|  |  |
| --- | --- |
| **SR.NO** | **NAME OF EXPERIMENT** |
| 1 | Develop Entity Relationship diagram for the selected topic/project |
| 2 | Write a System Requirement specification for library management system. |
| 3 | To develop Data Flow Diagram For payroll system. |
| 4 | Develop sequence diagram selected / allotted project. |
| 5 | Develop Class diagram selected / allotted project. |
| 6 | Develop Class diagram for the selected project |
| 7 | To Develop Prototype for the selected project |
| 8 | To draw system architecture diagram of selected project |

Extra Experiment:

1.Component  Diagram

2. Deployment Diagram.

|  |  |
| --- | --- |
| Ex. No. 1. | **Develop Entity Relationship diagram for the selected topic/project** |
| Date: |

**Theory:**

An entity-relationship diagram (ERD) is a graphical representation of an information system that shows the relationship between people, objects, places, concepts or events within that system. An ERD is a [data modeling](http://searchdatamanagement.techtarget.com/definition/data-modeling) technique that can help define business processes and can be used as the foundation for a [relational database](http://searchsqlserver.techtarget.com/definition/relational-database).

Any object, for example, entities, attributes of an entity, relationship sets, and attributes of relationship sets, can be represented with the help of an ER diagram.

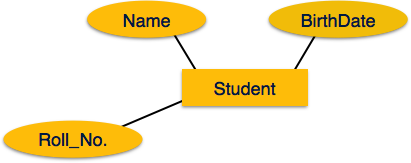
**Entity**

Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.

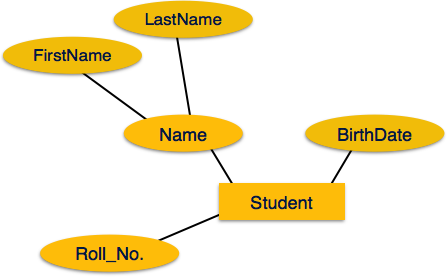
Entities in a school database

**Attributes**

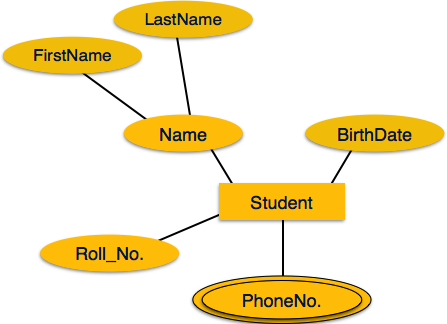
Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).



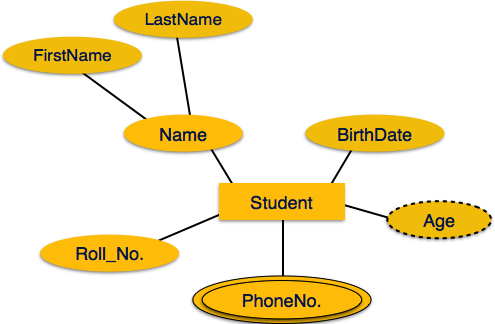
If the attributes are **composite**, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.



**Multivalued** attributes are depicted by double ellipse.



**Derived** attributes are depicted by dashed ellipse.



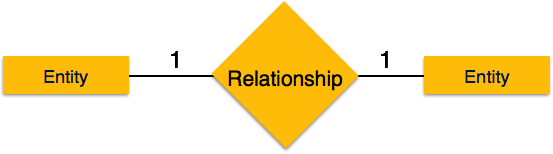
**Relationship**

Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship, are connected to it by a line.

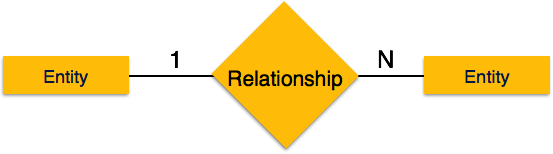
**Binary Relationship and Cardinality**

A relationship where two entities are participating is called a **binary relationship**. Cardinality is the number of instance of an entity from a relation that can be associated with the relation.

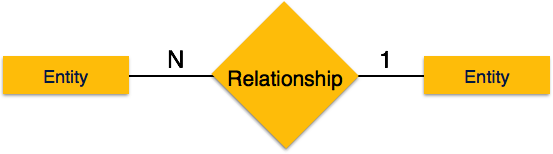
* **One-to-one** − When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.



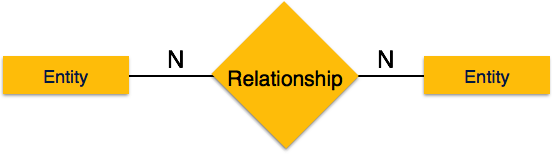
* **One-to-many** − When more than one instance of an entity is associated with a relationship, it is marked as '1:N'. The following image reflects that only one instance of entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts one-to-many relationship.



* **Many-to-one** − When more than one instance of entity is associated with the relationship, it is marked as 'N:1'. The following image reflects that more than one instance of an entity on the left and only one instance of an entity on the right can be associated with the relationship. It depicts many-to-one relationship.

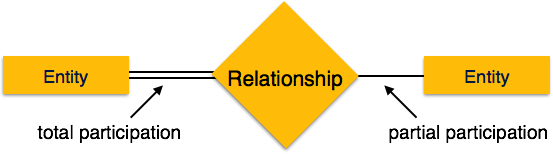


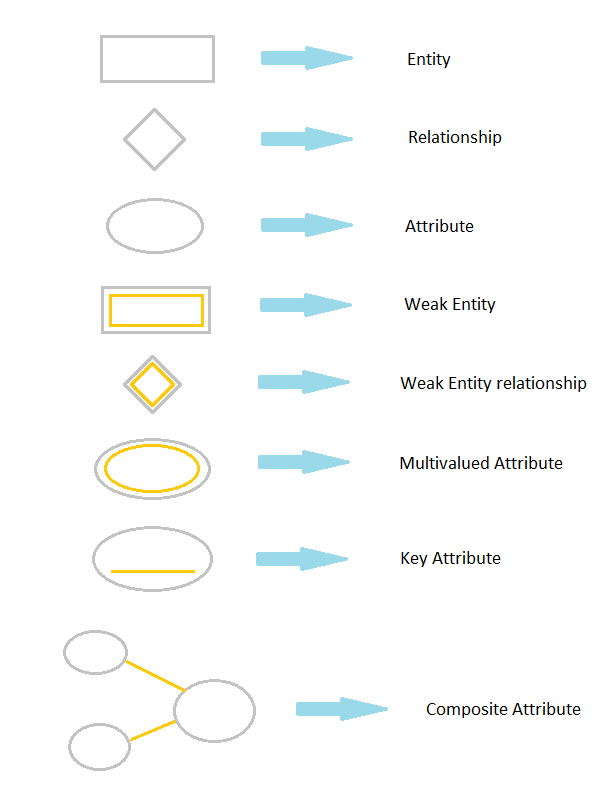
* **Many-to-many** − The following image reflects that more than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts many-to-many relationship.



**Participation Constraints**

* **Total Participation** − Each entity is involved in the relationship. Total participation is represented by double lines.
* **Partial participation** − Not all entities are involved in the relationship. Partial participation is represented by single lines.





**Viva Question:**

Q1. What is an ERD?

Q2.What is mean by attribute?

Q3.Explain the concept of relationship & cardinality?

Q4.What are the types of attributes?

Q5.what are the Participation constraints.

|  |  |
| --- | --- |
| Ex. No. 2. | **Write a System Requirement specification for the selected topic/project** |
| Date: |

Theory:-

Introduction-

**1.1 Purpose**

The main objective of this document is to illustrate the requirements of the project Library Management system. The document gives the detailed description of the both functional and non-functional requirements proposed by the client.The purpose of this project is to provide a friendly environment to maintain the details of books and library members.The main purpose of this project is to maintain easy circulation system using computers and to provide different reports. This project describes the hardware and software interface requirements using ER diagrams and UML diagrams.

**1.2 Scope of Development Project**

Library Management System is basically updating the manual library system into an internet-based application so that the users can know the details of their accounts, availability of books and maximum limit for borrowing.

The project is specifically designed for the use of librarians and library users. The product will work as a complete user interface for library management process and library usage from ordinary users. Library Management System can be used by any existing or new library to manage its books and book borrowing, insertion and monitoring. It is especially useful for any educational institute where modifications in the content can be done easily according to requirements.

The project can be easily implemented under various situations. We can add new features as and when we require, making reusability possible as there is flexibility in all the modules.

The language used for developing the project is Java as it is quite advantageous than other languages in terms of performance, tools available, cross platform compatibility, libraries, cost (freely available), and development process.

1.3 **Definitions, Acronyms and Abbreviations**

JAVA -> platform independence

SQL-> Structured query Language

ER-> Entity Relationship

UML -> Unified Modeling Language

IDE-> Integrated Development Environment

SRS-> Software Requirement Specification

ISBN -> International Standard Book Number

IEEE ->Institute of Electrical and Electronics Engineers

**2. Overall Descriptions**

**2.1 User Classes and Characteristics**

The system provides different types of services based on the type of users [Member/Librarian]. The Librarian will be acting as the controller and he will have all the privileges of an administrator. The member can be either a student or staff of the university who will be accessing the Library online.

The features that are available to the Librarian are:-

* A librarian can issue a book to the member.
* Can view the different categories of books available in the Library
* Can view the List of books available in each category
* Can take the book returned from students
* Add books and their information to the database
* Edit the information of existing books
* Can check the report of the existing books
* Can check the report of the issued books
* Can access all the accounts of the students

The features that are available to the Members are:-

* Can view the different categories of books available in the Library
* Can view the List of books available in each category
* Can own an account in the library.
* Can view the books issued to him
* Can put a request for a new book
* Can view the history of books issued to him previously
* Can search for a particular book

**2.2 Operating Environment**

The product will be operating in windows environment. The Library Management System is a website and shall operate in all famous browsers, for a model we are taking Microsoft Internet Explorer,GoogleChrome,and Mozilla Firefox.Also it will be compatible with the IE 6.0. Most of the features will be compatible with the Mozilla Firefox & Opera 7.0 or higher version. The only requirement to use this online product would be the internet connection.

The hardware configuration include Hard Disk: 40 GB, Monitor: 15” Color monitor, Keyboard: 122 keys. The basic input devices required are keyboard, mouse and output devices are monitor, printer etc.

**2.3 Assumptions and Dependencies**

The assumptions are:-

* The coding should be error free
* The system should be user-friendly so that it is easy to use for the users
* The information of all users, books and libraries must be stored in a database that is accessible by the website
* The system should have more storage capacity and provide fast access to the database
* The system should provide search facility and support quick transactions
* The Library System is running 24 hours a day
* Users may access from any computer that has Internet browsing capabilities and an Internet connection
* Users must have their correct usernames and passwords to enter into their online accounts and do actions

The dependencies are:-

* The specific hardware and software due to which the product will be run
* On the basis of listing requirements and specification the project will be developed and run
* The end users (admin) should have proper understanding of the product
* The system should have the general report stored
* The information of all the users must be stored in a database that is accessible by the Library System
* Any update regarding the book from the library is to be recorded to the database and the data entered should be correct

**2.4 Requirement**

Software Configuration:-

This software package is developed using java as front end which is supported by sun micro system. Microsoft SQL Server as the back end to store the database.

Operating System: Windows NT, windows 98, Windows XP

Language: Java Runtime Environment, Net beans 7.0.1 (front end)

Database: MS SQL Server (back end)

Hardware Configuration:-

Processor: Pentium(R)Dual-core CPU

Hard Disk: 40GB

RAM: 256 MB or more

**2.5 Data Requirement**

The inputs consist of the query to the database and the output consists of the solutions for the query. The output also includes the user receiving the details of their accounts. In this project the inputs will be the queries as fired by the users like create an account, selecting books and putting into account. Now the output will be visible when the user requests the server to get details of their account in the form of time, date and which books are currently in the account.

**2.6 Security Requirement**

* System will use secured database
* Normal users can just read information but they cannot edit or modify anything except their personal and some other information.
* System will have different types of users and every user has access constraints
* Proper user authentication should be provided
* No one should be able to hack users’ password
* There should be separate accounts for admin and members such that no member can access the database and only admin has the rights to update the database.

**2.7 Performance Requirement**

The proposed system that we are going to develop will be used as the Chief performance system within the different campuses of the university which interacts with the university staff and students. Therefore, it is expected that the database would perform functionally all the requirements that are specified by the university.

* The performance of the system should be fast and accurate
* Library Management System shall handle expected and non-expected errors in ways that prevent loss in information and long downtime period. Thus it should have inbuilt error testing to identify invalid username/password
* The system should be able to handle large amount of data. Thus it should accommodate high number of books and users without any fault

**2.8 Safety Requirement**

The database may get crashed at any certain time due to virus or operating system failure. Therefore, it is required to take the database backup so that the database is not lost. Proper UPS/inverter facility should be there in case of power supply failure.

Conclusion:-

**Viva Question:**

Q1.What is Mean by Requirement Specification?

Q2.What are the types of Requirements?

Q3.What is mean by software engineering?

Q4.What is the scope of development project?

Q5.What is the concept of assumptions & dependencies?

|  |  |
| --- | --- |
| Ex. No. 3. | **To develop Data Flow Diagram For the selected topic/project** |
| Date: |

**Theory:**-Overview

Data flow diagram (DFD) represents the flows of data between different processes in a business. It is a graphical technique that depicts information flow and the transforms that are applied as data move form input to output. It provides a simple, intuitive method for describing business processes without focusing on the details of computer systems. DFDs are attractive technique because they provide what users do rather than what computers do.

Representation of Components

DFDs only involve four symbols. They are:

* Process
* Data Object
* Data Store
* External entity

|  |  |
| --- | --- |
| http://members.tripod.com/%7Emyyee/cs457/process.gif | **Process** Transform of incoming data flow(s) to outgoing flow(s). |
| http://members.tripod.com/%7Emyyee/cs457/data.gif | **Data Flow** Movement of data in the system. |  |
| http://members.tripod.com/%7Emyyee/cs457/datastore.gif | **Data Store** Data repositories for data that are not moving. It may be as simple as a buffer or a queue or a s sophisticated as a relational database. |  |
| http://members.tripod.com/%7Emyyee/cs457/external.gif | **External Entity** Sources of destinations outside the specified system boundary. |  |

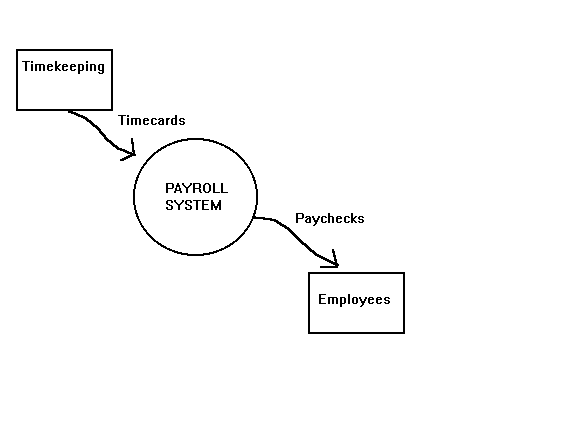
**Relationship and Rules**

Relationship

The DFD may be used for any level of data abstraction. DFD can be partitioned into levels. Each level has more information flow and data functional details than the previous level.

Highest level is Context Diagram of payroll system. Some important points are:

* 1 bubble (process) represents the entire system.
* Data arrows show input and output.
* Data Stores NOT shown. They are within the system.



A DFD may look similar to a flow chart. However, there is a significant difference with the data flow diagram. The arrows in DFDs show that there is a flow of data between the two components and not that the component is sending the data that must be executed in the following component. A component in DFD may not continue execution when sending data and during execution of the component receiving the data. The component sending data can send multiple sets of data along several connections. In fact, a DFD node can be a component that never ends.

Rules

* In DFDs, all arrows must be labeled.
* The information flow continuity, that is all the input and the output to each refinement, must maintain the same in order to be able to produce a consistent system.

**Strengths and Weaknesses**

Strengths

* DFDs have diagrams that are easy to understand, check and change data.
* DFDs help tremendously in depicting information about how an organization operations.
* They give a very clear and simple look at the organization of the interfaces between an application and the people or other applications that use it.

Weaknesses

* Modification to a data layout in DFDs may cause the entire layout to be changed. This is because the specific changed data will bring different data to units that it accesses. Therefore, evaluation of the possible of the effect of the modification must be considered first.
* The number of units in a DFD in a large application is high. Therefore, maintenance is harder, more costly and error prone. This is because the ability to access the data is passed explicitly from one component to the other. This is why changes are impractical to be made on DFDs especially in large system.

Conclusion

**Viva Question:**

Q1.What is mean by DFD?

Q2.Difference between DFD 0 & DFD 1?

Q3.Explain the components used in DFD?

Q4.Write down the strengths and weakness of DFD?

Q5.what is the use of DFD?

|  |  |
| --- | --- |
| Ex. No. 4. | **Develop sequence diagram selected / allotted project. .** |
| Date: |

**Theory:-**

UML provides two types of diagrams for the representation of interactions: the sequence diagram and the communication diagram. Both diagrams visualize the exchange of information. However, the emphasis is different: communication diagrams emphasize the relationships of individual objects and their topology; sequence diagrams emphasize the chronological course of exchanged information. In the external view, we opt for the representation through sequence diagrams and do without communication diagrams for two reasons:

* Sequence diagrams are easier to understand for developers and readers. In our practical work in projects we have observed a much higher acceptance of sequence diagrams because of their simplicity.
* We avoid using unnecessarily many diagram types for the same facts. Less is often more!

If a customer or business partner uses an offered service, partners communicate with each other. The process can be described as a series of interactions. These interactions are clearly laid out in the sequence diagram, whereas the activities of each partner and the conditions under which the interactions take place are omitted in the diagram. However, they can be described with supplementary comments.

Like the activity diagrams, sequence diagrams can be modeled spanning several use cases, as well as being used to refine business use cases. A sequence diagram illustrates the various scenarios of a business use case.

Draw Sequence diagram for selected project.

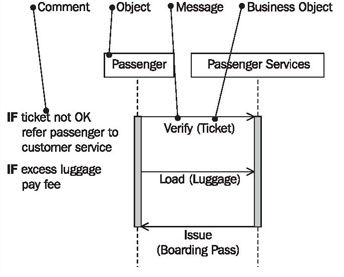
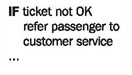


Figure 1. The elements of the sequence diagram

In a sequence diagram, we work with the following elements:

### Comment

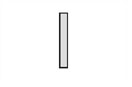
Sequence diagrams can be annotated with comments (UML generally permits comments in all diagrams.):



For instance, activities of partners or conditions can be specified as comments.

### Object

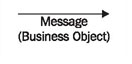
Objects that are involved in interactions are placed on the x-axis. Objects are senders and receivers of messages in the sequence diagram:



In the business system model (external view) these objects represent the actors of the business system and the business system itself.

### Message and Business Object

The messages that objects send and receive are shown on the y-axis. Messages are inserted in increasing chronological order from top to bottom. The direction of the arrow indicates the direction in which a message is sent:



The business object is listed in parenthesis. Business objects are conveyed together with messages. Some examples of business objects are tickets, boarding passes, and luggage. These examples will be treated in more detail in [Package Diagram](https://sourcemaking.com/uml/modeling-business-systems/internal-view/package-diagram).

## Reading Sequence Diagrams

Figure 2 shows a sequence diagram with the objects passenger and passenger services. The entire diagram documents the process of the business use case passenger check-in.

You begin reading a sequence diagram at the top (1). The starting point on the top left (1) is located on the vertical line that represents the passenger (2) as sender and receiver of messages. The flow begins when the passenger hands over his or her ticket (3) to passenger services for verification (4). The callverify (4) is the message; the ticket (3) that is handed over is the business object. The direction of the arrow indicates that the passenger is the sender of the message and passenger services the receiver (6). The receipt of the message at passenger services initiates activities, which is indicated by the gray vertical bar (7). The diagram does not show how passenger services handle the process, meaning that it does not show which activities are conducted:

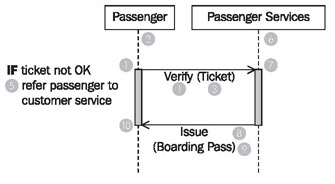


Figure 2 Sequence diagram "Passenger Check-In"

Only the comment (5) can include a clue. Comments can be inserted at the left margin of the sequence diagram.

In a final step, passenger services issues (8) a boardingpass (9) to the passenger. With that, the interaction that is illustrated in this sequence diagram is completed for both parties. This is indicated by the end of the wide gray vertical bar (10).

In the business model we do not utilize all the options of the sequence diagram. UML provides many more possibilities for this diagram type, but our experience showed that this is sufficient to communicate the essential aspects.

**Designate Actors and Business System—Who is Taking Part?**

Sequence diagrams illustrate the interactions between actors and the business system. Fundamentally we have a pool of interaction partners from the use case diagrams. Depending on the flow that is being depicted in the sequence diagram, the appropriate actors and business systems can be selected from this pool.

In our case study (see Figure 3), we find the interaction partners passenger and passenger servicesfor the above sequence diagram :

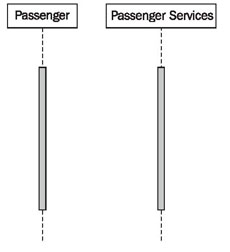


Figure 3 .Constructing sequence diagrams

**Designate Initiators—Who Starts Interactions?**

For every sequence of interactions the actor who starts the interaction has to be identified. This actor is called the initiator. Since in the external view of the business model each business use case is initiated by an actor, we can here also select the actor from the pool of actors in the use case diagrams.

In our sequence diagram passenger check-in, the passenger starts the interaction by utilizing the service check-in from passenger services.

**Describe the Message Exchange between Actors and the Business System—Which Messages are being Exchanged?**

After the initiator has been defined, the subsequent progression of interactions has to be identified. For each communication step it has to be determined what information is exchanged. In this way the message will be defined. Messages are requests to do something directed toward a particular partner. The business objects that are exchanged with these messages also have to be defined.

**Identify the Course of Interactions—What is the Order?**

All messages are exchanged in a chronological order that has to be identified. Messages are inserted along the y-axis in increasing chronological order, from top to bottom (see Figure 4):

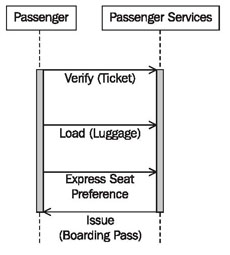


Figure .4 Constructing sequence diagrams

### Insert Additional Information—What Else is Important?

Important activities of involved actors and business systems and important conditions can be inserted into the diagram as comments. Comments are inserted at the level of the appropriate message. Restrict this to important comments that have significance so that the diagram is not overcrowded with text (see Figure 5):

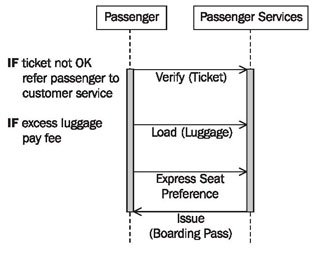


Figure 5 Constructing sequence diagrams

### Verify the View—Is Everything Correct?

Completed sequence diagrams can be verified with the following checklist:

#### Checklist :-Verifying Sequence Diagrams in the External View

* Are all required sequence diagrams completed and available? There should be a sequence diagram for each business use case.
* Are the sequence diagrams correct? Each sequence diagram contains only one object that represents the business system, and at most as many other objects as there are actors assigned to the business use case.
* Is each actor that is listed in the use case diagram mentioned in at least one sequence diagram?
* Is each actor who initiates a business use case mentioned as a starting point in one of the sequence diagrams?
* Have all the important comments been inserted into the diagram? Are there maybe too many comments inserted into the diagram thereby reducing its clarity?

**Viva Question:**

**Q1.What is mean by UML?**

**Q2.What are the constraints included in sequence diagram?**

**Q3.Why sequence diagram is used?**

**Q4.What are the drawback of sequence diagram?**

**Q5.Describe the key features of sequence diagram?**

|  |  |
| --- | --- |
| Ex. No. 5. | **Develop Class diagram selected / allotted project.** |
| Date: |

**Theory:-**

**Class**

A class represents a relevant concept from the domain, a set of persons, objects, or ideas that are depicted in the IT system:

Examples of classes are passengers, planes, or tickets.

**Attribute**

An attribute of a class represents a characteristic of a class that is of interest for the user of the IT system:

Characteristics of interest of a passenger, for example, are name and age.

**Generalization**

Generalization is a relationship between two classes: a general class and a special class:

https://sourcemaking.com/files/sm/images/uml/img_127.jpg

Refer to [Generalization, Specialization, and Inheritance](https://sourcemaking.com/uml/modeling-it-systems/structural-view/generalization-specialization-and-inheritance).

**Association**

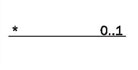
An association represents a relationship between two classes:

https://sourcemaking.com/files/sm/images/uml/img_128.jpg

An association indicates that objects of one class have a relationship with objects of another class, in which this connection has a specifically defined meaning (for example, "is flown with").

**Multiplicity**

A multiplicity allows for statements about the number of objects that are involved in an association:



Also see Figure 2.

### Aggregation

An aggregation is a special case of an association (see above) meaning "consists of":

The diamond documents this meaning; a caption is unnecessary.

## Reading Class Diagrams

Figure 1 shows a class diagram from our case study with the classes customer, ticket, and coupon, their attributes, and their associations:

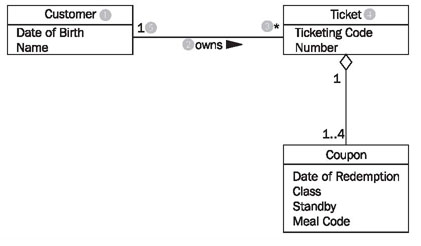


Figure 1 Class diagram with associations

Looking at the class diagram in Figure 1, you can read the association between the classes customer and ticket as follows:

* One (this sentence always begins with "one") object of the first class has an association with a number of objects of the second class.

The appropriate values from the diagram have to be inserted into this first abstract formulation, which can be universally applied. The name of one class is customer (1); the name of the other class is ticket (4). The name of the association is owns (2):

* A customer (1) owns (2) \* (3) ticket (4).

If the asterisk is exchanged with its meaning, a regular English sentence is created:

* A customer (1) owns (2) zero, one or several (3) ticket(s) (4).

Since associations usually are not directional, meaning usually go both directions, our association also has a meaning in the other direction:

* A ticket (4) is owned by (2) exactly one (5) customer (1).

The small triangle next to the name of the association (2) indicates in which direction the name of the association holds true. We can read all the associations in the class diagram in this way.

The specification of the number of objects of the second class (you always start with one object of the first class) is called the multiplicity. The course of action should always be according to the same pattern:

First, a statement of the lower limit (minimum number) followed by two periods (..) and a statement of the upper limit (maximum number).

Figure 2 shows the most common possibilities:

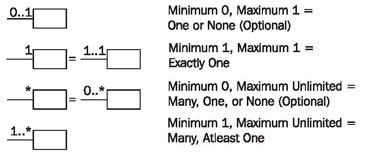


Figure 2 Multiplicities

However, in UML it is also possible to insert any values as the lower and upper limits, e.g., 2 .. 4 or 6 .. \*.

The association's name is necessary for understanding the domain meaning of the association. In contrast to the association itself, which applies to both directions, the name of the association applies to only one direction, which is indicated by a black triangle. If the association is not labeled, its meaning has to be derived from the domain context, or it takes on a general meaning such as has or belongs to. In case of doubt it is better to label associations too much than too little. Many diagrams that we have encountered in our practical experience were incomprehensible because associations were not labeled.

Associations can also be viewed as the implementation of static business rules (see [Static and Dynamic Business Rules](https://sourcemaking.com/uml/modeling-it-systems/structural-view/static-and-dynamic-business-rules)). Statements such as "a ticket belongs to exactly one customer" are documented in the class diagram by associations.

Roles are another possible way in UML to give relationships between classes a domain meaning. In this way, we can state what role an object of one class plays for the objects of another class:

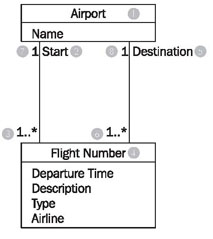


Figure 3 Class diagram with roles

Looking at the class diagram in Figure 3, we can read the left association with roles between the classes flight number and airport as follows:

* An airport (1) is a start (location) (2) for one or more (3) flightnumbers (4).

There is another association between the two classes flight number and airport:

* An airport (1) is a destination (5) for one or more flight numbers (4).

These two associations also have inversions, even though roles are only stated for one direction:

* A flight number (4) has as start (location) (2) exactly one (7) airport (1).
* A flight number (4) has as destination (5) exactly one (8) airport (1).

This records that a certain flight number has a departure airport and a destination airport. An example of a flight number is LX317, a daily flight of the Swiss airline Crossair from London to Zurich.

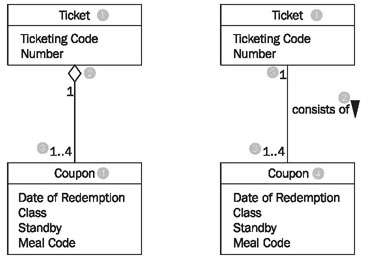


Figure 4 Class diagram with Aggregation

Among the many domain meanings that an association can have there is one that can be signified withUML by its own symbol: the whole-part relationship or aggregation. This type of relationship is always used when objects of one class are a part of objects of another class.

In the class diagram in Figure 4 aggregation is used on the left side (the white diamond), which can be read as follows:

* A ticket (1) consists of (2) 1 to 4 (3) coupons (4).

or the other way around:

* A coupon (4) is part of (2) exactly one (5) ticket (1).

The example without a diamond, but with a name for the association, has exactly the same meaning! The last missing element of UML that we use to model class diagrams is generalization/specialization, which serves to depict the relationship between a superclass and a subclass. The generalization/specialization in Figure 4.35 can be read from top to bottom or bottom to top. If you begin at the top, you find a classFreight (1) with the attributes: Identification, Weight, and ID-number (2). This class has two specializations, Piece of Luggage (3) and Piece of Cargo (4). The class Piece of Cargo has an additional attribute: Degree of Hazardousness (5).

If you begin at the bottom you will find the classes Piece of Luggage (3) and Piece of Cargo (4). These have a superclass, the class Freight (1), which contains the shared attributes (and functions) of the subclasses.

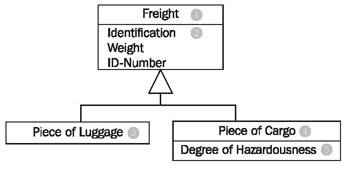


Figure 5 Class diagram with generalization/specialization

**Viva Question:-**

**Q1.Explain class diagram with example?**

**Q2.What are the components used in class diagram?**

**Q3.Describe the terms class ,object?**

**Q4.What is mean by generalization?**

**Q5.Explain the concept of association.**

|  |  |
| --- | --- |
| Ex. No. 6. | **Develop Class diagram selected / allotted project.** |
| Date: |

**Theory:**

**Class Diagram**

The class diagram is a static diagram. It represents the static view of an application. A class is a group of objects with similar properties (attributes), common behaviors (operations), common relationships to other objects, and common semantics.The purpose of a class diagram is to depict the classes within a model. In an object oriented application, classes have attributes (member variables), operations (member functions) and relationships with other classes. The UML class diagram can depict all these things quite easily. The fundamental element of the class diagram is an icon the represents a class.

The class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams which can be mapped directly with object oriented languages.

The class diagram shows a collection of classes, interfaces, associations, collaborations and constraints. It is also known as a structural diagram.

So the purpose of the class diagram can be summarized as:

* Analysis and design of the static view of an application.
* Describe responsibilities of a system.
* Base for component and deployment diagrams.

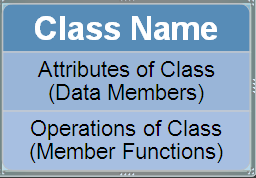
The following points should be remembered while drawing a class diagram:

* The name of the class diagram should be meaningful to describe the aspect of the system.
* Each element and their relationships should be identified in advance.
* Responsibility (attributes and methods) of each class should be clearly identified.
* For each class minimum number of properties should be specified. Because unnecessary properties will make the diagram complicated.
* Use notes when ever required to describe some aspect of the diagram. Because at the end of the drawing it should be understandable to the developer/coder.
* Finally, before making the final version, the diagram should be drawn on plain paper and rework as many times as possible to make it correct.

Class diagrams are used for:

* Describing the static view of the system.
* Showing the collaboration among the elements of the static view.
* Describing the functionalities performed by the system.
* Construction of software applications using object oriented languages.

**Class Diagram Notation**

****

The top text area contains the class name, the middle area defines the attributes, and the bottom area defines the operations.

**Attributes**

Attributes describe an object's characteristics. Use this syntax to specify data attributes in the middle text area of the class symbol.The attribute section is optional, but when used it contains each attribute of the class displayed in a list format.

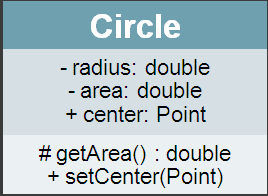
### Operations

Operations are the actions that can be carried out on or by the objects. Like the attributes, the operations of a class are displayed in a list format, with each operation on its own line

To specify the visibility of a class member (both attributes and methods) use the following before the member's name:

* + public: +
  + protected: #
  + private: -
  + package: ~

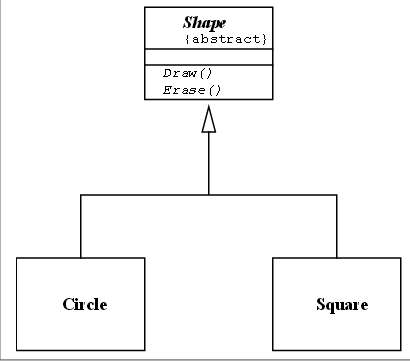
Following figure shows the shows a typical UML description of a class that represents a circle

****

A relationship is a general term covering the specific types of logical connections found on class and object diagrams

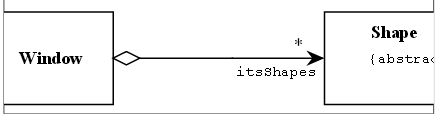
**Inheritance**

The inheritance relationship in UML is depicted by a peculiar triangular arrowhead. This arrow-head that looks rather like a slice of pizza, points to the base class. One or more lines proceed from the base of the arrowhead connecting it to the derived classes. Following Figure shows the form of the inheritance relationship. In this diagram we see that Circle and Square both derive from Shape. Note that the name of class Shape is shown in italics. This indicates that Shape is an abstract class.



**Aggregation / Association**

A class aggregates another if its objects contain objects of the other class. The weak form of aggregation is denoted with an open diamond. This relationship denotes that the aggregate class (the class with the white diamond touching it) is in some way the “whole”, and the other class in the relationship is somehow “part” of that whole



Above Figure shows an aggregation relationship. In this case, the Window class contains many Shape instances. In UML the ends of a relationship are referred to as its “roles”. Notice that the role at the Shape end of the aggregation is marked with a “\*”. This indicates that the Window contains many Shape instances. Notice also that the role has been named. This is the name that Window knows its Shape instances by. i.e. it is the name of the instance variable within Window that holds all the Shapes.

**Basic aggregation**  
An association with an aggregation relationship indicates that one class is a part of another class. In an aggregation relationship, the child class instance can outlive its parent class. To represent an aggregation relationship, you draw a solid line from the parent class to the part class, and draw an unfilled diamond shape on the parent class's association end. Figure 12 shows an example of an aggregation relationship between a Car and a Wheel.

##### Figure 12: Example of an aggregation association

Example of an aggregation association

**Composition aggregation**  
The composition aggregation relationship is just another form of the aggregation relationship, but the child class's instance lifecycle is dependent on the parent class's instance lifecycle. In Figure 13, which shows a composition relationship between a Company class and a Department class, notice that the composition relationship is drawn like the aggregation relationship, but this time the diamond shape is filled.

##### Figure 13: Example of a composition relationship

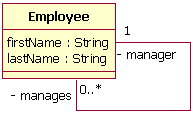
Example of a composition relationship

In the relationship modeled in Figure 13, a Company class instance will always have at least one Department class instance. Because the relationship is a composition relationship, when the Company instance is removed/destroyed, the Department instance is automatically removed/destroyed as well. Another important feature of composition aggregation is that the part class can only be related to one instance of the parent class (e.g. the Company class in our example).

### Reflexive associations

We have now discussed all the association types. As you may have noticed, all our examples have shown a relationship between two different classes. However, a class can also be associated with itself, using a reflexive association. This may not make sense at first, but remember that classes are abstractions. Figure 14 shows how an Employee class could be related to itself through the manager/manages role. When a class is associated to itself, this does not mean that a class's instance is related to itself, but that an instance of the class is related to another instance of the class.

##### Figure 14: Example of a reflexive association relationship



The relationship drawn in Figure 14 means that an instance of Employee can be the manager of another Employee instance. However, because the relationship role of "manages" has a multiplicity of 0..\*; an Employee might not have any other Employees to manage.

**What is the difference between an aggregation and an association?**

The difference is one of implication. Aggregation denotes whole/part relationships whereas associations do not. However, there is not likely to be much difference in the way that the two relationships are implemented. That is, it would be very difficult to look at the code and determine whether a particular relationship ought to be aggregation or association.

Aggregation and Association both correspond to the Has-by-reference relationship

**Viva Question:**

Q1.What is mean by aggregation?

Q2.Explain the term inheritance with example.

Q3.What is mean by Reflexive associations?

Q4.Explain the concept of Composite aggregation.

Q5. What are points should be remembered while drawing a class diagram.

|  |  |
| --- | --- |
| Ex. No. 7. | **To Develop Prototype for the selected project** |
| Date: |

**Theory:**

**Prototyping**

Information gathered in the prototyping phase allows the analyst to set priorities and redirectplans inexpensively, with a minimum of disruption. Because of this feature, prototyping and planning go hand-in-hand.

**Kinds of Prototypes**

The word prototype is used in many different ways.

1. **PATCHED-UP PROTOTYPE**

The first kind of prototyping has to do with constructing a system thatworks but is patched up or patched together. In engineering this approach is referred to asbreadboarding: creating a patched-together, working model of an (otherwise microscopic)integrated circuit.

An example in information systems is a working model that has all the necessary features butis inefficient. In this instance of prototyping, users can interact with the system, getting accustomedto the interface and types of output available. The retrieval and storage of information maybe inefficient, however, because programs were written rapidly with the objective of being workablerather than efficient.

1. **NONOPERATIONAL PROTOTYPE**

The second conception of a prototype is that of a nonworkingscale model that is set up to test certain aspects of the design. An example of this approach is afull-scale model of an automobile that is used in wind tunnel tests. The size and shape of the autoare precise, but the car is not operational. In this case only features of the automobile essential towind tunnel testing are included.Anon-working scale model of an information system might be produced when the coding requiredby the applications is too extensive to prototype but when a useful idea of the system canbe gained through the prototyping of the input and output only. In this instance, processing, becauseof undue cost and time, would not be prototyped. Users could still make decisions on theutility of the system, based on their use of prototyped input and output.

1. **FIRST-OF-A-SERIES PROTOTYPE**

A third conception of prototyping involves creating a first fullscalemodel of a system, often called a pilot. An example is prototyping the first airplane of aseries, then seeing if it flies before building a second. The prototype is completely operational andis a realization of what the designer hopes will be a series of airplanes with identical features.This type of prototyping is useful when many installations of the same information systemare planned. The full-scale working model allows users to experience realistic interaction withthe new system, but it minimizes the cost of overcoming any problems that it presents.

1. **SELECTED FEATURES PROTOTYPE**

A fourth conception of prototyping concerns building anoperational model that includes some, but not all, of the features that the final system will have.An analogy would be a new retail shopping mall that opens before the construction of all shopsis complete.When prototyping information systems in this way, some, but not all, essential features areincluded. For example, users may view a system menu on a screen that lists six features: add arecord, update a record, delete a record, search a record for a key word, list a record, or scan arecord. In the prototyped system, however, only three of the six may be available for use, so thatthe user may add a record (feature 1), delete a record (feature 3), and list a record (feature 5). Userfeedback can help analysts understand what is working and what isn’t. It can also help with suggestionson what features to add next.When this kind of prototyping is done, the system is accomplished in modules so that if thefeatures that are prototyped are evaluated by users as successful, they can be incorporated into thelarger, final system without undertaking immense work in interfacing. Prototypes done in thismanner are part of the actual system. They are not just a mock-up as in nonoperational prototypingconsidered previously.

**Guidelines for Developing a Prototype**

Once the decision to prototype has been made, four main guidelines must be observed when integratingprototyping into the requirements determination phase of the SDLC:

**1.** Work in manageable modules.

**2.** Build the prototype rapidly.

**3.** Modify the prototype in successive iterations.

**4.** Stress the user interface.

**WORKING IN MANAGEABLE MODULES.**

When prototyping some of the features of a system intoa workable model, it is imperative that the analyst work in manageable modules. One distinctadvantage of prototyping is that it is not necessary or desirable to build an entire working systemfor prototype purposes.

Amanageable module is one that allows users to interact with its key features but can be builtseparately from other system modules. Module features that are deemed less important are purposelyleft out of the initial prototype.

**BUILDING THE PROTOTYPE RAPIDLY**

Speed is essential to the successful prototyping of aninformation system. Analysts can use prototyping to shorten this gap by using traditional information-gatheringtechniques to pinpoint salient information requirements, and then quickly make decisions thatbring forth a working model. In effect the user sees and uses the system very early in the SDLCinstead of waiting for a finished system to gain hands-on experience.

**MODIFYING THE PROTOTYPE.**

A third guideline for developing the prototype is that itsconstruction must support modifications. Making the prototype modifiable means creating it inmodules that are not highly interdependent.

The prototype is generally modified several times, going through several iterations. Changesin the prototype should move the system closer to what users say is important. Each modificationnecessitates another evaluation by users.

The prototype is not a finished system.

**STRESSING THE USER INTERFACE**

The user’s interface with the prototype (and eventually thesystem) is very important. Because what you are really trying to achieve with the prototype is to getusers to further articulate their information requirements, they must be able to interact easily with thesystem’s prototype. They should be able to see how the prototype will enable them to accomplishtheir tasks. For many users the interface is the system. It should not be a stumbling block.

**Disadvantages of Prototyping**

It can be quite difficult to manage prototyping as a project in the larger systems effort.

Users and analysts may adopt a prototype as a completed systemwhen it is in fact inadequate and was never intended to serve as a finished system. Analysts needto work to ensure that communication with users is clear regarding the timetable for interactingwith and improving the prototype.

**Advantages of Prototyping**

The threemajor advantages of prototyping are the potential for changing the system early in its development,the opportunity to stop development on a system that is not working, and the possibility ofdeveloping a system that more closely addresses users’ needs and expectations.Successful prototyping depends on early and frequent user feedback, which analysts can useto modify the system and make it more responsive to actual needs. As with any systems effort,early changes are less expensive than changes made late in the project’s development

**Develop a Prototype for Selected project**

**Viva Question:**

**Q1.What is mean by prototyping?**

**Q2.State the advantages of prototyping.**

**Q3.What are the guidelines used in development of prototyping.**

**Q4.state the features of prototyping.**

**Q5.What are the different types of prototype?**

|  |  |
| --- | --- |
| Ex. No. 8. | **To draw system architecture diagram of selected project** |
| Date: |

**Theory:**

Today, most information systems are spread across two or more computers. A Webbased system, for example, can run in the browser on your desktop computer, but will interact with the Web server (and possibly other computers) over the Internet. A system that operates completely inside a company’s network may have a Visual Basic program installed on your computer, but interact with a database server elsewhere on the company network. An important step of the design phase is the creation of the architecture design, the plan for how the information system components will be distributed across multiple computers and what hardware, operating system software, and application software will be used on each computer (e.g., Windows or Linux operating system software). Designing the system architecture can be quite difficult; therefore, many organizations use the skills of experienced, expert system architects (consultants or employees) who specialize in the task

**ELEMENTS OF AN ARCHITECTURE DESIGN**

The objective of architecture design is to determine how the software components of the information system will be assigned to the hardware devices of the system. Although there are numerous ways in which the software components can be placed on the hardware components, the most common architecture is the client–server architecture.

**Architectural Components**

The major architectural components of any system are the software and the hardware. The major software components of the system being developed have to be identified and then allocated to the various hardware components on which the system will operate. Each of these components can be combined in a variety of different ways.

All software systems can be divided into four basic functions.

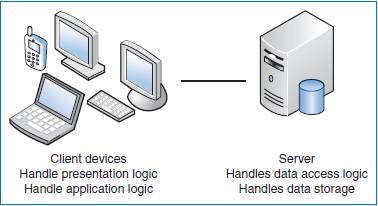
1. The first is data storage. Most information systems require data to be stored and retrieved, whether a small file, such as a list of lawn chemicals that are no longer authorized for residential applications, or a large database that stores an organization’s human resources records. These are the data entities documented in ERDs.
2. The second function is the data access logic: the processing required to access data, often meaning database queries in Structured Query Language (SQL).
3. The third function is the application logic: the logic documented in the DFDs, use cases, and functional requirements.
4. The fourth function is the presentation logic: the display of information to the user and the acceptance of the user’s commands (the user interface).

These four functions (data storage, data access logic, application logic, and presentation logic) are the basic building blocks of any information system.

The three primary hardware components of a system are client computers, servers, and the network that connects them.

* Client computers are the input–output devices employed by the user and are usually desktop or laptop computers, but can also be handheld devices, smartphones, special-purpose terminals, and so on.
* Servers typically are larger multi-user computers used to store software and data that can be accessed by anyone who has permission.
* The network that connects the computers can vary in speed from slow cell phones or modem connections that must be dialed, to medium-speed always-on frame relay networks, to fast always-on broadband connections such as cable modem, DSL to high-speed always-on Ethernet.

**Client–Server Architectures**

In these architectures, the client is responsible for the presentation logic, whereas the server is responsible for the data access logic and data storage. The application logic may reside on the client, reside on the server, or be split between both (Figure 8-1). If the client shown in Figure 8.1 contained all or most of the application logic, it is called a thick or fat client. Currently, ****

**Fig 8.1 :** Two-Tiered Client–Server Architecture

thinclients, containing just a small portion of the application logic, are popular because of lower overhead and easier maintenance.

For example, many Web based systems are designed with the Web browser performing presentation and only minimal application logic using such programming languages as JavaScript, while the server side has most of the application logic, all of the data access logic, and all of the data storage.

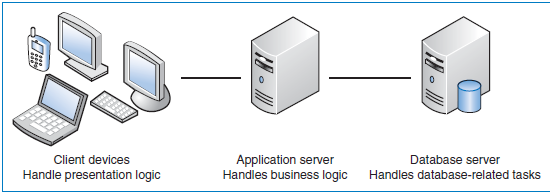
Client–server architectures have **four important benefits.**

1. First, they are scalable. That means it is easy to increase or decrease the storage and processing capabilities of the servers. If one server becomes overloaded, you simply add another server so that many servers are used to perform the application logic, data access logic, or data storage. The cost to upgrade is gradual, and you can upgrade in small increments.
2. Second, client–server architectures can support many different types of clients and servers. It is possible to connect computers that use different operating systems so that you are not locked into one vendor. Users can choose which type of computer they prefer (e.g., combining both Windows computers and Apple Macintoshes on the same network). Middleware is a type of system software designed to translate between different vendors’ software. Middleware is installed on both the client computer and the server computer. The client software communicates with the middleware, which can reformat the message into a standard language that can be understood by the middleware, which assists the server software.
3. Third it is simple to clearly separate the presentation logic, the application logic, and the data access logic and design each to be somewhat independent. For example, the presentation logic can be designed in HTML or XML to specify how the page will appear on the screen. Likewise, it is possible to change the application logic without changing the presentation logic or the data, which are stored in databases and accessed by SQL commands.
4. Finally, only the applications requiring that server will fail. The failed server can be swapped out and replaced and the applications can then be restored.

**Client–Server Tiers**

There are many ways in which the application logic can be partitioned between the

client and the server. The arrangement in Figure 8.1 is a common configuration.

****

**Fig. 8.2:** n-Tiered Client–Server Architecture

In this case, the server is responsible for the data and the client is responsible for the application and presentation. This is called a two-tiered architecture because it uses only two sets of computers—clients and servers.

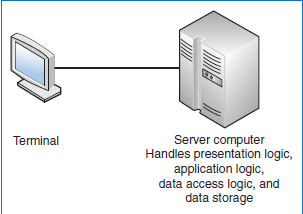
A three-tiered architecture uses three sets of computers, as shown in

Figure 8.2.

In this case, the software on the client computer is responsible for presentation logic, an application server(s) is responsible for the application logic, and a separate database server(s) is responsible for the data access logic and data storage. Typically, the user interface runs on a desktop PC or workstation and uses a standard graphical user interface. The application logic may consist of one or more separate modules running on a workstation or application server. Finally, a relational DBMS running on a database server contains the data access logic and data storage. The middle tier may be divided into tiers itself, resulting in an overall architecture called an “n-tier architecture”. An n-tiered architecture distributes the work of the application (the middle tier) among multiple layers of more specialized server computers. This type of architecture is common in today’s Web-based e-commerce systems.

**Less Common Architectures**

**Server-Based Architectures** The very first computing architectures were serverbased, with the server (usually, a central mainframe computer) performing all four application functions. The clients (usually, terminals) enabled users to send and receive messages to and from the server computer. The clients merely captured keystrokes and sent them to the server for processing, and accepted instructions from the server on what to display (Figure 8-4).

****

**Fig 8.4** Server-Based Architecture

This very simple architecture often works very well. Application software is developed and stored on the server, and all data are on the same computer. There is one point of control because all messages flow through the one central server. Software development and software administration are simplified because a single computer hosts the entire system (operating system and application software).

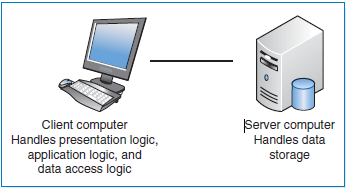
The server-based architecture was the first architecture used in information systems, but did not remain the only option as hardware and software evolved. The fundamental problem with early server-based systems was that the server processed all the work in the system. As the demands for more and more applications and the number of users grew, server computers became overloaded and unable to quickly process all the users’ demands. Response time became slower, and IS managers were required to spend increasingly more money to upgrade the server computer.

Today, the server-based architecture remains a viable architecture choice. Zero client, or ultrathin client, is a server-based computing model that is often used today in a virtual desktop infrastructure (VDI).

A typical zero client device is a small box that connects a keyboard, mouse, monitor, and Ethernet connection to a remote server. The server hosts everything: the client’s operating system and all software applications. The server can be accessed wirelessly or with cable.

Zero client computing has a number of benefits. Power usage can be significantly reduced compared to fat client configurations. This benefit is increasing in importance as more companies are investigating green computing. The devices used are much less expensive than PCs or even thin client devices. Since there is no software at the client device, there is no vulnerability to malware. The zero client computing model provides an efficient and secure way to deliver applications to end users.

**Client-Based Architectures** With client-based architectures, the clients are microcomputers on a local area network, and the server is a server computer on the same network. The application software on the client computers is responsible for the presentation logic, the application logic, and the data access logic; the server simply provides storage for the data (Figure 8.5).

****

**Fig.-8.5** Client-Based Architecture

This simple architecture often works very well in situations with low numbersof users or limited data access requirements.

The fundamental problem in theclient-based architecture is that all data on the server must travel to the client forprocessing. For example, suppose that the user wishes to display a list of allemployees with company life insurance. All the data in the employee database musttravel from the server, where the database is stored, over the network to the client,which then executes the query to find each record that matches the data requestedby the user.

**Draw Architecture diagram for selected topic**

**Viva Question:**

**Q1.What is mean by an architecture diagram?**

**Q2.Explain Client–Server Architecture.**

**Q3.What are the advantages of an architecture diagram?**

**Q4.what are the basic functions of software system?**

**Q5.What are the elements of an architecture design?**