712**
Instructor: **Dr. Nayyer Masood (nayyerm@yahoo.com)**
Lecture No: **8**

In previous lecture:

- Integrity Constraints
- Relational Algebra
- Selection, Projection Operations
- Cartesian Product, Theta Join, Equi-Join, Natural Join

In this lecture:

- Semi-join, division
- Networks and data communication
- Network topologies

Relational Algebra Operations:-

Semi join: In semi join, first we take the natural join of two relations then we project the attributes of first table only. So after join and matching the common attribute of both

relations only attributes of first relation are projected. For Example considering the relations Faculty and Course given below:

FACULTY

facId	facName	Dept	Salary	Rank
F234	Usman	CSE	21000	Lecturer
F235	Tahir	CSE	23000	Asso Prof
F236	Ayesha	ENG	27000	Asso Prof
F237	Samad	ENG	32000	Professor

COURSE

crCode	crTitle	fID
C3456	Database Systems	F234
C3457	Financial Management	
C3458	Money & Capital Market	F236
C3459	Introduction to Accounting	F237

Fig. 1: Example Faculty and Course tables

If we take the semi join of these two relations then the resulting relation would be as under:-

FACULTY ⋉ COURSE

facId	facName	Dept	Salary	Rank
F234	Usman	CSE	21000	lecturer
F236	Ayesha	ENG	27000	Asso Prof
F237	Samad	ENG	32000	Professor

Fig. 2: Semi-join of the two relations

Now the resulting relation has attributes of first relation only after taking the natural join of both relations. Advantage of semi-join is that we get the tuples of one relation that are involved in relation with other one. As in above example, we get the names of the faculty members who are involved in teaching, that is, the tuples of the Faculty that are related with the tuples in Course.

Division: The division of R with degree r with S with degree s is the set of (r-s) tuples t such that for all s- tuples u in the tuple tu is in R.

Suited to queries that include the phrase “for all”.

Let *r* and *s* be relations on schemas R and S respectively where

- $R = (A1, \ldots, Am, B1, \ldots, Bn)$
- $S = (B1, \ldots, Bn)$, and
- $R - S = (A1, \ldots, Am)$

Then the result of r s is a relation on schema

$$r \quad s = \{ t \mid t \quad _{R\text{-}S}(r) \quad \text{á } u \quad s (tu \quad r) \}$$

eNo	pNo	pName	budget
E1	P1	Bridge	11.5m
E2	P1	Bridge	11.5m
E1	P3	Tower	10.2m
E4	P2	Mosque	9.1m

WORK

pNo	pName	budget
P1	Bridge	11.5m
P3	Tower	10.2m

PROJ

Fig. 3: Example Work and PROJ tables

Now (WORK - PROJ) = (eNo), please not that (WORK – PROJ) does not denote set difference operator here, rather it simply shows the attributes of WORK that are not in PROJ

$_{WORK-PROJ}(WORK)$

When we take the projection of WORK on eNo, we get the table shown here. Now we have to combine all rows of PROJ with each row of this table and see if all the rows obtained by this combination are there in WORK or not. For example, from the output of the projection operation shown here, We combine the first row, that is, E1 with all rows of PROJ,

eNo
E1
E2
E1
E4

we get a table with two rows of PROJ plus the attribute eNo having the value E1 in both rows as shown in the table below:

eNo	**pNo**	**pName**	**budget**
E1	P1	Bridge	11.5M
E1	P3	Tower	10.2M

Both of these rows are there in WORK table, so E1 will be included in the output of the division operator. Next we consider E2 row from the projection table the output will be

eNo	**pNo**	**pName**	**budget**
E2	P1	Bridge	11.5M
E2	P3	Tower	10.2M

First of these two rows is included in the WORK table but not the second one, so E2 will not be included in the final output. Same is the case with E4. So the final output will contain just E1.

That was all about relational algebra, now we move to next relational manipulation language and that is the relational calculus

Relational Calculus

Relational Calculus is a nonprocedural formal relational data manipulation language in which the user simply specifies what data should be retrieved, but not how to retrieve it. It is an alternative standard for relational data manipulation languages. The relational calculus is not related to the familiar differential and integral calculus in mathematics, but takes its name from a branch of symbolic logic called the predicate calculus.

Based on first order predicate logic.

Expressed as { t | f (t) } well formed formula, that is, find the set of all tuples t such that f (t) is time, while R implies the predicate conditions.

It has two following two forms: -

Tuple Oriented Relational Calculus

Domain Oriented Relational Calculus

Atomic Formula: In relational calculus,

Tuple – variable membership expression is specified as R (t) or R.t. and Condition :

- S {A}θ t {B}
- S {A} θ C

Tuple Oriented Relational Calculus: In tuple oriented relational calculus we are interested primarily in finding relation tuples for which a predicate is true. To do so we need tuple variables. A tuple variable is a variable that takes on only the tuples of some relation or relations as its range of values. It actually corresponds to a mathematical domain. We specify the range of a tuple variable by a statement such as: -

RANGE OF S IS STUDENT

Here, S is the tuple variable and STUDENT is the range, so that S always represents a tuple of STUDENT. It is expressed as

{S | P (S)}

We will read it as find the set of all tuples S such that P(S) is true, where P implies the predicate condition Now suppose range of R is STUDENT

{R | R.Credits > 50}

We will say like find the stuId, stuName, majors etc of all students having more than 50 credits. Application programs for a database application are developed using both the Database language (the SQL in case of relational DBMSs) and any programming language. Both are required since SQL is used for interacting with the DBMS and the programming language is used for designing the user interface and other data manipulation activities required in the application. For example, we can access basic salary, different allowances and deductions of employees from table using SQL and then using the features of the programming language we can compute the gross income, income tax etc. and if required store the same in the database back again using SQL. For that we have a tight coupling of database and programming languages as in Vidual Basic or loose coupling where programming languages are extended with special concepts e.g a version of Pascal is present which is extended with the functionality of accessing the database.

We have covered the database component of the basic topics, now we move to the next component and that is the computer networks.

Computer network

A computer network is a system for communication between two are more computer, where computers are Interconnected and Autonomous.

Computers are called nodes, sites, host.

Other equipment at nodes. i.e. they may or may not be computers.

Equipment connected via links and channels .

Data Communication

Communication links connect equipment and carry data in form of digital or analog signal.

Each channel has a certain capacity measured in amount of data that it can carry within a unit time; this capacity is normally termed as bandwidth. Communication channels can be analogue, digital (ISDN) or broadband.

Modulation and Demodulation: when data is to be transmitted over an analogue carrier it needs to be modulated, that is, the digital signal is encoded into analogue carrier signal by changing the basic characteristics of the signal. On receiving end, it has to be demodulated.

Multiplexing, using the same link to transmit different signals simultaneously. Two types of multiplexing: frequency division multiplexing (FDM) and time-division multiplexing (TDM).

Mode of operation could be simplex (transferring data in one direction only), half-duplex (both directions, but one at a time) and full-duplex (both directions simultaneously).

Factors that matters

Bandwidth

Mode of operation.
Software employed

Types of Network: we can classify networks on different bases, following are the criteria according to which we can classify the networks:

Interconnection structure ------- topology.
Transmission mode.

Geographical distribution.

Based on the interconnection structure there are following categories of the networks

1) Star
2) Bus
3) Ring
4) Mesh

Star topology: Many home networks use the star topology. A star network features a central connection point called a "hub" that may be an actual hub or a *switch*. Devices typically connect to the hub with Unshielded Twisted Pair (UTP) Ethernet. Compared to the bus topology, a star network generally requires more cable, but a failure in any star network cable will only take down one computer's network access and not the entire LAN. (If the hub fails, however, the entire network also fails.)

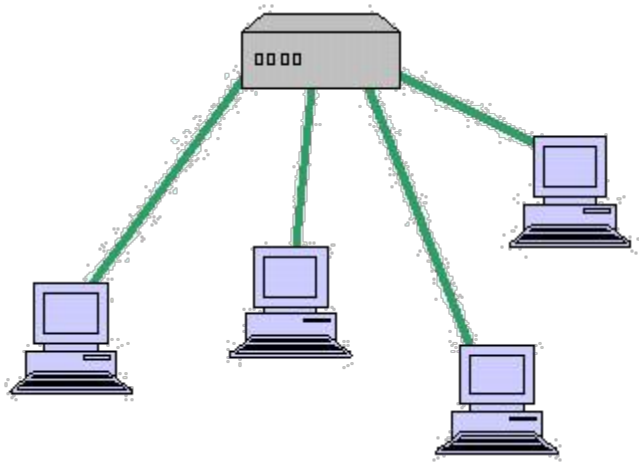


Fig. 4: Star network topology

Ring network: In a ring network, every device has exactly two neighbors for communication purposes. All messages travel through a ring in the same direction (effectively either "clockwise" or "counterclockwise"). A failure in any cable or device breaks the loop and can take down the entire network.

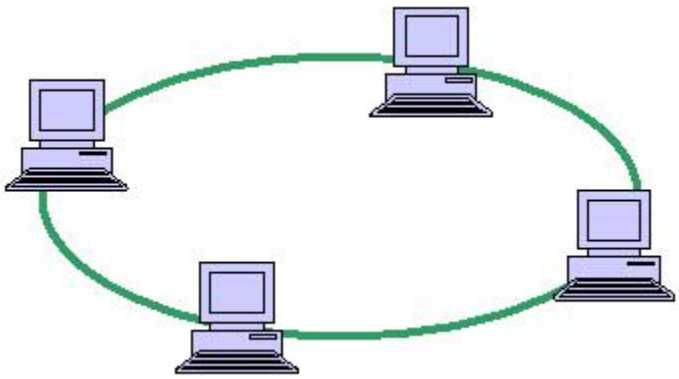


Fig. 5: Ring network

To implement a ring network, one typically uses FDDI, SONET, or Token Ring technology. Rings are found in some office buildings or school campuses.

Bus network: Bus networks use a common *backbone* to connect all devices. A single cable, the backbone functions as a shared communication medium, that devices attach or *tap* into with an *interface connector*. A device wanting to communicate with another device on the network sends a *broadcast* message onto the wire that all other devices see, but only the intended recipient actually accepts and processes the message.

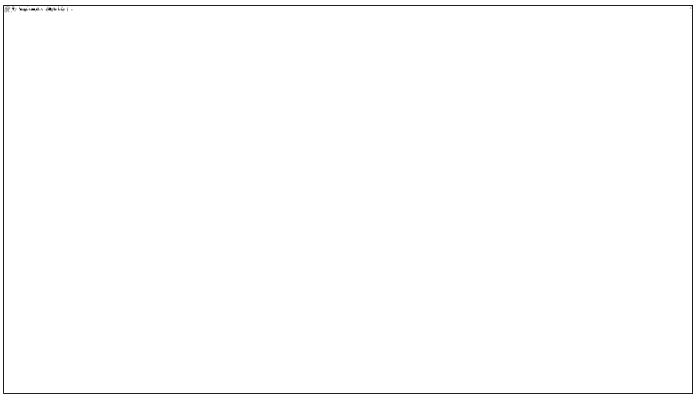


Fig. 6: A bus network

Meshed network: Mesh topologies involve the concept of *routes*. Unlike each of the previous topologies, messages sent on a mesh network can take any of several possible paths from source to destination. (Recall that in a ring, although two cable paths exist, messages can only travel in one direction.) Some WANs, like the Internet, employ mesh routing.

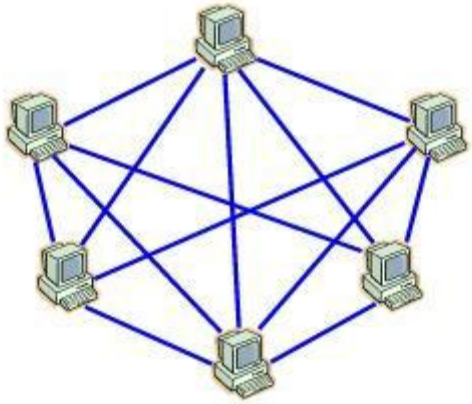


Fig. 7: A mesh network

The classification based on the **transmission mode** is

Point to point (unicast)
Broadcast (multi-point)

Point to point

One or more links between sender and receiver
Link may be direct or through switching.

Broadcast

Common channel utilized by all nodes.
Message received by all ownership checked.
Generally radio or satellite based or microwave.

Summary

In this lecture we concluded the discussion on relational manipulation languages. We studied the semi-join, and then we saw the division operator. After that we briefly discussed the relational calculus whose major types include tuple-oriented and domain oriented calculus. Then we

moved to the second component of the basic background topics. We defined networks and data communication. Then we saw different factors that matter in communication, and finally we discussed the types of network topologies.

Course Title: **Distributed Database Management Systems**
Course Code: **CS712**
Instructor: **Dr. Nayyer Masood (nayyerm@yahoo.com)**
Lecture No: **09**

In previous lecture:

- **Semi-join, division**
- **Networks and data communication**
- **Network topologies**

In this lecture:

- **Network Types**

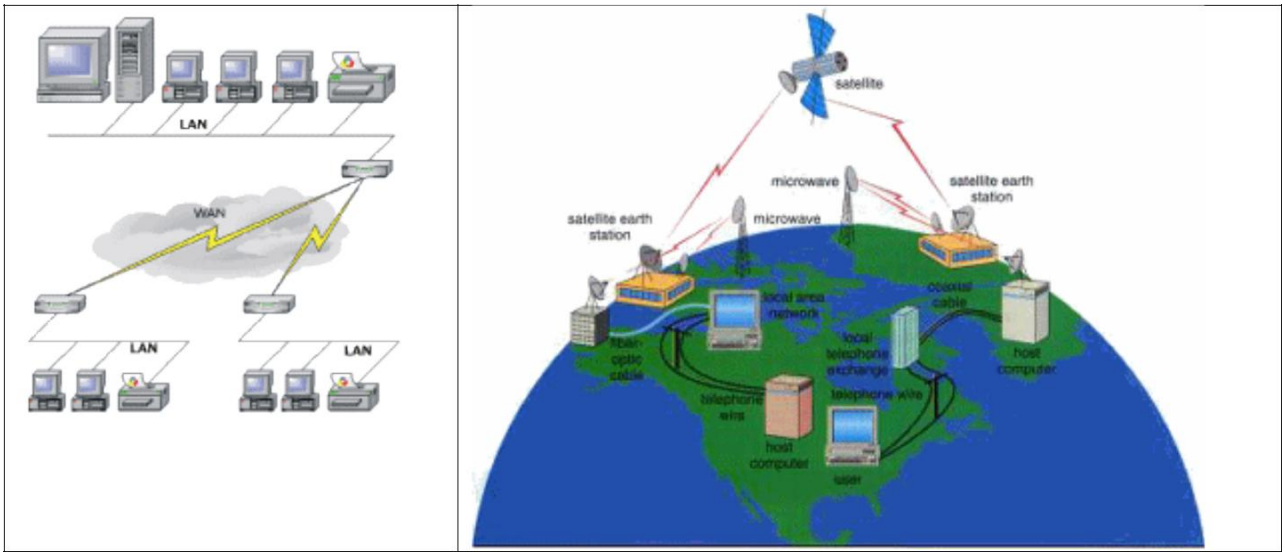
Scale: Geographical distribution of Network

Three major types of Net Work geographically.

1. Local Area Network
2. Metropolitan Area Network
3. Wide Area Network

WAN (Wide Area Network):

- Used inter city country or even continental
- gives low bandwidth high latency. (delay) more delay comparative to other Networks.
- Can be Broadcast and Point to Point.



- **Circuit Switching** – establish of link between two nodes.
- **Packet Switching**

message is divided into small Packets

some other information is added in Packet other than original msg. -
There are different paths available between sender and receiver.

Multiple Routes are available.

Packets can choose different routes but they ultimately reach at same destination each packet has a unique id.

This is not necessary that at the receiving end the sequencing is same which was at sending end

Note:
Received packets should be re-arranged to their original sequencing.

Circuit Switching:

- Dedicated link between the sender and the receiver
- Link is maintained during conversation.

Packet Switching:

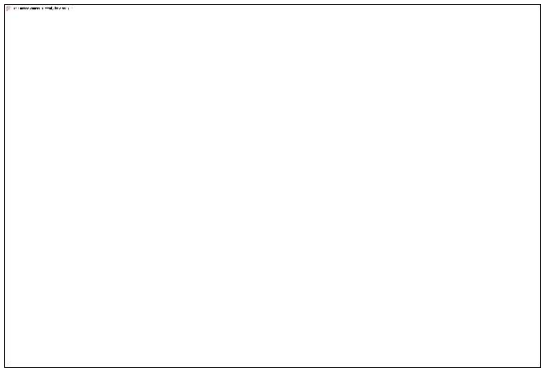
- Message is divided into packets
- May take different routes
- Need to be sorted.

Advantages of Ps:

- Better link Utilization
- Bursty nature of communication – break involve after some time the data send.
- Parallel transmission.

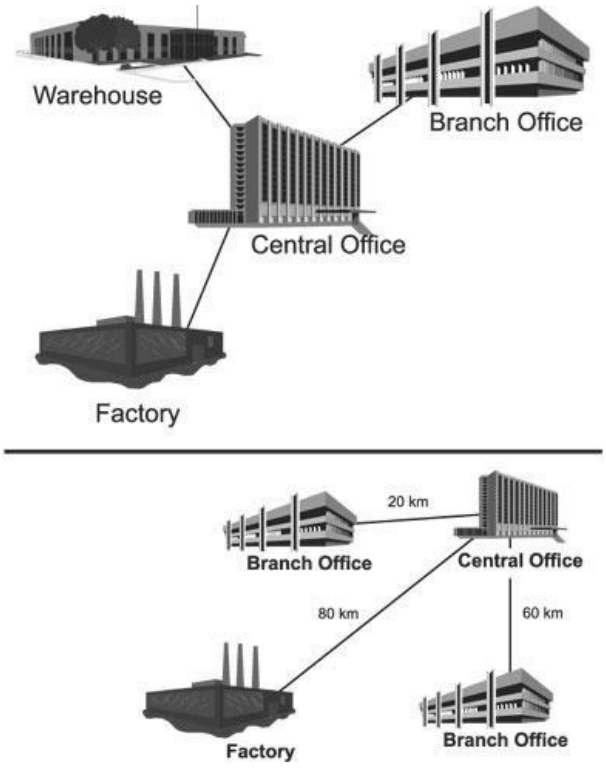
LAN (Local Area Network):

- Small geographical area
- High Band width
- Low latency



MAN (Metropolitan Area Network):

- Between LAN and WAN
- Cover city or Portion
- Larger LAN.



Protocol Standards:

- Involves software component for the communication to be held properly

Definition:

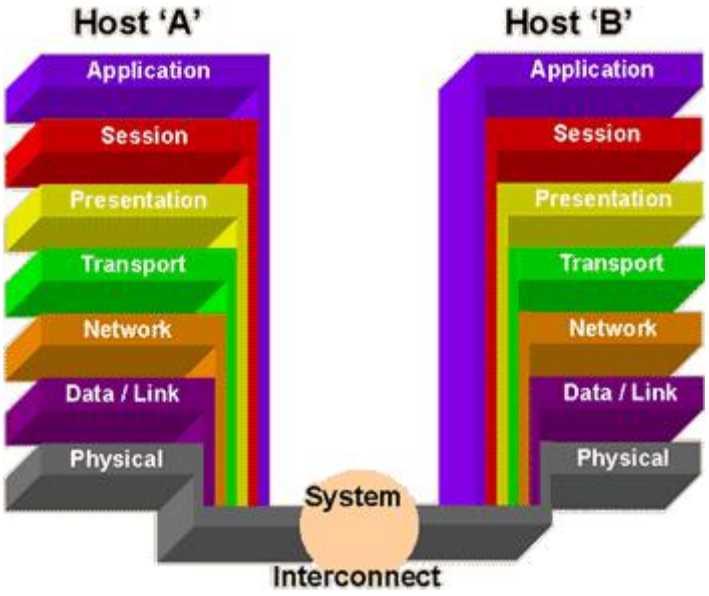
Set of rules and formats for exchanging data, arranged into layers called protocol suite /stack

- WAN involves more protocol
- Needs protocols

- ISO / OSI – give Standard to use
  - Open system Interface.
  - International Standard Organization.

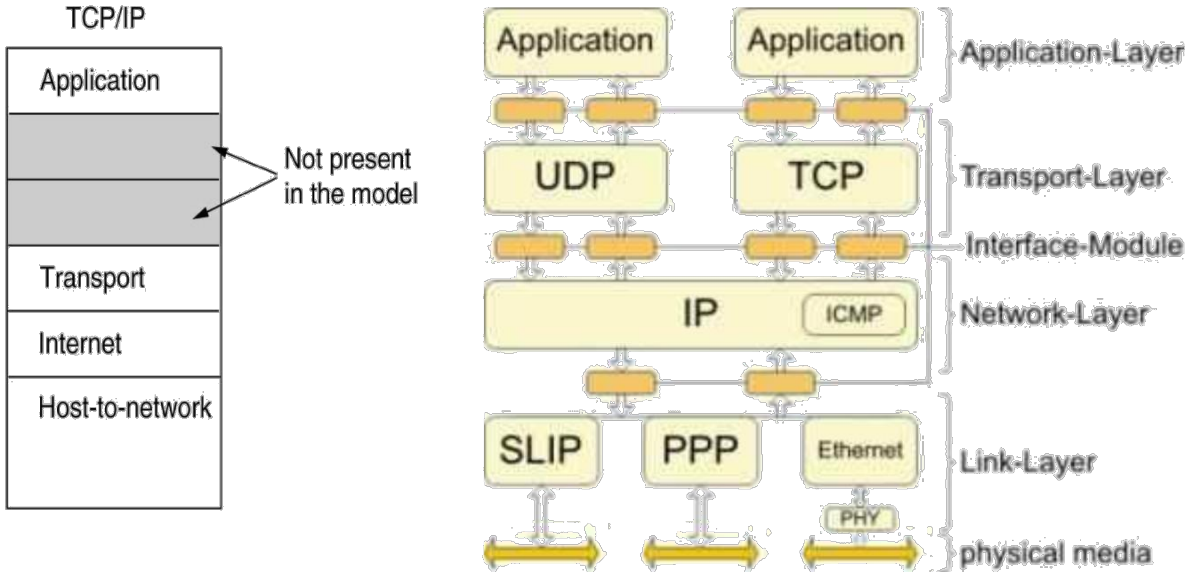
I SO / OSI Architecture:

- Network built in seven layers data pass through all layers
- Interfaces for passing information between layer.
- Protocols between corresponding layers at different sites.
- Lower three layers for Communication.



TCP / IP Architecture:

- Another Popular Architecture.
- Five Layers.
- IEEE committee to take decision for protocol and make standard.



Architecture of DDMS:

Different Architectures are available but the appropriate one should be selected

Architecture of a system defines it structure.

- What core components are present in the system.
- Relation of Components.

Three major (idealized) architecture for DDB MS

- Peer to Peer
- Client / Server
- Multi databases.

DDMS Standardization:

A conceptual framework to divide standardization work into manageable pieces.

Approaches:

- **component – based** – define component of system and their features. Difficult to determine the functionality of system
- **Function – based** – user interaction.

- **Data – based. –**
  These three approaches can apply so we can say this as orthogonal

Practically :

- Every aspect has to be considered
- These classification schemes are orthogonal.

Course Title: **Distributed Database Management Systems**
Course Code: **CS712**
Instructor: **Dr. Nayyer Masood (nayyerm@yahoo.com)**
Lecture No: **10**

In previous lecture:

- Network Types

In this lecture:

- DDBMS architectures
- Different Alternatives

In this lecture we are going to study different DDBMS architectures. The study of architecture is helpful in understanding the working, different components and how they interact with each other. There are different criteria to discuss the DDBMS architectures and we find different classifications of DDBMS architectures in literature. One particular approach has been adopted in our text book, yet we can also find some architectures in [Sheth and Larson 90] research paper on FDBS architecture. Some of the architectures proposed in this paper are also included in this lecture. We will begin our discussion from the architecture of normal DBMS.

The DBMS Architecture

In this section, we are going to discuss the structure of non-distributed DBMS architecture, which can also be called a centralized DBMS and which is also commonly used in the client-server DBMS applications. This is also called the ANSI/ SPARC Architecture, the components of architecture are shown in the figure below:

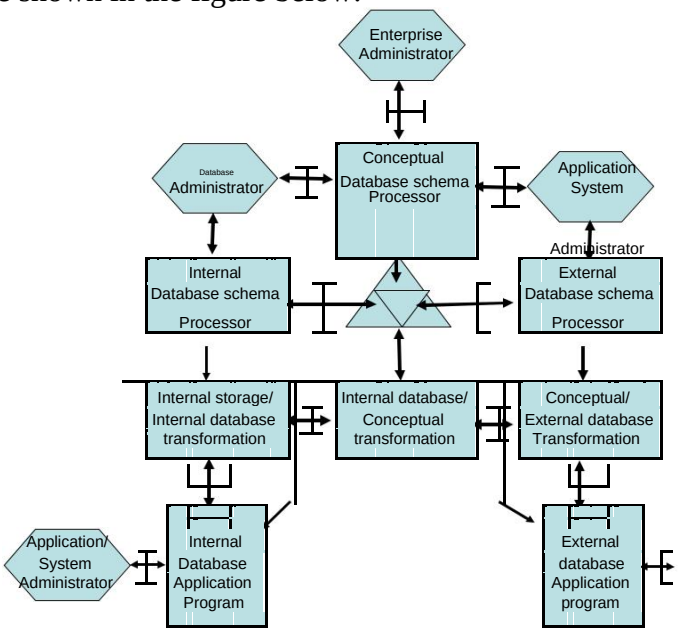


Figure 1: The components of ANSI/SPARC DBMS architecture

Square boxes in the architecture represent processing function, I represents the interfaces between different components. In the centre (triangular form) is the data dictionary (DD) that contains schema mapping data statistics and access control information. The interfacing between different components is possible only through DD. Hexagons represent different Administrators involved in a DBMS environment. Three types of administrators are shown here, The Database Administrator, Application Administrator and Enterprise Administrator. Their respective responsibilities are shown below:

DB administrator → responsible for internal schema

Application Administrator → responsible for external schema.

Enterprise Administrator → responsible for conceptual schema.

All these administrative roles may be assigned to a single person only.

Architectural models for DDBMS

The DDBMS architecture can be discussed from three different dimensions mentioned below: Autonomy

Distribution

Heterogeneity

Autonomy

Degree to which member databases can operate independently

Note:

Member databases are those database that constitute the distributed database system.

Autonomy is how much independently the member databases perform their functionality or make their decision.

Design Autonomy

Freedom for designer to make decision related to design, e.g., which data model he wants to design the database in, which DBMS he thinks is suitable for the organization. It may also include the decisions like machine to buy or operating system to be based on.

Communication Autonomy

What part of data we want to share. It depends on the nature of bond/coupling between member databases. If they are independent or are different organization, then obviously they own their data and can decide independently. However, if there is some strong bond between the member databases or they belong to the same organization then the member DBA does not have much autonomy and it is decided centrally somewhere what to be contributed from a member database and what not.

Execution Autonomy

Component databases the authority for when to share the data when to shut down etc. Again in case of same organization, the rules are formulated at the head office level and all member databases have to follow that, otherwise they are autonomous to make this decision.

Classification

One possible classification of DDBMSs follows: Tight integration

All users get a single common view with a feeling that there are no component databases.

Semi autonomous

Component database are present and they relate their data and at certain time they share their data to form a common view.

Total Isolation

All the systems are stand alone.

Distribution

Second of the three dimensions is distribution, that is, if data is distributed physically on multiple sites/machines. This is an essential component of a DDBMS, that is, a system cannot be called truly a DDBMS if the data is not distributed physically. The distribution could of data or of control; here it refers to Distribution of data, the physical distribution. There are two major architectures proposed in the text book:

The client / server: Client and server, both are machines not processes

Peer to peer: Servers performing independently and cooperating with each other. Both these architectures are discussed in detail in the next lecture.

Heterogeneity

Simply means variation / difference.
May be in different form
Different component method must be compatible.

Semantic Heterogeneity

Different component databases representing same real-world concept differently. It is most difficult to handle.

Architectural Alternative

We have studied three dimensions of the DDBMS architectures. These dimensions may exist at different levels in different architectures as given below:

Autonomy (0,1,2)
Distribution (0,1,2)
Heterogeneity (0,1)

These numbers can be interpreted as follows:

Autonomy 0 1 2

Tight hydrogenation
semi autonomous
total isolation

Distribution

0 → No distribution
1 → client/server
2 → Peer to Peer

Heterogeneity

0 → homogeneous
1 → Heterogeneous

Different combinations of these dimensions give rise to different architectures of DBMS, some examples are given belw:

Example
(A1,D2,H1)

A1(semi autonomous)
D2 (Peer to Peer)
H1 (Heterogeneity)
All possible combinations are shown in the diagram below:

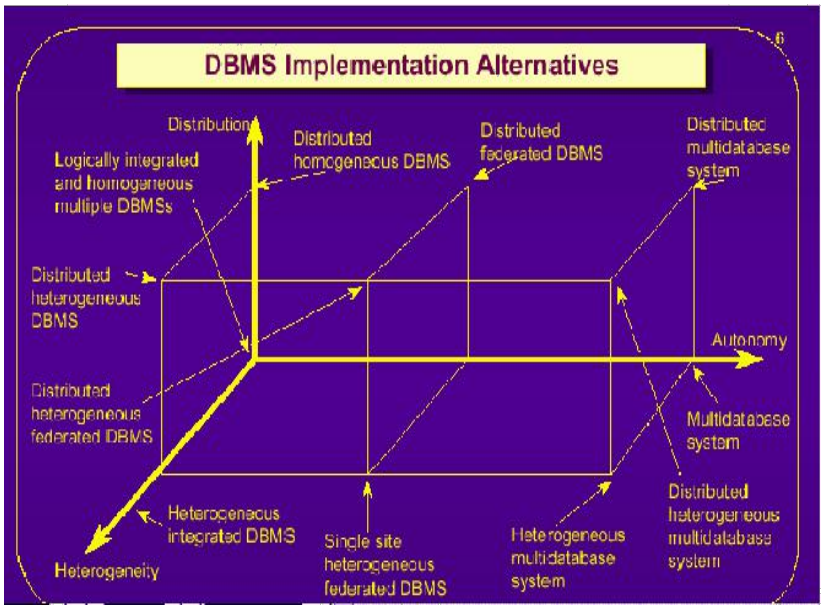


Figure 2: Combinations of different dimensions of DDBMS architecture

Some of these architectures are hypothetical, that is, they do not practically exist, moreover some are no truly distributed as they miss the distribution dimension which is an essential ingredient.

Summary

In this lecture we have started discussion on architecture of the DDBMS which is helpful in understanding the working of the system. The architecture of a non-distributed DBMS has been discussed first which is also called ANSI/SPARC architecture. After that different dimensions for the study of DDBMS architectures are discussed whose combination forms different architectures.

Course Title: **Distributed Database Management Systems**
Course Code: **CS712**
Instructor: **Dr. Nayyer Masood (nayyerm@yahoo.com)**
Lecture No: **11**

In previous lecture:

- DDBS Architecture

In this lecture:

- DDBS Architecture continued

Distributed DBMS Architecture:
The three distributed DBMS architecture are discussed below:

1) Client Server Architectures:

The term is used here in different meanings: generally Client and Server refer to processes may be running at same machine in the context of DDBS both client and server are machines not process.

Server:

Server does most f the data management work. This means that all of the query processing and optimization, transaction management and storage management is done at the server.

Client:

Application and user interface has a DBMS client module that is responsible for managing the data that is cached to the client and managing the transaction locks that may have been cached as well. Consistency checking of user queries is also applied at the client side.

Working

The client passes SQL queries to the server without trying to understand or optimize them. The server does most of the work and returns the result relation to the client.

Types

There are number of different types of client/server architecture.

1) Multiple client single server
There is only one server which is accessed by multiple clients.

2) Multiple client multiple server

Two alternative management strategies are possible: wither each client manages its own connection to the appropriate server or each client knows of only its home server which then communicates with other servers as required.

2) Peer to Peer Distributed System:

Physical data organization on each machine may be different. This means that there needs to be an individual internal schema definition at each site which we call the local internal schema (LIS). The enterprise view of the data is described by the global conceptual schema (GCS), which is global because it describes the logical structure of the data at all the sites.

In distributed database data is fragmented and replicated. To handle this phenomenon of fragmentation and replication, the logical organization of data at each site needs to be described. There needs to be a third layer in the architecture, the local conceptual schema (LCS). The global conceptual schema is the union of the local conceptual schemas. User applications and user access to the database is supported by external schemas (ESs).

This architectural model is shown in figure 1. Data independence is supported. Localization and replication transparencies are supported by the definition of local and global conceptual schemas and the mapping in between. Network transparency is supported by the definition of global conceptual schema.

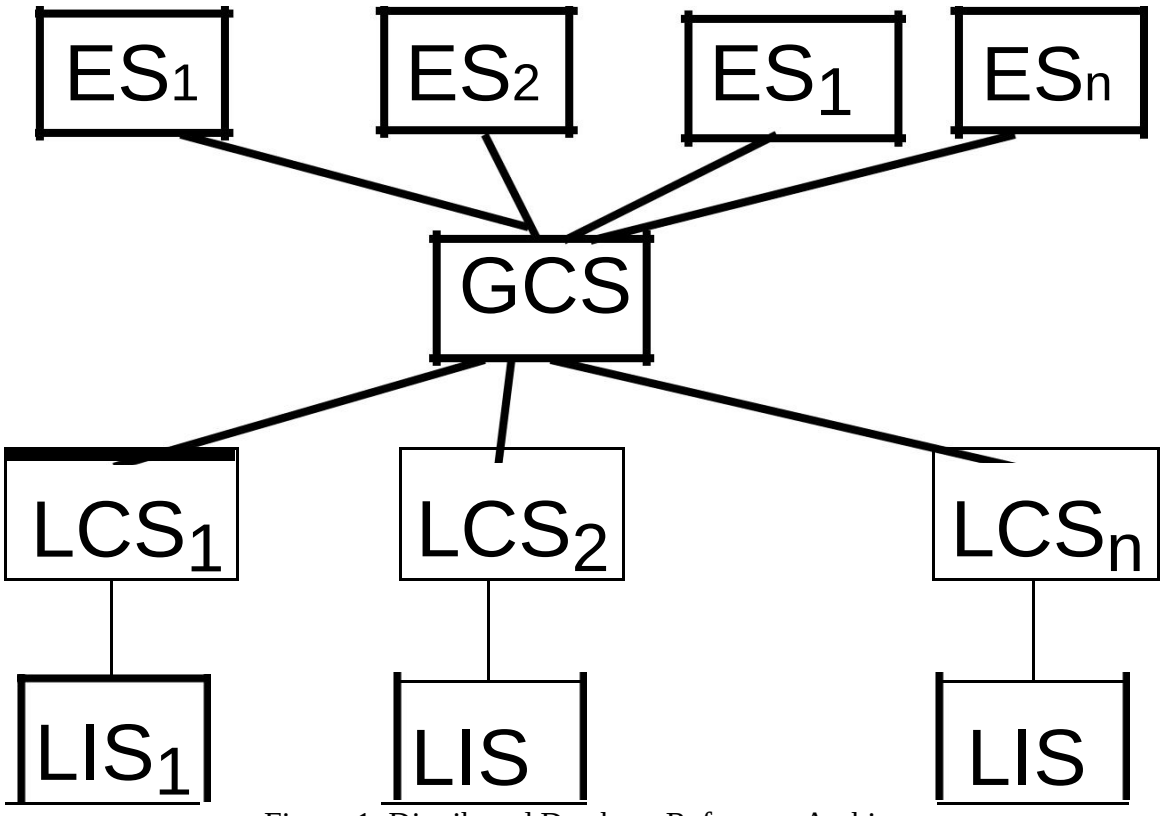


Figure 1: Distributed Database Reference Architecture

Components

The components of distributed DBMS are shown in figure 2. One component handles the interaction with users and other deals with the storage. The first components is called user processor, consists of four elements.

1. The *user interface handler* is responsible for interpreting user commands and formatting the result data as it is sent to the user.
2. The *semantic data controller* uses the integrity constraints and authorizations that are defined as part of the global conceptual schema to check if the user query can be processed.
3. The *global query optimizer and decomposer* determines an execution strategy to minimize a cost function and translates the global queries into local ones using global and local conceptual schema as well as global directory. The global query optimizer is responsible for generating the best strategy to execute distributed join operations.
4. The *distributed execution monitor* coordinates the distributed execution of the user request. The execution monitor is also called the distributed transaction manager.

The second major component of a distributed DBMS is the data processor and consists of three elements.

1. The *local query optimizer*, which actually acts as the access path selector, is responsible for choosing the best path to access any data item.
2. The *local recovery manager* is responsible for making sure that the local database remains consistent even when failures occur.
3. The *run-time support processor* physically accesses the database accordin to the physical commands in the schedule generated by the query optimizer. The run-time support processor is the interface to the operating system and contains the database buffer manager which is responsible for main memoy buffers and data accesses.

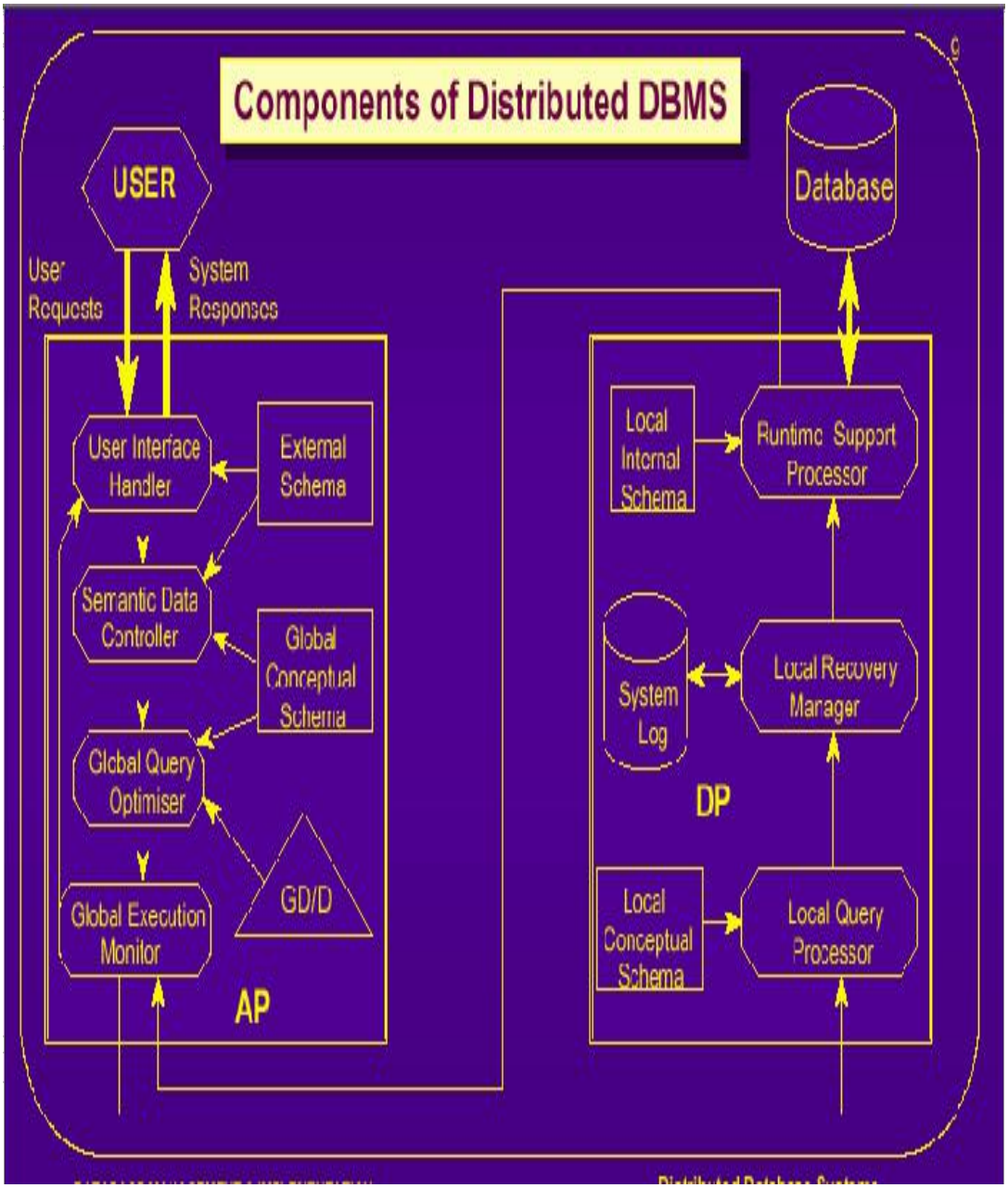


Figure 2: Components of a Distributed DBMS

MDBS Architecture:

The differences in the level of autonomy between the distributed multi-DBMSs and distributed DBMSs are also reflected in their architecture models. The fundamental difference relates to the definition of global conceptual schema. In case of logically integrated distributed DBMSs, the global conceptual schema defines the conceptual view of the entire database, while in case of distributed multi DBMSs; it represents only the collection of some of the local databases that each local DBMS wants to share. Thus the definition of global database is different in MDBSs than in distributed DBMSs.

Models Using a Global Conceptual Schema

Designing the global conceptual schema in multi-database systems involve the integration of either the local conceptual schemas or the local external schemas. A major difference between the deign of the GCS in the multi-DBMSs and in logically integrated distributed DBMSs is that in the former the mapping is from local conceptual schemas to a global schema.

Once the GCS has been designed, views over the global schema can be defined for users who require global access. It is not necessary for the GES and GCS to be defined using the same data model and language, whether they do or not determines whether the system is homogenous and heterogeneous.

If heterogeneity exists in the system, then two implementation alternatives exist: unilingual and multilingual. A unilingual multi-DBMS requires the users to utilize possibly different data models and languages when both a local database and the global database are accessed. One application may have a local external schema (LES) defined on local conceptual schema as well as global external schema (GES) defined on the global conceptual schema.

A multilingual architecture, where the basic philosophy is to permit each user to access the global database by means of an external schema.

Models without a Global Conceptual Schema

In this architecture there are two layers: the local system layer and the multi database layer on the top of it. The local system layer consist of a number of DBMSs, which present to the multi database layer the part of their local database they are willing to share with users of other databases. If heterogeneity is involved, each of these schemas, LCSi may use a different data model.

External views are constructed where each view may be defined on one local conceptual schema or on a multiple conceptual schemas. Thus the responsibility of providing access to multiple databases is delegated to the mapping between the external schemas and the local conceptual schemas.

The component based architectural model of a multi-DBMS is significantly different from a distributed DBMS.

Summary:

We have discussed the three distributed DBMS architecture. These architectures are Client and Server architecture, peer to peer architecture and multi database system architecture.

Course Code: **CS712**
Instructor: **Dr. Nayyer Masood (nayyerm@yahoo.com)**
Lecture No: **12**

In previous lecture:

- DDBMS Architectures
    0. Client/Server Systems
    1. Peer to peer DDBS
    2. Multi-database Systems

In this lecture:

- DDBS Architectures
- DDB Design

Global Data Dictionary:

A directory is a database that contains data about data (meta-data). It is called global directory in case of a DDBS. It includes information about the location of fragments as well as makeup of the fragments. There are some global directory issues discussed.

Number of directories:

A directory may be global to the entire database or local to each site. There might be a single directory containing information about all the data in the database or a number of directories, each containing the information stored at one site. We might either build hierarchies of directories to facilitate searches, or implement a distributed search strategy that involves considerable communication among sites holding the directories.

Location:

The directory may be maintained centrally at one site or in a distributed fashion by distributing it over a number of sites. Keeping the directory at one site might increase the load at that site, thereby causing a bottleneck as well as increasing message traffic around that site. Distributing it

over a number of sites, on the other hand increases the complexity of managing directories. In the case of multi-DBMSs, the choice is dependent on whether or not the system is distributed. If it is the directory is always distributed; otherwise of course, it is maintained centrally.

Replication:

There may be a single copy of the directory or multiple copies. Multiple copies would provide more reliability, since the probability of reaching one copy of the directory would be higher. Furthermore the delays in accessing the directory would be lower, due to less contention and the relative proximity of the directory copies. On the other hand, keeping the directory up to date would be considerably more difficult, since multiple copies would need to be updated. Therefore, the choice should depend on the environment in which the system operates and should be made by balancing such factors as the response time requirements, the size of directory, the machine capacities at the sites, the reliability requirements and the volatility of the directory. These choices are valid only in the case of a distributed DBMS. A non-distributed multi-DBMS always maintains a single copy of the directory, while a distributed one typically maintains multiple copies, once a each site.
All three issues are orthogonal to each other

Distributed Database Design

In distributed DBMSs, the distribution of applications involves two things: the distribution of distributed DBMS software and the distribution of the application programs that run on it. DDBMS concerns two major steps,

- Schema Design
- Schema Distribution

Schema Distribution decides placement of data. This course focuses on distribution of data.

Design Strategies

Two major strategies that have been identified for designing distributed databases are the:

Top down approach
Bottom up approach

Top Down Design Process

A framework for this process is shown in figure 1. The activity begins with a requirements analysis that defines the environment of the system. The requirements document is input to two parallel activities: view design and conceptual design. The view design activity deals with defining the interfaces for end users. The conceptual design is the process by which enterprise is examined to determine entity types and relationships among these entities. One can divide this process into two related activity groups entity analysis and functional analysis.

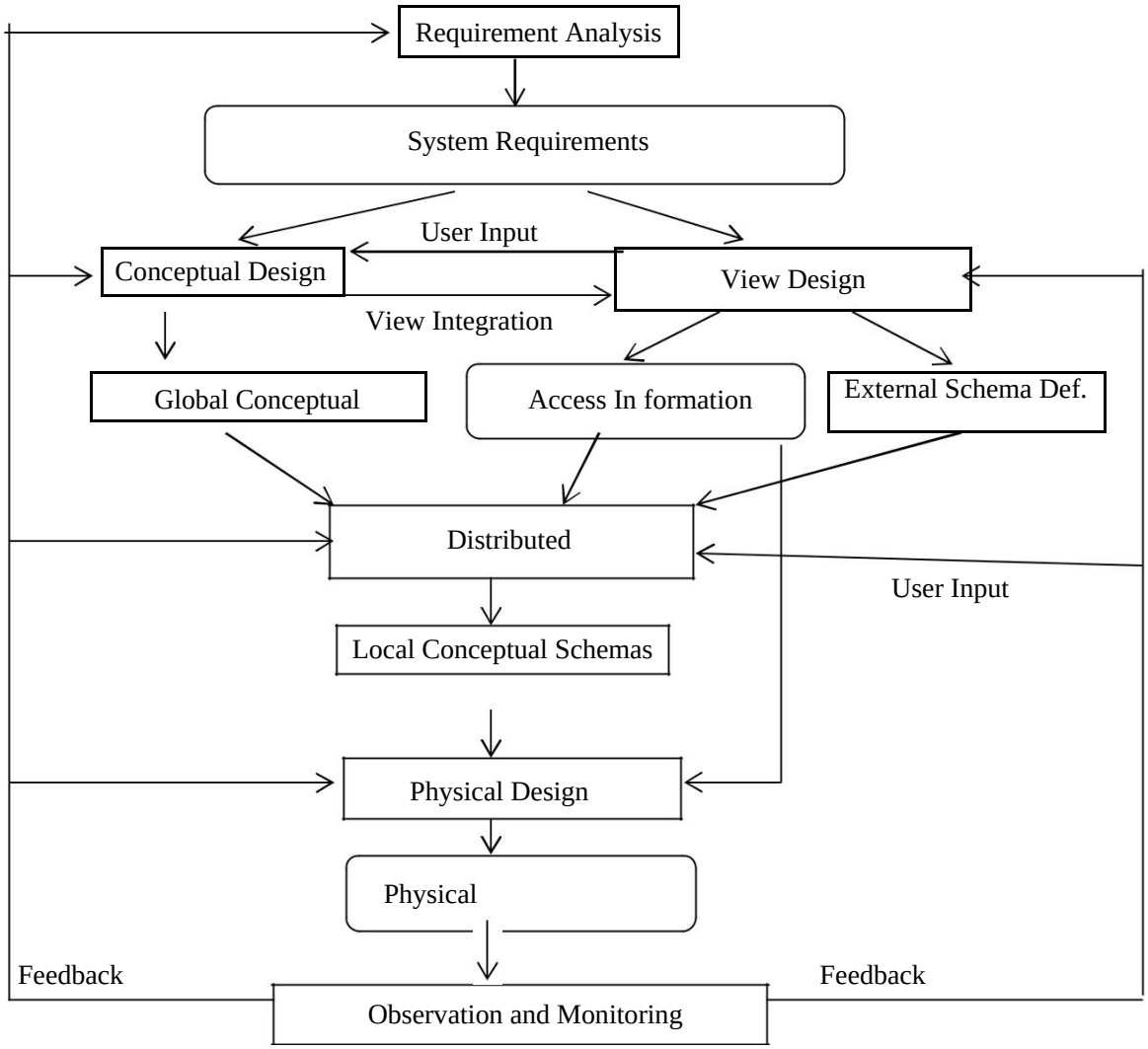


Figure 1: Top Down Design Process

There is a relationship between conceptual design and view design. The conceptual design can be interpreted as being an integration of user views. View integration should be used to ensure that entity and relationship requirements for all the views are covered in the conceptual schema. The global conceptual schema (GCS) and access pattern information collection as a result of view design are inputs to the distribution design step. Rather then distributing relations, it is common to divide them into sub relations, called fragments, which are then distributed. The distribution design activity consists of two steps:

Fragmentation
Allocation

The last step in the design process is physical design, which maps the local conceptual schemas to the physical storage devices available at the corresponding sites. The inputs to this process are the local conceptual schema and access pattern information about the fragmentation in these.

Bottom up approach

Top down approach is a suitable approach when a database system is being designed from scratch. A number of databases already exist, and the design task involves integrating them into

one database. The bottom up approach is suitable for this type of environment. The starting point of bottom up approach is the individual local conceptual schemas. The process consists of integrating local schemas into the global conceptual schema.

Distribution Design Issues:

The following set of interrelated questions covers the entire issue.

1. Why fragments
2. How should we
3. How much should be fragmented
4. Any way to test correctness
5. Allocation Strategy
6. Required Information

1. Reasons for fragmentation:

A Fragment is a subset of a relation that is stored on a different site from another subset of the same relation. There are some reasons why we do fragmentation:

- For large organizations, increased pressure of users
- Localization of Data and Local Autonomy
- Increased time/NW cost of centralized access
- Increased chances of failure
- Increased failure damage
- Distribution of data allows fast access of data due to localization
- Parallel execution of queries

The advantages of fragmentation are:

Increases the level of concurrency
System throughput

The disadvantages of fragmentation are:

Difficult to manage in case of non-exclusive Fragmentation (replication)
Maintenance of Integrity constraints

Summary

We have discussed the global data dictionary. There are some issues like number of directories, location and replication. We have also discussed distributed database design which consists of two approaches top down and bottom up approach.

Course Code: **CS712**
Instructor: **Dr. Nayyer Masood (nayyerm@yahoo.com)**
Lecture No: **13**

In previous lecture:

- DDBMS Architectures
- DDB Design

In this lecture:

- DDB Design
- Horizontal Fragmentation

Fragmentation

2. Fragmentation alternatives

Entire table is not a suitable unit, so dividing a table into smaller ones. There are two alternatives for dividing a table

1. Vertically
2. Horizontally

1. Vertical Fragmentation:

Different subsets of attributes are stored at different places,
like, Table EMP(eId, eName, eDept, eQual, eSal)
Interests of the local and head offices may result following vertical partitions of this table:

EMP1(eId, eName, eDept)
EMP2(eId, eQual, eSal)

2. Horizontal Fragmentation:

It is based on the localization of data rows of a table are split on multiple sites, like the data in CLIENT(cAC#, cName, cAdr, cBal)

Table is placed in different databases based on their location, like from Lahore, Pindi, Karachi, Peshawar, Quetta

3. Degree of Fragmentation:

The degree of fragment goes from one extreme not to fragment at all, to the other extreme, to fragment to the level of individual tuples (in case of horizontal fragmentation) or to the level of individual attributes (in case of vertical fragmentation).

4. Correctness Rules for Fragmentation:

The following three rules during fragmentation ensure that the database does not undego semantic change during fragmentation.
If a relation R is fragmented into R1, R2, …, Rn, then

- **Completeness**: each of the data item (a tuple or a attribute) that can be in R can also be in one or more Ri's.
  $\forall \quad x \in R, \exists \text{ Ri such that } x \in Ri$

- **Reconstruction**: it should be possible to define a relational operator such that the original relation can be reconstructed
  R = g(*R1, R2, …, Rn*)

- **Disjointness**: if data item x is in Rj, it is not in any other fragment
  $\forall \quad x \in R_i, \quad \exists \text{ Rj such that } x \in Rj, i \neq j$
  In case of vertical partitioning, disjoint ness is defined only on the nonprimary key attributes of a relation.

5. Allocation Strategy:

Assuming that the database is fragmented properly, one has to decide on the allocation of the fragments to various sites on the network. When data is allocated, it may be replicated or maintained as a single copy. The reasons for replication are reliability and efficiency of read-only queries. If there are multiple copies of a data item, there is a good chance that some copy of the data will accessible somewhere even when system failures occur. The execution of update queries cause trouble since the system has to ensure that all the copies of the data are updated properly.

A non-replicated database (called a partitioned database) contains fragments that are allocated to sites, and there is only one copy of any fragment on the network. In case of replication, either the database exists in it's entirely at each site (fully replicated database) or fragments are distributed to the sites in such a way that copies of a fragment may reside in multiple sites (partially replicated database)

6. Information Requirements:

The information needed for distribution design can be divided into four categories: database information, application information, communication network information and computer system information.

Fragmentation
The two fragmentation strategies are:

Horizontal
Vertical

Horizontal Fragmentation
Horizontal fragmentation partitions a table along its tuples. Each fragment has a subset of the tuples of the relation. There are two versions of horizontal partitioning primary and derived.

Primary horizontal fragmentation of a relation is performed using predicates that are defined on that relation.

Derived horizontal fragmentation is partitioning of a relation that result from predicates being defined on another relation.

Information Requirements Horizontal Fragmentation Database Information
The database information contains the global conceptual schema. In this context, it is important to note how the database relations are connected to one another with joins. In relational model, these relationships are depicted as relations.

Application Information
The fundamental qualitative information consists of the predicates used in user queries. If it is not possible to analyze all of the user applications to determine these predicates, one should at least investigate the most important ones.

Given a relation R(A1, A2, …, An), where Ai is an attribute with domain Di, then a simple predicate pj has the form
pj: Ai θ Value,
Where θ ∈ {=, <, ≠, ≤, >, ≥} and Value is chosen from the domain of Ai(Value ∈ Di).

Summary
We have discussed the distributed database design and the horizontal fragmentation. Horizontal fragmentation consists of primary and derived fragmentation.

Course Title: **Distributed Database Management Systems**
Course Code: **CS712**
Instructor: **Dr. Nayyer Masood (nayyerm@yahoo.com)**
Lecture No: **14**

In previous lecture:

- DDB Design
- Horizontal Fragmentation

In this lecture:

- Horizontal Fragmentation
    0. Primary Horizontal Fragmentation (PHF)

Horizontal Fragmentation

Simple predicates are quite elegant to deal with; user queries quite often include more complicated predicates, which are Boolean combinations of simple predicates. One combination that we are particularly interested is called a minterm predicate.

Minterm predicate is the conjunction of simple predicates. It is always possible to transform a Boolean expression into conjunction normal form the use of minterm predicates in the design algorithms does not cause any loss of generality.

Given a set of simple predicates

Pr = {pr1, pr2, …, pm}for a relation R, the set of minterm predicates M is defined as M = {m1, m2, ……, mz}

M = {mj | mj = ʎ p*k}, 1 ≤ k ≤ m, 1 ≤ j ≤ z pk∈ Pr

where p*k = pk or p*k = (pk)

So each simple predicate can occur in a minterm predicate wither in its natural form or its negated form. The reference to the negation of a predicate is meaningful for equality predicates of the form

Attribute = Value

For inequality predicates, the negation should be treated as the complement. For example, the negation of the simple predicate

Attribute <= Value

Is Attribute > Value

Example

Consider relation PAY(title, sal). The following are some of the possible simple predicates that can be defined on PAY.

p1: title = “El Eng” p2:
title = “Sys Ana” p3:
title = “Mech Eng” p4:
title = “Prog”
p5: SAL ≤ 20,000
p6: SAL > 20000

The following are some of the minterm predicates that can be defined based on these simple predicates.

m1: title = “El Eng” ʎ SAL ≤ 20,000
m2: title = “Sys Ana” ʎ SAL > 20000
m3: (title = “Sys Ana”) ʎ SAL > 20000 m4:
(title = “El Eng”) ʎ (SAL > 20000)

These are not all minterm predicates that can be defined, we are presenting only a representative sample. Some of these may be meaningful given the semantics of relation PAY. Here m3 can also be rewritten as

m3: title ≠ “Sys Ana” ʎ SAL > 20000

Quantitative Information

The quantitative information about user applications, we need to have two sets of data.

1. **Minterm Selectivity:**

sel(mi): number of tuples of the relation that would be accessed by a user query specified according to given minterm predicate. For example, the selectivity of m1 of previous example is 0 since there are no tuples in PAY that satisfy the minterm predicate. We denote the selectivity of minterm mi as

sel(mi)

2. Access Frequency:

Frequency with which user applications access data. For Q(q1, q2, …, qn} is a set of user queries, acc(qi) determines the access frequency of qi within a given period, like

acc(q2) = 24

Primary Horizontal Fragmentation

A PHF is defined by a selection operation on the owner relations of the database schema Given a relation R, its horizontal fragments are

$$Ri = \cdots Fi(R),\ 1 \leq i \leq w$$

Where *Fj* is a selection formula, which is (preferably) a minterm predicate. A horizontal fragment, *Ri* of relation *R* consists of all the tuples of *R* which satisfy a minterm predicate *mi.* Given a minterm of predicates *M*, there are as many horizontal fragments of relation R as there are minterm predicates. This set of horizontal fragments also referred to as minterm fragments.

PHF (Simple Example)

Consider a relation PAY

Title	Sal
Elect. Eng	40000
Sys Analyst	34000
Mech. Eng	27000
Programmer	24000

The two fragments are:

PAY1 = σ sal ≤ 30000 (PAY)

PAY2 = σ sal > 30000 (PAY)

PAY1

Title	Sal
Mech. Eng	27000
Programmer	24000

PAY2

Title	Sal
Elect. Eng	40000
Sys Analyst	34000

An important aspect of simple predicates is their completeness, another is their minimality. A set of simple predicates Pr is said to be complete if and only if there is an equal probability of access

by every application to any tuple belonging to any minterm fragment that is defined according to Pr.

Example
Consider a relation PROJ

PNo	pName	Budget	Loc
P1	Instrumentation	3.5M	Lhr
P2	Database	2.3M	Rwp
P3	CAD/CAM	1.9M	RWP
P4	Maintenance	1.6M	Pesh

Find the budgets of projects at each location. (1)
Find projects with budgets less than 2M. (2)

According to (1),
Pr = {LOC="Lahore", LOC="Rawalpindi", LOC="Peshawar"}

Which is not complete with respect to (2).

Modify Pr so Pr becomes

Pr = {LOC="Lahore", LOC="Rawalpindi", LOC="Peshawar",
BUDGET<=2000000, BUDGET >2000000}

Now Pr is complete and minimal.

Summary
We have discussed the minterm predicate with some example. And after that we have also discussed the primary horizontal fragmentation.

In previous lecture:

- Horizontal Fragmentation

0. Primary Horizontal Fragmentation (PHF)

In this lecture:

- Primary Horizontal Fragmentation (PHF)

0. Minimality of Pr
1. PHF Algorithm

- Derived Horizontal Fragmentation (DHF)

Primary Horizontal Fragmentation Minimality of Pr

A relevant predicate is the one if it influences how fragmentation is performed (fragments f into fi and fj) then there should be at least one application that accesses fi and fj differently. If all predicates in a set Pr are relevant then the set is minimal.

A formal definition of relevance can be given as follows. Let mi and mj be two minterm predicates that are identical in their definition, except that mi contains the simples predicate pi in its natural form while mj contains pi. Let fi and fj be two fragments defined according to mi and mj, resp. Then pi is relevant if and only if

$$\frac{acc(mi)}{card(fi)} \neq \frac{acc(mj)}{card(fj)}$$

Example:

The set Pr defined as

Pr = {LOC="Lahore", LOC="Rawalpindi", LOC="Peshawar",
BUDGET<=2000000, BUDGET >2000000}

is complete and minimal. If we were to add the predicate

PNAME = “Instrumentation”

To Pr, the resulting set would not be minimal since the new predicate is not relevant with respect to Pr. There is no application that would access the resulting fragments any differently.

We can now present an iterative algorithm that would generate a complete and minimal set of predicates Pr’ given a set of simple predicates Pr. This algorithm is called COM_MIN.

PHF-COM-MIN Algorithm

The algorithm begins by finding a predicate that is relevant and those partitions the input relation.

Input: Given a relation *R* and a set of simple predicates *Pr*.

Output: A complete and minimal set of simple predicates *Pr’* for *Pr.*

Rule 1: fundamental rule of completeness and minimality which states that a relation or fragment is partitioned into at least two parts which are accessed differently by at least one application.

1-Initialization: Find a pi ∈ Pr such that pi, partitions R according to Rule 1

Pr’ ← pi

Pr ← Pr – pi

2- Iteratively add predicates to *Pr'* until it is complete, find a *pj* ∈ *Pr* such that *pj* partitions R according to *Rule 1*

Set Pr' = Pr' U pj

Pr = Pr – pj

3- Elimination of some of the minterm fragments that may be meaningless. If *pk in Pr'* is non-relevant then

Pr' = Pr' – pk

Primary Horizontal Partitioning Algorithm

The algorithm for primary horizontal fragmentation is partitioning algorithm. The input to the algorithm is a relation *R* and a set of simple predicates *Pr*. The output is a set of minterm predicates *M* according to which relation *R* is to be fragmented. The steps of algorithm are:

- *P*r‘ ← COM_MIN *(*R,*P*r)
- determine the set *M* of minterm predicates
- determine the set *I* of implications among *pi Pr*
- eliminate the contradictory minterms from *M*

PHF-Summary

- First Step is to determine the complete and minimal set of simple predicates; determined and checked based on the nature of applications
- Secondly, to derive the set of minterm predicates that are later used to establish fragments and to determine the allocation of fragments
- Identify the implications and remove the conflicting minterm predicates

PHF Example

Let us consider relation PROJ. Assume that there are two applications. The first is issued at three sites and finds the names and budgets of projects given their location. The simple predicates that would be used are the following:

p1 : LOC = "Lahore”

p2 : LOC = “Rawalpindi"
p3 : LOC = "Peshawar“

The second application is issued at two sites and has to do with the management of the projects. Those projects that have a budget of less than 2000000 are managed at one site and those with larger budgets are managed at a second site. The simple predicates that should be used to fragment are:

p4 : BUDGET ≤ 2000000

p5 : BUDGET > 2000000

If the algorithm COM_MIN is followed the set Pr’ = {p1, p2, p3, p4, p5} is complete and minimal.

Based on Pr’, the following six minterm predicates that form M can be defined:
m1 : (LOC = "Lahore") ^ (BUDGET ≤ 2M)

m2 : (LOC = "Lahore") ^ (BUDGET > 2M)

m3 : (LOC = " Rawalpindi ") ^ (BUDGET ≤ 2M)

m4 : (LOC = "Rawalpindi") ^ (BUDGET > 2M)

m5 : (LOC = "Peshawar") ^ (BUDGET ≤ 2M)

m6 : (LOC = "Peshawar") ^ (BUDGET > 2M)

These are not the only minterm predicates that can be generated. It is for example possible to specify predicates of the form

p1 p2 p3 p4 p5 The obvious implications are

- p1p2p3
- p2p1p3
- p3p1p2
- p4p5
- p5p4
- p4 p5
- p5 p4

Eliminate these minterm predicates and we are left with m1 to m6. Implications should be de fined according to the semantics of database, not according to the current values. Some of the fragments defined according to M = {m1, …, m6} may be empty, but they are fragments.

The result of primary horizontal fragmentation of PROJ is to form fragments Fproj = {PROJ1, PROJ2, PROJ3, PROJ4, PROJ5, PROJ6} of relation PROJ according to minterm predicates M.

PROJ

PNo	pName	Budget	Loc
P1	Instrumentation	3.5M	Lhr
P2	Database	2.3M	Rwp
P3	CAD/CAM	1.9M	RWP
P4	Maintenance	1.6M	Pesh

PNo	pName	Budget	Loc
P1	Instrumentation	3.5M	Lhr
P2	Database	2.3M	Rwp
P3	CAD/CAM	1.9M	RWP
P4	Maintenance	1.6M	Pesh

Here we see that some fragments are empty.

Derived Horizontal Fragmentation (DHF)

It is defined on a member relation of a link according to a selection operation specified on its owner. Two important points are:

- Each link between owner and member is defined as an equi-join.
- Equi-join can be implemented by means of semi-joins

So we are interested in defining the partitions of member based on fragmentation of its owner, but want to see attributes only from member,. Accordingly, given a link L where owner(L) = S and member(L) = R, the derived horizontal fragments of R are defined as

$R_i = R \ltimes S_i \quad 1 \le i \le w$

where w is the maximum number of fragments that will be defined on R and Si = ···· Fi (S), where Fi is formula according to which the primary horizontal fragment Si is defined.

DHF Example

Consider a link L1 in figure 1, where owner (L1) = PAY and member (L1) = EMP.

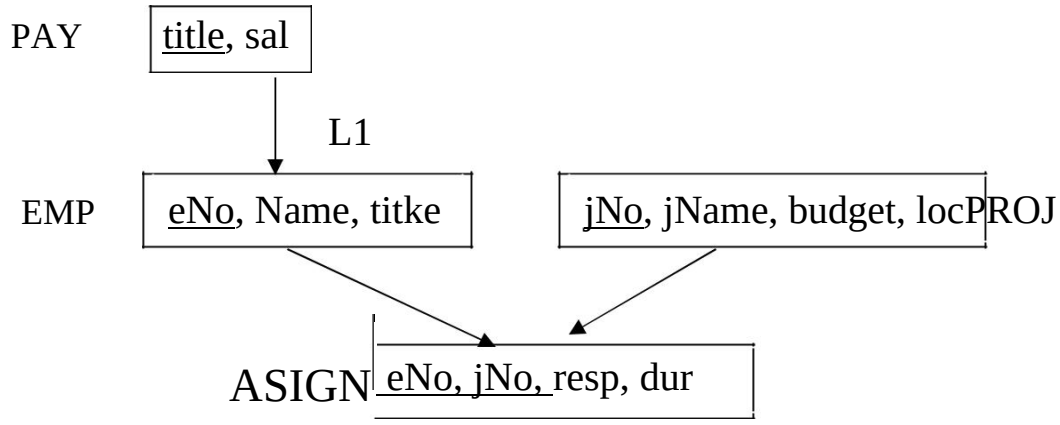


Figure 1: Expression of Relationships among Relations Using Links

Then we group engineers into two groups according to their salary: those making less than or equal to 30,000/- and those making more than 30,000.

Consider EMP and Pay relation

eNo	eName	title
E1	T Khan	Elec Eng
E2	W Shah	Sys Ana
E3	R Dar	Mech Eng
E4	K Butt	Programme
E5	F Sahbai	Sys Ana
E6	A Haq	Elec Eng
E7	S Farhana	Mech Eng
E8	M Daud	Sys Ana

Title	Sal
Elect. Eng	40000
Sys Analyst	34000
Mech. Eng	27000
Programmer	24000

The two fragments PAY1 and PAY2 of PAY and EMP1 and EMP2 of EMP are defined as follows:

- PAY1 = ···· SAL >30000 (PAY)
- PAY2 = ···· SAL ≤ 30000 (PAY)
- EMP1 = EMP ⋉ PAY1
- EMP2 = EMP ⋉ PAY2

The result of this fragmentation is depicted in figure 2.

EMP1

eNo	eName	title
E1	T Khan	Elec Eng
E2	W Shah	Sys Ana
E5	F Sahbai	Sys Ana
E6	A Haq	Elec Eng
E8	M Daud	Sys Ana

EMP2

E3	R Dar	Mech Eng
E4	K Butt	Programme
E7	S Farhana	Mech Eng

Figure2: Derived Horizontal Fragmentation of Relation EMP

Summary

We have discussed primary horizontal fragmentation which includes the minimality of Pr and PHF algorithms. We have also discussed the derived horizontal fragmentation with an example.

Course Title: **Distributed Database Management Systems**
Course Code: **CS712**
Instructor: **Dr. Nayyer Masood (nayyerm@yahoo.com)**
Lecture No: **16**

In previous lecture:

- Primary Horizontal Fragmentation (PHF)
    0. Minimality of Pr
    1. PHF Algorithm
- Derived Horizontal Fragmentation (DHF)

In this lecture:

- Conclude DHF
- Vertical Fragmentation

Input Required for DHF

To carry out a derived horizontal fragmentation, three inputs are needed:

The set of partitions for owner relation
Member relation
The set of semi-join predicates between owner and member

There is one potential complication that deserves some attention in a database schema; it is common that there are more than two links into a relation R (e.g. ASG has two incoming links). In this case there is more than one possible derived horizontal fragmentation of R. the decision as to which candidate fragmentation to choose is based on two criteria:

1- The fragmentation used in more applications
2- The fragmentation with better join characteristics

First one is straight forward if we take into consideration the frequency with which applications access some data. If possible, one should try to facilitate heavy users so that their total impact on system performance is minimized.

The second criterion is not the straightforward. Consider the fragmentation that we have discussed in previous lecture. The effect of this fragmentation is that the join of the EMP and PAY relations to answer the query is assisted

(1) by performing it on smaller relations (i.e fragments) and
(2) by potentially performing joins in a distributed fashion.

The first point is obvious. The fragments of EMP are smaller than EMP itself it will be faster to join any fragment of PAY with any fragment of EMP than to work with the relations themselves. The second point is more important and is at the heart of distributed databases. If besides executing a number of queries at different sites, we can execute one query in parallel, the response time or throughput of the system can be expected to improve. In case of joins, this is possible under certain circumstances. Consider the join graph between the fragments of EMP and PAY as shown in figure 1.

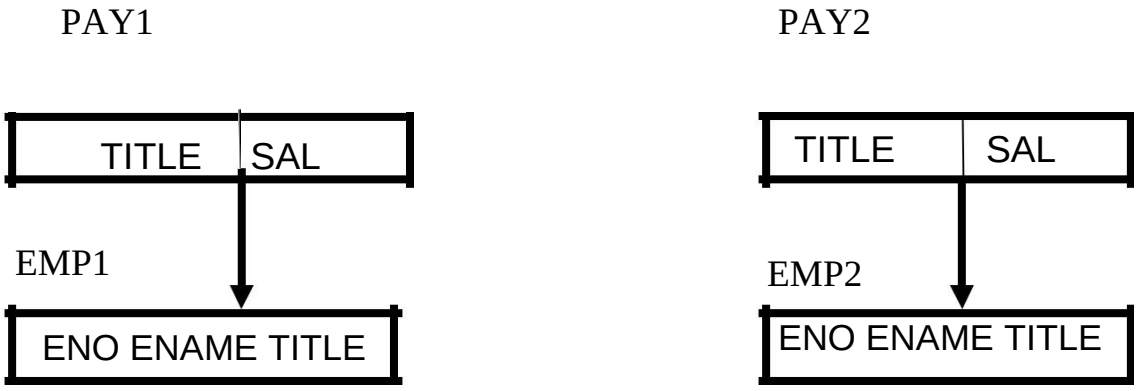


Figure1: Join Graph between Fragments

There is only one link coming in or going out of a fragment. Such a join graph is called a simple graph. The advantage of a design where the join relationship between fragments is simple is that the member and owner of a link can be allocated to one site and the joins between different pairs of fragments can proceed independently and in parallel. DHF demonstrates two things:

1- DHF may follow a chain where one relation is fragmented as a result of another one's design and it causes the fragmentation of another relation, like

PAY-EMP-ASIGN

2-Typically, there will be more than one candidate fragmentation for a relation (e.g. relation ASG). The final choice of the fragmentation schema may be a decision problem addressed during allocation.

Checking for Correctness
Completeness

The completeness of a PHF is based on the selection predicates used. The selection predicates are complete; the resulting fragmentation is guaranteed to be complete as well. Since the basis of fragmentation algorithm is a set of complete and minimal predicates, Pr', completeness is guaranteed as long as no mistakes are made in defining Pr'. The completeness of DHF is more difficult to define. The difficulty is due to the fact that the predicate determining the fragmentation involves two relations. let R be the member relation o a link whose owner is relation S, which is fragmented as Fs = {S1, S2, ….., Sn}. Let A be the join attribute between R and S then for each tuple t of R, there should be a tuple t' o S such that

t[A] = t'[A]

Reconstruction

Reconstruction of a global relation from its fragments is performed by union operator in both primary and derived horizontal fragmentation. Thus for a relation R with fragmentation FR = {R1, R2, …, Rn}

R = U Ri, áRi FR

Disjointness

It is easier to establish disjointness of fragmentation for primary than for derived horizontal fragmentation. Disjointness is guaranteed as long as the minterm predicates determining the fragmentation are mutually exclusive. In DHF, disjointness can be guaranteed if the join graph is simple. If it is not simple, it is necessary to investigate actual tuple values.

Vertical Fragmentation (VF)

A VF of a relation R produces fragments R1, R2, …. Rn, each of which contains a subset of R's attributes as well as the primary key of R. the objective of VF is to partition a relation into a set of smaller relations so that many of the use applications will run on only one fragment. An optimal fragmentation is one that produces a fragmentation scheme which minimizes the execution time of user applications that run on these fragments.

Vertical partitioning is inherently more complicated than horizontal partitioning. This is due to the total number of alternatives that are available. For example, in horizontal partitioning, if the total number of simple predicates in Pr is n, there are 2n possible minterm predicates that can be defined on it. VF is mainly based on heuristics.

Two alternative of VF

Two types of heuristics approaches exist for the VF of global relations:

1) Grouping: Starts by assigning each attribute to one fragment and at each step, joins some of the fragments until some criteria are satisfied.

2) Splitting: Starts with a relation and decides on beneficial partitioning based on the access behavior of applications to the attributes. Splitting generates nonoverlapping fragments whereas grouping typically results in overlapping fragments. Nonoverlapping refers only to nonprimary key attributes.

Let us discuss the issue of replication of the global relation's key in the fragments. This is a characteristic of vertical fragmentation that allows the reconstruction of the global relation. Splitting is considered only for those attributes that do not participate in the primary key.

There is a strong advantage to replicating the key attributes despite the obvious problems it causes. This advantage has to do with semantic integrity enforcement. Replication of the key attributes at each fragment reduces the chances of this occurring but does not eliminate it completely, since such communication may be necessary due to integrity constraints that do not involve the primary key as well as due to concurrency control.

Summary

We have discussed derived horizontal fragmentation and its checking for correctness. We have also discussed vertical fragmentation. VF includes two approaches grouping and splitting.

Course Title: **Distributed Database Management Systems**
Course Code: **CS712**
Instructor: **Dr. Nayyer Masood (nayyerm@yahoo.com)**
Lecture No: **17**

In previous lecture:

- Concluded DHF
- Started Vertical Fragmentation (VF)

In this lecture:

- Continue with VF
    0. Information Requirement
    1. Attribute affinities

VF-Information Requirements

The major information required for vertical fragmentation is related to applications. Since vertical partitioning places in one fragment those attributes usually accessed together, there is a need for some measure that would define more precisely the notion of togetherness this measure is the affinity of attributes which indicates how closely related the attributes are. It is not realistic to expect the designer or the users to be able to easily specify these values.

The major data requirement related to applications is their access frequencies. Let $Q = \{q1 , q2 ,\ldots, qq\}$ be the set of user queries (applications) that will run on relation R[A1, *A*2 ,…, *A*n]. Then, for each query qi and each attribute Aj, we associated an attribute usage value, denoted as *use* (*qi, Aj)* and defined as follows:

$$use(q_i, A_j) = \begin{cases} 1 \text{ if attribute } A_j \text{ is referenced by query } q_i \\ 0 \text{ otherwise} \end{cases}$$

The $use(q_i, \bullet)$ vectors for each application are easy to define if the designer knows the applications that will run on the database.

Example

Consider relation PROJ(jNo, jName, budget, loc). Assume that the following applications are defined to run on this relation. In each case we also give the SQL specification.

q1: SELECT BUDGET FROM PROJ WHERE JNO=Value

q2: SELEC JNAME, BUDGET FROM PROJ

q3: SELECT JNAME FROM PROJ WHERE LOC=Value

q4: SELECT SUM(BUDGET) FROM PROJ WHERE LOC=Value

According to these four applications, the attribute usage values can be defined. As a notational convenience, we let A1= jNo, A2= jName, A3= budget, and A4= loc. The usage values are defined in matrix form as shown in figure 1, where entity (i, j) denotes use(qi, Aj).

Attribute usage values are not sufficiently general to form the basis of attribute splitting and fragmentation. This is because these values do not represent the weight of application frequencies. The frequency measure can be included in the definition of attribute affinity measure aff(Ai, Aj), which measures the bond between two attributes of a relation according to how they are accessed by applications.

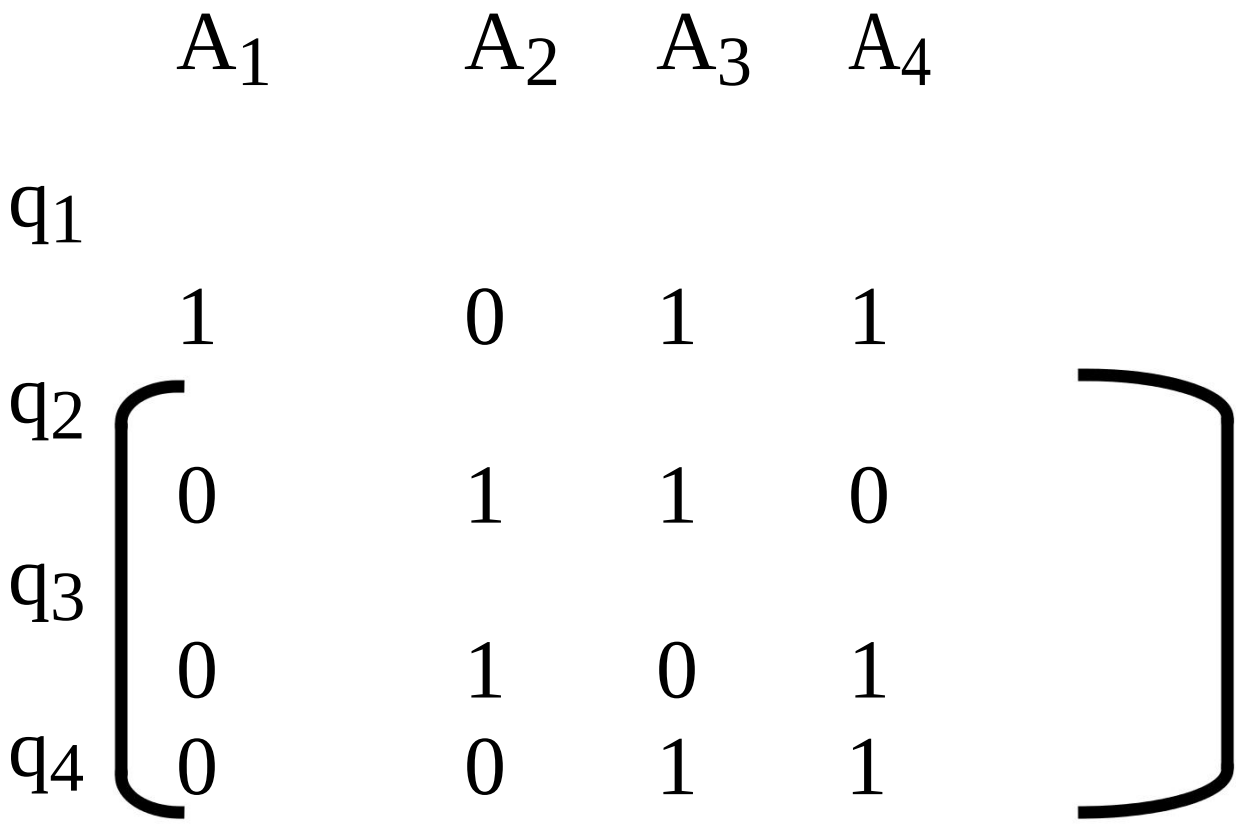


Figure1: Attribute Usage Matrix

The attribute affinity measure between two attributes Ai and Aj of a relation R(A1, A2, …., An) with respect to the set of applications Q = {q1, q2, …, qq) is defined as

$$aff(Ai, Aj) = \sum_{k|use(qk, Ai) = 1 \wedge use(qk, Aj) = 1 \forall PAY1} \sum refl(qk)\, accl(qk)$$

Where refl(qk) is number of accesses to attributes (Ai, Aj) for each execution of qk at site Sl, and accl(qk) is application access frequency measure from Sl.

The result of this computation is an n * n matrix, each element of which is one of the measures defined above. We call this matrix the attribute affinity matrix (AA).

Example
Let us assume that ref1(qk) = 1 for all qk and S1. If the application frequencies are

acc1(q1) = 15, acc2(q1) = 20, acc3(q1) = 10

acc1(q2) = 5, acc2(q2) = 0, acc3(q2) = 0

acc1(q3) = 25, acc2(q3) = 25, acc3(q3) = 25

acc1(q4) = 3, acc2(q4) = 0, acc3(q4) = 0

Since the only application that accesses both of the attributes is q1. the complete attribute affinity matrix is shown in figure 2.

	A_1	A_2	A_3	A_4
A_1	45	0	45	0
A2	0	80	5	75
A_3	45	5	53	3
A_4	0	75	3	78

Figure2: Attribute affinity matrix (AA)

Clustering Algorithm
The fundamental task in designing a vertical fragmentation algorithm is to find some means of grouping the attributes of a relation based on the attribute affinity values in AA. It has been suggesting that the bond energy algorithm (BEA) should be used for this purpose. It is considered appropriate for the following reasons:

- It is designed specifically to determine groups of similar items as opposed to say a linear ordering of the items (i.e. it clusters the attributes with larger affinity values together, and the ones with smaller values together).
- The final groupings are insensitive to the order in which items are presented to the algorithm.

- The computation time of the algorithm is reasonable [O(n2), where n is the number of attributes].
- Secondary interrelationships between clustered attribute groups and identifiable.

The bond energy algorithm takes as input the attribute affinity matrix, permutes its rows and columns and generates a clustered affinity matrix (CA).
Generation of clustered affinity matrix (CA) is done in three steps:

1. Initialization. Place and fix one of the columns of AA arbitrarily into CA. Column 1 was chosen in the algorithm.

2. Iteration. Pick each of the remaining n-I columns (where I is the number of columns already placed in CA) and try to place them in the remaining i+1 positions in the CA matrix. Continue this step until no more columns remain to be placed.
3. Row ordering. Once the column ordering is determined, the placement of the rows should also be changed so that their relative positions match the relative positions of the columns.

Global Affinity Measure (AM)
AM is a single value that is calculated on the basis of positions of elements in AA and their surrounding elements.

$$AM = \sum_{i=1}^{n} \sum_{j=1}^{n} aff(A_i, A_j) [aff(A_i, A_{j-1}) + aff(A_i, A_{j+1}) + aff(A_{i-1}, A_i) + aff(A_{i+1}, A_i)]$$

aff(A0, Aj)= aff(Ai, A0) = aff(An+1, Aj)= aff(Ai, An+1)=0

Summary
We have discussed vertical fragmentation which includes information requirements and attribute affinity. We have also discussed the clustering algorithm.

Course Title: **Distributed Database Management Systems**
Course Code: **CS712**
Instructor: **Dr. Nayyer Masood (nayyerm@yahoo.com)**
Lecture No: **18**

In previous lecture:

- Continue with VF
    0. Information Requirement
    1. Attribute affinities

In this lecture:

- Continue with VF
  - Global Affinity Measure
  - Bond Energy Algorithm

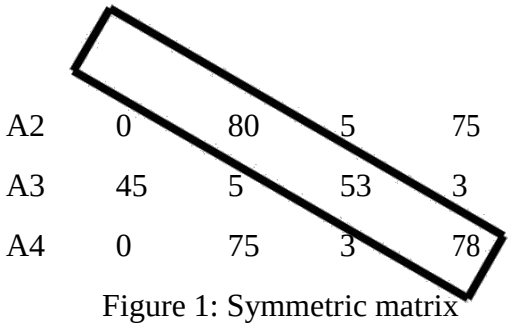
Global Affinity Measure

Global affinity measure can also be written as

$$AM = \sum_{i=1}^{n} \sum_{j=1}^{n} aff(A_i, A_j) [aff(A_i, A_{j-1}) + aff(A_i, A_{j+1}) + aff(A_{i-1}, A_j) + aff(A_{i+1}, A_j)]$$

The calculation of the bound between two attributes requires the multiplication of the respective elements of the two columns representing these attributes and taking the row-wise sum. Affinity matrix AA is symmetric as shown in figure 1.

	A1	A2	A3	A4
A1	45	0	45	0



A2	0	80	5	75
A3	45	5	53	3
A4	0	75	3	78

Figure 1: Symmetric matrix

Rather than taking this:

$$AM = \sum_{i=1}^{n} \sum_{j=1}^{n} aff(A_i, A_j) \, [aff(A_i, A_{j-1}) + aff(A_i, A_{j+1}) + aff(A_{i-1}, A_j) + aff(A_{i+1}, A_j)]$$

We take only..

$$AM = \sum_{j=}^{n} \left[\sum_{i=}^{n} aff(A_i, A_j) \, aff(A_i, A_{j-1}) + \sum_{i=}^{n} aff(A_i, A_j) \, aff(A_i, A_{j+1}) \right]$$

Let us define the bond between two attributes Ax and Ay as

$$bond(Ax, Ay) = \sum_{=1}^{n} aff(A_z, A_x) aff(A_z, A_y) \; z$$

Then AM can be written as

$$AM = \sum_{j=1}^{n} \left[bond(A_j, A_{j-1}) + bond(A_j, A_{j+1}) \right]$$

Example

Let us consider the AA matrix given in figure1 and study the contribution of moving attribute A4 between attribute A1 and A2 given by formula

Cont(A1, A4, A2) = 2bond(A1,A4)+2bond(A4,A2)-2bond(A1,A2)

Computing each term we get,

Bond(A1,A4) = 45*0+0*75+45*3+0*78 = 135
Bond(A4,A2) = 11865

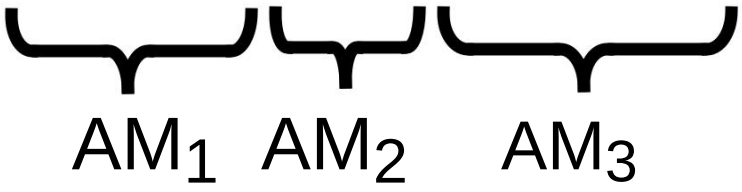
Bond(A1,A2) = 225

Therefore,
Cont(A1, A4, A2) = 2*135+2*11865-2*225 = 23550

The calculation of the bond between two attributes requires the multiplication of the respective elements of the two columns representing these attributes and taking the row-wise sum.

Approach for VF
Find the combination that has the max AM. It will represent best grouping among attributes. We can find all possible AMs or We can use the BEA (Bond Energy Algorithm).

Bond Energy Algorithm (BEA)
A1, A2, …, Ai-1, Ai, Aj, Aj+1, ……, An

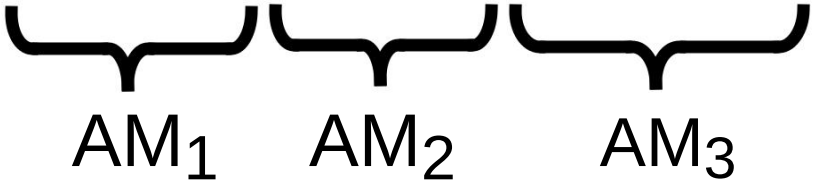


The global affinity measure for these attributes can be written as

AMold = AM1 + AM2+ AM3

AM2 involves Bond(Ai, Aj) twice

A1, A2, …, Ai-1, Ai, Ak, Aj, Aj+1, ……, An



AMnew = AM1 + AM2+ AM3

AM2 involves Bond(Ai, Ak) and Bond(Ak, Aj) twice

Contribution
Thus the net contribution to the global affinity measure of placing attribute Ak between Ai and Aj is

Cont(Ai, Ak, Aj) = AMnew- AMold
= 2Bond(Ai,Ak) + 2Bond(Aj, Ak) – 2Bond(Ai, Aj)

Steps in BEA

Input: The *AA* matrix

Output: The clustered affinity matrix *CA* which is a permutation of *AA*

Initialization: Place and fix one of the columns of *AA* in *C*A.

Iteration: Place the remaining *n-i* columns in the remaining i+1 positions in the *CA* matrix. For each column, choose the placement that makes the most contribution to the global affinity measure.

Row order: Order the rows according to the column ordering.

Bond Energy Algorithm

```
Input AA
Output CA
Begin
                CA (•, 1) ← AA (•, 1)
                CA (•, 2) ← AA (•, 2)
                Index ← 3
While index ≤ n do
Begin
For I from 1 to index  –1 by 1 do
     calculate cont(Ai-1, Aindex, Ai)    end-for
Calculate cont(Aindex-1, Aindex, Aindex+1)
Loc ← placement given by maximum cont value
For j from index to loc by –1 do

          CA (•, j) ← AA (•, j-1)       end-for
 CA (•, loc) ← AA (•, index)
Index ← index + 1
End-while
Order the rows according to relative order of columns
End {BEA}
```

Summary

We have discussed vertical fragmentation which global affinity measure and bond energy algorithm.

Course Title: **Distributed Database Management Systems**
Course Code: **CS712**
Instructor: **Dr. Nayyer Masood (nayyerm@yahoo.com)**
Lecture No: **19**

In previous lecture:

- Continued with VF
    0. Global Affinity Measure
    1. Bond Energy Algorithm

In this lecture:

- Continue with VF
    0. Example of VF
    1. Partitioning in CA

Clustered Affinity Matrix

According to the initialization step, we copy columns 1 and 2 of AA matrix to the CA matrix as shown in figure1 and start with column 3 (i.e. attribute A3).

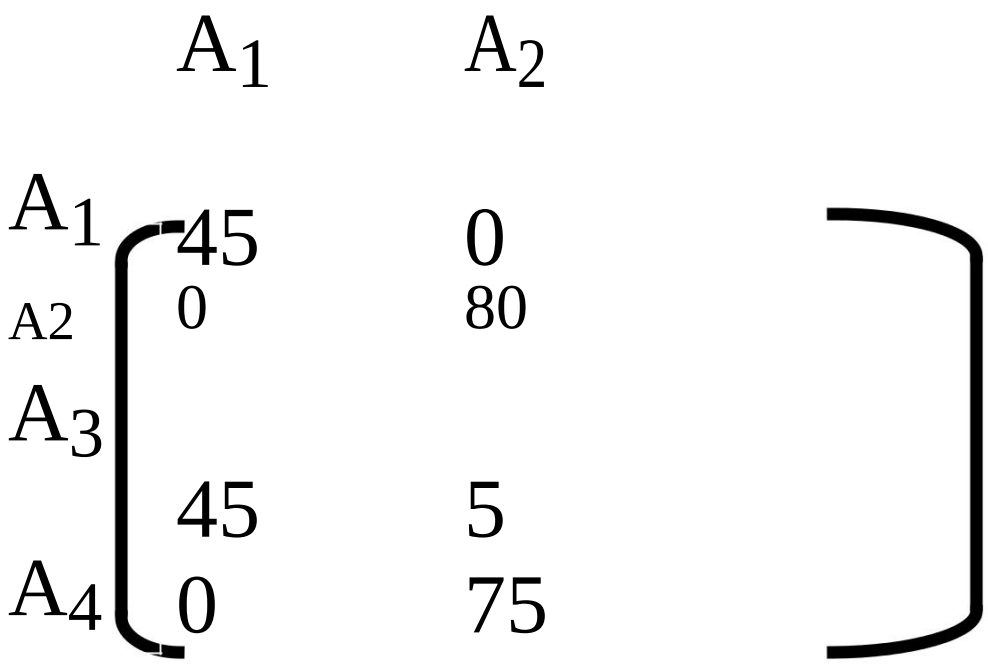


Figure 1: Calculation of Clustered Affinity Matirx

There are three alternative places where column 3 can be placed: to the left of column 1, resulting in ordering (3-1-2), in between columns 1 and 2, giving (1-3-2) and to right of 2, resulting (1-2-3). To compute the contribution of last ordering we have to compute cont (A2, A3, A4) rather than cont(A1, A2, A3). In this context A4 refers to the fourth index position in the CA matrix, which is empty as shown in figure 2, not to the attribute column A4 of the AA matrix.

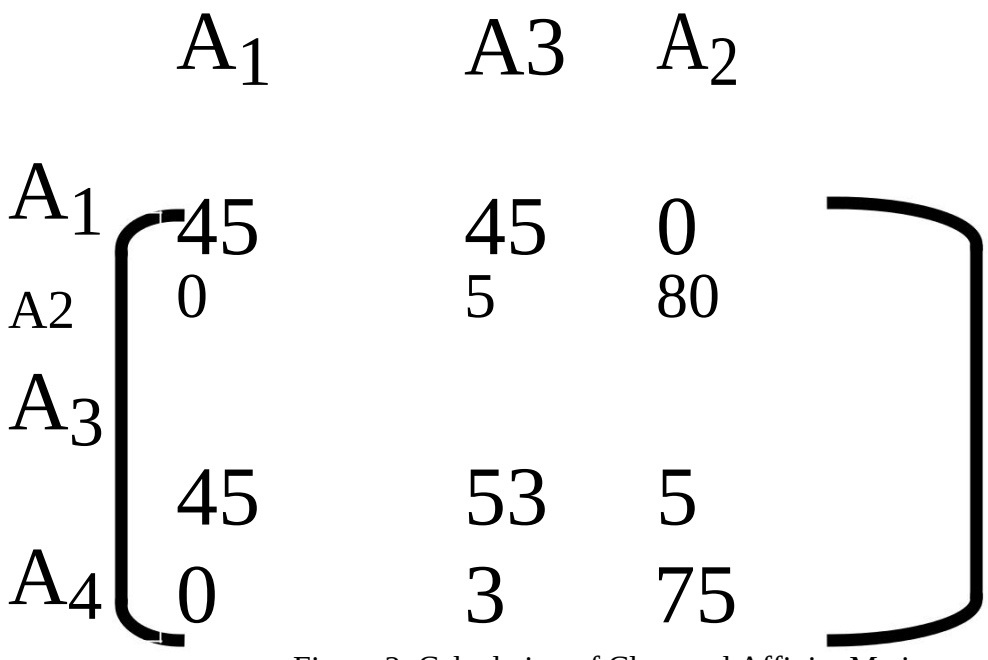


Figure 2: Calculation of Clustered Affinity Matirx

Let us calculate the contribution to the global affinity measure of each alternative.

Ordering (0-3-1):

cont (A0, A3, A1) = 2 bond(A0, A3) + 2 bond(A3, A1) – 2bond(A0, A1)

we know that bond(A0, A3) = 0 and (A0, A1) = 0 because A0 refers to the left of leftmost attribute.

$$bond(A_3, A_1) = \sum_{z=}^{4} aff(A_z, A_3)aff(A_z, A_1)$$

= aff(A1, A3)aff(A1, A1) + aff(A2, A3) aff(A2, A1) + aff(A3, A3)aff(A3, A1) + aff(A4, A3)aff(A4, A1)

= 45 * 45 + 5 * 0 + 53 * 45 + 3 * 0 = 4410

Thus

cont(A0, A3, A1) = 2 bond(A0, A3) + 2 bond(A3, A1) - 2bond(A0, A1)
= 2 * 0 + 2 * 4410 – 2 * 0
= 8820

Ordering (1-3-2)

cont (A1, A3, A2) = 2 bond(A1, A3) + 2 bond(A3, A2) - 2bond(A1, A2) bond(A1, A3) = bond(A3, A1) = 4410

bond(A3, A2) =0+400+265+225= 890 bond(A1, A2) =0+0+45*5= 225

cont(A1, A3, A2) = 2*4410+2*890–2*225
= 8820 + 1780 – 450
= 10150

Ordering (2-3-4)

cont (A2, A3, A4) = 2 bond(A2, A3) + 2 bond(A3, A4) - 2bond(A2, A4) bond(A2, A3) = 890

bond(A3, A4) = 0

bond(A2, A4) = 0

cont (A2, A3, A4) = 2 * 890 + 0 +
0 = 1780

Ordering (0-3-1) = 8820
Ordering (1-3-2) = 10150
Ordering (2-3-4) = 1780

Compute the contribution by placing 4th attribute at different places in CA
We need to work out

- Ordering (0-4-1)
- Ordering (1-4-3)
- Ordering (3-4-2)
- Ordering (2-4-5)

Finally, the columns order changed but rows still in same order as shown in figure 3. in figure 3 we see the creation of two clusters: one is in the upper left corner and contains the smaller affinity values and the other is the lower left corner and contains the larger affinity value. When the CA matrix is big, usually more than two clusters are formed and there are more than one candidate partitioning. Thus there is a need of approach this problem more systematically.

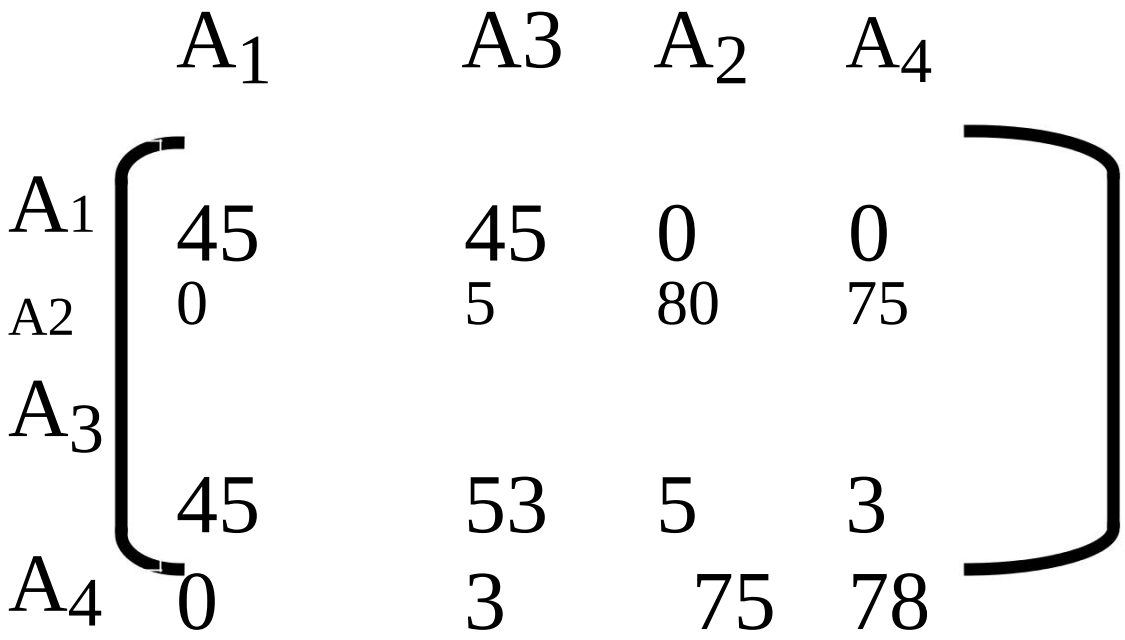


Figure 3: Calculation of Clustered Affinity Matirx

Clustering Summary

- We need AUM that reflects the Query-Attribute relationship
- AUM and FM are used to make AA
- Global Affinity Measure is used to establish the clusters of attributes
- Stronger affinities attributes and weaker ones are grouped in CA

Partitioning

The objective is of splitting activity is to find sets of attributes that are accessed solely, or for the most part by distinct sets of applications. For example, if it is possible to identify two attributes A1 and A2, which are accessed only by application q1 and attributes A3 and A4 which are accessed by two applications q2 and q3, it would be quite straight forward to decide on the fragments.

Consider the clustered attribute matrix of figure 4. if a point along the diagonal is fixed , two sets of attributes are identified. One set {A1, A2,….., Ai} is at the upper left hand corner and the second set {Ai+1,….,An} is to the right and to the bottom of this point. We call set top and set bottom and denote the attribute sets as TA and BA resp.

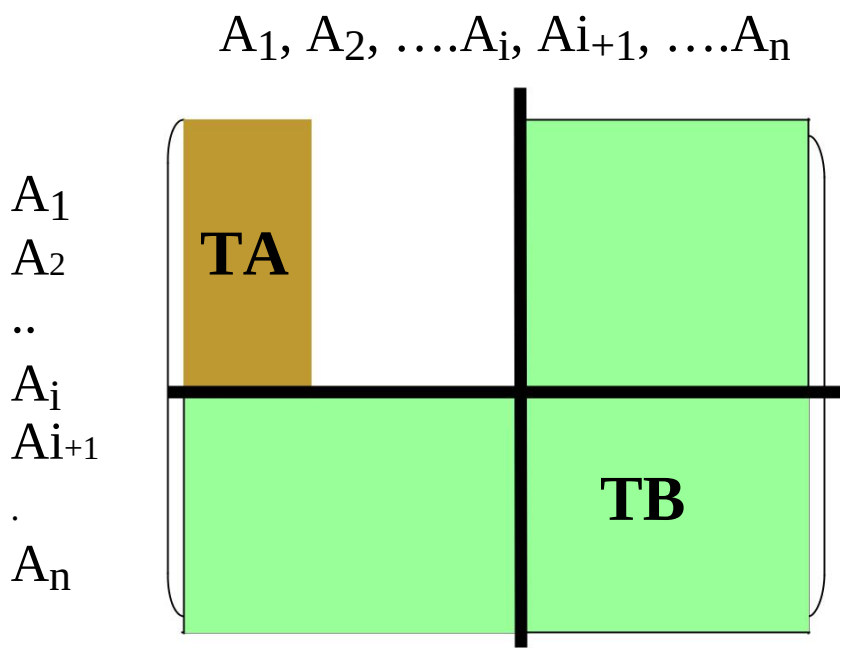


Figure 4: Locating a splitting point

OQ is the set of applications that access both TA and BA. CTQ is the number of accesses to attributes by applications that access only TA. CBQ is the number of accesses to attributes by applications that access only BA. COQ is the number of accesses to attributes by applications that access both TA and BA. Then find the point z along the diagonal that maximizes

z = CTQ * CBQ - COQ2

TQ = {q1} CTQ = 15 * 1 + 20 *1 + 10 * 1 = 45
BQ = {q3} CBQ = 25 * 1 + 25 * 1 + 25 * 1 = 75
OQ = {q2, q4} COQ = 5 * 1 + 0 + 0 + 3 * 1 + 0 + 0 = 8

z = CTQ * CBQ – COQ2
= 45 * 75 – 82 = 3375 – 64
= 3311

z2 = 3311
z1 = 0 – (45)2
z3= 0 – (78)2

Summary:
We have discussed the vertical fragmentation (VF) and the example of VF. We have also discussed the clustered affinity matrix and partitioning algorithm.

Course Title: **Distributed Database Management Systems**
Course Code: **CS712**
Instructor: **Dr. Nayyer Masood (nayyerm@yahoo.com)**
Lecture No: **20**

In previous lecture:

- Continue with VF
    0. Example of VF
    1. Computed CA
    2. Partitioning Algorithm

In this lecture:

- Continue with VF
- Hybrid Fragmentation
- Allocation Problem

VF- Two Problems

1- Clusters not in the sides, rather in the middle of CA
2- m-way partitioning

VF Correctness

A relation R, defined over attribute set *A* and key K, generates the vertical partitioning
$FR = \{R1, R2, \ldots, Rr\}$

Completeness
The set of attributes A over which the relation R is defined consists of:

$$A = U\, Ri$$

Completeness of vertical fragmentation is ensured.

Reconstruction

The reconstruction of the original global relation is made possible by the join operation. For a relation R with vertical fragmentation FR and key attribute(s) K,

$$R = \bowtie_K Ri, \quad \forall Ri \in FR$$

As long as each Ri is complete, the join operation will properly reconstruct R. each Ri should contain the key attribute(s) of R, or it should contain the system assigned tuple IDs (TIDs).

Disjointness

The disjointness of fragments is not as important in vertical fragmentation as it is horizontal fragmentation. There are two cases here:

1. TIDs are used in which case the fragments are disjoint since the TIDs that are replicated in each fragment are system assigned and managed entities, totally invisible to the users.
2. The key attributes are replicated in each fragment, in which case one can not claim that they are disjoint in the strict sense of the term. This duplication of the key attributes is known and managed by the system and does not have the same implications as tuple duplication in horizontally partitioned fragments.

Hybrid Fragmentation

Practically, applications require the fragmentation of both the types to be combined. So the nesting of fragmentations, i.e., one following the other, this alternative is called hybrid fragmentation. Disjointness and completeness have to be assured at each step, and reconstruction can be obtained by applying Join and Union in reverse order

Allocation

Assume that there are set of fragments F = {F1, F2 , …, Fn} and a network consisting of sites S ={S1 , S2 , …, Sm} on which a set of applications Q = {q1, q2 ,…, qq } is running. The allocation problem involves finding the optimal distribution of F to S.

Optimality

One of the important issues is the definition of optimality. The optimality can be defined with respect to two measures.

1. Minimal cost:

   The cost function consists of the cost of storing each Fi at a site Sj, the cost of querying Fi at site Sj, the cost of updating Fi at all sites where it is stored, and the cost of data

communication. The allocation problem finds an allocation schema that minimizes a combined cost function.

2. Performance:

   The allocation strategy is designed to maintain a performance metric. Two well known ones are to minimize the response time and to maximize this system throughput at each site.

It is a complex problem to be solved mathematically, to make the things very simple, consider the allocation of a single fragment Fk, we make a number of assumptions and definitions that will enable us to model the allocation problem.

1) Assume that Q can be modified so that it is possible to identify the update and the retrieval only queries and to define the following for a single fragment Fk:

   T = {t1, t2, …, tm}

   Where ti is the read only traffic generated at site Si for Fk and

   U= {u1, u2, .., um}

   Where ui is the update traffic generated at site Si for Fk.

2) Assume that the communication cost between any two pair of sites Si and Sj is fixed for a unit of transmission. Assume that it is different for updates and retrievals in order that the following can be defined:

   C(T) = {c1,2, c1,3, …., c1,m, ….cm-1, m}
   C'(T) = {c'1,2, c'1,3, …., c'1,m, ….c'm-1, m}

   Where cij is the unit of communication cost for retrieval requests between sites Si and Sj and c'ij is the unit communication cost for update requests between sites Si and Sj.

3) Let the cost of storing the fragment at site Si be di. We can define D = {d1, d2, ……., dm} for the storage cost of fragment Fk at all the sites.
4) Assume there are no capacity constraints for either the sites or the communication links.

Allocation problem is to find the sites out of set of sites S, where the copy of Fk will be stored. In the following xj denotes the decision variable for the placement such that

$$x_j = \begin{cases} 1 \text{ if the fragment } F_k \text{ is assigned to site } S_j \\ 0 \text{ otherwise} \end{cases}$$

The specification of the allocation problem will be

Subject to

Xj = 0 or 1

That concludes our discussion on Fragmentation. Lets summarize it

Fragmentation Summary

- Fragmentation is splitting a table into smaller tables
- Alternatives
  - Horizontal
  - Vertical
  - Hybrid

Horizontal Fragmentation

- Splits a table into horizontal subsets (row wise)
- Primary and Derived Horizontal Fragmentation
- We need major simple predicates (Pr); should be complete & minimal
- Pr is transformed into Pr'
- Minterm (M) predicates from Pr'
- Correctness of PHF depends on the Pr'
- Derived Horizontal Fragmentation is based on Owner-member link

Vertical Fragmentation is more complicated due to more options

- Based on attributes' affinities
- Calculated using usage data and access frequencies from different sites
- AA is transformed into CA using BEA
- CA establishes clusters of attributes that are split for efficient access

Hybrid Fragmentation combines HF and VF

- That concludes Fragmentation

Summary:

We have discussed the vertical fragmentation (VF) and hybrid fragmentation. We have also discussed the allocation problem.

Course Title: **Distributed Database Management Systems**
Course Code: **CS712**
Instructor: **Dr. Nayyer Masood (nayyerm@yahoo.com)**
Lecture No: **21**

In previous lecture:

- Continue with VF
- Hybrid Fragmentation
- Allocation Problem

In this lecture:

- Replication

Replication

Storing a separate copy of database at each of two or three sites

Advantages

- Reliability.
- Fast response.
- May avoid complicated distributed transaction integrity routines (if replicated data is refreshed at scheduled intervals.)
- De-couples nodes (transactions proceed even if some nodes are down.)
- Reduced network traffic at prime time (if updates can be delayed.)

Disadvantages

- Additional requirements for storage space.
- Additional time for update operations.
- Complexity and cost of updating.

– Integrity exposure of getting incorrect data if replicated data is not updated simultaneously.

Therefore, better when used for non-volatile data.

Replication Architecture in SQL Server

Replication uses a publish-subscribe model for distributing data. A Publication is group of related data and objects that we want to replicate together. A Publisher is a server that is the source of data to be replicated. A Subscriber is a server that receives the data replicated by the publisher. The Subscriber defines a subscription to a particular publication. A Distributor is a server that performs various tasks when moving articles from Publishers to Subscribers. The actual tasks performed depend on the type of replication performed.

SS Agents: software components used in replication; 4 types, Snapshot, Merge, Distribution, Log Reader.

1) Snapshot:
   Runs at least once in all reps

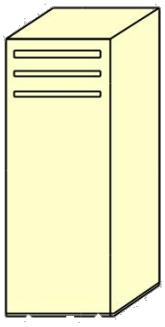
2) Distribution:
   Performs different activities, mainly distributes publication to subscribers

3) Log reader:
   Used in transactional rep, reads log files of all servers involved

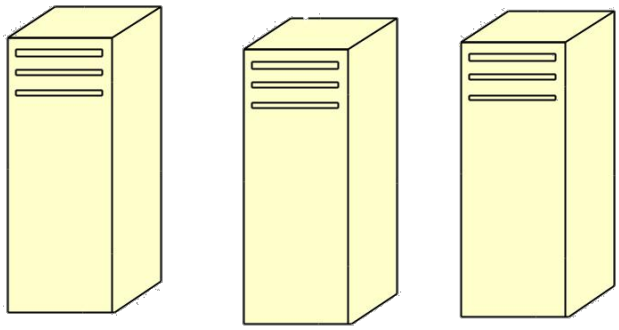
4) Merge:
   Meld changes from different servers made since last snapshot

Push/Pull subscriptions; depends where the distributor is running.

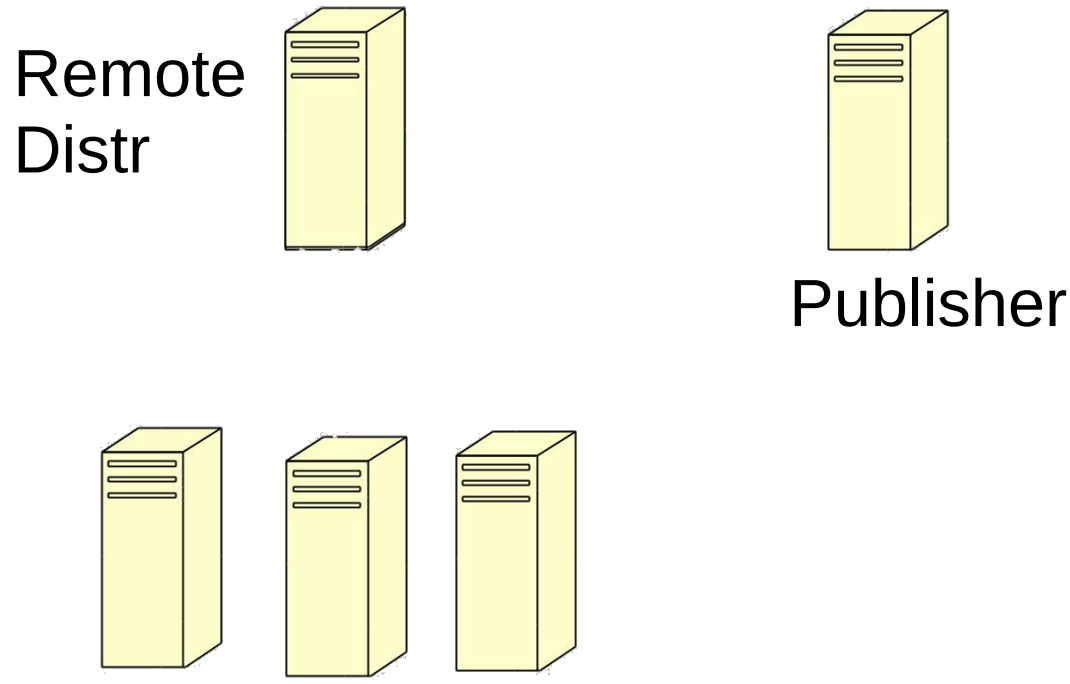
Replication Models



Publ/Dist



Central Publisher / Distributor



Replication Methods

- Snapshot Replication
- Transaction Rep.
- Merge Replication.

Snapshot Replication

- Preferred when subscribers need read-only access
- Higher latency (working without updated data) low Bandwidth
- Used in all types, initially
- Can be push or pull
- How to do it.

Summary:

We have discussed the replication, its advantages and disadvantages. We have also discussed Replication Architecture in SQL Server, replication models and replication methods.

Course Title: **Distributed Database Management Systems**
Course Code: **CS712**
Instructor: **Dr. Nayyer Masood (nayyerm@yahoo.com)**
Lecture No: **22**

In previous lecture:

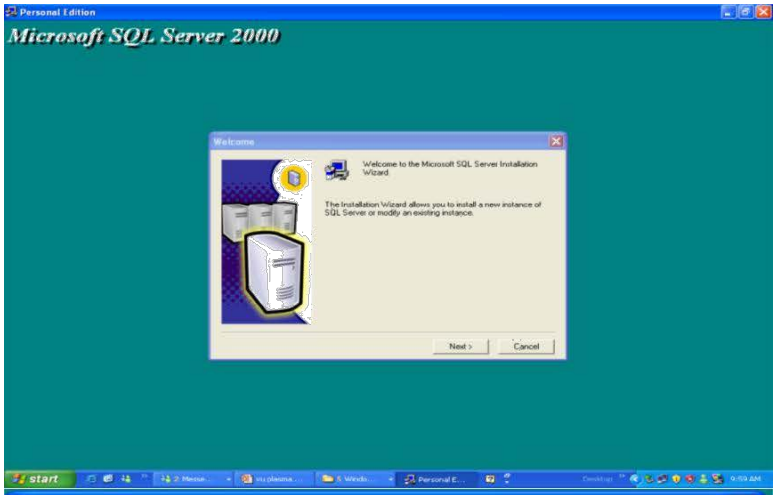
- Replication
- Snapshot replication

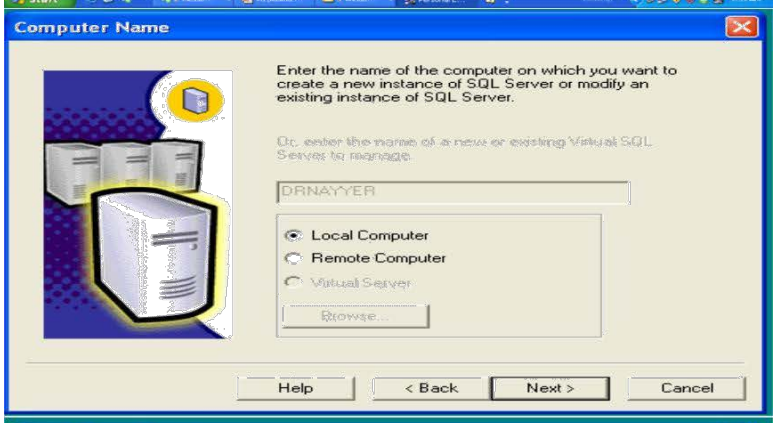
In this lecture:

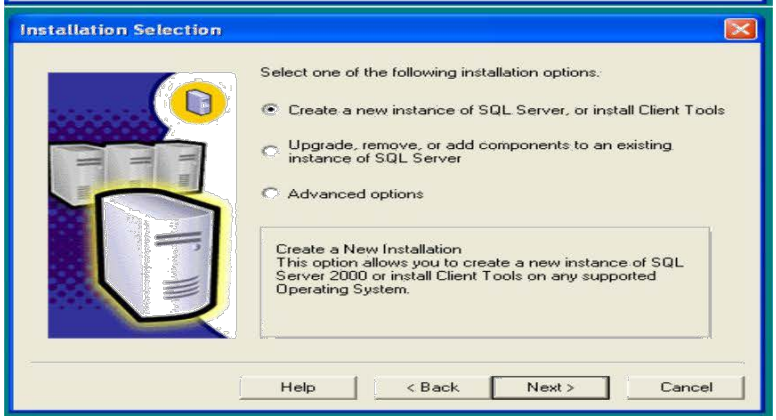
- Replication

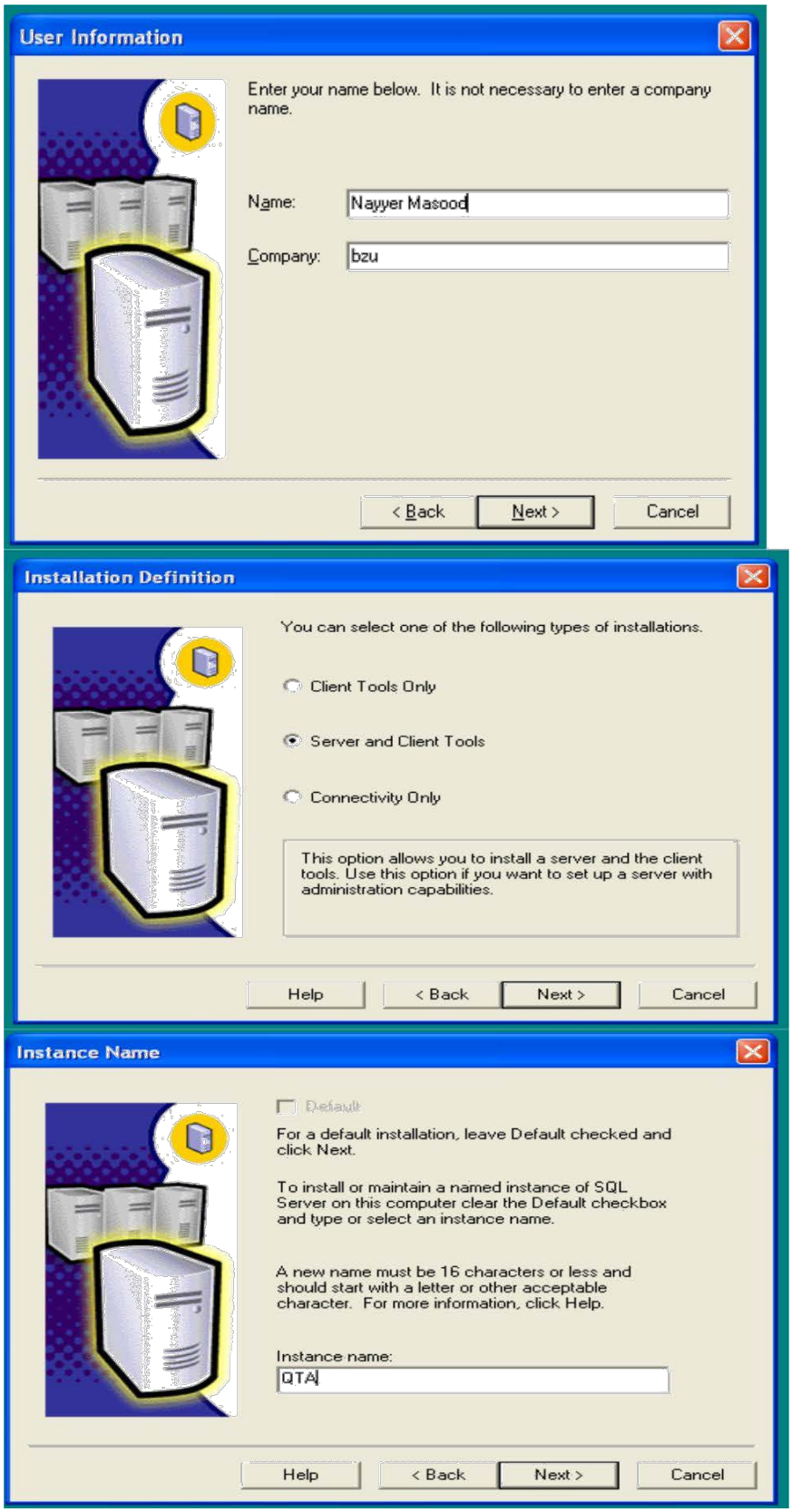
Setting up Replication

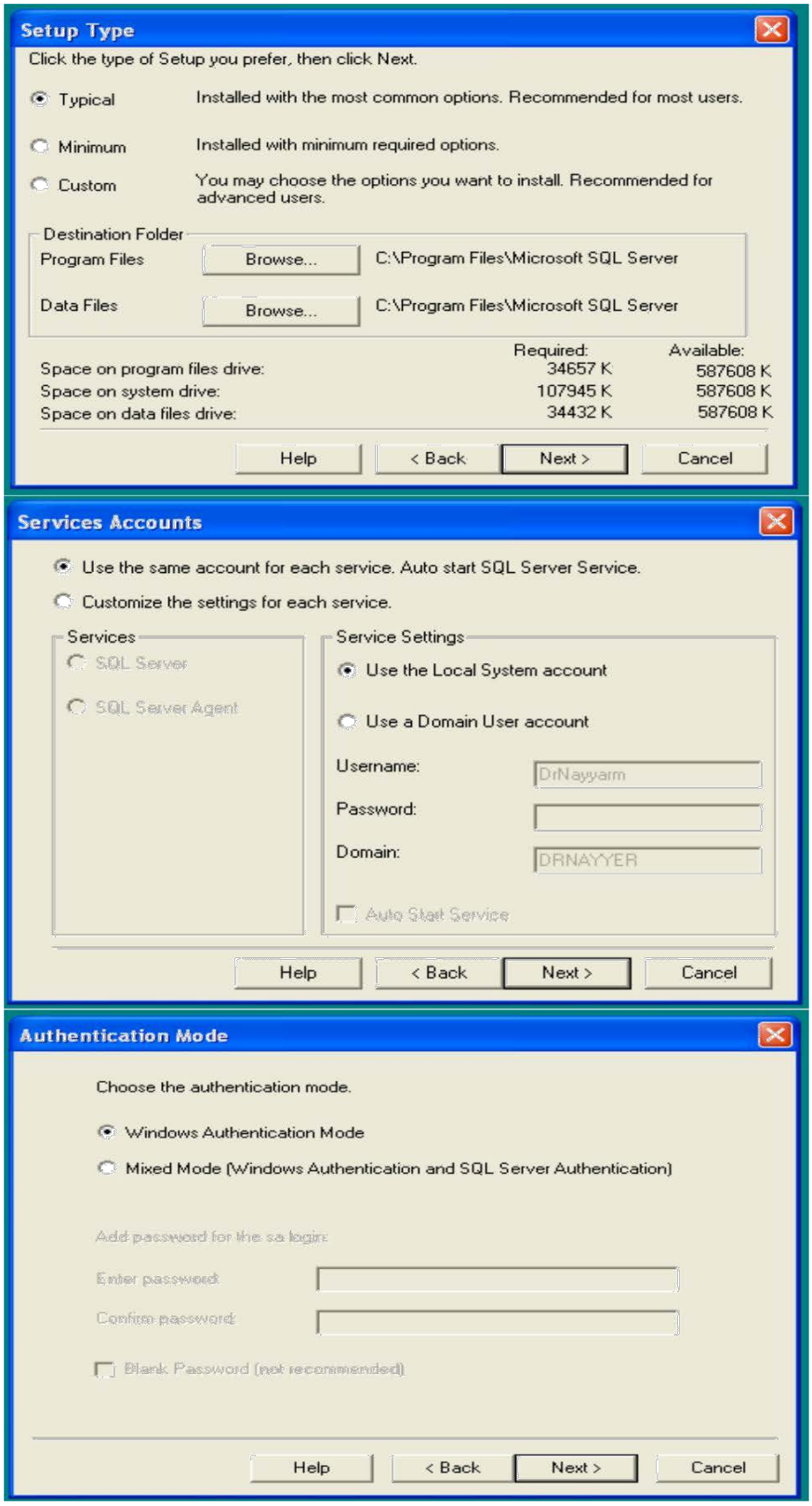
- Step by step process, **follow the sequence, it's a must**
- The Environment.
- To begin with, you can perform the experiment on a single machine, though it won't be a distributed environment, but it will be very simple
- Install multiple instances of SQL Server on your computer
- This simple setup will work with Windows XP and Personal Edition of SQL Server, but later experiments won't.
- Installation of two instances of SQL Server
- Lets go through one installation
- Instance names being PESH and QTA.

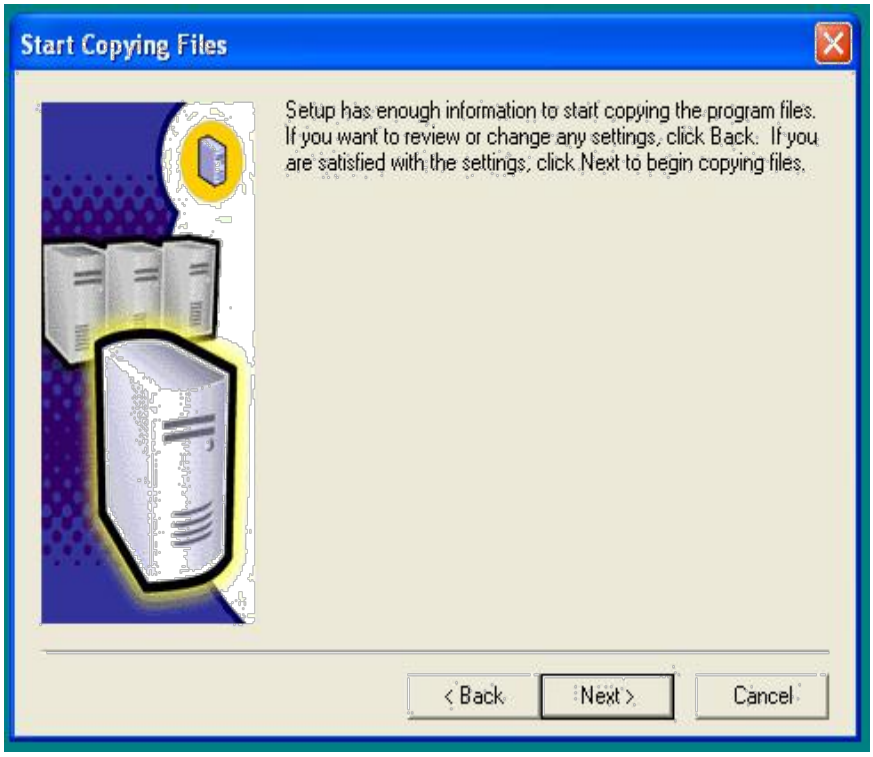

Personal Edition
Microsoft SQL Server 2000
Welcome
Welcome to the Microsoft SQL Server Installation Wizard.
The Installation Wizard allows you to install a new instance of SQL Server or modify an existing instance.
Next >
Cancel
start

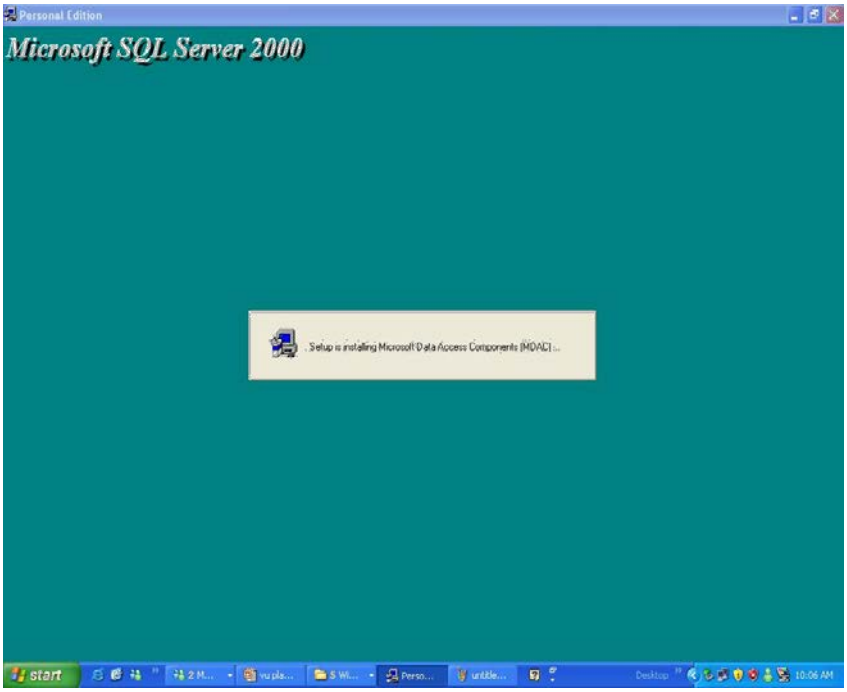

Computer Name
Enter the name of the computer on which you want to create a new instance of SQL Server or modify an existing instance of SQL Server.
Or, enter the name of a new or existing Virtual SQL Server to manage.
DRNAYYER
Local Computer
Remote Computer
Virtual Server
Browse...
Help
< Back
Next >
Cancel

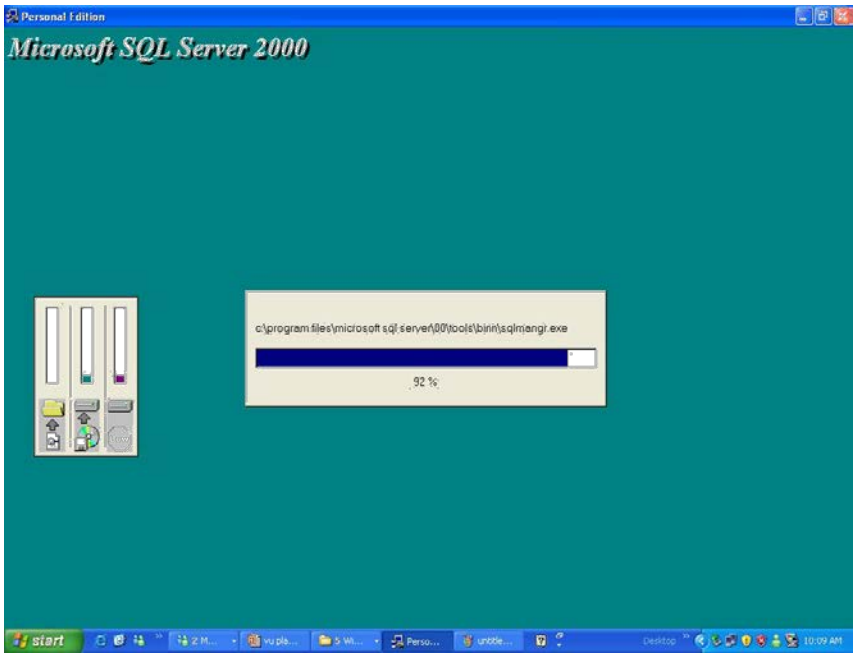

Installation Selection
Select one of the following installation options.
Create a new instance of SQL Server, or install Client Tools
Upgrade, remove, or add components to an existing instance of SQL Server
Advanced options
Create a New Installation
This option allows you to create a new instance of SQL Server 2000 or install Client Tools on any supported Operating System.
Help
< Back
Next >
Cancel


User Information
Enter your name below. It is not necessary to enter a company name.
Name:
Nayyer Masood
Company:
bzu
< Back
Next >
Cancel
Installation Definition
You can select one of the following types of installations.
Client Tools Only
Server and Client Tools
Connectivity Only
This option allows you to install a server and the client tools. Use this option if you want to set up a server with administration capabilities.
Help
< Back
Next >
Cancel
Instance Name
Default
For a default installation, leave Default checked and click Next.
To install or maintain a named instance of SQL Server on this computer clear the Default checkbox and type or select an instance name.
A new name must be 16 characters or less and should start with a letter or other acceptable character. For more information, click Help.
Instance name:
QTA
Help
< Back
Next >
Cancel

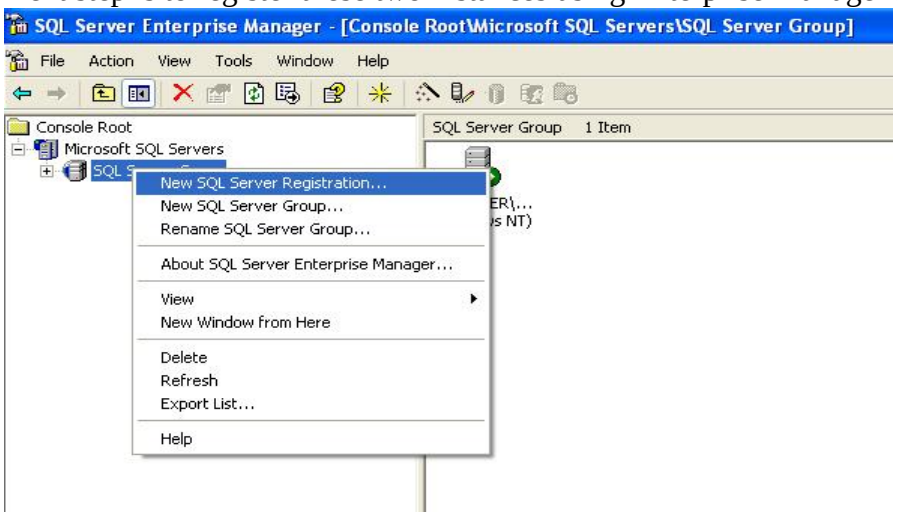

Setup Type
Click the type of Setup you prefer, then click Next.
Typical
Installed with the most common options. Recommended for most users.
Minimum
Installed with minimum required options.
Custom
You may choose the options you want to install. Recommended for advanced users.
Destination Folder
Program Files
Browse...
C:\Program Files\Microsoft SQL Server
Data Files
Browse...
C:\Program Files\Microsoft SQL Server
Required:
Available:
Space on program files drive:
34657 K
587608 K
Space on system drive:
107945 K
587608 K
Space on data files drive:
34432 K
587608 K
Help
< Back
Next >
Cancel
Services Accounts
Use the same account for each service. Auto start SQL Server Service.
Customize the settings for each service.
Services
SQL Server
SQL Server Agent
Service Settings
Use the Local System account
Use a Domain User account
Username:
DrNayyarm
Password:
Domain:
DRNAYYER
Auto Start Service
Help
< Back
Next >
Cancel
Authentication Mode
Choose the authentication mode.
Windows Authentication Mode
Mixed Mode (Windows Authentication and SQL Server Authentication)
Add password for the sa login:
Enter password:
Confirm password:
Blank Password (not recommended)
Help
< Back
Next >
Cancel

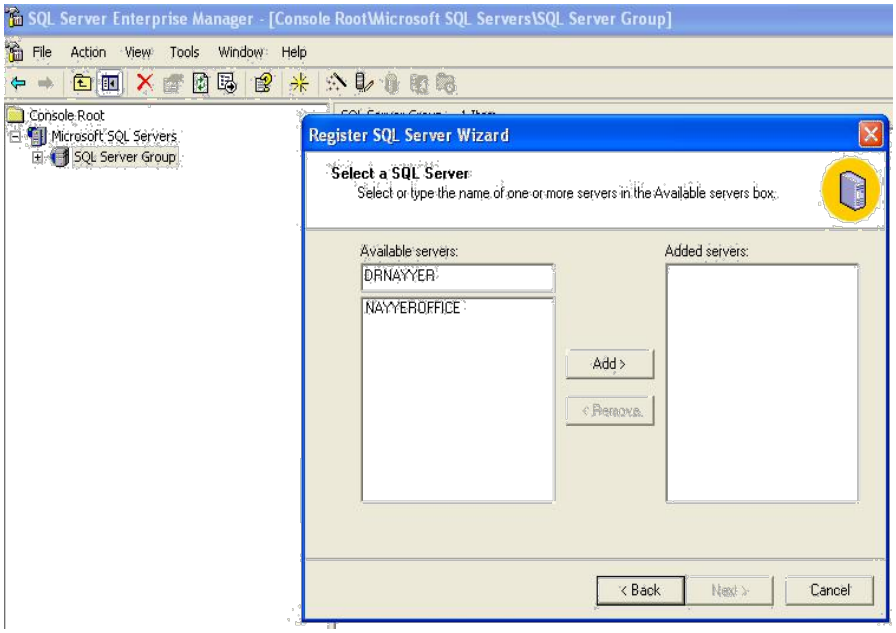

Start Copying Files
Setup has enough information to start copying the program files. If you want to review or change any settings, click Back. If you are satisfied with the settings, click Next to begin copying files.
< Back
Next >
Cancel

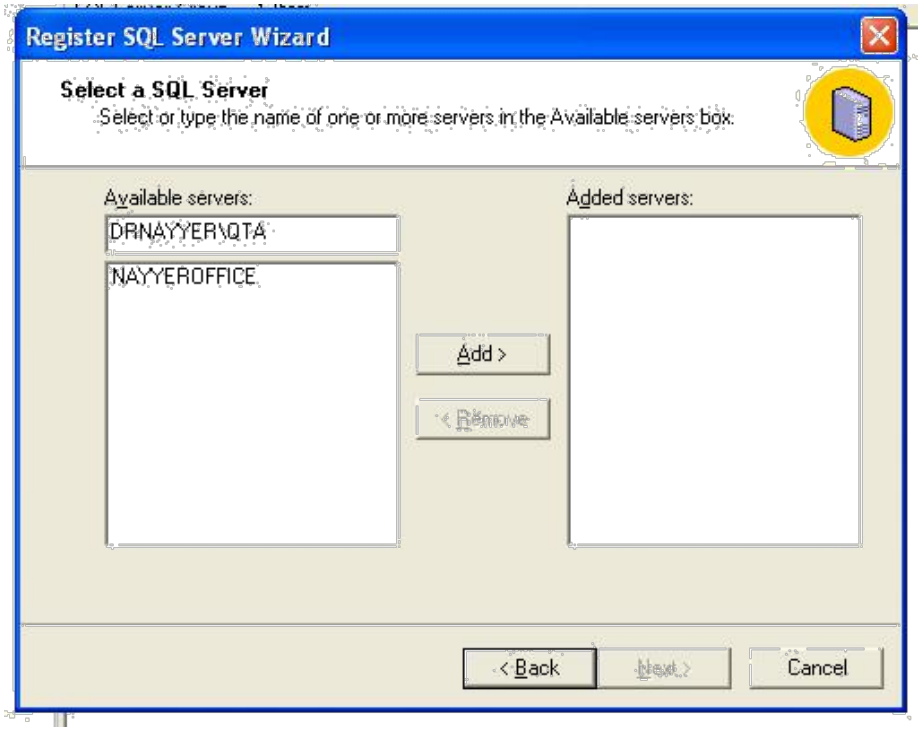

Personal Edition
Microsoft SQL Server 2000
Setup is installing Microsoft Data Access Components (MDAC)...
start



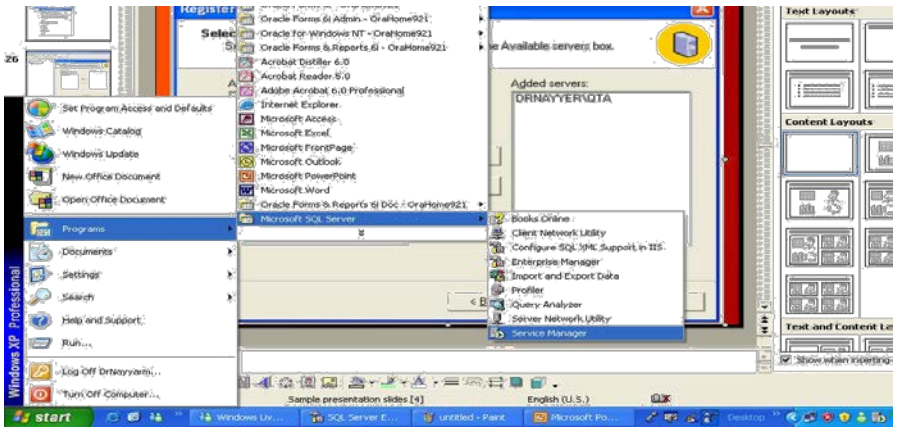
Next step is to register these two instances using Enterprise Manager as shown below

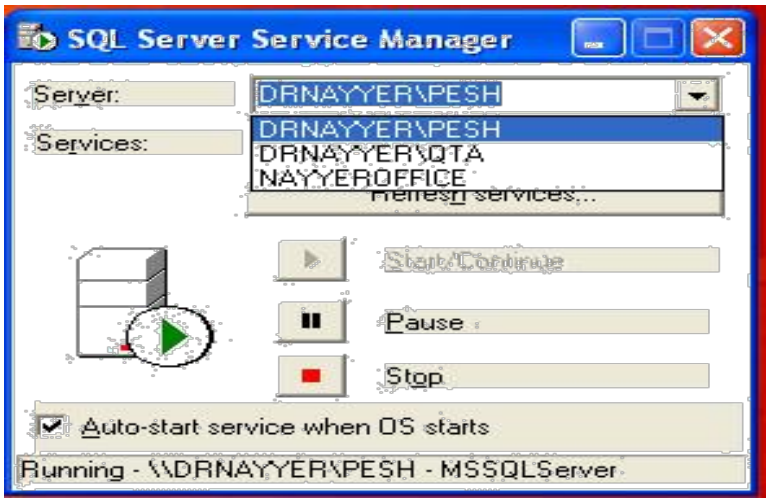



SQL Server Enterprise Manager - [Console Root\Microsoft SQL Servers\SQL Server Group]
File Action View Tools Window Help
Console Root
Microsoft SQL Servers
SQL Server Group
Register SQL Server Wizard
Select a SQL Server
Select or type the name of one or more servers in the Available servers box.
Available servers:
DRNAYYER
NAYYEROFFICE
Add >
< Remove
Added servers:
< Back
Next >
Cancel

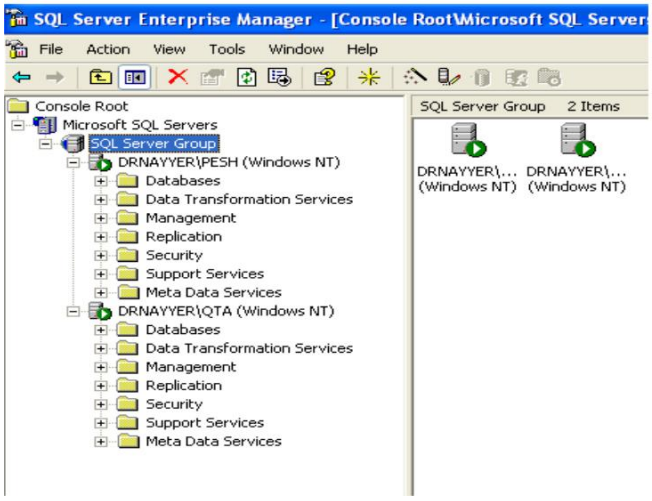

Register SQL Server Wizard
Select a SQL Server
Select or type the name of one or more servers in the Available servers box.
Available servers:
DRNAYYER\QTA
NAYYEROFFICE
Add >
< Remove
Added servers:
< Back
Next >
Cancel

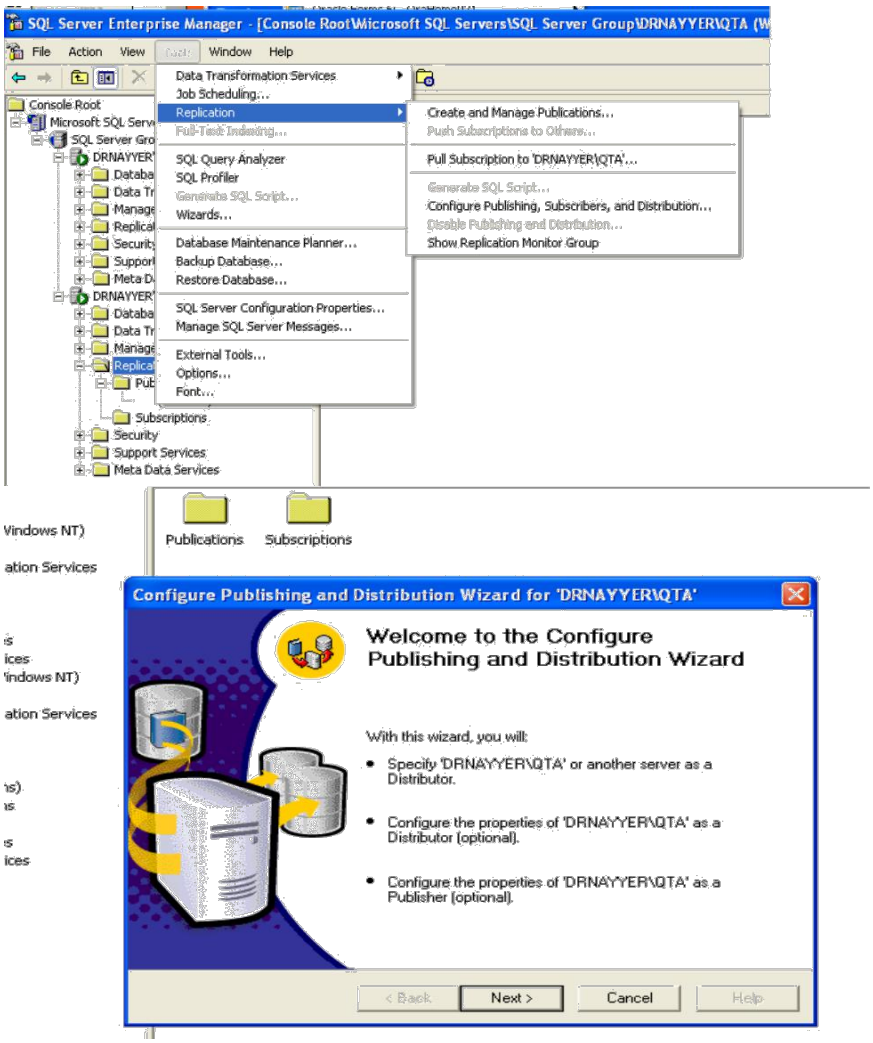

Register SQL Server Wizard
Select a SQL Server
Select or type the name of one or more servers in the Available servers box.
Available servers:
NAYYEROFFICE
Add >
< Remove
Added servers:
DRNAYYER\QTA
< Back
Next >
Cancel

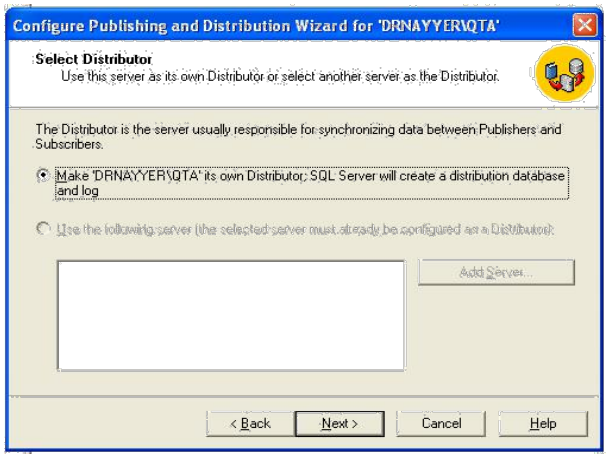

Added servers:
DRNAYYER\QTA
Set Program Access and Defaults
Windows Catalog
Windows Update
New Office Document
Open Office Document
Programs
Documents
Settings
Search
Help and Support
Run...
Log Off Drnayyer...
Turn Off Computer...
Windows XP Professional
Microsoft SQL Server
Books Online
Client Network Utility
Enterprise Manager
Import and Export Data
Profiler
Query Analyzer
Server Network Utility
Service Manager
Content Layouts
start



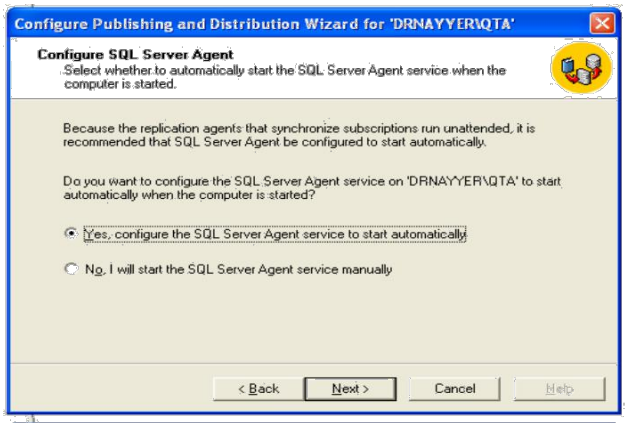
After a couple of default selections

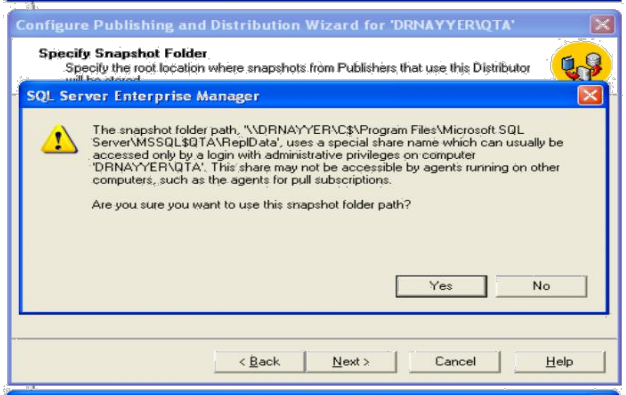


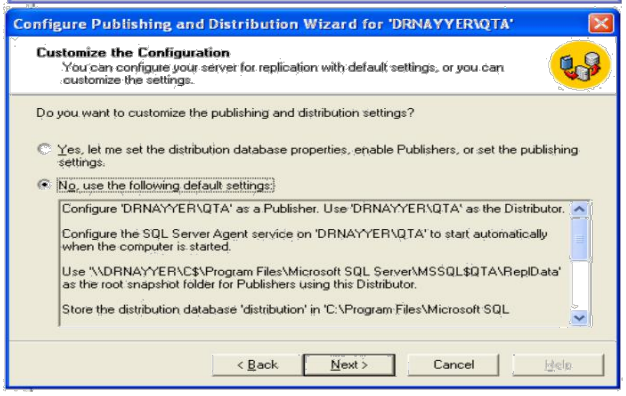


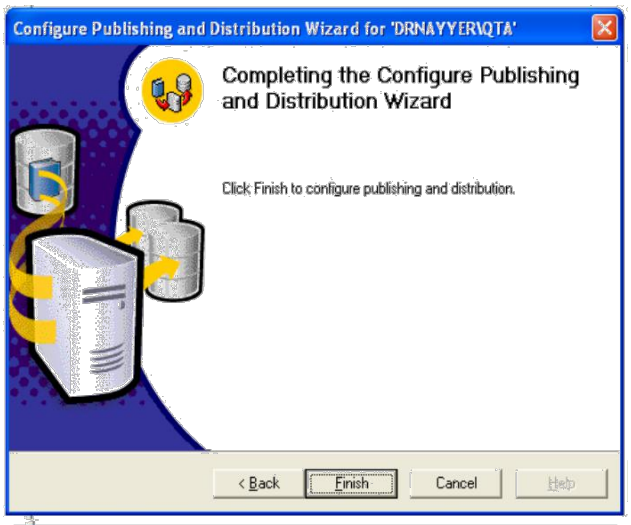


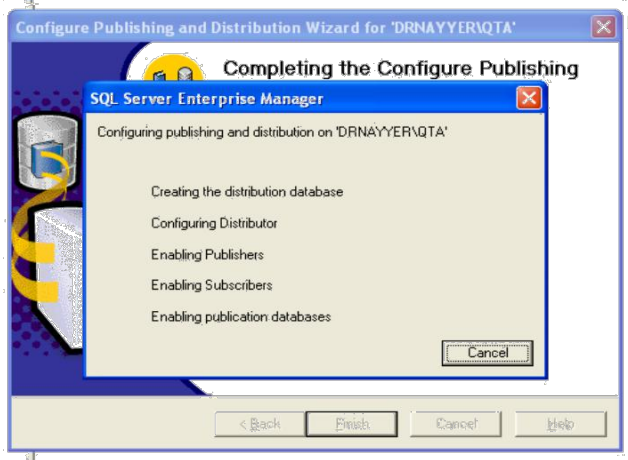
SQL Server Agent Startup Account


Configure Publishing and Distribution Wizard for 'DRNAYYER\QTA'
Configure SQL Server Agent
Select whether to automatically start the SQL Server Agent service when the computer is started.
Because the replication agents that synchronize subscriptions run unattended, it is recommended that SQL Server Agent be configured to start automatically.
Do you want to configure the SQL Server Agent service on 'DRNAYYER\QTA' to start automatically when the computer is started?
Yes, configure the SQL Server Agent service to start automatically
No, I will start the SQL Server Agent service manually
< Back
Next >
Cancel
Help

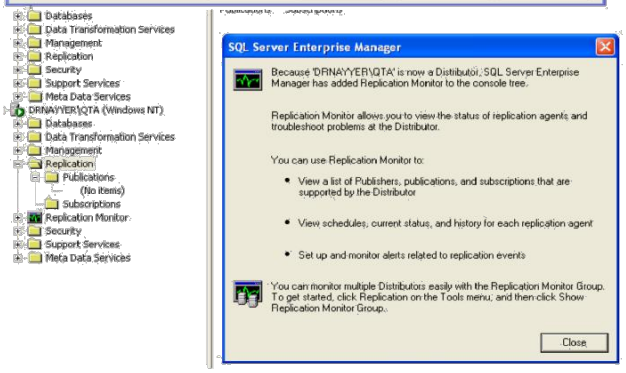

Configure Publishing and Distribution Wizard for 'DRNAYYER\QTA'
Specify Snapshot Folder
Specify the root location where snapshots from Publishers that use this Distributor
SQL Server Enterprise Manager
The snapshot folder path, '\\DRNAYYER\C$\Program Files\Microsoft SQL Server\MSSQL$QTA\ReplData', uses a special share name which can usually be accessed only by a login with administrative privileges on computer 'DRNAYYER\QTA'. This share may not be accessible by agents running on other computers, such as the agents for pull subscriptions.
Are you sure you want to use this snapshot folder path?
Yes
No
< Back
Next >
Cancel
Help


Configure Publishing and Distribution Wizard for 'DRNAYYER\QTA'
Customize the Configuration
You can configure your server for replication with default settings, or you can customize the settings.
Do you want to customize the publishing and distribution settings?
Yes, let me set the distribution database properties, enable Publishers, or set the publishing settings.
No, use the following default settings:
Configure 'DRNAYYER\QTA' as a Publisher. Use 'DRNAYYER\QTA' as the Distributor.
Configure the SQL Server Agent service on 'DRNAYYER\QTA' to start automatically when the computer is started.
Use '\\DRNAYYER\C$\Program Files\Microsoft SQL Server\MSSQL$QTA\ReplData' as the root snapshot folder for Publishers using this Distributor.
Store the distribution database 'distribution' in 'C:\Program Files\Microsoft SQL
< Back
Next >
Cancel
Help


Configure Publishing and Distribution Wizard for 'DRNAYYER\QTA'
Completing the Configure Publishing and Distribution Wizard
Click Finish to configure publishing and distribution.
< Back
Finish
Cancel
Help

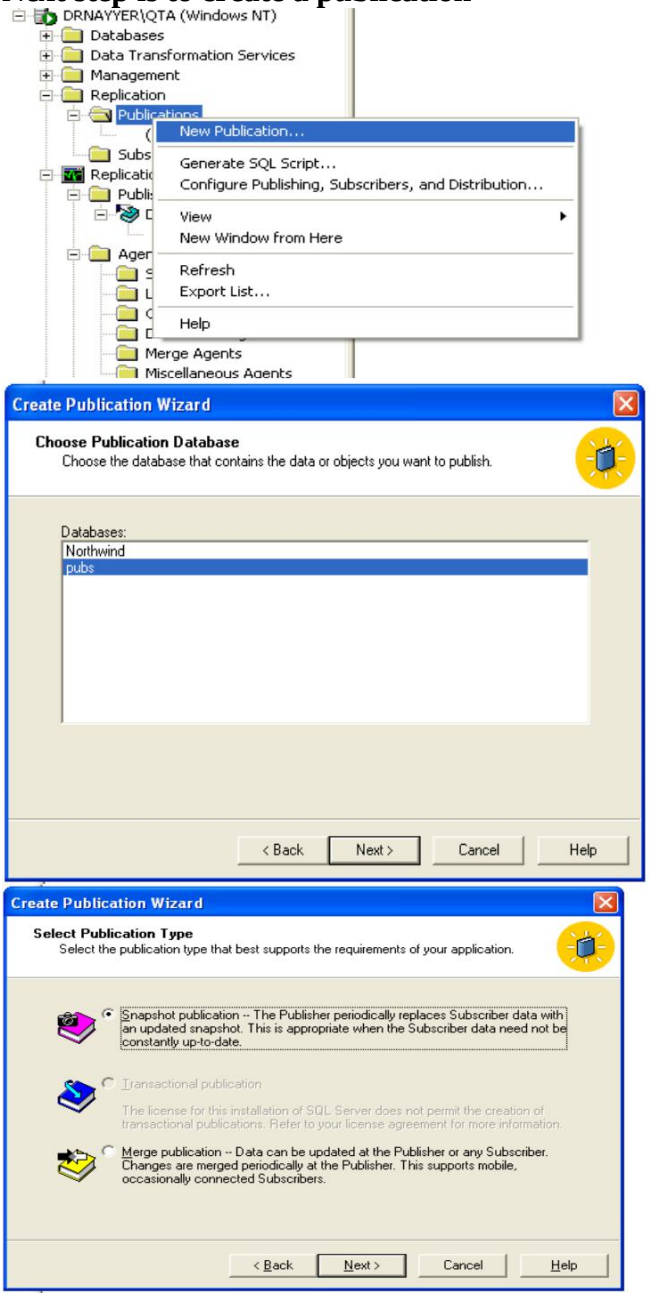

Configure Publishing and Distribution Wizard for 'DRNAYYER\QTA'
Completing the Configure Publishing
SQL Server Enterprise Manager
Configuring publishing and distribution on 'DRNAYYER\QTA'
Creating the distribution database
Configuring Distributor
Enabling Publishers
Enabling Subscribers
Enabling publication databases
Cancel
< Back
Finish
Cancel
Help


Configure Publishing and Distribution Wizard for 'DRNAYYER\QTA'
Completing the Configure Publishing and Distribution Wizard
SQL Server Enterprise Manager
SQL Server Enterprise Manager successfully enabled 'DRNAYYER\QTA' as the Distributor for 'DRNAYYER\QTA'.
OK
< Back
Finish
Cancel
Help

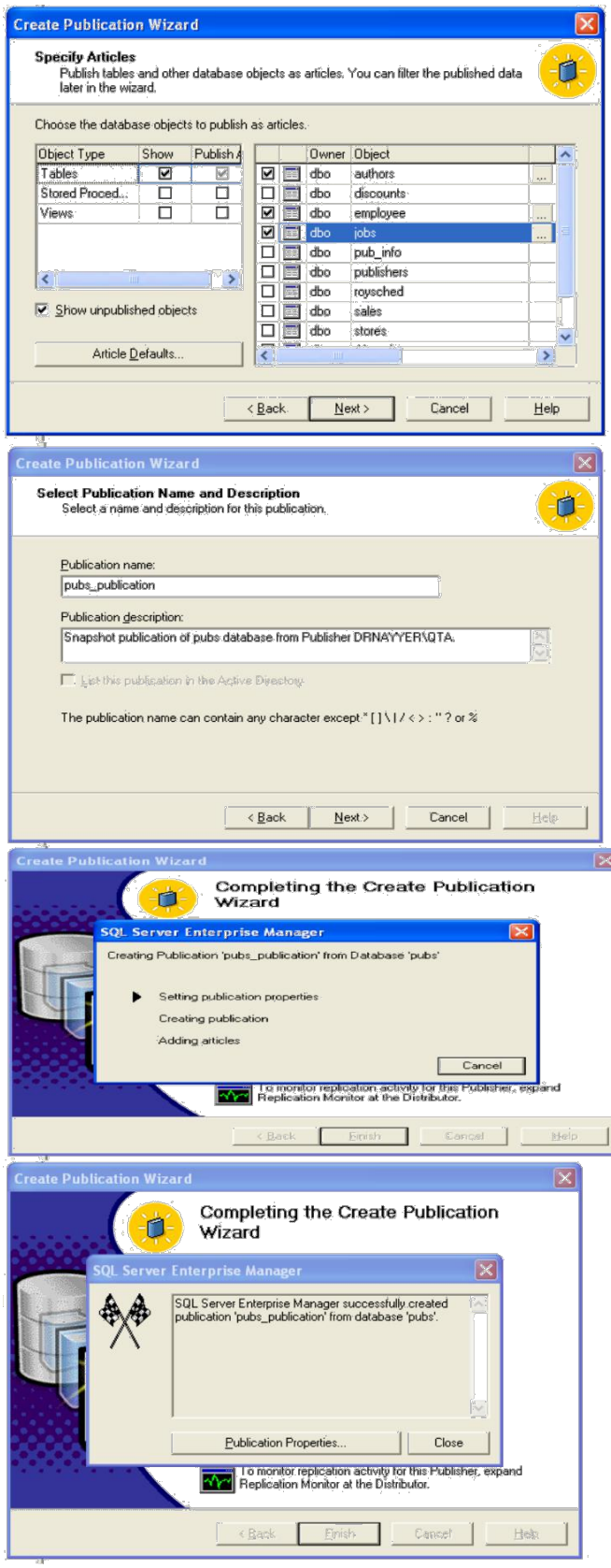

Databases
Data Transformation Services
Management
Replication
Security
Support Services
Meta Data Services
DRNAYYER\QTA (Windows NT)
Databases
Data Transformation Services
Management
Replication
Publications
(No items)
Subscriptions
Replication Monitor
Security
Support Services
Meta Data Services
SQL Server Enterprise Manager
Because 'DRNAYYER\QTA' is now a Distributor, SQL Server Enterprise Manager has added Replication Monitor to the console tree.
Replication Monitor allows you to view the status of replication agents and troubleshoot problems at the Distributor.
You can use Replication Monitor to:
View a list of Publishers, publications, and subscriptions that are supported by the Distributor
View schedules, current status, and history for each replication agent
Set up and monitor alerts related to replication events
You can monitor multiple Distributors easily with the Replication Monitor Group. To get started, click Replication on the Tools menu, and then click Show Replication Monitor Group.
Close

Nothing in the folders yet QTA is also set as Publisher

Next step is to create a publication

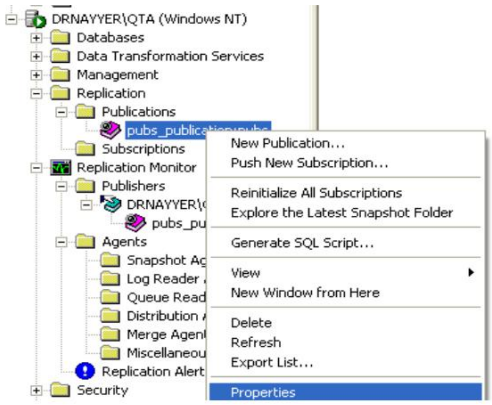


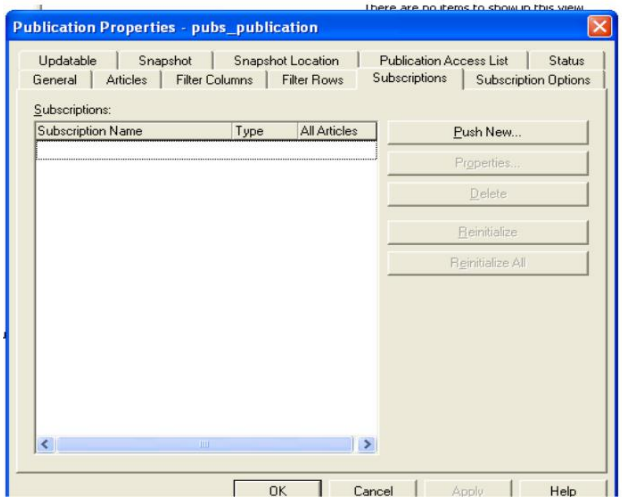
A couple of Defaults and then



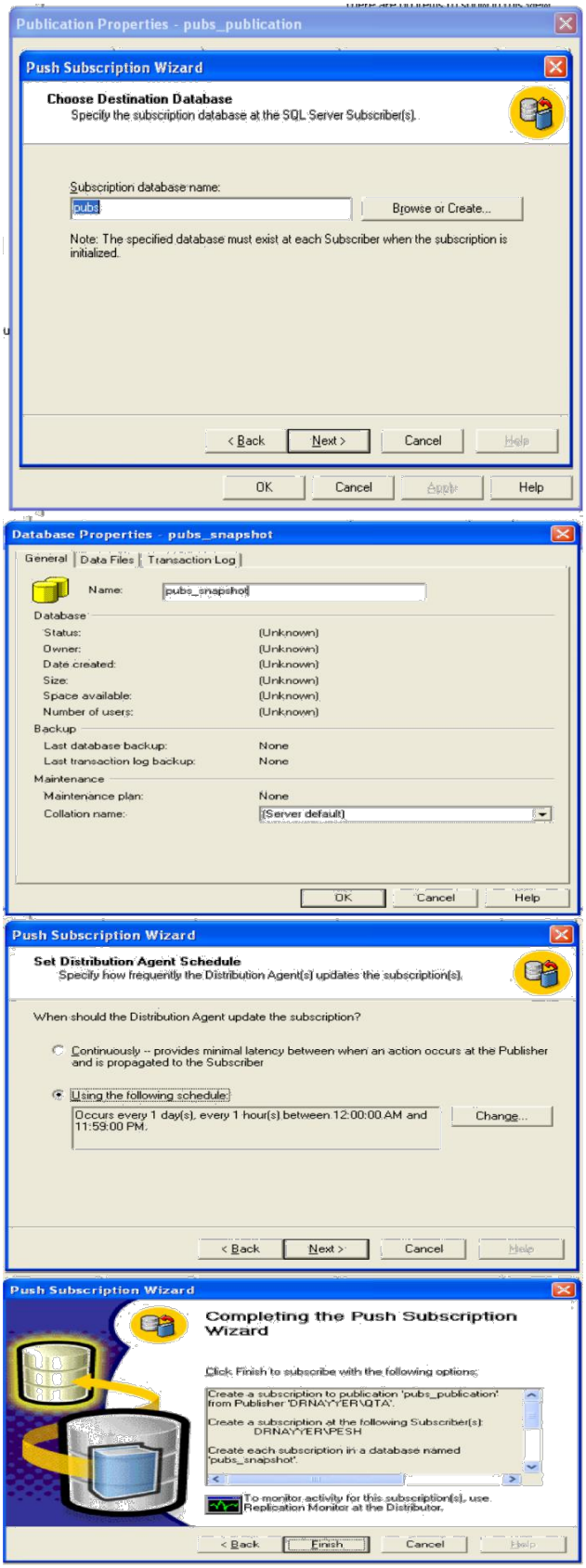
Publication Created

Now to create Subscriber

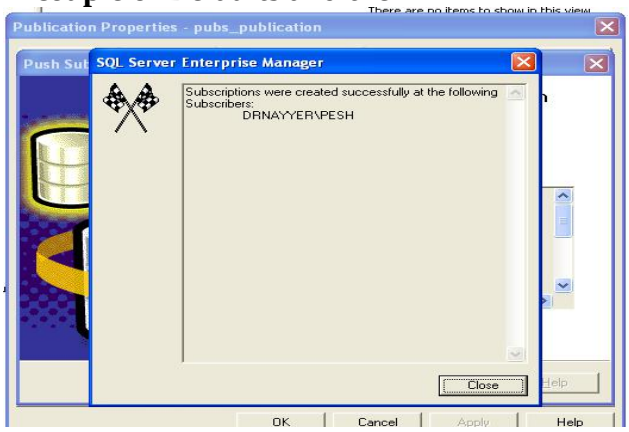




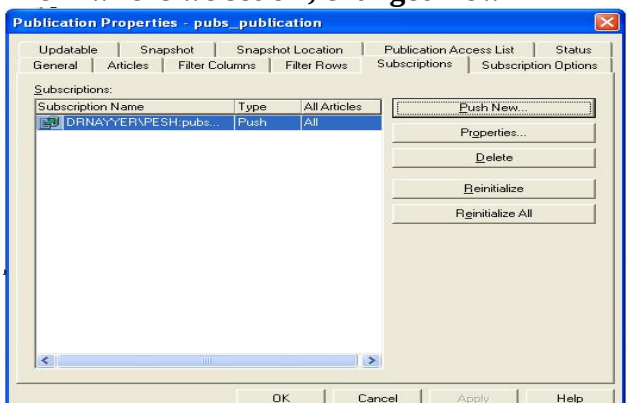
A welcome screen and then


Publication Properties - pubs_publication
Push Subscription Wizard
Choose Destination Database
Specify the subscription database at the SQL Server Subscriber(s).
Subscription database name:
pubs
Browse or Create...
Note: The specified database must exist at each Subscriber when the subscription is initialized.
< Back
Next >
Cancel
Help
OK
Cancel
Apply
Help
Database Properties - pubs_snapshot
General
Data Files
Transaction Log
Name:
pubs_snapshot
Database
Status:
(Unknown)
Owner:
(Unknown)
Date created:
(Unknown)
Size:
(Unknown)
Space available:
(Unknown)
Number of users:
(Unknown)
Backup
Last database backup:
None
Last transaction log backup:
None
Maintenance
Maintenance plan:
None
Collation name:
(Server default)
OK
Cancel
Help
Push Subscription Wizard
Set Distribution Agent Schedule
Specify how frequently the Distribution Agent(s) updates the subscription(s).
When should the Distribution Agent update the subscription?
Continuously -- provides minimal latency between when an action occurs at the Publisher and is propagated to the Subscriber
Using the following schedule:
Occurs every 1 day(s), every 1 hour(s) between 12:00:00 AM and 11:59:00 PM.
Change...
< Back
Next >
Cancel
Help
Push Subscription Wizard
Completing the Push Subscription Wizard
Click Finish to subscribe with the following options:
Create a subscription to publication 'pubs_publication' from Publisher 'DRNAYYER\QTA'.
Create a subscription at the following Subscriber(s):
DRNAYYER\PESH
Create each subscription in a database named 'pubs_snapshot'.
To monitor activity for this subscription(s), use Replication Monitor at the Distributor.
< Back
Finish
Cancel
Help

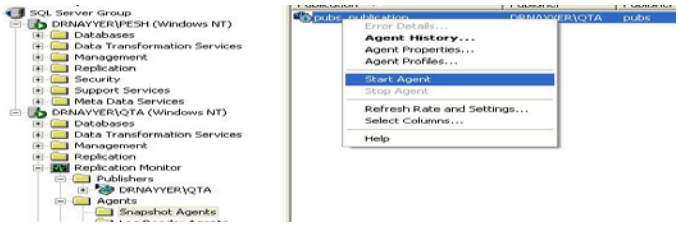
A couple of Defaults and then



From where we set off, changed now

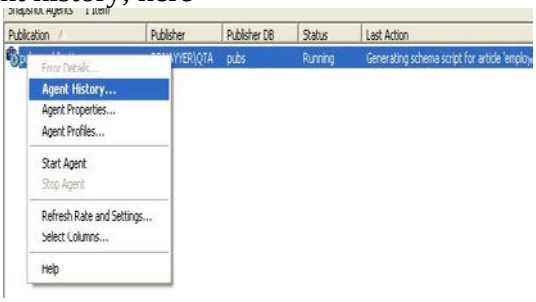


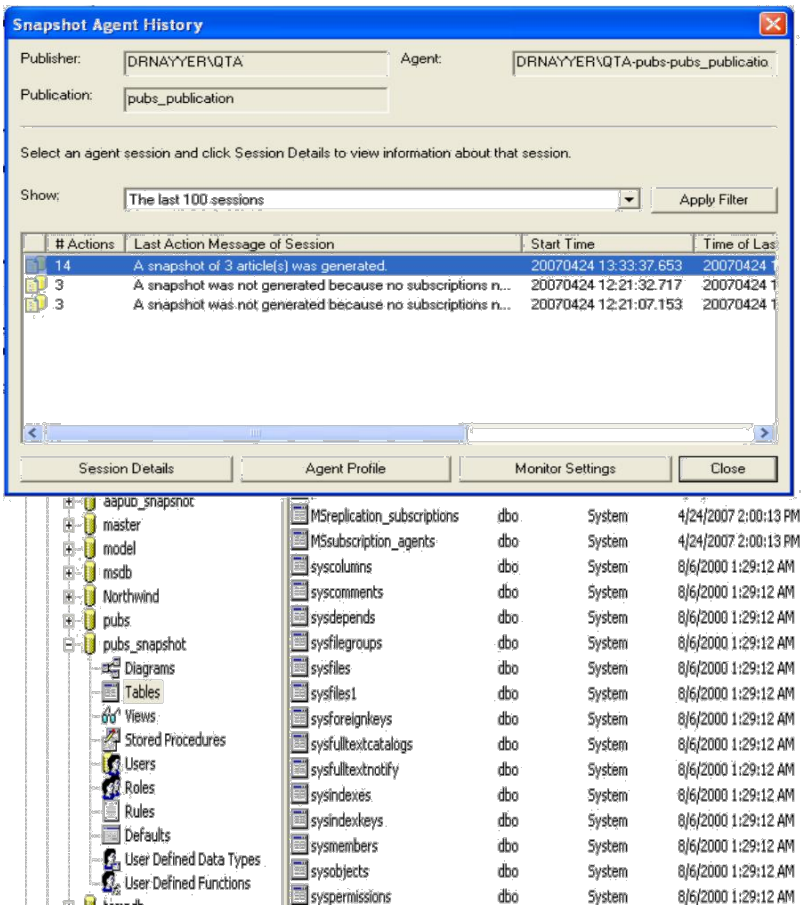
- This activity has created the agents (Snapshot agent and Distributor agent)
- Also the subscription database at the subscriber server (PESH in our case)
- Agents are created, not worked yet, so no snapshot created at the subscriber
- Execute the Snapshot Agent, that will create a snapshot of your publication
- See How?



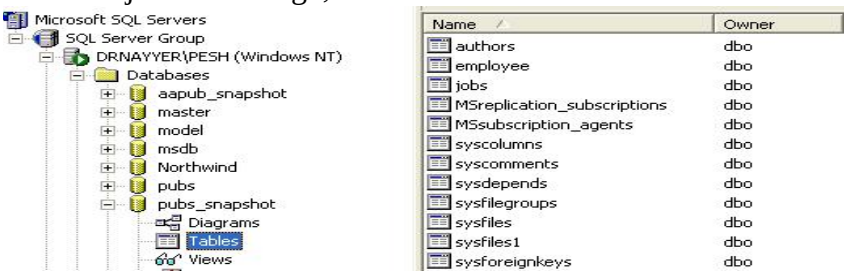
Snapshot is created, you can verify

Verification from the message in the description of the agent, or you can see it from agent history, here

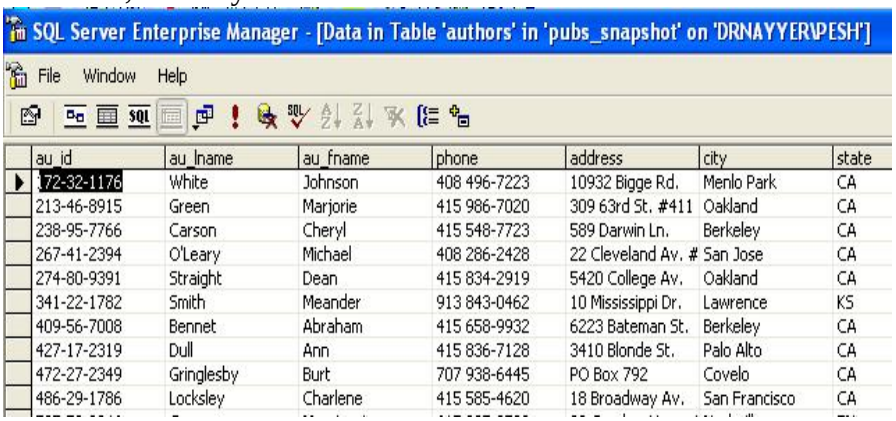




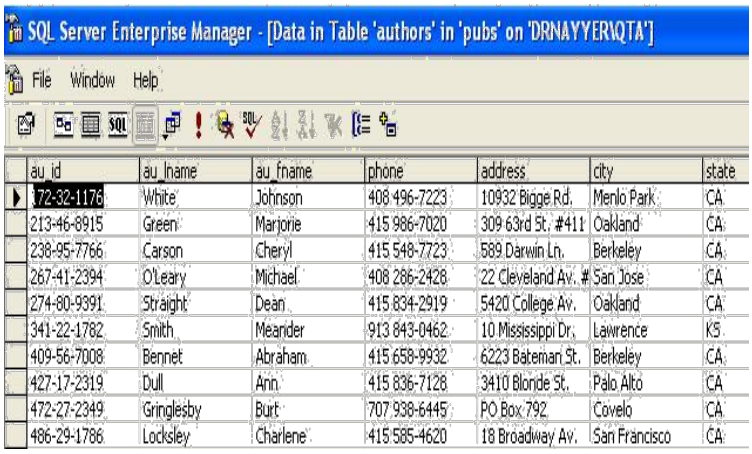
Which is just not enough, Publication has to be there



It is there, not only the tables But the data as well

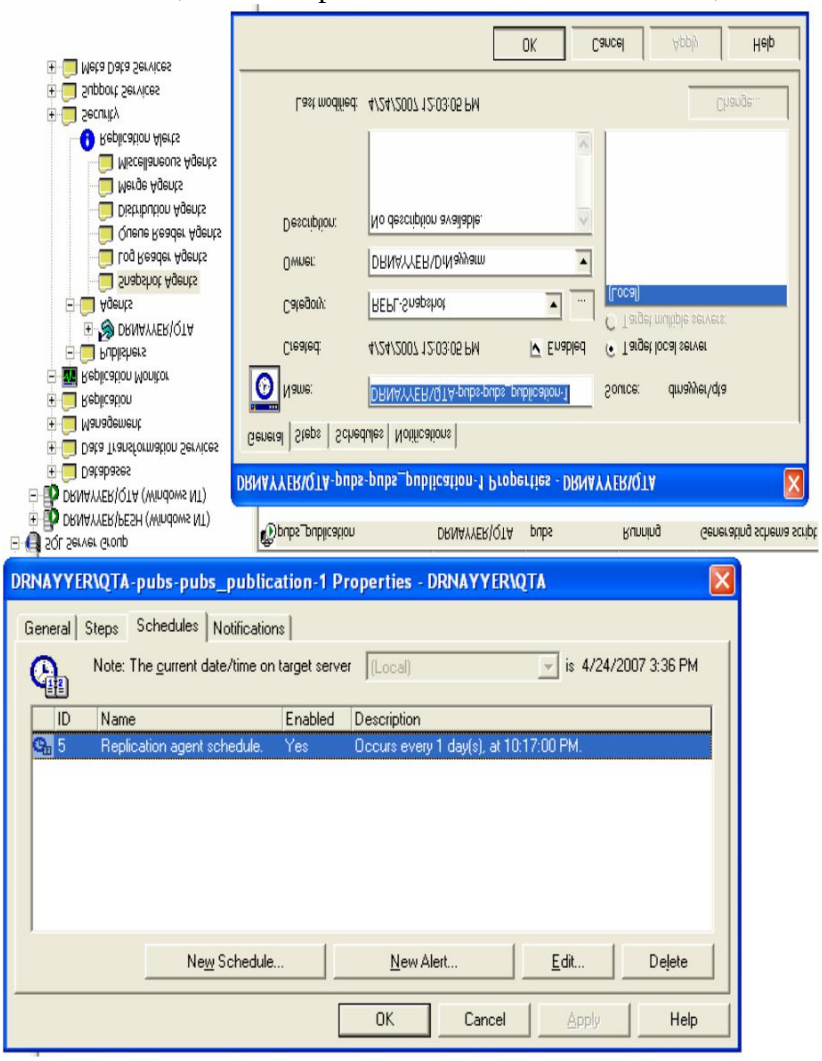


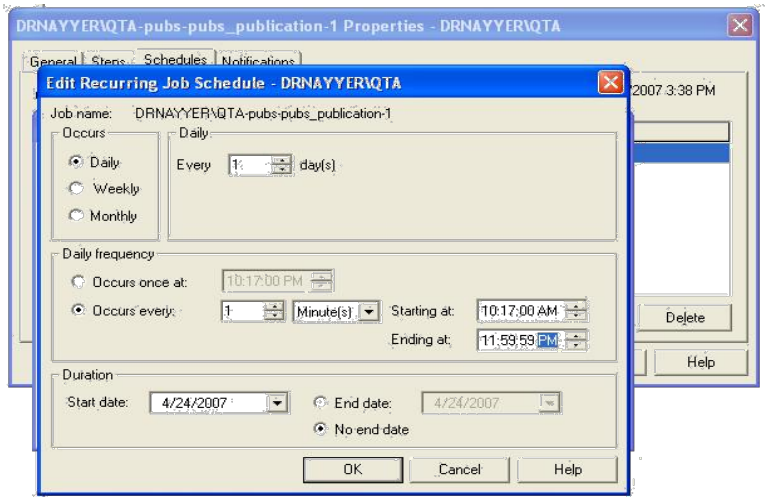
au_id	au_lname	au_fname	phone	address	city	state
172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA
213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA
238-95-7766	Carson	Cheryl	415 548-7723	589 Darwin Ln.	Berkeley	CA
267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #	San Jose	CA
274-80-9391	Straight	Dean	415 834-2919	5420 College Av.	Oakland	CA
341-22-1782	Smith	Meander	913 843-0462	10 Mississippi Dr.	Lawrence	KS
409-56-7008	Bennet	Abraham	415 658-9932	6223 Bateman St.	Berkeley	CA
427-17-2319	Dull	Ann	415 836-7128	3410 Blonde St.	Palo Alto	CA
472-27-2349	Gringlesby	Burt	707 938-6445	PO Box 792	Covelo	CA
486-29-1786	Locksley	Charlene	415 585-4620	18 Broadway Av.	San Francisco	CA



au_id	au_lname	au_fname	phone	address	city	state
172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA
213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA
238-95-7766	Carson	Cheryl	415 548-7723	589 Darwin Ln.	Berkeley	CA
267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #	San Jose	CA
274-80-9391	Straight	Dean	415 834-2919	5420 College Av.	Oakland	CA
341-22-1782	Smith	Meander	913 843-0462	10 Mississippi Dr.	Lawrence	KS
409-56-7008	Bennet	Abraham	415 658-9932	6223 Bateman St.	Berkeley	CA
427-17-2319	Dull	Ann	415 836-7128	3410 Blonde St.	Palo Alto	CA
472-27-2349	Gringlesby	Burt	707 938-6445	PO Box 792	Covelo	CA
486-29-1786	Locksley	Charlene	415 585-4620	18 Broadway Av.	San Francisco	CA

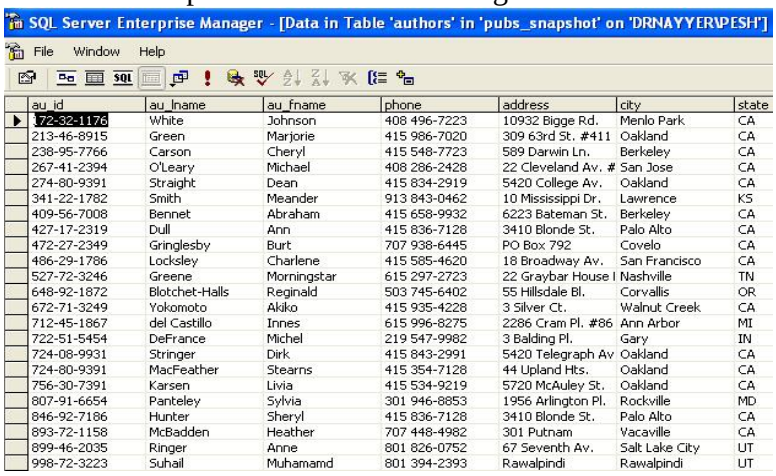
Now the behavior of SSt Replication is that at the predefined intervals, snapshot agent generates the snapshot and distributor agent copies it to the subscriber, let's see the timing issue. Mind it, the SSt replication suits the environment, where higher latency is acceptable





Now add a record in any of the tables in your publication (lets say author) at your publisher (QTA). We selected this table because after adding the record of Muhammad Suhail of Rawalpindi in the table at QTA, wait for one minute.

After one minute, both agents will execute as set in their schedules. They will perform their respective job and as a result the snapshot will be copied at subscriber and the record entered just now will be copied there Proof? From agents' histories and from the subscriber's table it, like



au_id	au_lname	au_fname	phone	address	city	state
172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA
213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA
238-95-7766	Carson	Cheryl	415 548-7723	589 Darwin Ln.	Berkeley	CA
267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #	San Jose	CA
274-80-9391	Straight	Dean	415 834-2919	5420 College Av.	Oakland	CA
341-22-1782	Smith	Meander	913 843-0462	10 Mississippi Dr.	Lawrence	KS
409-56-7008	Bennet	Abraham	415 658-9932	6223 Bateman St.	Berkeley	CA
427-17-2319	Dull	Ann	415 836-7128	3410 Blonde St.	Palo Alto	CA
472-27-2349	Gringlesby	Burt	707 938-6445	PO Box 792	Covelo	CA
486-29-1786	Locksley	Charlene	415 585-4620	18 Broadway Av.	San Francisco	CA
527-72-3246	Greene	Morningstar	615 297-2723	22 Graybar House I	Nashville	TN
648-92-1872	Blotchet-Halls	Reginald	503 745-6402	55 Hillsdale Bl.	Corvallis	OR
672-71-3249	Yokomoto	Akiko	415 935-4228	3 Silver Ct.	Walnut Creek	CA
712-45-1867	del Castillo	Innes	615 996-8275	2286 Cram Pl. #86	Ann Arbor	MI
722-51-5454	DeFrance	Michel	219 547-9982	3 Balding Pl.	Gary	IN
724-08-9931	Stringer	Dirk	415 843-2991	5420 Telegraph Av	Oakland	CA
724-80-9391	MacFeather	Stearns	415 354-7128	44 Upland Hts.	Oakland	CA
756-30-7391	Karsen	Livia	415 534-9219	5720 McAuley St.	Oakland	CA
807-91-6654	Panteley	Sylvia	301 946-8853	1956 Arlington Pl.	Rockville	MD
846-92-7186	Hunter	Sheryl	415 836-7128	3410 Blonde St.	Palo Alto	CA
893-72-1158	McBadden	Heather	707 448-4982	301 Putnam	Vacaville	CA
899-46-2035	Ringer	Anne	801 826-0752	67 Seventh Av.	Salt Lake City	UT
998-72-3223	Suhail	Muhamamd	801 394-2393	Rawalpindi	Rawalpindi	UT

Here we described connecting Oracle Server with SS, and using Oracle ODBS System DSN, we implemented snapshot replication on Oracle,

2- Transactional Replication

- Preferred in higher bandwidth and lower latency
- Transaction is replicated as it is executed
- Preferred in low latency and higher bandwidth
- Begins with a snapshot, changes sent at subscribers as they occur or at timed intervals
- A special Type allows changes at subscribers

3- Merge replication

It lets multiple sites work autonomously with a set of Subscribers, and then later merge the combined work back to the Publisher. During the merge, some conflicts may be found where multiple Subscribers modified the same data. Merge replication supports the definition of conflict resolves, which are sets of rules that define how to resolve such conflicts. Merge

replication is used when it is important for the Subscriber computers to operate autonomously (such as a mobile disconnected user), or when multiple Subscribers must update the same data.

Summary:
We have discussed the types of replication and the Setting up the Replication.

Course Title: **Distributed Database Management Systems**
Course Code: **CS712**
Instructor: **Dr. Nayyer Masood (nayyerm@yahoo.com)**
Lecture No: **23& 24**

In previous lecture:

- Fragmentation

In this lecture:

- Reasons for Fragmentation
    0. Maximizes local access

1. Reduces table size, etc.

- PHF using the SQL Server on same machine
- Implemented PHF in a Banking Environment

Fragmentation

- We know there are different types, we start with the simplest one and that is the PHF
- Supposedly, you have already gone through the design phase and have got the predicates to be used as basis for PHF, just as a reminder
- From the user queries, we first collect the simple predicates and then we form a complete and minimal set of minterm predicates, a minterm predicate is ….., you know that otherwise refer back to lecture 16, 17
- Lets say we go back to our Bank example, and lets say we have decided to place our servers at QTA and PESH so we have two servers where we are going to place our PHFs
- As before we register our servers and now at our Enterprise Manager we can see two instances of SS
- At each of the three sites we define one database named BANK and also one relation, normal base table, however, for the fragmentations to be disjoint (a correctness requirement) we place a check on each table at three sites, how….
- We name our fragments/local tables as custQTA, custPESH
- Each table is defined as

```
create table custPESH(custId char(6), custName varchar(25), custBal
number(10,2), custArea char(5))
```

- In the same way we create 2 tables one at each of our two sites, meant for containing the local users
- Users that fall in that area and the value of the attribute custArea is going to be the area where a customer's branch is, so its domain is {pesh, qta)
- To ensure the disjointness and also to ensure the proper functioning of the system, we apply a check on the tables
- The check is
- Peshawar customers are allocated from the range C00001 to C50000, likewise
- QTA is C50001 to C99999
- So we apply the check on both tables/fragments accordingly, although they can be applied while creating the table, we can also apply them later, like
- Alter table custPesh add constraint chkPsh check ((custId between 'C00001' and 'C50000') and (custArea = 'Pesh'))
- Alter table custQTA add constraint chkQta check ((custId between 'C50001' and 'C99999') and (custArea = 'Qta'))
- Tables have been created on local sites and are ready to be populated, start running applications on them, and data enters the table, and the checks ensure the entry of proper data in each table. Now, the tables are being populated

Example Data in PESH

C0001	Gul Khan	4593.33	Pesh
C0002	Ali Khan	45322.1	Pesh
C0003	Gul Bibi	6544.54	Pesh
C0005	Jan Khan	9849.44	Pesh

Example Data at QTA

C50001	Suhail Gujjar	3593.33	Qta
C50002	Kauser Perveen	3322.1	Qta
C50003	Arif Jat	16544.5	Qta
C50004	Amjad Gul	8889.44	Qta

- Next thing is to create a global view for the global access/queries, for this we have to link the servers with each other, this is required
- You have already registered both the servers, now to link them
- You can link them using Enterprise Manager or alternatively through SQL, we do here using SQL

Connect Pesh using Query Analyzer

- Then execute the stored procedure sp_addlinkedserver
- The syntax is
-

```
sp_addlinkedserver
@server = ‘QTA',
@srvproduct = '',
@provider = 'sqloledb', @datasrc = ‘mysystem\QTA‘
```

- You will get two messages, if successful, like ‘1 row added’ and ‘1 row added’
- You have introduced QTA as a linked server with PESH.
- We have to perform this operation on the other server, that is, we have to add linked server PESH at QTA
- Setup is there, next thing is to create a partitioned view
- In SQL Server, a partitioned view joins horizontally partitioned data across multiple servers
- The statement to create the partitioned view is

Create view custG as select * from custPesh
Union All
select * from QTA.bank.dbo.custQTA

- Likewise, we have to apply same command at QTA
- Create view custG as

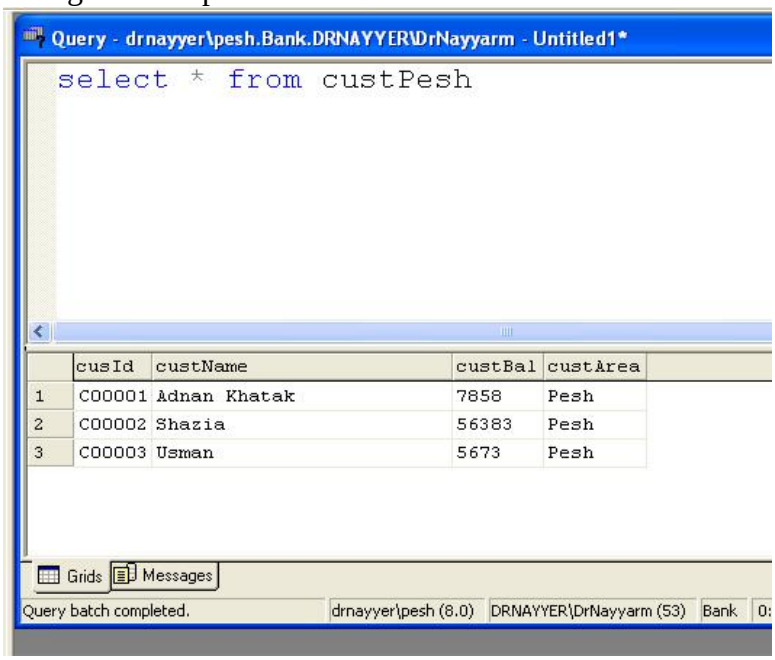
select * from
custQta Union All
select * from PESH.bank.dbo.custPesh

- Once it is defined, now when you access data from custG, it gives you data from all four sites.
- It is also transparent

- Now lets say if you are connected with Pesh, and you give the command

Select * from custPesh

You get the output



- Same is the case with the users of the QTA server, they pose the query
- Select * from custQTA
- Like a local user

```
Select * from custQTA
```

	custId	custName	custBal	custArea
1	C50001	Muneer	4628	QTA
2	C50002	Mudassar	5674	QTA
3	C50003	Saleem	8737	QTA

- The previous two examples represent access of a local user, now if the global users, like from Management, want to access data across all sites, they will pose the query against the global view, like
- Select * from custG

```
Select * from custG
```

	custId	custName	custBal	custArea
1	C50001	Muneer	4628	QTA
2	C50002	Mudassar	5674	QTA
3	C50003	Saleem	8737	QTA
4	C00001	Adnan Khatak	7858	Pesh
5	C00002	Shazia	56383	Pesh
6	C00003	Usman	5673	Pesh

- All this is transparent from the user, that is, the distribution of data
- Global user gets the feeling, as if all the users' data is placed on a single place
- For the administrative purposes they can perform analytical types of queries on this global view, like

Summary:

We have discussed the fragmentation, maximize local access and reduce table size etc. and also discussed the PHF using the SQL Server on same machine.