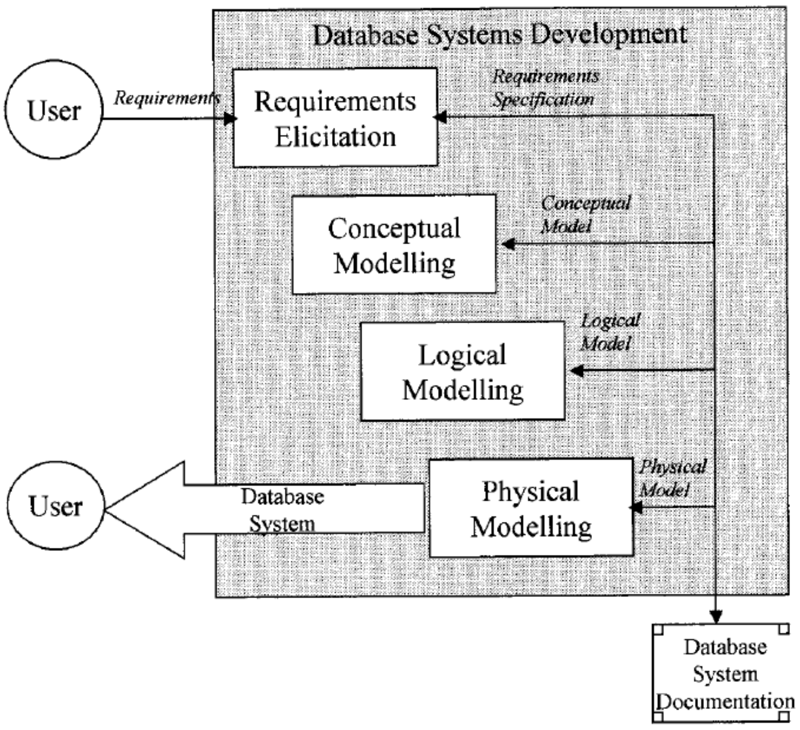
CHAPTER - 2
Data Models

2.1 Introduction to Data Models

- Data models describe the underlying structure of database. It is a conceptual tool for describing data, relationship among data, data semantics and consistency constraints.
- A data model is a picture or description which shows how the data is to be arranged to achieve a given task. DBMS organize and structure data so that it can be retrieved and manipulated by different users and application programs.
- Data model form the blueprint of a database design. Data model is based on some mathematical background and maintains some rules and restrictions.

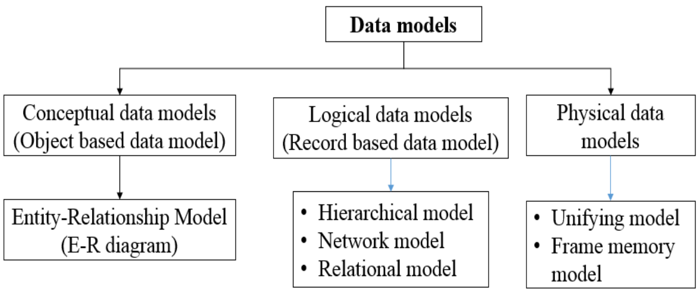


Design process of DBMS

1. **Determine the purpose of the database**
2. **Find and organize the information required** - Gather all of the types of information to record in the database, such as product name and order number.
3. **Divide the information into tables** - Divide information items into major entities or subjects, such as Products or Orders. Each subject then becomes a table.
4. **Turn information items into columns** - Decide what information needs to be stored in each table. Each item becomes a field, and is displayed as a column in the table. For example, an Employees table might include fields such as Last Name and Hire Date.
5. **Specify primary keys** - Choose each table's primary key. The primary key is a column, or a set of columns, that is used to uniquely identify each row. An example might be Product ID or Order ID.
6. **Set up the table relationships** - Look at each table and decide how the data in one table is related to the data in other tables. Add fields to tables or create new tables to clarify the relationships, as necessary.
7. **Refine the design** - Analyze the design for errors. Create tables and add a few records of sample data. Check if results come from the tables as expected. Make adjustments to the design, as needed.
8. **Apply the normalization rules** - Apply the data normalization rules to see if tables are structured correctly. Make adjustments to the tables

Database development is generally a process of modelling. It is a process of successive refinement through three levels of model: conceptual models, logical models and physical models. The table compares the different features of these three types of data models

Feature	Conceptual	Logical	Physical
Entity Names	✓	✓	
Entity Relationships	✓	✓	
Attributes		✓	
Primary Keys		✓	✓
Foreign Keys		✓	✓
Table Names			✓
Column Names			✓
Column Data Types			✓



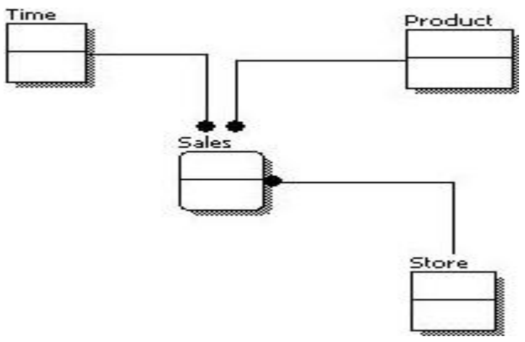
Data models
Conceptual data models (Object based data model)
Logical data models (Record based data model)
Physical data models
Entity-Relationship Model (E-R diagram)
• Hierarchical model
• Network model
• Relational model
• Unifying model
• Frame memory model

Fig. Categories of Data models

Conceptual, Logical and physical data models are very different in their objectives, goals and content. Key differences noted below.

Conceptual Data Model	**Logical Data Model**	**Physical Data Model**
Includes high-level data constructs	Includes entities (tables), attributes (columns/fields) and relationships (keys)	Includes tables, columns, keys, data types, validation rules, database triggers, stored procedures, domains, and access constraints
Non-technical names, so that executives and managers at all levels can understand the data basis of Architectural Description	Uses business names for entities & attributes	Uses more defined and less generic specific names for tables and columns and any company defined standards
Uses general high-level data constructs from which Architectural Descriptions are created in non-technical terms	Is independent of technology (platform, DBMS)	Includes primary keys and indices for fast data access.
May not be normalized	Is normalized to fourth normal form (4NF)	May be de-normalized to meet performance requirements based on the nature of the database.

A. Conceptual data model



After identification of user requirements, a conceptual data model identifies the highest-level relationships between the different entities of the problem domain. The information provide via the conceptual data model is the entities that describe the data and the relationships between those entities. No other information is shown through the conceptual data model, as shown in figure. Due to its highly abstract nature (external overall entities), it may be referred to as a *conceptual model*.

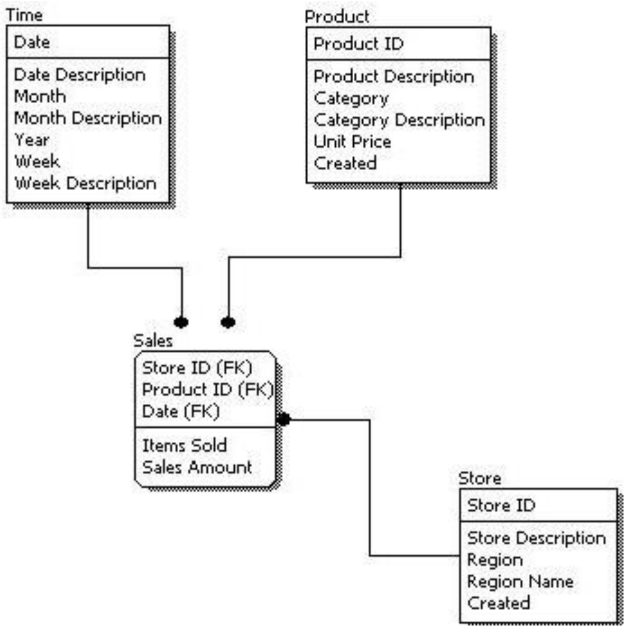
The *object based models* are relate to the conceptual model on which data are treated as objects having all their operating principles along with data values. The one common example of object based model is the *Entity-Relation (E-R) model*.

B. Logical data model

A logical data model describes the data in as much detail as possible, without regard to how they will be physical implemented in the database. The given example illustrate a logical data model for various entities represented by conceptual data models above.

Features of a logical data model include:

- Includes all entities and relationships among them.
- All attributes for each entity are specified.
- The primary key for each entity is specified.
- Foreign keys (keys identifying the relationship between different entities) are specified.
- Normalization occurs at this level.



The ***steps*** for designing the logical data model are as follows:

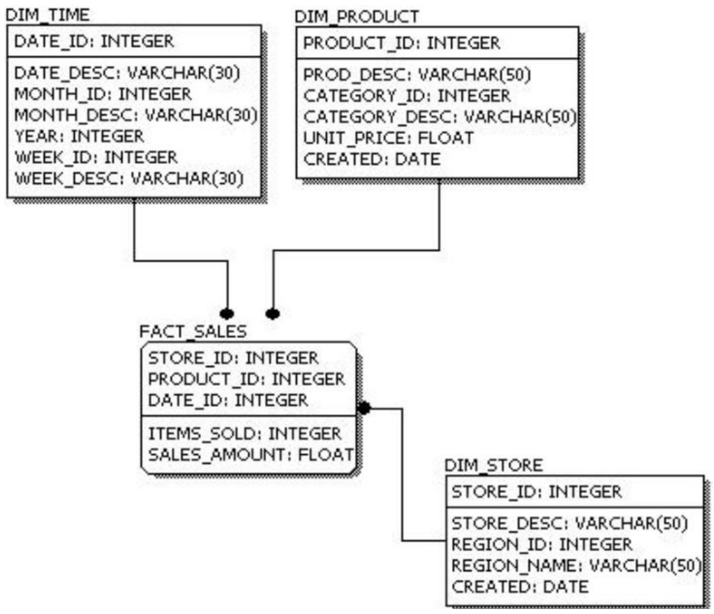
1. Specify primary keys for all entities.
2. Find the relationships between different entities.
3. Find all attributes for each entity.
4. Find the relationships.
5. Normalization.

The logical data models are based on the mathematical principles and realizable to physical data model. The *record based models* are related to logical data model. The three most widely-accepted models under record-based logical models are: *Relational model*, *Network model*, and *Hierarchical model*.

C. **Physical data model**

Physical data models are used to describe data at the lowest level. Unifying model and frame memory are follows under physical data model. A physical database model shows all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships between tables. Features of a physical data model include:

- Specification of all tables and columns.
- Foreign keys are used to identify relationships between tables.
- De-normalization may occur based on user requirements.
- Physical considerations may cause the physical data model to be quite different from the logical data model.
- Physical data model will be different for different RDBMS. For example, data type for a column may be different between Oracles, DB2 etc.



The ***steps*** for physical data model design are as follows:

1. Convert entities into tables.
2. Convert relationships into foreign keys.
3. Convert attributes into columns.
4. Modify the physical data model based on physical constraints / requirements.

2.2 Conceptual Data Model

A. Entity Relationship Model

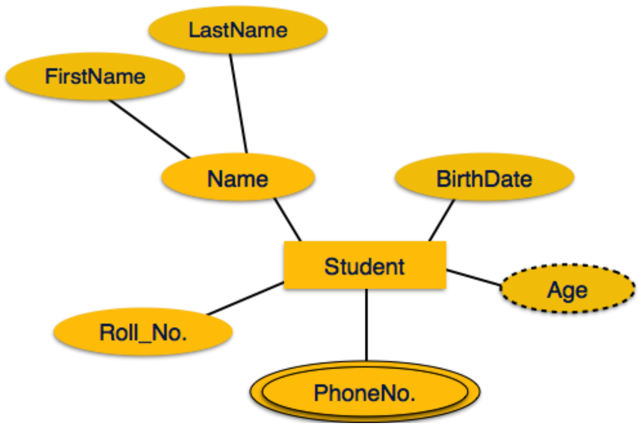
Entity relationship model defines the conceptual view of database. It works around real world entity and association among them. At view level, ER model is considered well for designing databases. E-R model describes the design of database in terms of entities and relationship among them. An entity is a "thing" or "object" in real world that are distinguishable from other objects. An entity is describes by a set of attributes or properties.

Entity: An entity may be an object with a physical existence that can be easily identifiable and distinguishable. For example, in a school database, student, teachers, class and course offered can be considered as entities. All entities have some attributes or properties that give them their identity. An entity set is a collection of similar types of entities. Entity set may contain entities with attribute sharing similar values. For example, Students set may contain all the student of a school; likewise Teachers set may contain all the teachers of school from all faculties. Entities sets need not to be disjoint.

Student
Teacher
Projects

Attributes: Entities are represented by means of their properties, called attributes. All attributes have values. For example, a student entity may have name, class, age as attributes. There exist a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Attributes are represented by means of eclipses. Every eclipse represents one attribute and is directly connected to its entity (rectangle). If the attributes are ***composite***, they are further divided in a tree like structure. Every node is then connected to its attribute. That is composite attributes are represented by eclipses that are connected with an eclipse. ***Multivalued*** attributes are depicted by double eclipse. ***Derived*** attributes are depicted by dashed eclipse.



For *example*

- Attributes `account_number` and `balance` may describe entity "`account`".
- Attributes `customer_id`, `customer_name`, `customer_city` may describe entity "`customer`".

Types of attributes:

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

Entity-set and Keys
Key is an attribute or collection of attributes that uniquely identifies an entity among entity set. For example, roll_number of a student makes her/him identifiable among students.

- **Super Key:** Set of attributes (one or more) that collectively identifies an entity in an entity set.
- **Candidate Key:** Minimal super key is called candidate key that is, supers keys for which no proper subset are a superkey. An entity set may have more than one candidate key.
- **Primary Key:** This is one of the candidate key chosen by the database designer to uniquely identify the entity set.

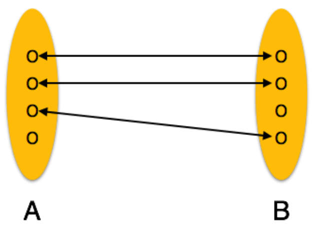
Relationship
The association among entities is called relationship. For example, employee entity has relation works_at with department. Another example is for student who enrolls in some course. Relationships are represented by diamond shaped box. Name of the relationship is written in the diamond-box. All entities (rectangles), participating in relationship, are connected to it by a line.

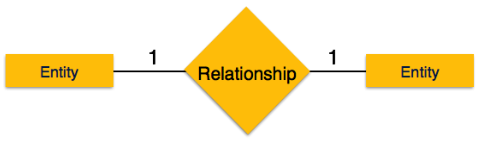
- *Relationship Set:* Relationship of similar type is called relationship set. Like entities, a relationship too can have attributes. These attributes are called descriptive attributes.
- *Degree of relationship:* The number of participating entities in a relationship defines the degree of the relationship. Binary = degree 2, Ternary = degree 3, n-ary = degree n

Constraints in E-R Model
E-R model has a capability to enforce constraints. Two most important type of constraints in E-R model are *Mapping Cardinalities* (Cardinality ratio) and *Participation Constraints*.

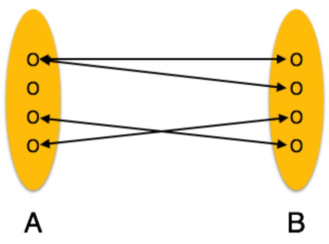
a. Mapping Cardinalities: Cardinality defines the number of entities in one entity set which can be associated to the number of entities of other set via relationship set.

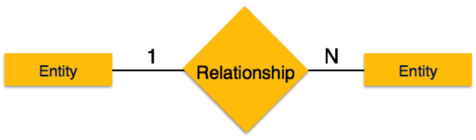
- **One-to-one:** one entity from entity set A can be associated with at most one entity of entity set B and vice versa.



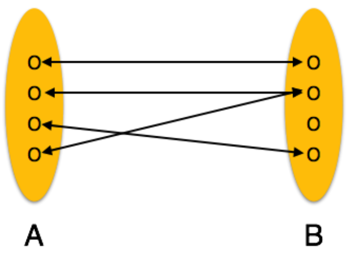


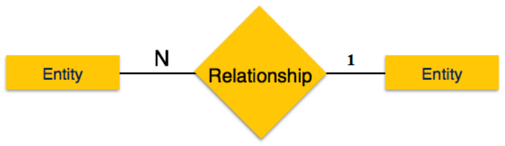
- **One-to-many:** One entity from entity set A can be associated with more than one entities of entity set B but from entity set B one entity can be associated with at most one entity.





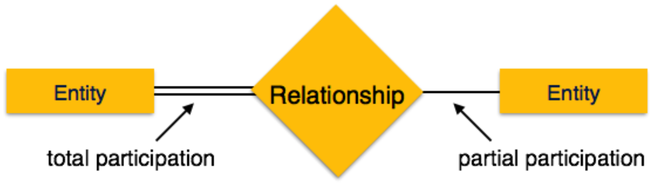
- **Many-to-one:** More than one entities from entity set A can be associated with at most one entity of entity set B but one entity from entity set B can be associated with more than one entity from entity set A.





b. Participation Constraints

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

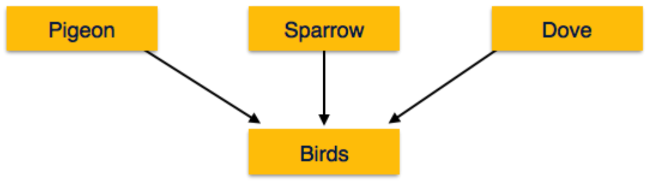


Generalization Aggregation

ER Model has the power of expressing database entities in conceptual hierarchical manner such that, as the hierarchical goes up it generalize the view of entities and as we go deep in the hierarchy it gives us detail of every entity included. Going up in this structure is called generalization, where entities are clubbed together to represent a more generalized view. For example, a particular student named, Mira can be generalized along with all the students, the entity shall be student, and further a student is person. The reverse is called specialization where a person is student, and that student is Mira.

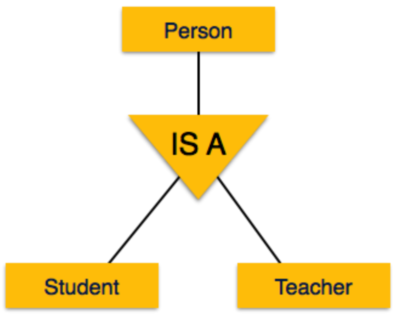
Generalization

As mentioned above, the process of generalizing entities, where the generalized entities contain the properties of all the generalized entities is called Generalization. In generalization, a number of entities are brought together into one generalized entity based on their similar characteristics. For an example, pigeon, house sparrow, crow and dove all can be generalized as Birds.



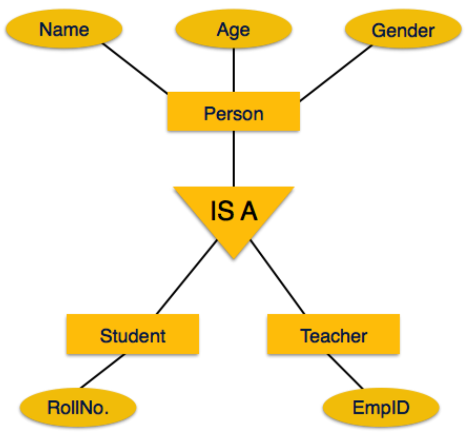
Specialization

Specialization is a process, which is opposite to generalization, as mentioned above. In specialization, a group of entities is divided into sub-groups based on their characteristics. Take a group Person for example. A person has name, date of birth, gender etc. These properties are common in all persons, human beings. But in a company, a person can be identified as employee, employer, customer or vendor based on what role do they play in company.

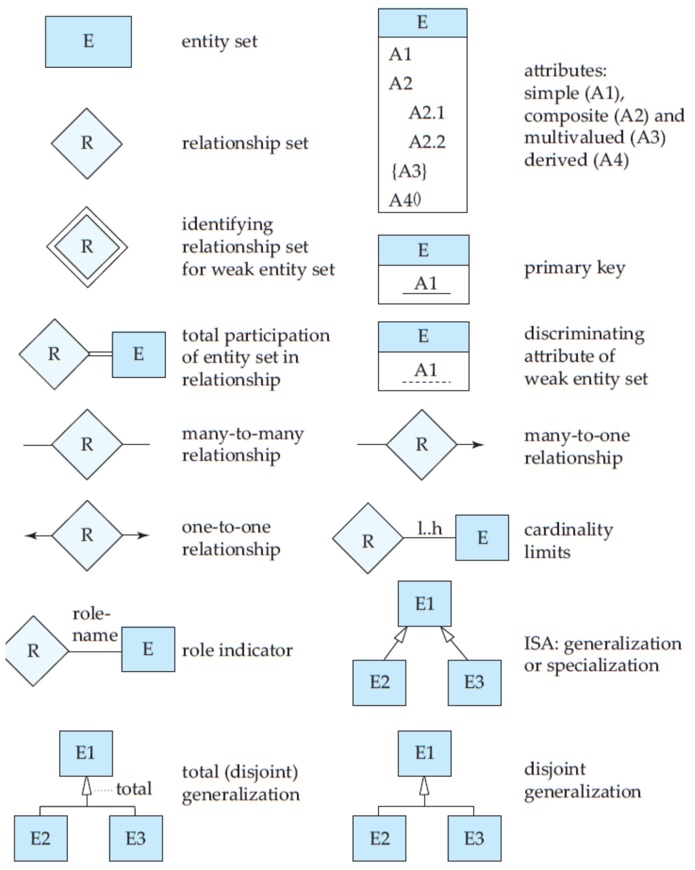


Inheritance

We use all above features of ER-Model, in order to create classes of objects in object oriented programming. This makes it easier for the programmer to concentrate on what she is programming. Details of entities are generally hidden from the user, this process known as abstraction. One of the important features of Generalization and Specialization, is inheritance, that is, the attributes of higher-level entities are inherited by the lower level entities. For example, attributes of a person like name, age, and gender can be inherited by lower level entities like student and teacher etc.

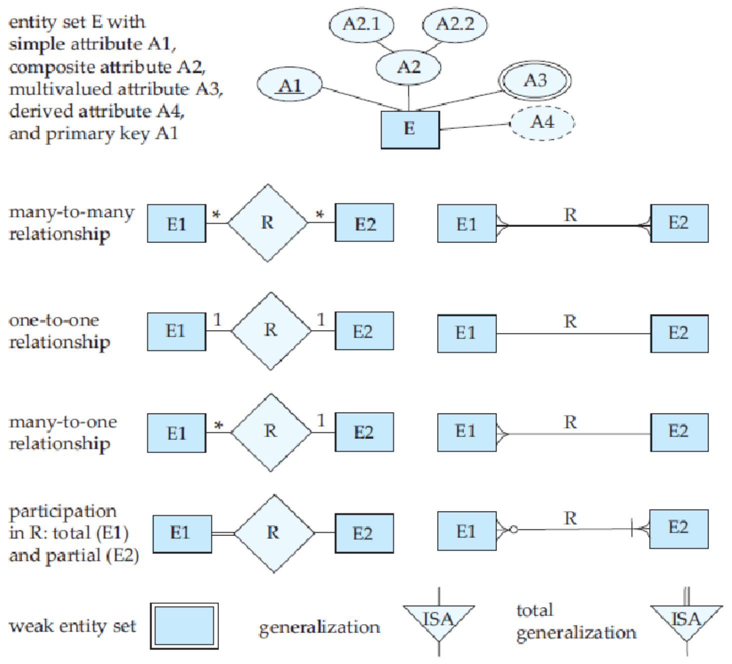


The ERM can be easily converted into a corresponding relational model because the relational model is a universally accepted state of art of database system.



E
entity set
R
relationship set
R
identifying relationship set for weak entity set
R E
total participation of entity set in relationship
R
many-to-many relationship
R
one-to-one relationship
role-name
R E
role indicator
E
A1
A2
A2.1
A2.2
{A3}
A4()
attributes: simple (A1), composite (A2) and multivalued (A3) derived (A4)
E
A1
primary key
E
A1
discriminating attribute of weak entity set
R
many-to-one relationship
R l..h E
cardinality limits
E1 E2 E3
ISA: generalization or specialization
E1 total E2 E3
total (disjoint) generalization
E1 E2 E3
disjoint generalization

Figure Symbols used in the E-R notation.

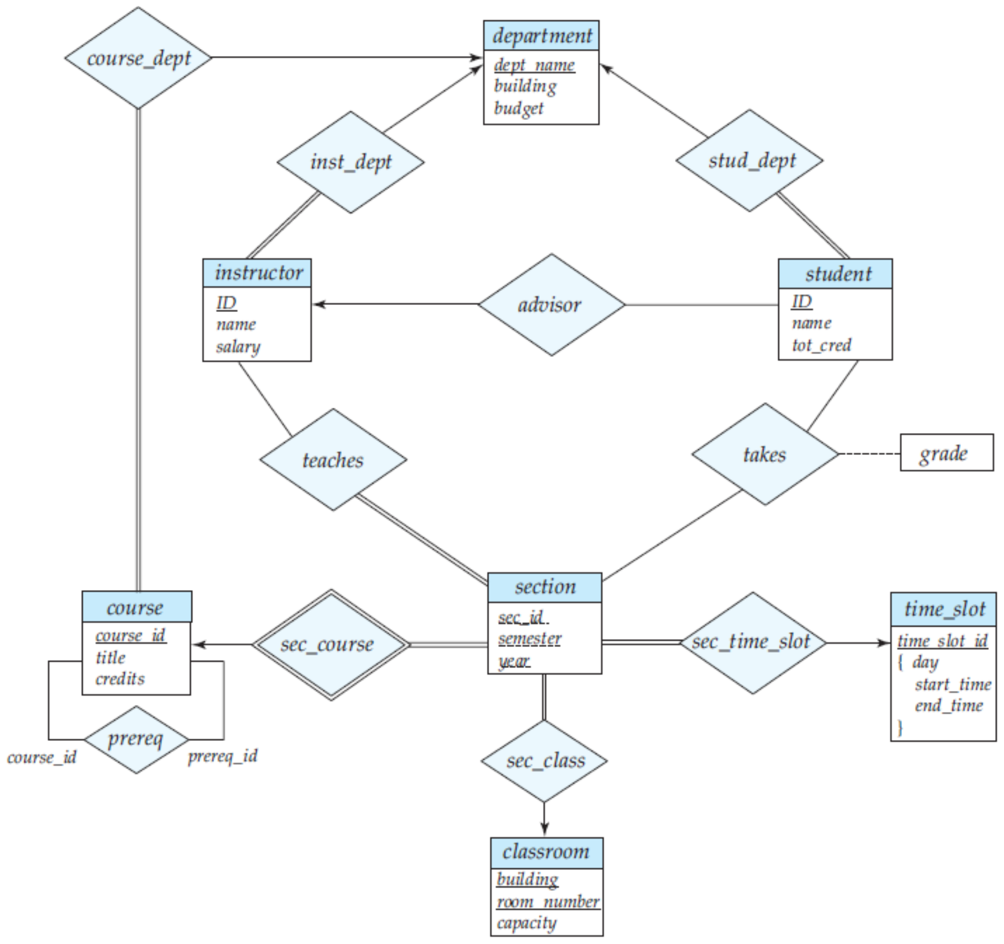


entity set E with simple attribute A1, composite attribute A2, multivalued attribute A3, derived attribute A4, and primary key A1
A2.1
A2.2
A2
A1
A3
A4
E
many-to-many relationship
E1 * R * E2
E1 R E2
one-to-one relationship
E1 1 R 1 E2
E1 R E2
many-to-one relationship
E1 * R 1 E2
E1 R E2
participation in R: total (E1) and partial (E2)
E1 R E2
E1 R E2
weak entity set
generalization
ISA
total generalization
ISA

Figure Alternative E-R notations.

Example: E-R diagram for the University Enterprise

Exercise: Draw an E-R diagram for Bank, College, Book store, Plane reservation etc.



B. Relation with UML class diagram

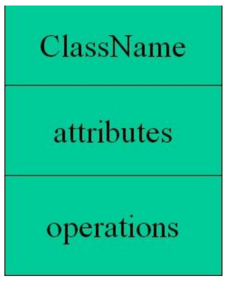
Entity-relationship diagrams help model the data representation component of a software system. Data representation is only one part of an overall system design. Other components include models of user interactions with the system, specification of functional modules of the system and their interaction, etc. The *Unified Modeling Language* (UML) is a standard developed for creating specifications of various components of a software system. Some of the parts of UML are:

- **Class diagram**: A class diagram is similar to an E-R diagram. A *class* is a description of objects that share the same attributes, operations, relationships, and semantics. The model of system using E-R diagram can be converted into its equivalent class diagram.
- **Use case diagram**: Use case diagrams show the interaction between users and the system such as withdrawing money or registering for a course.
- **Activity diagram**: Activity diagrams depict the flow of tasks between various components of a system.
- **Implementation diagram**. Implementation diagrams show the system components and their interconnections, both at the software component level and the hardware component level.

Class diagram

A *class diagram* is a *static model* that shows the classes and the relationships among classes that remain constant in the system over time. The class diagram depicts classes, which include both behaviors and states, with the relationships between the classes. The main building block of a class diagram is the *class*, which stores and manages information in the system. During analysis, classes refer to the people, places, events, and things about which the system will capture information. Later, during design and implementation, classes can refer to implementation-specific artifacts such as windows, forms, and other objects used to build the system.

A *class* is a description of a set of objects that share the same attributes, operations, relationships, and semantics. Graphically, a class is rendered as a rectangle, with the class's name at top, attributes in the middle, and operations at the bottom. The class is only a conceptual model (or blue print) of objects that does not appears on run time as the objects. Thus, an object is a run time entity that has physical appearance at run time but the class has not. Class Diagram models class structure and contents using design elements such as classes, packages and objects. It also displays relationships such as containment, inheritance, associations and others.

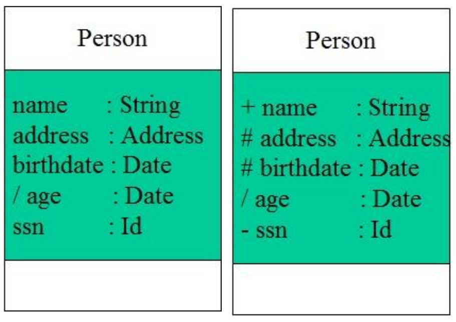


a. Class Names

The name of the class is the only required tag in the graphical representation of a class. It always appears in the top-most compartment.

b. Class Attributes

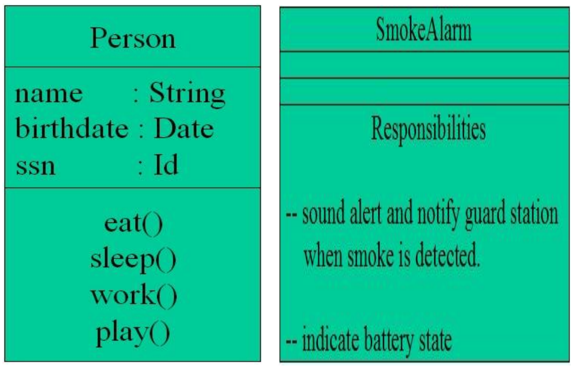
An *attribute* is a named property of a class that describes the object being modeled. In the class diagram, attributes appear in the second compartment just below the name-compartment. A *derived* attribute is one that can be computed from other attributes, but doesn't actually exist. For e.g., a Person's age can be computed from his birth date. A derived attribute is designated by a preceding '/' as in: ***/ age: Date***



Attributes can be: + ***public, # protected, - Private & / derived***

c. **Class Operations**

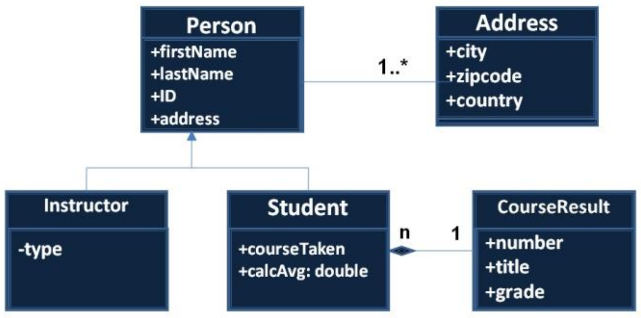
Operations describe the class behavior and appear in the third compartment. You can specify an operation by stating its signature: listing the name, type, and default value of all parameters, and, in the case of functions, a return type. A class may also include its responsibilities in a class diagram. A responsibility is a contract or obligation of a class to perform a particular service.

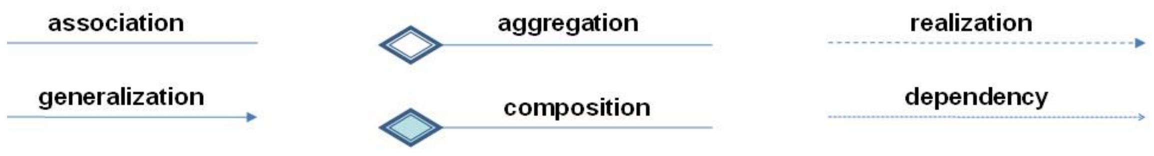


d. **Relationships**

In UML, object interconnections (logical or physical), are modeled as relationships. There are three kinds of relationships in UML: dependencies, generalizations, and associations.

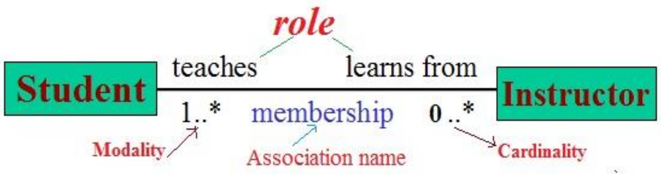
- **Association** (Loosely bounded) – Two classes are associated if one class has to know about the other.
- **Aggregation** (Belong to) –An association is one in which one class belongs to a collection in the other.
- **Generalization** (Inheritance) – An inheritance link indicating one class is a base class of the other.
- **Dependency** –A labeled dependency between classes (such as friend, classes)
- **Composition** – Has a relation





Note:

- **Cardinality:** - There may or may not be the association to other (i.e. 0 or more relation or option provided).
- **Modality: -** There must have at least one relation or association to other item (i.e. 1 relation is compulsory).

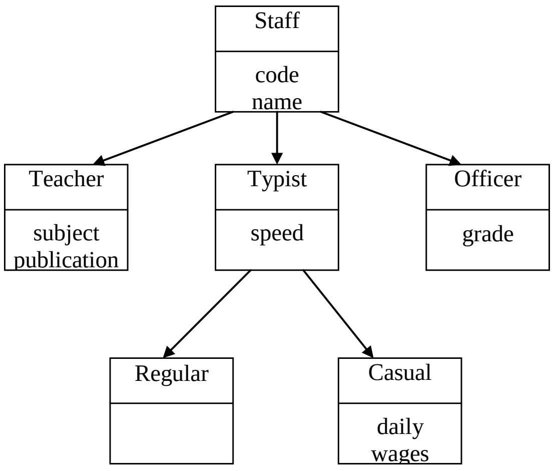


2.3 Logical Data Models

The logical models or record based models are used to describe data at the logical and view levels. These models are so named because the database is structured in fixed format records of several types. The available data model for logical view of data are described below:

A. Hierarchical Data Model

It is one of the oldest database model defined in late 1950s by IBM. The hierarchical data model organizes data in a tree structure. There is a hierarchy of parent and child data segments. The nodes of the tree represent record types. Hierarchical tree consists one root record type along with zero more occurrences of its dependent subtree and each dependent subtree is again hierarchical. In hierarchical model, no dependent record can occur without its parent record. Furthermore, no dependent record may be connected to more than one parent record. The segment without a parent is called the root and the segment that have no children are called as the leaves of the hierarchical model.



The hierarchical model are widely used as semantic models in practice because many of the real-world phenomenon are hierarchical in nature like administrative structure, biological structures, political and social structures. These models are also used in physical model of disk storage e.g. IMS by IBM, IBM and system 2000, NOMAD by NCSS.

Advantages

- **Simplicity:** Since the database is based on the hierarchical structure, the relationship between the various layers is logically simple.
- **Data sharing:** Since all data are held in a common database hence the data sharing becomes practical.
- **Data Security:** Hierarchical model was the first database model that offered the data security that is provided by the DBMS.
- **Data Integrity**: The word integrity refers to the *no modification* on data. Since it is based on the parent child relationship, there is always a link between the parent segment and the child segment under it.
- **Efficiency**: It is very efficient because when the database contains a large number of one-to-many (1:N) relationship and when the user require large number of transaction.

Disadvantages

- **Implementation complexity**: Although it is simple and easy to design, it is quite complex to implement.
- **Inflexibility**: The changes on new relation or segments often yield very complex system management task. A deletion of one segment may lead to the deletion of all segments under it.
- **Database Management Problem**: If you make any changes in the database structure, then you need to make changes in the entire application program that access the database.
- **Lack of Structural Independence**: there is lack of structural independence because when we change the structure then it becomes compulsory to change the application too.
- **Operational Anomalies:** Hierarchical model suffers from the insert, delete and update anomalies, also retrieval operation is difficult. So this model is not suitable for all cases.
- **No standards:** This model have not any standardization.
- **Implementation limitation:** The hierarchical model only have one-to-many (1:N) relation format but many real-life problem also perform many-to-many (M:N) relationship which is not possible with this model.

B. Network Data Model

In late 1960s, the Database Task Group (DBTG) formalized this model. Network data model is the extension of hierarchical data model. The network model replaces the hierarchical tree with a *graph* thus allowing more general connections among nodes. It supports many-to-many relationships i.e. it allows a record to have more than one parent.

In network database terminology, a relationship is a set. Each set is made up of at least two types of records:

- An *owner* record (equivalent to *parent* in hierarchical model)
- A *member* record (equivalent to *child* in hierarchical model)

Data in the network are represents by collection of records and relationship among data are represented by links. This links can view as a *pointer*.

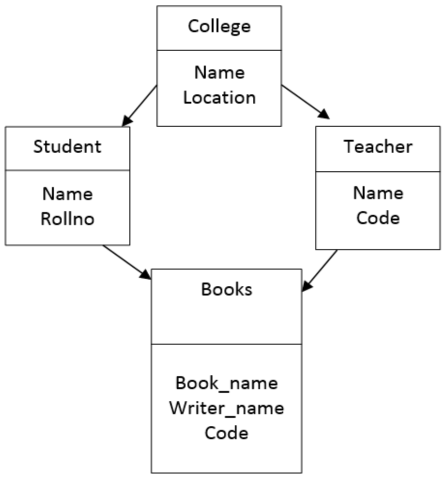


Figure: Network Data Model

Advantages

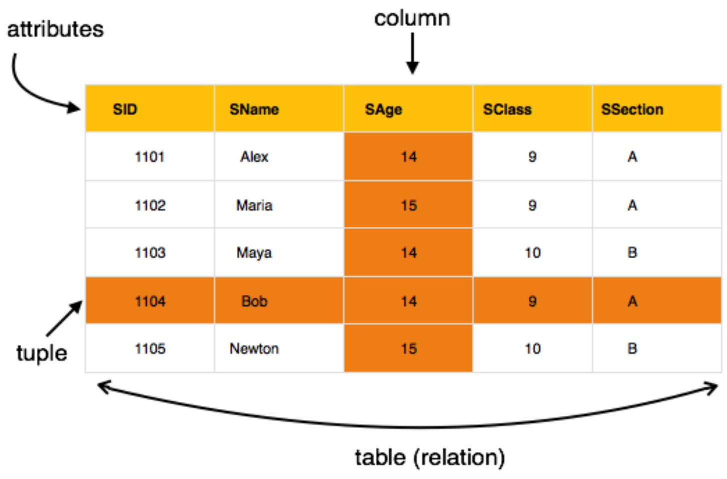
- **Conceptual Simplicity:** just like hierarchical model it also simple and easy to implement.
- **Capability to handle more relationship types**: the network model can handle one to one1:1 and many to many N: N relationship.
- **Ease to access data:** the data access is easier than the hierarchical model.
- **Data Integrity:** Since it is based on the parent child relationship, there is always a link between the parent segment and the child segment under it.
- **Data Independence:** The network model is better than hierarchical model in case of data independence.
- **Database standard:** Network model have universal standards formulated by the DBTG.

Disadvantages

- **System Complexity**: All the records have to maintain using pointers thus the database structure becomes more complex.
- **Operational Anomalies:** As discussed earlier in network model large number of pointers is required so insertion, deletion and updating more complex.
- **Not user-friendly:** The network model is not a design for user-friendly system and is a highly skilled-oriented system.
- **Absence of structural Independence:** There is lack of structural independence because when we change the structure then it becomes compulsory to change the application too.

C. Relational Model

A relational data model group its data items into one or more independent tables that can be related to one another by using fields common to each related table. This model provides a means of describing data with its natural structure only that yields maximal independence between the machine representation and organization of data. It also provides a sound basis of treating derivability, redundancy and consistency of relations.



SID	SName	SAge	SClass	SSection
1101	Alex	14	9	A
1102	Maria	15	9	A
1103	Maya	14	10	B
1104	Bob	14	9	A
1105	Newton	15	10	B

This model is based on first-order predicate logic and defines table as an n-ary relation. The main highlights of this model are:

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

Advantages

- **Conceptual Simplicity**: We have seen that both the hierarchical and network models are conceptually simple, but relational model is simpler than both of those two.
- **No anomalies**: Unlike traditional model, the relational model does not suffer from insert, delete, and update anomalies. The retrieval operation is very simple and symmetric.
- **Structural Independence:** In the Relational model, changes in the structure do not affect the data access.
- **Easier design, implementation, administration, maintenance and usage.**
- **Ad hoc query capability**: the presence of very powerful, flexible and easy to use query capability is one of the main reasons for the immense popularity of the relational database model.

Disadvantages

- **Hardware overheads**: The relational database systems hide the implementation complexities and the physical data storage details from the user. For doing this, the relational database system need more powerful hardware computers and data storage devices.
- **Ease of design can lead to bad design:** The relational database is easy to design and use. The user needs not to know the complexities of the data storage. This ease of design and use can lead to the development and implementation of the very poorly designed database management system.

Hierarchical Data Model	**Network Data Model**	**Relational Data Model**
Relationship between records is of the parent-child type (Trees)	Relationship between records is expressed in the form of pointers or links (Graphs)	Relationship between records is presented by a relation that contains a key for each record involved in the relationship.
Many-to-many relationship cannot be expressed in this model.	Many-to-many relationship can also be implemented.	Many-to-many relationship can be easily implemented.
It is simple, straightforward and natural method of implementing record relationships.	Record relationship implementation is very complex due to the use of pointers.	Relationship implementation is very easy through the use of a key or composite key field (s).
This type of model is useful only when there is some hierarchical character in the database.	Network model is useful for representing such records which have many-to-many relationship.	Relational model is useful for representing most of the real world objects and relationship among them.
In order to represent the links among records, pointers are used. Thus relations among records are physical.	In Network model also the record relations are physical.	Relational model does not maintain physical connection among records. Data is organized logically in the form of rows and columns, and stored in table.
Searching for record is very difficult since one can retrieve a child only after going through its parent record.	Searching a record is easy since there are multiple access paths to the data elements.	A unique indexed key field is used to search for a data elements.