

## Unit -II

### Data modelling

- A data model is a picture or description which shows how the data is to be arranged to achieve a given task.
- It is a clear model which specifies how the data items are arranged in a given model.
- Some data models which gives a clear picture which shows the manner in which the data records are connected or related within a file structure. These are called structural data models.
- Dbms organize and structure data so that it can be retrieved and manipulated by different users and application programs.
- The data structures and access techniques provided by a particular dbms are called its data model.
- A data model determined both the personality of a dbms and the applications for which it is particularly well suited.

### Advantages

- 1. Simplicity:** since the database is based on the hierarchical structure, the relationship between the various layers is logically simple.
- 2. Data security:** hierarchical model was the first database model that offered the data security that is provided by the dbms.
- 3. 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.
- 4. Efficiency:** it is very efficient because when the database contains a large number of 1:n relationship and when the user require large number of transaction.

### Disadvantages :

- 1. Implementation complexity:** although it is simple and easy to design, it is quite complex to implement.

**2. 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.

**3. 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.

**4. Operational anomalies:** hierarchical model suffers from the insert, delete and update anomalies, also retrieval operation is difficult.

### **Basic building blocks**

The basic building blocks of all data models are entities, attributes, relationships, and constraints.

#### **Entity:**

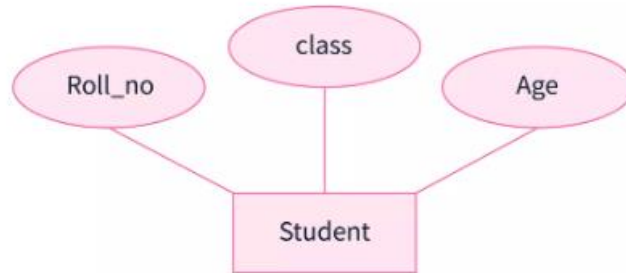
- An entity is anything (a person, a place, a thing, or an event) about which data are to be collected and stored.
- An entity represents a particular type of object in the real world. Because an entity represents a particular type of object, entities are —distinguishable||— that is, each entity occurrence is unique and distinct.
- For example, a customer entity would have many distinguishable customer occurrences, such as ajay, pravin, aniket, etc. Entities may be physical objects, such as customers or products, but entities may also be abstractions, such as flight routes or musical concerts.

#### **Attribute:**

- An attribute is a characteristic of an entity.
- For example, a customer entity would be described by attributes such as customer last name, customer first name, customer phone, customer address, and customer credit limit.
- Attributes are the equivalent of fields in file systems.
- An entity is represented by a set of attributes, that is descriptive properties possessed by all members of an entity set
- Domain – the set of permitted values for each attribute attribute types:

## 1. Simple and composite attributes

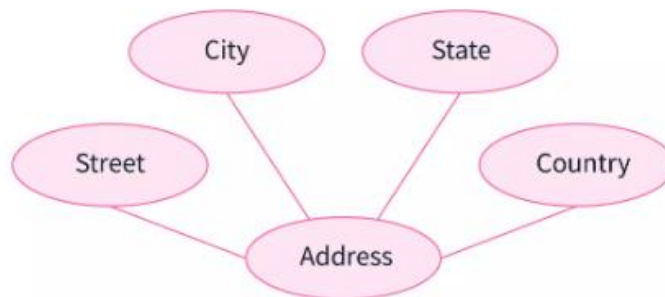
- simple attributes in an er model diagram are independent attributes that can't be classified further and also, can't be subdivided into any other component. These attributes are also known as **atomic attributes**.



- As we can see in the above example, student is an entity represented by a rectangle, and it consists of attributes: roll\_no, class, and age. Also, there is a point to be noted that we can't further subdivide the roll\_no attribute and even the other two attributes into sub-attributes. Hence, they are known as simple attributes of the student entity.

## 2. Composite attributes

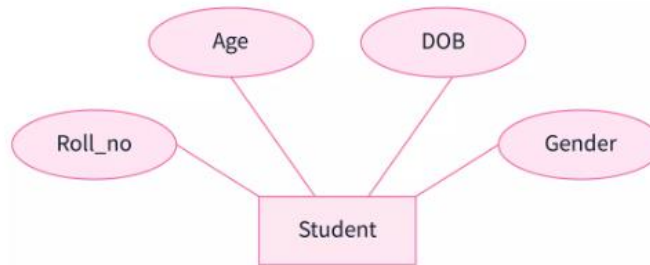
- Composite attributes have opposite functionality to that of simple attributes as we can further subdivide composite attributes into different components or sub-parts that form simple attributes.
- **in simple terms, composite attributes are composed of one or more simple attributes.**



- As we can see in the above example, address is a **composite attribute** represented by an elliptical shape, and it can be further subdivided into many simple attributes like street, city, state, country, landmark, etc.

### 3. Single-valued attributes

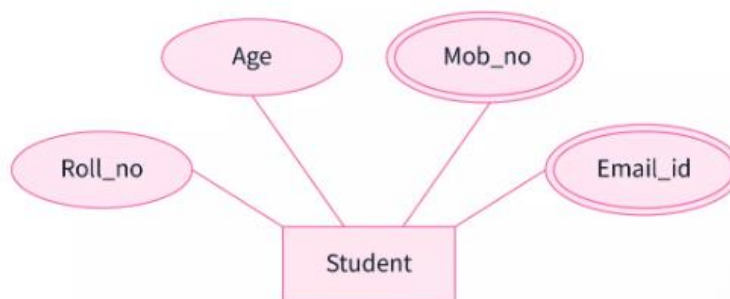
- single valued attributes are those attributes that consist of a single value for each entity instance and can't store more than one value. The value of these single-valued attributes always remains the same, just like the name of a person.



- Each entity instance can have only one roll\_no, which is a unique, single dob by which we can calculate age and also fixed gender. Also, we can't further subdivide these attributes, and hence, they are **simple as well as single-valued attributes**.

### 4. Multi-valued attributes

- multi-valued attributes have opposite functionality to that of single-valued attributes, and as the name suggests, multi-valued attributes can take up and store more than one value at a time for an entity instance from a set of possible values. These attributes are represented by **co-centric elliptical shape** to

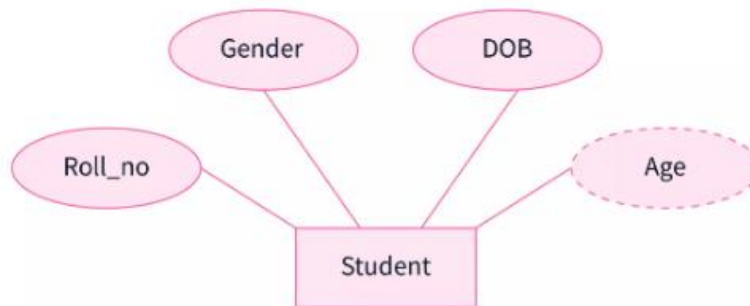


represent multi-valued attributes inside it.

- As we can see in the above example, the student entity has four attributes: roll\_no and age are simple as well as single-valued attributes as discussed above but mob\_no and email\_id are represented by co-centric ellipse **are multi-valued attributes**.

### 5. Derived attributes

- Derived attributes are those attributes whose values can be derived from the values of other attributes. They are always dependent upon other attributes for their value.
- **for example**, as we were discussing above, dob is a single-valued attribute and remains constant for an entity instance. From dob, we can derive the age attribute, which changes every year, and can easily calculate the age of a person from his/her date of birth value. Hence, the age attribute here is derived

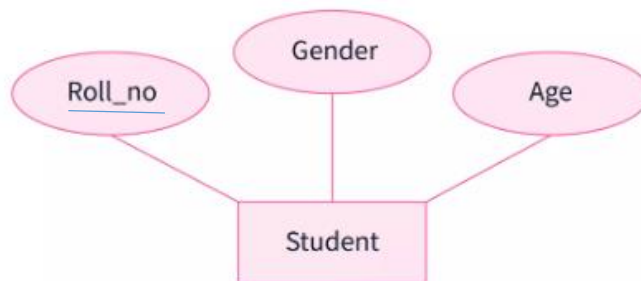


attribute from the dob single-valued attribute.

- **Derived attributes are always represented by dashed or dotted elliptical shapes.**

## 6. Key attributes

- key attributes are special types of attributes that act as the primary key for an entity and they can uniquely identify an entity from an entity set. The values that key attributes store must be unique and non-repeating.
- 

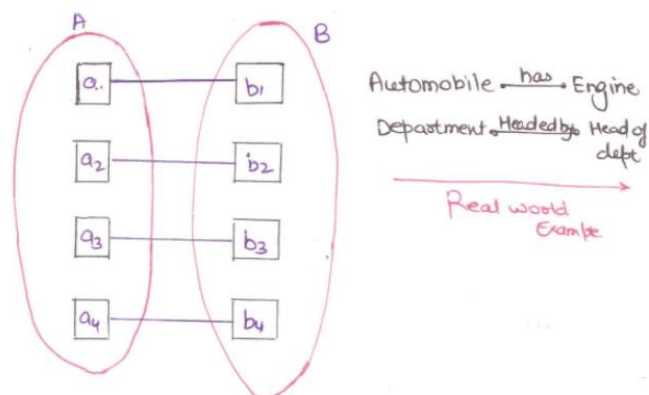


## Relationship:

- A relationship describes an association among entities.
- For example, a relationship exists between customers and agents that can be described as follows: an agent can serve many customers, and each customer may be served by one agent.
- Data models use three types of relationships: **one-to-many, many-to-many, and one-to-one.**
- Database designers usually use the shorthand notations 1:m, m:n and 1:1 respectively. (although the m:n notation is a standard label for the many-to-many relationship, the label m:m may also be used.)

## Mapping cardinalities

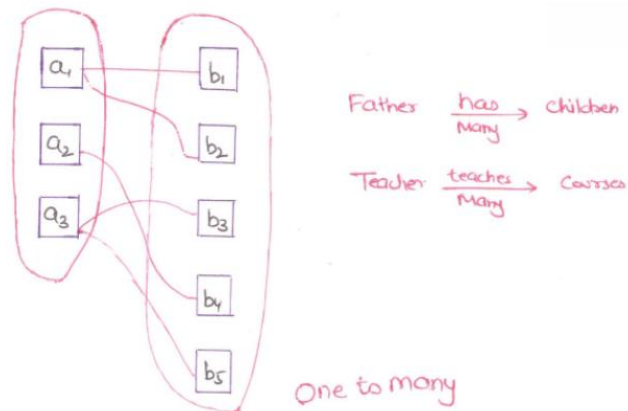
- The relationship set advisor, between the instructor and student entity sets may be one-to-one, one-to-many, many-to-one, or many-to-many.
- To distinguish among these types, we draw either a directed line ( $\rightarrow$ ) or an undirected line ( $—$ ) between the relationship set and the entity set in question, as follows:
- **One-to-one mapping cardinality**
- an entity in a is associated with "at most" one entity in b and an entity in b is



associated with at most one entity in a.

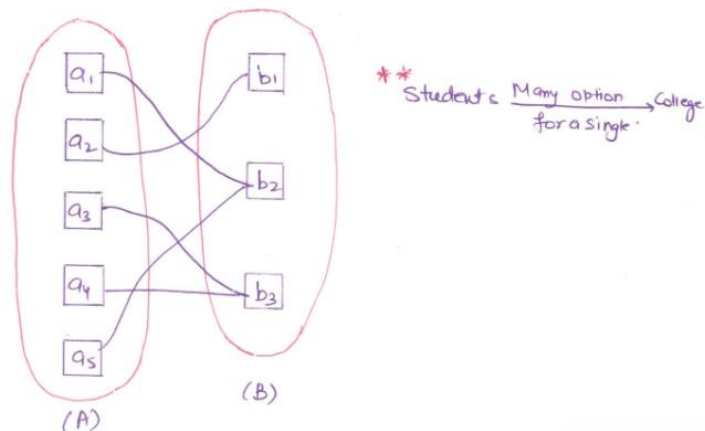
- **One to many mapping cardinality**

- an entity in a is associated with any number (zero more) of entities b. An entity in b, however, can be associated with at most one entity in a.

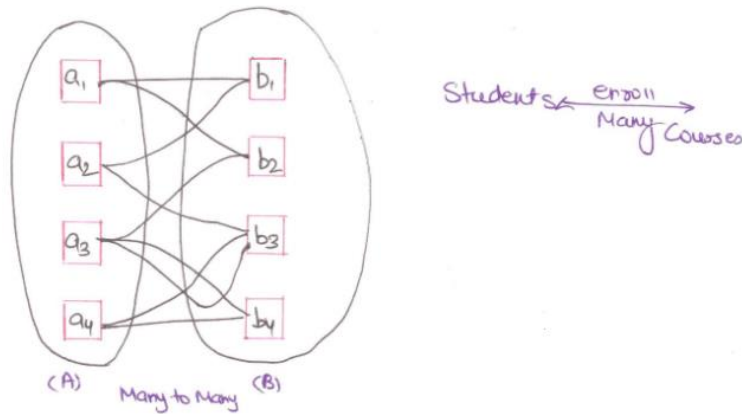


- **Many to one mapping cardinality**

- an entity in a is associated with at most one entity in b. An entity in b, however can be associated with any number (zero or more) of entities in a.



- **Many to many mapping cardinality**
- an entity in a is associated with any number (zero or more) of entities in b, and an entity in b is associated with any number (zero or more) of entities in a.



### Constraints

- A constraint is a restriction placed on the data. Constraints are important because they help to ensure data integrity.
- Constraints are normally expressed in the form of rules.
- For example:
- An employee's salary must have values that are between 6,000 and 350,000.
- A student's gpa must be between 0.00 and 4.00.
- Each class must have one and only one teacher.

What are the Constraints of DBMS?

In DBMS, constraints refer to limitations placed on data or data processes. This indicates that only a particular type of data may be entered into the database or that only a particular sort of operation can be performed on the data inside.

Constraints thereby guarantee data accuracy in a database management system (DBMS).

### The following can be guaranteed via constraints

**Data Accuracy** – Data accuracy is guaranteed by constraints, which make sure that only true data is entered into a database. For example, a limitation may stop a user from entering a negative value into a field that only accepts positive numbers.

**Data Consistency** – The consistency of data in a database can be upheld by using constraints. These constraints are able to ensure that the primary key value in one table is followed by the foreign key value in another table.



**Data integrity** – The accuracy and completeness of the data in a database are ensured by constraints. For example, a constraint can stop a user from putting a null value into a field that requires one.

### **Types of Constraints in DBMS**

- Domain Constraints
- Key Constraints
- Entity Integrity Constraints
- Referential Integrity Constraints
- Tuple Uniqueness Constraints

#### **Domain Constraints**

In a database table, domain constraints are guidelines that specify the acceptable values for a certain property or field. These restrictions guarantee data consistency and aid in preventing the entry of inaccurate or inconsistent data into the database. The following are some instances of domain restrictions in a DBMS –

- **Data type constraints** – These limitations define the kinds of data that can be kept in a column. A column created as VARCHAR can take string values, but a column specified as INTEGER can only accept integer values.
- **Length Constraints** – These limitations define the largest amount of data that may be put in a column. For instance, a column with the definition VARCHAR(10) may only take strings that are up to 10 characters long.
- **Range constraints** – The allowed range of values for a column is specified by range restrictions. A column designated as DECIMAL(5,2), for example, may only take decimal values up to 5 digits long, including 2 decimal places.
- **Nullability constraints** – Constraints on a column's capacity to accept NULL values are known as nullability constraints. For instance, a column that has the NOT NULL definition cannot take NULL values.
- **Unique constraints** – Constraints that require the presence of unique values in a column or group of columns are known as unique constraints. For instance, duplicate values are not allowed in a column with the UNIQUE definition.
- **Check constraints** – Constraints for checking data: These constraints outline a requirement that must hold for any data placed into the column. For instance, a column with the definition CHECK (age > 0) can only accept ages that are greater than zero.
- **Default constraints** – Constraints by default: Default constraints automatically assign a value to a column in case no value is provided. For example, a column with a DEFAULT value of 0 will have 0 as its value if no other value is specified.

#### **Key Constraints**

Key constraints are regulations that a DBMS uses to ensure data accuracy and consistency in a database. They define how the values in a table's one or more columns are related to the values in other tables, making sure that the data remains correct.

In DBMS, there are several key constraint kinds, including –

- **Primary Key Constraint** – A primary key constraint is an individual identifier for each record in a database. It guarantees that each database entry contains a single, distinct value—or a pair of values—that cannot be null—as its method of identification.
- **Foreign Key Constraint** – Reference to the primary key in another table is a foreign key constraint. It ensures that the values of a column or set of columns in one table correspond to the primary key column(s) in another table.
- **Unique Constraint** – In a database, a unique constraint ensures that no two values inside a column or collection of columns are the same.

#### Entity Integrity Constraints

A database management system uses entity integrity constraints (EICs) to enforce rules that guarantee a table's primary key is unique and not null. The consistency and integrity of the data in a database are maintained by EICs, which are created to stop the formation of duplicate or incomplete entries.

Each item in a table in a relational database is uniquely identified by one or more fields known as the primary key. EICs make a guarantee that every row's primary key value is distinct and not null. Take the "Employees" table, for instance, which has the columns "EmployeeID" and "Name." The table's primary key is the EmployeeID column. An EIC on this table would make sure that each row's unique EmployeeID value is there and that it is not null.

If you try to insert an entry with a duplicate or null EmployeeID, the database management system will reject the insertion and produce an error. This guarantees that the information in the table is correct and consistent.

EICs are a crucial component of database architecture and assist guarantee the accuracy and dependability of the data contained in a database.

#### Referential Integrity Constraints

A database management system will apply referential integrity constraints (RICs) in order to preserve the consistency and integrity of connections between tables. By preventing links between entries that don't exist from

being created or by removing records that have related records in other tables, RICs guarantee that the data in a database is always consistent.

By the use of foreign keys, linkages between tables are created in relational databases. A column or collection of columns in one table that is used as a foreign key to access the primary key of another table. RICs make sure there are no referential errors and that these relationships are legitimate.

Consider the "Orders" and "Customers" tables as an illustration. The primary key column in the "Customers" database corresponds to the foreign key field "CustomerID" in the "Orders" dataset. A RIC on this connection requires that each value in the "CustomerID" column of the "Orders" database exist in the "Customers" table's primary key column.

If an attempt was made to insert a record into the "Orders" table with a non-existent "CustomerID" value, the database management system would reject the insertion and notify the user of an error.

Similar to this, the database management system would either prohibit the deletion or cascade the deletion in order to ensure referential integrity if a record in the "Customers" table was removed and linked entries in the "Orders" table.

In general, RICs are a crucial component of database architecture and assist guarantee that the information contained in a database is correct and consistent throughout time.

#### Tuple Uniqueness Constraints

A database management system uses constraints called tuple uniqueness constraints (TUCs) to make sure that every entry or tuple in a table is distinct. TUCs impose uniqueness on the whole row or tuple, in contrast to Entity Integrity Constraints (EICs), which only enforce uniqueness on certain columns or groups of columns.

TUCs, then, make sure that no two rows in a table have the same values for every column. Even if the individual column values are not unique, this can be helpful in cases when it is vital to avoid the production of duplicate entries.

Consider the "Sales" table, for instance, which has the columns "TransactionID," "Date," "CustomerID," and "Amount." Even if individual column values could be duplicated, a TUC on this table would make sure that no two rows have the same values in all four columns.

The database management system would reject the insertion and generate an error if an attempt was made to enter a row with identical values in each

of the four columns as an existing entry. This guarantees the uniqueness and accuracy of the data in the table.

TUCs may be a helpful tool for ensuring data correctness and consistency overall, especially when it's vital to avoid the generation of duplicate entries.


### **Enhanced entity relationship model (eer model)**

- EER is a high-level data model that incorporates the extensions to the original er model.
- It is a diagrammatic technique for displaying the following concepts
  - Sub class and super class
  - Specialization and generalization
  - Union or category
  - Aggregation
- These concepts are used when the comes in eer schema and the resulting schema diagrams called as eer diagrams.

### **Features of EER model**

- EER creates a design more accurate to database schemas.
- It reflects the data properties and constraints more precisely.
- It includes all modeling concepts of the ER model.
- Diagrammatic technique helps for displaying the EER schema.
- It includes the concept of specialization and generalization.
- It is used to represent a collection of objects that is union of objects of different of different entity types.

### **Sub class and super class**

- Sub class and super class relationship leads the concept of inheritance.
- the relationship between sub class and super class is  denoted with symbol.
- **Super class**
- Super class is an entity type that has a relationship with one or more subtypes.
- An entity cannot exist in database merely by being member of any super class. For example: shape super class is having sub groups as square, circle, triangle.
- **Sub class**
- Sub class is a group of entities with unique attributes.

- sub class inherits properties and attributes from its super class. For example: square, circle, triangle are the sub class of shape super class

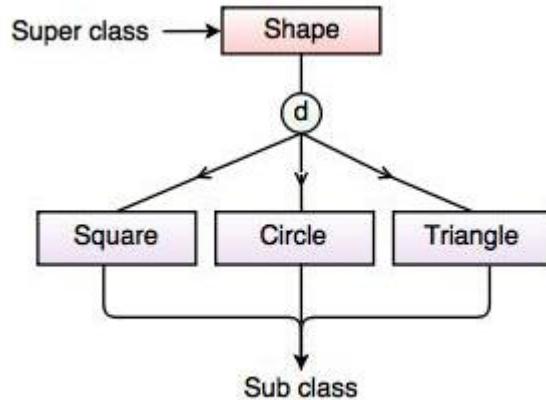
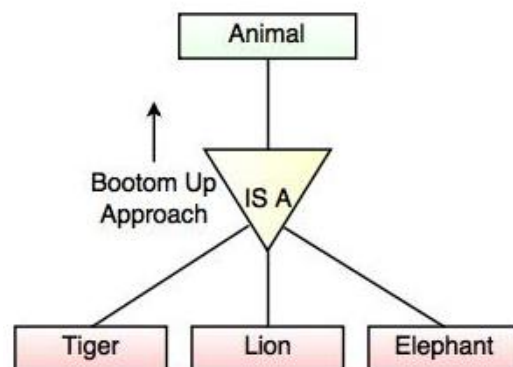


Fig. Super class/Sub class Relationship

### Specialization and generalization

- **generalization**
- Generalization is the process of generalizing the entities which contain the properties of all the generalized entities.
- It is a bottom approach, in which two lower level entities combine to form a higher level entity.
- Generalization is the reverse process of specialization.
- It defines a general entity type from a set of specialized entity type.
- It minimizes the difference between the entities by identifying the common features.
- for example:



- **Specialization**
- Specialization is a process that defines a group entities which is divided into sub groups based on their characteristic.

- It is a top down approach, in which one higher entity can be broken down into two lower level entity.
- It maximizes the difference between the members of an entity by identifying the unique characteristic or attributes of each member.
- It defines one or more sub class for the super class and also forms the superclass/subclass relationship
- for example:

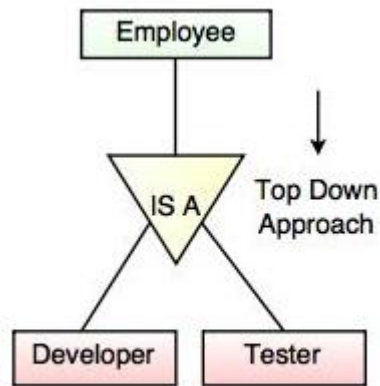


Fig. Specialization

### Category or union

- Category represents a single super class or sub class relationship with more than one super class.
- It can be a total or partial participation.
- For example car booking, car owner can be a person, a bank (holds a possession on a car) or a company.
- category (sub class) → owner is a subset of the union of the three super classes → company, bank, and person. A category member must exist in at least one of

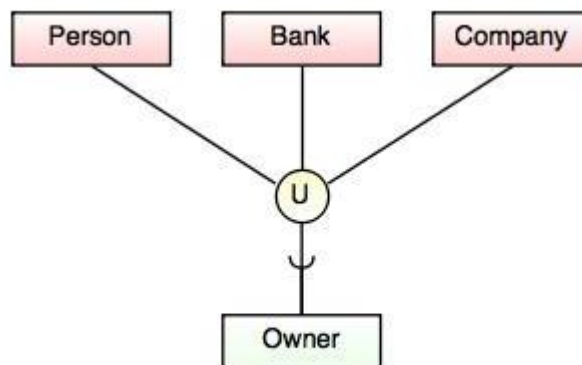


Fig. Categories (Union Type)

its super classes.

## Aggregation

- Aggregation is a process that represent a relationship between a whole object and its component parts.
- It abstracts a relationship between objects and viewing the relationship as an object.
- it is a process when two entity is treated as a single entity.

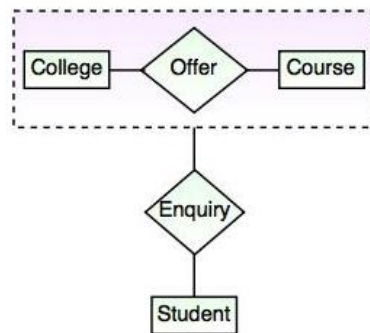


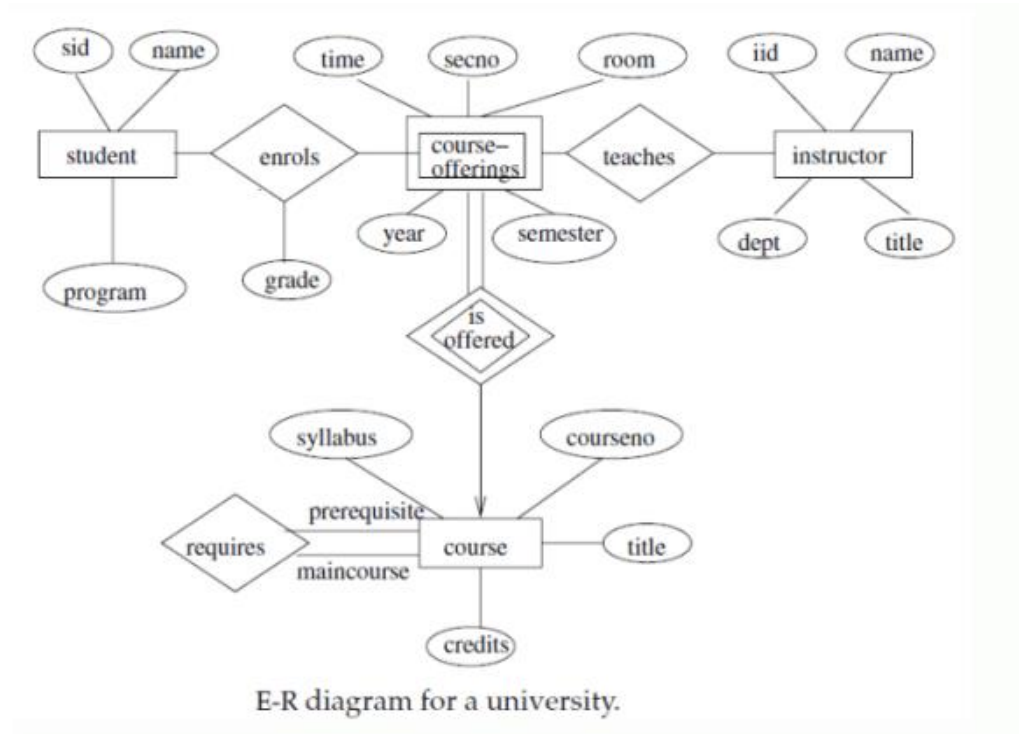
Fig. Aggregation

## Example of er diagram

Q.1) a university registrar's office maintains data about the following entities:

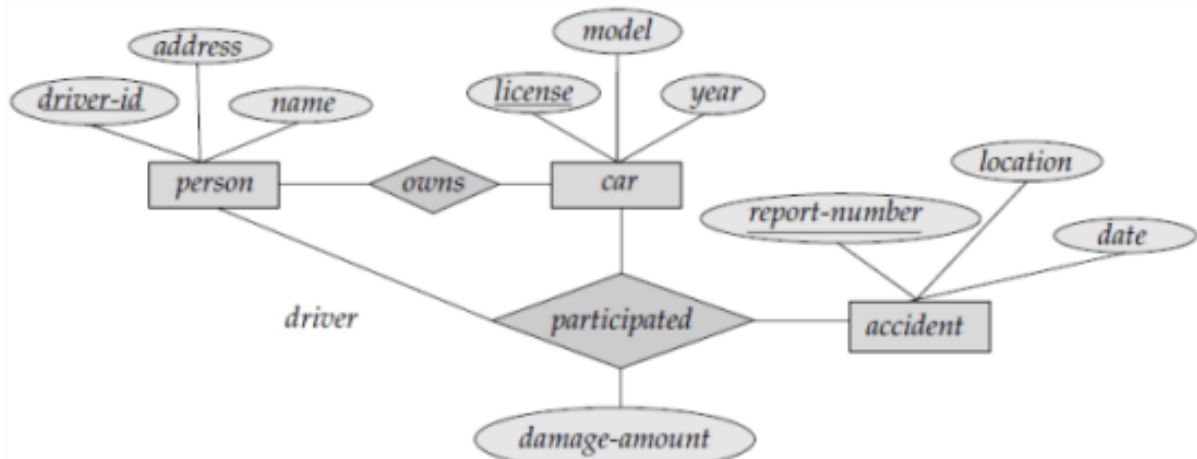
1. Courses, including number, title, credits, syllabus, and prerequisites;
2. Course offerings, including course number, year, semester, and section number, instructor(s), timings, and classroom;
3. Students, including student-id, name, and program;
4. Instructors, including identification number, name, department, and title.

further, the enrollment of students in courses and grades awarded to students in each course they are enrolled for must be appropriately modeled. Construct an e-r diagram for the registrar's office. Document all assumptions that you make about the mapping constraints.



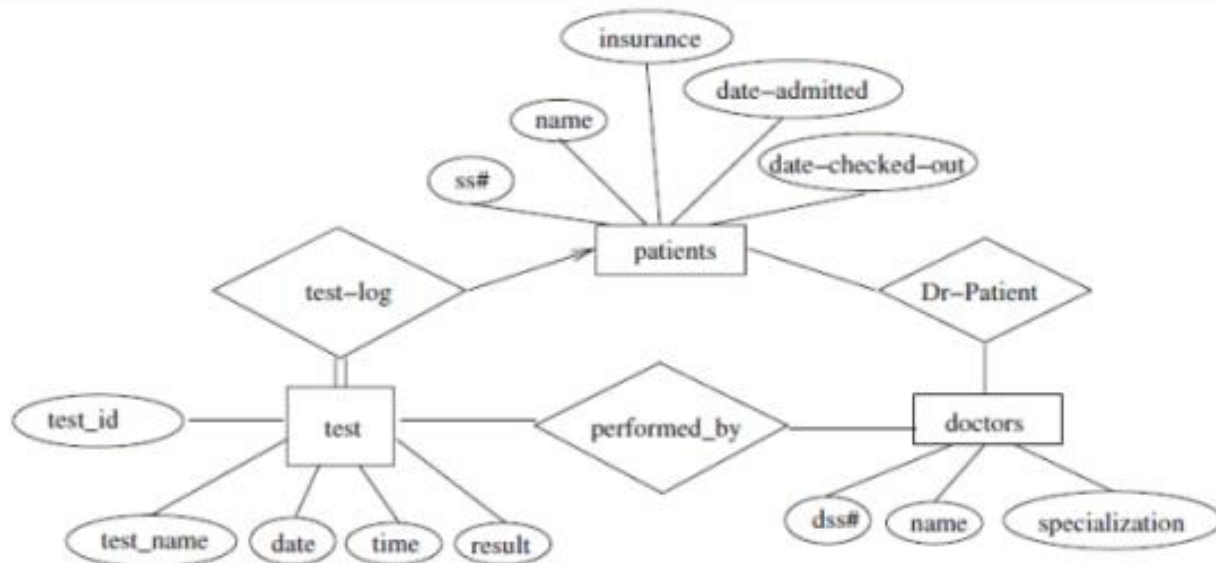
Q.2) construct an e-r diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.





E-R diagram for a Car-insurance company.

q.3) construct an e-r diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted.



E-R diagram for a hospital.

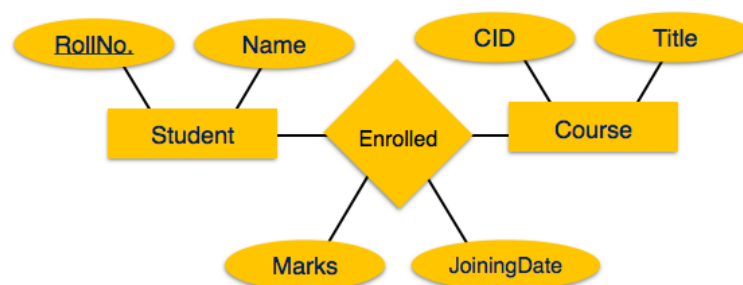
## Mapping from ER model to relational model

- **ER** model, when conceptualized into diagrams, gives a good overview of entity-relationship, which is easier to understand. **ER** diagrams can be mapped to relational schema, that is, it is possible to create relational schema using er diagram. We cannot import all the **ER** constraints into relational model, but an approximate schema can be generated.
- There are several processes and algorithms available to convert **ER** diagrams into relational schema. Some of them are automated and some of them are manual. We may focus here on the mapping diagram contents to relational basics.
- **ER** diagrams mainly comprise of –
  - Entity and its attributes
  - Relationship, which is association among entities.

## Mapping entity

- Create table for each entity.
- Entity's attributes should become fields of tables with their respective data types.
- Declare primary key.

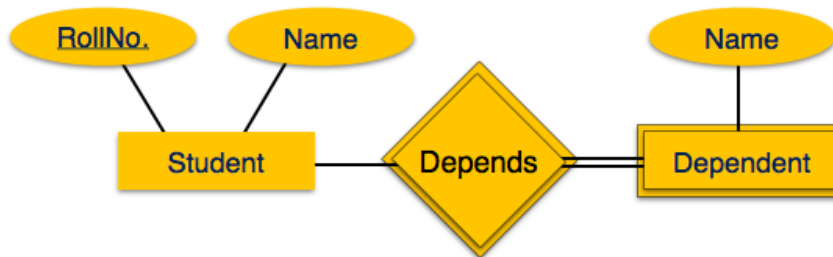
## Mapping Relationship



- Create table for a relationship.
- Add the primary keys of all participating entities as fields of table with their respective data types.
- If relationship has any attribute, add each attribute as field of table.
- Declare a primary key composing all the primary keys of participating entities.
- Declare all foreign key constraints.

## Mapping weak entity sets

- Create table for weak entity set.
- Add all its attributes to table as field.
- Add the primary key of identifying entity set.
- Declare all foreign key constraints.



## Problems on reduction of ER Diagram :

Sure, let's go through the process of reducing an Entity-Relationship (E-R) schema to tables, and I'll provide an example along the way.

**\*\*Entity-Relationship (E-R) Model Recap:\*\***

- In an E-R model, you represent entities (objects, concepts) and their relationships in a diagram.
- Entities have attributes that describe their properties.
- Relationships represent how entities are connected to each other.

**\*\*Reduction of an E-R Schema to Tables:\*\***

### 1. **\*\*Identify Entities:\*\***

- Look at the E-R diagram and identify all the entities. Each entity becomes a table in the database.

### 2. **\*\*Identify Relationships:\*\***

- Find the relationships between entities. Relationships may be one-to-one, one-to-many, or many-to-many.

### 3. **\*\*Create Tables for Entities:\*\***

- For each entity, create a table. The table will have columns for each attribute of the entity. Include a primary key to uniquely identify each record.

#### 4. **\*\*Create Tables for Relationships:\*\***

- For each many-to-many relationship, create a new table. This table will have foreign keys that reference the primary keys of the entities involved in the relationship.

#### 5. **\*\*Handle One-to-Many Relationships:\*\***

- If there's a one-to-many relationship from Entity A to Entity B, you can either:

- a. Add the primary key of Entity A as a foreign key in Entity B.
- b. Create a separate table to represent the relationship.

#### 6. **\*\*Define Constraints:\*\***

- Define constraints such as primary keys, foreign keys, unique constraints, and any other rules that ensure data integrity.

#### 7. **\*\*Normalization (Optional):\*\***

- Consider normalizing your tables to eliminate redundancy and improve data integrity. This involves breaking tables into smaller tables and establishing relationships between them.

#### **\*\*Example: Library Database\*\***

Let's use a simple example of a library database:

Entities:

1. Books
2. Authors
3. Members (Library Members)

Relationships:

- Books are written by Authors (Many-to-Many)
- Members borrow Books (Many-to-Many)

Table Creation:

1. Books Table:

- Columns: BookID (Primary Key), Title, ISBN, ...

2. Authors Table:

- Columns: AuthorID (Primary Key), Name, ...

3. Members Table:

- Columns: MemberID (Primary Key), Name, Address, ...

4. BooksAuthors Table (for Many-to-Many relationship):

- Columns: BookID (Foreign Key), AuthorID (Foreign Key)

5. MembersBooks Table (for Many-to-Many relationship):

- Columns: MemberID (Foreign Key), BookID (Foreign Key), DueDate, ...

In this example, we've created tables for each entity (Books, Authors, Members) and separate tables (BooksAuthors, MembersBooks) for the many-to-many relationships. The foreign keys in these relationship tables link the entities involved.

This process creates a relational database schema that allows you to store and manage information about books, authors, library members, and their relationships in a structured and organized manner.

### **Tabular representation of Strong, Weak entity Sets and Relationship Sets**

:

Certainly! Let's provide a tabular representation of strong and weak entity sets along with a relationship set, along with examples:

**\*\*Strong Entity Set: Student\*\***

Attribute	Data Type
-----	-----
StudentID (PK)	INT
FirstName	VARCHAR
LastName	VARCHAR
DateOfBirth	DATE

| ...

**\*\*Example:\*\***

- Strong Entity Set: Student
- Primary Key: StudentID
- Sample Data:
  - StudentID: 101
  - FirstName: John
  - LastName: Smith
  - DateOfBirth: 1995-03-15
  - ...

**\*\*Weak Entity Set: Address\*\***

Attribute	Data Type
-----	-----
AddressID (PK)	INT
Street	VARCHAR
City	VARCHAR
State	VARCHAR
PostalCode	VARCHAR
...	

**\*\*Example:\*\***

- Weak Entity Set: Address
- Partial Key: AddressID
- Sample Data:
  - AddressID: 201

- Street: 123 Main St
- City: Anytown
- State: CA
- PostalCode: 12345
- ...

**\*\*Relationship Set: Enrollment\*\***

Attribute	Data Type
-----	-----
EnrollmentID (PK)	INT
StudentID (FK)	INT
CourseID (FK)	INT
EnrollmentDate	DATE
...	

**\*\*Example:\*\***

- Relationship Set: Enrollment
- Relationship Key: EnrollmentID
- Sample Data:
  - EnrollmentID: 301
  - StudentID: 101 (Foreign Key referencing Student)
  - CourseID: 501 (Foreign Key referencing Course)
  - EnrollmentDate: 2023-09-01
  - ...

In the examples above:

- **\*\*Student\*\*** is a strong entity set with a primary key 'StudentID' that uniquely identifies each student.

- **Address** is a weak entity set with a partial key `AddressID`. It relies on a relationship with a strong entity (e.g., Student) to uniquely identify addresses for students.

- **Enrollment** is a relationship set representing the enrollment of students in courses. It has a relationship key `EnrollmentID` and foreign keys `StudentID` and `CourseID` to link to the strong entity sets Student and Course.

These examples demonstrate how strong and weak entity sets, along with relationship sets, can be represented in a database schema.