UNIT-1 INTRODUCTION

* **Database** is a collection of inter-related data, large volumes of facts and figures in an orderly manner. It has the following implicit properties.
  i) A database represents aspects of real world.
  ii) A database is designed, built and contains data usi for specific purpose and it has an intended groups of users.

* **DBMS** → A Database Management system is collection of programs that enables users to create and maintain a database. It facilitates the process of Defining, constructing, Manipulating and sharing database among various users and applications.

i) Defining :- It involves specifying the datatypes, structures, constraints of the data to be stored.
i) constructing :- It is a process of storing the data on some storage medium that is controlled by DBMS.
i) Manipulating :- It includes functions such as querying the database to retrieve specific data, updating database to reflect the changes in mini-world and generating the reports from the data.
Sharing :- It allows multiple users and programs to access the database simultaneously.

A DBMS is not only responsible for facilitating the above four but also must be able to protect and maintain the database for a long period of time.
→ protection includes system protection against the hardware and software crashes and security protection against unauthorized access or malcious access.
→ Maintaining refers to allowing the system to evolve overtime as the requirements change over time.

Characteristics of the Database approach :

A number of characteristics distinguish the database approach from the traditional approach of programming with files such as :

i) In traditional file-processing, each user defines and implements the files needed for his specific application. Each user maintains seperate files which results in redundancy, wastage of memory space and inconsistency.

The database approach on the otherhand maintains a single repository of data which can be accessed by various users. Hence it avoids redundancy & inconsisten

ii) Self-describing nature of the database system :

The traditional file approach does not contain the description of itself. However in database approach it not only stores database but also stores a complete description of database structure and constraints.

This definition is stored in DBMS catalog. which contains
→ Information such as structure of each file.
→ The type and storage format of each data item.
→ various constraints on data item.

These information stored in the catalog is referred to as "meta-data".

iii) Insulation between programs and data, Data abstracti

* In traditional file approach, data is defined as a part application program, Hence programs would be able to work only on specific database.

However in Database approach, data definition is stored in catalog seperately from application programs. This property is called "program-data-independence". Further application programs can operate on the data by invoking functions or operations regardless of how they are implemented, This is termed as "program-operation independence".

The database that facilitates program-data-independen and program-operation-independence is called as data abstraction.

iv) Support of multiple views of the data:

* A database typically has many users, each may require a different perspective or view of database.
* A traditional file processing approach supports a single view of data, However a database approach supports multiple view of the data irrespective of whether the data that they refer is stored or derived.

Consider the example of UNIVERSITY database

STUDENT	Name	Studentno	Class	Major
	Uday	10	1	CS
	Nithin	12	2	IS

COURSE	Coursename	Courseno	Hrs	Dept	Studentno
	ICS	1	5	CS	10
	DS	2	6	IS	12
	DBMS	3	7	CS	10
	OS	4	8	IS	12

PRE-REQUISITE	Courseno	Prerequisiteno
	1	2
	2	4
	3	1

Two views that can be derived are

TRANSCRIPT View	Studentname	Courseno	class	Studentno
	Uday	1	1	10
	Nithin	2	2	12
	Uday	3	1	10
	Nithin	4	2	12

PREREQUISITES View

coursename	courseno	prerequisite
DS	2	4
DBMS	3	1
ICS	1	2

v) Sharing of data and multi-user transaction

* Traditional file processing approach did not support sharing of data as well as multi-user transactions.

A multi-user DBMS, as the name implies must allow multiple users to access the database at the same time.

* DBMS includes features such as "concurrency control" to ensure that several users trying to update the same data do it in a controlled manner so that the result of the updates is correct.

For ex :- Online reservation system, clerks try to assign the seat, the DBMS should ensure that each seat is accessed by only one clerk at a time. These type of applications are generally called "On-line Transaction processing".

→ A transaction is an executing program or process that makes one or more database accesses such as reading or updating database records.

Actors on the scene :

* People whose job involves day-to-day use of large database, we call them as "Actors on the scene".

i) Database Administrators (DBA's) :

* The main responsibility of DBA is to look after the database, the DBMS and its associated softwares.
* He is responsible for authorizing access to the database, co-ordinating and monitoring it's use and also acquiring the hardware and software resources that are required.
* The DBA would also be responsible for problems such as breach of security or low response time (slow execution).

ii) Database designers :

* Database designers are responsible for identifying the data to be stored in the database and also for choosing appropriate structures to represent and store this data.
* It is his responsibility to communicate with all perspectives of database users inorder to understand their requirements and to create a design that meet these requirements.
* Database designers typically interact with each potential group of users and develop "views" of the database as per the requirement of each user group.

iii) Database endusers :

* End users are the people for which the database is primarily created for. They are the people whose job involves access to database for querying, updating, generating reports etc.

There are several categories of end-users.

→ Casual end-users : They are the users who occassionally access the database, They may need different information each time they query the database. These users include managers, occassional browsers etc.

→ Naive or parametric end-users : They make-up the major portion of the database users. They constantly query the database using standard types of queries and updates called as "canned transactions".

ex : Bank tellers check account balances, withdrawals, deposits etc.

Reservation clerks check availability for a given request and make reservations.

→ Sophisticated end-users : These include Engineers, Scientists, Business analysts and others who thoroughly familarize themselves with DBMS to implement applications (programs) that meets complex requirement.

→ Standalone users : They maintain personal databases by using readymade program packages that provide easy-to-use menu based interfaces.

* Normally Naive and casual end-users need to learn very little about facilities provided by DBMS, whereas sophisticated users learn most of the facilities of DBMS.

iv) System analyst and Application programmers :

* System analysts determine the requirements of end-users and develop specifications for canned transaction

* Application programmers implement these specifications as programs, they test, debug, document and maintain these canned transactions. They are also referred to as Software Engineers.

Workers behind the scene :

* Workers behind the scene would involve such persons who are typically not interested in database itself. They include people such as :

i) DBMS system designers : They are responsible for designing the modules. Modules such as implementing catalog, modules for controlling redundancy, handling data recovery and security.

ii) Tool developers : They develop tools (software packages). They facilitate database system design and improving performance. Tools are optional packages that are often purchased seperately.

iii) operators and maintainance personnel : They are the System administration personnel who are responsible for actual running and maintainance of hardware and software system.

Advantages of using THE DBMS approach

i) A good DBMS must support the concept of catalog or meta-data and hence must posses a self-describing nature. It must achieve insulation between program and data and between program and operation and hence must support " data-abstraction". It must support for multiple views on data and should allow sharing of data and multi-user transactions. In general DBMS must support all these characteristics.

ii) controlling redundancy : A good DBMS must control redundancy. Redundancy refers to existence of same data multiple times in the database. This leads to duplication of efforts, waste of storage space and inconsistency. Ideally a good DBMS design must store each logical data item at only one place in the database.

However, it is sometimes necessary to use controlled redundancy for improving the performance of queries. For. ex The view TRANSCRIPT contains redundant information.

TRANSCRIPT	studentname	courseno	class	studentno
	Uday	1	1	10
	Nithin	2	2	12
	Uday	3	1	10
	Nithin	4	2	12

It gives details of studentname, courseno, class along with studentno. By placing all the data together, we do not have to search multiple files to collect this data. In such cases DBMS should have the capability to control this redundancy so as to avoid inconsistencies among various files.

iii) Restricting un-authorized access :
In a multi-user environment, all users must not be allowed to access all the information that is present in the database. Typically user or user groups are given account numbers protected by passwords, which can access database for information. A DBMS must provide a security and authorized subsystem which the DBA uses to create accounts with access restrictions.

iv) Providing persistent storage for program objects :
It is another important expectation from a good database. Traditional database system normally sufferred from "impedance-mismatch" problem, since the data structures provided by DBMS and the programming languages were incompatible.
However object-oriented database systems typically offer datastructure compatibility.
DBMS offers buffering module that maintains a part of database to process the records needed by particular query. The main purpose for persistent storage is for efficient query processing.

vi) Providing backup and recovery:
A good DBMS must provide facilities for re-covering from hardware and software failures. The backup and recovery subsystem of DBMS is responsible for recovery. For ex: if computer fails in the middle of a transaction the recovery subsystem is responsible for making sure that the database is restored to the state it was before and ensure that the transaction is resumed from the point, where it was interrupted.

vii) providing multi-user interfaces:
Since many types of users will be using DBMS, each user will be varying with different levels of technical knowledge to use the database. Therefore a DBMS must provide a variety of user-interfaces for each of the users such as menu-driven interface, natural-language interface, graphical user interfaces (GUI) etc.

viii) Enforcing integrity constraints:
Most database applications have certain integrity constraints that must hold for the data such as specifying a datatype for each item, specifying uniqueness on data item values. For ex: In a STUDENT record we can specify the studentname as varchar with not more than 30 characters.

```
Studentname varchar(30)
```

Some constraints can be specified to DBMS and auto-mati-cally enforced but other constraints may have to be checked at the time of data entry.
For ex: If grade is an attribute, if the grade value is A and if it is wrongly entered, DBMS cannot automatically discover this error. Such data errors have to be discovered manually.

ix) Permitting inferencing and actions using rules:
Some database systems provide capabilities for defining deduction rules for inferencing new information from stored database

Such systems are called deductive database systems.
For ex: There may be a complex rule(s) for an application for determining when student is on probation, This can be specified as rules which when compiled and maintained by DBMS results on determining the students who are on probation. If the rule changes, then program written to access it must be modified. ∴ It is generally convienient to change declared deduction rules than recoding the programs.

* This functionality is provided by active-database Systems which automatically initiate actions when certain events and conditions occur.

A Brief history of Database Applications:

* Database applications have generally evolved overtime. Early database applications used "Hierarchical" and "Network" model.
One of the main problem with them was inter-mixing of conceptual relationships with physical storage and placement of records on the disk.
* Another disadvantage was that they provided only programming language interfaces, this made time consuming and expensive to implement new queries, since the programs had to be written, tested and debugged.

→ Due to these short-comings, the early database models gave a way for a new model called "The relational database model".

Relational Databases:

* They were designed such that it seperated physical storage of data from its conceptual representation.
* It also introduced a high-level query language as an alternative to programming languages called "Structured Query Language" (SQL). Now they exist on almost all computers, from small computers to large servers.

* Further, the emergence of object-oriented programming languages led to emergence of object-oriented database (OODB) which are mainly used in applications such as Engineering design, multimedia etc.

* With Emergence of world wide web (WWW) interchanging of data on the web became important and hence languages such as Xtended mark-up language (XML) were designed.

Database systems and Information retrieval systems

* Database systems are different from modern information retrieval systems.

→ Database system applies to structured and formatted storage of data that arises in day-to-day life such as Government, business, industry etc.
They are mainly used in manufacturing, retail, finance etc.

→ On the other hand information retrieval (IR) systems deal with books, manuscripts and various forms of library based articles. Data is indexed, annotated using Keywords. They are basically concerned with searching for material based on these Keywords.

Dis-advantages associated with DBMS
WHEN NOT TO USE DBMS

* Inspite of many advantages associated with using DBMS, there are some dis-advantages such as :-

→ High initial investment in Hardware, software and training if necessary.

→ Overhead for providing security, concurrency control and data recovery etc.

* Additional problems arise if the database designers and DBA do not properly design and administer the databases.

* DBMS is not suitable for such applications where
→ Applications are not expected to change.
→ Multiple user access are not required.
* Further general purpose DBMS's are inadequate for specific purposes such as applications related to processing maps, contours, lines required by certain industries.

Chapter-2 DATABASE SYSTEM CONCEPTS AND ARCHITECTURE

Data Model : A collection of concepts that can be used to describe the structure of a database, The Structure includes specifying data types, relationship and constraints that hold on data.

Main categories of Data models :
Data models can be categorized mainly into 3 groups.

i) High-level or conceptual data models :
These data models provides concepts that are close to many users perciere data. It uses concepts such as Entity, attributes, relationship etc.
→ An entity represents a real world object such as Employee.
→ An attribute represents some property that further describes the entity such as employee's name, salary, age.
→ The relationship among two or more entities represents an association among entities such as Works-on is a relationship between employee and project entities.

ii) Representational or Implementational data models :
These are used most frequently in commercial DBMS, it includes models such as Network models, Hierarchial models, Relational models etc.

iii) Low-level or Physical data model :
Provides concepts that describe the details of how the data is stored in computer, it specifies the record formats, record orderings, access paths etc.
→ An access path is a structure that makes search for particular database records efficient.

Database schema and Database state :

* The description of a database is called Database Schema It is specified during database design phase and is not expected to change frequently.

→ A pictorial representation of the database schema is called as schema diagram.

* The actual data present in the database at any point of time is called as Database state (snapshot). The database state (actual data) may change from time to time frequently.

→ The distinction between database schema and database state is very important. When we define a new database the database schema is only specified, At this point database state is empty state with no data.

* We get initial data (state) when insertion takes place, when an update operation is performed, we get another database state.

→ The database schema is sometimes called as "Intension"

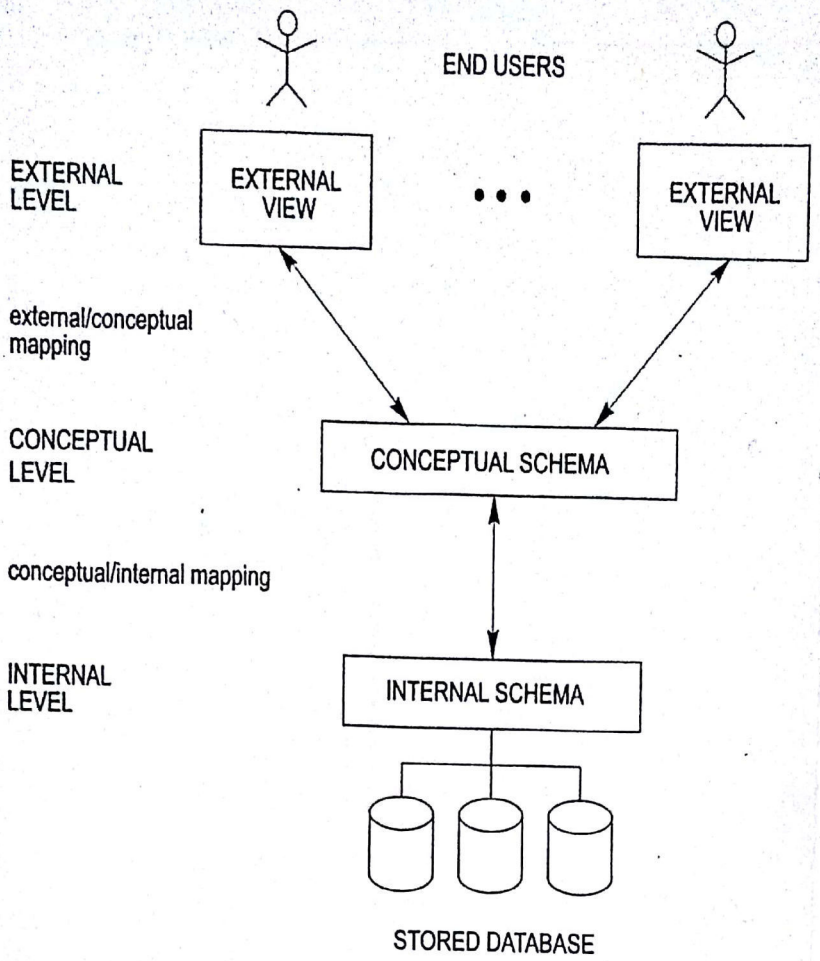
→ The database state is called "Extension" of schema.

Three-schema Architecture

* The goal of 3-schema architecture is to seperate the user applications and the physical database. In this architecture schema's can be defined at the following 3 levels.

→ Internal schema
→ Conceptual schema
→ External schema

i) Internal schema : This level describes the physical storage structure of the database, It uses a physical model that describes the complete details of data storage and access paths for database.



ii) Conceptual schema : It describes the structure of whole database for a group of users. It avoids details of storage and concentrates on describing entities, attributes, data types, relationships, user-operations and constraints. It makes uses of representational data model. to describe conceptual schema.

iii) External schema (view): Each external schema describes the part of the database that a particular group is interested in and hides the rest of the database from that user group.

→ The 3-schemas given above are only descriptions of data. The actual data exists at physical level.

→ In 3-schema architecture, each user group refers to its own external schema, Hence DBMS must transform a request specified on external schema into a request on to conceptual schema and then to internal schema.

→ This process of transforming requests and results between levels is called "Mapping".

Further, the 3-schema architecture is used to promote the concept of logical data independence and physical data independence.

Logical data independence : refers to the capacity to change the conceptual schema without having to change the external schema. changes to conceptual schema may include expansion of database (ie adding a record-type or data-item), change constraints, reduce the database (ie removing a record-type or data item).

→ only view definition and mappings need to be changed in DBMS that supports logical-data-independence.

→ changes to constraints can be applied to conceptual schema without affecting external schemas or application programs.

Physical data independence : refers to the capacity to change internal schema without having to change conceptual schema, Hence external schemas need not be changed as well.

→ changes to internal schema may be needed because some files may have to be reorganized ie improving the performance of retrieval or update.

* physical data independence exists in most of databases, ie exact location of data on disk, splitting, merging of records are hidden from the user.
* On the other hand logical data independence is hard to achieve since it requires schema and next higher level schema to be intact and only mappings between the levels can be changed.

Database Languages and Interfaces :

There are many DBMS Languages :-

i) Data Definition Language : which is used by database designers and database administrators to specify the conceptual schema. Ex: Create .

ii) Storage Definition Language (SDL):
*It is used to specify the internal schema.

iii) View Definition Language (VDL):
*It is used to specify the user views and their moppings to the conceptual schema.

iv) Data Manipulation Language (DML);
* It is used to perform manipulation on the database once the database is populated with data, such as retrieval, insertion, deletion, modification etc.

There are 2 main types of DML's
a) High-level or non-procedural DML (Record at a time)
b) Low-level or procedural DML. (Set-at-a time).

→ High-level DML or non-procedural DML can be used on its own to specify complex database applications or operations.
* These statements can also be embedded in a general purpose programming language called "Host-language".

→ Low-level or procedural DML cannot be used on its own to specify complex database operations, they must be embedded in a general purpose programming language called Host language, In such cases DML is called as "Data Sub language".

DBMS interfaces:
User-friendly interfaces provided by DBMS includes

1) Menu based interfaces:
* It presents the users with a list of options (called menus) that help the user to make a request. The advantage of this is that user need not memorize specific commands and syntaxes.

ii) Form based interfaces:
* It presents a form to each user. Users can fill out all of the form entries (or a few entries) to insert new data

onto the database. Forms are usually designed and Programmed for naive users.

iii) Natural language interface:

* It accepts requests in English or some other language and attempts to understand them. A natural language interface would have a dictionary of important words. If the interpretation is successful it generates a high level query, otherwise a dialogue is started with the user to clarify the request.

iv) Graphical user interface:

* It presents a pictorial form of the schema, the user can then use a pointing device (such as mouse) to make a choice out of many options provided by GUI.

v) Interfaces for parametric users:

* They are specifically designed keeping in mind the parametric users (such as bank tellers) who often have a small set of operations to be performed. Functional keys on the keyboard can be programmed to initiate various commands. This allows users to operate within a minimum number of key strokes.

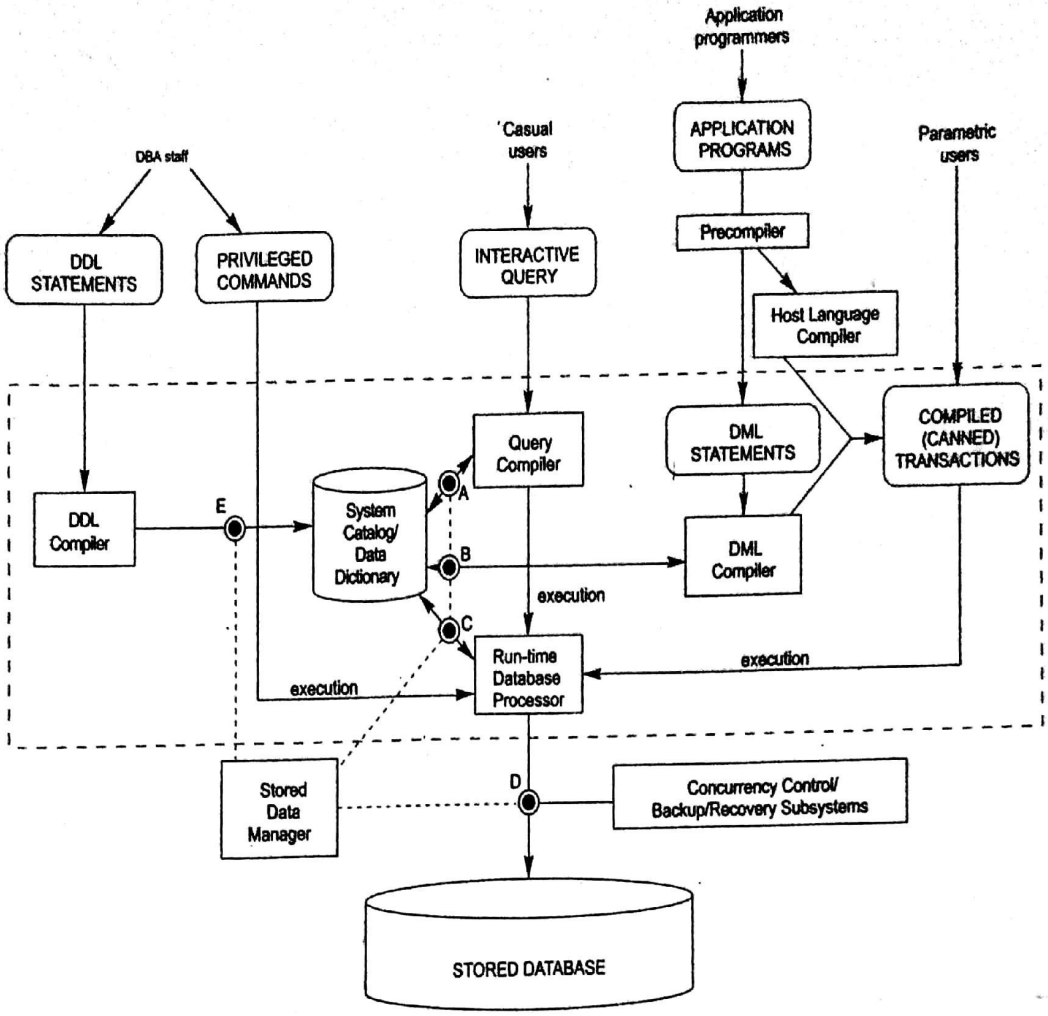
vi) Interfaces for DBA:

* They contain priviliged commands that can be used only by DBA staffs. They include commands for creating accounts, granting account authorization etc.

The Database System Environment

* A DBMS is a complex software system, The component modules of DBMS and their interactions are shown in the following diagram.

* It contains 2 halves, the top half refers to the various users of the database environment and their interfaces. The lower half shows the internals of DBMS responsible for storage of data.



* The top half contains interfaces for DBA staff, casual end users, application programmers and parametric users.
* The DDL compiler processes schema definitions (specified in DDL) and stores the description of the schemas in the catalog. The catalog includes information such as name of
* The Query files, data types of data items, storage details of each file etc.
* The query compiler accepts interactive queries that are entered, converts them into an intermediate form and then passes it onto query optimizer.
* The Query optimizer would further optimize the queries by elimination of redundancies, reordering of operations etc
* The precompiler extracts DML commands from an application program (written in host language) and sends it to DML compiler.

The rest of the program is sent to the host language compiler.

* The lower half of the diagram contains Runtime database processor, and a stored Data manager.

→ The Runtime database processor executes privilged commands queries from query optimizer and compiled transactions.

→ The Data manager helps the runtime database processor by using basic operating system services for carrying out low level input/output operations.

Database system utilities : (NOT REQUIRED)

* Database utilities refer to additional facilities that help the DBA to manage the database system, some of the common utilities are :

* Loading : A loading utility is used to load existing data files such as text or sequential files into the database. Usually current (source) format of the data file and desired (target) database structure are specified to the utility, which then automatically reformats data and stores it in database.

* Backup : A backup utility creates a backup copy of the database by usually storing the entire database onto the tape. This backup copy can be used to restore the database in case of failure.

* File-reorganization (Database storage reorganization) : This utility can be used to re-organize a set of database files into a different file organization to improve the performance.

* Performance monitoring : This utility provides the performance statistics of the database to DBA, by using this information DBA can make scientific decisions about improving performance.

UNIT-2 DATA MODELING USING THE ENTITY-RELATIONSHIP MODEL

Using high-level conceptual Data models for Database design

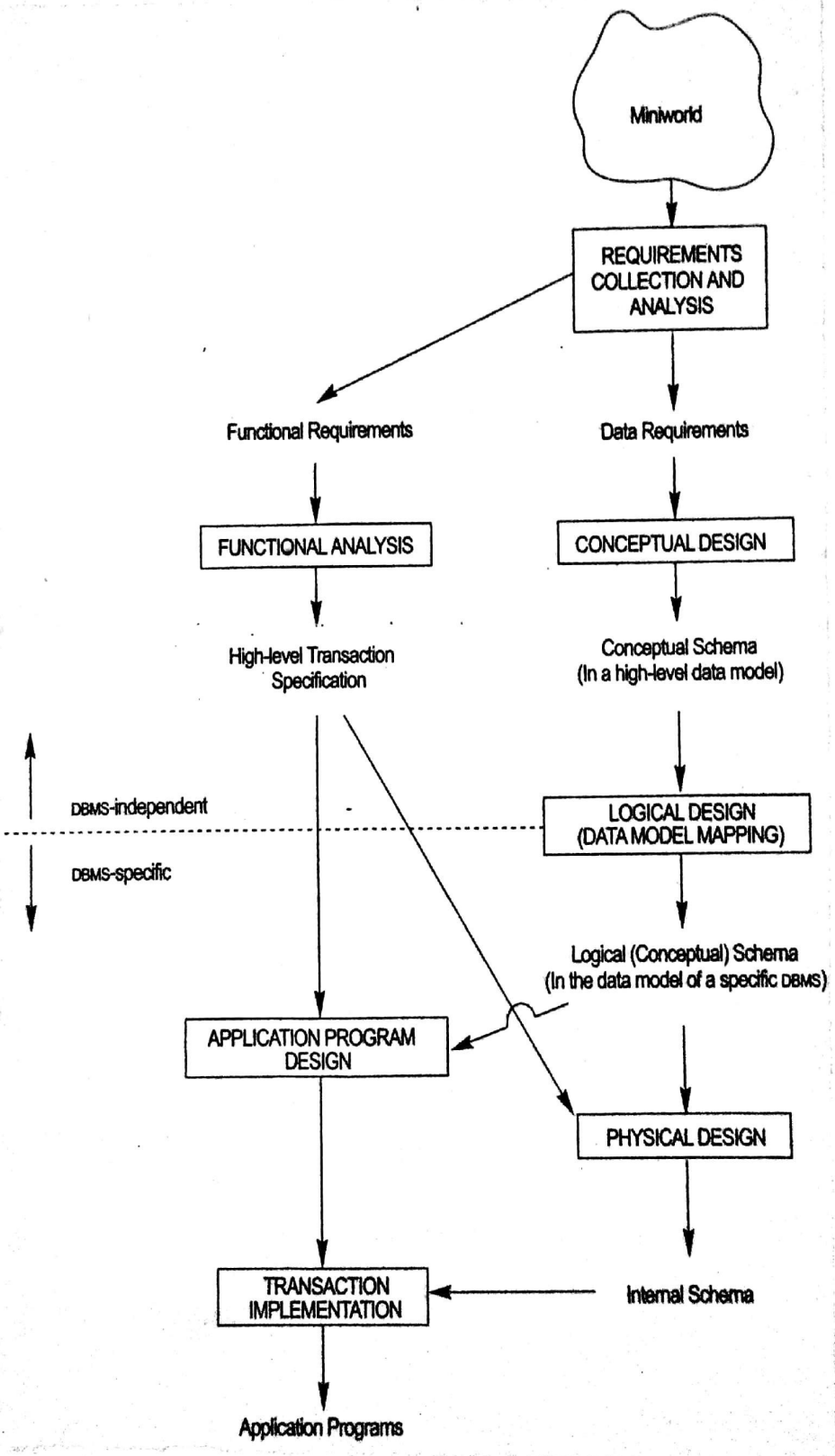


Fig shows the different phases of database design

* The first step is called "Requirement collection and analysis" phase. During this phase the Database designers interview the database users to understand and collect their requirements.
The result of this phase is a written set of users requirements. These requirements should be detailed.
* It is useful to specify the Known functional requirements of the application, which consists of user-defined operations that will be applied to the database.
Dataflow diagrams, sequence diagrams, scenarios etc can be used for specifying functional requirements.

* Once all the requirements are collected and analyzed, the second step involves creating conceptual schema using the data requirements, this is called as conceptual Design.
* The design includes creation of Entity types, relationships and constraints. These concepts do not include implementation details which makes even a non-technical user easy to understand.
* Modifications can be done to the conceptual schema if some function requirements cannot be specified in the initial schema.
* The third step is the actual implementation of database using a commercial DBMS such as ORACLE and implementation models usage such as relational or object data model.
* The conceptual schema is transformed from high-level data model to implementation data model. This step is called Logical Design or Data model mapping, which results in database schema.
* The last step is the physical design, during which the structures, Indexes, access path and file organizations, for the database files are specified.

* In parallel to these application programs are designed and implemented as transactions to meet functional requirements.

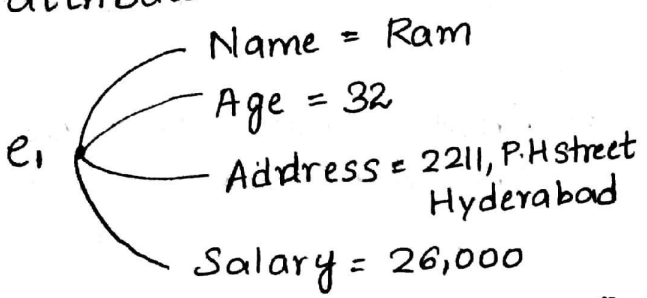
Entity Types, Entity Sets, Attributes And Keys:

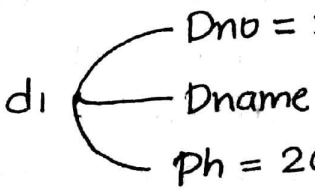
Entities are the objects of ER model. Entity represent a "thing" in the real world that has an Independent existence.

*Each entity has attributes.

Attributes are the properties that describe fully a particular entity.

Ex:- The EMPLOYEE is an entity which can be described by attributes such as name, age, address, salary etc.





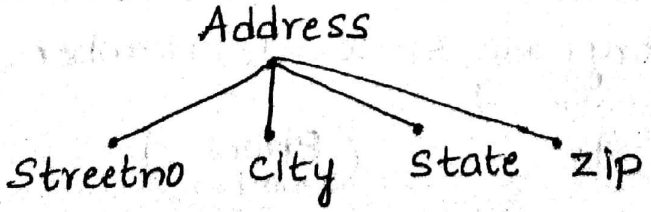
Two entities, Employee e and Department d and their attributes.

Attributes can be classified into the following types:-

i) Composite vs atomic attributes.
ii) Single valued vs multivalued attributes.
iii) stored vs derived attributes.

* Composite attributes are such attributes which can be further divided into smaller sub-parts. These sub-parts would represent more basic attributes.

For ex: address attribute can be further divided into streetno, city, state, zip.



* Atomic attributes are such attributes which cannot be further divided into smaller sub-parts.
Ex: sex cannot be further divided, ∴ it is atomic.

* Single-valued attributes are such attributes which takes a single value for a particular entity.
For ex: Date_of-birth of a person is a single-valued attribute.

* Multi-valued attributes are such attributes that can have set of values for the same entity.
For ex: Ph_no of a person is a multi-valued attribute.

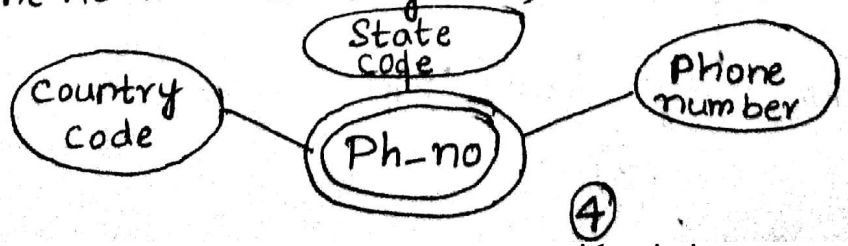
* Derived attributes are such attributes that can take a value which is obtained by using the value of another attribute. Ex: Age of a person can be obtained from Date_of-birth and present date.

* Stored attributes are such attributes that cannot be obtained using the value of another attribute.
For ex: date-of-birth is stored attribute.

NULL values:

* In some cases, a particular entity may not have an applicable value for an attribute. For such situations a special value called "NULL" value must be used.
For ex: student may have Landline no or may not have.
There are 2 situations in which NULL are used.
→ The attribute value exists but it is missing.
→ The attribute is not known or not applicable.

Complex attributes: An attribute which is both ~~Complex~~ COMPOSITE and multi-valued is called complex attribute.
For ex: Person can have more than one phone no and each phone no has country code, state code, number.



Entity type: defines a collection of Entities that have the same attributes. Each entity type in the database is described by its name and attributes.
An entity type is also called as "schema" or "Intension".

Entity set: A collection of all entities of a particular entity type in the database at any point of time is called entity set.
An entity set is also called as "Extension".

ENTITY TYPE:
ATTRIBUTES

EMPLOYEE

Name	Age	Salary
	• e1	
RAM	26	25000
	• e2	
AJAY	23	15000
	• e3	
RAVI	26	24000
	⋮	

ENTITY SET;
(Extension)

DEPARTMENT

Dno	Dname
	• d1
1	CS
	• d2
2	EC
	• d3
3	IS
	⋮

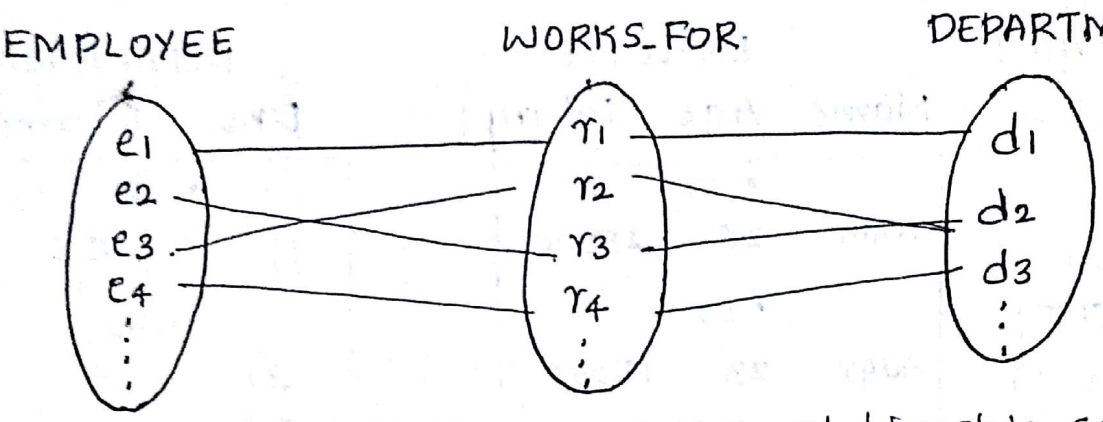
Key attributes of an entity type:
* An entity type usually has an attribute whose values are distinct for each individual entity in entity set. Such an attribute is called "Key attribute".
Its value can be used to identify each entity set uniquely.
For ex: PERSON entity type has Key attribute called ssn.

Value set or Domain of attributes:
* Each attribute of an entity would be associated with the value set (or Domain). This specifies the set of values that can be assigned to that attribute.
For ex, we can specify the value set of age attribute of EMPLOYEE to be an integer between 18 and 70. (111)
Sex as Male and Female

Relationship types, Relationship sets, roles and structural constraints

* A Relationship type R among n entity types $E_1, E_2, E_3 \ldots E_n$, defines a set of associations among entities.
* A Relationship set R is a set of relationship instances r_i, where each r_i associates n individual entities.

Ex:

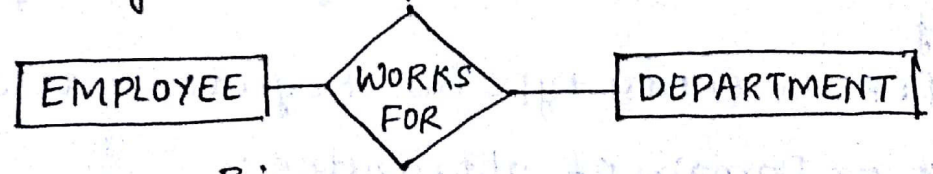


Some instances in the WORKS_FOR relationship set, which represents a relationship between EMPLOYEE & DEPARTMENT.

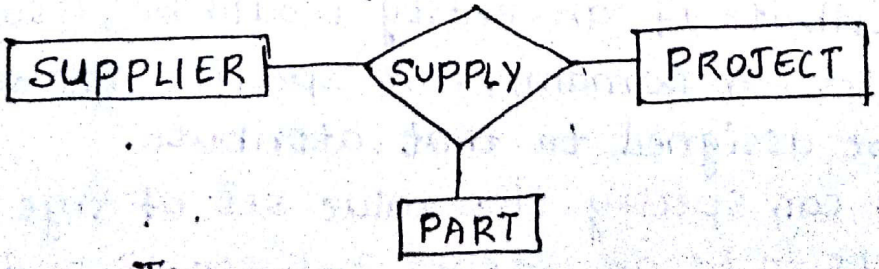
Degree of a relationship:

* The degree of a relationship type is the number of participating entity types. If 2 entities are involved in a relationship, then it is called "Binary relationship".
* If 3 entities are involved within a relationship, it is called "Ternary relationship".

Ex:



Binary.



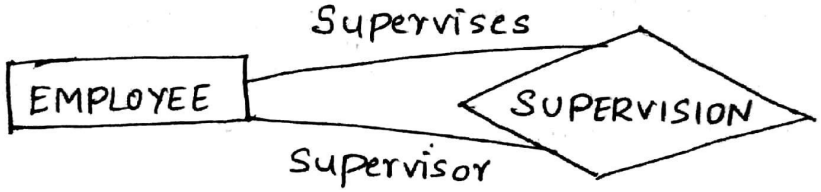
Ternary.

Role names and Recursive relationships:

* The role name signifies the role that the participating entities play in each relationship. For Ex consider the EMPLOYEE and DEPARTMENT entities as

* In the above example, the role name is works_for and it signifies that the employee works for a particular department. However role names are not required when the participating entities are distinct. However if the participating entities are same, role name becomes very much essential for distinguishing the meaning of each participation. Such relationships are called "Recursive" relationships.

For ex:



Here supervisor and supervises are role names.

*Employee participates twice in supervision, once in the role of supervisor and next in the role of a supervisee, such relationships are recursive where role names are very essential.

Constraints on relationship types:

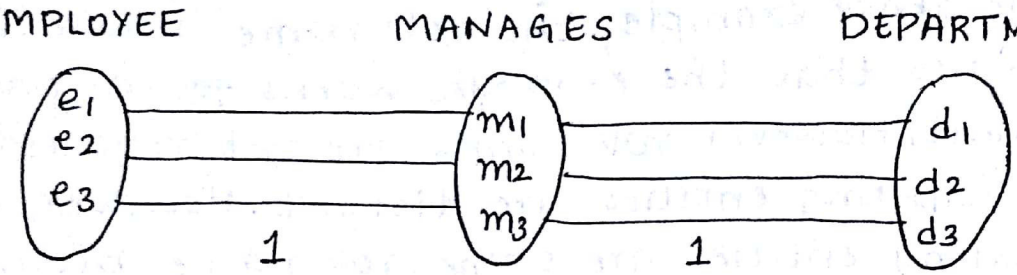
* There are 2 main relationship constraints
  - i) Cardinality ratio
  - ii) Participation

  } Also called "Structural constraints"

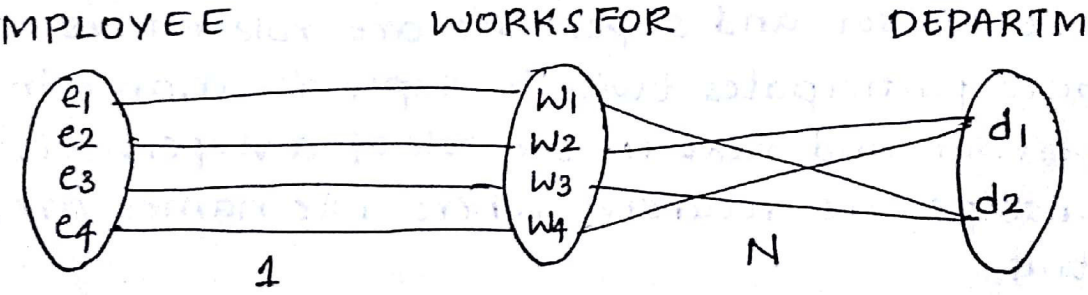
Cardinality ratio:

* The cardinality ratio of a binary relationship specifies the maximum number of relationship instances that an entity can participate in. The possible cardinality ratios for binary relationship are 1:1, 1:N, N:1 and M:N.

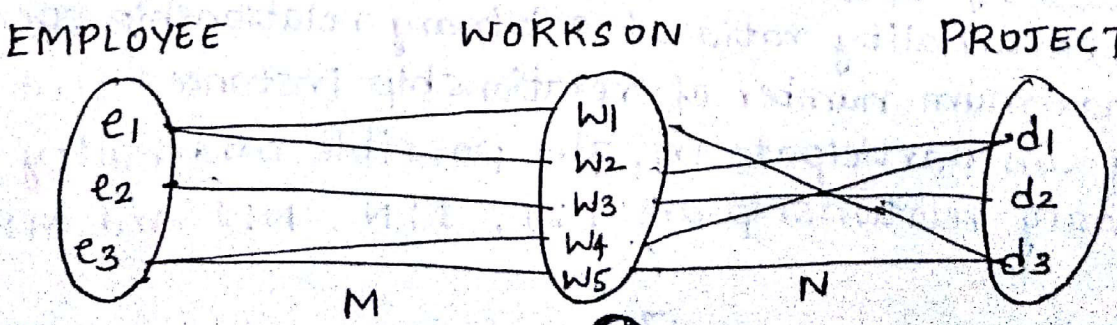
* An example for 1:1 binary relationship is MANAGES, which relates a department entity to the employee who manages the department. This represents the constraint that at any point of time, an employee can manage one department only and a department can have one manager only.



* An example for N:1 is WORKS for, which relates a department entity to employee entity. This represents a constraint that at any point in time, a department may have many employees but an employee works for only one department.



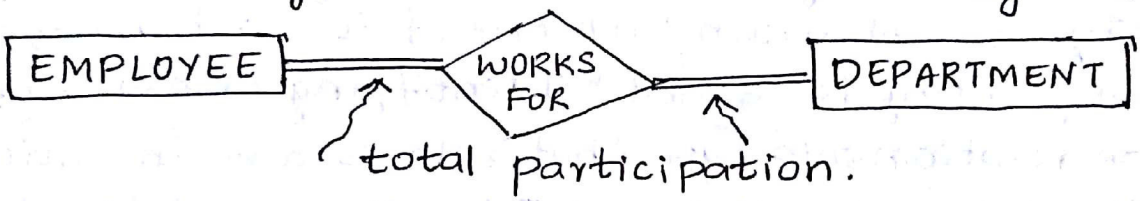
* An example for M:N binary relationship is WORKSON which relates the employee entity to project entity. This represents the constraint that at any point of time, an employee may work on more than one project and that project also can have more than one employee



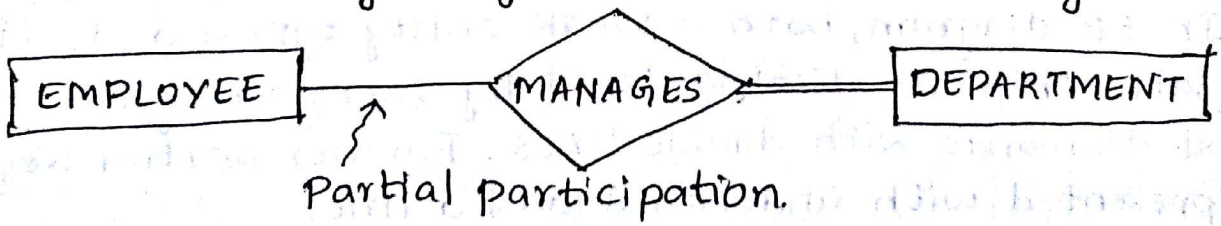
Participation constraints:

* Participation constraint specifies whether the existence of an entity depends on its being related to another entity via a relationship type. There are 2 types of participation constraints
  i) Total participation (Existence Dependency)
  ii) Partial participation.

* For ex, if the company policy states that every employee must work for a department, then employee entity can exist if and only if it participates in the works for relationship. Thus, the participation of an employee in worksfor is called "Total participation". (also called as Existence dependency). Total participation is represented by double lines in an ER-diagram.

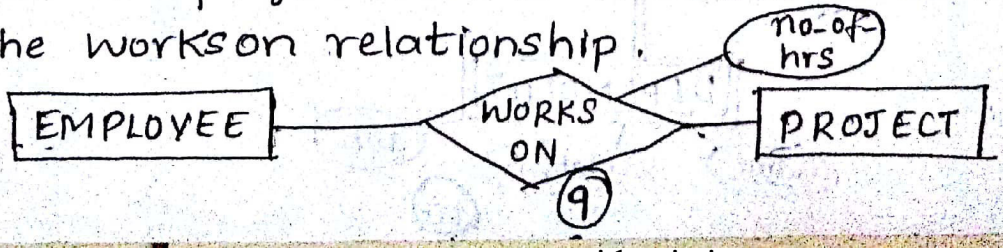


* For ex, we do not expect every employee to manage a department and hence the participation of Employee in manages relationship is partial. Partial participation is represented by single line in an E-R diagram.

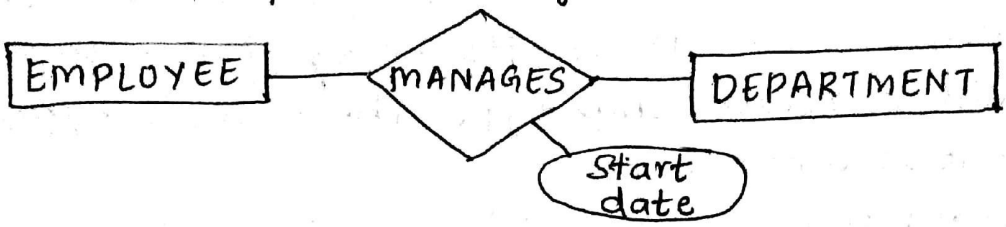


Attributes on relationship types:

* Similar to how an entity can have attributes, even relationship types can have attributes.
* For ex: The number of hours per week that an employee works on a project can be included as an attribute of the workson relationship.



* For ex: the manager start date also can be made as an attribute of the manages relationship as shown



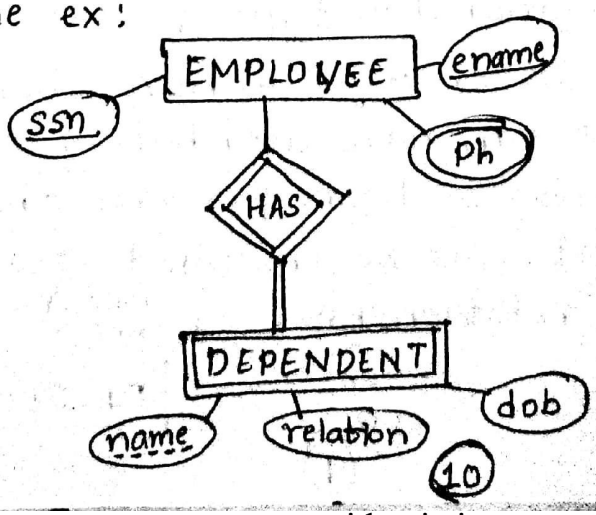
Weak entity types:

* Entities that do not have key attributes of their own are called as "weak entities". on the other hand the entities are such entities which have a key attribute of their own are called as "strong entities".

Ex: Dependent may be weak entity.
Employee may be strong entity.

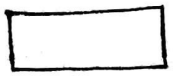
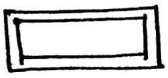
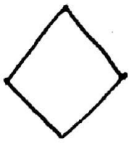
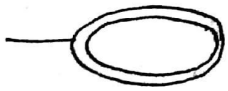
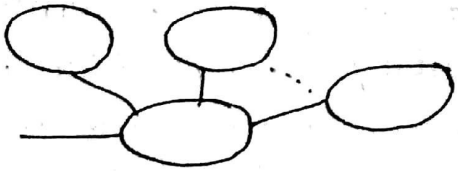
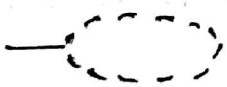
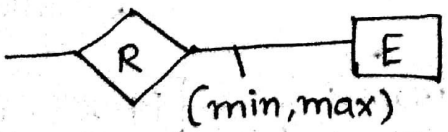
* A weak-entity is identified through another strong entity in combination with one of its attribute, such a strong entity is called "Identifying or owner entity".
* The relationship type that relates a weak entity type to its owner is called as "Identifying relationship".
* A weak entity normally has a "Partial key" which is an attribute (or set of attributes) that can uniquely identify weak entities that are related to some owner entity.
* In ER diagram, both a weak-entity type and identifying relationship are distinguished by surrounding their boxes and diamonds with double lines. Further partial key is represented with underlined dashed line.

Consider the ex:



ER diagrams, Naming conventions and Design issues

* The graphical or pictorial representation of relational schema is called as "ER diagram" (Entity-relationship).
* Some symbols used in ER diagram.

SYMBOL	MEANING
	Entity.
	Weak entity.
	Relationship.
	Weak relationship
	Attribute
	Key attribute
	Multi-valued attribute
	Composite attribute
	Derived attribute
	Total participation of E_2 in R
	Cardinality ratio 1:N for E_1 : E_2 in R
	Structural constraint (min,max) on participation of E in R

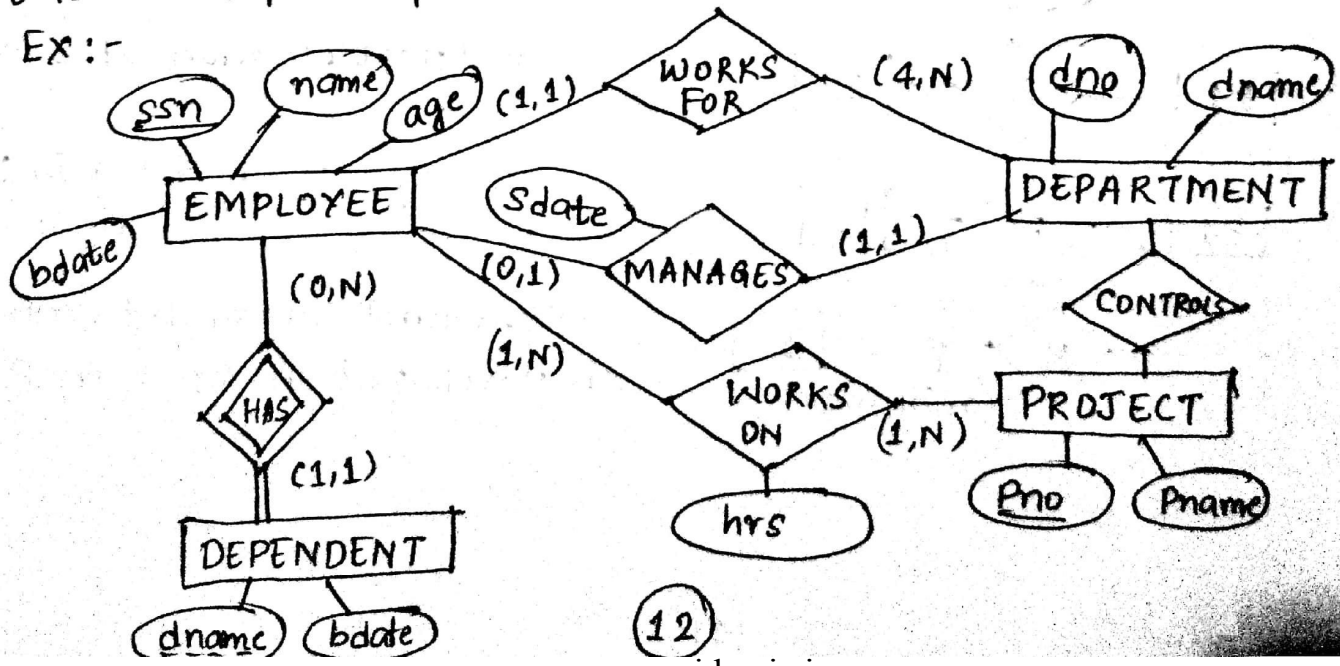
Naming conventions:

* One should choose names that convey as much as possible the meanings attached to them in ER schema.
* Normally singular names are chosen for entities rather than plural ones.
* Normally entities and relationship names are in uppercase letters where as attribute names are initial letter capitalized. Role names would be in lower case.
* As a general practise, given a narrative description of the database requirements, the nouns appearing in the narrative tend to give rise to entities, where as verbs tend to indicate relationships.
* Another naming consideration Involves choosing binary relationship names to make the ER diagram of the schema readable from left to right and from top to bottom.

Alternate notations for ER diagrams:

* other than the Standard form of representation of ER diagram, an alternative notation includes specifying structural constraints on relationship. This notation involves association of a pair of integer values (min, max) with each participation of an entity E in relationship R, where $0 \leq min \leq max$ and $max \geq 1$. In this method min = 0 implies partial participation and min > 0 implies it is total participation.

Ex:-

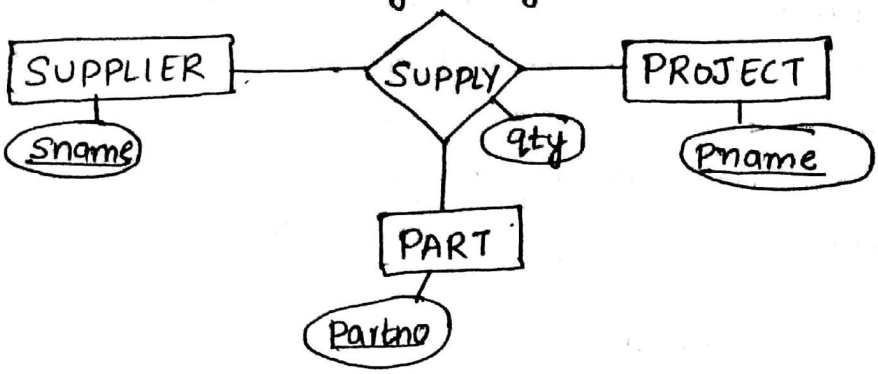


Relationship types of degree higher than two

Choosing between binary and ternary relationships :

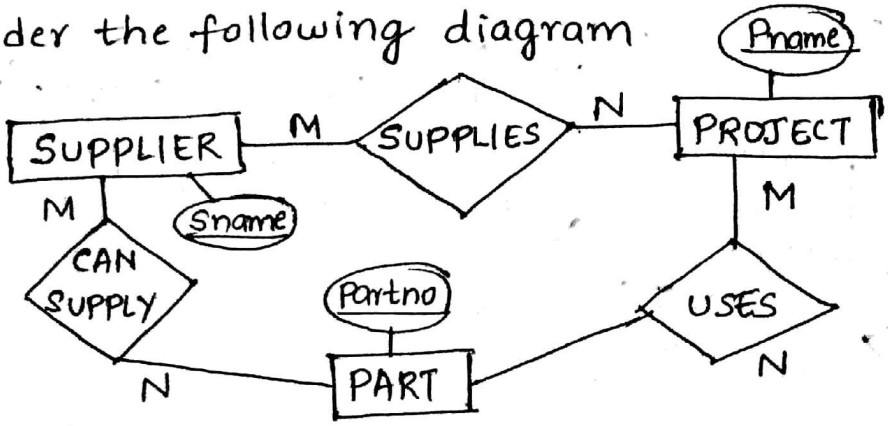
* The degree of a relationship refers to the number of entities participating in the relationship. If 2 entities participate in a relationship then it is referred to as binary. If 3 entities participate in a relationship then it is referred to as ternary relationship.

Consider the following diagram



* In the above example, SUPPLY represents a ternary relationship since it is associated with 3 entities namely SUPPLIER, PROJECT, and PART.

consider the following diagram

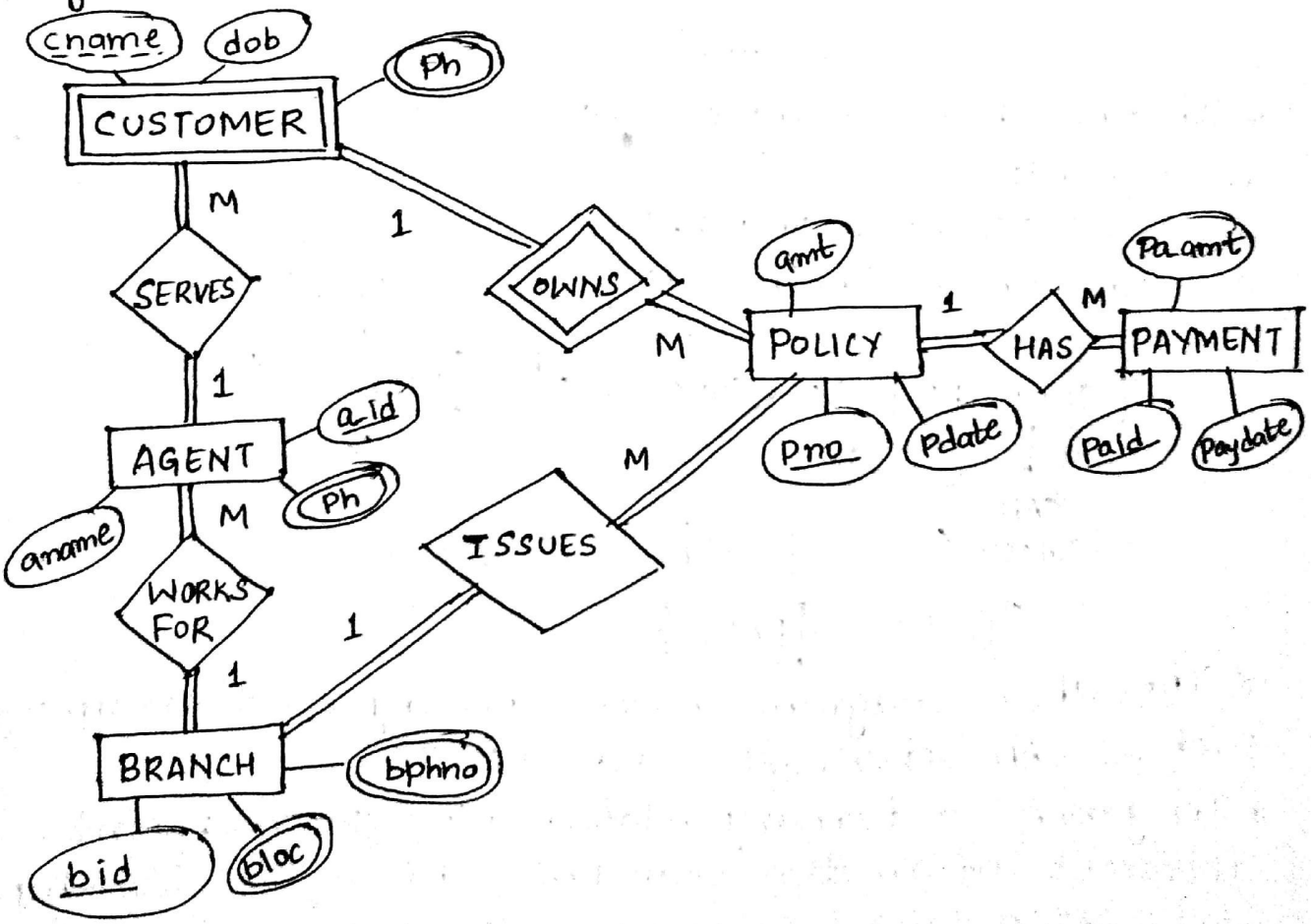


* The above diagram shows 3 binary relationship types such as CANSUPPLY, USES and SUPPLIES.

* In general a ternary relationship type represent different information than the 3 binary relationships. It is often tricky to decide if a relationship should be represented as a relationship of degree 'n' or should be broken down into several relationship types of smaller degree.

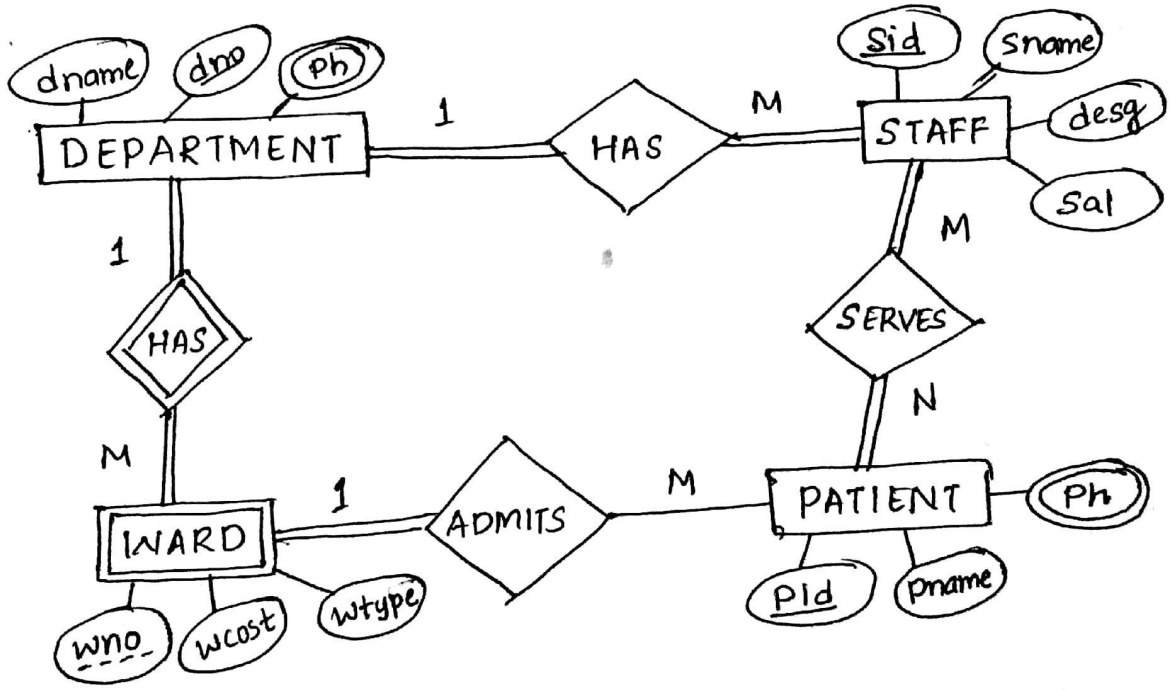
Insurance Company Database

* A branch may employee many agents however, an agent must work for only that branch for which he has been employed.
* An agent can serve any number of customers, however a customer while purchasing a particular policy must purchase it from a single agent.
* One branch may issue many policies however a policy can be issued by a single branch.
* A customer can own many policies however a policy can be owned by a single customer.
* A policy may have many payment details, However a payment detail is always with respect to single policy.



Hospital Database:

* A department may employ any number of staffs, however a staff belongs to one and only one department.
* A department has many wards. However a ward belongs to one and only one department.
* A staff would serve many patients, a patient may be served by many staffs.
* A ward would admit many patients, however a patient can get admitted to single ward at a time.
* Wardno would be unique only with department.
* All patients may not get admitted to a ward.



Refer to all other database ER diagrams from CLASS NOTES.