DATABASE MANAGEMENT SYSTEM

MODULE - I

Chapter - 1

→ Database is a collection of related data and data is a collection of facts and figures that can be processed to produce information.

Example:- If we have data about marks obtained by all students, we can then conclude about toppers and average marks.

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

Functions of a database Management System:-

1) Storing and Retrieving Data:

i) The database sees the Physical view of the data.
→ How the data are compressed and formatted, which data are stored near each other, and which indexes are created to simplify and speed finding data on the storage medium.

ii) The database presents a logical view to the users and programmer.
→ It organizes and presents data elements in ways that managers and other users find helpful.

2) Managing Metadata:

→ Metadata are data about data in the database. (OR) The descriptive information is stored by the database management system (DBMS) in the form of a database catalog or dictionary.

Example: The fact that a company's invoice numbers are six digits long, with the first digit being either a 1 or 3.

3) Limiting and controlling Redundant Data in multiple systems:

→ companies often collect and store the same data in two or more different information systems.

4) Supporting Simultaneous Data Sharing:

Concurrency control

→ It describes the proper management of simultaneous attempts to update a database by multiple users or multiple software programs.

5) providing Transaction Atomicity

→ The concept that a transaction cannot be split into smaller parts.

→ If systems fails in midst of transaction, system is restored as if the entire transaction completed or no part of it completed.

6) Providing Backup and Recovery Services:

→ Databases cannot be backed up like files because they are too large and in constant use.

Backup techniques include -

* Operating in parallel on two storage devices.
* Use of temporary database during backup.

7) Providing Authorization and Security Services:

→ Most DBMS can limit who has access to specific data.

→ DBMS can create limited views of data so that users can see only what they are authorized to see.

8) Enforcing Business Rules:

→ A DBMS enforces rules that ensure related data are logically consistent.

Characteristics of the Database:

→ Traditional file processing.

→ Each user defines and implements the files needed for a specific software application.

→ Database approach.

→ Single repository maintains data.

Main Characteristics of database approach:

1) Self-describing nature of a database system:-

→ A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete defination or description of the data base structure and constraints.

→ The information is stored in the catalog is called meta-data.

→ The catalog is used by the DBMS software and also by database users who need information about the database structure.

→ DBMS software can access diverse databases by extracting the database definations from the catalog and using these definations.

2) Insulation between programs and data, and data abstraction:

→ In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access that file. By contrast, DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs. We call this property program-data independence.

→ An operation (function or method) is specified in two parts. The interface of an operation includes the operation name

and the data types of its arguments (or parameters). The implementation of the operation is specified seperately and can be changed without affecting the interface. User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed as program-operation independence.

→ The characteristic that allows program-data independence and program-operation independence is called data abstraction.

→ A DBMS provides users with a conceptual represention of data that doesnot include many of the details of how the data is stored or how the operations are implemented.

→ Informally, a datamodel is a type of data abstraction that is used to provide this conceptual representation.

3) Support of multiple view of the data:-

→ A database typically has many types of users, each of whom may require a different perspective or view of the database. A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not exactly explicitly stored. Some users may not need to be aware of whether the data they refer to is stored or derived. A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views.

4) Sharing of data and multiuser transaction processing:-

→ The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.

→ The concept of transactions has become central to many database applications. A transaction is an executing program or process that includes one or more data accesses, such as reading. Each transaction is supposed to execute a logically correct database access if executed in it's entirely without interference from other transaction.

→ The DBMS must enforce several transaction properties:

* The isolation property ensures that each transaction appears to execute in isolation from other transactions, even though hundreds of transactions may be executing concurrently.

* The atomicity property ensures that either all the database operations in a transaction are executed or none are.

Advantages of Using the DBMS Approach:-

1) Improved strategic use of corporate data:-

→ Accurate, complete, up-to-date data can be made available to decision makers where, when and in the form they need it.

2) Reduced data redundancy:

→ The database approach can reduce or eliminate data redundancy. Data is organized by the DBMS and stored in only one location. This results in more efficient utilization of system storage space.

3) Improved data integrity:

→ With the traditional approach, some changes to data were not reflected in all copies of the data kept in seperate files. This is prevented with the database approach because there are no seperate files that contain copies of the same peice of data.

4) Easier modification and updating :

→ With the database approach, the DBMS coordinates updates and data modifications. Programmers and users do not have to know where the data is physically stored. Data is stored and modified once. Modification and updating is also easier because the data is stored at only one location in most cases.

5) Data and program independence :-

→ The DBMS organizes the data independently of the application program. With the database approach, the application program is not affected by the location or type of data. Introduction of new data types not relevant to a particular application does not require the rewriting of that application to maintain compatability with the data file.

6) Better access to data and information :-

Most DBMSs have software that makes it easy to access and retrieve data from a database. In most cases, simple commands can be given to get important information. Relationships between records can be more easily investigated and exploited, and applications can be more easily combined.

7) Standardization of data access :-

A primary feature of the database approach is a standardized, uniform approach to database access. This means that the some overall procedures are used by all application programs to retrieve data and information.

8) A framework for program development :-

Standardized database access procedures can mean more standardization of program development. Because programs go through the DBMS to gain access to data in the database. Standardized database access can provide a consistent framework for program development. In addition, each application program

need address only the DBMS, not the actual data files, reducing application development time.

9) Better overall protection of the data :-
The use of and access to centrally located data are easier to monitor and control. Security codes and passwords can ensure that only authorized people have access to particular data and information in the database, thus ensuring privacy.

10) Shared data and information resources :-
The cost of hardware, software and personnel can be spread over a large number of applications and users. This is a primary feature of DBMS.

History of Database Applications :-

1) Early Database Applications using Hierarchial and Network Systems:
many early database applications maintained records in large organizations, there were large number of records of similar structure. Ex:- University application for a student. the main problems with early database systems was the ~~conceptual~~ intermixing of conceptual relationships with physical storage and placement of records on disk. These systems do not provide sufficient data abstraction and program-data independence capabilities. One more problem was it is designed for original queries and transactions that was designed to handle, if any new queries and transaction it cannot handle, no flexibility. Another shortcoming of early system was that they provided only programming language interfaces, this made it time-consuming & expensive. The systems were based on : heirarchial, network model-based and inverted file systems.

2) Providing Data Abstraction and Application Flexibility with Relational Databases:
Relational databases were used to seperate the physical storage of data from it's conceptual representation and to provide a mathematical foundation for a data representation and querying. The

relational data model also introduced high-level query languages that provided an alternative to programming language interfaces, making it much faster to write new queries. Hence data abstraction and program data indepence were much improved when compared to earlier. These RDBMS were slow since they did not use physical storage pointers or record placement to access related data records. Eventually, relational database became dominant type now exist in all types of computer.

3) Object-Oriented Applications and the need for more complex databases :-

OO programming language emergence and the need to store and share complex, structured objects led to development of OODBS's. They were competition for RDB's, since they provided more data structures. The complexity of the model and the lack of an early standard contributed to their limited use, their overall penetration into database products market remains low. But many OOC were incorporated to newversion of relational DBMS. called as 'ORDBM's.

4) Interchanging Data on the web for E-commerce using XML

WWW provides a large network of interconnected computers so users can create static web pages using HTML, and store these documents on web servers, the documents can be linked through hyperlinks. E-commerce was emerged that time much of the critical information was gathered from the DBMS, such as flight information, product price. The extended markup language (XML) is used for interchanging data among various types of databases and web pages.

5) Extending Database for new capabalities

* Scientific application like to store large data from science experiments.
* Storage and retreival of images, storage and retrieval of vedios, such as movies, and medical purposes like x-ray, MRI.
* Data mining - credit card fraud detection
* Spatial applications - weather information, geographical information
* Time series applications - daily sales and montly gross figures.

Chapter - 2

Data models:

A set of concepts to describe the structure of a database, and certain constraints that the database should obey.

Categories of Data Models:

→ High level or conceptual data models provide concepts that are close to the way many users perceive data.

* Conceptual data models use concept such as entities, attributes and relationships.
* An entity represents a real world object or concept.
* An attribute represents some property of interest that further describes an entity.
* A relationship among two (or) more entities represents an association among the entities.

→ Low-Level or physical data models provide concepts that describe the details of how data is stored on the computer (storage media, typically magnetic disks.

* physical data models describe how data is stored as files in the computer by representing information such as record formats, record orderings, and access paths.
* An access path is a structure that makes the search for particular database records efficient.
* An index is an example of an access path that allows direct access to data using an index term or a keyword.

→ Representational or implementational data models provide concepts that may be easily understood by end users but that are not too far removed from the way data is organized in computer storage.

* These are the models used most frequently in traditional commercial DBMSs. These include the widely used relational data model, as well as the so-called legacy data models.
* Representational data models represent data by using record structures and hence are sometimes called record-based data models.

Schemas, Instances, and Database State:-

→ Database Schema

The description of a database. Includes descriptions of the database structure and the constraints that should hold on the database.

* A displayed schema is called schema daigram, each object is a schema construct and it displays some aspects of schema some types of constraints, many types are not represented in schema diagram.

→ The data in the database at a particular moment in time is called a database state. It is also called the current set of instances in the database.

→ The distinction between Database schema and database state is very important. When we define a new database, we specify its database we specify its database schema only to the DBMS. At this point, the corresponding database state is empty state with no data. We get the initial state of the database when the database is first populated or loaded with the initial data.

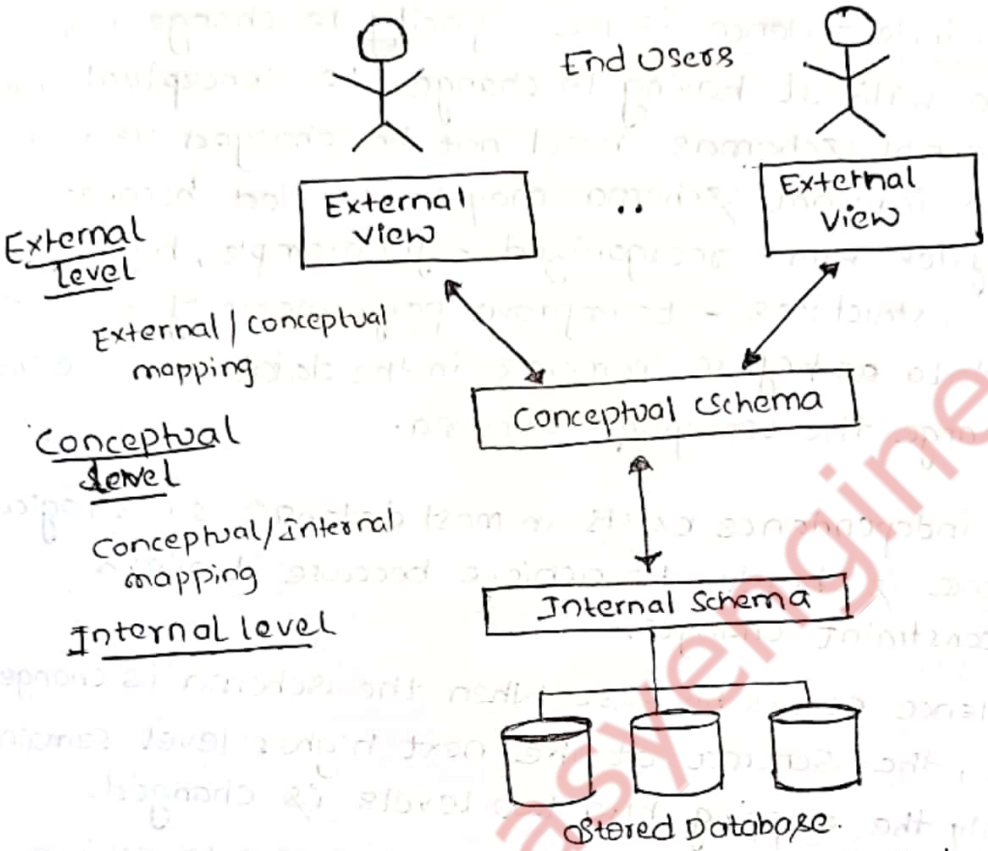
→ the DBMS is partly responsible for ensuring that every state of database is a valid state, the schema is called intension sometimes, and a database state is called extension of the schema.

Three-Schema Architecture:

→ The internal level has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.

→ The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, and constraints. A representational data model is used to describe the conceptal schema when database is implemented.

→ The external or view level includes a number of external schemas. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. As in previous level, each external schema is typically implemented using a representational data model, possibly based on external schema design in a high-level data model.



→ The three-schema architecture is a convenient tool with which the user can visualize the schema levels in a database system. most DBMSs do not separate the three levels completely and explicitly, but support the three-schema architecture to some extent.

→ In most DBMSs that support user views, external schemas are specified in the same data model but some allows different data models.

→ The three schemas are only descriptions of data. the stored data is at physical level only, so DBMS must transform request specified on an external schema into a request against conceptual schema, and then into a request on internal schema for processing over stored database.

→ The processes of transforming requests and results between levels are called mappings.

Data Independence :-

→ Logical data independence is the capacity to change the conceptual schema without having to change external schemas (or) application programs. We may change the conceptual schema to expand the database to change constraints, or to reduce the database.

→ Physical data independence is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well. Changes to the internal schema may be needed because some physical files were reorganized - for example, by creating additional access structures - to improve performance of retrieval. If the same data as before remains in the database, we should not have to change the conceptual schema.

→ physical data independence exists in most database since logical data independence is harder to achieve because it allows structural and constraint changes.

→ Data independence occurs because when the schema is changed at some level, the schema at the next higher level remains unchanged; only the mapping b|w two levels is changed.

→ The three-schema architecture can make it easier to achieve true data independence, both physical and logical. However, the two levels of mappings create an overhead during compilation of a program, leading inefficiencies in DBMS. Because of this, few DBMSs have implemented the full three-schema architecture.

Database Languages -

→ In many DBMSs where no strict seperation of levels is maintained one language called data defination language (DDL) is used by DBL and data designers. The DBMS will have a DDL compiler whose function is to process DDL statements in order to identify descriptions of the schema constructs and to store schema description in DBMS catalog.

→ In DBMSs (seperation is maintained between the conceptual and internal levels, DDL is for conceptual level only, the storage defination language (SDL) for internal levels, the mapping b/w two schema is specified in one of these languages. In most relational database today, no specific language performs the role of SDL.

→ For a true three-schema architecture, we would need a third language, the view defination language (VDL), to specify user views and their mappings to the conceptual schema, but in most DBMSs the DDL is used to define both conceptual and external schemas.

→ Typical manipulations include retrieval, insertion, deletion and modification of the data. The DBMS provides a set of operations or a language called data manipulation language (DML) for these purposes.

→ In current DBMSs, a comprehensive integrated language is used, for example - A comprehensive database language is the (SQL relational database language which represents DDL, VDL, DML, as well as statements for constraint specification and other features. SDL was early versions of SQL but has been removed to keep it at conceptual and external levels only.

→ There are two types of DML's -

* A high level or non-procedural DML can be used on its own to (specify complex database operations concisely. Many DBMSs allow high-level DML (statements either to be entered interactively from a display monitor or terminal. In the latter case, DML (statements must be identified within the program (so that they can be extracted by a pre-compiler and processed by the DBMS. High level DML's, (such as (SQL, can (specify and retrieve many records in a single DML (statement; therefore, they are called set-at-a-time or set-oriented DMLs. A query in a high-level DML often (specifies which data to retrieve rather than how to retrieve it; therefore, (such languages ~~such~~ are called declorative.

* A low-level or procedural DML must be embedded in a general-purpose programming language. This type of DML typically retrieves individual records or objects from the database and processes each seperately. Therefore, it needs to use programming language constructs, such as looping, to retrieve and process each record from a set of records. It is also called record-at a-time DMLs because of this property.

→ Whenever DML commands, whether high level or low level, are embedded in a general-purpose programming language, that language is host ~~query~~ language and DML is data sub language. A high level DML used in a standalone interactive manner is called a query language.

DBMS interfaces:-

→ Menu-Based Interfaces for Web clients (or) Browsing:

* Menus do away with the need to memorize the specific commands and syntax of a query language; rather, the query is composed step-by-step by picking options from a menu that is displayed by the system.

→ Forms-Based Interfaces:

* Forms are usually designed and programmed for naive users as inter-faces to canned transactions.

* Many DBMSs have forms specification languages, which are special languages that help programmers specify such forms. SQL Forms is a form-based language that specifies queries using a form designed in conjunction with the relational database schema.

→ Graphical User Interfaces:

* A GUI typically displays a schema to the user in diagrammatic form.

→ Natural language Interfaces:

* These interfaces accept requests written in english (or) some other language and attempt to understand them. A natural language interface usually has its own schema, which is similar to the database conceptual schema, as well as a dictionary of important words.

* The natural language interface refers to the words in its schema, as well as to the set of standard words in its dictionary, that are used to interpret the request.

→ Speech input and Output:-

* The speech input is detected using a library of predefined words and used to set up the parameters that are supplied to the queries.

* For output, a similar conversion from text or numbers into speech takes place.

→ Interfaces for Parametric Users:

* Usually a small set of abbreviated commands is included, with the goal of minimizing the number of keystrokes required for each request.

→ Interfaces for the DBA:

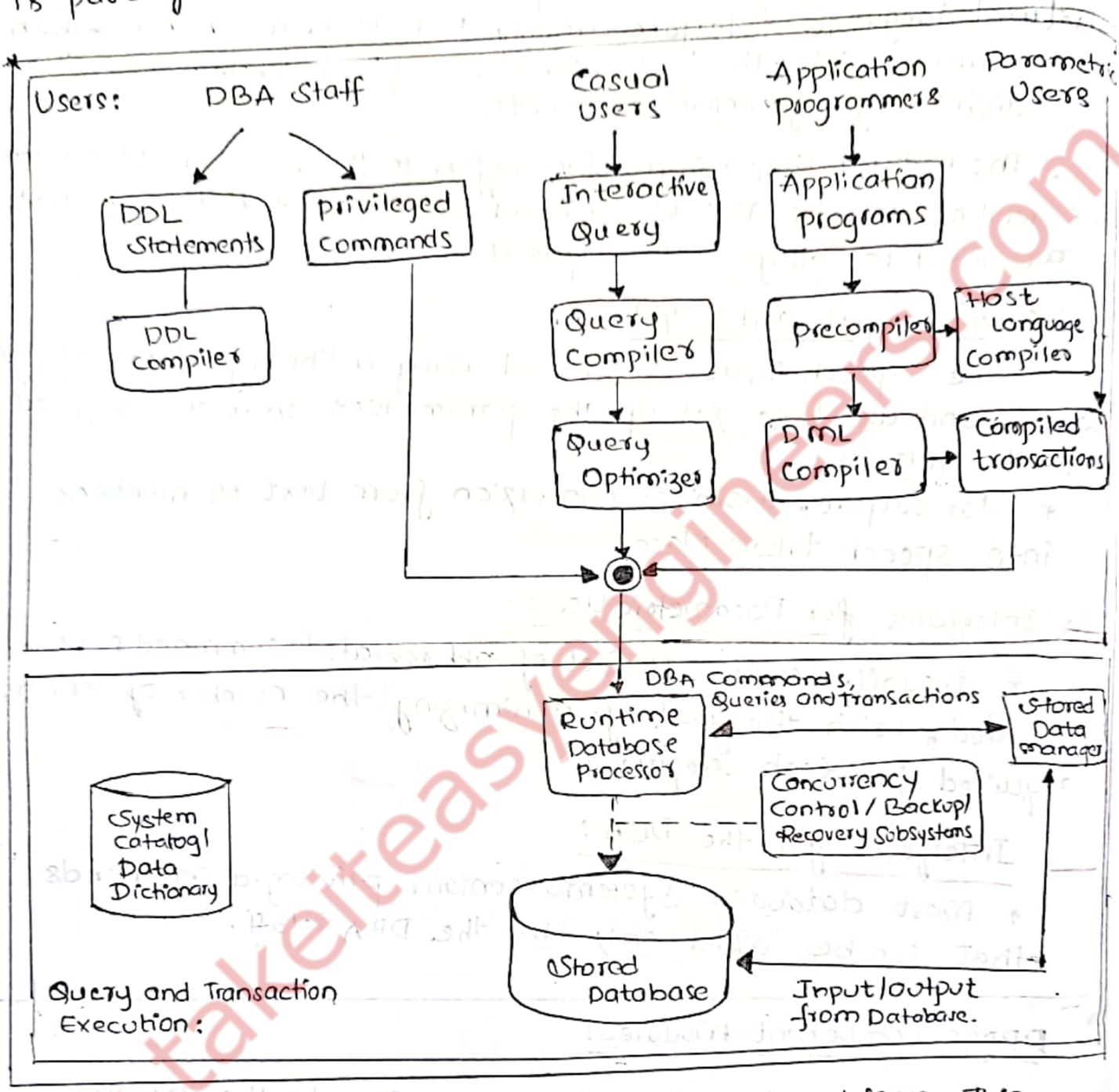
* Most database systems contain privileged commands that can be used only by the DBA staff.

DBMS Component Modules:

* The top part of the below figure refers to the various users of the database environment and their interfaces. The lower part shows the internals of the DBMS responsible for storage of data and processing of transactions.

* Access to the disk is controlled primarily by the OS, which schedules disk read/write. Many DBMSs have their own buffer management module to schedule disk read/write, because this has a considerate effect on performance. Reducing disk read/write improves performance considerably. A higher-

level stored manager module of the DBMS controls access to DBMS information that is stored on disk, whether it is part of database or the catalog.



* A query compiler compiles queries into an internal form. This internal query is subjected to query optimization. The query optimizer is concerned with the rearrangement and possible reordering of operations and indexes during execution. It consults the system catalog for statistical and other physical information about the stored data and generates executable code that performs the necessary operations for the query and makes calls on the runtime processor.

* The pre-compiler extracts DML commands from an application program written in a host programming language. These commands are sent to the DML compiler for compilation into object code for database access. The rest of the program is sent to the host language compiler and they object codes for the DML commands and rest of the program are linked, forming a canned transaction whose executable code includes calls to the run time database processor. Canned transactions are executed repeatedly by parametric users. Each execution is considered to be a seperate transaction.

* The run time database processor executes
  - The privileged commands
  - The executable query plans
  - The canned transaction with runtime parameters.

  It works with the system catalog and may update it with statistics. It also works with the stored data manager, which in turn uses OS services for carrying out R/W operations b/w disk and main memory.

* Concurrency control and backup and recovery systems seperately as a module. they are integrated into the working of the runtime database processor for purposes of transaction management.

* The DBMS interacts with OS when disk accesses - to the database or to the catalog - are needed. If computer is shared by many users, the OS will schedule DBMS disk access requests and DBMS processing along with other processes. If computer system is mainly dedicated to running database server, the DBMS will control main memory buffering of disk pages.

Database System Utilities:

* DBMSs have database utilities that help the DBA manage the database system. Common utilities have following functions

• Loading: A loading utility is used to load existing data files - such as text files in to database. The current format of the data files and desired databas file structure are specified to the utility, which then automatically re-formats the data and stores it in the database. With the proliferation of DBMSs, transferring data is common in organizations. Some offering products that generate appropriate loading programs, given the existing source and internal schemas. Such tools are called conversion tools.

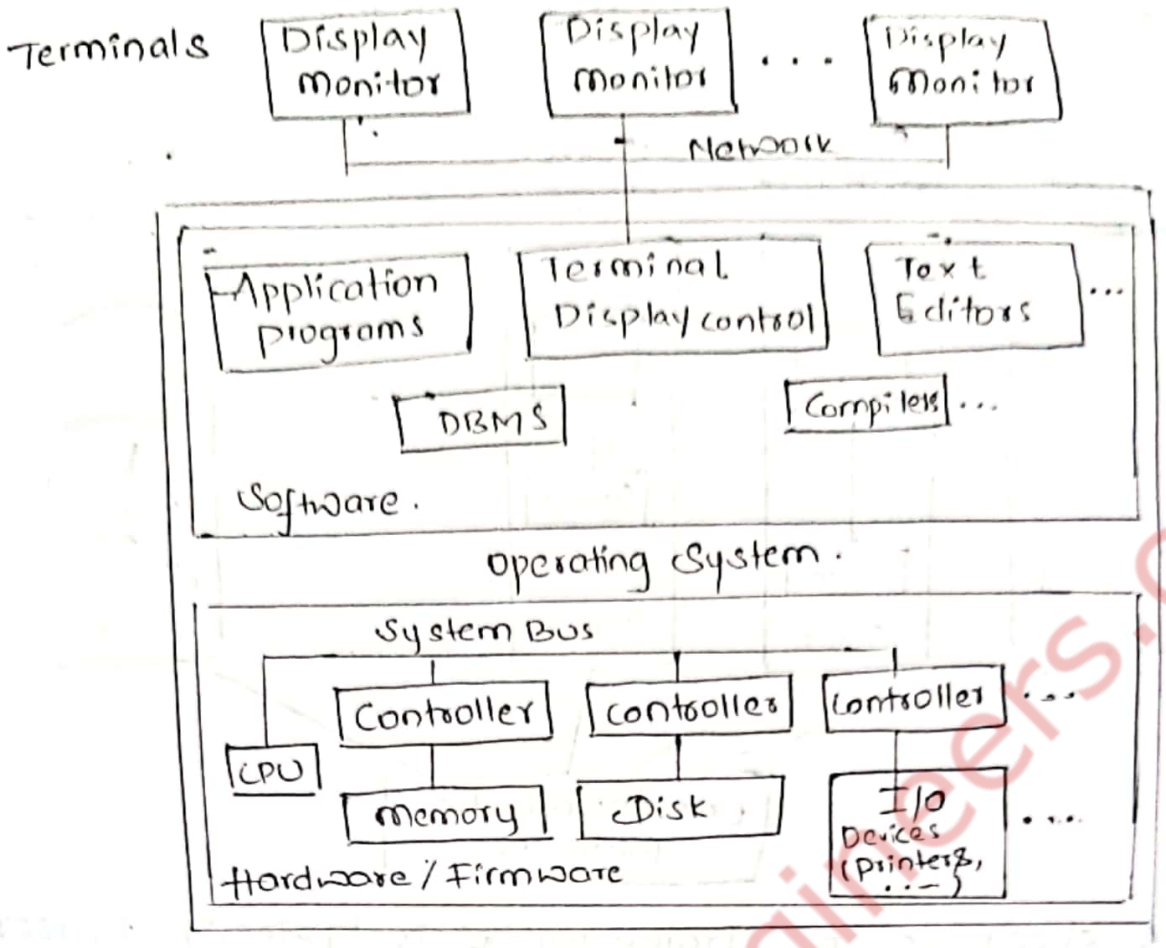
• Backup: A backup utility creates a backup copy of the database, usually by dumping the entire database onto tape or other mass storage medium. Incremental backup is more complex, but saves storage space.

• Database storage reorganization: This utility can be used to reorganize a set of database files into different file organisations, and create new access paths to improve performance.

• Performance monitoring: Such a utility monitors database usage and pro-vides statistics to the DBA. The DBA uses the statistics in making decisions such as whether (or) not to reorganize files or whether to add or drop index to improve performance.

Centralized and Client/Server Architecture:

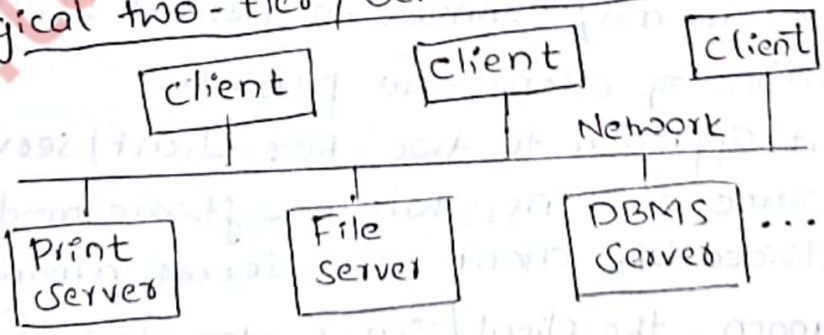
* All the DBMS functionality, application program execution, and user inter-face processing were carried out on one machine.
* As prices of hardware declined, most users replaced their terminals with PCs and workstations and now mobile. At first, database systems used these computers similarly to how they had used display terminals, so that the DBMS itself was centralized.



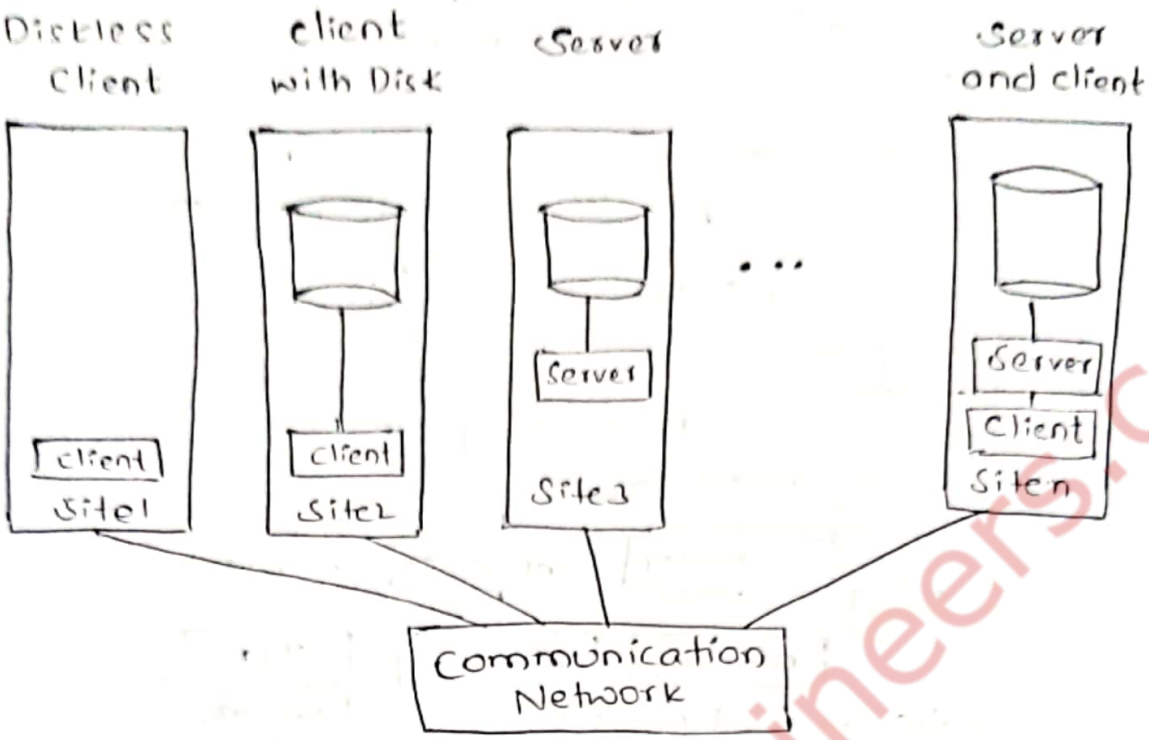
Basic client / server Architecture:

* The client/server architecture was developed to deal with computing environments in which a large no. of PC's, webservers etc.
* The idea is to define specialized servers with specific functionalities.
* The client machines provide the user with the inappropriate interfaces to utilize these servers, as well as with local processing power to run local applications.

* Logical two-tier / server architecture:



physical two-tier client/server architecture:



* A client in this frame work is typically a user machine that provides user interface capabilities and local processing. When a client requires access to additional functionality such as database access that does not exist at that machine, it connects to a server that provides the needed functionality.
* A server is a system containing both hardware and software that can provide services to the client machines.

Two-Tier client/server Architectures

* A client program may connect to several DBMSs.
* other variations of clients are possible.
* the different approach to two-tier client/server architecture was taken by some OODBMSs, where software modules ~~are~~ of DBMS are divided b/w client and server. in more integrated way
* In this approach, the client/server interaction is more tightly coupled and is done internally by the DBMS modules.
* The server has been called a data server bcz it provides data in disk pages to the client.

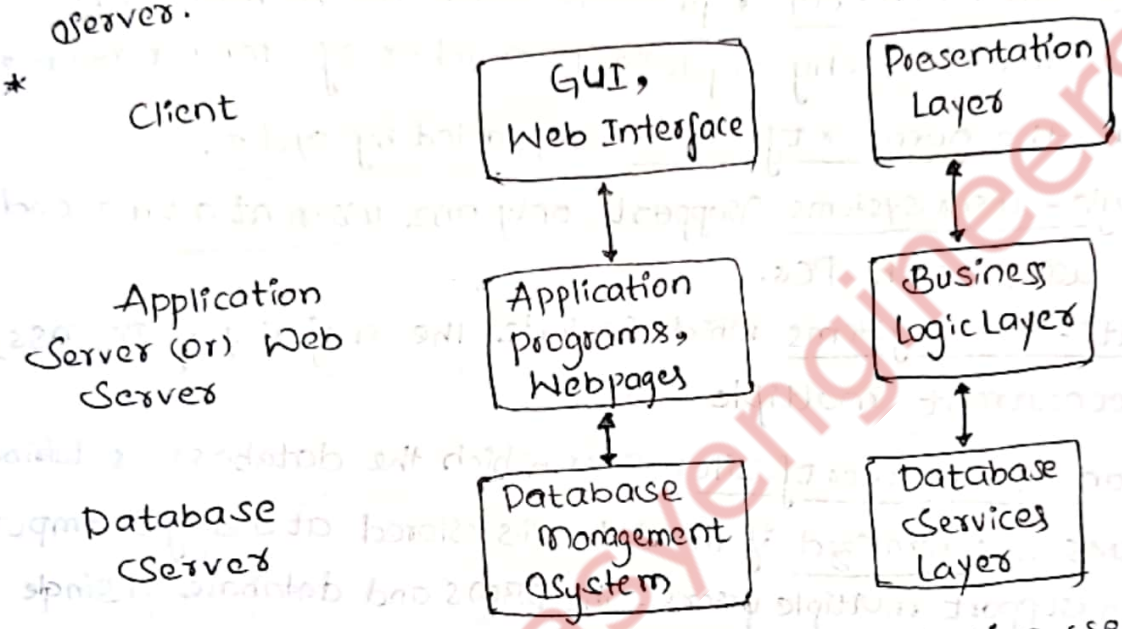
* The architectures described here are called two-tier architectures because the software components are distributed over two system : client and server.

* A standard called Open Database Connectivity (ODBC) provides an application program interface (API), which allows client-side programs to call the DBMS, as long as both client and server machines have the necessary software installed.

Three-Tier and n-Tier Architecture for Web Applications:

* It adds an intermediate layer b/w the client and the database server.

*



* This intermediate layer is called the application server, depending on application. It plays an important role by running application programs and storing business rules that are used to access data from the database server. It improved database security.

* They accept requests from client, processes the request and sends database queries and commands to the database server, then passes the processed data from the database server to the clients, and then processed further and presented to users of GUI format.

* The business logic layer handles intermediate rules and constraints before data is passed up to the user or down to the DBMS.

Classification of Database Management System.

→ Based on data model.

* Some experimental DBMSs are based on XML model, which is tree-structure data model. These are called XML DBMSs.

* The object data model defines interms of objects, their properties. The operations of each class are specified in terms of predefined procedures called methods.

* The network model represents data as record types and also represents a limited type of 1:N relationship, called a set type.

* The heirarchial model represents data as heirarchial tree structures. Each hierarchy represents number of related records

→ Based on the number of users supported by system.

* Single-user systems support only one user at a time and are mostly used with PCs.

* Multi-user systems which include the majority of DBMSs, support concurrent multiple users.

→ Based on the number of sites over which the database is distributed

* A DBMS is centralized if the data is stored at a single computer site. It can support multiple users, the DBMS and database in single computer site.

* A distributed DBMS (DDBMS) can have a actual database and DBMS software distributed over many sites.

  • Homogeneous DDBMSs use the same DBMS software at all the sites.

  • Heterogeneous DDBMSs can use different DBMS software at each site. It is possible to develop middle software to access preexisting data base stored under heterogeneous DBMSs.

* Based on cost

* Based on access path options for storing file. Well known family of DBMS is inverted file structures.

* Based on general purpose or special purpose. When performance is a primary consideration, a special purpose DBMS can be designed and built for a specific application.

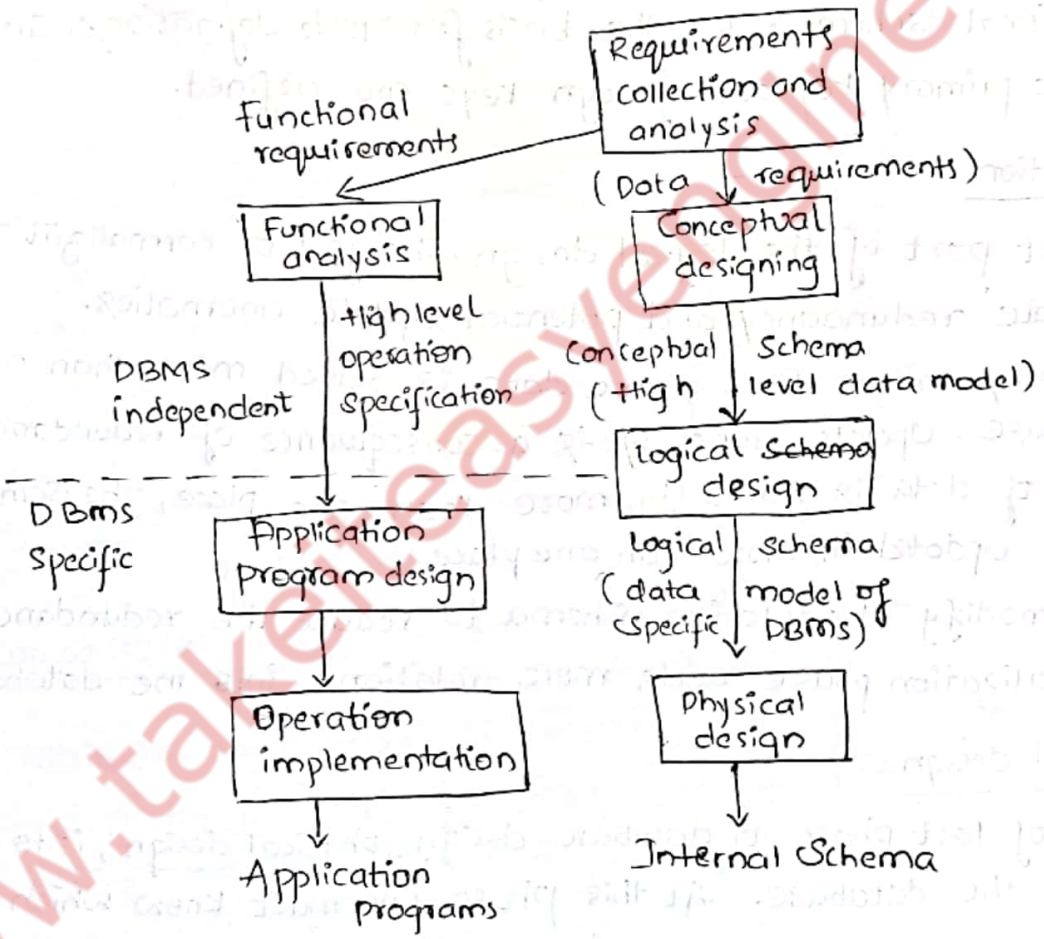
Main phases of Database Design

→ Database design is connected with application design.

→ The requirements and the collection analysis phase produce both data requirements and functional requirements.

→ The data requirements are used as a source of database design. and they should be specified in as detailed and complete form as possible.

→ The functional requirements consists of user-defined operations that will be applied to the database and they are used as a source of application software design.



Conceptual design

After all the requirements have been collected and analyzed, the next step is to create conceptual schema for database, using a high level conceptual data model. This phase is conceptual design. This result of this phase is an Entity-Relationship diagram.

It is a high-level data model of the specific application area. It describes how different entities are related two each other. It also describes what attributes each entity has.

During or after conceptual schema design, the basic data model operation can be used to specify the high-level data user operations identified during the functional analysis. It also confirms to serve the indenfied functional requirements.

Logical Design

The result of the logical design phase is a set of relation schemas. The ER diagram is the basis for these relation schemas. To create this schema is a quite mechanical operation. There are rules how the ER model is transferred to relation schemas. The relational schemas are the basis for table definations. In this phase the primary key and foriegn keys are defined.

Normalization

It is last part of the logical design. The goal of normalization is to eliminate redundancy and potential update anomalies.

Redundancy means that same data is saved more than once in a database. Update anomaly is a consequence of redundancy. If a piece of data is saved in more than one place, the same data must be updated in more than one place.

It can modify the relation schema to reduce the redundancy. Each normalization phase adds more relations into the database.

physical design:

The goal of last phase of database design, physical design, is to implement the database. At this phase one must know which database management system is used.

The SQL clauses to create the database are writlen. The idexes, the integrity constraints and the user's access rights are defined. Finally the data to test the database is added in.

In parallel, application programs are designed. The implementation of the programs can start when the database is created and data has been added in.

Entity

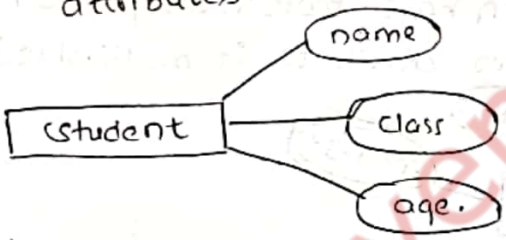
An entity is an object or component of data. An entity is represented as rectangle in an ER diagram.

For example: In the following ER diagram we have two entities Student and College and these two entities have many to one relationship as many students study in a single college.

Attributes:

An attribute describes the property of an entity. An attribute is represented as Oval in an ER diagram. All attributes have values. it describes the entity class.

For example: A student entity may have name, class, and age as attributes.



Types of attribute:

1) Simple attribute :- Simple attributes are atomic values, which cannot be divided further.

Example:- A student's phone number is an atomic value of 10 digits.

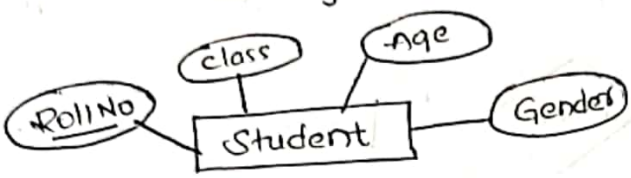
2) Composite attribute:- An attribute that is a combination of other attributes is known as composite attribute. For example; In student address is a composite ~~function~~ attribute as an address is composed of other attributes such as pincode, State, Country.

Address is a compiste attribute.

Single Valued attribute :-

Single value attributes contain single value (or) in some cases an attribute can have a set of values for the same entity.

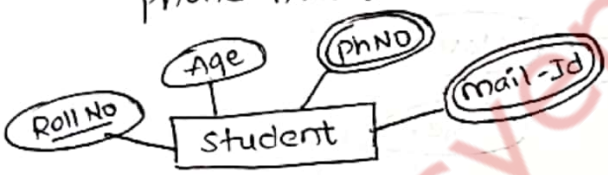
For example :- Age is a single attribute of a person, Cars with one single colour have single value.



Multivalued attribute :-

An attribute that can hold multiple values is known as multivalued attribute. It is represented with double ovals in an ER diagram.

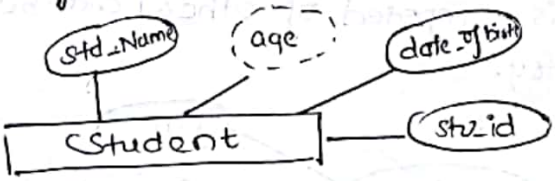
For example : A person can have more than phone number so the phone number attribute is multivalued.



Derived attribute :-

A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by dashed oval in an ER diagram.

For example :- Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).

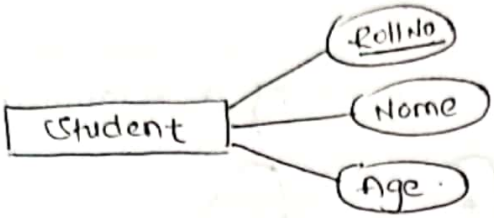


Here the date of birth is called as the Stored attribute.

Key attribute:-

A key attribute can uniquely identify an entity from an entity set.

For example:- Student roll number can uniquely identify a student from a set of students. Key attribute is represented by oval same as other attributes however text of key attribute is underlined.



Entity types and Entity Sets:

→ A database usually contains group of entities that are similar.

Forexample: the EMPLOYEE and COMPANY entities both have the same attributes. These have similar names but take different values.

→ An entity type defines a collection of entities that have the same attributes.

→ The collection of all entities of a particular entity type in the database at any point in time is called an entity set. The entity set usually have the same name as the entity type.

Example: EMPLOYEE refers to both a type of entity and the current set of all employee entities in the database.

→ An entity type describes the schema for a set of entities that share the same structure. A collection of a particular type are grouped into an entity set, called extension of entity type.

ENTITY TYPE NAME:

EMPLOYEE
Name, Age, Salary

COMPANY
Name, Headquarters, President

ENTITY SET: (EXTENSION)

e1 • (John, 55, 80K)
e2 • (Fred, 40, 30K)
e3 • (Judy, 25, 20K)
⋮

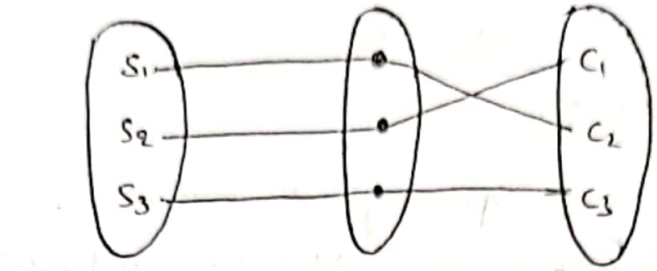
c1 • (Surro Oil, Houston, John)
c2 • (Fast Compute, Dallas, Bobbing)
⋮

Relationship and Relationship set :

→ Relationship is an association between two entities.

For example: A STUDENT entity being associated to an ACADEMIC_COURSE entity via, an ENROLLED_IN relationship.

→ A relationship is set of instances of a relationship type.



Student Entity Set | Relationship Entity set | Course entity Set

Degree of relationship

1) Unary

A unary relationship exists when both the participating entity type are the same. When such a relationship is present we say that the degree of that relationship is 1.

Ex :-

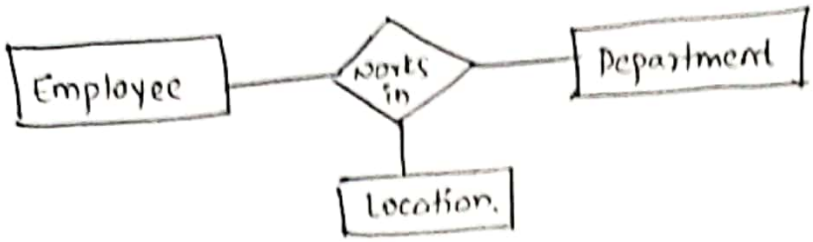
2) Binary

A binary relationship exists when exactly two entity type participates. When such a relationship is present we say that the degree is 2. This is the most common degree of relationship.

Ex :-

3) Ternary

A ternary relationship exists when exactly three entity type participates. When such a relationship is present we say that the degree is 3. As no. of entity increases it becomes complex to convert them into relational tables.



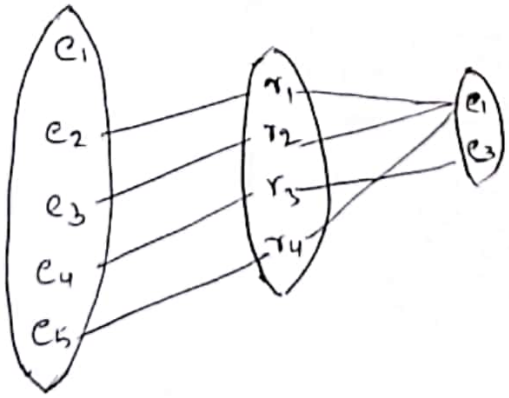
Recursive Relationship:-

A relationship between two entities of similar entity type is called a recursive relationship. Here the same entity type participates more than once in a relationship type with a different role for each instance.

Example:-

Let us suppose that we have employee table. A manager supervises a subordinate. Every employee can have a supervisor except the CEO and there can be atmost one boss for each employee. One employee may be the boss of more than one employee. Let's suppose that REPORTS_TO is a recursive relationship on the Employee entity type where each Employee plays two roles.

1) Supervisor
2) Subordinate



Supervisor and subordinate are called "Role Names". there the degree of the REPORTS_To relationship is 1.

* The minimum cardinality of supervisor entity is ZERO since the lowest level employee may not be a manager for anyone.
* The maximum cardinality of Supervisor entity is N since an employee can manage many employees.
* Similarly the subordinate entity has a minimum cardinality of ZERO to account for the case where CEO can never be a subordinate.
* It maximum cardinality is ONE since a subordinate employee can have at most one supervisor.

Here, the participants none of them have total participation since minimum cardinalities are zero. Hence, the relationships are connected by single line in ER diagram.

Constraints on Relationship Types :-

In order to make a relationship type be an accurate model, we impose certain constraints that limit the possible corresponding sets.

→ Let R be a relationship set consisting of ordered pairs of entity types A and B respectively.

Cardinality ratio :

→ 1:1 ⟶ Under this constraint, no instance of A may participate in more than one instance of R, similarly for instances of B.

In other words if (a_1, b_1) and (a_2, b_2) are (distinct) instances of R, then neither $a_1 = a_2$ nor $b_1 = b_2$.

Example: COMPANY says that every department has one employee who manages it. The manager should not be done by employee for more than one dept.

→ 1:N ⟶ Under this constraint, no instance of B may participate in more instance of R, Similarly but instances of A are under no such restriction.. If (a_1, b_1) and (a_2, b_2) are distinct instances of R, then it cannot be the case that $b_1 = b_2$.

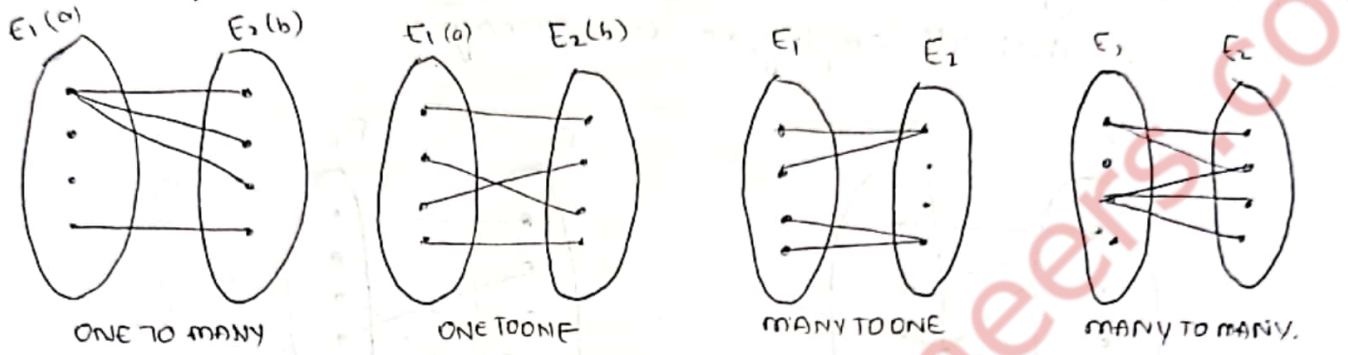
Example: CONTROLS is 1:N because no project may controlled by more than one department. Department may control any number of projects

1:N N:1 → This just the same as 1:N but with the roles of two entity types reversed.

Example: WORKS_FOR and DEPENDS_ON are N:1.

M:N → Under this constraint, there are no restrictions.

Example: WORK_ON is M:N, because an employee may work on any number of projects and a project may have any number of employees who work on it.



Participation constraint :-

It specifies whether or not the existence of an entity depends upon its being related to another entity via the relationship.

Total participation : To say that entity type A is constrained to participate totally in relationship R is to say that if R's instance set is $\{(a_1, b_1), (a_2, b_2) \dots (a_m, b_m)\}$, then A's instance set must be $\{a_1, a_2 \dots a_m\}$. In other words, there can be no member of A's instance set that doesnot participate in atleast one instance of R.

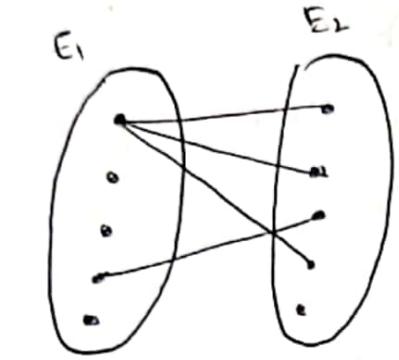
Example: According to our informal description of COMPANY, every employee must be assigned to some department. That is, every employee instance must participate in atleast once instance of WORKS_FOR, which is to say that EMPLOYEE satisfies the total participation constraint with respect to the WORKS_FOR relationship.

In an ER diagram, if entity type A must participate totally in relation type R, the two are connected by a double line.

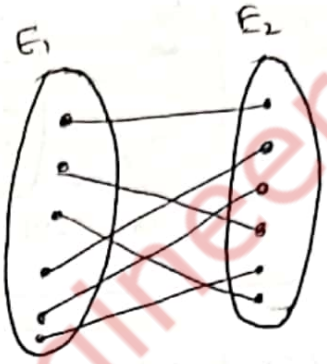
Partial participation :-

The absence of total participation constraint (or) All entities in the given set do not participate in a relation.

Example: Not every employee has to participate in MANAGES; hence we say that, with respect to MANAGES, EMPLOYEE participates partially. This is not to say that for all employees to be managers is not allowed; it only says that it need not be the case that all employees are managers.



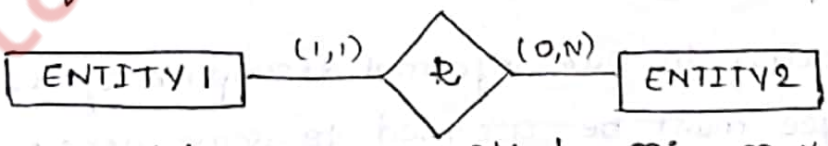
PARTIAL PARTICIPATION



TOTAL PARTICIPATION

Structural Constraints:

These are also called as structural properties of a database management system. Cardinality Ratio and participation constraints taken together are called Structuaral constraints. The nome constraints refer to the fact that such limitations must be imposed on the data, for the DBMS System to be consistent with the requirements.



The structual constraints are represents by min-max notation. This is a pair of numbers (m,n) that appear on the connecting line between the entities and their relationships. The minimum number of times an entity can appear in a relation is represented by whereas, the maximum time it is available is denoted by n. If m is 0 it signifies that the entity is participating in the relation partially, whereas, if m is either greater than or equal to 1, it denotes total participation of the entity.

Weak entity Types :-

→ Regular entity types that have a key attribute are called strong entity type. Entity types the do not have key attribute of their own are called weak entity type.

→ An entity of a weak identity type is uniquely identified by the specific entity to which it is related (by a so called identifying relationship that relates the weak entity type with its so called identifying or owner entity type) in combination with some set of its own attributes (called a partial key).

→ Example: A DEPENDENT entity is identified by it's first name together with the EMPLOYEE entity to which it related via DEPENDS_ON.

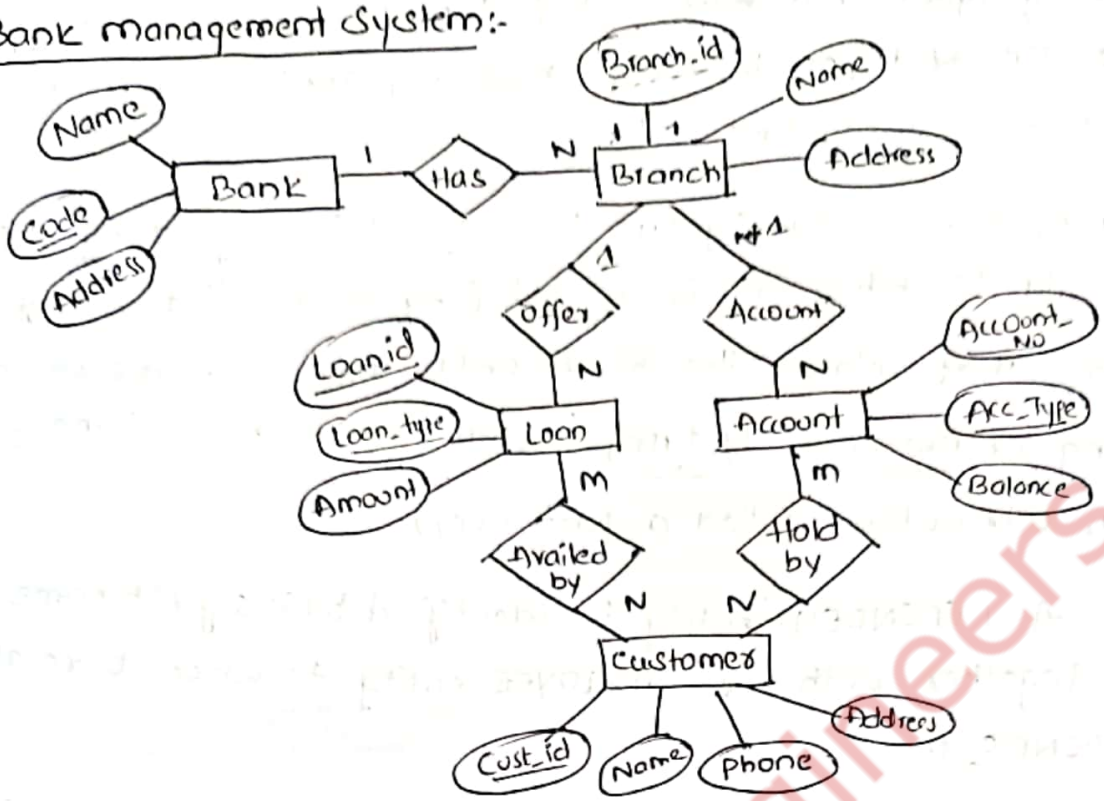
→ Because an entity of a weak entity type connot be indentified otherwise, that type has a total participation constraint with respect to the identifying relationship.

→ This should not be taken to mean that any entity type on which a total participation constraint exists is weak.
For example:- DEPARTMENT has a total participation constraint exists is ~~weak~~ with respect to MANAGES, but it is not weak.

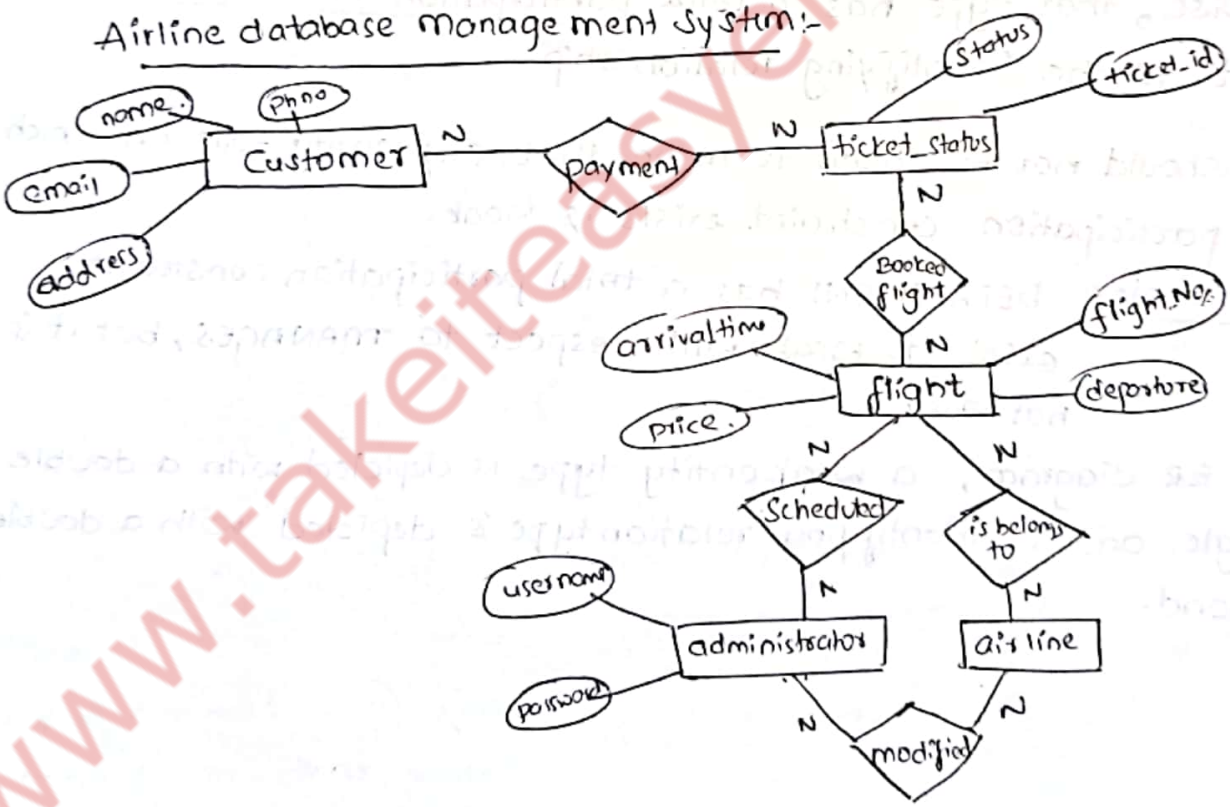
→ In an ER diagram, a weak entity type is depicted with a double rectangle and an identifying relation type is depicted with a double diamond.

ER diagrams:-

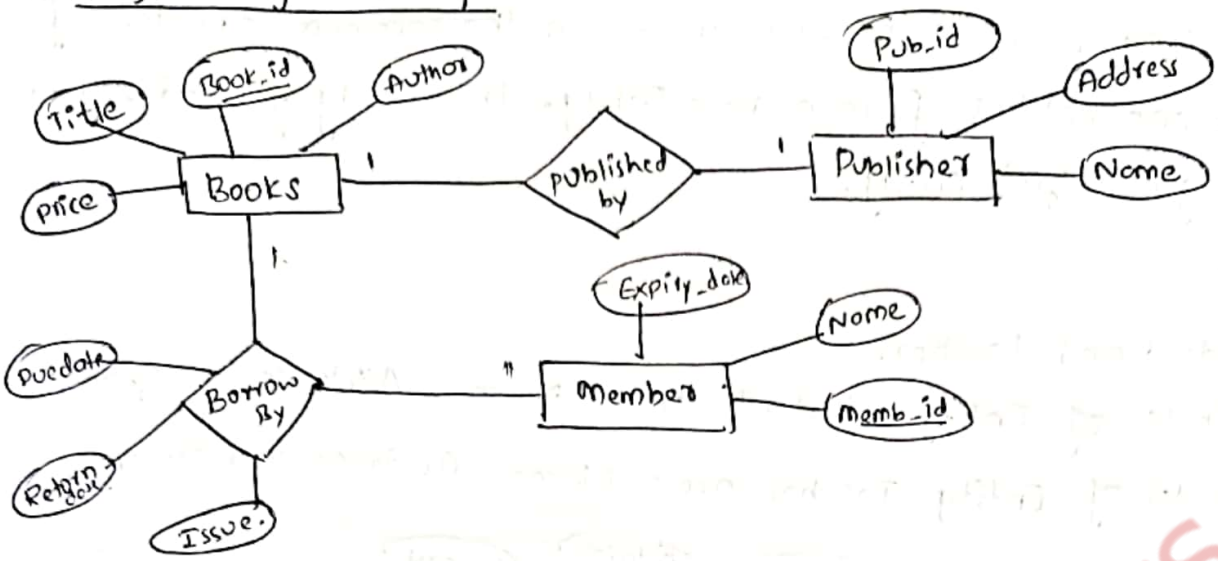
Bank management System:-



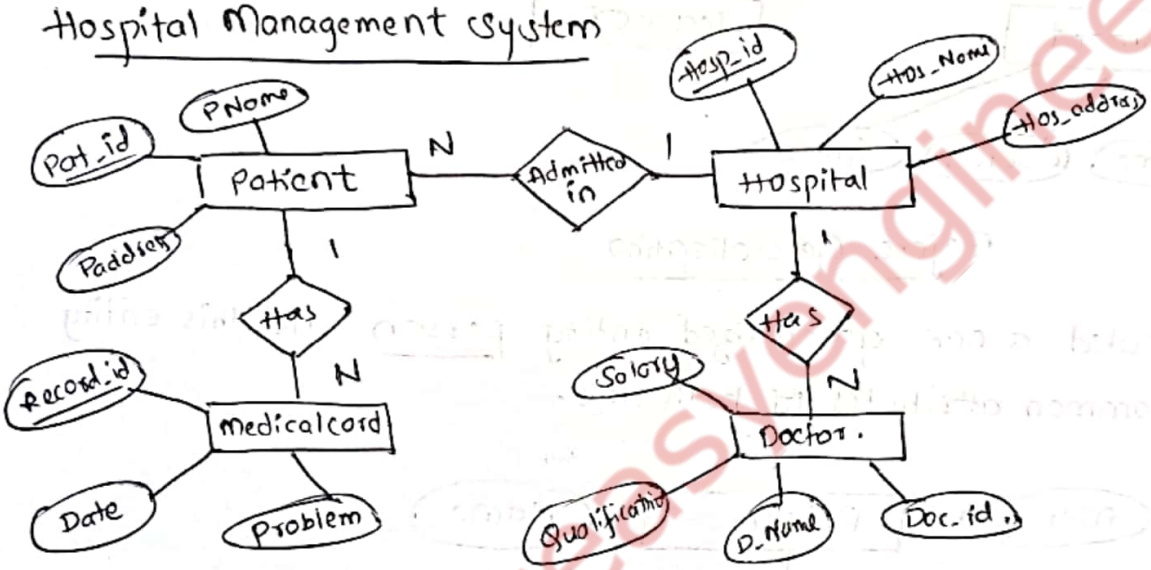
Airline database manage ment system:-



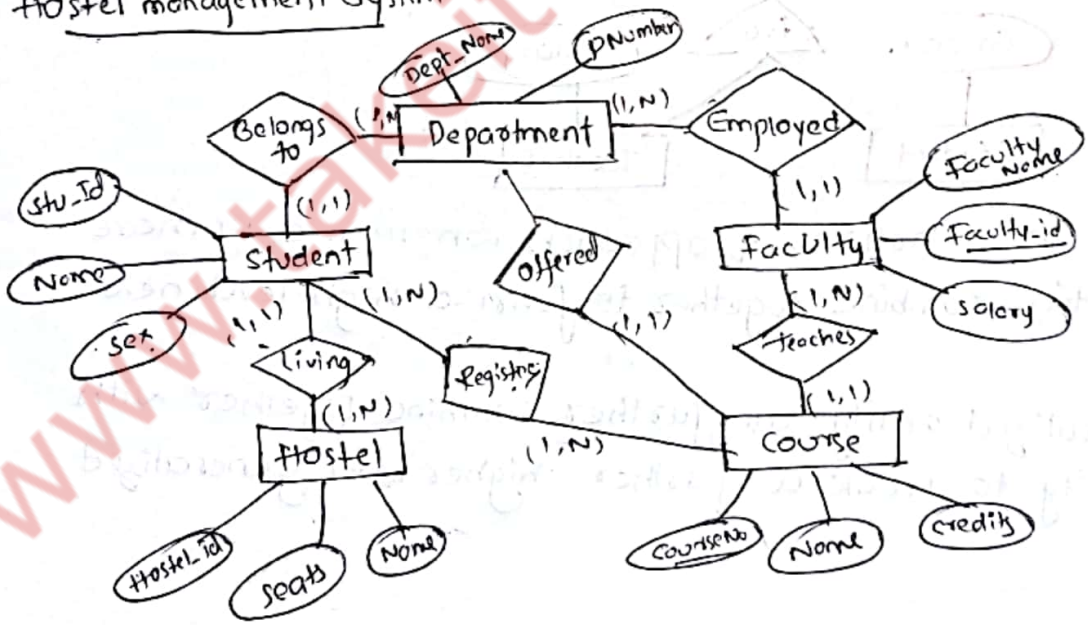
Library Management System:-



Hospital Management System



Hostel management system



Generalization:-

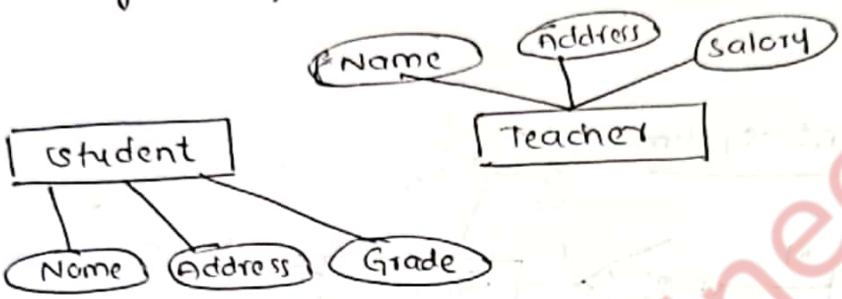
Generalization is a process in which the common attributes of more than one entities from a new entity. This newly formed entity is called generalized entity.

Example:-

Student and teacher.

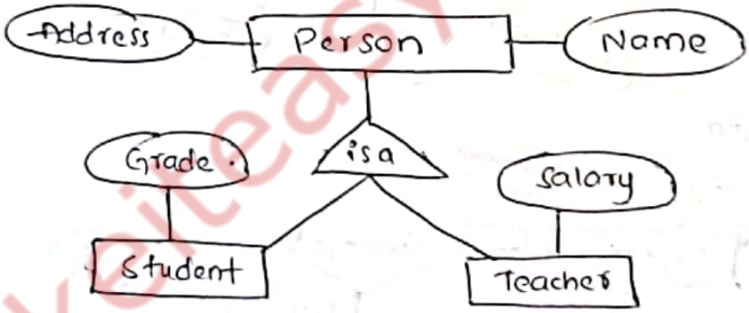
→ Attributes of Entity Student are : Name, Address and Grade

→ Attributes of Entity Teacher are : Name, Address and Salary.



Before Generalization

We have created a new generalized entity person and this entity will have common attributes of both class.



→ Generalization uses bottom-up approach where two or more lower level entities combine together to form a high level new entity.

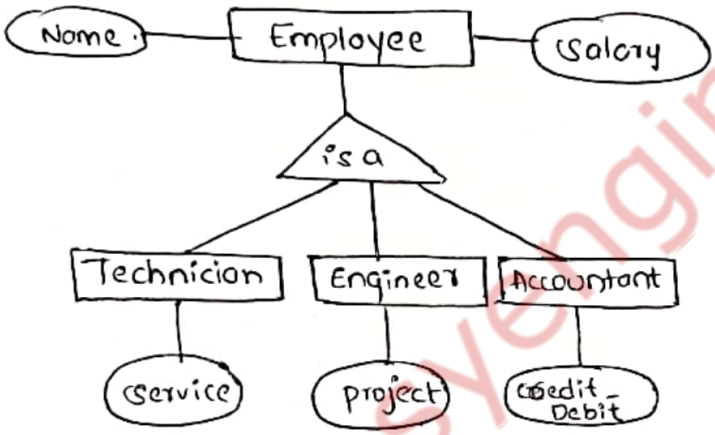
→ the new generalized entity can further combine together with lower level entity to create a further higher level generalized entity.

Specialization :

Specialization is a process in which an entity is divided into sub-entities. It can be said as reverse process of generalization, in generalization two entities combine together to form a new higher level entity. Specialization is a top-down process.

The idea behind Specialization is to find the subsets of entities that have few distinguish attributes.

For example: Consider an entity employee which can further classified as sub-entities Technician, Engineer and Accountant because these sub entities have some distinguish attributes.



In the above diagram, we can see that we have higher level entity "Employee" which we can divide in subentities. "Technician", "Engineer" and "Accountant". All of these are just an employee of a company, however their role is completely different and they have few different attributes. And also shown that Technician handles service requests, Engineer work on a project and Accountant handles the credit and debit details. All of these three employee types have few attributes common such as name and salary which we had left associated with the parent entity "Employee".