CHAPTER 3

DATA MODELING USING THE ENTITY-RELATIONAL MODEL

LEARNING OBJECTIVES

After comprehensive study of this Chapter, you will be able to:

- Using High-Level Conceptual Data Models for Database Design
- Entity Types, Entity Sets, Attributes, and Keys
- Relationship Types, Relationship Sets, Roles, and Structural Constraints
- Weak Entity Types
- ER Diagrams, Naming Conventions, and Design Issues
- Relationship Types of Degree Higher Than Two
- Subclasses, Super classes, and Inheritance
- Specialization and Generalization
- Constraints and Characteristics of Specialization and Generalization

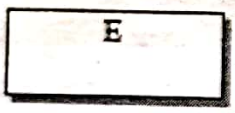
3.1 Introduction to Entity-Relationship Diagram

The E-R data models is based on a perception of real world that consist of a collection of basic objects called entities and relationship among these objects. In an E-R model a database can be modeled as a collection of entities, and relationship among entities.

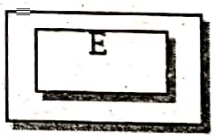
Once the entity types, relationships types, and their corresponding attributes have been identified, the next step is to graphically represent these components using entity-relationship (E-R) diagram. An E-R diagram is a specialized graphical tool that demonstrates the interrelationships among various entities of a database. It is used to represent the overall logical structure of the database. While designing E-R diagrams, the emphasis is on the schema of the database and not on the instances. This is because the schema of the database is changed rarely; however, the instances in the entity and relationship sets change frequently. Thus, E-R diagrams are more useful in designing the database. E-R diagram focuses high level database design and hides low level details of database representation therefore it can be used to communicate with users of the system while collecting information.

3.2 Symbols used in ER-Diagram/ER Naming Conventions

Entity-relationship diagrams (ERD) are essential to modeling anything from simple to complex databases, but the shapes and notations used can be very confusing. There are various types of symbols are used for ER diagram some of well-defined symbols are listed below;

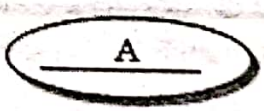


Entity set



Weak entity

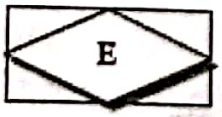
Attribute



Primary key attribute

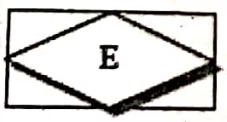
Derived attribute

Multi-valued attribute

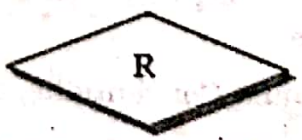


Associative entity

Relationship



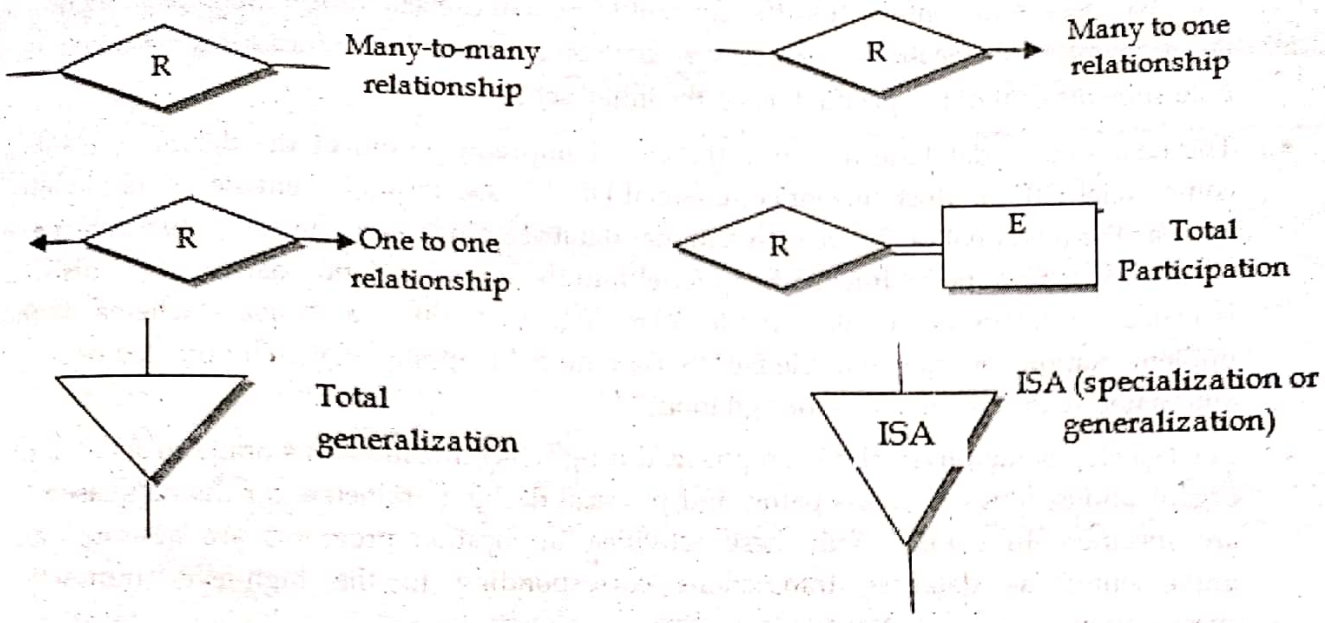
Associative entity



Relationship

Identifying Relationship

Discriminating attribute of weak entity set



3.3 Using High-Level Conceptual Data Models for Database Design

Following figure shows a simplified overview of the database design process.

- The first step shown is **requirements collection and analysis.** During this step, the database designers interview prospective database users to understand and document their data requirements. The result of this step is a concisely written set of users' requirements. These requirements should be specified in as detailed and complete a form as possible. In parallel with specifying the data requirements, it is useful to specify the known functional requirements of the application. These consist of the user-defined operations (or transactions) that will be applied to the database, including both retrievals and updates. In software design, it is common to use data flow diagrams, sequence diagrams, scenarios, and other techniques to specify functional requirements.
- Once the requirements have been collected and analyzed, the next step is to create a **conceptual schema** for the database, using a high-level conceptual data model. This step is called conceptual design. The conceptual schema is a concise description of the data requirements of the users and includes detailed descriptions of the entity types, relationships, and constraints; these are expressed using the concepts provided by the high-level data model. Because these concepts do not include implementation details, they are usually easier to understand and can be used to communicate with nontechnical users. The high-level conceptual schema can also be used as a reference to ensure that all users' data requirements are met and that the requirements do not conflict. This approach enables database designers to concentrate on specifying the properties of the data, without being concerned with storage and implementation details. This makes it is easier to create a good conceptual data-base design.
- During or after the conceptual schema design, the basic data model operations can be used to specify the high-level user queries and operations identified during functional analysis.

This also serves to confirm that the conceptual schema meets all the identified functional requirements. Modifications to the conceptual schema can be introduced if some functional requirements cannot be specified using the initial schema.

- The next step in database design is the **actual implementation of the database**, using a commercial DBMS. Most current commercial DBMSs use an implementation data model—such as the relational or the object-relational database model—so the conceptual schema is transformed from the high-level data model into the implementation data model. This step is called logical design or data model mapping; its result is a database schema in the implementation data model of the DBMS. Data model mapping is often automated or semi-automated within the database design tools.
- The last step is the **physical design phase**, during which the internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files are specified. In parallel with these activities, application programs are designed and implemented as database transactions corresponding to the high-level transaction specifications.

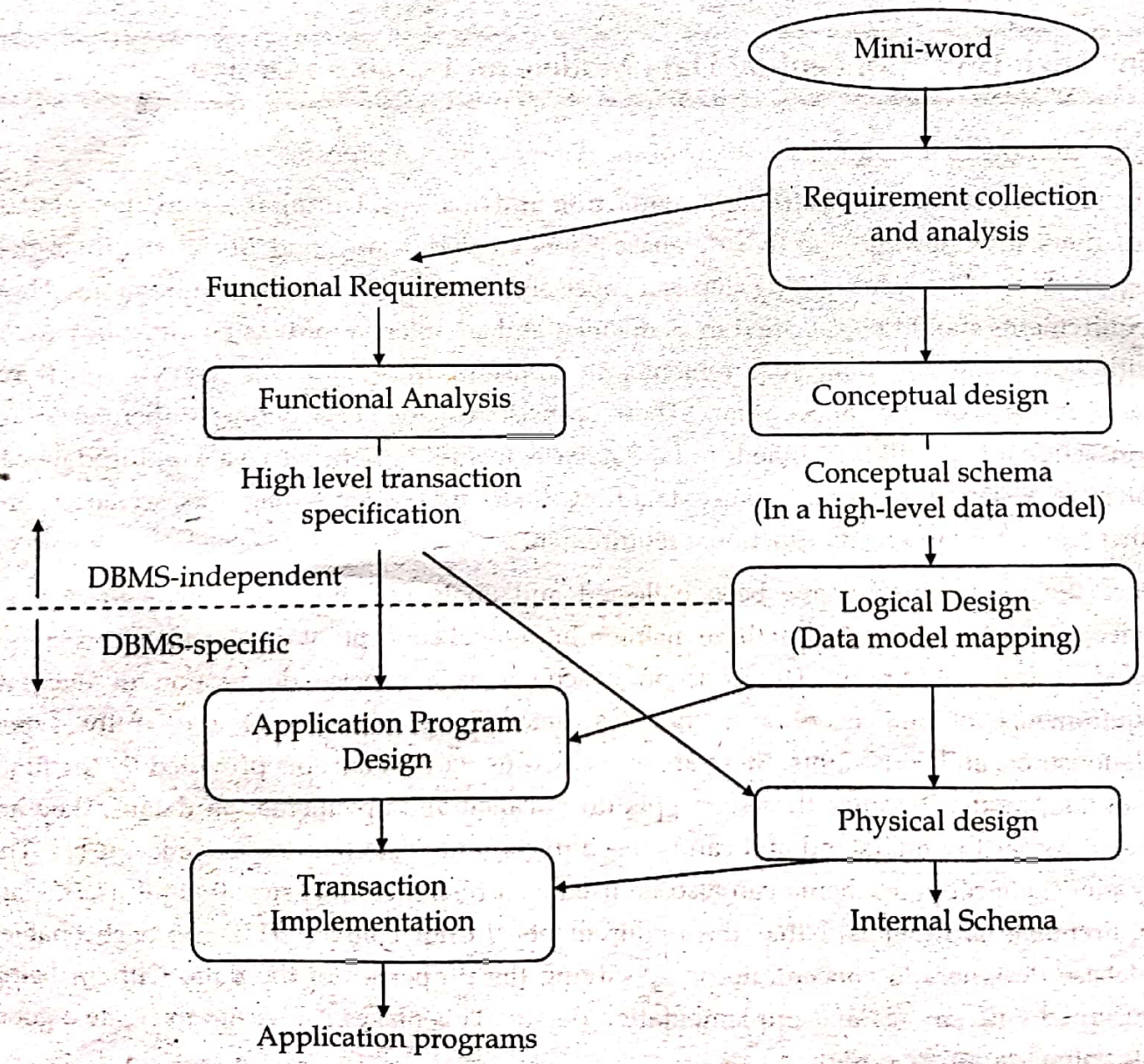


Figure 3.1 Diagram showing the main phases of database design

3.4 Elements of ER Diagram / ER Design Issues

Entity, Entity type and Entity set

Entity

An entity is a real-world thing either living or non-living that is easily recognizable and non-recognizable. It is anything in the enterprise that is to be represented in our database. It may be a physical thing or simply a fact about the enterprise or an event that happens in the real world. An entity can be place, person, object, event or a concept, which stores data in the database. The characteristics of entities are must have an attribute, and a unique key. Every entity is made up of some 'attributes' which represent that entity.

Simply, it is something which has real existence. Like tuple1 contains information about Aarav (id, name and Age) which has existence in real world. So the tuple1 is an entity. So we may say each tuple is an entity.

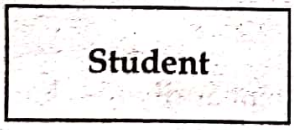
Example: let "Aarav" is a particular member of entity type student.

Entity

Entity type types

It is collection of entities having common attribute. As in Student table each row is an entity and has common attributes. So STUDENT is an entity type which contains entities having attributes id, name and Age. Also each entity type in a database is described by a name and a list of attribute. So we may say a table is an entity type.

Example: Collection of entities with similar properties

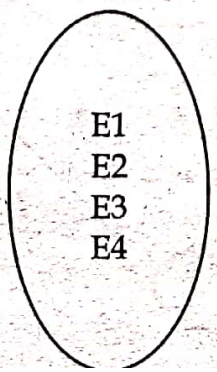


Entity type

Entity set (instance) ⇒

It is same as an entity type, but defined at a particular point in time, such as students enrolled in a class on the first day. Other examples: Customers who purchased last month, cars currently registered in Florida. A related term is instance, in which the specific person or car would be an instance of the entity set.

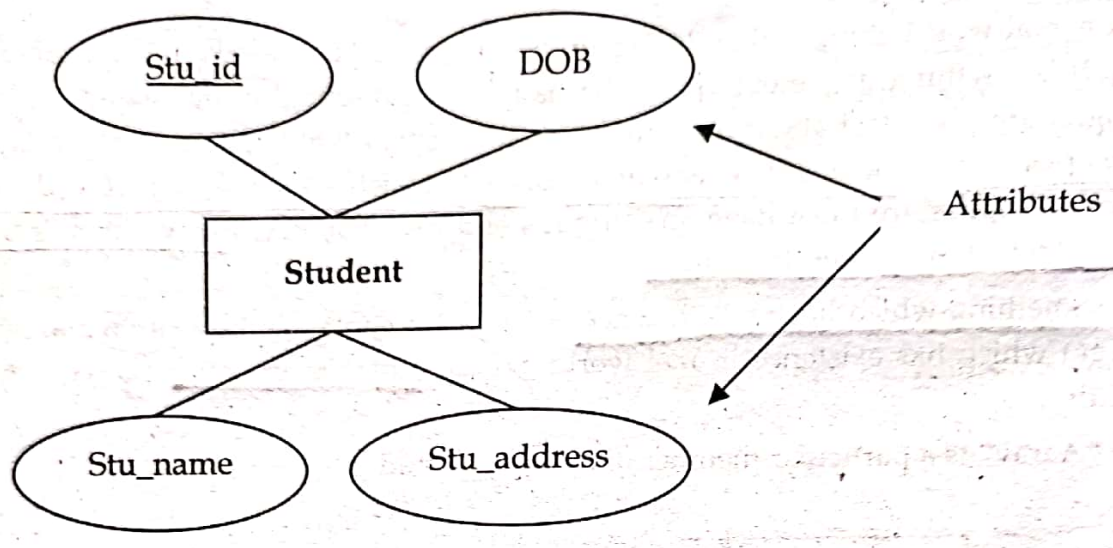
Example: let E1 is an entity having entity type Student and set of all students is called entity set.



Entity set

Attributes

Attributes are the properties which define the entity type. For example, Student_id, student_name, DOB, Age, Address, Mobile_No etc. are the attributes which defines entity type **Student**. In ER diagram, attribute is represented by an oval.



Types of Attributes

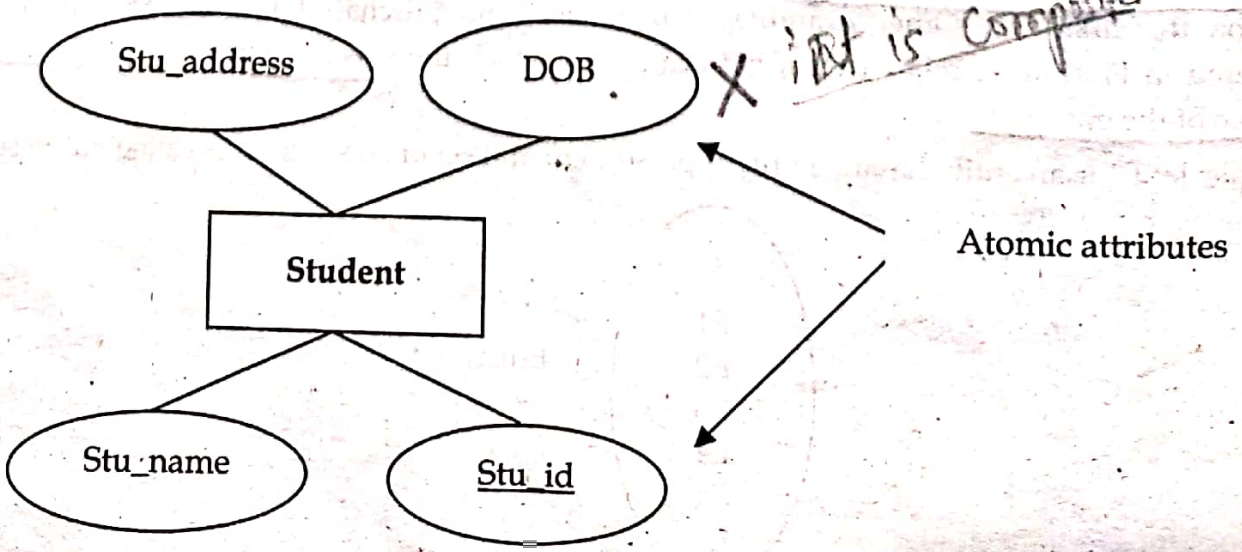
Attributes of an entity type can be further divided into following types

- Atomic vs. composite attributes
- Single valued vs. multi valued attributes
- Stored vs. derived attributes
- NULL value attribute
- Key attribute

Atomic vs. composite attributes

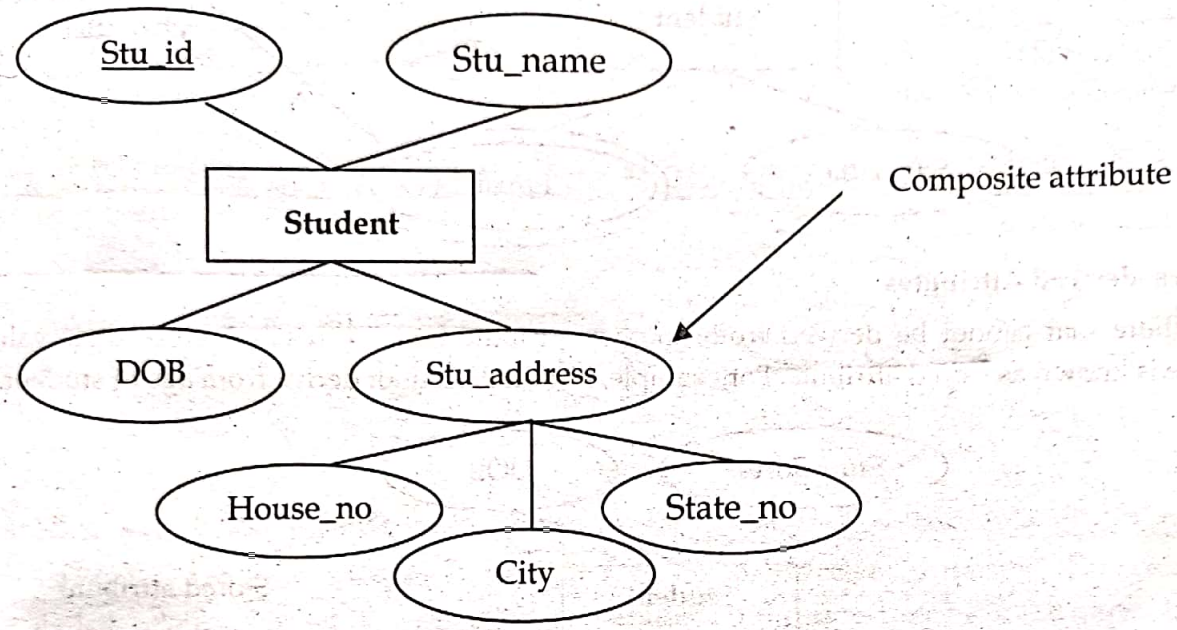
An attribute that cannot be divided into smaller independent attribute is known as atomic attribute.

For example, assume Student is an entity and its attributes are Name, Age, DOB, Address and Phone no. Here the Stu_id, DOBattributes of student (entity) cannot further divide. In this example Stu_id and DOB are atomic attribute.



An attribute that can be divided into smaller independent attribute is known as composite attribute.

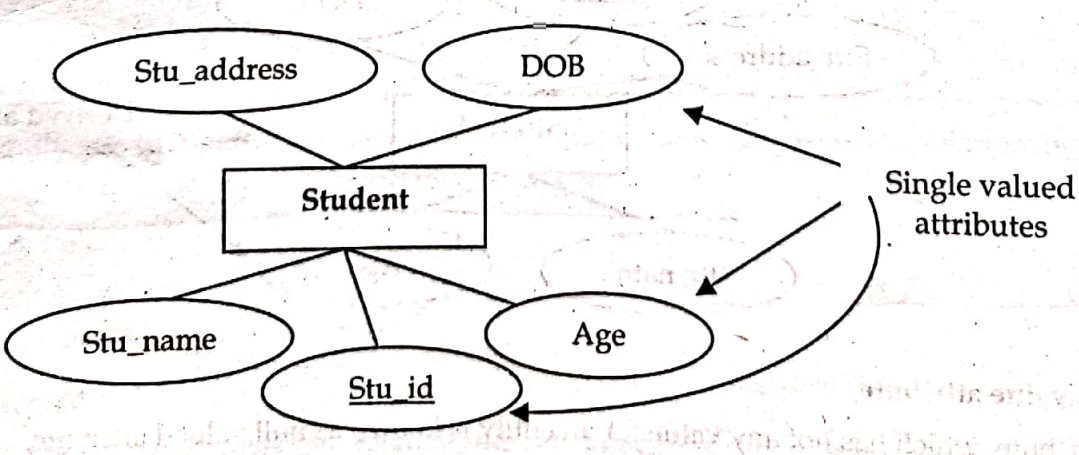
For example, assume Student is an entity and its attributes are Stu_id, Name, DOB, Address and Phone no. Here the address (attribute) of student (entity) can be further divide into House no, city and so on. In this example address is **composite** attribute.



Single valued vs. multi valued attributes

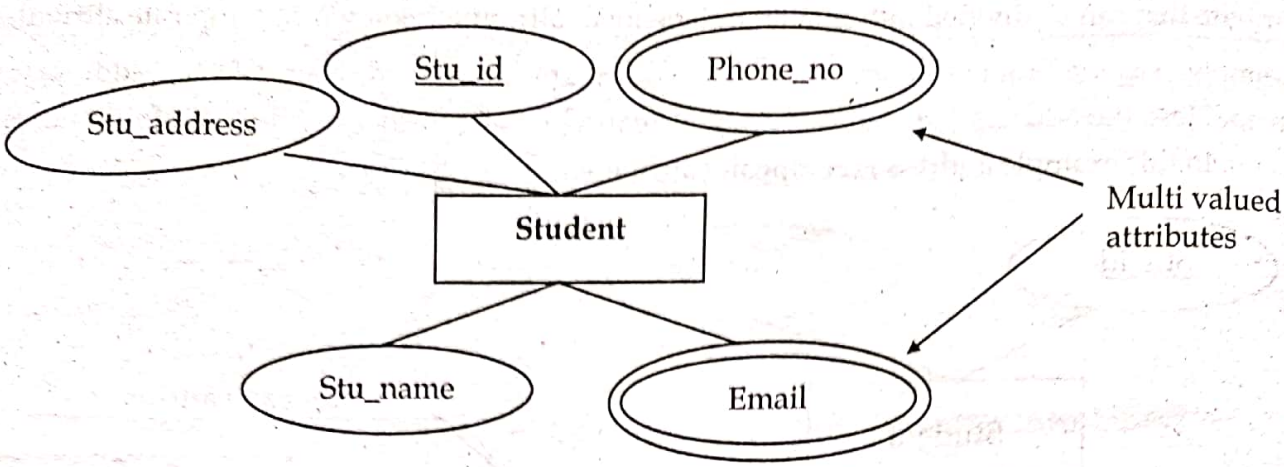
An attribute that has only single value for an entity is known as single valued attribute.

For example, assume Student is an entity and its attributes are Stu_id, Name, DOB, age, Address and Phone no. Here the age (attribute) of student (entity) can have only one value. Here, age is single valued attribute.



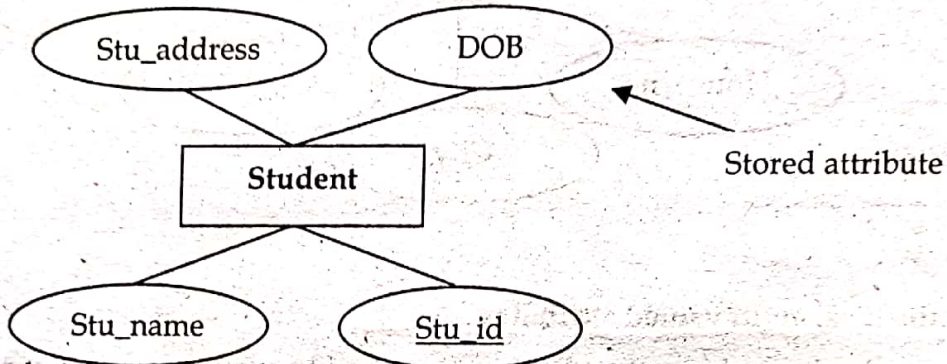
An attribute that can have multiple values for an entity is known as multi valued attribute.

For example, assume Student is an entity and its attributes are Stu_id, Name, Age, Address and Phone no. Here the Phone no (attribute) of student (entity) can have multiple value because a student may have many phone numbers. Here, Phone no is multi valued attribute.

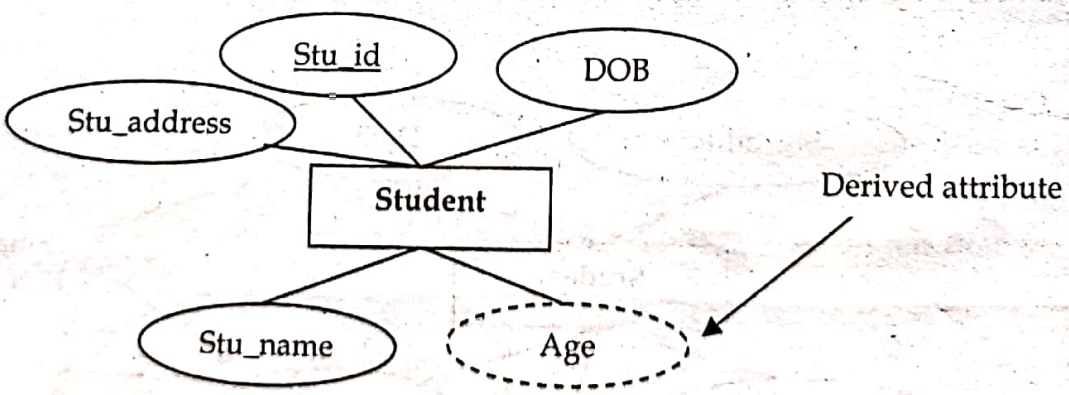


Stored vs. derived Attributes

An attribute that cannot be derived from another attribute and we need to store their value in database is known as stored attribute. For example, birth date cannot derive from age of student.



An attribute that can be derived from another attribute and we do not need to store their value in the database due to dynamic nature is known as derived attribute. It is denoted by doted oval. For example, age cannot derive from birth date of student.



NULL value attribute

An attribute, which has not any value for an entity is known as null valued attribute.

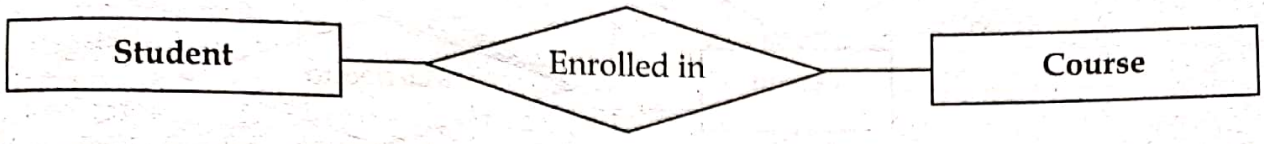
For example, assume Student is an entity and its attributes are Name, Age, Address and Phone no. There may be chance when a student has no phone no. In that case, phone no is called null valued attributes.

Key attribute

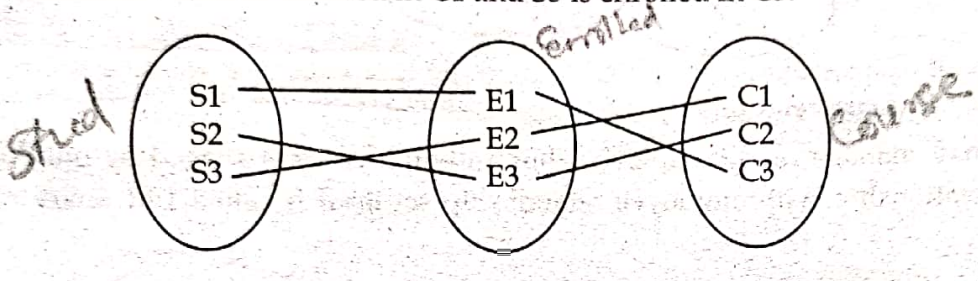
An attribute that has unique value of each entity is known as key attribute. For example, every student has unique roll no. Here roll no is key attribute.

Relationship type and Relationship set

A relationship type represents the association between entity types. For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, relationship type is represented by a diamond and connecting the entities with lines.

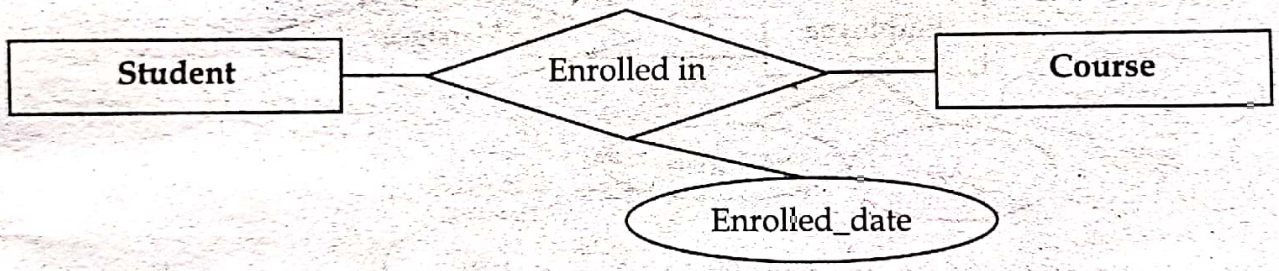


A set of relationships of same type is known as relationship set. The following relationship set depicts S1 is enrolled in C2, S2 is enrolled in C1 and S3 is enrolled in C3.



In another way, we can say that association between two entity sets is called relationship set.

A relationship set may also have attributes called **descriptive attributes**. For example, the Enrolled_in relationship set between entity sets **student** and **course** may have the attribute enrolled-_date.



Degree of a Relationship

Number of entity sets that participate in a relationship set is called degree of the relationship set. On the basis of degree, relationships can be divided as below:

- Unary Relationship
- Binary Relationship
- N-ary Relationship

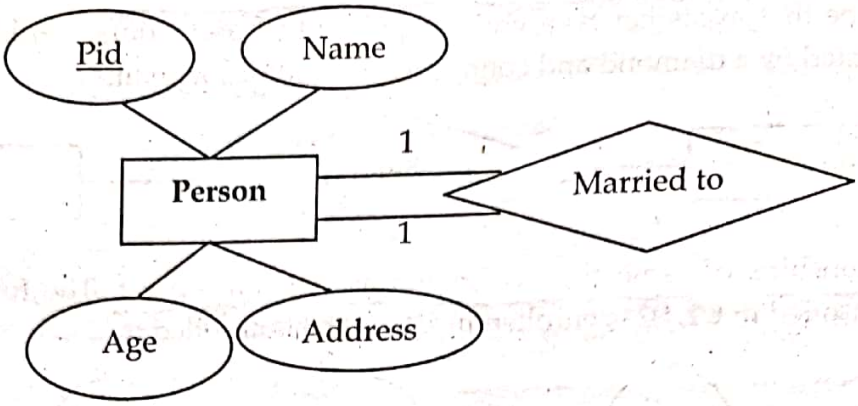
Unary Relationship

If only one entity set participates in a relation, the relationship is called as unary relationship. Here same entity set participates in relationship twice with different roles. Role names are specified above the link joining entity set and relationship set. This type of relationship set is sometimes called a recursive relationship set. There are three types of unary relationships:

- 1:1 unary relationship
- 1:M unary relationship
- M:N unary relationship

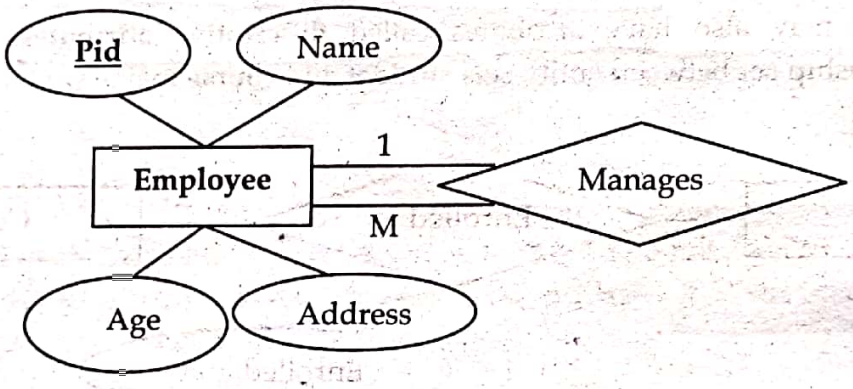
One to one (1:1) unary relationship

In the example below, one **person** is married to only one person.



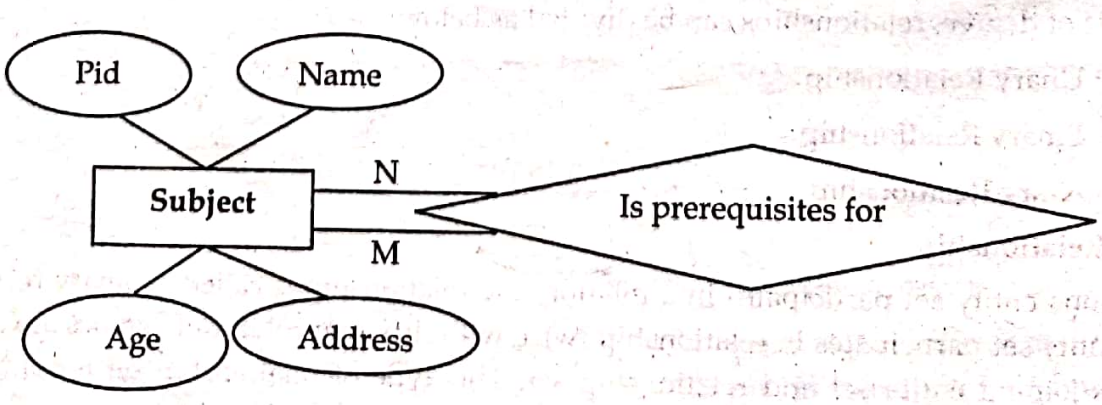
One to many (1: M) unary relationship

An employee may manage many employees but an employee is managed by only one employee. This types of relationship with employee relationship set itself is called 1:M unary relationship as shown in below;



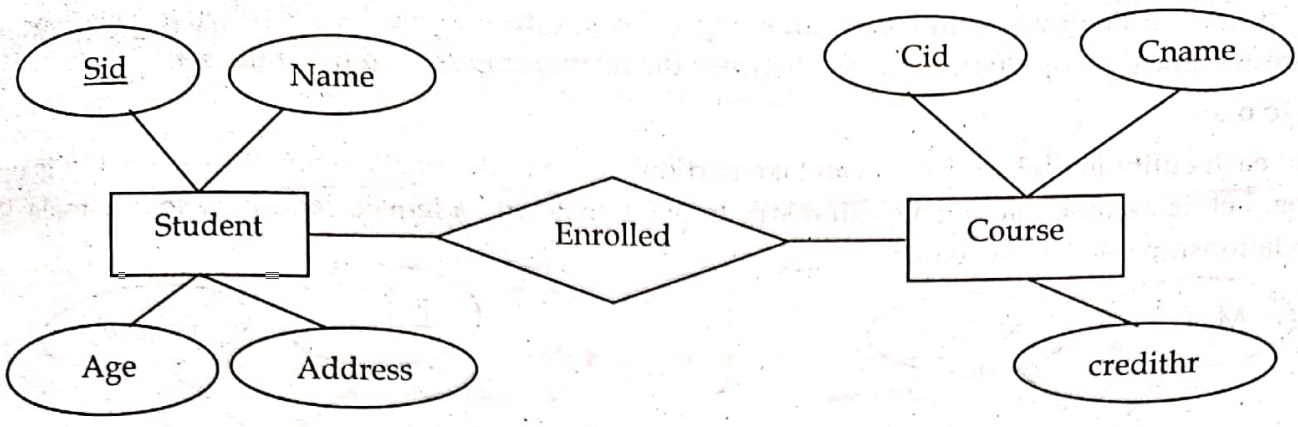
Many to many (M: N) unary relationship

A subject may have many other subjects as prerequisites and each subject may be a prerequisite to many other subjects.



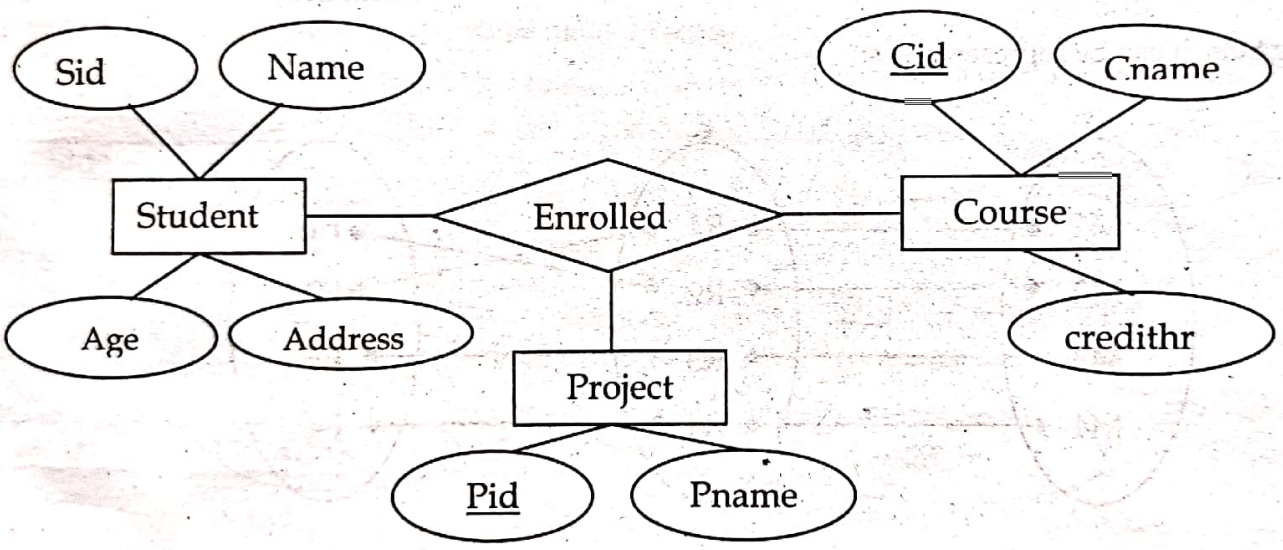
Binary relationship

When there are two entities set participating in a relation, the relationship is called as binary relationship. For example, Student is enrolled in Course. This is the most common type of relationship in database systems.



N-ary relationship

When there are n entities set participating in a relation, the relationship is called as n-ary relationship. If n=1 then it is called unary relationship, if n=2 then it is called binary relationship. Generally in N-ary relationship there are more than two entities participating with a single relationship i.e. n > 2.



3.5 Constraints on ER Model/Structural Constraint in ER

Relationship sets in ER model usually have certain constraints that limit the possible combinations of entities that may involve in the corresponding relationship set. Database content must confirm these constraints. The most important structural constraints in ER are listed below:

- Mapping cardinalities and
- Participation constraints

Mapping Cardinality Constraints

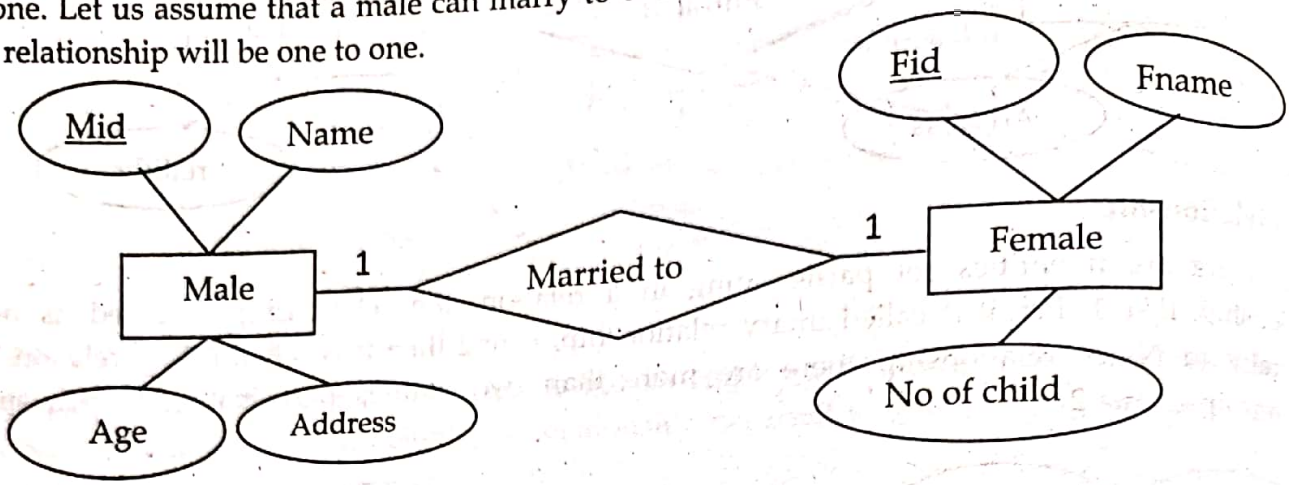
The number of times an entity of an entity set participates in a relationship set is known as cardinality. Cardinality can be of different types:

- One-to- One
- One-to- Many
- Many-to-One
- Many-to-Many

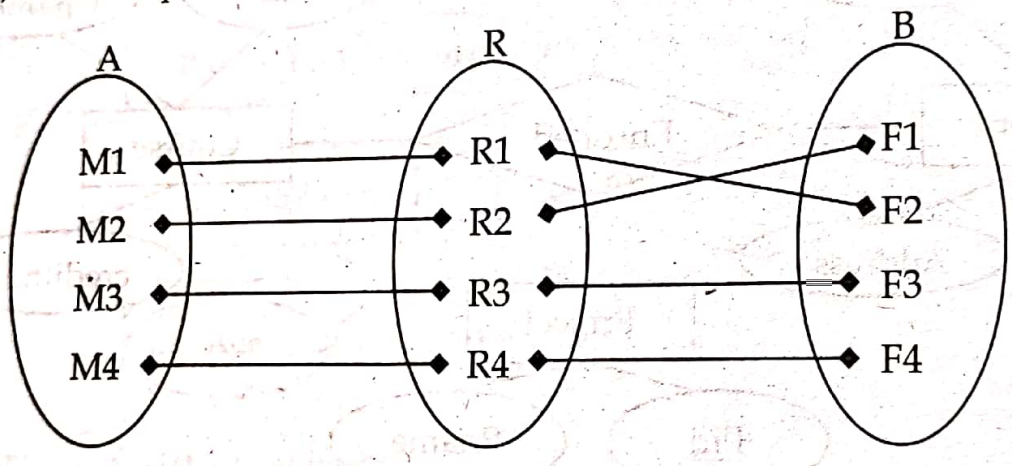
We express cardinality constraints by drawing either a directed line (→), signifying "one," or an undirected line (—), signifying "many," between the relationship set and the entity set.

One to one

When each entity in each entity set can take part only once in the relationship, the cardinality is one to one. Let us assume that a male can marry to one female and a female can marry to one male. So the relationship will be one to one.

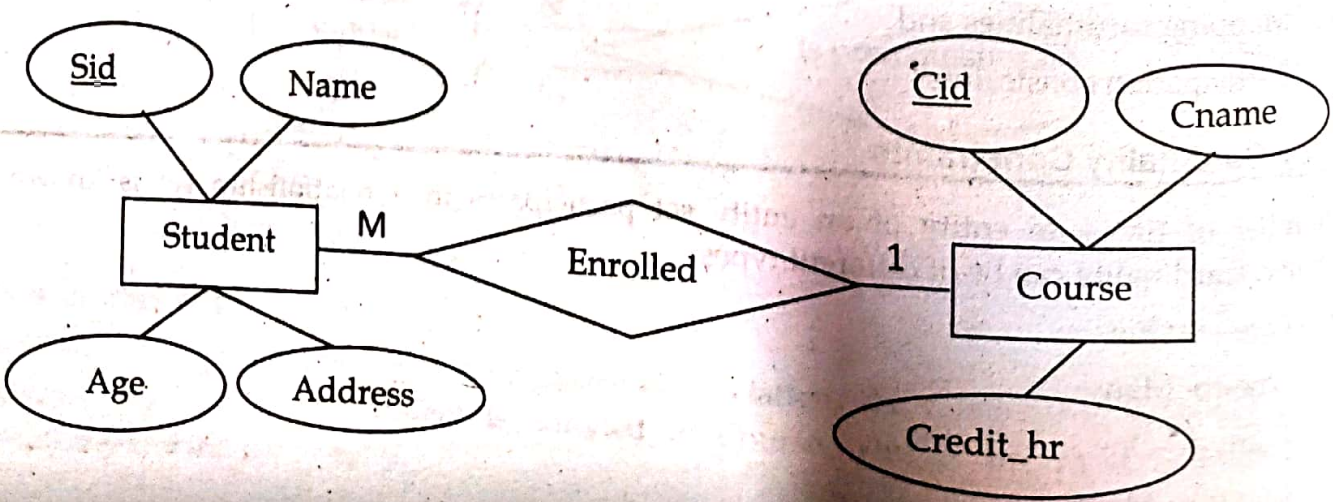


Using Sets, it can be represented as:

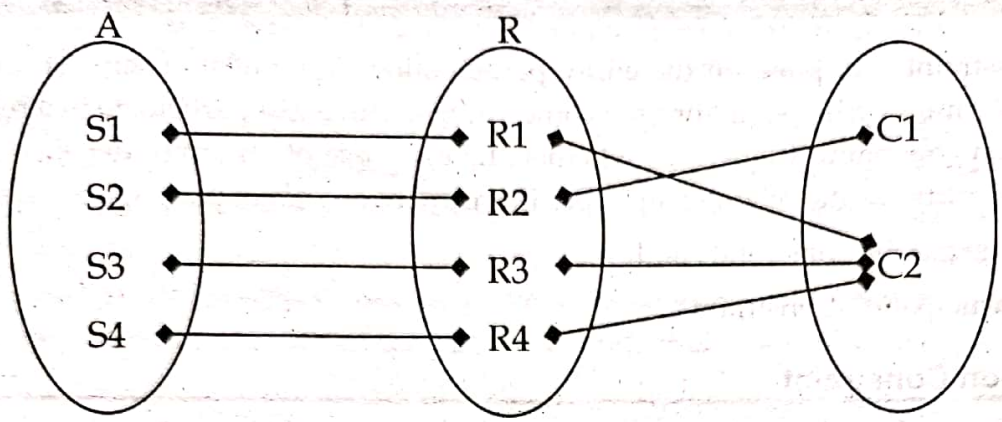


One to many or many to one

When entities in one entity set can take part only once in the relationship set and entities in other entity set can take part more than once in the relationship set, cardinality is many to one. Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student there will be only one course.

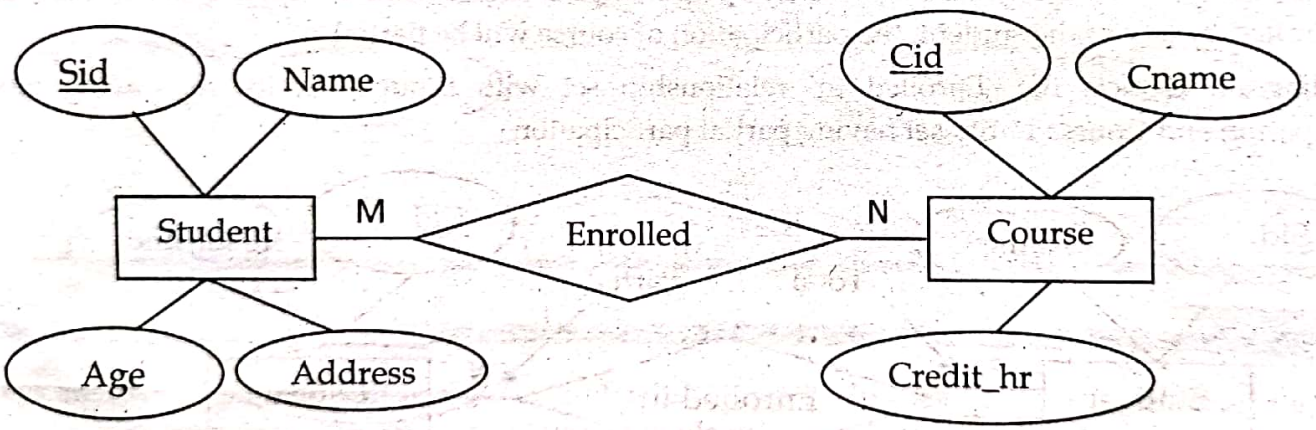


Using Sets, it can be represented as:

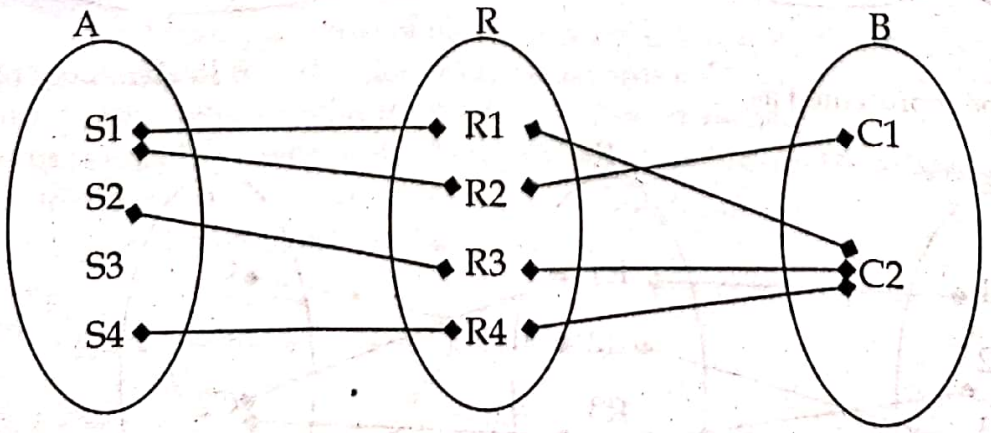


Many to many

When entities in all entity sets can take part more than once in the relationship cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.



Using Sets, it can be represented as:



In this example, student S1 is enrolled in C1 and C3 and Course C2 is enrolled by S1, S2 and S4. So it is many to many relationships.

3.6 Participation Constraints

Participation Constraint is applied on the entity participating in the relationship set. Constraint on ER model that determines whether all or only some entity occurrences participate in a relationship is called participation constraint. It specifies whether the existence of an entity depends on its being related to another entity via the relationship type. There are two types of participation constraints:

- Total Participation Constraints and
- Partial Participation Constraints.

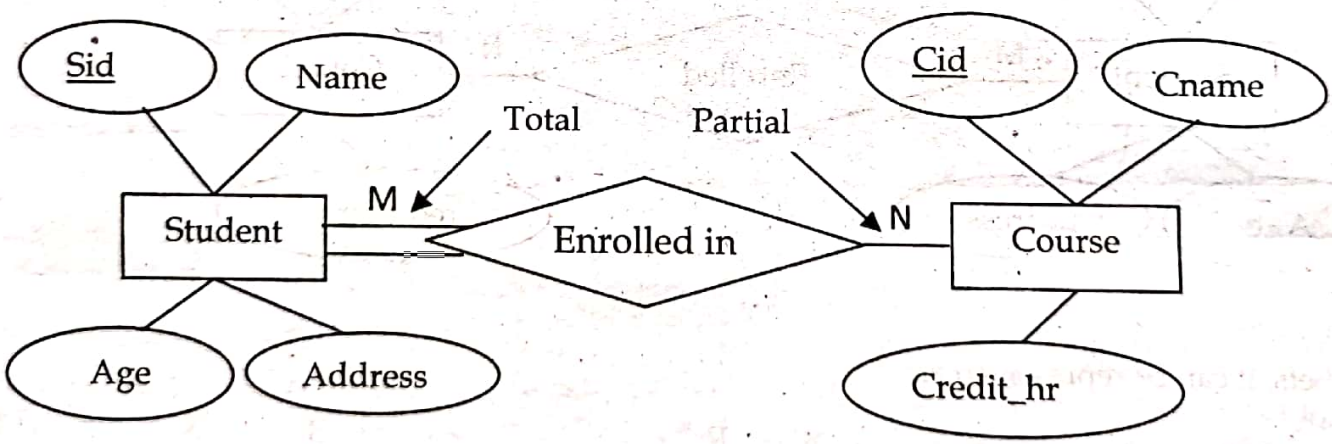
Total participation Constraint

Here each entity in the entity set must participate in the relationship. If each student must enroll in a course, the participation of student will be total. Total participation is shown by double line in ER diagram.

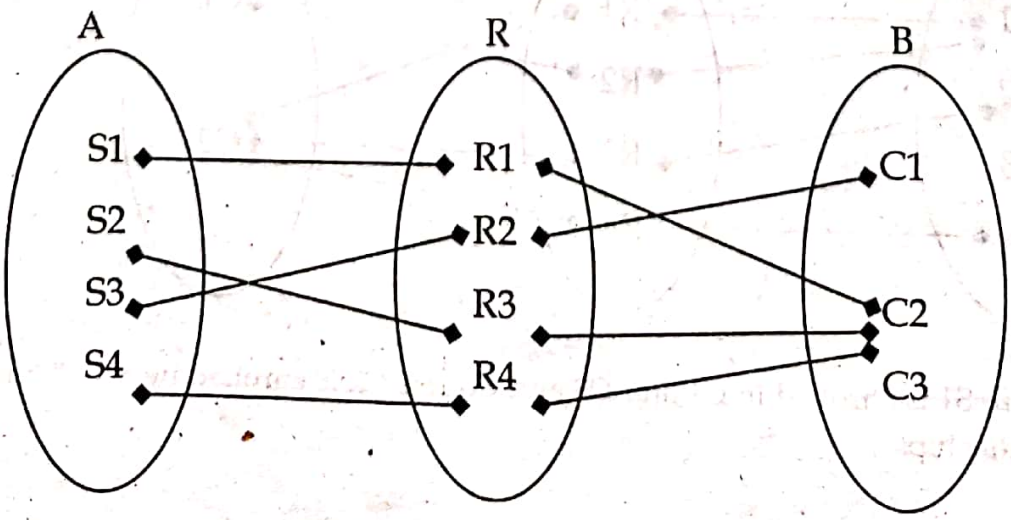
Partial Participation Constraint

Here the entity in the entity set may or may not participate in the relationship. If some courses are not enrolled by any of the student, the participation of course will be partial.

The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation.

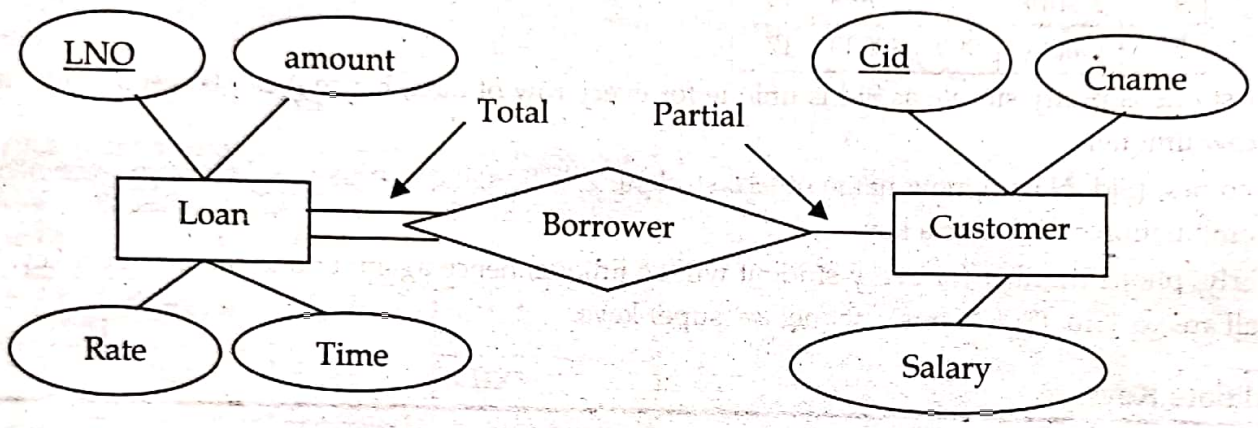


Using set, it can be represented as,



Every student in **Student** Entity set is participating in relationship but there exists a course C3 which is not taking part in the relationship. Thus participation of student relation with relationship 'Enrolled in' is called total and participation of course relation with given relationship is called partial.

Similarly, let's take an another example, consider **Customer** and **Loan** entity sets in a banking system, and a relationship set **borrower** between them indicates that only some of the customers have Loan but every Loan should be associated with some customer. Therefore there is total participation of entity set Loan in the relationship set borrower but participation of entity set customer is partial in relationship set borrower. Here, Loan entity set cannot exist without Customer entity set but existence of Customer entity set is independent of Loan entity set.



3.7 Keys in DBMS

Keys are very important part of Relational database model. They are used to establish and identify relationships between tables and also to uniquely identify any record or row of data inside a table. A Key can be a single attribute or a group of attributes, where the combination may act as a key.

Why we need a Key?

Here, are reasons for using Keys in the DBMS system.

- Keys help us to identify any row of data in a table. In a real-world application, a table could contain thousands of records. Moreover, the records could be duplicated. Keys ensure that we can uniquely identify a table record despite these challenges.
- Allows us to establish a relationship between and identify the relation between tables
- Help us to enforce identity and integrity in the relationship.

There are different types of keys which are listed below:

- Super key
- Candidate key
- Primary key
- Composite key
- Foreign key
- Alternate Key
- Compound Key
- Surrogate Key

Super Key

A super key of an entity set is a set of one or more attributes whose values uniquely determine each entity in the entity set. If K is a key attribute and any superset of K is also super key. A Super key may have additional attributes that are not needed for unique identification.

Let's try to understand about all the keys using a simple example

Sid	Name	Phone	age
1	Aarav	9876723452	17
2	Aarav	9991165674	19
3	Binod	7898756543	18
4	Ashna	8987867898	19
5	Geeta	9990080080	17

The first one is pretty simple as Sid is unique for every row of data, hence it can be used to identity each row uniquely.

Next comes, (Sid, Name), now name of two students can be same, but their Sid can't be same hence this combination can also be a key.

Similarly, phone number for every student will be unique, hence again; phone can also be a key. So they all are i.e. {Sid, (Sid, Name), phone} are super keys.

Candidate Keys

Candidate keys are defined as the minimal set of fields which can uniquely identify each record in a table. It is an attribute or a set of attributes that can act as a Primary Key for a table to uniquely identify each record in that table. There can be more than one candidate key. Simply, a candidate key of an entity set is a minimal super key.

In our example, **Sid** and **phone** both are candidate keys for table Student.

Features of candidates key

- A candidate key is a minimal form of super key.
- There is no possible subset of candidate key that acts as a key attribute.
- A candidate key can never be NULL or empty. And its value should be unique.
- There can be more than one candidate keys for a table.
- A candidate key can be a combination of more than one column (attributes).

Primary key

Primary key is a candidate key that is most appropriate to become the main key for any table. It is a key that can uniquely identify each record in a table.

A primary key is a candidate key that is chosen by the database designer as the principle means of uniquely identifying entities within an entity set. There may exist several candidate keys, one of the candidate keys is selected to be the primary key. Primary key must satisfy following two characteristics:

- It cannot be null
- It cannot be duplicate

For the table Student if database designer choose the candidate key **Sid** as key attribute then it acts as the primary key for their purpose.

Composite Key

Key that consists of two or more attributes that uniquely identify any record in a table is called Composite key. But the attributes which together form the Composite key are not a key independently or individually. Simply, if a primary key contains more than one attribute then it is called composite key. For example, if database designer choose Sid as primary key then it not composite key but if database designer choose {Sid, phone} as primary key then it is called composite key.

Foreign Key

Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables. A foreign key (FK) is an attribute or combination of attributes that is used to establish and enforce relationship between two relations (table).For example, if a student enrolls in course then course id (primary key of relation course) can be used as foreign key in student relation.

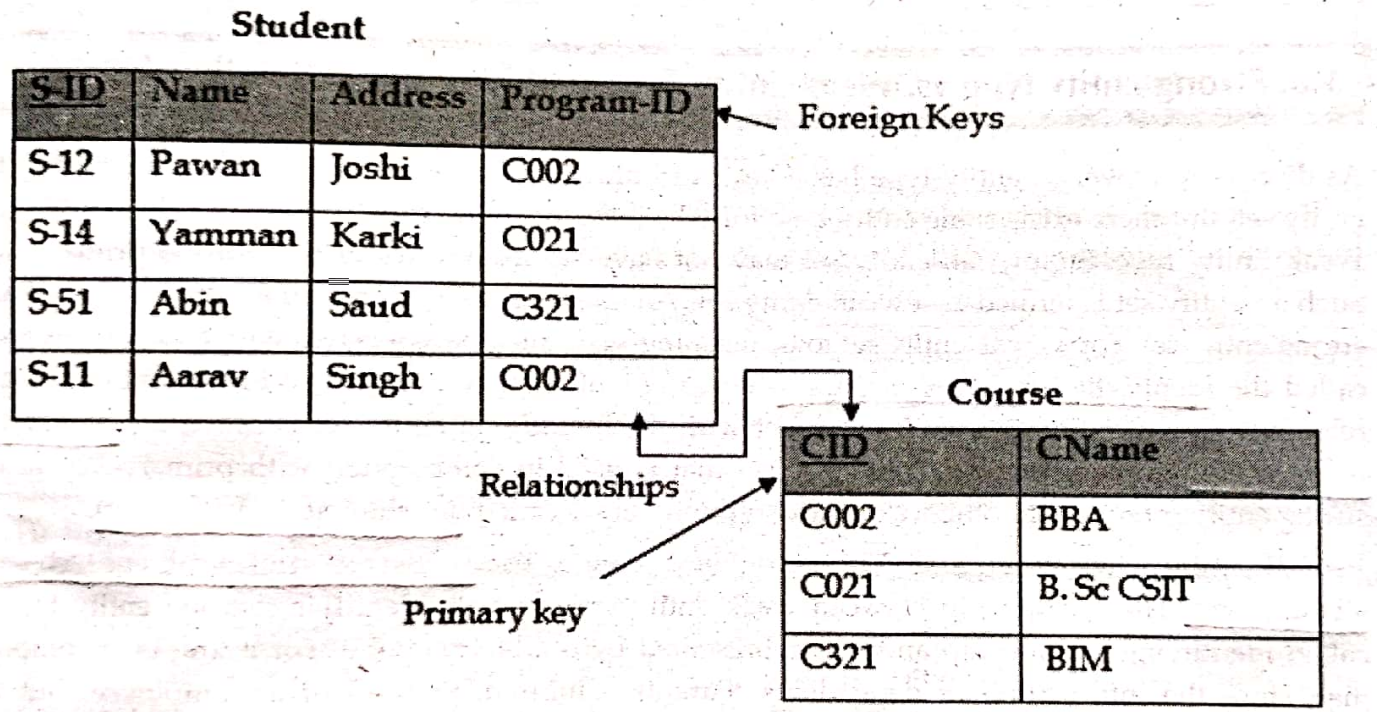


Figure 3.2 Primary key and foreign key

Alternate Key/Secondary key

Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternate or secondary keys.

Compound Key

Like other keys Compound key is also used to uniquely recognize a record in relation. This can be an attribute or a set of attributes, but the attributes in relation cannot be use as independent keys. If we use them individually, we will not get any unique record.

Surrogate Key

An artificial key which aims to uniquely identify each record is called a surrogate key. These kinds of key are unique because they are created when you don't have any natural primary key. They do not lend any meaning to the data in the table. Surrogate key is usually an integer.

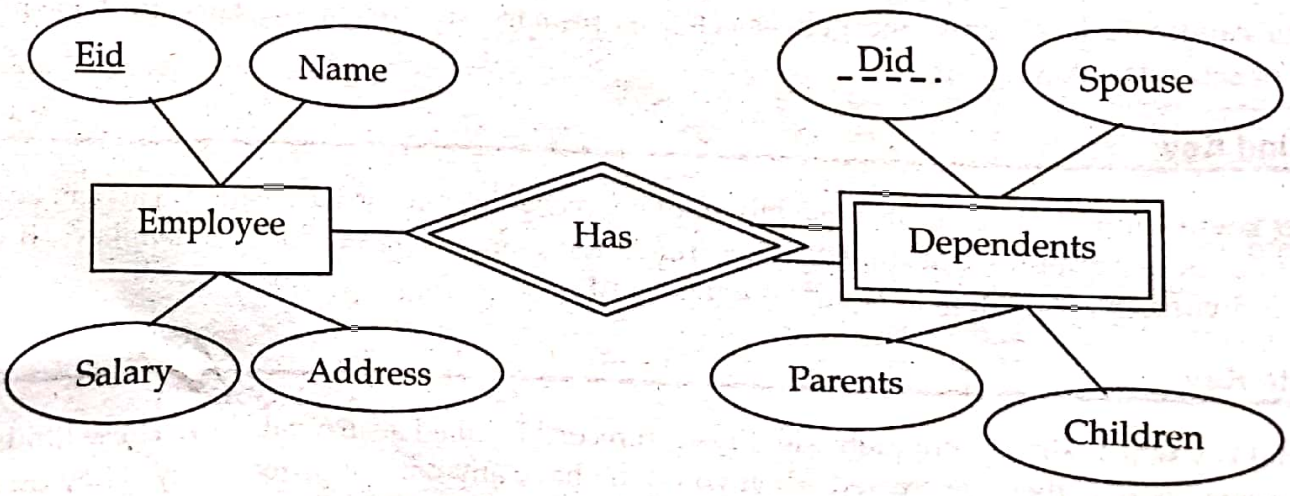
Difference between Primary key & foreign key

Primary Key	Foreign Key
It helps us to uniquely identify a record in the table.	It is a field in the table that is the primary key of another table.
Primary key never accepts NULL values.	A foreign key may accept multiple NULL values.
Primary key is a clustered index and data in the DBMS table are physically organized in the sequence of the clustered index.	A foreign key cannot automatically create an index, clustered or non-clustered. However, you can manually create an index on the foreign key.
We can have the single primary key in a table.	We can have multiple foreign keys in a table.

3.8. Strong Entity type vs. Weak Entity Type and Identifying Relationship

As discussed above, an entity type has a key attribute which uniquely identifies each entity in the entity set. But there exists some entity type for which key attribute can't be defined. These are called Weak Entity type. Simply, an entity set may not have sufficient attributes to form a primary key. Such an entity set is termed as a weak entity set. An entity set that has a primary key is termed as a strong entity set. For a weak entity set to be meaningful, it must be associated with another entity set, called the identifying or owner entity set, using one of the key attribute of owner entity set. The relationship associating the weak entity set with the identifying entity set is called the identifying relationship. An attribute of weak entity set that is used in combination with primary key of the strong entity set to identify the weak entity set uniquely is called discriminator (partial key).

A weak entity type is represented by a double rectangle. The participation of weak entity type is always total. The relationship between weak entity type and its identifying strong entity type is called identifying relationship and it is represented by double diamond. **For example,** a company may store the information of dependents (Parents, Children, Spouse) of an Employee. But the dependents don't have existence without the employee. So Dependent will be weak entity type and Employee will be Identifying Entity type for Dependent.



3.9 Differentiate between weak entity and Strong Entity

Strong Entity Set	Weak Entity Set
Strong entity set always has a primary key.	It does not have enough attributes to build a primary key.
It is represented by a rectangle symbol.	It is represented by a double rectangle symbol.
It contains a Primary key represented by the underline symbol.	It contains a Partial Key which is represented by a dashed underline symbol.
The member of a strong entity set is called as dominant entity set.	The member of a weak entity set called as a subordinate entity set.
Primary Key is one of its attributes which helps to identify its member.	In a weak entity set, it is a combination of primary key and partial key of the strong entity set.
In the ER diagram the relationship between two strong entities set shown by using a diamond symbol.	The relationship between one strong and a weak entity set shown by using the double diamond symbol.
The connecting line of the strong entity set with the relationship is single.	The line connecting the weak entity set for identifying relationship is double.

3.10 Roles in E-R Diagrams

The function that an entity plays in a relationship is called its role. Roles are normally explicit and not specified. They are useful when the meaning of a relationship set needs clarification. For example, the entity sets of a relationship may not be distinct. The relationship works-for might be ordered pairs of employees (first is manager, second is worker).

In the E-R diagram, this can be shown by labeling the lines connecting entities to relationships.

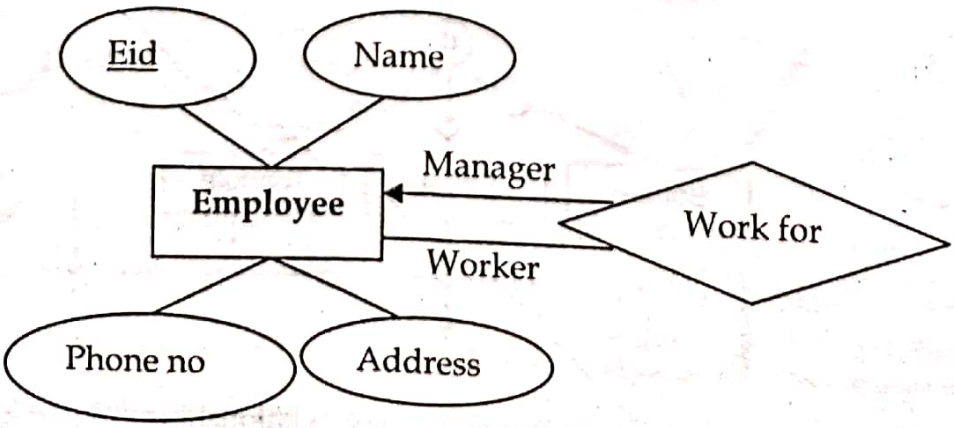


Figure 3.3 E-R diagram with role indicators

3.11 Extended E-R Model (EER Model)

As the complexity of data increased in the late 1980s, it became more and more difficult to use the traditional ER Model for database modeling. Hence some improvements or enhancements were made to the existing ER Model to make it able to handle the complex applications better. Hence, as part of the Enhanced ER Model, along with other improvements, three new concepts were added to the existing ER Model, they were:

- Subclasses and Super classes
- Specialization and Generalization
- Category or union type
- Aggregation

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.

Subclasses and super classes

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, and Triangle.

Sub class is a group of entities with unique attributes. Sub class inherits properties and attributes from its super class. For example: Square, Circle and Triangle are the sub class of Shape super class.

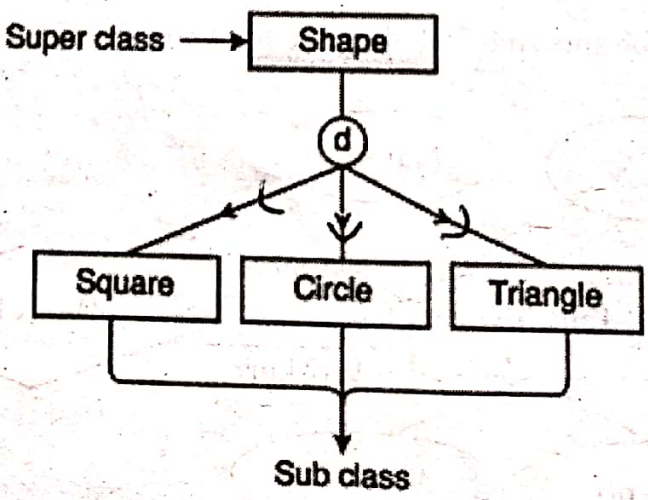


Figure 3.4 subclass super class relationship

Specialization and Generalization

Generalization

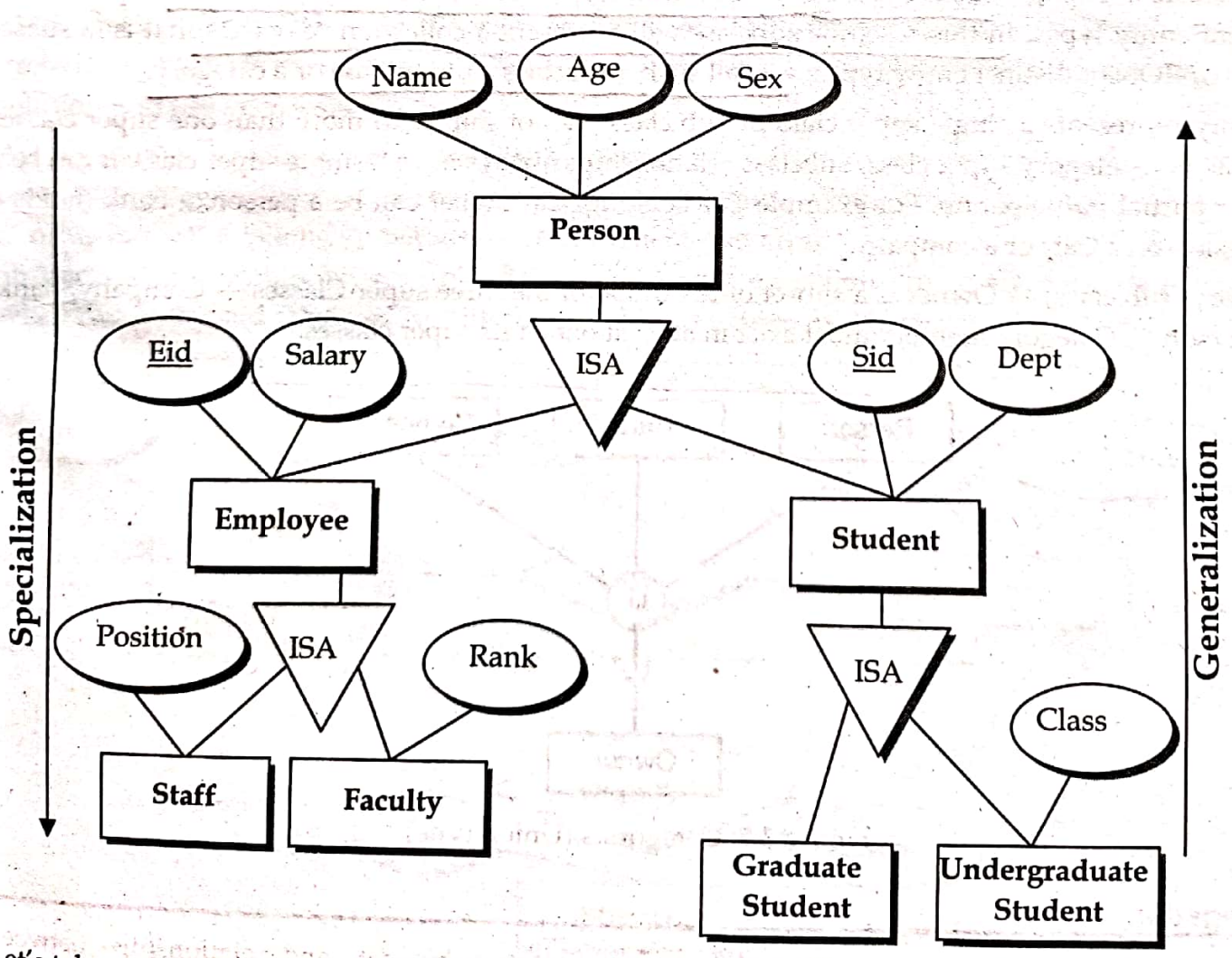
Generalization is a bottom-up approach in which two lower level entities combine to form a higher level entity. In generalization, the higher level entity can also combine with other lower level entities to make further higher level entity.

It's more like Super class and Subclass system, but the only difference is the approach, which is bottom up. Hence, entities are combined to form a more generalized entity, in other words, sub-classes are combined to form a super-class.

Specialization

Specialization is opposite to Generalization. It is a top-down approach in which one higher level entity can be broken down into two lower level entity. In specialization, a higher level entity may not have any lower-level entity sets, it's possible. 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 super class/subclass relationship.

Let's take an example



Let's take separate examples of generalization and specializations as below;

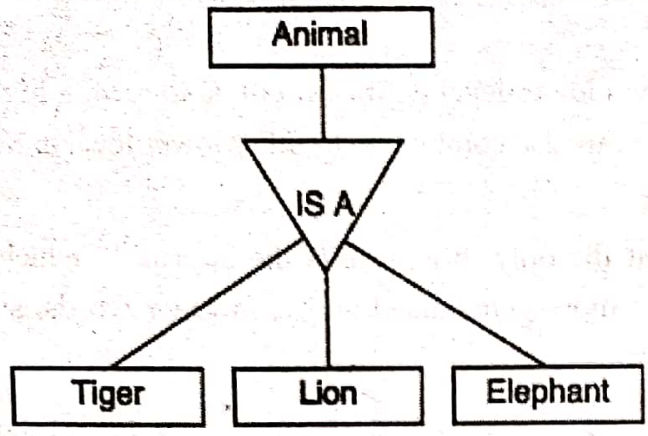


Figure 3.5 Generalization

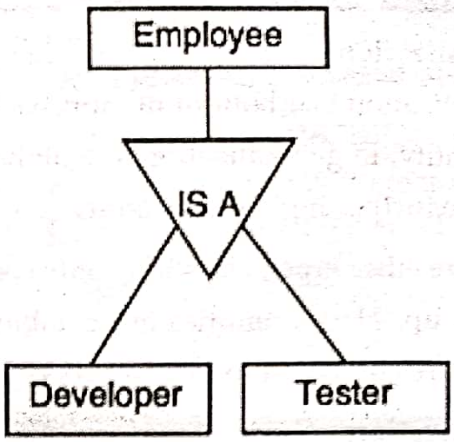


Figure 3.6 Specialization

In the above first example, Tiger, Lion, Elephant can all be generalized as Animals. Also in the above second example, Employee can be specialized as Developer or Tester, based on what role they play in an Organization.

Category or Union

It is possible that single super class/subclass relationship has more than one super-class representing different entity types. In this case, the subclass will represent a collection of objects that is (a subset of) the UNION of distinct entity types; we call such a subclass a union type or a category.

Category represents a single super class or sub class relationship with more than one super classes whereas non-category super class/subclass relationships always have a single 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 its super classes.

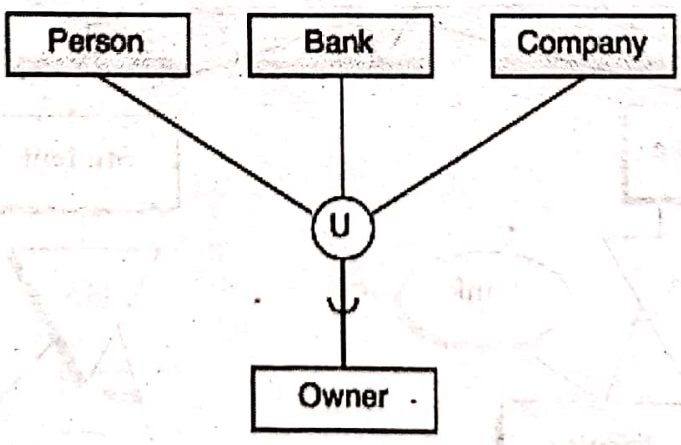
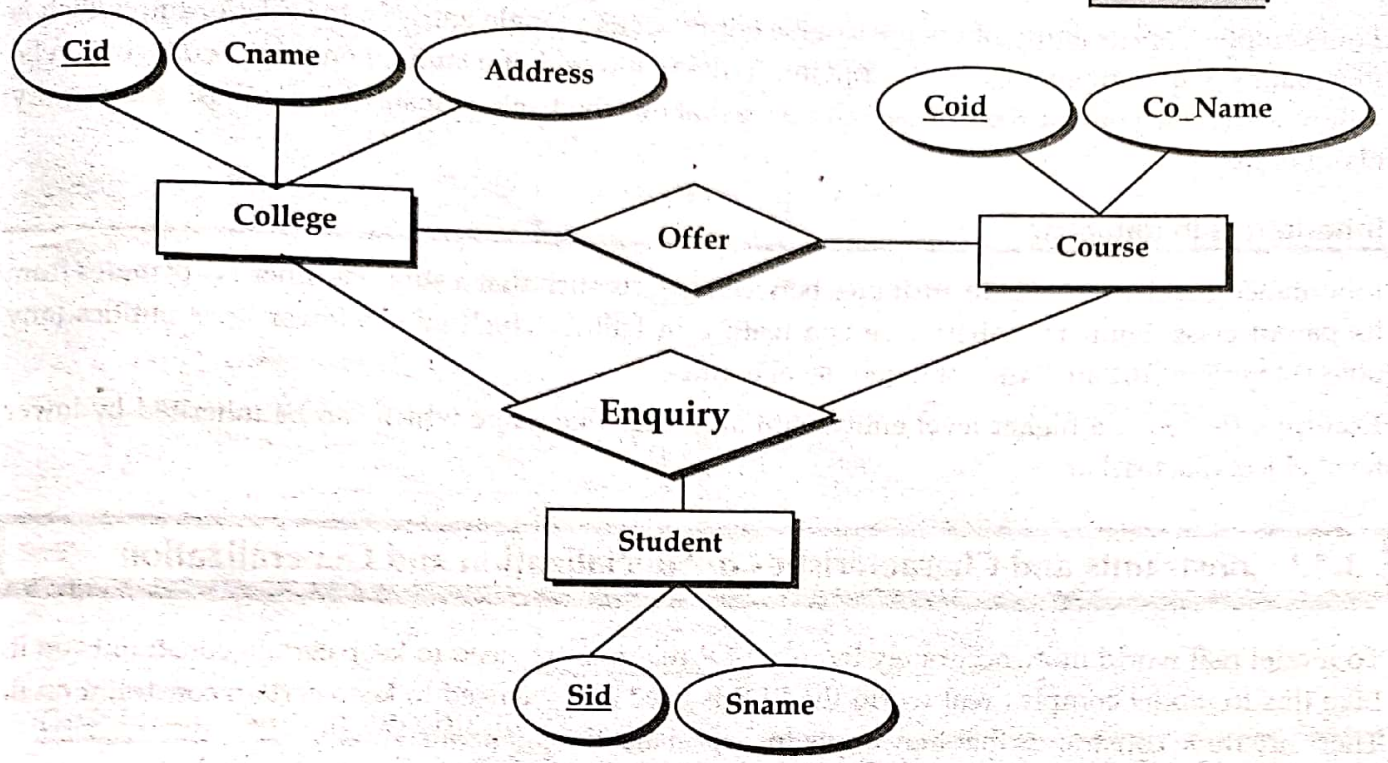


Figure 3.7: Categories (Union type)

Aggregation

The E-R model cannot express relationships among relationship sets and relationships between relationship sets and entity sets. Consider a DB with information about Colleges that offer courses and students enquiry colleges and courses offer by college. One alternative in this case is to define ternary relationship between {College, course, and Student} as in figure below.



Main problem with above ER diagram is that it has redundant relationships. Every {Employee, Project} combination that appears in uses relationship set also appears in works relationship set. The solution is to use aggregation.

Aggregation is a process that represents a relationship between a whole object and its component parts. Using aggregation we can express relationship among relationships. Aggregation shows 'has-a' or 'is-part-of' relationship between entities where one represents the 'whole' and other 'part'. It abstracts a relationship between objects and viewing the relationship as an object. It is a process when two entity and relationship with these entities is treated as a higher level single entity set.

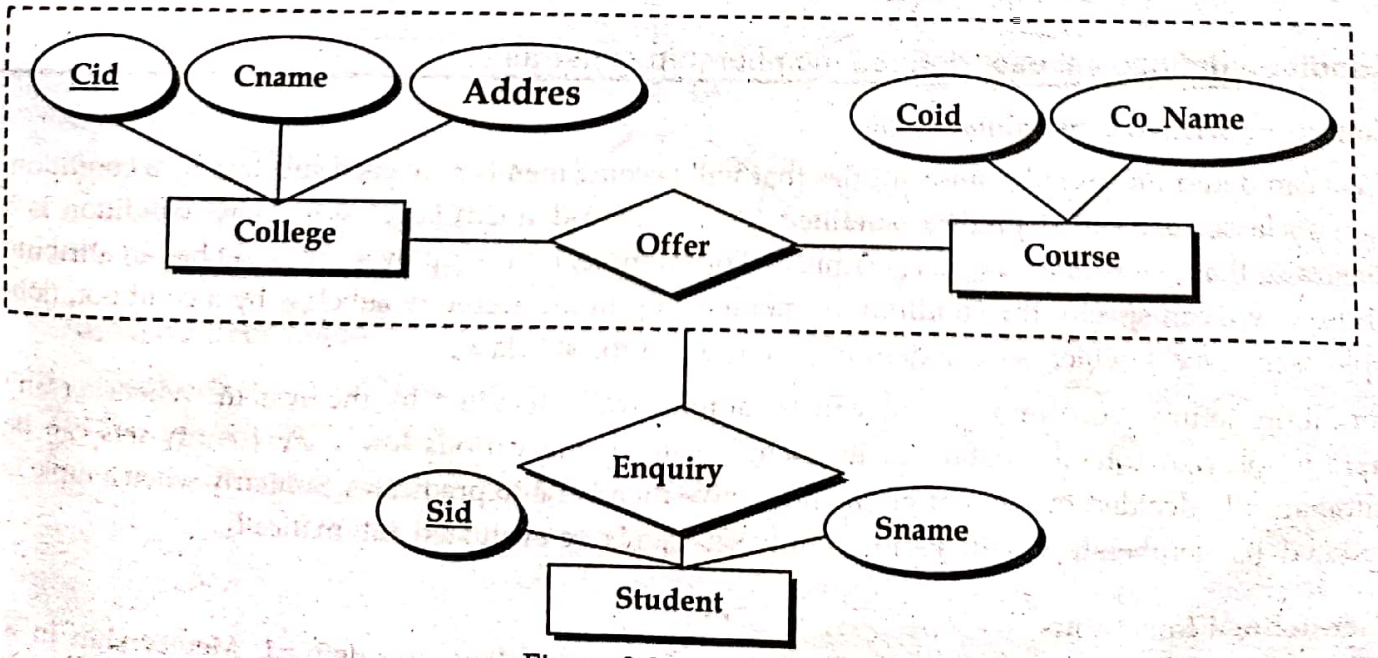


Figure 3.8 Aggregation

In the above example, the relation between College and Course is acting as an Entity in Relation with Student.

For example: College entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity student. In the real world, if a student enquiry a college then he will never enquiry about the Course only or just about the College instead he will ask the enquiry about both.

Inheritance in database

Inheritance enables us to share attributes between objects such that a subclass inherits attributes from its parent class. Similarly Inheritance is a feature in DBMS which allows lower level entities (any object) to inherit the attributes of higher level entities.

Example: Person is a higher level entity with attributes like name which can be inherited by lower level object like teacher.

3.12 Constraints and Characteristics of Specialization and Generalization

To model real world more accurately by using ER diagram we need to keep certain constraints on it. Like this to model complex real world the EER is used and we need to keep certain constraint on it. There are three constraints that may apply to a specialization/generalization:

- **Membership constraints** (predicate defined) (attribute defined)
  - ✓ Condition defined membership constraint
  - ✓ User defined membership constraint
- **Disjoint constraints**
  - ✓ Disjoint constraint
  - ✓ Overlapping constraint
- **Completeness constraints**
  - ✓ Total completeness constraint
  - ✓ Partial completeness constraint

Condition defined vs. user defined membership constraint

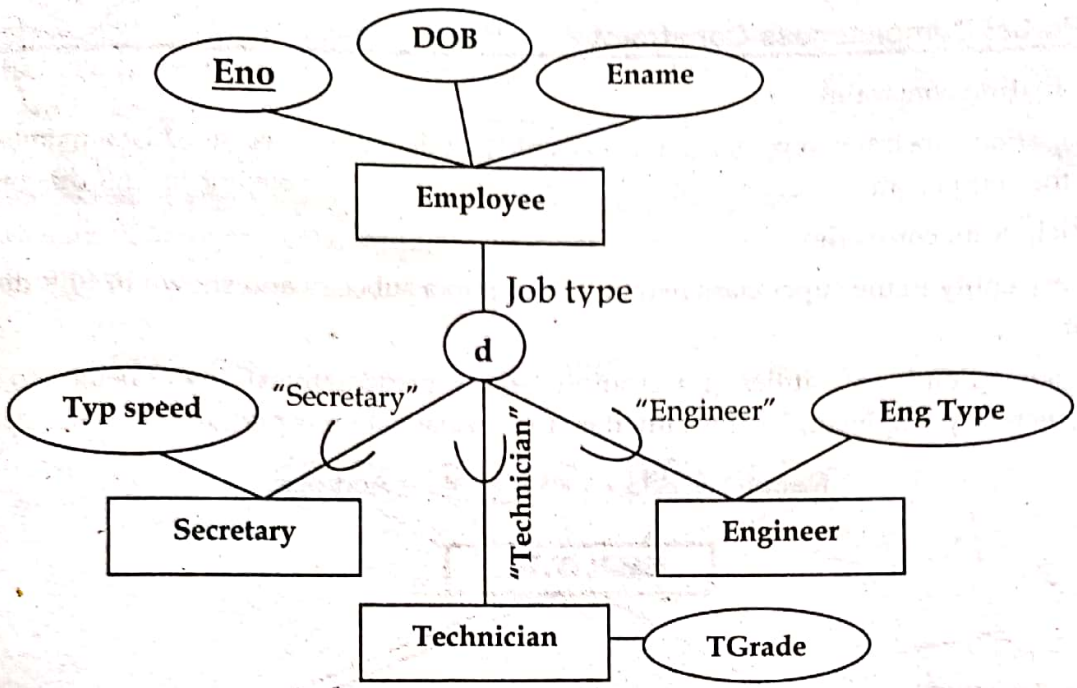
Condition defined constraints

If we can determine exactly those entities that will become members of each subclass by a condition, the subclasses are called predicate-defined (or condition-defined) subclasses. Here, condition is a constraint that determines subclass members. For example, if the employee entity set has an attribute job-type, we can specify the condition of membership in the secretary subclass by a condition (job-type="secretary"), which we call defining predicate of the subclass.

Condition-defined constraints alone can be automatically handled by the system. Whenever any tuple is inserted into the database, its membership in the various lower level entity-sets can be automatically decided by evaluating the respective membership predicates. Similarly when a tuple is updated, its membership in the various entity sets can be re-evaluated automatically.

User-defined constraints

If no condition determines membership, the subclass is called user-defined. Membership in a subclass is determined by the database users by applying an operation to add an entity to the subclass. Membership in the subclass is specified individually for each entity in the super class by the user.



Disjoint vs. overlapping constraint

Another type of constraint relates to whether or not entities may belong to more than one lower-level entity set within a single generalization.

Disjoint Constraint

It specifies that the subclasses of the specialization must be disjoint. Here an entity can be a member of at most one of the subclasses of the specialization and it is represented by d in EER diagram. Simply, if an entity can be a member of at most one of the subclasses of the specialization, then the subclasses are called disjoint.

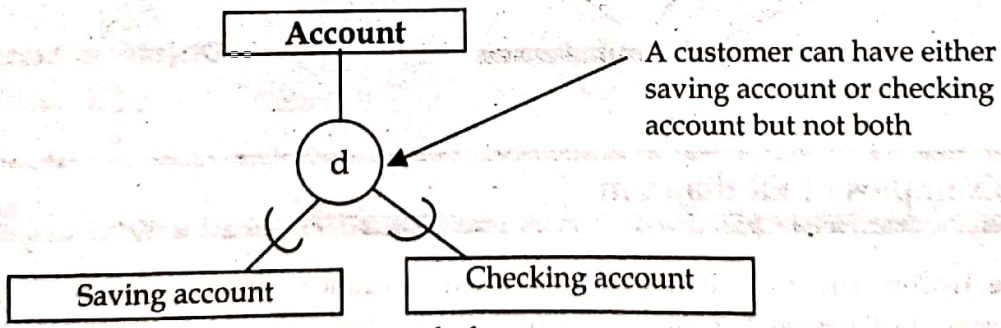


Figure 3.9: disjoint rule for account entity

Overlapping constraint

It specifies that the subclasses are not constrained to be disjoint, i.e., the same entity may be a member of more than one subclass of the specialization and it is represented by o in EER diagram.

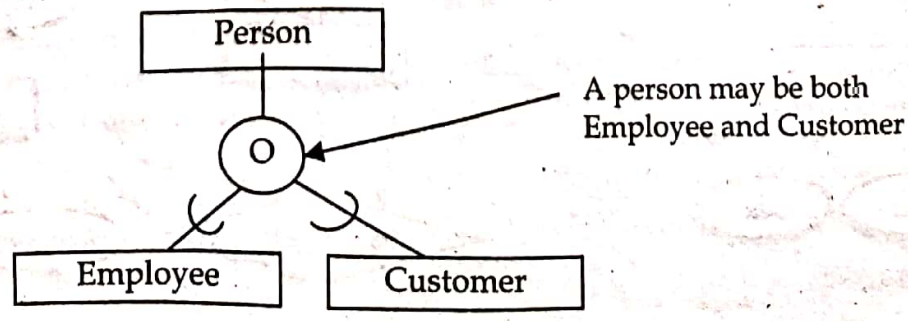


Figure 3.10: Overlap rule for person entity

Total vs. Partial Completeness Constraint

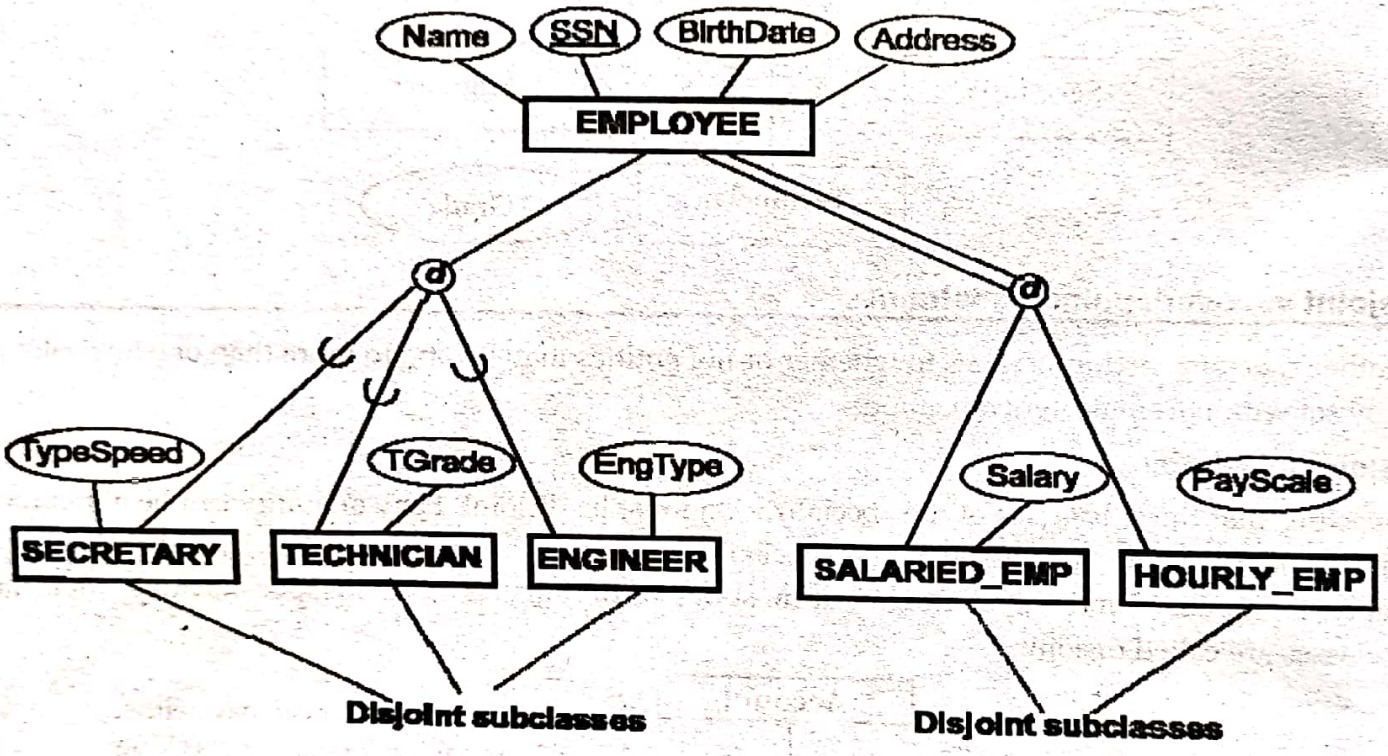
Total participation constraint

Total participation constraint specifies that every entity in the super class must be a member of some subclass in the specialization/generalization. It is represented by double line in EER diagram.

Partial participation constraint

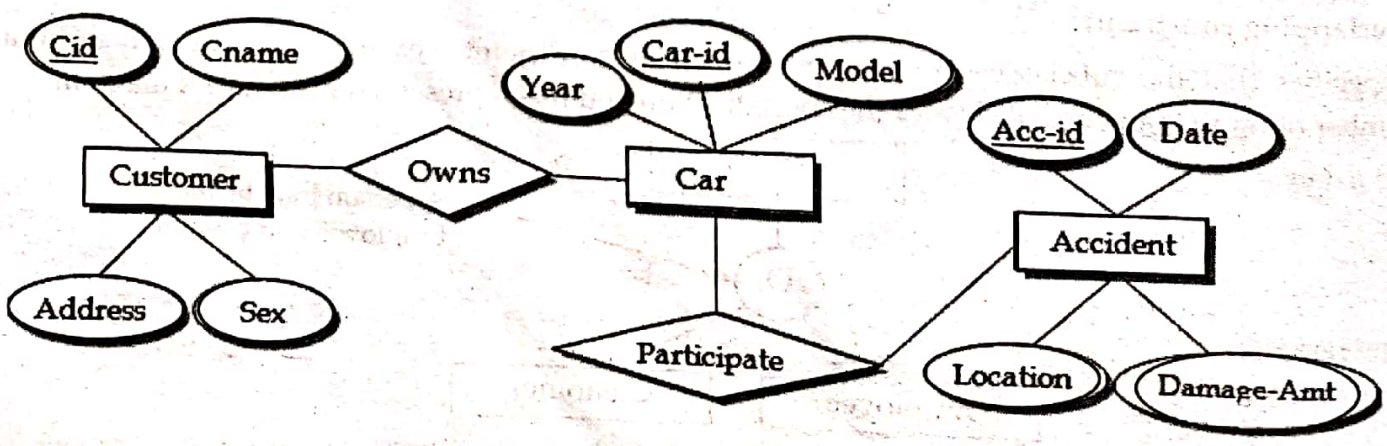
It allows every entity in the super class may not belong to a subclass and shown in EER diagrams by a single line.

Example: if some Employee entities, (for example, sales representatives) do not belong to any of the subclasses {Secretary, Engineer, Technician}, then the specialization is partial.



3.13 Examples of ER diagram

Example 1: 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. Assume attributes of your own interest:



Example 2: Consider a bus ticketing system that records information about the passenger, bus and route. Passenger is assigned to a bus travels to route. A bus contains many passengers and a passenger can be assigned into only one bus. Many buses travel in same route but a bus can travel in only one route. The attributes of passenger are pid (unique), gender and telephone (multi-valued). Similarly bus contains regno (unique) and color and route contains rid (unique), distance and rate (based on distance). Now draw the E-R diagram to represent this situation.