UNIT-2

Syllabus

Database Design,Design process,Entity Relation Model,ER diagram,Keys , Attributes and Constraints,Mapping Cardinality,Extended ER - Generalization, Specialization and Aggregation,ER Diagram Issues,Weak Entity,Relational Model,Conversion of ER to Relational Table.

DATABASE DESIGN

The database design process can be divided into six steps. The ER model is most relevant to the first three steps

(i) **Requirements Analysis**: The very first step in designing a database application is to understand what data is to be stored in the database, what applications must be built on top of it, and what operations are most frequent and subject to performance requirements. In other words, we must find out what the users want from the database.

(ii) **Conceptual Database Design**: The information gathered in the requirements analysis step is used to develop a high-level description of the data to be stored in the database, along with the constraints that are known to hold over this data. This step is often carried out using the ER model, or a similar high-level data model, and is discussed in the rest of this chapter.

(iii**) Logical Database Design:** We must choose a DBMS to implement our database design, and convert the conceptual database design into a database schema in the data model of the chosen DBMS. We will only consider relational DBMS's, and therefore, the task in the logical design step is to convert an ER schema into a relational databaseschema.

Beyond the ER Model: ER modeling is sometimes regarded as a complete approach to designing a logical database schema. This is incorrect because the ER diagram is just an approximate description of the data, constructed through a very subjective evaluation of the information collected during requirements analysis. The remaining three steps of database design are briefly described below:

(iv) **Schema Refinement:** The fourth step in database design is to analyze the collection of relations in our relational database schema to identify potential problems, and to refine it. In contrast to the requirements analysis and conceptual design steps, which are essentially subjective, schema refinement can be guided by some elegant and powerful theory.

(v) **Physical Database Design:** In this step we must consider typical expected workloads that our database must support and further refine the database design to ensure that it meets desired performance criteria. This step may simply involve building indexes on some tables and clustering some tables, or it may involve a substantial redesign of parts of the database schema obtained from the earlier design steps.

(vi) **Security Design**: In this step, we identify different user groups and different roles played by various users (e.g., the development team for a product, the customer support representatives, and the product manager). For each role and user group, we must identify the parts of the database that they must be able to

access and the parts of 19 the database that they should not be allowed to access, and take steps to ensure that they can access only the necessary parts.

Database Design Techniques

Generally we can design the database in two different approaches.

1. Top-Down Design (Analysis) methodology

   It starts with the major entities of their interest, their attributes and their relationships. And then we add other entities and may split these entities into a number of specialized entities and add the relationships between these entities.

2. Bottom-Up Design

   It starts with a set of attributes. And group these attributes into entities. Then find out the relationship between these entities. Identify the higher-level entities, generalized these entities and locate relationships at this higher level.

Problems with bad schema

1. Redundant storage of data:
2. Wastage of disc space

ER Model

Entity- Relationship (E-R) Diagram

ER Model is used to model the logical view of the system from data perspective which consists of these components:

Entity, Entity Type, Entity Set –

ENTITY

An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.

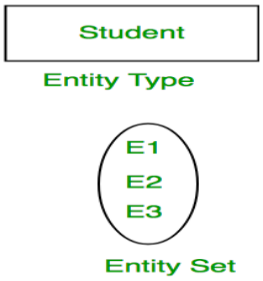
An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

ENTITY TYPE

An Entity is an object of Entity Type and set of all entities is called as entity set. e.g.; E1 is an entity having Entity Type Student and set of all students is called Entity Set. In ER diagram, Entity Type is represented as

ENTITY SET

An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

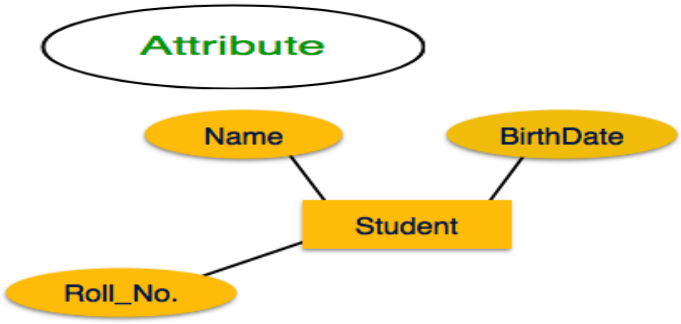


Attribute(s)&Constraints

Entities are represented by means of their properties, called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes.

There exists 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 the **properties which define the entity type**. For example, Roll_No, Name, DOB, Age, Address, Mobile_No are the attributes which defines entity type Student. In ER diagram, attribute is represented by an oval.



Types of Attributes

1. **Key Attribute –**

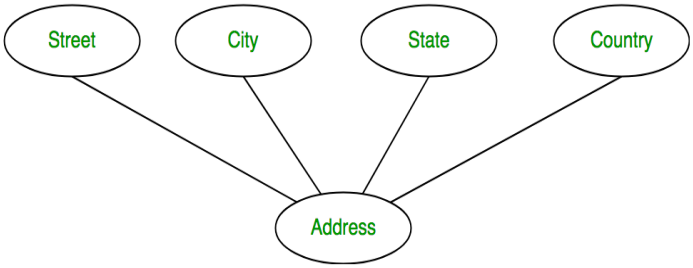
The attribute which **uniquely identifies each entity** in the entity set is called key attribute. For example, Roll_No will be unique for each student. In ER diagram, key attribute is represented by an oval with underlying lines.

Simple attribute − Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.

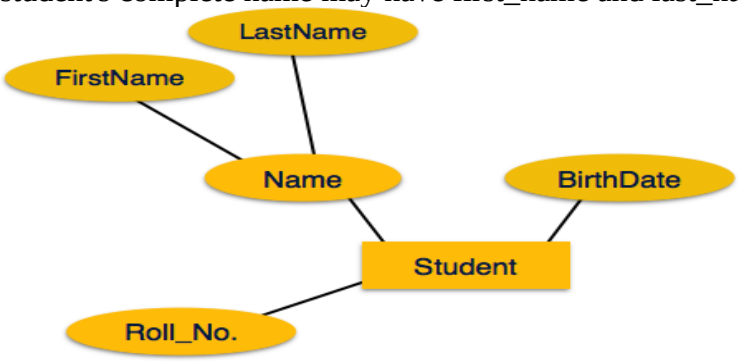
Single-value attribute − Single-value attributes contain single value. For example − Social_Security_Number.

2. Composite Attribute –

An attribute **composed of many other attribute** is called as composite attribute. For example, Address attribute of student Entity type consists of Street, City, State, and Country. In ER diagram, composite attribute is represented by an oval comprising of ovals.



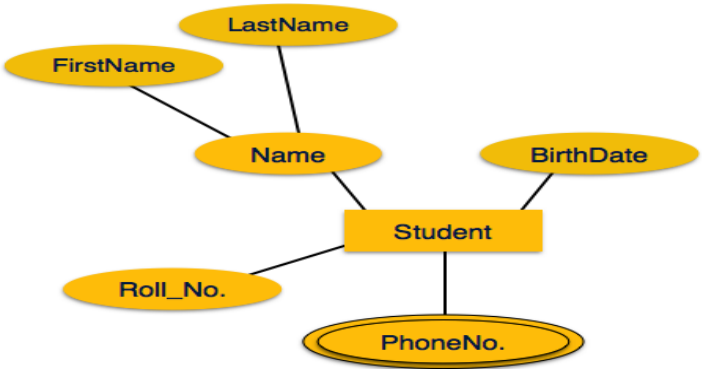
Composite attribute − Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.



3.Multivalued Attribute –

An attribute consisting **more than one value** for a given entity. For example, Phone_No (can be more than one for a given student). In ER diagram, multivalued attribute is represented by double oval.

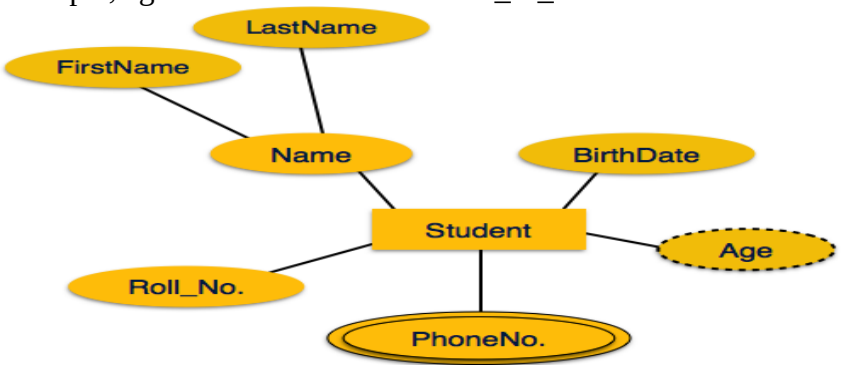
Multi-value attribute − Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.



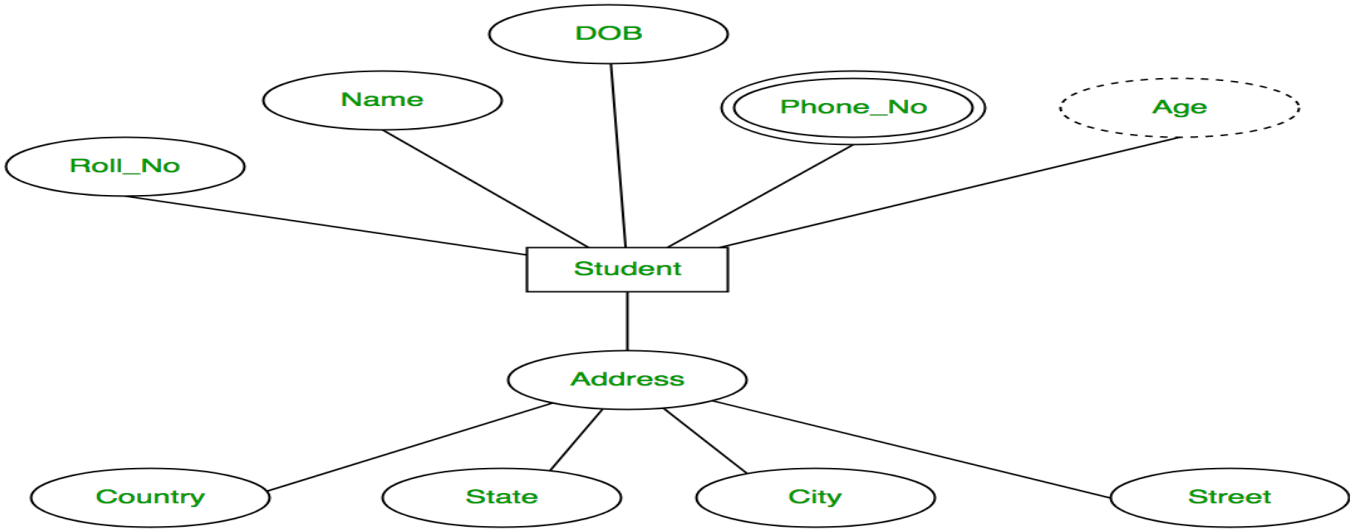
4.Derived Attribute –

An attribute which can be **derived from other attributes** of the entity type is known as derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, derived attribute is represented by dashed oval.

Derived attribute − Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.



The complete entity type **Student** with its attributes can be represented as:



Relationship Type and Relationship Set:

Relationship

The association among entities is called a relationship. For example, an employee **works_at** a department, a student **enrolls** in a course. Here, Works_at and Enrolls are called relationships.

Relationship Set

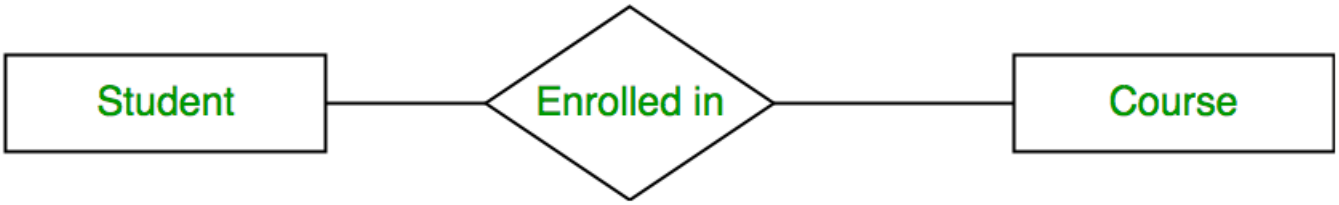
A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called **descriptive attributes**.

Degree of Relationship

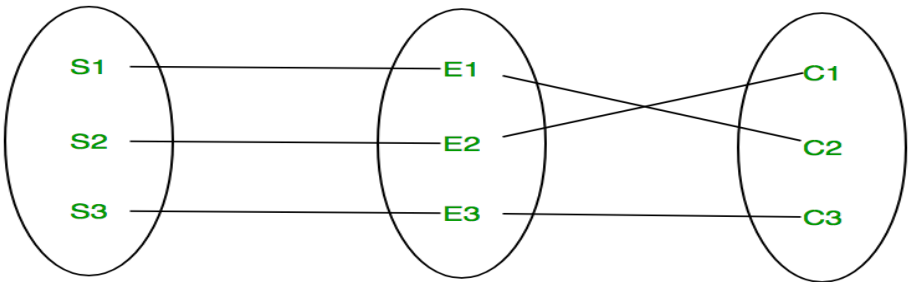
The number of participating entities in a relationship defines the degree of the relationship.

- Binary = degree 2
- Ternary = degree 3
- n-ary = degree

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.



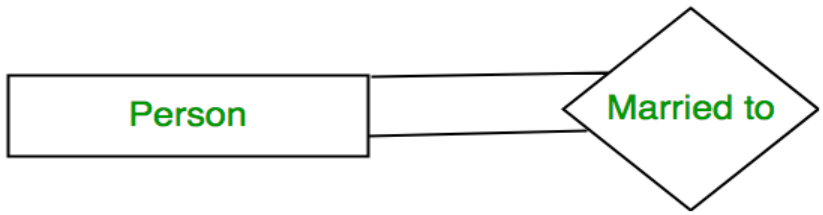
A set of relationships of same type is known as relationship set. The following relationship set depicts S1 is enrolled in C2, S2 is enrolled in C1 and S3 is enrolled in C3.



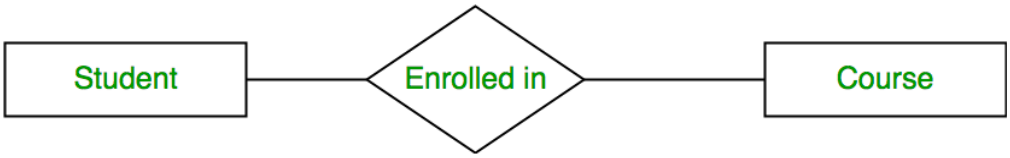
Degree of a relationship set

The number of different entity sets **participating in a relationship** set is called as degree of a relationship set.

1. Unary Relationship –
When there is **only ONE entity set participating in a relation**, the relationship is called as unary relationship. For example, one person is married to only one person.

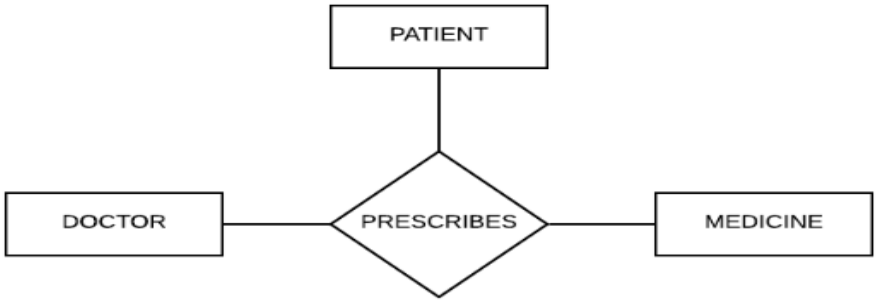


2. Binary Relationship –
When there are **TWO entities set participating in a relation**, the relationship is called as binary relationship.For example, Student is enrolled in Course.



3. Ternary Relationship

When there is a relationship between three different entities, it is known as a ternary relationship. An example of a ternary relationship can be shown as follows −



In this example, there is a ternary relationship between Doctor, Patient and Medicine.

4. n-ary Relationship –
When there are n entities set participating in a relation, the relationship is called as n-ary relationship.

CONSTRAINTS

Constraints are used for modeling limitations on the relations between entities.

There are two types of constraints on the Entity Relationship (ER) model −

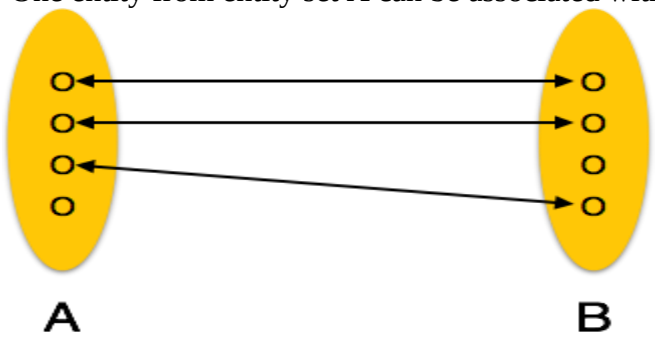
1. Mapping cardinality or cardinality ratio.
2. Participation constraints.

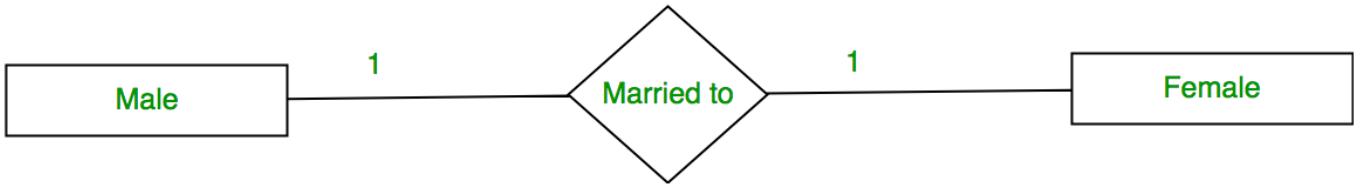
1. Mapping Cardinality:

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:

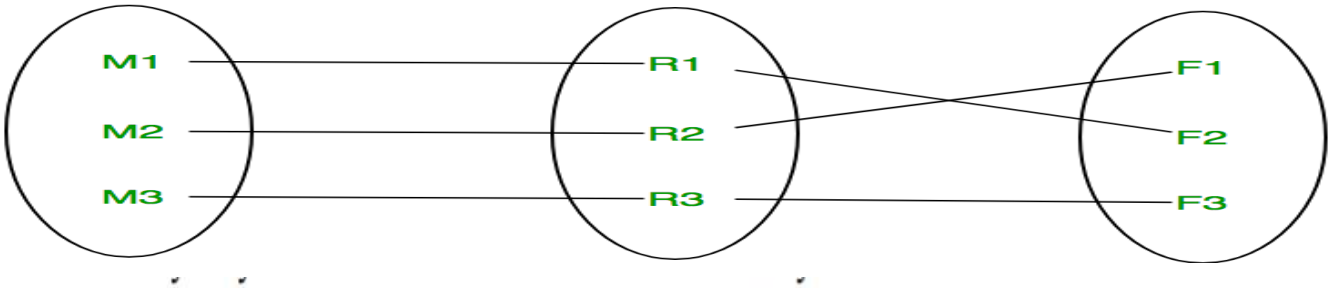
i. **One to one –** When each entity in each entity set can take part **only once in the relationship**, the cardinality is one to one. Let us assume that a male can marry to one female and a female can marry to one male. So the relationship will be one to one.

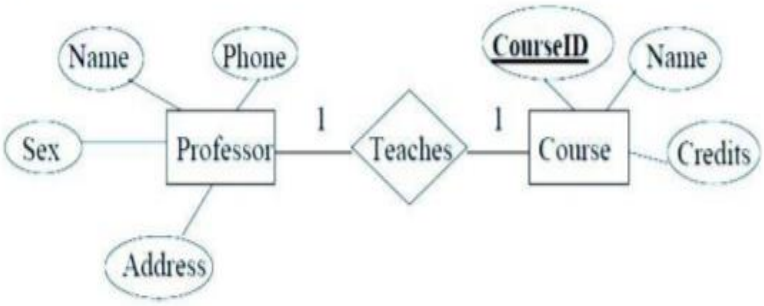
One entity from entity set A can be associated with at most one entity of entity set B and vice versa.





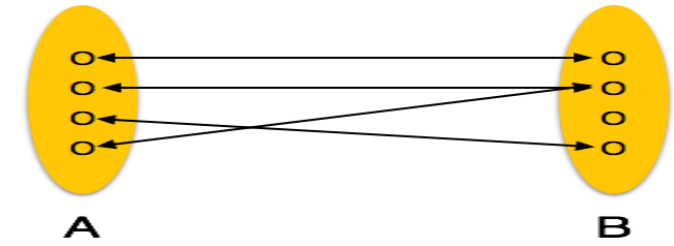
Using Sets, it can be represented as:



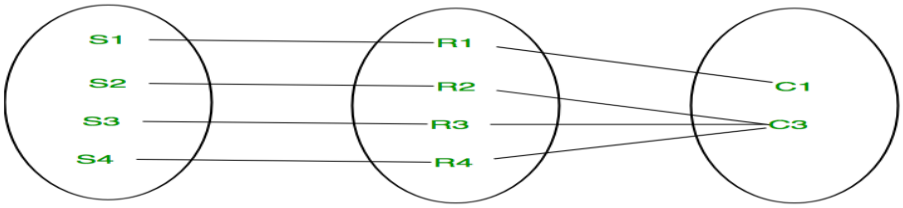


ii) **Many to one & one to Many–** When entities in one entity set **can take part only once in the relationship set and entities in other entity set can take part more than once in the relationship set,** cardinality is many to one. Let us assume that a student can take only one course but one course 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.

Many-to-one − More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.

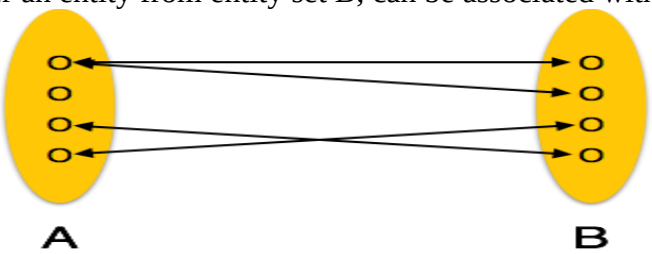


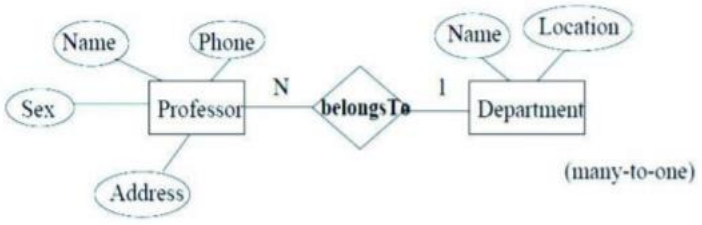
Using Sets, it can be represented as:

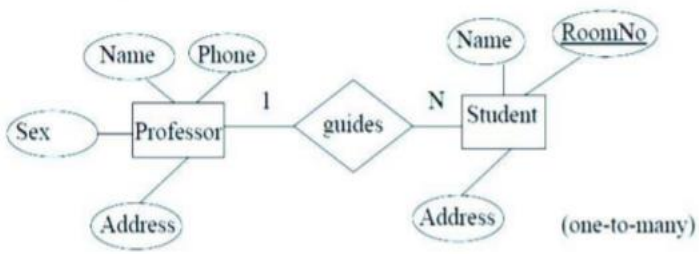


In this case, each student is taking only 1 course but 1 course has been taken by many students.

One-to-many − One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.

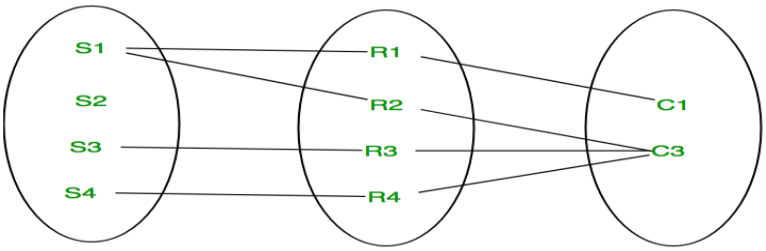




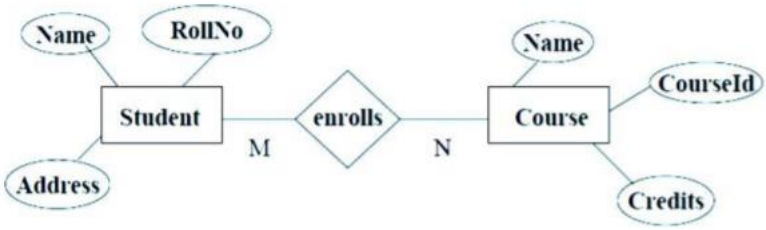


iii) **Many to many –** When entities in all entity sets can **take part more than once in the relationship** cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.

Using sets, it can be represented as:



In this example, student S1 is enrolled in C1 and C3 and Course C3 is enrolled by S1, S3 and S4. So it is many to many relationships.



2.Participation Constraint:

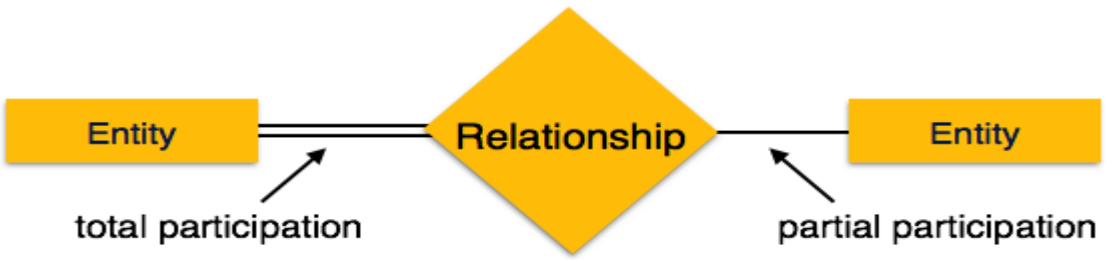
Participation Constraint is applied on the entity participating in the relationship set.

i. **Total Participation –** 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.

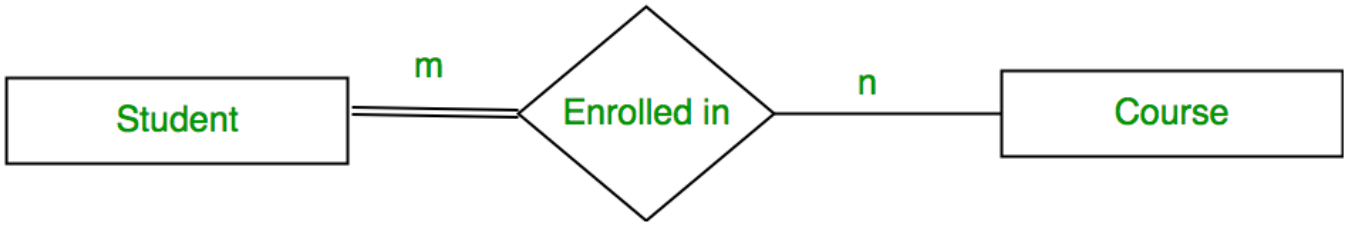
-Each entity is involved in the relationship. Total participation is represented by double lines.

ii. **Partial Participation –** The entity in the entity set **may or may NOT participat**e in the relationship. If some courses are not enrolled by any of the student, the participation of course will be partial.

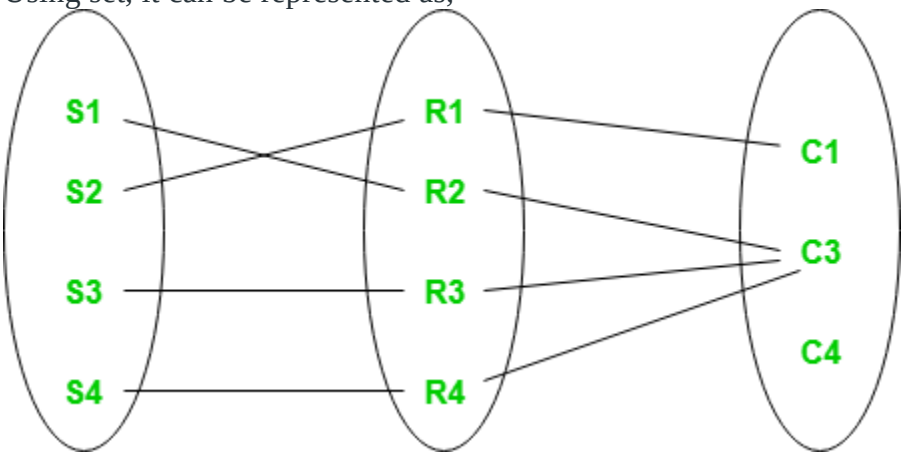
-Not all entities are involved in the relationship. Partial participation is represented by single lines.



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



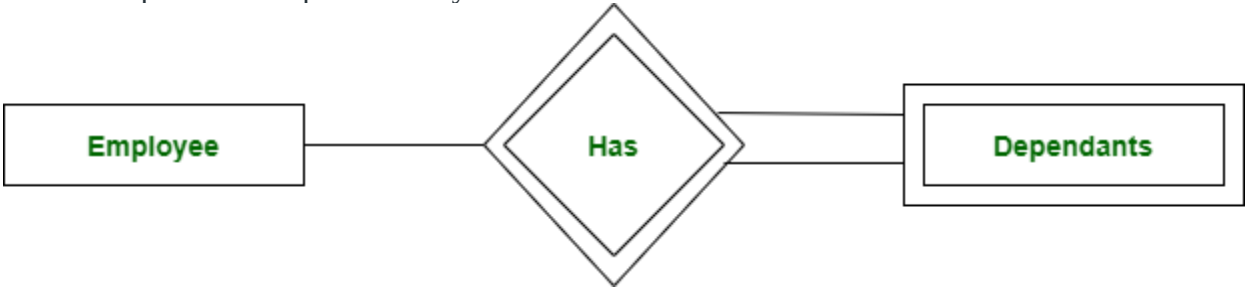
Using set, it can be represented as,



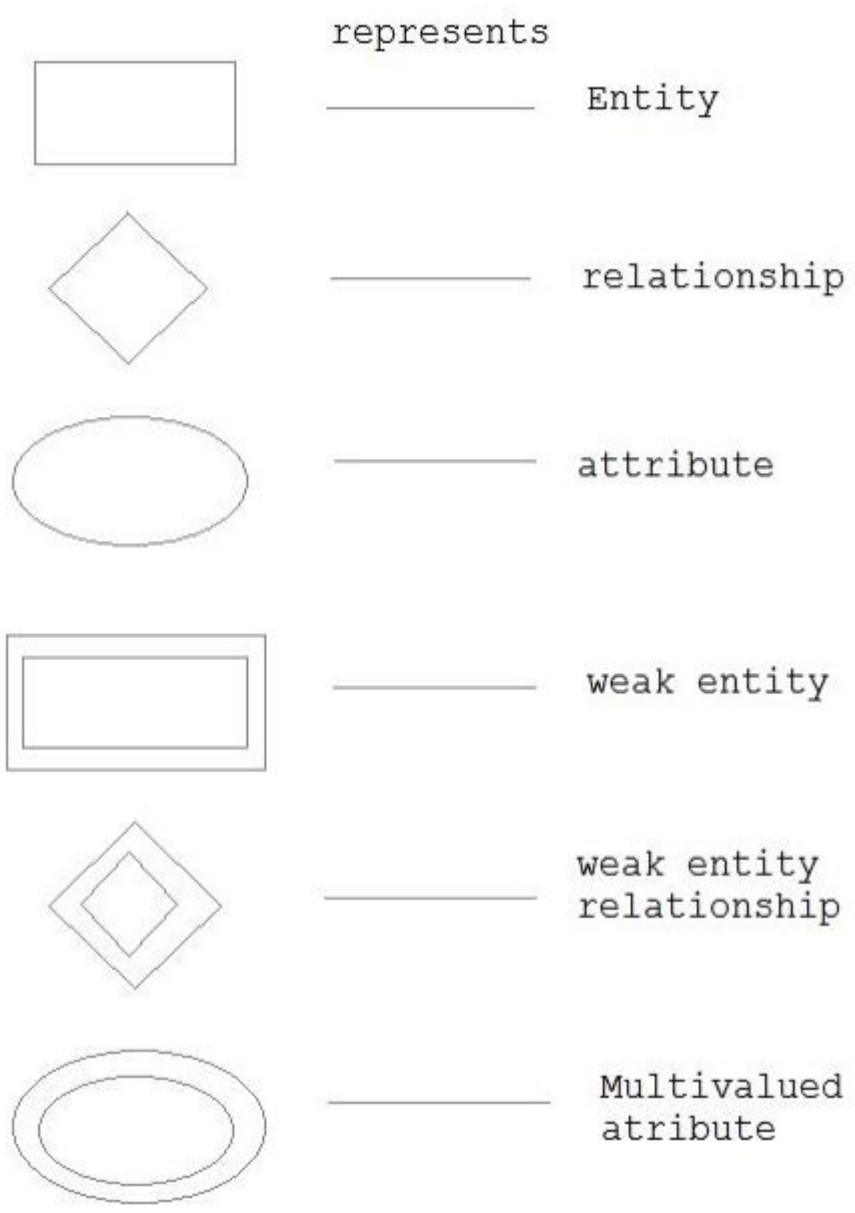
Every student in Student Entity set is participating in relationship but there exists a course C4 which is not taking part in the relationship.

Weak Entity Type and Identifying Relationship:

As discussed before, 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.
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.
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.



E-R DIAGRAM REPRESENTATIONS



KEY

Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.

For example, the roll_number of a student makes him/her identifiable among students.

- **Super Key** − A set of attributes (one or more) that collectively identifies an entity in an entity set.
- **Candidate Key** − A minimal super key is called a candidate key. An entity set may have more than one candidate key.
- **Primary Key** − A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

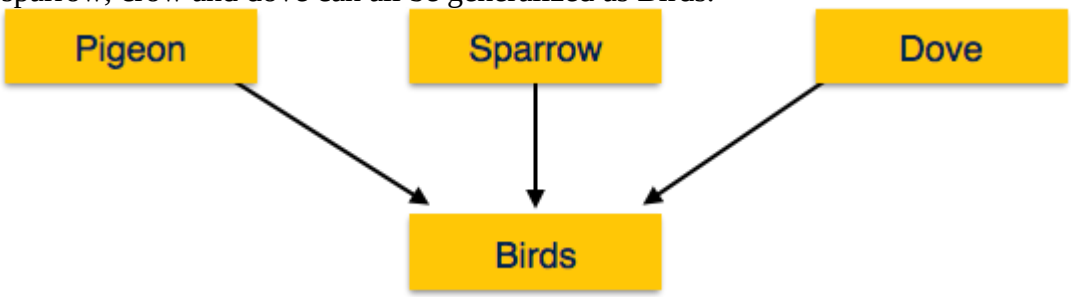
Extended ER

The ER Model has the power of expressing database entities in a conceptual hierarchical manner. As the hierarchy goes up, it generalizes the view of entities, and as we go deep in the hierarchy, it gives us the detail of every entity included.

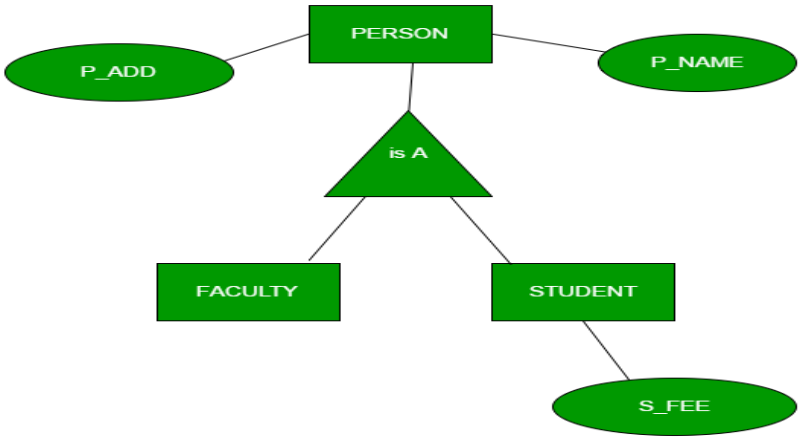
Going up in this structure is called **generalization**, where entities are clubbed together to represent a more generalized view. For example, a particular student named Mira can be generalized along with all the students. The entity shall be a student, and further, the student is a person. The reverse is called **specialization** where a person is a 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 example, pigeon, house sparrow, crow and dove can all be generalized as Birds.



Generalization is the process of extracting common properties from a set of entities and create a generalized entity from it. It is a bottom-up approach in which two or more entities can be generalized to a higher level entity if they have some attributes in common. For Example, STUDENT and FACULTY can be generalized to a higher level entity called PERSON as shown in Figure 1. In this case, common attributes like P_NAME, P_ADD become part of higher entity (PERSON) and specialized attributes like S_FEE become part of specialized entity (STUDENT).



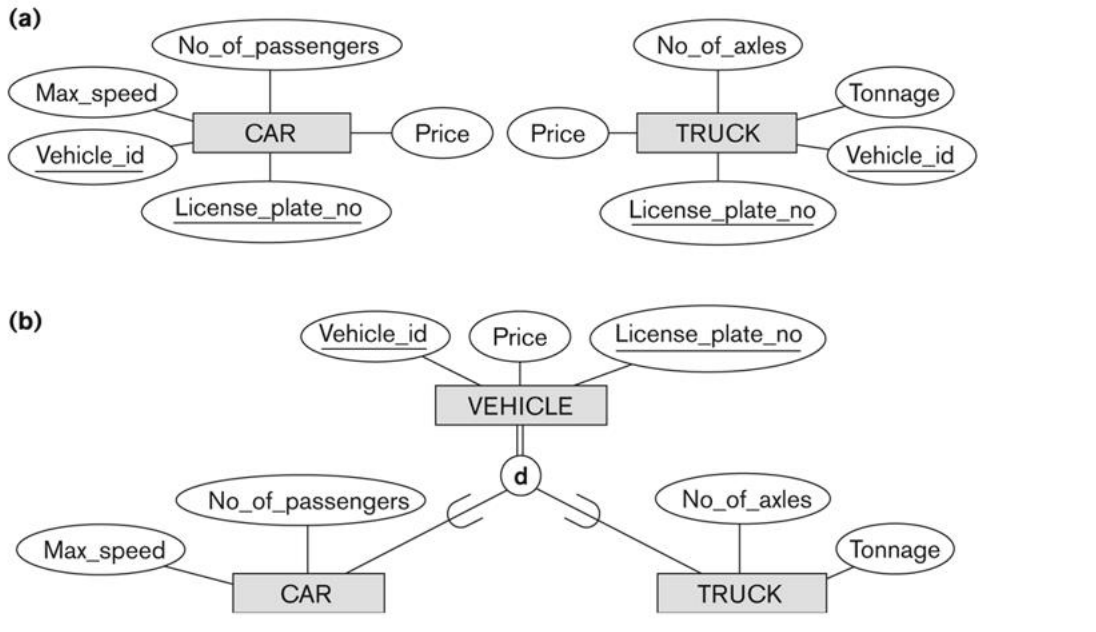
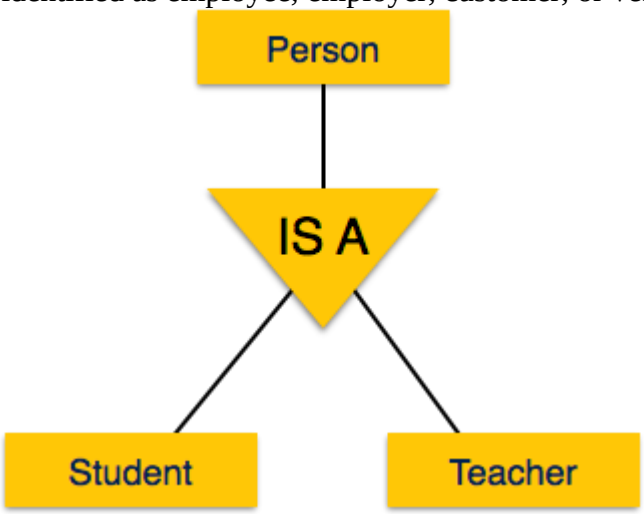


Figure 4.3
Generalization. (a) Two entity types, CAR and TRUCK.
(b) Generalizing CAR and TRUCK into the superclass VEHICLE.

Specialization

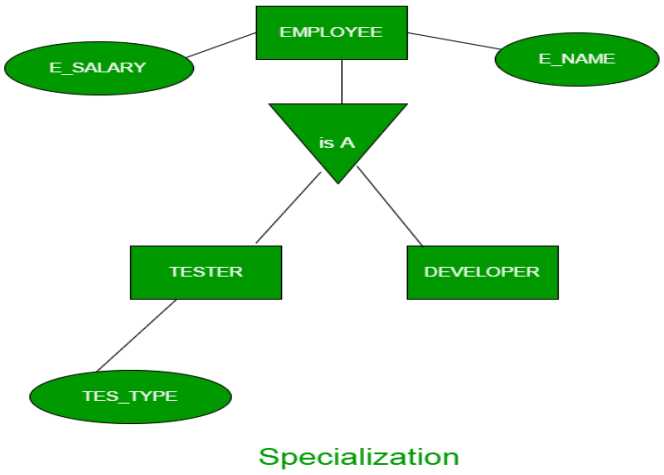
Specialization is the opposite of generalization. In specialization, a group of entities is divided into sub-groups based on their characteristics. Take a group 'Person' for example. A person has name, date of birth, gender, etc. These properties are common in all persons, human beings. But in a company,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



Similarly, in a school database, persons can be specialized as teacher, student, or a staff, based on what role they play in school as entities.

In specialization, an entity is divided into sub-entities based on their characteristics. It is a top-down approach where higher level entity is specialized into two or more lower level entities. For Example, EMPLOYEE entity in an Employee management system can be specialized into DEVELOPER, TESTER etc. as shown in Figure 2. In this case, common attributes like E_NAME, E_SAL etc. become part of

higher entity (EMPLOYEE) and specialized attributes like TES_TYPE become part of specialized entity (TESTER).



Specialization

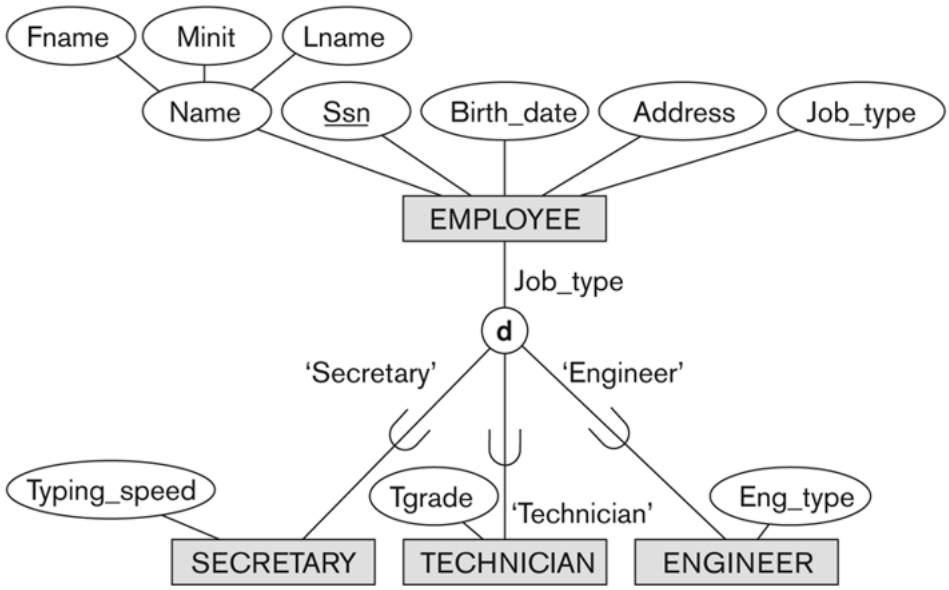
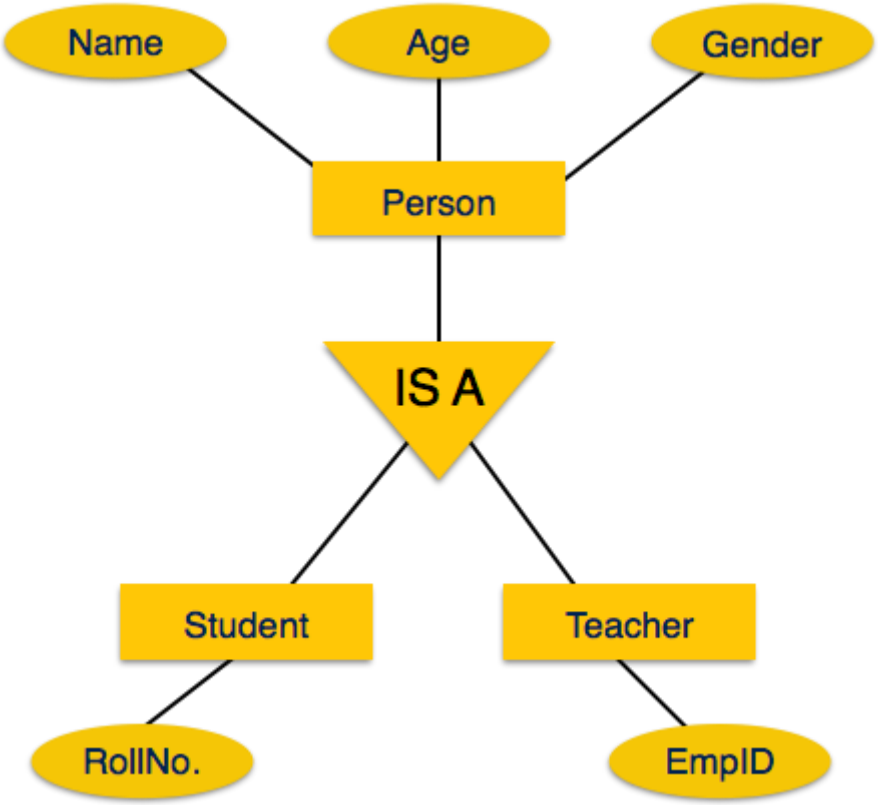


Figure 4.4
EER diagram notation for an attribute-defined specialization on Job_type.

Inheritance

We use all the above features of ER-Model in order to create classes of objects in object-oriented programming. The details of entities are generally hidden from the user; this process known as **abstraction**. Inheritance is an important feature of Generalization and Specialization. It allows lower-level entities to inherit the attributes of higher-level entities.

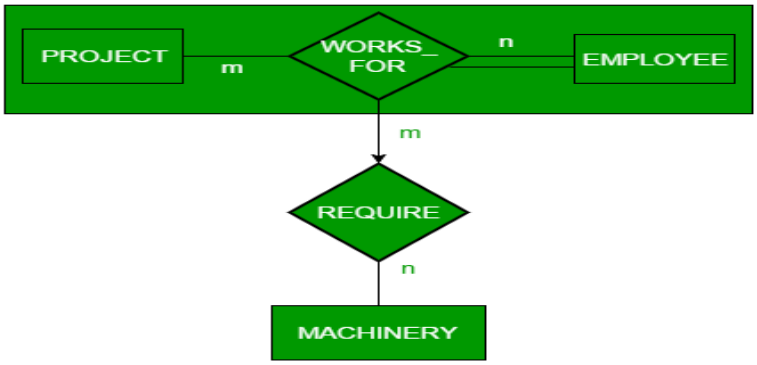


For example, the attributes of a Person class such as name, age, and gender can be inherited by lower-level entities such as Student or Teacher.

Aggregation –

An ER diagram is not capable of representing relationship between an entity and a relationship which may be required in some scenarios. In those cases, a relationship with its corresponding entities is aggregated into a higher level entity. Aggregation is an abstraction through which we can represent relationships as higher level entity sets.

For Example, Employee working for a project may require some machinery. So, REQUIRE relationship is needed between relationship WORKS_FOR and entity MACHINERY. Using aggregation, WORKS_FOR relationship with its entities EMPLOYEE and PROJECT is aggregated into single entity and relationship REQUIRE is created between aggregated entity and MACHINERY.



Aggregation

Representing aggregation via schema –
To represent aggregation, create a schema containing:

1. primary key of the aggregated relationship
2. primary key of the associated entity set
3. descriptive attribute, if exists.

ER Design Issues

In the previous sections of the data modeling, we learned to design an ER diagram. We also discussed different ways of defining entity sets and relationships among them. We also understood the various designing shapes that represent a relationship, an entity, and its attributes. However, users often mislead the concept of the elements and the design process of the ER diagram. Thus, it leads to a complex structure of the ER diagram and certain issues that does not meet the characteristics of the real-world enterprise model.

Here, we will discuss the basic design issues of an ER database schema in the following points:

1) Use of Entity Set vs Attributes

The use of an entity set or attribute depends on the structure of the real-world enterprise that is being modelled and the semantics associated with its attributes. It leads to a mistake when the user use the primary key of an entity set as an attribute of another entity set. Instead, he should use the relationship to do so. Also, the primary key attributes are implicit in the relationship set, but we designate it in the relationship sets.

2) Use of Entity Set vs. Relationship Sets

It is difficult to examine if an object can be best expressed by an entity set or relationship set. To understand and determine the right use, the user need to designate a relationship set for describing an action that occurs in-between the entities. If there is a requirement of representing the object as a relationship set, then its better not to mix it with the entity set.

3) Use of Binary vs n-ary Relationship Sets

Generally, the relationships described in the databases are binary relationships. However, non-binary relationships can be represented by several binary relationships. For example, we can create and represent a ternary relationship 'parent' that may relate to a child, his father, as well as his mother. Such relationship can also be represented by two binary relationships i.e, mother and father, that may relate to their child. Thus, it is possible to represent a non-binary relationship by a set of distinct binary relationships.

4) Placing Relationship Attributes

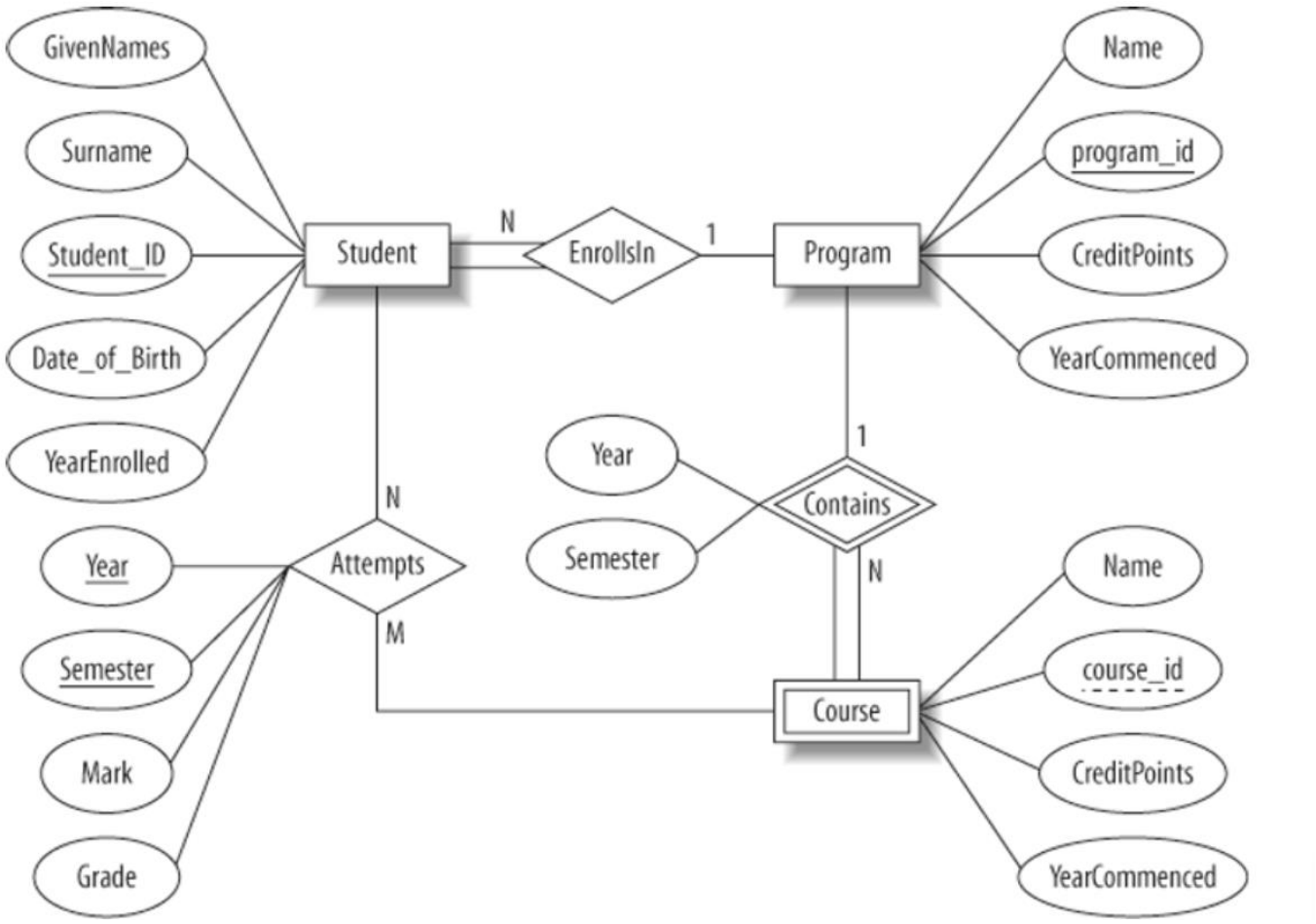
The cardinality ratios can become an affective measure in the placement of the relationship attributes. So, it is better to associate the attributes of one-to-one or one-to-many relationship sets with any participating entity sets, instead of any relationship set. The decision of placing the specified attribute as a relationship or entity attribute should possess the charactestics of the real world enterprise that is being modelled.

For example, if there is an entity which can be determined by the combination of participating entity sets, instead of determing it as a separate entity. Such type of attribute must be associated with the many-to-many relationship sets.

Thus, it requires the overall knowledge of each part that is involved inb desgining and modelling an ER diagram. The basic requirement is to analyse the real-world enterprise and the connectivity of one entity or attribute with other.

EXAMPLE

Question: Draw an ER Diagram for University Database?



HINT:

The university database stores details about university students, courses, the semester a student took a particular course (and his mark and grade if he completed it), and what degree program each student is

enrolled in. The database is a long way from one that'd be suitable for a large tertiary institution, but it does illustrate relationships that are interesting to query, and it's easy to relate to when you're learning SQL. We explain the requirements next and discuss their shortcomings at the end of this section.
Consider the following requirements list:

- The university offers one or more programs.
- A program is made up of one or more courses.
- A student must enroll in a program.
- A student takes the courses that are part of her program.
- A program has a name, a program identifier, the total credit points required to graduate, and the year it commenced.
- A course has a name, a course identifier, a credit point value, and the year it commenced.
- Students have one or more given names, a surname, a student identifier, a date of birth, and the year they first enrolled. We can treat all given names as a single object—for example, "John Paul."
- When a student takes a course, the year and semester he attempted it are recorded. When he finishes the course, a grade (such as A or B) and a mark (such as 60 percent) are recorded.
- Each course in a program is sequenced into a year (for example, year 1) and a semester (for example, semester 1).

In our design:

- Student is a strong entity, with an identifier, student_id, created to be the primary key used to distinguish between students (remember, we could have several students with the same name).
- Program is a strong entity, with the identifier program_id as the primary key used to distinguish between programs.
- Each student must be enrolled in a program, so the Student entity participates totally in the many-to-one EnrollsIn relationship with Program. A program can exist without having any enrolled students, so it participates partially in this relationship.
- A Course has meaning only in the context of a Program, so it's a weak entity, with course_id as a weak key. This means that a Course is uniquely identified using its course_id and the program_id of its owning program.
- As a weak entity, Course participates totally in the many-to-one identifying relationship with its owning Program. This relationship has Year and Semester attributes that identify its sequence position.
- Student and Course are related through the many-to-many Attempts relationships; a course can exist without a student, and a student can be enrolled without attempting any courses, so the participation is not total.
- When a student attempts a course, there are attributes to capture the Year and Semester, and the Mark and Grade.

RELATIONAL MODEL

Basic Structure of relational model - The relational model for database management is a data model based on predicate logic and set theory. It was invented by Edgar Codd. The fundamental assumption of the relational model is that all data are represented as mathematical n-ary **relations**, an n-ary relation being a subset of the Cartesian product of n sets.

1) Relation - The fundamental organizational structure for data in the relational model is the relation. A *relation* is a two-dimensional table made up of rows and columns. Each relation also called a table, stores data about *entities.*

2) Tuples - The rows in a relation are called tuples. They represent specific occurrences (or records) of an entity. Each row consists of a sequence of values, one for each column in the table. In addition, each row (or record) in a table must be unique. A tuple variable is a variable that stand for a tuple.

3) Attributes – The column in a relation is called attribute. The attributes represent characteristics of an entity.

4) Domain – For each attribute there is a set of permitted values called domain of that attribute. For all relations 'r', the domain of all attributes of 'r' should be atomic. A domain is said to be **atomic** if elements of the domain are considered to be indivisible units.

Database Schema – Logical design of the database is termed as database schema.

Database instance – Database instance is a snapshot of the data in a database at a given instant of time.

Relation schema – The concept of relation schema corresponds to the programming notion of type definition. It can be considered as the definition of a domain of values. The database schema is the collection of relation schemas that define a database.

Relation instance – The concept of a relation instance corresponds to the programming language notion of a value of a variable. For relation instance, we actually mean the "relation" itself.

Schema diagram – A database schema, along with primary key and foreign key dependencies, can be depicted pictorially by schema diagrams. Each relation in the database schema is represented as a box, with the attributes listed inside it and the relation name above it. If there are primary key attributes, a horizontal line crosses the box, with the primary key attributes listed above the line. Foreign key dependencies appear as arrows from the foreign key attributes of the referencing relation to the foreign key attributes of the referenced relation.

Relational algebra – The relational algebra is a procedural query language. (A query language is a language in which a user requests information from the database.) It consists of a set of operations that take one or two

relations as input and produce a new relation as the result. The fundamental operations in relational algebra are select, project, union, set difference, Cartesian product and rename. There are several other operations namely, set intersection, natural join, division and assignment.

E.g. Consider the borrow relation in the banking example:

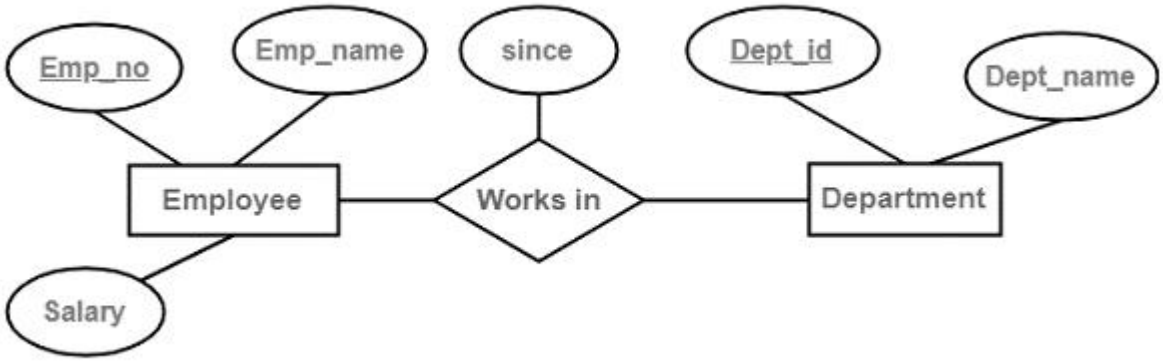
Branch name	Loan#	Customer name	Amount
Downtown	17	Jones	1000
Round Hill	23	Smith	2000
Redwood	13	Hayes	1300

Borrow relation

Conversion of ER to Relational Table

Entity relationship diagram is the graphical representation of entities and relationships among those entities in the database.

Example



Conversion of ER diagrams to tables

Follow the steps given below for the conversion of the ER diagrams to tables in the database management system (DBMS) −

Step 1 − Conversion of strong entities

- For each strong entity create a separate table with the same name.
- Includes all attributes, if there is any composite attribute divided into simple attributes and has to be included.
- Ignore multivalued attributes at this stage.
- Select the p key for the table.

Step 2 − Conversion of weak entity

- For each weak entity create a separate table with the same name.
- Include all attributes.
- Include the P key of a strong entity as foreign key is the weak entity.
- Declare the combination of foreign key and decimator attribute as P key from the weak entity.

Step 3 − Conversion of one-to-one relationship

- For each one to one relation, say A and B modify either A side or B side to include the P key of the other side as a foreign key.
- If A or B is having total participation, then that should be a modified table.
- If a relationship consists of attributes, include them also in the modified table.

Step 4 − Conversion of one-to-many relationship

- For each one to many relationships, modify the M side to include the P key of one side as a foreign key.
- If relationships consist of attributes, include them as well.

Step 5 − Conversion of many-many relationship

- For each many-many relationship, create a separate table including the P key of M side and N side as foreign keys in the new table.
- Declare the combination of foreign keys as P for the new table.
- If relationships consist of attributes, include them also in the new table.

Step 6 − Conversion of multivalued attributes

- For each multivalued attribute create a separate table and include the P key of the present table as foreign key.
- Declare the combination of foreign key and multivalued attribute as P keys.

Step 7 − Conversion of n-ary relationship

- For each n-ary relationship create a separate table and include the P key of all entities as foreign key.
- Declare the combination of foreign keys as P key.

Table

After successful conversion, the result will be as follows −

Emp_no	Dept_id	since

Schema : Works in (Emp_no , Dept_id , since)