

sources include a native database; external (remote) databases; host file base; data input, storage, and presentation (output) devices; and even data-generating and data-consuming programs (such as a text processing system).

- 3a) The ability to represent arbitrary data types (including compound documents) and specification of procedures (programs) that **interact with** arbitrary data sources.
-) The ability to query, update, insert and delete multimedia data (**including** retrieval of multimedia **data** via associative **search** within **multimedia** data; minimally, text).
- c) The ability **to specify and execute** abstract operations on multimedia data; for example, to play, fast forward, pause, and rewind such one-dimensional data as audio and **text**; to display, expand and **condense such two-dimensional** data as a bit-mapped image.
- d) The ability **to deal with heterogeneous** data sources in a **uniform** manner; this includes **access** to data in these **sources** and migration of **data from one data** source to another.

1.12 FURTHER READINGS

- 1. Modern Database Systems—the Object Model, Interoperability and Beyond, By **WON KIM**, Addison Wesley, 1995.
- 2. Object-Oriented **DBMS** : Evolution & Performance Issues, **A.R.Hurson & S'imin H. Pakzad**, **IEEE** Computer, Feb. 1993.

UNIT 2

INTRODUCTION TO CLIENT/
SERVER DATABASE

Structure

- 2.0 Introduction
- 2.1 Objectives
- 2.2 Evolution of Client/Server
- 2.3 Emergence of Client/Server Architecture
- 2.4 The Client/Server Computing

2.4.1 Basics of Client/Server Computing Paradigm

2.4.2 Why need Client/Server Computing?

2.4.3 Advantages of Client/Server Computing

2.4.4 Components of Client/Server Computing
- 2.5 The Critical Products

2.5.1 Object Oriented Technology (OOT)

2.5.2 Distributed Computing Environment

2.5.3 Application ProgrammingInterface (API)

2.5.4 Multithreaded Processes

2.5.5 Remote Procedure Calls (RPC)

2.5.6 Dynamic Data Exchange (DDE)

2.5.7 Object Linking and Embedding (OLE)
- 2.6 Developing an Application
- 2.7 Structured Query Language (SQL)

2.7.1 Data Definition Language (DDL)

2.7.2 Data Manipulation Language (DML)
- 2.8 Client/Server : Where to next?
- 2.9 Summary
- 2.10 Model Answers
- 2.11 Further Readings

2.0 INTRODUCTION

The concept behind the Client/Server solution is concurrent, cooperative processing. It is an approach, that presents a single systems view from a user’s viewpoint, involves processing on mulliple, interconnected machines provides coordination of activities in a manner transparent to end users.

This unit is broadly divided into three parts. The first part (sections 2.2 and 2.3) address as the basics of client server computing. The second part (section 2.4) discusses the critical products used in implementing client/server model. The focal point of the last part is to develop an application in client server environment.

2.1 OBJECTIVES

At the end of this course, the reader should be able to understand

- the broad level issues in Client’ Server computing
- the product components of Clicnl' Server architecture
- how to develop application in Client’ Server model
- discuss the possible emerging scenario in Client' Server computing.

2.2 EVOLUTION OF CLIENT/SERVER

Mainframe Scenario

After twenty years of existence starling in the middle of seventies, the computer based application proliferated the business and scientific application throughout the world. The

scenario was dominated by mainframe computers. The development of hardware has always outpaced the development of software yet user requirements did outgrow the capacity of the mainframe computers coupled with the fact that better hardware releases were being launched at rapid succession. The large EDP houses typically opted for the better version of hardware every alternate year. The new model of the hardware would take over the major share of applications rendering the earlier model unutilized or underutilized.

PCs as Environment for Business Computing

In the late seventies first version of PC with 64 KB of main memory were launched, they were typically used to do wordprocessing jobs and spreadsheet calculations. In 1980, IBM launched its 640 KB PC, this is single most important development in the field of computers, which revolutionised the concept of computing profoundly. In 1980 people did not think that the PCs could really become a serious computing environment because of the advancements of technologies in many other related fields. In the decade of 80s PCs grew in power and speed in leaps and bounds. Because of the standard environment and non-proprietary architecture and also because of the very low price tag PCs and software that runs on PC spreaded at a rate which has never been witnessed in field of Information Technology.

Emergence of Open Systems

Till the middle of 70s for over two and half decades proprietary networking solutions dominated the networking scenario. The solutions used to be very expensive, each company used to set its own networking and connectivity standards. Each company believed in giving the complete network, software and networking solution to the end client rendering the solutions extremely high priced and beyond the budget of most Information Technology organisations. Further this did not enable sharing machines from multiple vendors on a network. The advent of non-proprietary standards in network and software product components allowed increasing use of open system. The chip, the peripherals, the architecture, the networking protocols, the operating system even the software components became standard. These developments allowed growth of non-proprietary solution, network solution involving network and software components from multiple vendors. This also enabled usage of downsized environments. PC users grew to upsized environment with Novell network among other software as the servers.

In 80s networking of PCs in local area network (LAN) or connectivity between PCs running between DOS and VAX running VMS and PCs, connectivity between almost all standard machines gave rise to a new way of looking at computing. The R & D labs dealing with software started working on solution which would distribute the computing load on multiple machines on network. Client/Server architecture took birth around these developments. It basically tries to utilize and distribute computing requirements on PCs, UNIX servers, VMS servers, even mainframe depending on the computing requirements.

2.3 EMERGENCE OF CLIENT- SERVER ARCHITECTURE

Some of the pioneering work that was done by some of the relational database vendors allowed the computing to be distributed on multiple computers on network using contemporary technologies involving:

- Low Cost, High Performance PCs and Servers
- Graphical User Interfaces
- Open Systems
- Object-Oriented
- Workgroup Computing
- EDI and E-Mail
- Relational Databases
- Networking and Data Communication

2.4 THE CLIENT/SERVER COMPUTING

In this application we will take up basics of Client/Server model: how to define client and

2.4.1 Basics of Client/Server Computing Paradigm

A Client is an application that initiates peer to peer communication and users usually involve client software when they use a network service. Most client software consists of conventional application programs. Each time a client application executes, it contact a server, sends a request and awaits a response. When the response arrives, the client continues processing. Clients are often easier to build than servers and usually require no, special system privileges to operate.

By comparison, a server is any program that provides services to requesting processes in client. It waits for incoming communication requests from a client. It receives a client's request, perhaps the necessary computation and returns the result to the client.

Generally, it does not send information to the requester until the requesting process tells it to do so. But the server must also manage synchronisation of services as well as communication once a request has been initiated.

The client may initiate a transaction with the server, while normally the server does not initiate a transaction with the client.

The client is therefore the more active partner in this association, requesting specific functions, accepting corresponding results from the server, and acknowledging the completion of services. The client, however, does not manage the synchronization of services and associated communication. Because servers often need to access data, server software usually requires special system privileges. Because a server executes with special system privilege, care must be taken to ensure that it does not inadvertently pass privileges on to the clients that use it. For example, a file server that operates as a privileged program must contain code to check whether a given file can be accessed by a given client, the server cannot rely on the usual operating system because its privileged status overrides them. Servers must contain code that handle the issue of:

- Authentication — Verifying the identity of a client
- Authorisation — Determining whether a given client is permitted to access the service by the server supplies.
- Data Security — Generating that data is not unintentionally revealed or compromised.
- Privacy — Keeping information about an individual from unauthorised access.
- Protection — Guaranteeing that network application cannot abuse system resources.

The general case of client/server implementation is shown in figure.

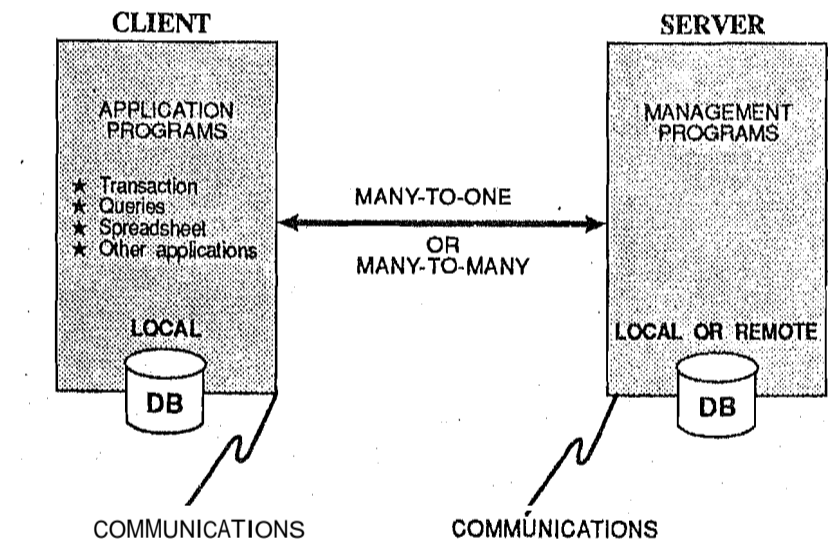


Figure 1 : The general case of client/server implementation

With the client/server computing model of distributed operations

- Many clients may share one server.
- The same client may access many servers, both logical and remote.

The client, the server, or both can be a workstation. The server can also be a supermicro, a database computer, or a supercomputer. Purposeful old minis and obsolete mainframes have not been included in the definition.

Since server-based supports can be complex, the term server does not necessarily refer to a piece of hardware, a database unit, a gateway, or a special-purpose processor dedicated to run software. The concept is much broader: Server requires both software and hardware for a range of functions—though typically each server is specialised.

The technical solutions should be able to assure networkwide shared access by applications processes that address databases, number printers, gateways, and other resources. The environment can be:

- Simple, such as a small work group sharing applications and peripherals
- or
- Complex, resulting from wider sharing across multiple systems and topologies

Whether the solution we are after is simple or complex, totally new or a conversion of mainframe applications, a basic design principle is never to build a system to support the current organisational divisions and their departments. A great deal of the necessary flexibility can be provided by solutions that are modular and independent of current structures.

A different way of making this statement is that the information technology (IT) solutions we develop must be organisation-independent:

- They should not reflect the current organisation chart.
- They should be immune to structural changes—hence flexible.
- They should integrate with existing resources, assuring the end user of seamless access to them.

The goal of the solutions through client/server computing is to enable a client program anywhere in a network to request services from anywhere else in the network in a way that is both transparent and independent of any particular interconnected software and hardware. This reference raises a number of questions:

- What computers, communications, and software solutions should be used to create a client/server network?
- What range of functionality should be targeted to optimise cost-effectiveness?
- How can the system be kept flexible to develop new applications, add new users, and enhance its own structure?
- How can we ensure that server platforms and client stations will deliver the high availability demanded by the most competitive applications?
- What facilities will be able to manage all the nodes and links in the client/server network, distributing the software and controlling the physical assets?

Not all approaches that represent themselves as client/server models can answer these questions in an able manner. Without any doubt, exceptions should be made of mainframes and nonintelligent terminals connected to mainframes or minis.

Another major exception is the stand-alone workstation. Under no stretch of the imagination can it qualify as a client/server model, although some vendors try to sell it as such. A PC or any other unit that is not networked is not a workstation.

Integrating what has been said so far, we are converging toward definition that client/servers are excellent multiuser systems with a flexible but all defined applications perspective. These applications must be designed to work together through adherence to rules.

The foregoing concepts are not necessarily new. To a substantial extent, they have existed for four decades in computing. What is new is the truly peer-to-peer structure of the implementation environment.

The wider acceptance of the outlined solutions largely depends on the functionality provided by the system as a whole. What makes the client/server architecture distinct from mainframe-based processes are its distributed but cooperative applications characteristics:

- Clients and servers function across platforms within the network, whether in a local or in a wide area.
- Distributed software artifacts execute on multiple platforms within the supported architecture.
- Processes on the network can be dynamically distributed to the most appropriate (and available) platform for execution.

Graphics applications can be assisted through graphics processors. A numerically intensive process within an application can be migrated from a client to the network's number cruncher server, and a complex database query may access a different database server if the information elements it requires are themselves distributed. This emphasises the need for first-class solutions in networking.

2.4.2 Why need Client/Server Computing?

Client/Server (C/S) architecture involves running the application on multiple machines in which each machine with its component software handles only a part of the job. Client machine is basically a PC or a workstation that provides presentation services and the appropriate computing, connectivity and interfaces while the server machine provides database services, connectivity and computing services to multiple users. Both client machines and server machines are connected to the same network. As the users grow more client machines can be added to the network while as the load on the database machine increases more servers can be connected to the network. The client could be character terminals or GUI PCs or workstations connected to the network. Server machines are slightly more heavy duty machines which gives database services to the client requests.

The network need not be Local Area Network (LAN) only, it can be on much wider distributed Wide Area Network (WAN) across multiple cities. The client and server machines communicate through standard application program interfaces (API) and remote procedure calls (RPC). The language through which RDBMS based C/S environment communicate is known as structured query language (SQL).

2.4.3 Advantages of Client/Server Computing

C/S computing caters to low cost and user friendly environment. It can be used to develop highly complex multiuser database application being handled by any mainframe computers until about 5 years back. It offers expandability. It ensures that the performance degradation is not so much with increased load. It allows connectivity with the heterogeneous machines and also with real time data feeders like ATMs, Numerical machines. It allows the database management including security, performance, backup, server enforced integrity to be part of the database machine avoiding the requirement to write large number of redundant piece of code dealing with database field validation and referential integrity. Since PCs can be used as clients, the application can be connected to the spreadsheets and other applications through Dynamic Data Exchange (DDE) and Object Linking and Embedding (OLE). If the load on database machine grows, the same application can be run slightly upgraded machine like disk machine provided it offers the same version of RDBMSs on diverse machines, legacy applications on old machines or geographically separated can meet all requirements of an enterprise.

2.4.4 Components of Client/Server Computing

The Server

The server machines could be running NOVELL LAN or INTEL based server or UNIX from SCO or AT&T or UNIX being run on RISC machines like HP, SUN Microsystems, IBM, Compaq etc.

These server machines should be running on RDBMS engine like Sybase, Oracle, Informix etc. The server machine takes care of data storage, backup, recovery and database services. It is typically a multiuser machine catering to large number of requests submitted from the client machine or executing requests for RPCs/Stored procedures. The database engine executes the requests and sends the result to the client machine and the presentation service of the client machine puts the received data in required format. Some of the databases take care of the file handling, the task and user handling themselves. The server also allows certain constraints at table level or field level to be incorporated. The field level validations are generally called rules e.g. if employee code is 4 chars, all numeric starting with digit other than 0. In employee table this constraint can be attached to the field itself. It would eliminate writing a code for field validation on this field in each table. If the existence of the employee code is to be checked before entering employee pay details for a month, it would involve two tables : employee pay detail and employee master. These types of checks are called referential integrity constraints. If these constraints can be incorporated in the database, then we can reduce large number of application code. More than that, it will be ensured that application errors do not effect the integrity/reliability of the data stored in the database.

These types of integrity checks are called **Database Triggers**. The advanced RDBMSs also allow on-line database backup, schema modification and performance and tuning. Database stored procedures are certain more repetitively executed pieces of code stored in the database itself, written in the extended form of SQL called T-SQL in Sybase. PL/SQL in Oracle 7. These fast and compiled server resident procedures improve performance by reducing network traffic and by allowing a single copy of the procedure to large number of users. Stored procedures can be executed by client machine. Remote procedure calls on the contrary are generally invoked by servers which enables distributed database processing when the information is available on multiple servers. RPCs can handle the situation very efficiently, In the earlier versions of RDBMSs process for users design was followed, so number of processes on a machine would be directly proportional to the number of users currently using the RDEMS. This resulted in tremendous degradation of performance as number of users grew in number. The reason being, after sometime the user processes go into swap. Sybase pioneered the multithreaded RDBMS design in which, irrespective of the number of users taking database services, the database process code just be one. Today, most of the important RDBMSs provide for multithreaded server design. The multithreaded architecture combined with server integrity, Client Server Architecture, connectivity to networking protocol, connectivity to heterogeneous databases has resulted in major movement towards enterprise wide computing.

2.5 THE CRITICAL PRODUCTS

In this section we will briefly look at some tools to implement client/server environment.

2.5.1 Object Oriented Technology (OOT)

The fundamental ideas underlying OOT are:

- Abstraction
- Objects
- Encapsulation
- Classes and Instance
- Inheritance
- Message
- Methods

How OOT differs from structured programming?

Structured Programming

Data and code are separate and code operates on data.

OOT

Data and procedures are together and the object responds to messages.

It is the act of removing certain distinctions between objects so that we can see commonalties. The result of an abstraction process is a concept or object type. One of the forms of abstraction is Data Abstraction. Here only the selected properties of the object are made visible to the outside world and their internal representation are hidden. The object model has a grater advantage over conventional languages that the lower level implementation details are not visible.

Object

An object is any thing, real or abstract, about which we store data and those methods that manipulate the data. Its an encapsulated abstraction that includes state information and a clearly defined set of access protocol (messages to which object responds). It is a software package which contains related data and procedures. The basic concept is to define software objects that can intersect with each other just as their real-world counterparts do.

An object type is a category of object. An object is an instance of an object type.

Encapsulation

Packaging data and methods together, is called Encapsulation.

Its advantages are:

- Unnecessary details are hidden,
- Unintentional data modification is avoided i.e. provides security and reliability.
- . Presents interfcrence with the internals and also hides the complexity of he components. Thus, encaysuladon is important because it separates how an object behaves, from how it is implemented.

Classes

A class is an implementation of an object type and is defined by and is defined by a class description that defines both the attributes and messages for an object that class. It specifies a data structure and the permissible operational methods that apply to each of its objects. Classes can also be objects in some object oriented language.

An object is an instance of a class. The properties of any instance (objcct) are given by the class description of its class. Thus,

- Class is template that helps us to create objects.
- Classes have names (class identifier) that indicates the kind of objects they represent.
- Classes may be arranged in hierarchy with subclass representing more specific kinds of objects than their super class.

Inheritance

Inheritance allows the developer to create a new class for object from an existing one by inheriting h e behaviour and then modifying or adding to it. It provides an ability to create classes that will automatically model themselves on other classes. Sometimes a class inherits properties of more than one superclass, then its called MULTIPLE INHERITANCE. This inheritance leads to a "Class Hierarchy". It is a network of classes that starts with the most general as the uppermost branches and descends to the bottom leaves which arc most specific. The power of an Object Oriented environment is defined by the Clnss Hierarchy and its capabilities.

Its advantages are:

Reusability of code

- Avoid duplication of code
- Reduce cost of maintenance

Message

A message is

- A specific symbol, identifier or **key-word(s)** with or without parameters **that** represents an action to be taken by an object.
- Only way to communicate **with** an object is through message passing.
- An **object** knows what another intrinsic **property/capability** object has, but not **how** it does it
- A message is not restricted to **one recipient** object but to multiple **object**.

In a conventional programming language an operation is invoked by calling a **named procedure** and supplying with it the data to act on. If the wrong **data** are supplied the **wrong** results will be returned.

Methods

They are often called SELECTORS since when they are called by name they allow the system to select which code is to be executed.

- **Methods** are description of **operations**.
- Method appears as a component of **object**.
- There is a 1-1 correspondence **between** messages and methods that are executed when a **message** is received by a given object.
- **The** same **message** might result in **different** method.

These concepts have been also explained in **the** previous unit,

2.5.2 Distributed Computing Environment

Distributing computing environment **integrates** computers in geographically distant location and underlying applications. These computers could be of **different** types with different operating systems, **even the RDBMSs**. It can also utilize heterogeneous client environment like Powerbuilder, VisualBasic, Uniface etc.

Issues involve:

- Networking multiple machines.
- **Integrating RDBMS** applications using global dictionary or two-phase **commit (2PC)** or replicated **server/table**.

Global Dictionary

The data dictionary of these geographically separate databases is **stored centrally** to **take** action on **distributed** transactions. This caters to location transparency e.g. if the salary of **all** the people in a corporation belonging to **grade G2** is higher by **25%**, the transaction **statement need not specify** the location name where **G2's** data has to be changed. A statement similar to **increased** salary of G2 by 25% globally will do.

Two Phase Commit (2PC)

In **2PC**, the **commit** server in phase **one** checks out the availability of participating servers in all the locations. In **phase two**, after sending **the** transactions to the individual participating servers and receiving OK from them on updation, the commit server END **COMMIT** status to all **the** participating **servers**.

Replicated Server/Table

In this method, **the tables required** for **distributed** transaction **are copied/replicated** to **all** participating server sites. The replication **server ensures** that when there is a change in **the replicated** table, the change transmitted to **all** the locations. This enables reduced **network traffic** and lays prone to application storage owing to network failure.

The communication between **heterogeneous database** is done through database gateways.

2.5.3 Application Programming Interface (API)

It is simply a specification of a set of functions that allow client and server processes to communicate. It hides the underlying platform hardware and software from the developer. APIs show the developer a single-system image across a heterogeneous network of processors. e.g. Open Database Connectivity (ODBC). The primary advantage of developing the client application using a standard API is that the resulting application can use any back-end database server rather than just a specific server. The primary disadvantage is that they generally include the least common denominator of all tools and database servers that support the standard. So consider the use of API only if two or more databases servers are used.

2.5.4 Multithreaded Processes

A single process can have multiple threads of execution. This simply means that a single multithreaded process can do the work of multiple single-threaded processes. The advantage of a multithreaded process is that it can do the work of many single-threaded processes but requires far less system overheads. If a database server uses multithreaded server processes, it can support large number of clients with minimal system overheads. The user processes and server processes are different and one server process can serve multiple user processes. This configuration is called Multithreaded Architecture.

2.5.5 Remote Procedure Calls (RPC)

With RPC one component communicate with a remote component using simple procedure calls. This involves peer to peer messaging. If an application issues a functional request and this request is embedded in an RPC, the requested function can be located anywhere in the enterprise, the caller is authorised to access. The advantage of this process to process communication is evident when processors are involved in any simultaneous processes. New client applications that use object-level relationships between processes provide need for this type of communication e.g. a client requests information from a server by connecting to the server, making the request by calling low-level procedure native to the database server, and then disconnecting. To respond to a request, the server connects to the application and calls a low-level procedure in the application.

RPC is typically transparent to a user, making it very easy to use. The RPC provides facility for the invocation and execution of requests from processors running different operating systems and using different hardware platforms from the callers.

2.5.6 Dynamic Data Exchange (DDE)

Through a set of APIs, windows provide calls that support to the DDE protocol for message-based exchange of data among applications. DDE can be used to construct HOT-LINKS between applications where data can be fed from window to window without operation intervention. DDE support WARM-LINKS that we can create so the server application notifies the client that the data has changed and client can issue an explicit request to receive it. We create REQUEST-LINKS to allow direct copy and paste operation between a server and client without the need for an immediate clipboard. No notification of change in data by the server application is provided. EXECUTIVE LINKS cause the execution of one application to be controlled by the another. This provides an easy to use batch processing capability. Thus, using DDE, applications can share data, execute commands remotely and check error conditions.

2.5.7 Object Linking and Embedding (OLE)

Linking is one way of attaching information from one application to another. Link can be locked, broken or reconnected. Linked information is stored in source application.

Embedding makes the information to be embedded, a part of the destination document, thus increases its size.

OLE is designed to let users focus on data rather than on the software required to manipulate the data. A document becomes a collection of objects, rather than a file. Applications that are OLE-capable provide an API that passes the description of the object to any other application that requests the object. OLE is perhaps the most powerful way to share information between documents. In order to link an object created in another application to another file, but applications need to be running in the same environment i.e. either DDE or OLE.

- We can **display** an embedded or linked object **as** an icon instead of its full size.
- We can convert an embedded or linked to a different application.

2.6 DEVELOPING AN APPLICATION

C/S application **developments requires** broadly dividing the application into two categories:

Server Coding

Client Coding

Server Coding :

Creation of Database

It has two major phases—the design phase and the creation phase. The design phase includes planning file limits, **size** and location of the initial data **files**, size and location of **the new database** transaction log groups and members, **determining** the character set to store database data. Once we have planned a new database we can execute it using the SQL commands.

Creation **of Tables**

Once database is created, the tables **to** be kept under this database are designed. The table is comprised of columns and the **properties** of these columns are decided i.e. whether the **field** is null or not null, which is **the primary/foreign key etc.**

Creation of Database Triggers

Triggers ensure that when a specific action is performed, related actions are **performed**. It also ensures **that centralised**, global operations should be fired **for the** triggering statement, regardless of which **user** or database application **issues** the statement. By **default**, **triggers** are **automatically** enabled when they are created. **A pre-defined** or user-defined error conditions or exception may be raised during execution of a trigger, if so, all effects of the trigger execution **are** rolled back, unless the exception is specifically handled.

Creation of Stored **Procedures**

A stored procedure is a schema object that logically groups a **set** of SQL and PL/SQL programming language in **Oracle** and **T-SQL statements** in **Sybase** together to perform a **specific task** these are created in a **user schema** and stored in a **database** for **continued** use. These can be **invoked** by calling explicitly in the code of a **database application**, by **another procedure**, or function or by a trigger. It is defined to complete a **single, focussed** task. It should not duplicate functionality provided by another feature of the server e.g. defining **procedures** to enforce data integrity rules **that** may be **enforced** using integrity **constraints**.

Creation of Server Enforced Validation Checks

This is **done** through database **integrity** rules. It **guarantees** that the data in a database adheres to a **predelined** set of constraints. Its a **rule** for a column of a table which **prevents** invalid data-entry into the **tables** of a database. It is stored in a data dictionary. It supports entity integrity and **referential** integrity. Rules **depend** on type of data and condition specified at time of access of **data** and **frequently** accessed as a check on transaction and not as a constraint **on** the database.

Creation of Indexes

Indexes are used to **provide** quick access to rows in a **table**. It provide faster access **to data** for **operations** that return a small portion of the rows of a table. Indexes should be created **after** loading **data** in a table. **Index** those columns which re-used for joins to improve **performance** on joins of multiple tables.

Creation of Views

Views are **created** to see **the** same data **that** is in **database** tables, but with a different **perspective**. A view is a **virtual table**, deriving its **data** from base tables. Views are used to limit access to specific table columns and create value-based **security** by defining a view for

a specific rows. Views can also be used to derive other columns not present in any table e.g. calculated field.

2.7 STRUCTURED QUERY LANGUAGE (SQL)

It has emerged as the standard for query language for relational DBMSs. Its original version was called SEQUEL. It is still pronounced as SEQUEL. SQL is both the data definition and data manipulation language of a number of relational database systems e.g. Oracle, Ingres, Sybase, Informix etc.

Note : In this discussion, we would be taking examples for Hotel database having two tables:

Employee (Emp_no, Name, skill, Pay-rate)

Duty_allocation(Posting_no, Emp_no, Day, Shift)

2.7.1 Data Definition Language (DDL)

Data definition in SQL is via the create statement.

Create A Table:

Syntax:

Create table < relation (attribute list)

< attribute list > = < attribute name > (< data type >)
[< attribute list >]

< data type > = < integer > | < smallint > | < char(n) > | < varchar > | < float > | < decimal >
(p [, q]) >

Note: In above syntax, relation means table (i.e. file), while attribute means fields or columns. In addition, some data types may be implementation dependent.

For example, the employee relation for the Hotel database can be created as:

```
create table Employee
(Emp_no integer not null,
Name char (25),
Skill char (20),
Pay-rate decimal (10,2))
```

Alter Table

The definition of an existing relation can be altered by using the alter statement. It allows the new column to be added. The physical alteration occurs only during an update of the record.

Syntax:

```
alter table existing-table-name
add column-name data type{...}
```

For example, to add phone-number attribute to the employee relation alter table Employee
add phone-number decimal(10)

Create Index

It allows the creation of an index for an already existing table.

Syntax:

```
create[unique] index name-of-index
on existing-table-name
(column_name[ascending or descending]
[,column_name[order],...]) [cluster]
```

Cluster option is used to indicate that the records are to be placed in physical proximity to each other.

For example, create an index (named empindex) on Employee relation using columns: Name and Pay_rate

```
Create index empindex
on Employee (Name asc, Pay_rate desc)
```

Drop **Table/Index**

It is used to delete **relation/index** from the database.

Syntax:

```
drop table existing-table-name
drop index existing_index_name
```

2.7.2 Data Manipulation Language (DML)

Select Statement

It is the only **data** retrieval statement in SQL. It is **based** on **relational** calculus and entails selection, join and projection.

Syntax :

```
select [distinct/unique] <target list>
from <relation list>
[where <predicate>]
[order by attribute_name desc/asc]
[group by attribute-name]
[having value-expression]
```

For example, find the **values** for **attribute** Name in the employee **relation**.

```
select Name
from Employee
```

For example, get Duty-allocation details in ascending order of Day for **Emp_no** 123461 for **the month** of April 1986 as well as for all employees for shift 3 regardless of dates.

```
select *
from Duty-allocation
where (Emp_no = 123461 and Day 19860401 and
      Day = 19860430) or shift = 3)
order by Day asc
```

Update Statement

Syntax :

```
update <relation> set <target_value_list>
[where <predicate> ]
<target-value-list> = <attribute name> =
    <value exp> [, <target-value-list>
```

For example, change Pay-rate to 8 of the employee Ron in the Employee relation.

```
update Employee
set Pay_rate = 8
where Name = 'Ron'
```

Delete Statement

It deletes one or more records in the relation.

Syntax :

```
delete < relation >

[where < predicate >]
```

If where clause is left out, all **the** tuples in the relation are **deleted**. In this case, the relation **is** still known to the database **although** it is an empty relation.

For example, delete **the** tuple for employee Ron in the Employee relation.

delete < relation >
[where < predicate >]

Insert Statement

It is used to insert new tuples in a specified relation.

Syntax :
insert into < relation > (< target list >)
value (< values list >)
< value list > = < value expression > [, < target list >]

We can replace value clause by select statement.

e.g. Inset a tuple for the employee Ron.
 insert into Employee
 values (123456, 'Ron', 'waiter', 8)

Condition Specification

SQL supports the following Boolean and comparison operators: and, or, not, =, < >, >=, <=, <=, like. If more than one of the Boolean operators appear together, not has the highest priority while or has the lowest. Parentheses may be used to indicate the desired order of evaluation.

Arithmetic and Aggregate Operators

- Avg
- Min
- Max
- Sum
- Count

For example, find the average pay rate for employee working as a chef.

```
select avg (Pay_rate)
from Employee
where skill = 'chef'
```

For example, get the number of distinct pay rates from the Employee relation.

```
select count (distinct Pay_rate)
from Employee
```

For example, get minimum and maximum pay-rates.

```
select min pay-rate), max(Pay_rate)
from employee
```

Join

SQL does not have a direct representation of the JOIN operator. However, the type of join can be specified by an appropriate predicate in the where clause of the select statement.

For example, retrieve the shift details for employee RON.

```
select Posting_No, Day, Shift
from Duty-allocation, Employee
where Duty-allocation. Emp_No
= Employee.Emp_No
and
Name = 'Ron'
```

SQL uses the concept of tuple variables from relational calculus. In SQL a tuple variable is defined in the from clause of the select statement.

For example, get employees whose rate of pay is more than or equal to the pay of employee Pierre.

```
select e1.Name, e2.Pay-rate
from Employee e1, Employee e2
where e1.Pay-rate > e2.Pay
and
e2.Name = 'Pierre'
```

Set Manipulation

SQL provides following **set** of operators:

Any

In

Exists

Not exists

Union

Minus

Intersects

Contains

When using these operators, remember that the statement '**select...**' returns a set of tuples.

Any

It allows the testing of a value against a set of values.

For example, get the names and pay rates of employees with employee number less than 123469 whose rate of pay is more than the rate a pay of at least one employee with employee-No \geq 123460.

```
select Name, Pay-rate
from Employee
where Emp-No 123480
and
Pay-rate > any (select Pay-rate
from Employee
when: Employee  $\geq$  123460
```

In

Its equivalent to $=$ any.

For example, get employees who are working either on the date **19860419** or **19860420**

```
select Emp_No
from Duty-allocation
where Day in (19860419, 19860420)
```

Contains

It is **used** to test for the containment of one set in another

For example, find the names of **all** the employees who are assigned to all the positions that require chef's skill.

```
select e.Name
from employee e
where
(select Posting-no
from Duty_allocation d
where e.Emp_no = d.Emp_no
contains
(select p.Posting_no
from Position p
where p.skill = 'chef')
```

* Position is another relation of Hotel database:

Position (**Posting_no**, skill)

All

For example, find the employees with the lower pay-rate

```
select Emp-No, Name, Pay-rate
from Employee
where Pay-rate <= all
(select*
```

pay-rate from Employee)

Not In

It is equivalent to # all

Not Contain

It is complement of contains

Exists

```
exists (select x from ...)
```

It evaluates to true if and only if the result of "select x from ..." is not empty,

For example, find the names and pay-rate of all the employees who are allocated a duty.

```
select Name, Pay-rate
from Employee
where exists
(select *
from Duty-allocation
where Employee.Emp_no =
Duty_allocation.Emp_no
```

Not Exists

It is complement of exists,

For example, find the name of pay rate of all the employees who are not allocated a duty.

```
select Name, Pay-rate
from Employee
where not exists
(select *
from Duty-allocation
where Employee.Emp_no =
Duty_allocation.Emp_no)
```

Union

Duplicates are removed from the result of a union.

For example, get employees who are waiters or working at posting-no 321.

```
select Emp-No
from Employee
where skill = 'waiter'
union
(select Emp-No
from Duty-ullocation
where Posting-No = 321)
```

Minus

For example, get a list of employees not assigned a duty.

```
select Emp-No
from Employee
minus
(select Emp-No
from Duty-allocation)
```

Intersect

For example, get a list of names of employees with the skill of chef who are assigned a duty.

```
select Name
from Employee
where Emp_no in
((select Emp-No
  from Employee
   where skill = 'chef'
 intersect
 (select Emp-No
  from Duty-allocation))
```

Categorization

Sometimes we need to group the tuples with some common property and perform some group operations on them. For this, we use group by and having options in where clause.

For example, get a count of different employees on each shift.

```
select shift, count(distinct Emp-No)
from Duty-allocation
group by shift
```

For example, get a count of different employees on each shift.

```
select shift, count(distinct Emp-No)
from Duty-allocation
group by shift
```

For example, get employee number of all employees working on at least two dates.

```
select Emp-No
from Duty-allocation
group by Emp-No
having count(*) > 1
```

View

Conceptual or physical relations are called base relations. Any relation that is no part of the physical database i.e. a virtual relation, is made available to the users as a view. A view can be defined using a query expression,

```
create view <view name>
as <query expression>
```

For example, create a view named Emp_view containing the fields Emp-No and Name from Employee relation.

```
create view Emp_view
(select Emp-No, Name
 from Employee)
```

DROP VIEW

Drop view view-name

2.8 CLIENT/SERVER: WHERE TO NEXT?

Client/Server Computing has a great future ahead. The successful organizations have to be market driven and competitive in the times to come, and they will use Client/Server Computing as the enabling technology to add values to their business.

In future cheap and powerful workstations will be available to all end users to be used as clients to access the information on the servers which are distributed globally, The future Client/Server Information System will provide the information from data in its original form e.g. image, video, graphics, documents, spreadsheets etc. without the need to be specific about the software used to store and process each of these.

The future trends in networking show that there is going to be an explosion in the number network users and more than 70% users, and obviously most of them will use Client/Server as the underlying technology. The networks of the future will support much higher bandwidth (of the order of 100 Mbps) by using the technologies like corporate networks will cut across the boundaries of cities or even countries and they will be connected to major networks around the world. An organization living in isolation will not survive in future.

The future Client/Server Information Systems will use the object oriented programming— OOP. Techniques to make zero defect applications. The OOPs will provide the capability to reuse previously tested components. The reuse of already tested components is quite common in most engineering and manufacturing applications (or even the hardware design), the OOPs makes it for the software development too.

The future Client/Server will cover the systems such as Expert Systems, Geographic Information Systems, Point-of- Services, Imaging, Text Retrieval, Document Management Systems or Electronic Filing Systems. Executive Information Systems, Decision Support Systems etc., alongwith the data handling in OLTP (On Line Transaction Processing) and real time environments.

Check Your Progress

1. Discuss trade-off between mainframe and Client/Server environment.
.....
.....
.....
.....

2.9 SUMMARY

The first major challenge for business today is staying competitive in a changing liberalized global economy. Success, even survival, depends on how quickly and accurately one can get up-to-date information so that major business decision can be taken without any delay.

The availability of a vast amount of knowledge and information needs integration of computer and communication systems.

There is a paradigm shift today in using technology for finding solutions/information from the earlier days.

- In the 1960s by centralized mainframes
- In the 1970s by minicomputers, as distributed data processing
- In the 1980s by the personal computers (PCs) and Local Area Networks (LANs)
- In the 1990s by Client/Server architecture which use as server product ranging from UNIX and NT box's to Database Computers (DBC's) and supercomputers.

Companies that have moved out of this mainframe system to Client/Server architecture have found three major advantages:

- Client/Server technology is more flexible and responsive to user needs
- A significant reduction in data processing costs
- An increase in business compctidvcness as the market edge turns towards merchandising

2.10 MODEL ANSWERS

1. In a mainframe all operations take place on one system. This type of environment is being used for the last 30 years, but it is:

- Highly costly
- Highly inflexible
- Quite slow because of contention
- Less reliable

By contrast to this monolithic approach **Client/Server** (shown in figure 2) provides a low-priced robust solution to user requirements. This approach permits downsizing production subsystem while allowing the clients and servers the necessary tools and facilities to control, manage and tune the environment in which they operate.

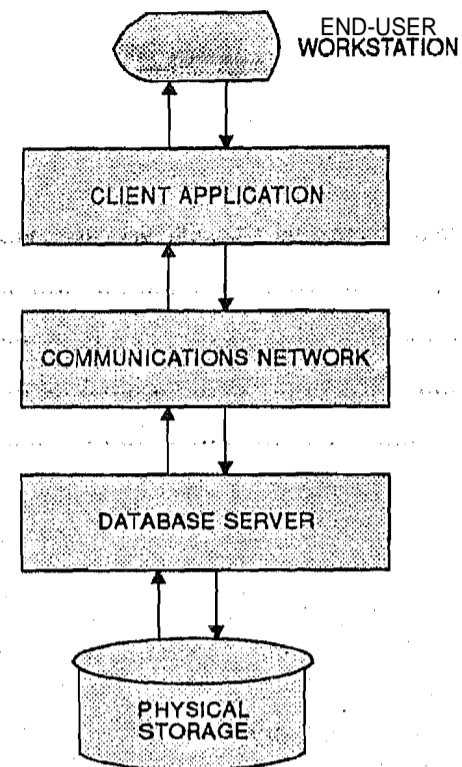


Figure 2 : A layered approach to computing enhances Functionality and increases flexibility, speed and reliability at low cost.

Most **Client/Server** solution are also very **attentive** in matters of security. Access to any resource can be defined to the file level, with such access being controlled through identification and **authorisation**. Logically **defined closed** use groups can be setup to enable the enhancing of **security** measures by **network** administrators.

2.11 FURTHER READINGS

1. Developing **Client/Server** Applications by **W.H.Inmon**
2. Guide to **Client/Server** Database by **Joe Salemi**
3. Mastering Oracle and **Client/Server** Computing by **Steven M.Bobrowski**
4. BSG **Client/Server** Computing by **SHL System House**
5. Featuring SQL standard by **Hayden Book**
6. **Datapro** report on **Client/Server** Computing by Emerging Trends, Solutions and Strategies.
7. **Datapro** report on **Client/Server** Challenge by FDDI or **Ethernet** Switches
8. **Datapro** report on The Hidden Cost of **Client/Server** Computing

9. Client/Server Computing by Patrick Smith and Steve Guengerich

10. GUI based Design and Development for Client/Server Applications using Power Builder, SQL Windows, Visual Basic, PARTS Workbench by Jonathan S.Sayles, Steve Karlen, Peter Molchan and Gary Bilodeau.

11. Beyond LANS Client/Server Computing By Dimitris N. Chorafas; McGraw-Hill Series on Computer Communication;1994.

Introduction to Client/
Sewer Database