

< Project Name >

Design Document

Version 1.0



Group Id: F160....

Supervisor Name: Sir / Miss

Revision History

Date	Version	Description	Author
Current date	<u>1.0</u>	Design Document includes Introduction of Design Document, Entity Relationship Diagram (ERD), Sequence Diagrams, Architecture Design Diagram, Class Diagram, Database Design, Interface Design and Test Cases	Student name Student VU id

Table of Contents

1. Introduction of Design Document
2. Entity Relationship Diagram (ERD)
3. Sequence Diagrams
4. Architecture Design Diagram
5. Class Diagram
6. Database Design
7. Interface Design
8. Test Cases

1. Introduction of Design Document

The Software Design Document is a document to provide documentation which will be used to aid in software development by providing the details for how the software should be built. Within the Software Design Document are narrative and graphical documentation of the software design for the project including use Entity Relationship Diagram (ERD), Sequence, Diagrams, Architecture Design Diagram, class diagram, Database Design, Interface Design, Test Cases, and other supporting requirement information. The purpose of the Software Design Document is to provide a description of the design of a system fully enough to allow for software development to proceed with an understanding of what is to be built and how it is expected to built. The Software Design Document provides information necessary to provide description of the details for the software and system to be built.

2. Entity Relationship Diagram (ERD)

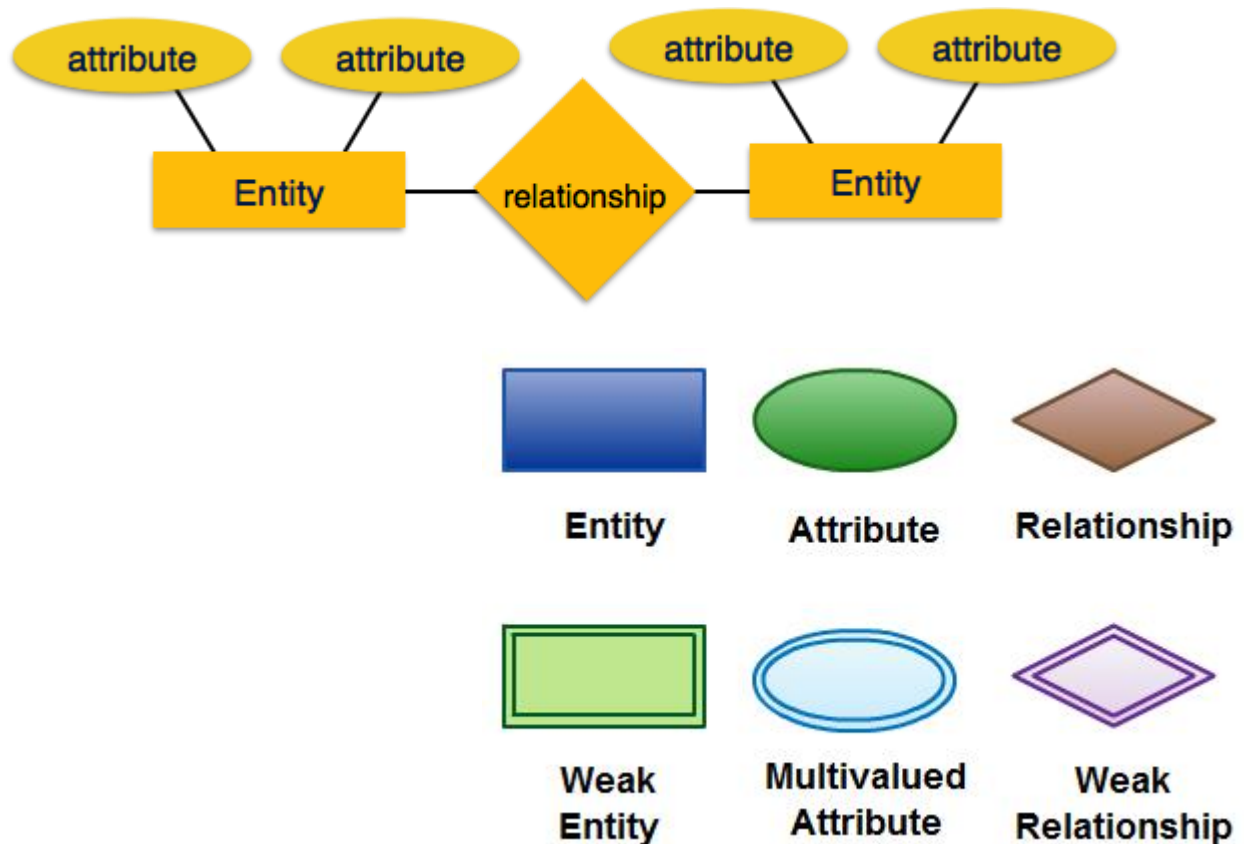
An entity-relationship diagram—otherwise known as an ERD—is a data modeling technique that creates an illustration of an information system's entities and the relationships between those entities.

3 ingredients of entity-relationship diagram

Entities, which represent people, places, items, events, or concepts.

Attributes, which represent properties or descriptive qualities of an entity. These are also known as data elements.

Relationships, which represent the link between different entities.



ER Diagram Symbols and Notations

Cardinality and Modality

Cardinality and Modality work together to define the relationship.

- Cardinality indicates the maximum number of times an instance in one entity can be associated with instances in the related entity.
- Modality indicates the minimum number of times an instance in one entity can be associated with an instance in the related entity.

Thus, Modality is also called participation because it denotes whether or not an instance of an entity **MUST** participate in the relationship.

Cardinality and Modality are both shown on the relationship line by symbols. We will go over each of the symbols and how to interpret them.

Cardinality



Cardinality indicates the maximum number of times an instance of one entity can be associated with instances in the related entity. Cardinality can have the values of one or many, no more detail than that. It is either one or more than one. On the relationship line, the cardinality is the closest to the entity box. The cardinality symbol in the diagram on the slide is in the red circle. Cardinality is indicated at **BOTH** ends of the relationship line, so there is a left to right cardinality and a right to left cardinality.

Modality



Modality indicates the minimum number of times an instance in one entity can be associated with an instance in the related entity. Modality can have the values of zero or one, two or three are not allowed. The modality symbol is located next to the cardinality symbol, on the inside, i.e., **NOT** next to the entity box. A modality of one is denoted by a straight vertical line and a modality of zero is denoted by a circle. Like cardinality, modality is indicated at both ends of the relationship.

Reading Modality and Cardinality

from Zero to Many



from One to Many



from One to One

i.e., one and only one



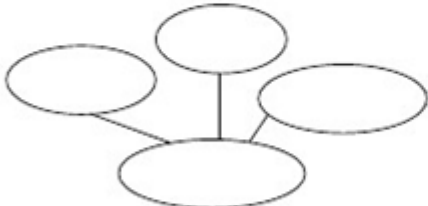



from Zero to One



Types of Attributes

- Simple attribute – Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.
- Composite attribute – Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.
- Derived attribute – Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.
- Single-value attribute – Single-value attributes contain single value. For example – Social_Security_Number.
- Multi-value attribute – Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

	Simple Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute

Types of Entity –


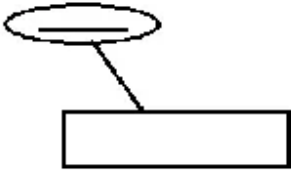
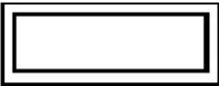
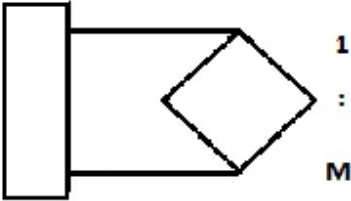

Strong Entity Types

Recursive Entity Types

Weak Entity Types

Composite Entity Types or Associative Entity Types

Notations Of different Entity Type in ER Diagram

	Entity
	Strong Entity Type
	Weak Entity Type
	Recursive Entity Type
	Composite Entity Type (or) Associative Entity

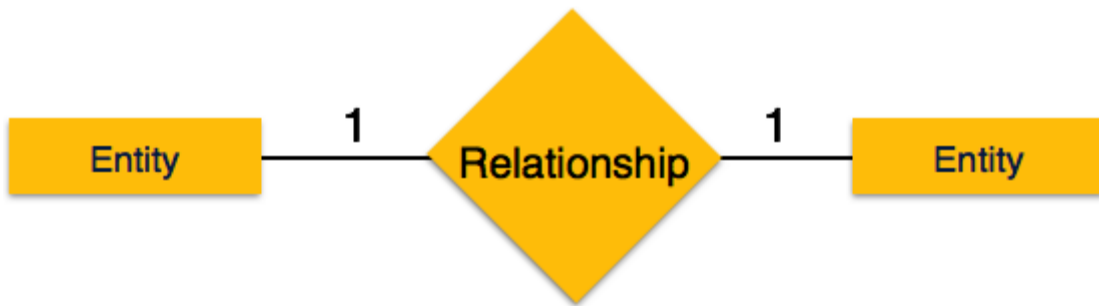
Relationship

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.

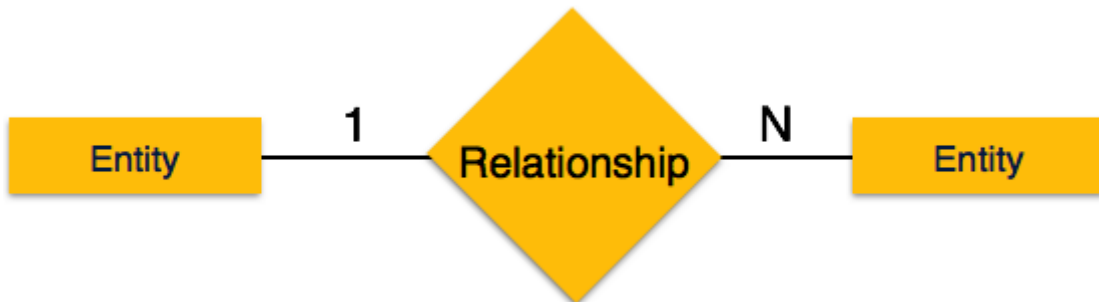
Binary Relationship and Cardinality

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.

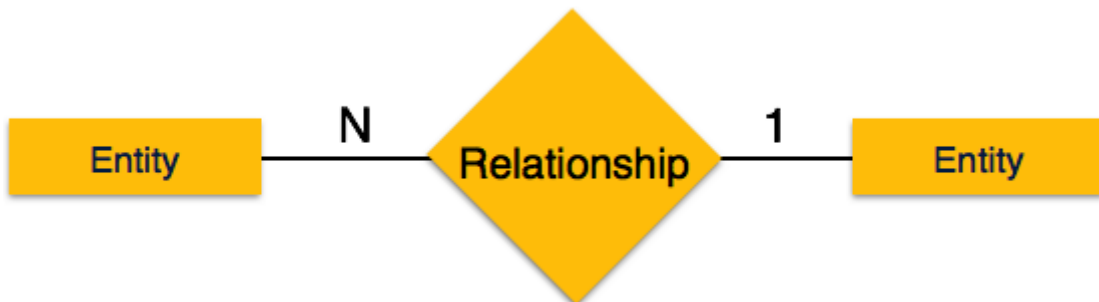
- One-to-one – When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.



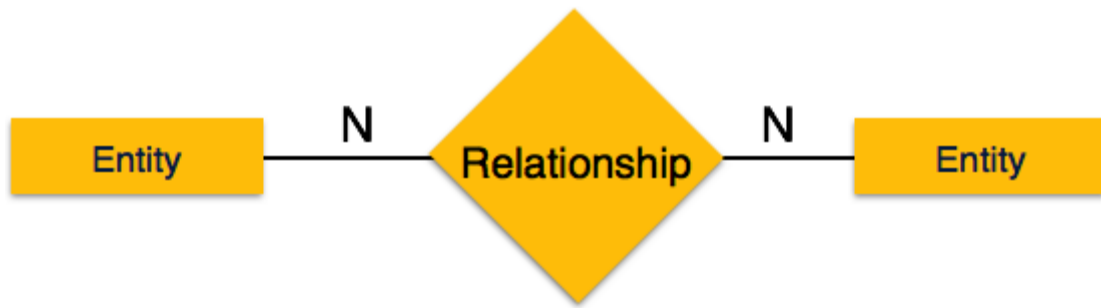
- One-to-many – When more than one instance of an entity is associated with a relationship, it is marked as '1:N'. The following image reflects that only one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts one-to-many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



3. Sequence Diagrams

The Sequence Diagram shows how the objects interact with others in a particular scenario of a use case.

Basic Sequence Diagram Symbols and Notations

The Sequence Diagram shows how the objects interact with others in a particular scenario of a use case.

Basic Sequence Diagram Symbols and Notations

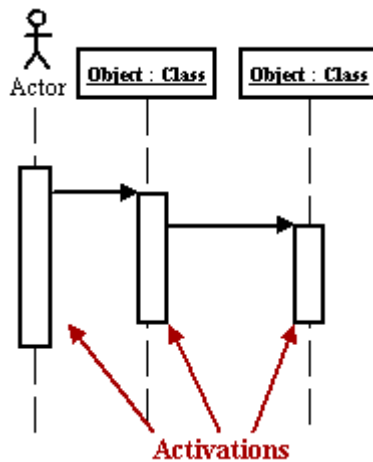
Class roles

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.

Object : Class

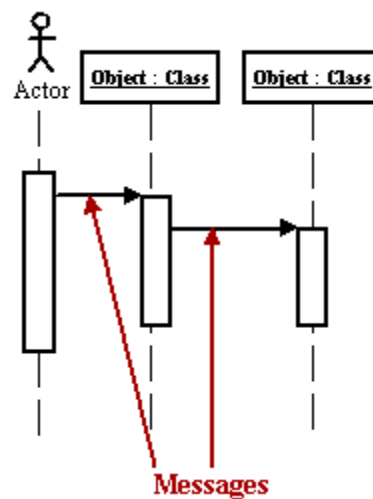
Activation

Activation boxes represent the time an object needs to complete a task.



Messages

Messages are arrows that represent communication between objects. Use half-arrowed lines to represent asynchronous messages. Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks.

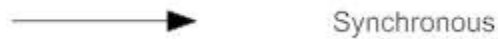


Various message types for Sequence and Collaboration diagrams

Types of Messages in Sequence Diagrams

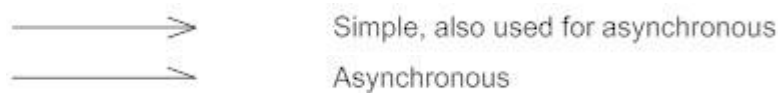
Synchronous Message

A synchronous message requires a response before the interaction can continue. It's usually drawn using a line with a solid arrowhead pointing from one object to another.



Asynchronous Message

Asynchronous messages don't need a reply for interaction to continue. Like synchronous messages, they are drawn with an arrow connecting two lifelines; however, the arrowhead is usually open and there's no return message depicted.



Reply or Return Message

A reply message is drawn with a dotted line and an open arrowhead pointing back to the original lifeline.

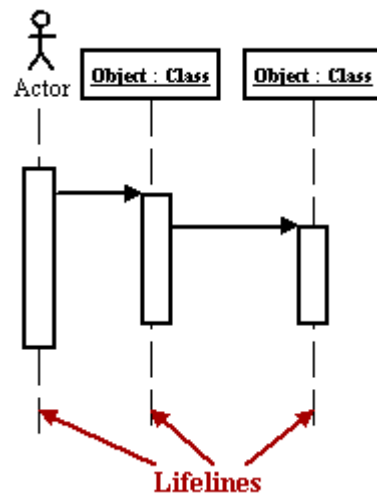


Self Message

A message an object sends to itself, usually shown as a U shaped arrow pointing back to itself.

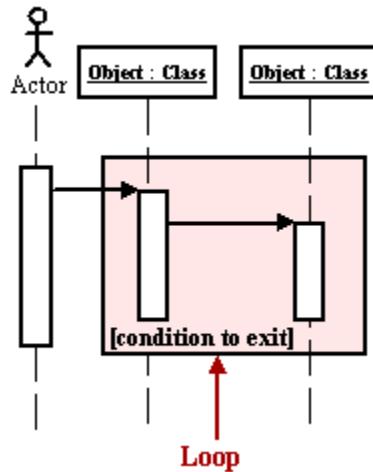
Lifelines

Lifelines are vertical dashed lines that indicate the object's presence over time.



Loops

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets [].



Basic Sequence Diagram Symbols and Notations

Class Roles or Participants

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.



Activation or Execution Occurrence

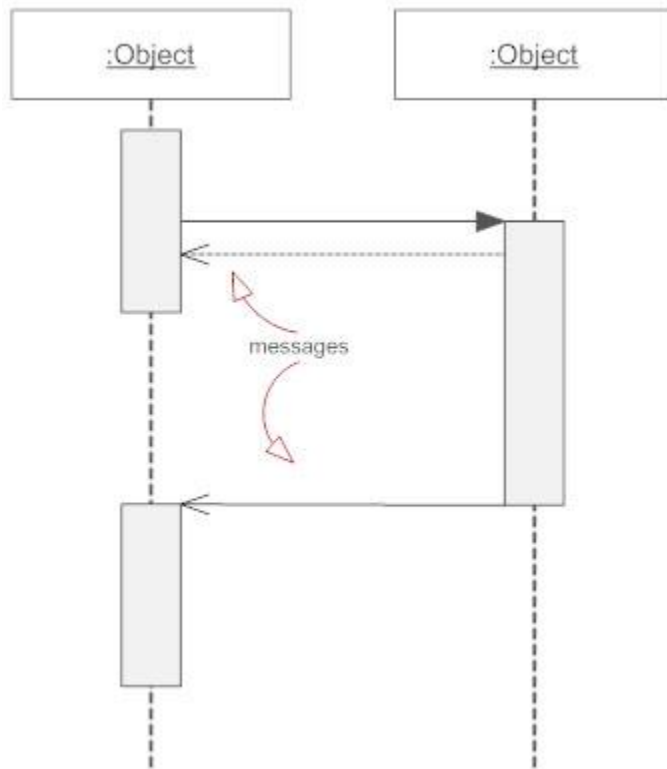
Activation boxes represent the time an object needs to complete a task. When an object is busy executing a process or waiting for a reply message, use a thin gray rectangle placed vertically on its lifeline.



Messages

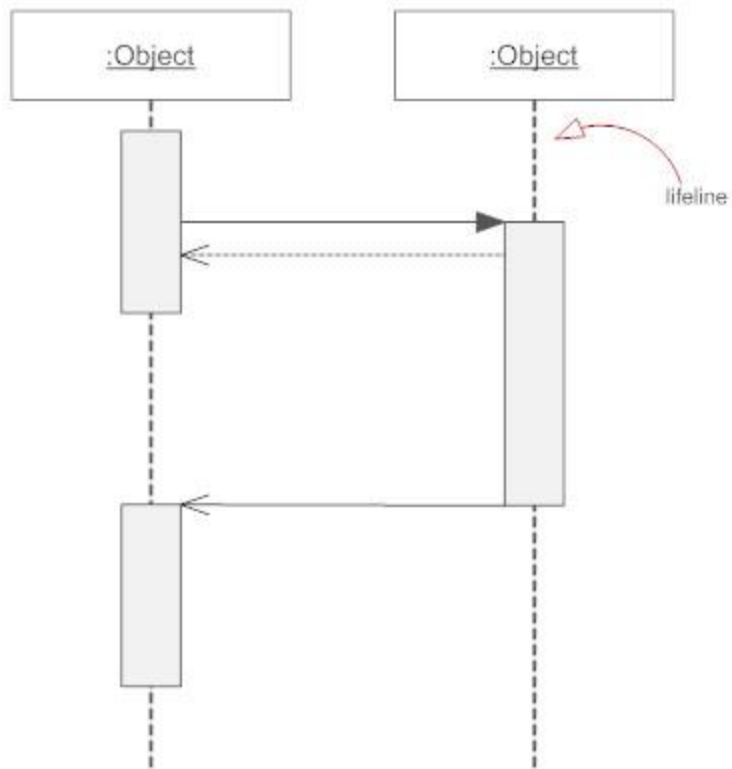
Messages are arrows that represent communication between objects. Use half-arrowed lines to

represent asynchronous messages. Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks. For message types, see below.



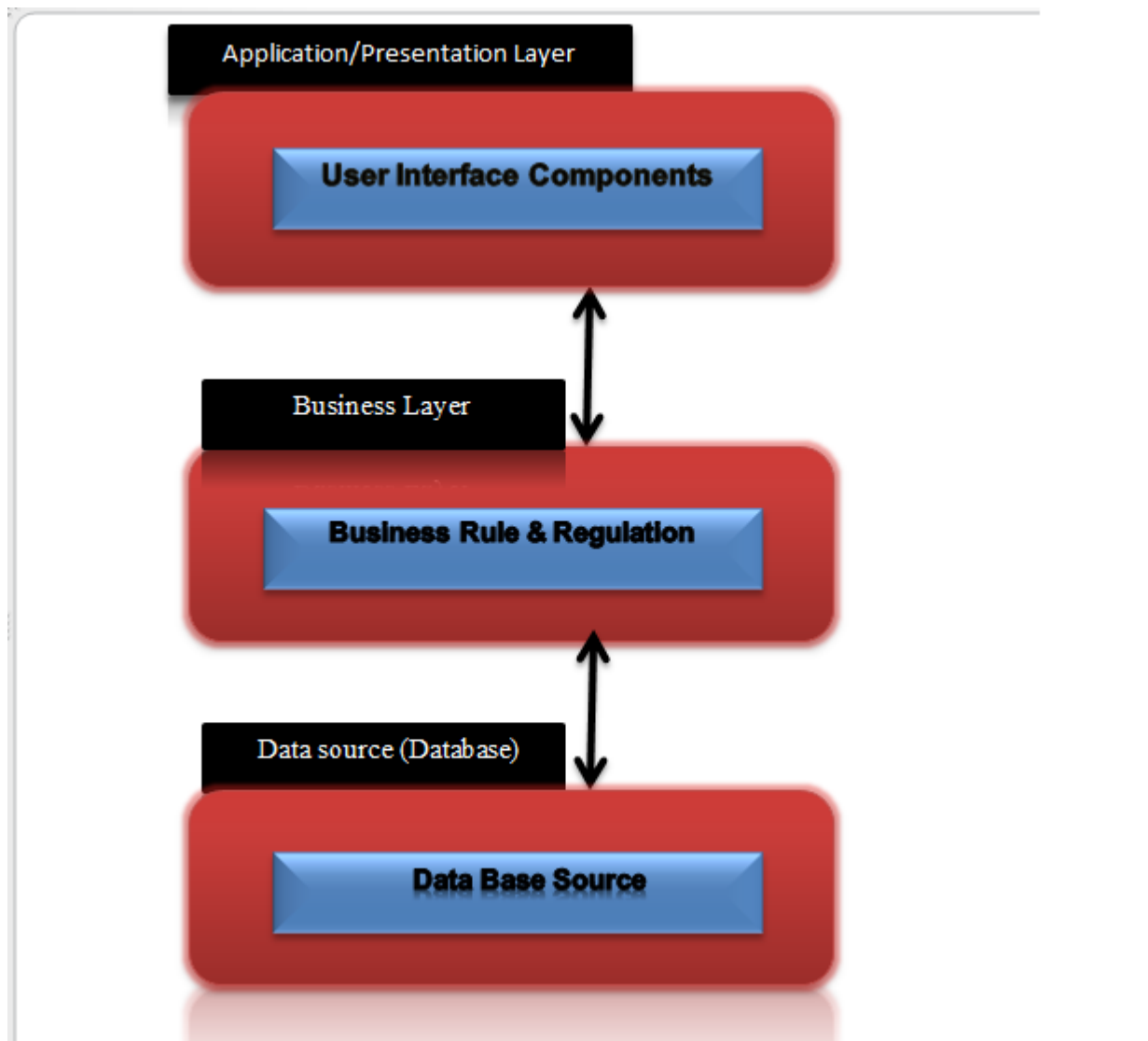
Lifelines

Lifelines are vertical dashed lines that indicate the object's presence over time.



4. Architecture Design Diagram

The architecture adopted is Three-tier web architecture. The architecture consists of three layers which are:



1. Presentation tier

Front End is designed in programming language like PHP, C# or Java etc...

2. Application tier

The middle dynamic content processing and generation level server layer is created in PHP, C# or Java etc...

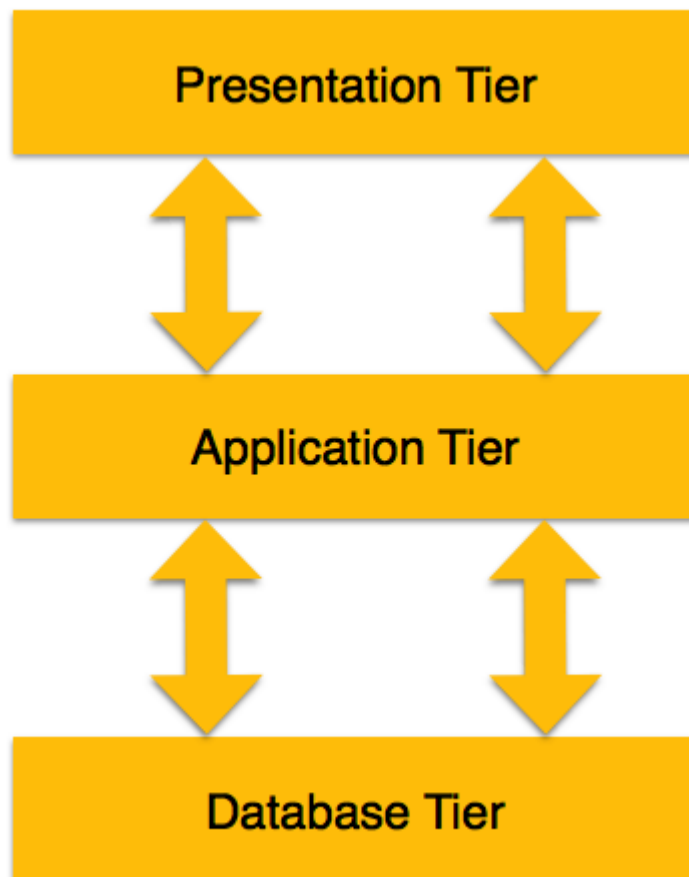
3. Data tier

SQL or MySQL is used as back-end database server.

SQL or MySQL is used as back-end database server.

3-tier Architecture

A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.

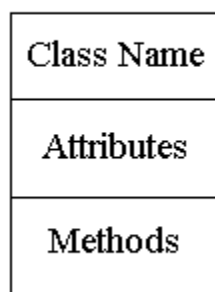


- **Database (Data) Tier** – At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.
- **Application (Middle) Tier** – At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.
- **User (Presentation) Tier** – End-users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.

5. Class Diagram

Class diagrams are used to represent real life objects in our design by capturing its data methods and relationships of these methods with each other. These classes are basic building blocks of our object oriented system. Class diagrams are represented with white boxes which consist of three parts. Each of these parts has unique characteristics.

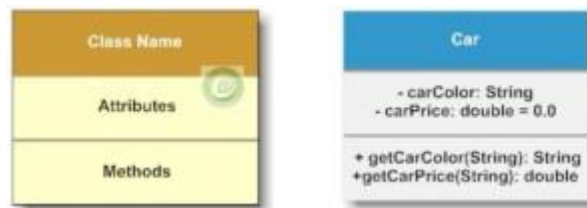
Syntax



The details of these parts are as follow.

1. The upper part contains the name of the class
2. The next part contains the attributes of the class.
3. The third part contains the methods and operations on the class.

Class Diagram Example



What is the Difference Between a Class and an Object?

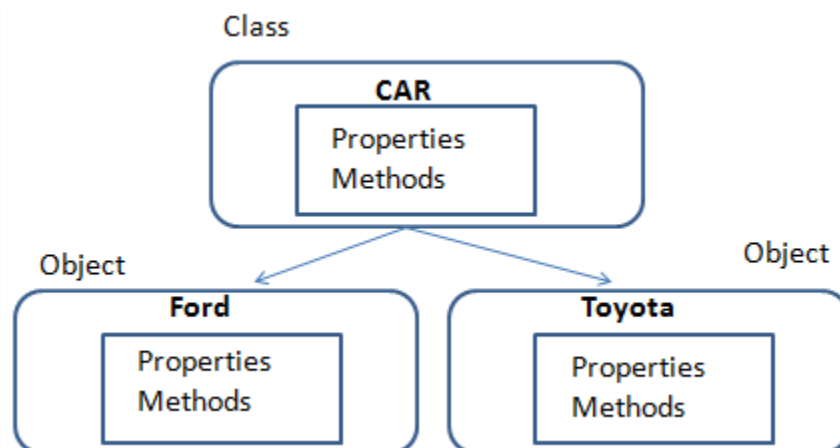
Classes and objects are separate but related concepts. Every object belongs to a class and every class contains one or more related objects.

So what exactly are classes and objects and what is the difference between them?

A Class is static. All of the attributes of a class are fixed before, during, and after the execution of a program. The attributes of a class don't change.

The class to which an object belongs is also (usually) static. If a particular object belongs to a certain class at the time that it is created then it almost certainly will still belong to that class right up until the time that it is destroyed.

An Object on the other hand has a limited lifespan. Objects are created and eventually destroyed. Also during that lifetime, the attributes of the object may undergo significant change.



class	objects
Human being	Gaurav, lalit , Utkarsh
Bird	Sparrow ,Eagle etc
Car	Accent,Corsa etc
Shape	Line,Rectangle,circle etc
user interface element	Menu,Toolbar,etc.
Complete hardware	Mouse , Keyboard,Printer

Class:

A class is the core of any modern Object Oriented Programming language. Class is a blueprint of an object that contains variables for storing data and functions to performing operations on these data.

Object:

Objects are the basic run-time entities in an object oriented system. They may represent a person, a place or any item that the program has to handle.

"Object is a Software bundle of related variable and methods. "

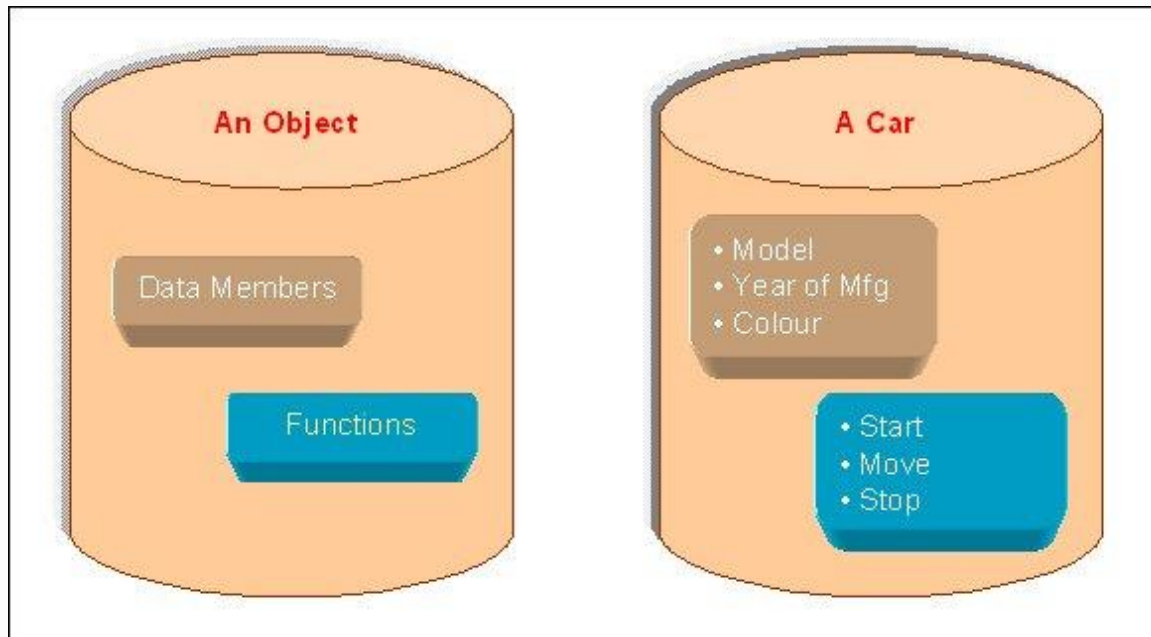
"Object is an instance of a class"

what are data member and functions

Instance of Object

An instance is an object in memory. Basically you create object and instantiate them when you are using them.

Instance: instance means just creating a reference (copy)



OOP : Abstraction, Encapsulation, Inheritance, Polymorphism

Abstraction is the “process of representing only essential features”. That means Abstraction doesn’t show the complexity behind features. Abstraction is used for “Making things more general, simpler, and abstract”.

Real Time Example for Abstraction:

The Best real time example for abstraction is ATM (Automated Teller Machine). We don't know how ATM internally works when we are using ATM, we know only select options like withdraw, Balance Inquiry, Mini Statement etc.

Here abstraction hides all unnecessary things, it shows only necessary things.

Encapsulation is defined as “Wrapping up data member and method together into a single unit”.

Real time example of Encapsulation is Car Driving, car driver knows how to start car by pressing start button. The driver doesn't know what happened inside when pressing start button. Here the starting process is hidden from driver. So this process can be called as “starting process is encapsulated from driver”

Polymorphism name itself tells “many forms”. That is Poly means “Many” morphism means “forms”. So Polymorphism meaning in Oop “one name. Or Polymorphism can also define as “same operation may behave differently on different classes”.

Real Time Example for Polymorphism

Real time example for polymorphism is “Door”, why mean we can use these doors for home, car, lift etc.

Inheritance is defined as “one class (child class) inherits or acquire the property (members) of another class”

Real time example for Inheritance is “parent child relationship” Because child receives all the property from parent.

Parent class : A class that is inherited from by another class. This is also called a base class or super class.

Child Class: A class that inherits from another class. This is also called a subclass or derived class

OOP: Association, Aggregation, and Composition

Association is a relationship where all objects have their own lifecycle and there is no owner.

Let's take an example of Teacher and Student. Multiple students can associate with single teacher and single student can associate with multiple teachers, but there is no ownership between the objects and both have their own lifecycle. Both can be created and deleted independently.

Aggregation is a specialized form of Association where all objects have their own lifecycle, but there is ownership and child objects can not belong to another parent object.

Let's take an example of Department and teacher. A single teacher can not belong to multiple departments, but if we delete the department, the teacher object will not be destroyed. We can think about it as a “has-a” relationship.

Composition is again specialized form of Aggregation and we can call this as a “death” relationship. It is a strong type of Aggregation. Child object does not have its lifecycle and if parent object is deleted, all child objects will also be deleted.

Let's take again an example of relationship between House and Rooms. House can contain multiple rooms - there is no independent life of room and any room can not belong to two different houses. If we delete the house - room will automatically be deleted.

Let's take another example relationship between Questions and Options. Single questions can have multiple options and option can not belong to multiple questions. If we delete the questions, options will automatically be deleted.

Association, Aggregation and Composition



Association – Loose form of relationship (Student can enroll in multiple Course, and A Course can have multiple Student)

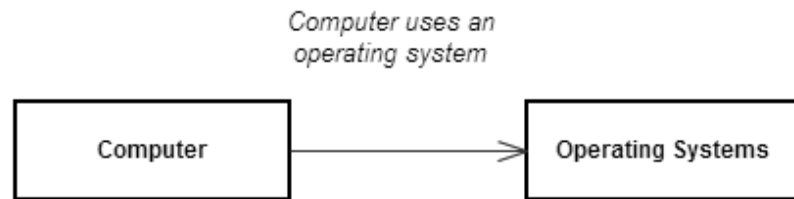
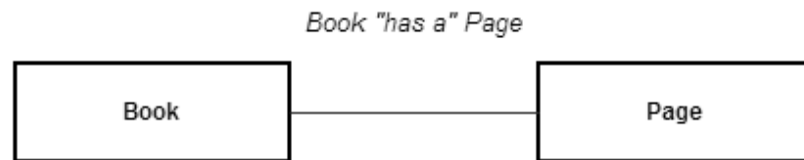
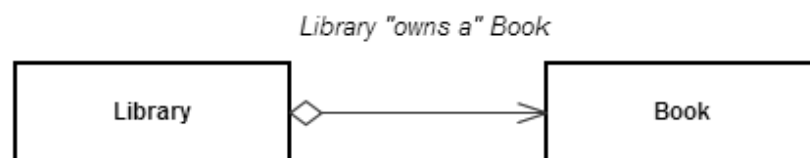
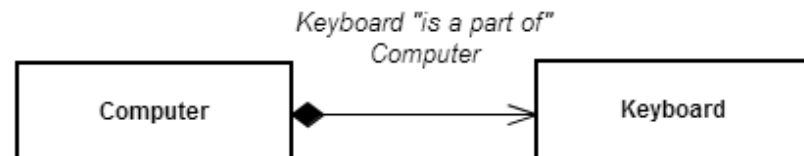


Aggregation - Whole part relationship. Part can exist without Whole. (Engine can exist even if Car is destroyed, the same Engine could be used in a different Car)



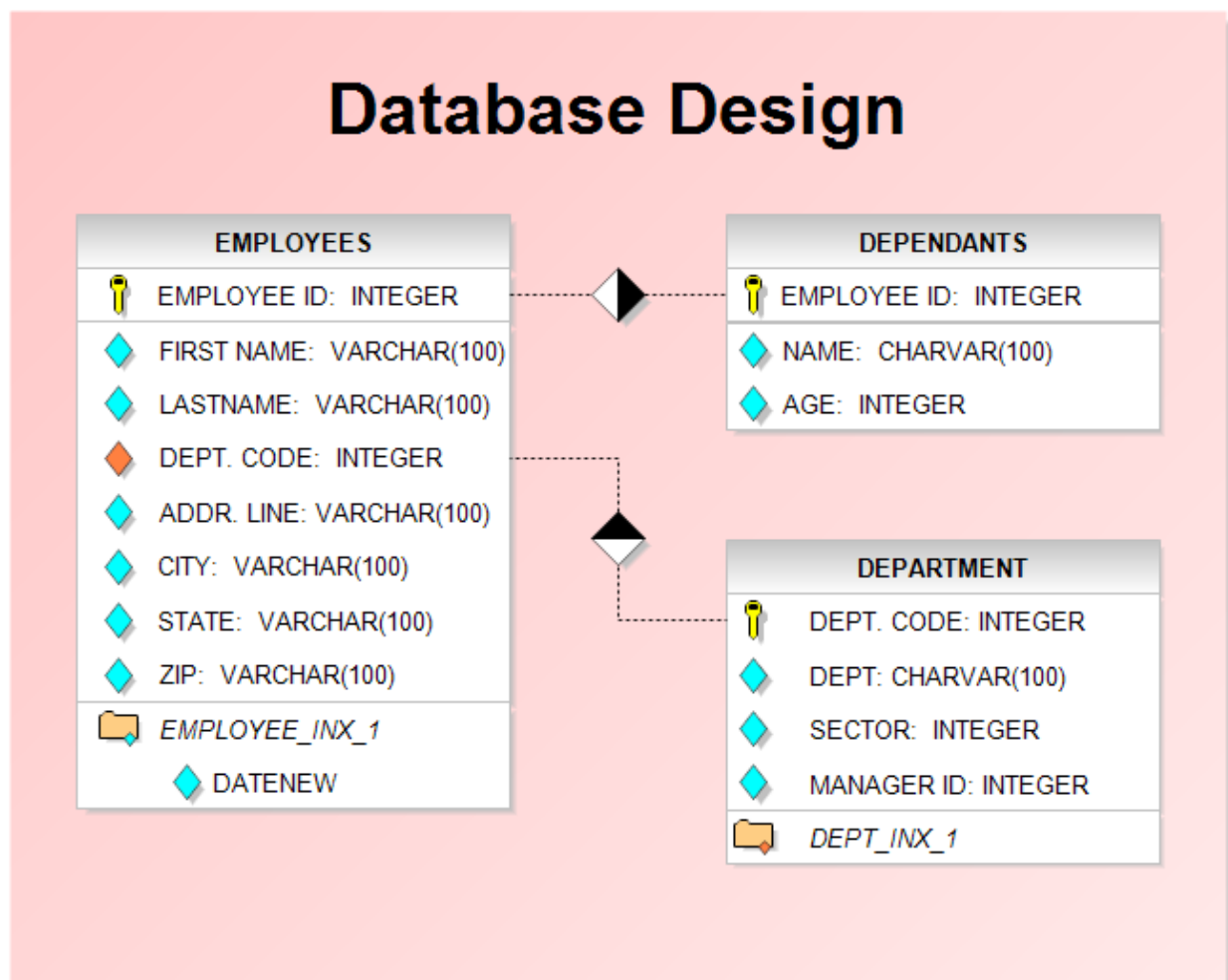
Composition – Stronger form of whole part relationship. Part can not exist without Whole. (OrderDetail can not exist if Order is deleted. If Order is deleted, OrderDetail also gets deleted)



Dependency**Association****Aggregation****Composition**

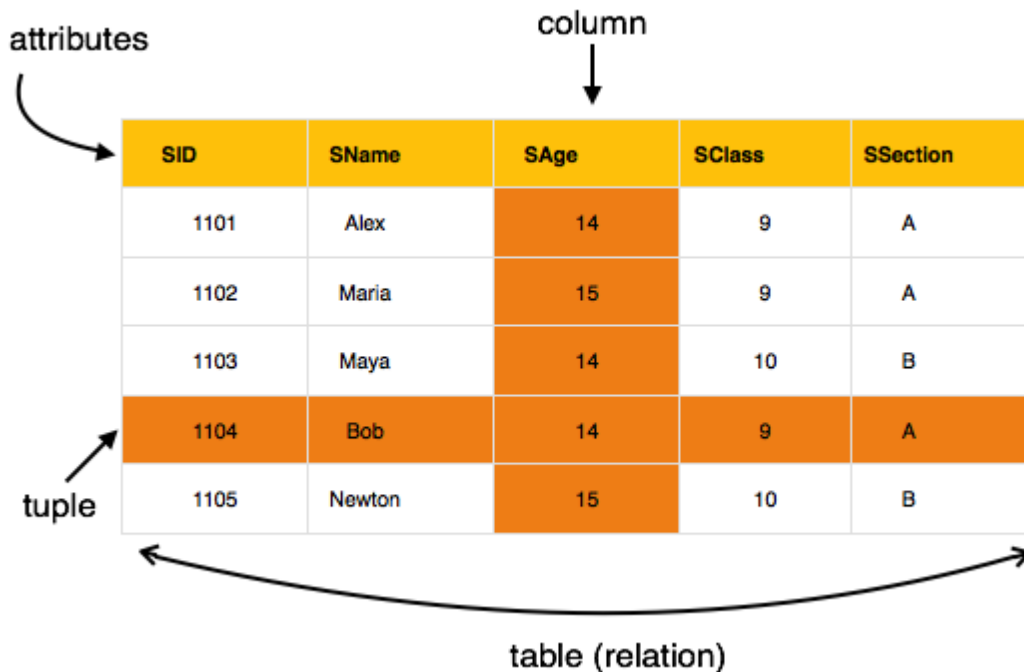
6. Database Design Diagram

In our daily life every object has some data associated with it. This data can be managed through data base design. Data large data can be represented by this data base design. Data base design can be represented in the form of physical, conceptual and logical.



Relational Model

The most popular data model in DBMS is the Relational Model. It is more scientific a model than others. This model is based on first-order predicate logic and defines a table as an n-ary relation.



The main highlights of this model are –

- Data is stored in tables called relations.
- Relations can be normalized.
- In normalized relations, values saved are atomic values.
- Each row in a relation contains a unique value.
- Each column in a relation contains values from a same domain.

Relation Data Model Concepts

Tables – In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represents records and columns represent the attributes.

Tuple – A single row of a table, which contains a single record for that relation is called a tuple.

Relation instance – A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

Relation schema – A relation schema describes the relation name (table name), attributes, and their names.

Relation key – Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

Attribute domain – Every attribute has some pre-defined value scope, known as attribute domain.

Database Management System or DBMS

Database Management System or DBMS in short refers to the technology of storing and retrieving users' data with utmost efficiency along with appropriate security measures. This tutorial explains the basics of DBMS such as its architecture, data models, data schemas, data independence, E-R model, relation model, relational database design, and storage and file structure and much more.

Database Management System: is the set of programs or system which is used to create and maintain database.

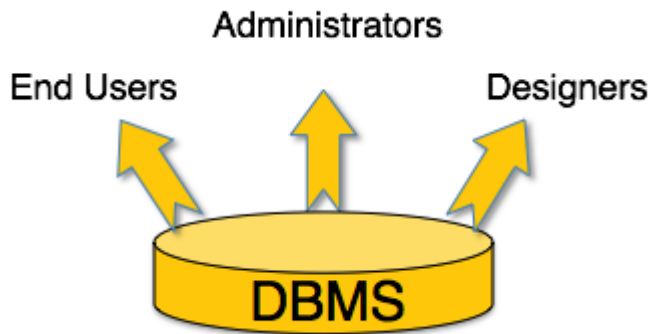
Data: is the collection of raw, facts and figures like college admission form consists of data.

Database: is organized collection of related data.

A database management system stores data in such a way that it becomes easier to retrieve, manipulate, and produce information.

Users

A typical DBMS has users with different rights and permissions who use it for different purposes. Some users retrieve data and some back it up. The users of a DBMS can be broadly categorized as follows –



- **Administrators** – Administrators maintain the DBMS and are responsible for administrating the database. They are responsible to look after its usage and by whom it should be used. They create access profiles for users and apply limitations to maintain isolation and force security. Administrators also look after DBMS resources like system license, required tools, and other software and hardware related maintenance.
- **Designers** – Designers are the group of people who actually work on the designing part of the database. They keep a close watch on what data should be kept and in what format. They identify and design the whole set of entities, relations, constraints, and views.
- **End Users** – End users are those who actually reap the benefits of having a DBMS. End users can range from simple viewers who pay attention to the logs or market rates to sophisticated users such as business analysts.

Keys

Super Key

Super key is a set of one or more than one keys that can be used to identify a record uniquely in a table. Example: Primary key, Unique key, Alternate key are subset of Super Keys.

Candidate Key

A Candidate Key is a set of one or more fields/columns that can identify a record uniquely in a table. There can be multiple Candidate Keys in one table. Each Candidate Key can work as Primary Key.

Example: In below diagram ID, RollNo and EnrollNo are Candidate Keys since all these three fields can be work as Primary Key.

Primary Key

Primary key is a set of one or more fields/columns of a table that uniquely identify a record in database table. It can not accept null, duplicate values. Only one Candidate Key can be Primary Key.

Alternate key

A Alternate key is a key that can be work as a primary key. Basically it is a candidate key that currently is not primary key.

Example: In below diagram RollNo and EnrollNo becomes Alternate Keys when we define ID as Primary Key.

Composite/Compound Key

Composite Key is a combination of more than one fields/columns of a table. It can be a Candidate key, Primary key.

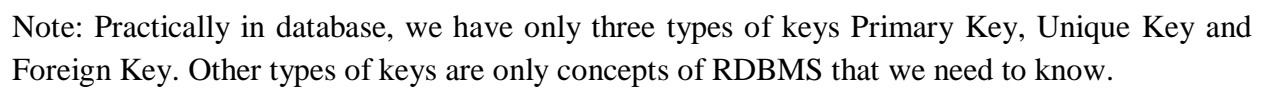
Unique Key

Uniquekey is a set of one or more fields/columns of a table that uniquely identify a record in database table. It is like Primary key but it can accept only one null value and it can not have duplicate values. For more help refer the article Difference between primary key and unique key.

Foreign Key

Foreign Key is a field in database table that is Primary key in another table. It can accept multiple null, duplicate values. For more help refer the article Difference between primary key and foreign key.

Example : We can have a DeptID column in the Employee table which is pointing to DeptID column in a department table where it a primary key.



Interface diagram are used to represents Graphical User Interface (GUI) of our system. GUI consists of Buttons, text fields and check boxes etc.

ان یٹ لفظ:	باتونی													
سابقہ:	یا													
راس:	تو													
لاحقہ:	نی													
اردو تختی														
ا	آ	ب	پ	ت	ٹ	ث	ج	چ	ح	خ	د	ڈ	ذ	ر
ژ	ز	س	ش	ص	ض	ط	ظ	ع	غ	ف	ق	ک	گ	
ل	م	ن	و	ؤ	ہ	ھ	ء	ی	ئی	ے	-	؟		
زیر	زیر	پیش	دو	دو	درمیانی	کھڑی	تشدید	کوما						
			زیر	زیر	حمزہ	زیر								

8. Test Cases:

A test case in software engineering is a set of circumstances or variables under which a tester will determine whether a software system is working according to implementation and giving expected results or not. It may take many test cases to determine that a software system is functioning correctly. When test case are documented then referred to as test scripts.

Test Case Title	Add User
Test Case Id	1
Description: This test case is about adding a new user to existing system with the privileges defined at time of user account creation.	
Pre Conditions: 1. All must-required information about the new user should be available. 2. Database should be available in online mode.	
Task Sequence	Results
1. Administrator opts to add a new user account.	Pass/Fail
2. System asks for necessary information.	
3. Administrator provides all the required information and opts to complete the operation.	
4. There is a problem in the data provided; some data needs to be corrected. – Administrator checks the available information and corrects the error. – Administrator continues from the step 3.	
5. System after confirmation adds the new account.	
6. System sends the account creation email to the administrator’s email id and user’s email address.	
Post Conditions: – A new user account is successfully created.	
Unresolved issues:	
Authority: Administrator	
Modification history: 1.0	
Author: <Project or Group ID>	
Description:	