Chapter-2
Data Model

Introduction

A data models is a collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints. Every database and database management system is based on particular database model. Data model consists of set of rules and standards that define how the data is organized in a database. A data model also provides a set of operation for manipulation the data stored in the database. Different models provide partial conceptualizations of databases and they have different outlooks and different perspectives.

There are three different data models.

1. Physical data model
2. Logical data model
3. Conceptual data model

1. Physical data model (Low Level data model)

The physical data model describes the way how data is stored in the computer by representing records format, record ordering and access path. The physical data model describes the way data are saved on storage media such as disks or tapes. The physical model requires the definition of both the physical storage devices and the access method required to reach the data within the storage device. With the physical model, it is possible to implement the database at the system level. A very few physical data models have been purposed so far. Two of these well known models are the unifying model and the frame memory model.

Advantage: possible to implement the database at system level.

Disadvantage: Complex to implement.

2. Logical data model (Record Based logical data model)

These models are used to specify the overall logical structures of database. Record based logical models are used in describing data at the conceptual level and view levels. Three most widely used record based data model are

i. Hierarchical data model
ii. Network data model
iii. Relational data model

Hierarchical data model

Hierarchical data model organizes the data in tree structure i.e. in the form of parent-child relationship. The origin of the data tree is root. Data accessed at different levels along with a particular branch from the root is called node. The last node in the chain is called leaf. Here the parent can have any numbers of children but each child record has only one parents. Deleting a parent record requires deleting all its child records. This structure implies that a record can have replicating information generally in the child data segments. So it increases the problem of data redundancy. It can handle one to many relationship. Relationship is represented by pointer. Searching operation is very difficult however if the parent is known searching speeds up. Modification and addition of data is a very hard task so it is not flexible data model. DML is a low level language.

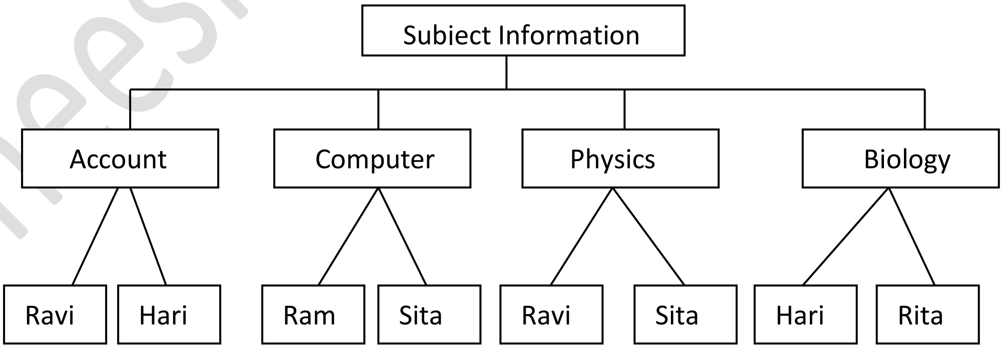


Fig: Hierarchical data model of subject information tables

Advantages:

1. Promotes data integrity.
2. High Data Security.
3. Very useful when database includes large number of one to many relationship.

Disadvantages:

1. Complex to implement.
2. Application programming is very time consuming and complicated.
3. The data of this model are not portable.
4. Greater chance of data loss.

Network data model

The network data model extends the hierarchical data model by permitting the child to have zero, one or more parents. Hence it accepts any relationship (one to many and many to many). So Network data model is more flexible than hierarchical model. Searching is faster because multidirectional pointer are available. However the use of pointer leads to complexity of structure. Also insertion, updation and deletion of any records require pointer adjustment. Since data should not be repeated so it reduces the problem of data redundancy that exists in hierarchical model. It is very complex data model so requires long program to handle the relationship. DML is a low level language. This model is less secured in comparison to hierarchical model because it is open to all. Network model are used for high volume transaction processing.

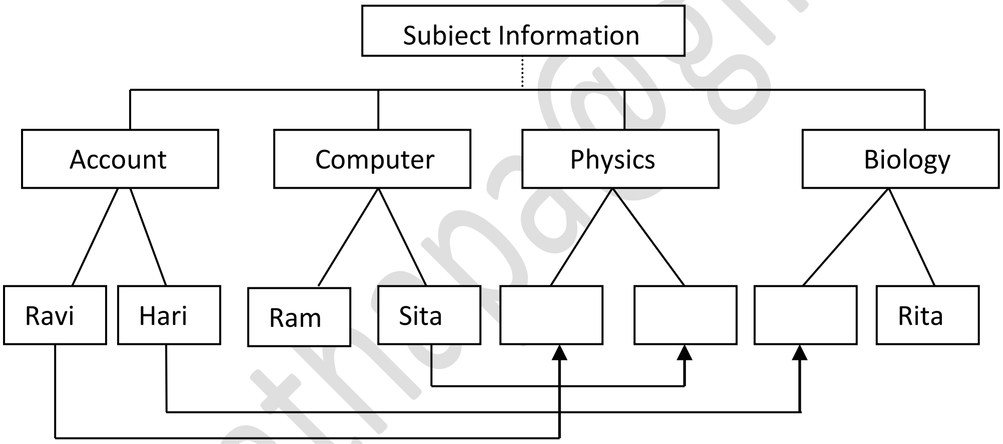


Fig: Network data model for subject information

Advantages:

1. It supports many to many relationships.
2. Simple and easy to design.
3. Data is consistent because data is located at only one place.
4. Searching is faster because of multidirectional pointer

Disadvantages:

1. Use of pointer leads to complexity in structure.
2. Insertion, updation, and deletion of any record(s) records would require pointer adjustment.
3. Less secured.

Relational data model

Note: See chapter 3

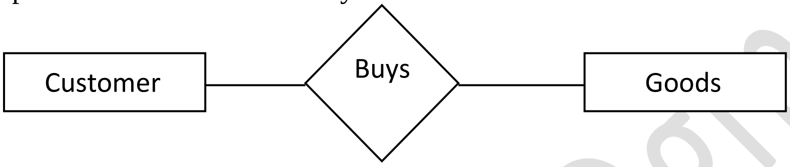
3. **Conceptual model (High level or object based logical model)**
   These models are used for describing data and data relationship. So it uses the concept such as entities, attributes, relationships etc. the most widely used conceptual model is entity relationship (E-R) model. The conceptual model represents the global view of data.

Data modeling using E-R model

The E-R model is a conceptual model which is based on the perception of the world consisting of collection of basic objects (entities) and relationship among these objects. To develop a database model based on E-R technique, entities and their relationship should be identified first. The E-R model is very useful in mapping the meaning and interactions of real word enterprise onto a conceptual schema. E-R diagram can be used to graphically represent the overall structure of database.

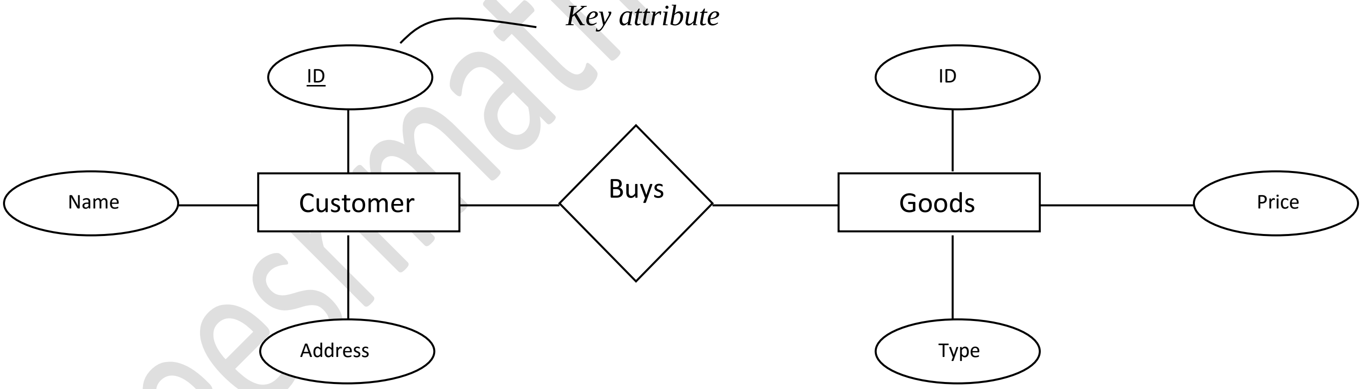
Basic Concepts

Entity and Entity set: An entity is an object or thing that has its existence in the real world and is distinguishable from other objects. It includes all those things about which data is collected. An entity may be person, place or thing which can be distinctly identified.
For example, each person in a university is an entity. An entity has a set of properties, and the values for some set of properties may uniquely identify an entity. For instance, a person may have a person id property whose value uniquely identifies that person.
An entity set is a set of entities of same type that share same properties or attributes. Example: The set of all persons who are customers at a given shop can be defined as the entity set costumer.



In the above figure, if we say that a customer buys goods, it means customers and goods are entity set. The notation of **Entity set** in E-R diagram is **Rectangle**. **Diamond** with a name inside is used to represent their **relationship.**

Attributes: Attributes are units that are used to describe the characteristics or properties of entities set. Attributes are descriptive properties possessed by all member of entity set. The customer entity set might have attributes like ID, Name, Address etc. similarly goods entity set might have attributes like ID, type, price, cost etc. The notation of attributes in E-R diagram is **elliptical** shape.



Attributes Types:

i. **Simple and Composite Attributes:** Simple attributes are those attributes which can't be divided into subparts i.e. other attributes. E.g. attribute ID. Composite attributes are those attributes which can be divided into subparts. E.g. an attribute "Name" can be considered as composite attribute consisting of First Name, Middle Name and Last Name.

ii. **Single-Valued and Multi-Valued Attributes:** The attributes which has a single value for a particular entity are called single-valued attributes. E.g. "Social Number" attribute because each person have only on social number. `

The attributes which has a set of values for a particular entity are called Multi-Valued attributes. The notation for Multi-valued Attributes in ERD is double elliptical shape. E.g. “Phone Number” attribute because each person may have zero, one or many phone numbers.

iii. **Derived Attributes:** The value for this type of attributes can be derived from the values of other related attributes. E.g. the attribute “age” is a derived attribute because we can calculate age from another attribute “DOB” and “Current Date”. The value of derived attribute is not stored, but is computed when required. The notation for Derived attribute in ERD is Dashed Elliptical shape.

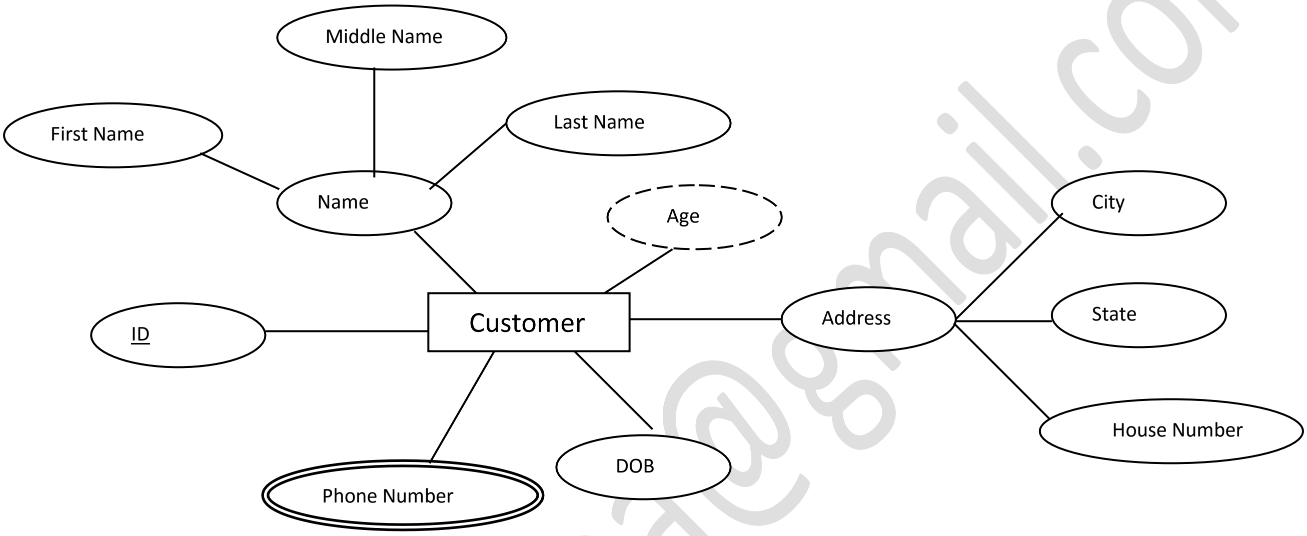
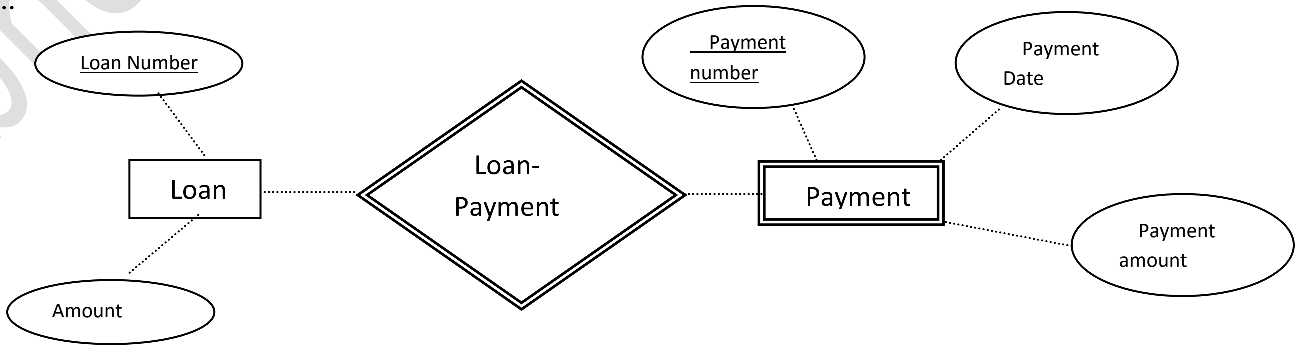


Fig: ERD showing composite, multi-valued and derived attributes

1. **Domain:** For each attributes, there is a set of permitted values called the domain of that attributes. The domain of attribute “Name” for “Customer entity set” might be a set of all text string of a certain length.
2. **Field/Column:** A column represents one related part of a table which hold one piece of information about an item of subject.
3. **Record/Row/Tuple:** A record is a collection of multiple related fields that can be treated as a unit.
4. **Table:** A table is a collection of logically related multiple records. It is a collection of rows and column. A table has fixed number of column but any number of rows.
5. **Weak-Strong Entity Set:** An entity set that doesn’t have primary key is referred to as a weak entity set. The existence of a weak entity set depends on the existence of a strong entity set.The notation for weak entity (Dependent entity) is double lined rectangle. And strong/weak relationship is represented by Double Diamond.
   If an entity set has primary key then it is a strong entity (independent entity) set.
   E.g.:



6. **Relationship Set:** A relationship set is set of relationship of same type. Consider the two entity sets instructor and student in below figure. In ERD, relationship set is represented by a diamond with a name inside. We define the relationship set **advisor** to denote the association between instructors and students. Figure below depicts this association.

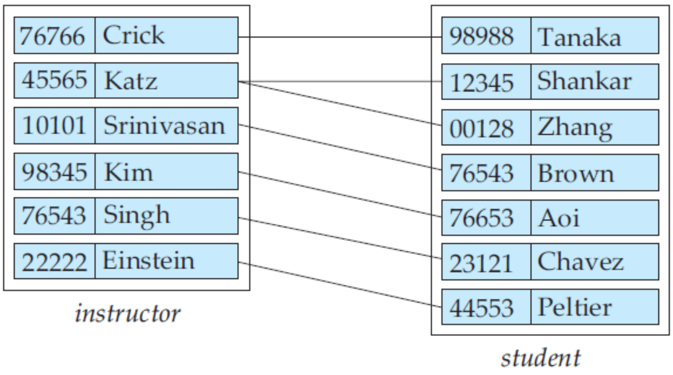
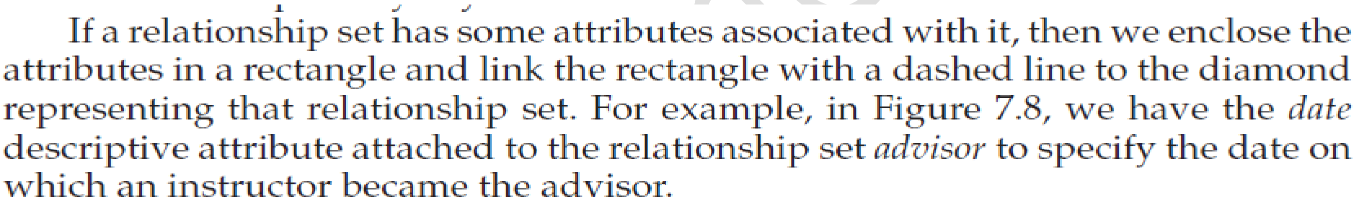


Figure 7.2 Relationship set *advisor*.

7. **Relationship:**A relationship is an association that exists between entities and may be looked as a mapping between two or more entity set.
Example: In the above figure, we can define a relationship advisor that associates instructor Katz with student Shankar. This relationship specifies that Katz is an advisor to student Shankar.



If a relationship set has some attributes associated with it, then we enclose the attributes in a rectangle and link the rectangle with a dashed line to the diamond representing that relationship set. For example, in Figure 7.8, we have the *date* descriptive attribute attached to the relationship set *advisor* to specify the date on which an instructor became the advisor.

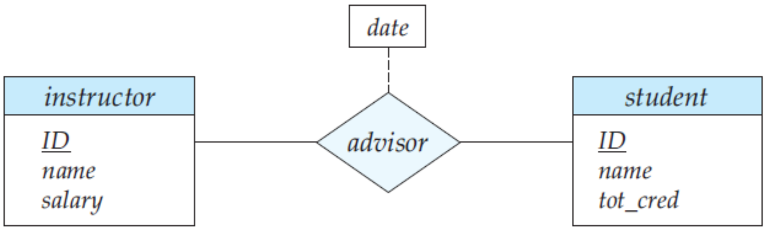
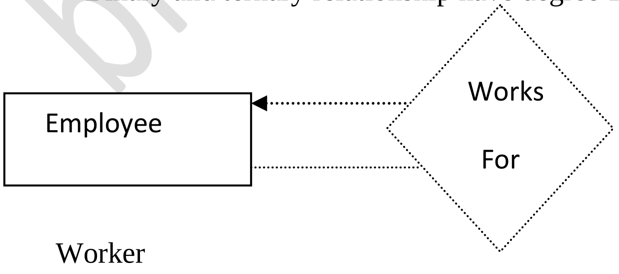


Figure 7.8 E-R diagram with an attribute attached to a relationship set.

1. **Degree of Relationship:** The degree of relationship is the number of entities associated in the relationship. Unary, Binary and ternary relationship have degree 1, 2 and 3 respectively.



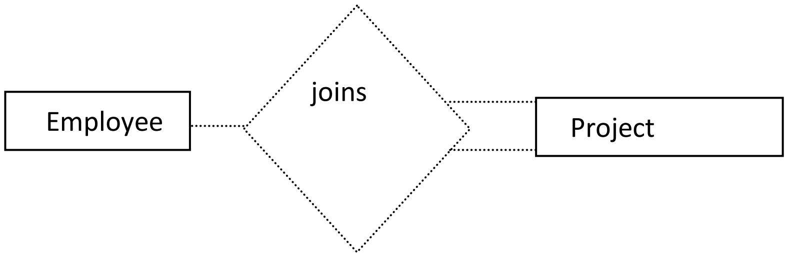


Fig 1: Unary/Recursive Relationship type

fig 2: Binary relationship type

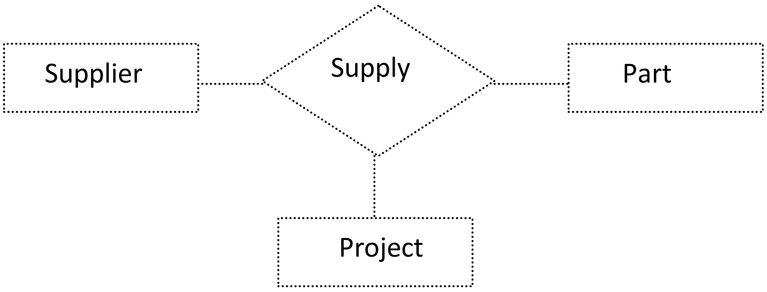
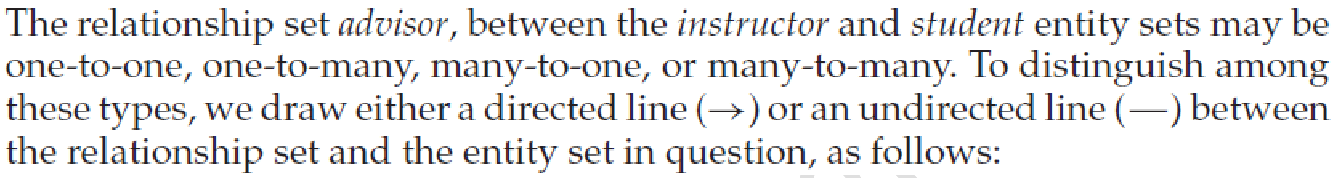
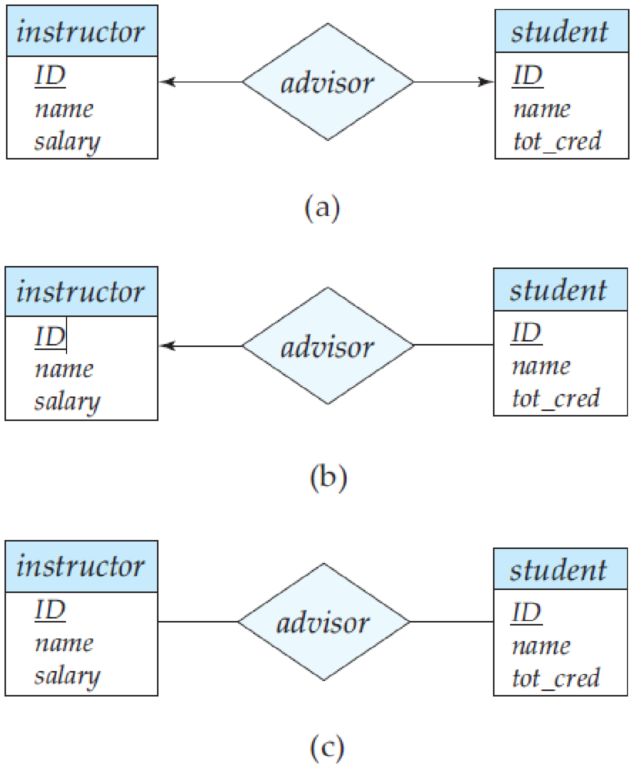


Fig 3: Ternary Relationship type

2. **Mapping Cardinality:** It describes the maximum number of entities that a given entity can be associated by a relationship.



The relationship set *advisor*, between the *instructor* and *student* entity sets may be one-to-one, one-to-many, many-to-one, or many-to-many. To distinguish among these types, we draw either a directed line (→) or an undirected line (—) between the relationship set and the entity set in question, as follows:



(a)

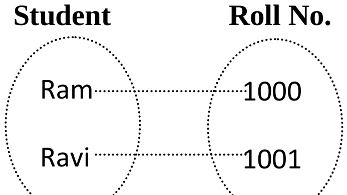
(b)

(c)

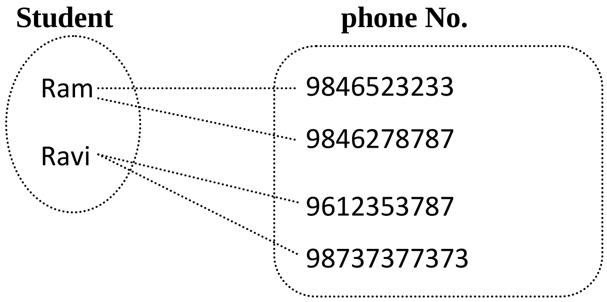
Relationships. (a) One-to-one. (b) One-to-many. (c) Many-to-many.

The cardinality constraints for binary relationship are:

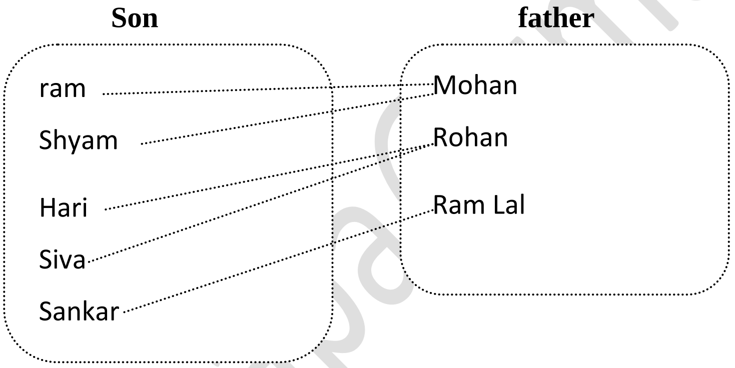
i. **One to One :** An entity in A is associated with almost one entity in B, and an entity in B is associated with almost one entity in A .E.g. drive and car



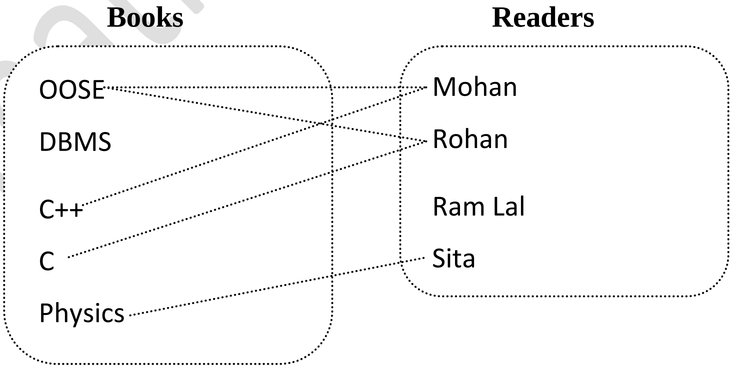
ii. **One to many:** An entity in A is associated with single or multiple entities in B. An entity in B is associated with at most one entity in A. E.g. Teacher and Students



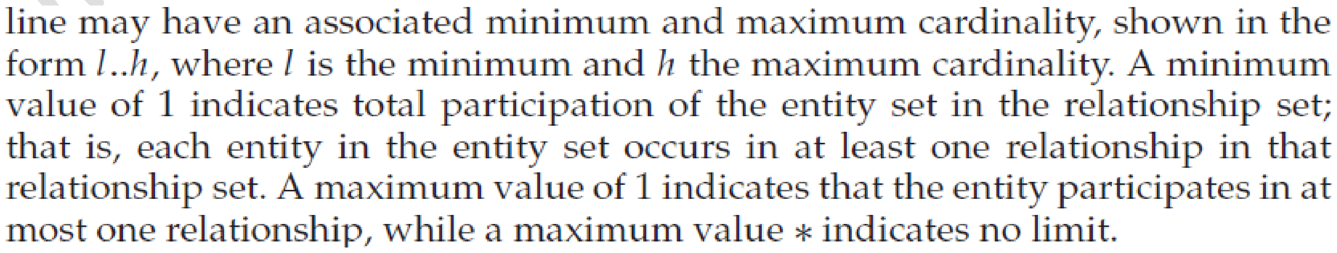
iii. **Many to One:** An entity A is associated with at most one entity in B. An entity in B can be associated with single or multiple entities in A. E.g. Students and teachers



iv. **Many to Many:** An entity in A is associated with single or multiple entities in B and an entity in B is associated with single or multiple entities in A. E.g. Books and Readers



A


line may have an associated minimum and maximum cardinality, shown in the form $l..h$, where l is the minimum and h the maximum cardinality. A minimum value of 1 indicates total participation of the entity set in the relationship set; that is, each entity in the entity set occurs in at least one relationship in that relationship set. A maximum value of 1 indicates that the entity participates in at most one relationship, while a maximum value $*$ indicates no limit.

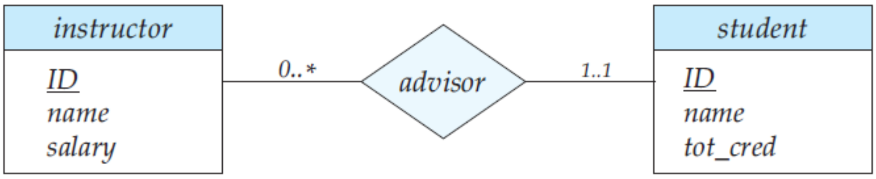


Figure 7.10 Cardinality limits on relationship sets.

For example, consider Figure 7.10. The line between *advisor* and *student* has a cardinality constraint of 1..1, meaning the minimum and the maximum cardinality are both 1. That is, each student must have exactly one advisor. The limit 0..∗ on the line between *advisor* and *instructor* indicates that an instructor can have zero or more students. Thus, the relationship *advisor* is one-to-many from *instructor* to *student*, and further the participation of *student* in *advisor* is total, implying that a student must have an advisor.

3. **Participation:** The participation constraints specify whether existence of an entity depends on its being related to another entity via the relationship type. Participation may be
   - **i.** Total: Every entity in the entity set participatesin at least one relationship in the relationship set.
   - **ii.** Partial: Some entity may not participate in any relationship in the relationship set.

   The notation for total participation is double line.

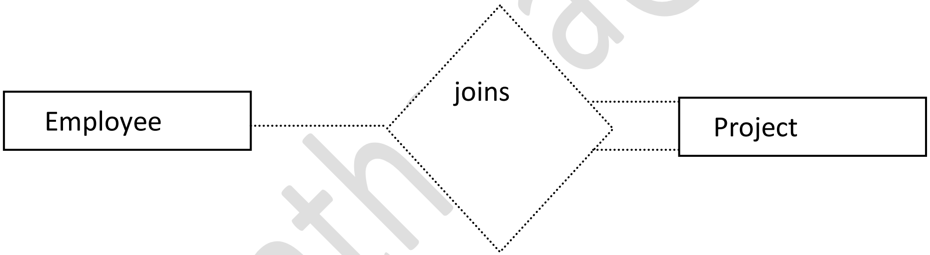
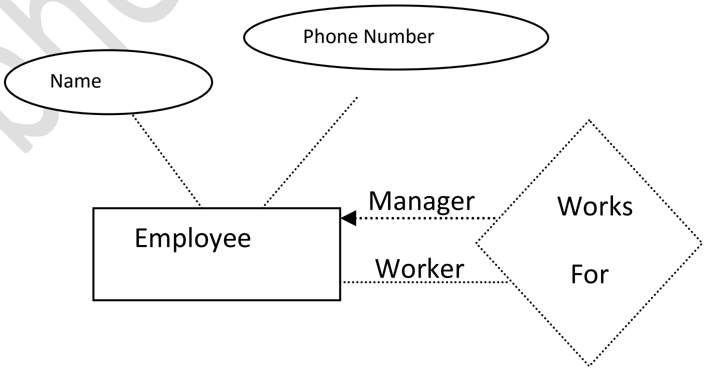


Fig: Participation in relationship notation

In the above ERD, every project has at least one employee joined in it. So Participation of entity "project " in "joins" relationship is total. It also shows that, not every employee in the company joins in a project.

8. **Roles:** The functions that an entity plays in a relationship is called role. Roles are indicated in E-R diagram by labeling the lines that connect diamonds to rectangles. The positions i.e. manager and worker are called the roles. In the below ERD, it specifies how "employee" entities interact via "works for" relationship set. Role label are optional and are used to clarify the structure of relationship.



Advantage of ER model

Advantage of ER model

1. Easy and simple to understand with minimal training.
2. It is possible to find connection from one node to all other nodes.
3. It has explicit linkages of entities.

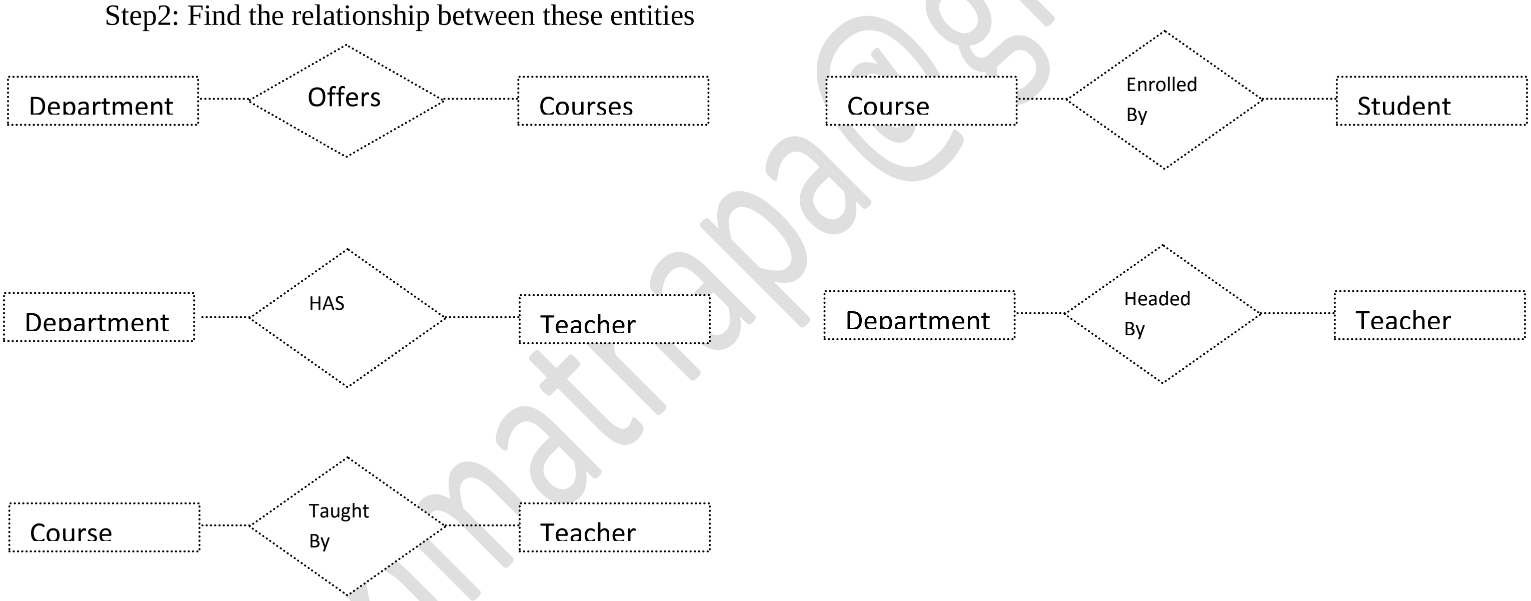
Disadvantages of ER model

1. Limited relationship representation.
2. Limited constraint representation.
3. No representation of data manipulation.

Solved ERD

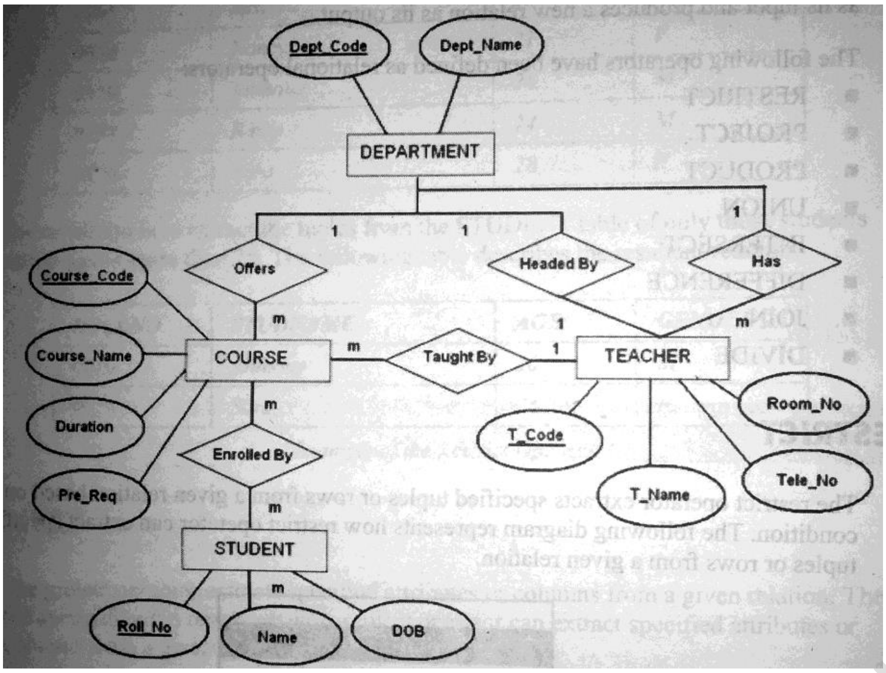
1. Draw ERD for your college database with as set of department, courses, teachers and students.
   Step1: Identify the entities
   - Here there are four entities named department, courses, teacher, and student.

Step2: Find the relationship between these entities

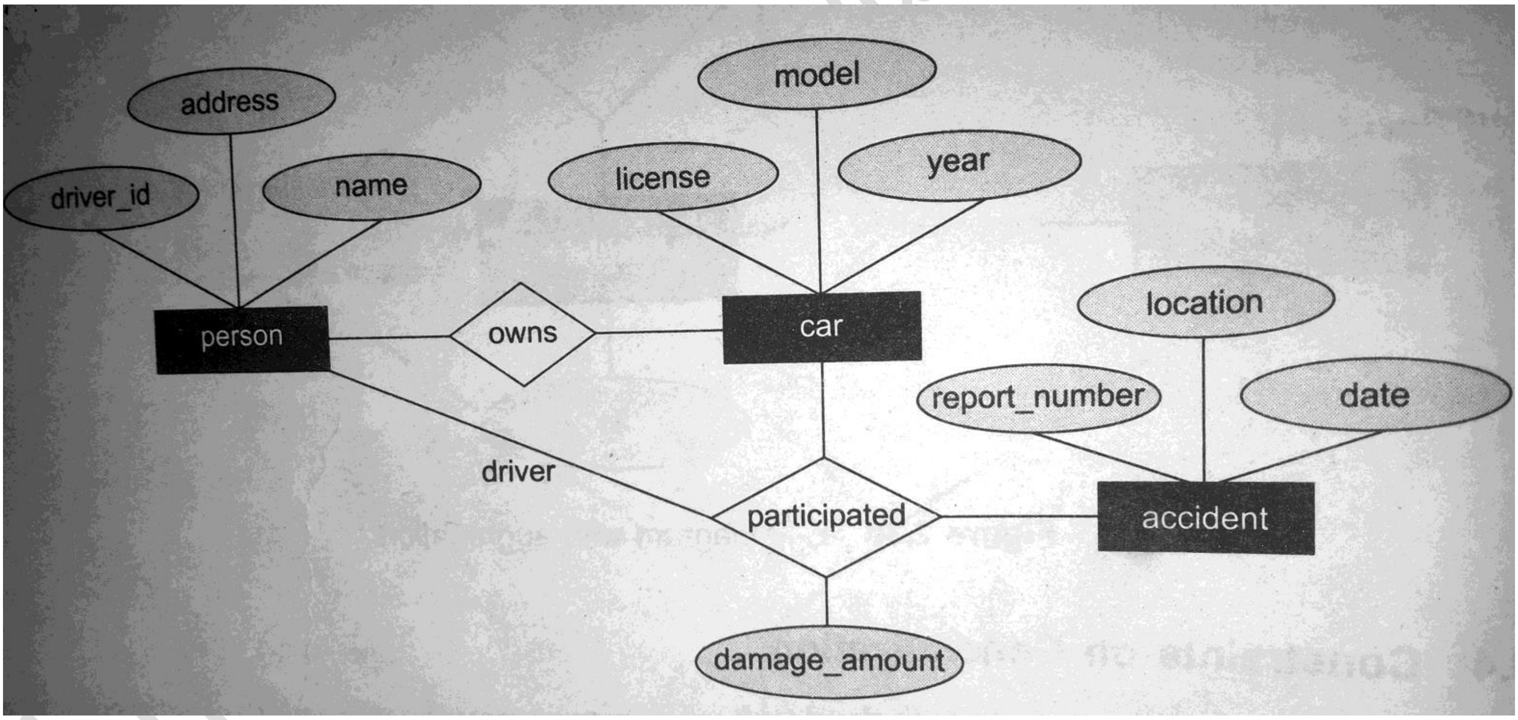


Step3: Identify the attributes for each entity.
For Department:Department code(Dept_code), Department Name (Dept_Name)
For Course: Course_Code, Course_Name, Duration, Prerequisite (Pre-Req)
For Student: RollNo., Student Name(Name), Date of birth (DOB)
For Teacher: Teachers code (T_Code), Teacher Name (T_name), Telephone Number (Tele_No)

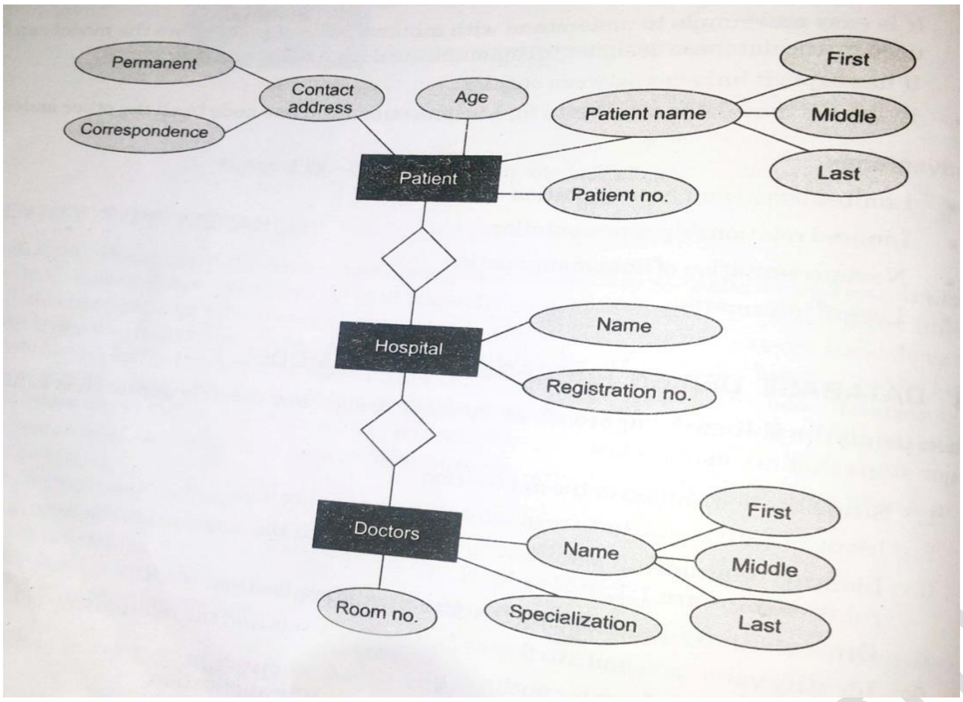
Step4: Now draw ERD with the above information



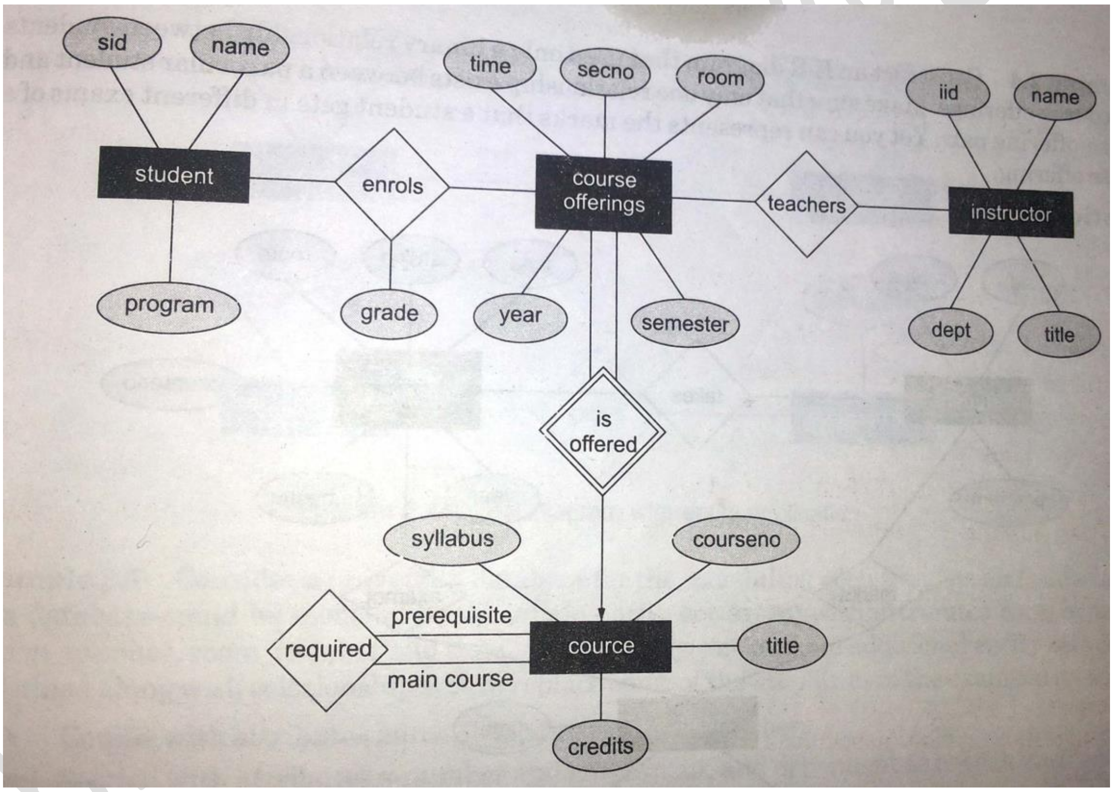
2. Construct ERD for a car insurance company whose customer owns own one or more cars.



3. Construct ED with a set of patient and medical doctors.

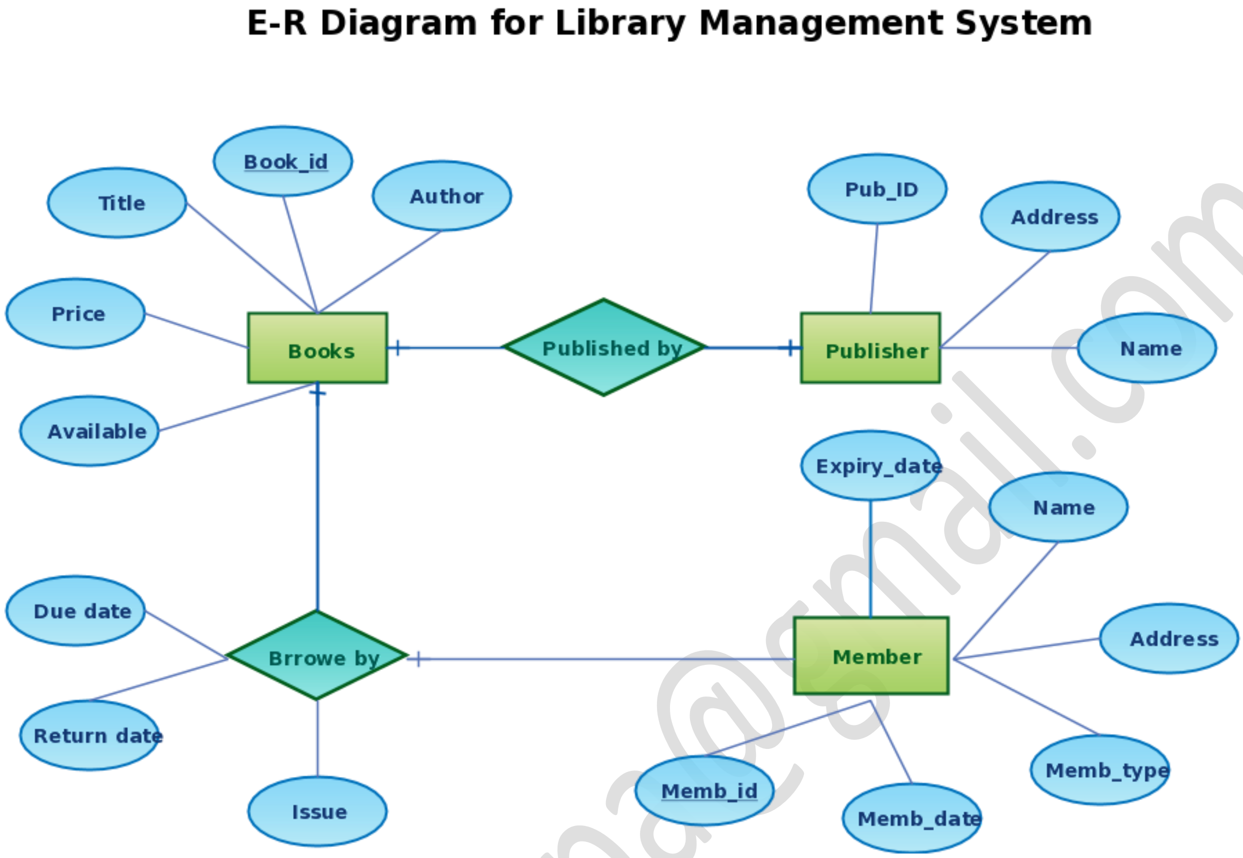


4. Construct ERD for EEMC Exam Section database.

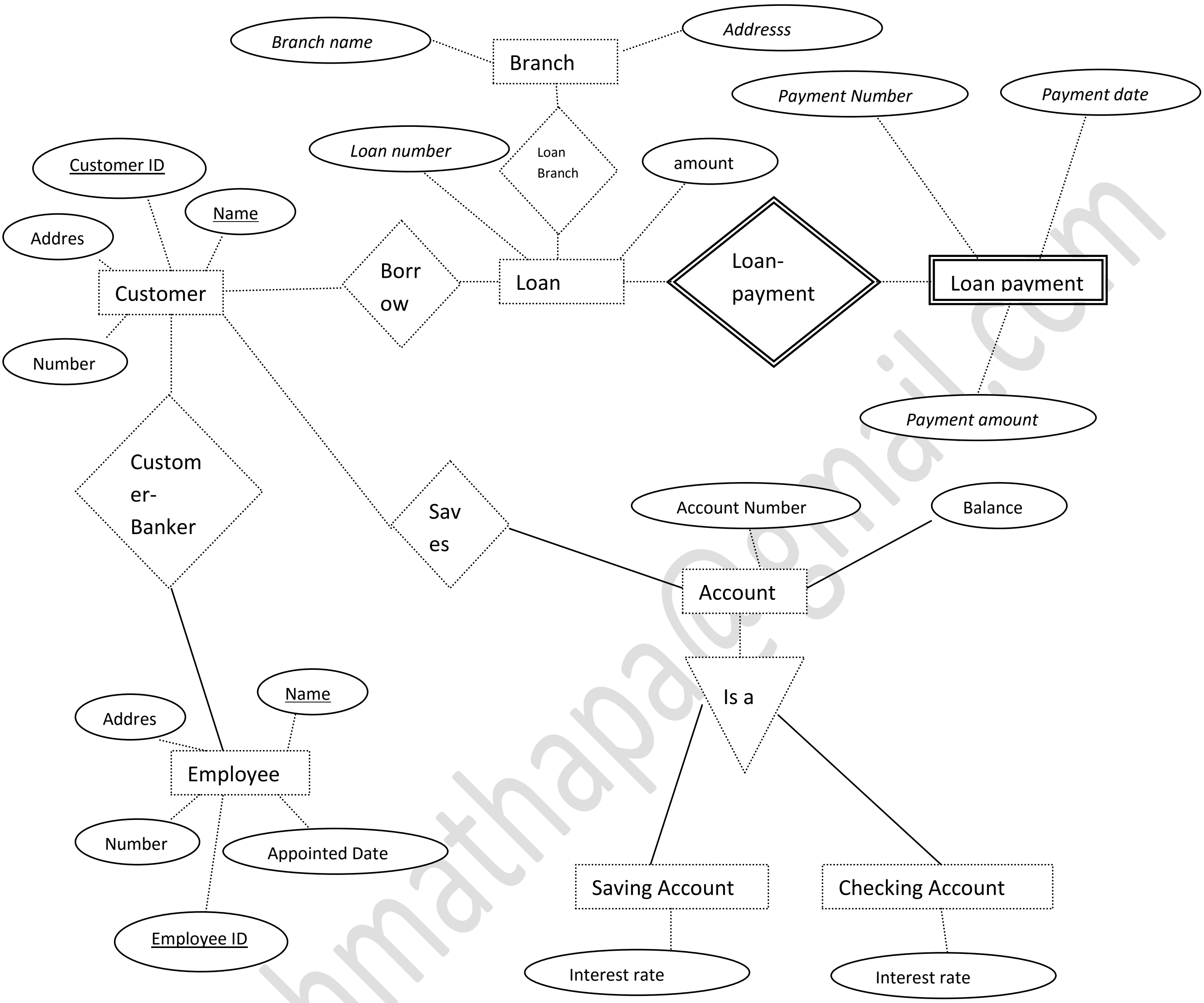


Note: the entity set course offerings is weak entity so it must be represented by double line rectangle.

ER Diagram OF library Management System



5. Construct ERD for Banking System.



Construct E-R diagram for the university enterprise.

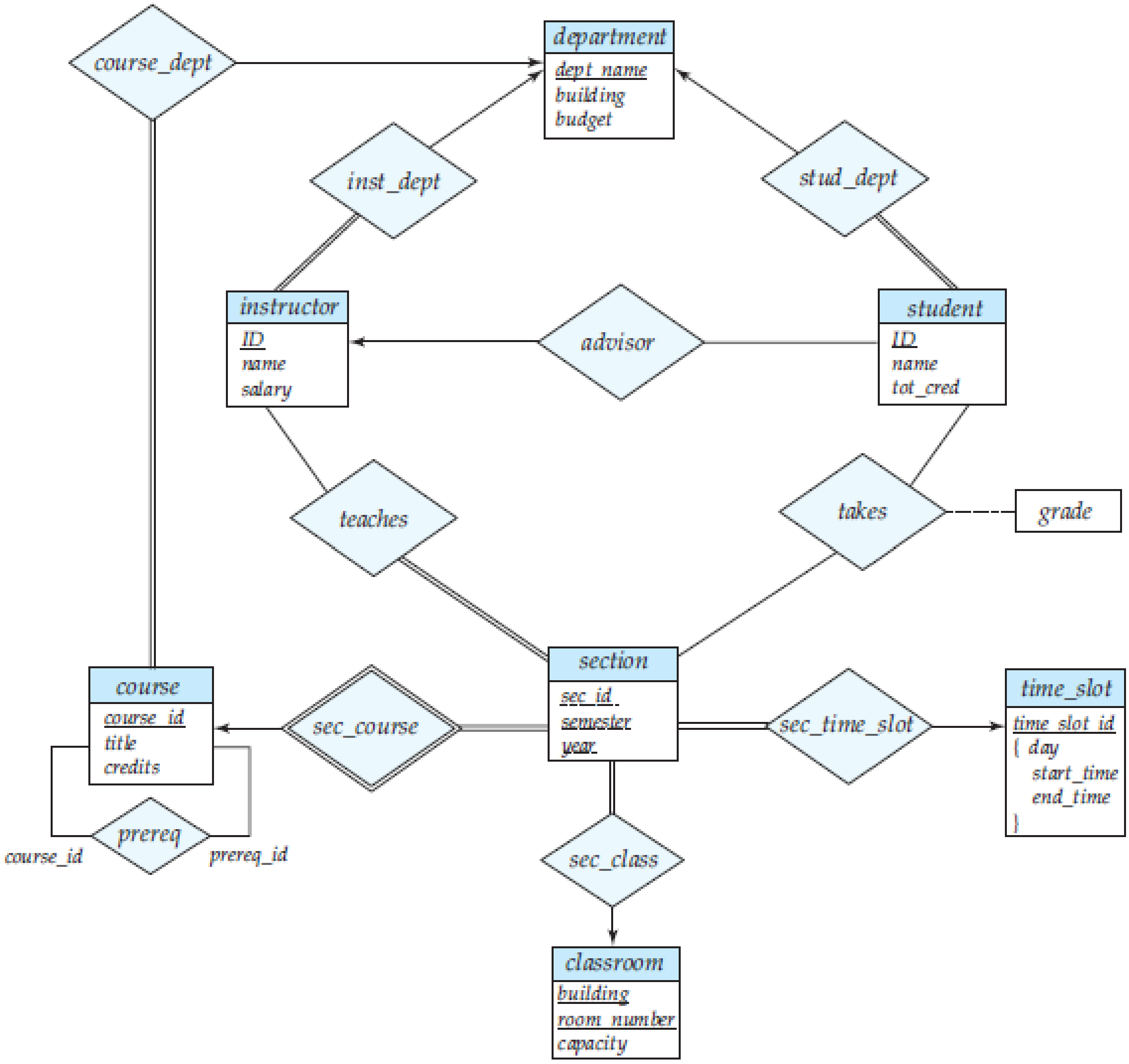
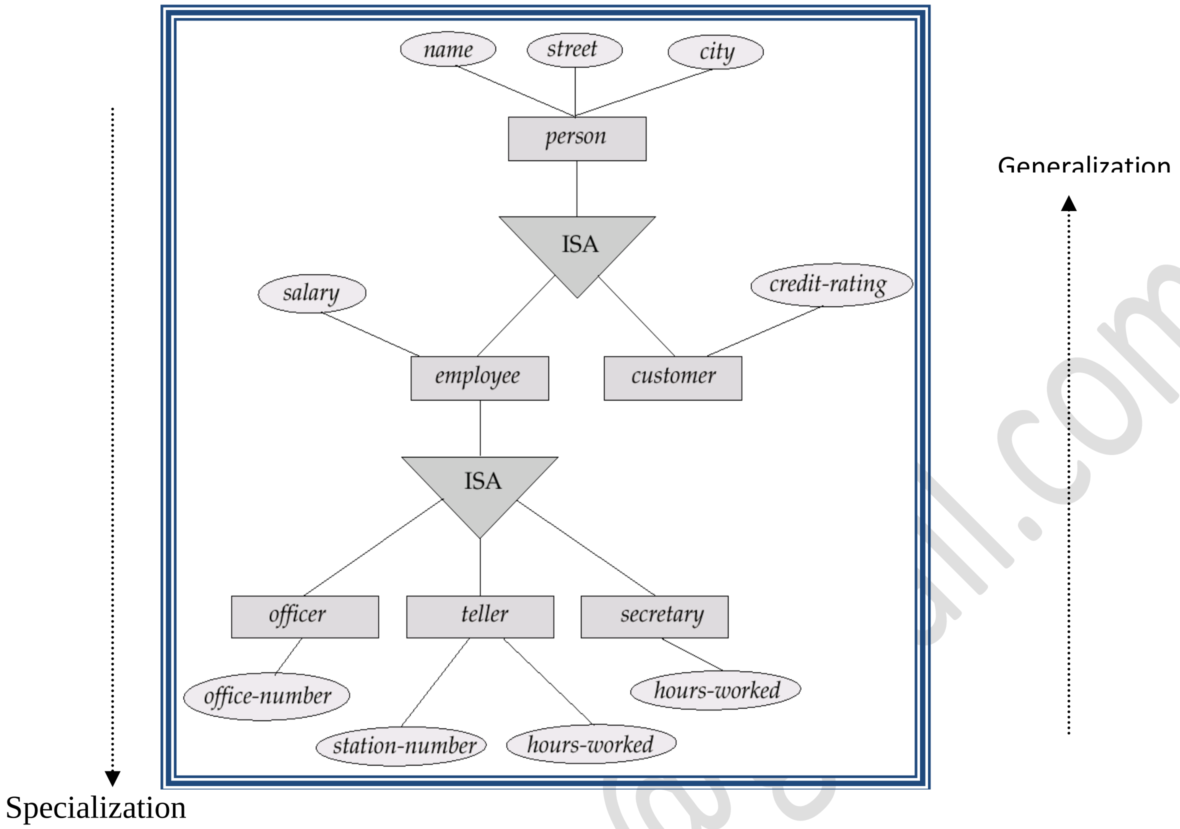


Figure 7.15 E-R diagram for a university enterprise.

Extender ER features: Specialization (Categorization), Generalization, and Aggregation

1. **Specialization:** The process of sub grouping a higher level entity set into multiple lower level entity set is called specialization. It is a top down design process. These sub grouping become lower level entity sets that have attributes or participate in relationships that do not apply to higher level higher level entity set. A lower level entity set inherits all the attributes and relationship participation of the higher level entity set to which it is linked. This is called as attribute inheritance. In ERD specialization is represented by triangle symbol labeled by ISA. The ISA relationship is also referred to as a superclass –subclass relationship.

2. **Generalization:** It is the process of taking union of two or more lower level entity set to produce a higher level entity set. It is a bottom-up design process because we combine all the entities sets that share same features into higher level entity. Specialization and Generalization are simple inversion of each other. They are represented in ERD by the same way.
The ISA relationship is also called as superclass-subclass relationship.



3. **Aggregation:** Aggregation is a technique to express relationship among relationship. Aggregation is an abstraction through which relationship are treated as higher level entities.
   Consider the following ERD (Fig I) with ternary relationship

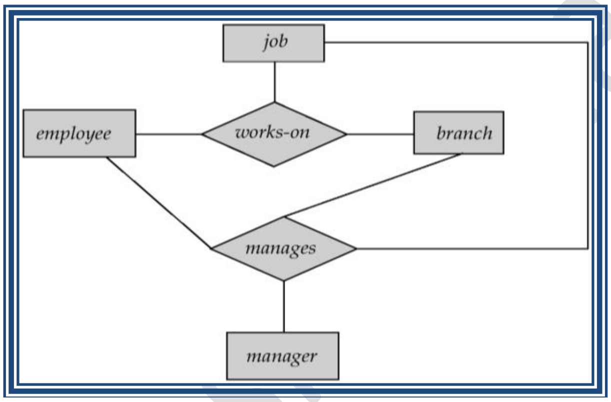


Fig I

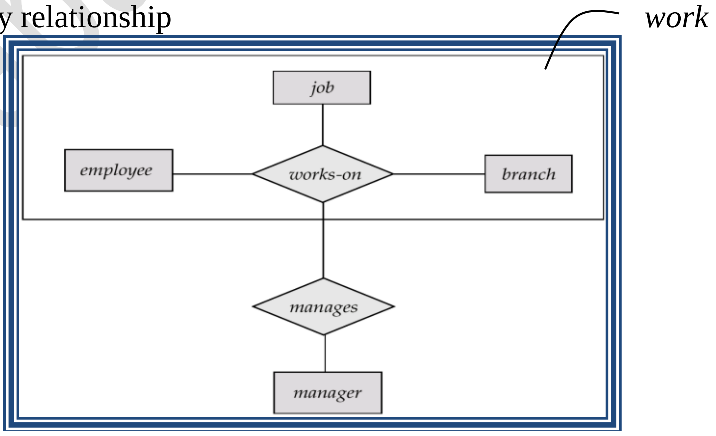


Fig II

In the above ERD (Fig I) the relationship set “manages” and “works-on” represent overlapping information. Every “manages” relationship corresponds to “works-on” relationship but some “works-on” relationship may not correspond to any “manages” relationship so we can’t discard “works-on” relationship. This problem of redundancy can be eliminated via aggregation i.e. treat relationship as higher level entity.
So, using aggregation, we treat the relationship set “works-on” and the entity set “employee” and “branch” as higher level entity set called “work”. Fig II shows ERD with aggregation and without redundancy.

Another example:

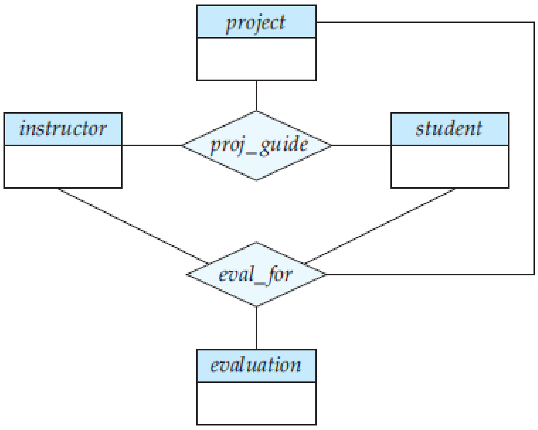


Figure 7.22 E-R diagram with redundant relationships.

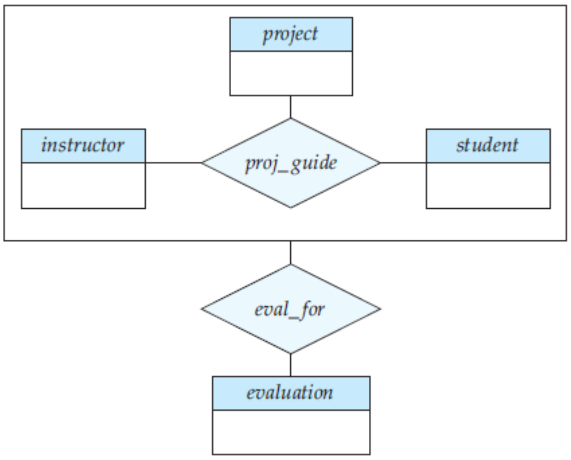


Figure 7.23 E-R diagram with aggregation.

Keys

A key is an important constraint which is a group of one or more attributes that uniquely identify an entity in the entity set.
A key is a column value in a table that is used to uniquely identify a row of data in a table or establish a relationship with another table. There are six types of keys.

1. Super Key: a set of one or more attributes which taken collectively allows to uniquely identify entity in entity set.
   E.g. for entity set "Student"={roll no, class, name, address, dob, year}
   Above entity set has many super keys. Some of them are {roll no, class, name, address}, {roll no, class, Address, dob}, {roll no, class, name, dob, year} etc.
2. Candidate Key: The smallest possible super key that uniquely identifies record in a table is called candidate key for primary key. Super key may have redundancy but candidate doesn't have.
   One of the candidate key for above entity set are {roll no, class}.

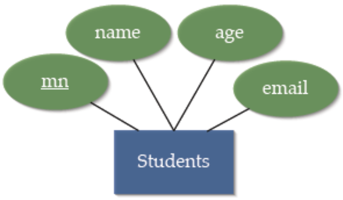
3. Primary Key: Primary key is a key which uniquely identify records in a table. The primary key field doesn't accept redundant data. Any of the candidate key can be used as primary key.
4. Alternate Key: Any attribute that is a candidate for primary key but is not used as the primary key is called the alternate key.
5. Composite Key: When a key that uniquely identifies the records of the table is made up of more than one attributes, it is called composite key.
6. Foreign Key: A foreign key is a linking pin between two tables. The key connects to another table when a relationship is being established. A foreign key is a copy of a primary key in another table.

Reduction E-R diagram to Table

We can represent a database that conforms to an E-R database schema by a collection of relation schemas. For each entity set and for each relationship set in the database design, there is a unique relation schema to which we assign the name
of the corresponding entity set or relationship set.

1. **Representation of Strong Entity Sets with Simple Attributes**

Let E be a strong entity set with only simple descriptive attributes a1, a2, . . . , an. We represent this entity by a schema called E with n distinct attributes. Each tuple in a relation on this schema corresponds to one entity of the entity set E.
Example:
consider the entity set students of the E-R diagram in Figure below. This entity set has four attributes: mn, name, age and email. We represent this entity set by a schema called students with four attributes:



Students(<u>MN, name, age email</u>)

2. Representation of Strong Entity Sets with Complex Attributes

When the entity set consists of complex attributes such as composite, derived and multivalued, things are a bit more complex.

1. For the composite attribute *name*, the schema generated for customer contains the attributes first name, middle name, and last name; there is no separate attribute or schema for name.
2. Multivalued attributes generally map directly into attributes for the appropriate relation schemas.
3. Derived attributes are not represented explicitely in relational model.

Example:

Consider the entity set customer of the E-R diagram in figure below. Here we see composite attributes "name" and address, multivalued attribute "phone number" and derived attribute "Age". We represent this entity set by a two schema as below.

Customer(<u>ID</u>, First Name, Middle Name, Last Name, City, State, House Number, DOB)
Customer_Phone(<u>ID</u>, <u>Phone Number</u>)

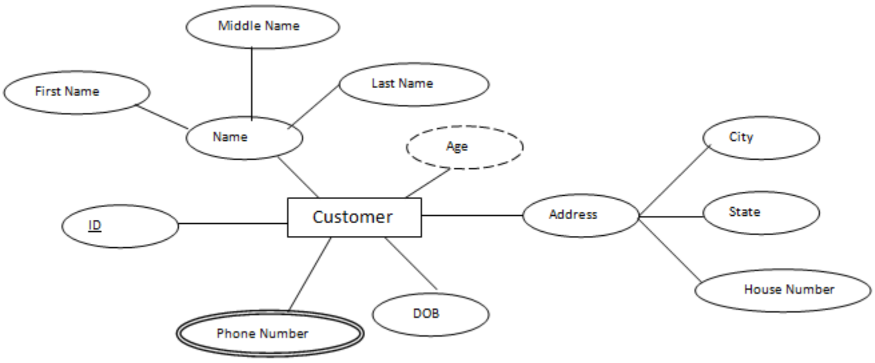
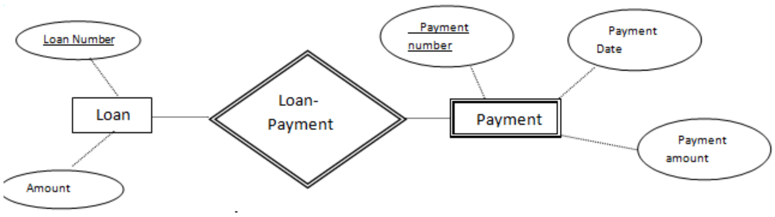


Fig: ERD showing composite, multi-valued and derived attributes

3. Representation of Weak Entity Sets

Weak entity types are converted into a table of their own, with the primary key of the strong entity acting as a foreign key in the table. This foreign key along with the key of the weak entity form the composite primary key of this table. The entitities in the weak entity set must be automatically deleted (cascade deleted) for which there are no owners in strong entity set.



We represent this entity set by a two schema as below.

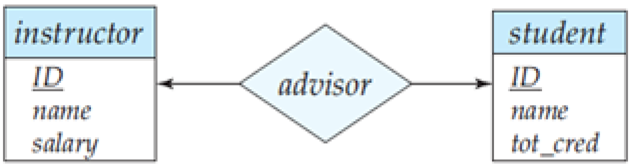
Loan(<u>Loan Number</u>, Amount)
Payment(<u>Loan Number, Payment Number</u>, Payment Date, Payment Amount)

Note: foreign key "Loan Number" references primary key " Loan number" in Loan

4. **Representation of Relationship Set**

1. Binary one to one Relationship

   The primary key of either of the participants can become a foreign key in the other



We represent these entity set by a two schema as below.

Way1:

Student(Student.ID, name, tot_cred, instructor.ID)

Instructor(Instructor.ID, name, salary)

Note: here in relational schema Student, instructor.ID_is foreign key that references primary key instructor.ID in instructor.

Way2:

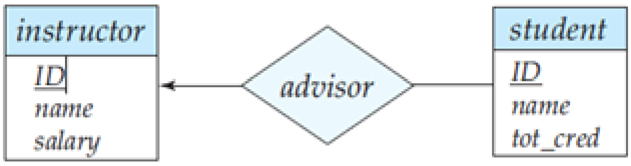
Instructor(instructor.ID**,name,salary, student.ID)**

Student(student.ID, name, tot_cred)

Note: here in relational schema Instructor, student.ID is a foreign key that references primary key student.ID in student.

2. **Binary one to many or many to one relationship**

   The primary key of the relation on the "one" side of the relationship becomes a foreign key in the relation on the "many" side



We represent these entity set by a two schema as below.

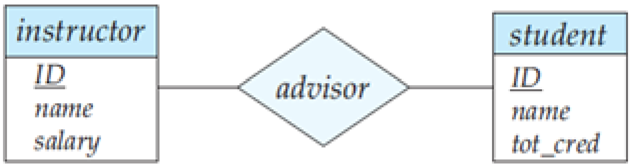
Instructor(ID, name, salary)

Student(Student.ID, name, tot_cred, instructor.ID)

*Note: in relational schema Student, the foreign key instructo***.***ID references primary key ID in instructor*

3. **Binary Many to many relationship**
   - A new table is created to represent the relationship set.
   - Contains two foreign keys - one from each of the participants in the relationship
   - The primary key of the new table is the combination of the two foreign keys

Example:



We represent these entity set by a three schema as below.

Instructor(ID, Name, salary)

Student(ID, name,tot_cred)

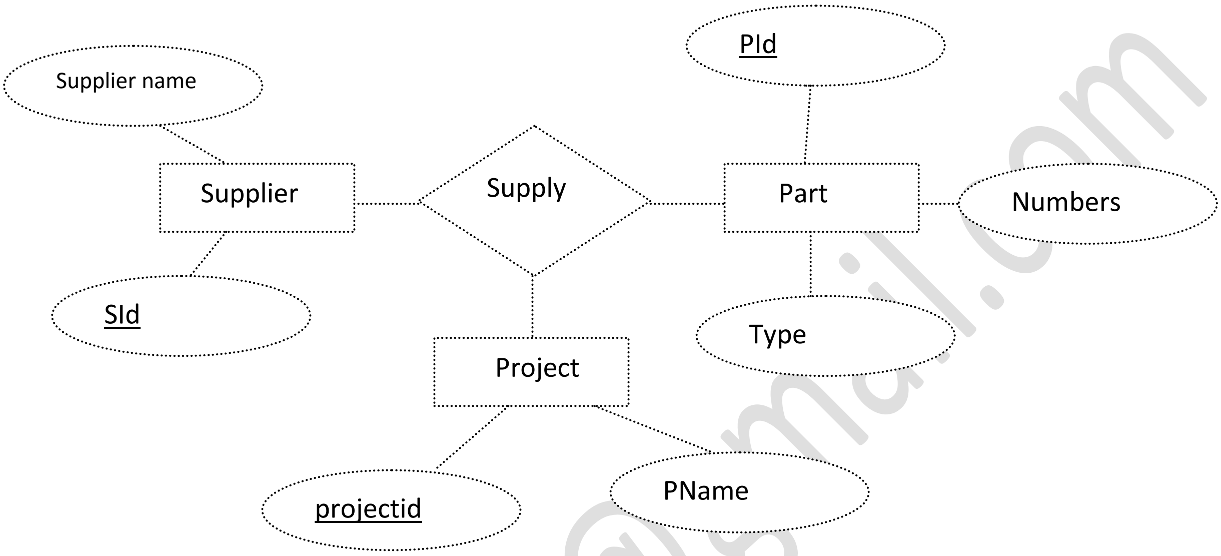
Advisor(instructor.ID , student.ID)

Note: here in relational schema Advisor, foreign key instructo.ID references primary key ID in instructor, foreign key student.ID references primry key ID in student and combination of these two act as primary key in schema Advisor.

Question:

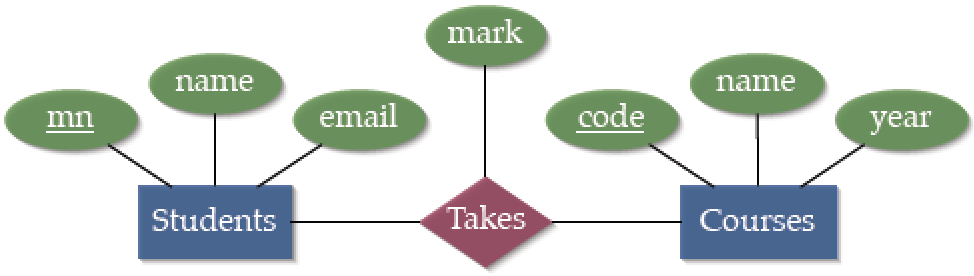
Convert the following E-R diagram to equivalent Relational schema.

1.



Supplier(sid, Supplier name)
Project(projectid, Pname)
Part(Pid,Type,Number)
Supply(sid, projected, Pid)

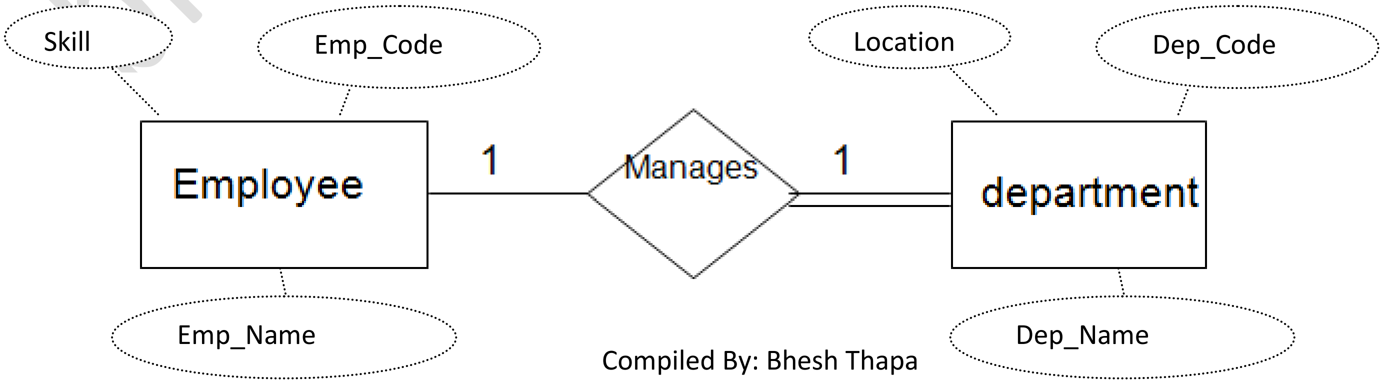
2.



Student(mn , name , email)
Courses(code , name , year)
Takes(mn,code,mark)

Note:In Take schema, foreign key mn references primary key mn in student while foreign key code references primary key code in courses

3.



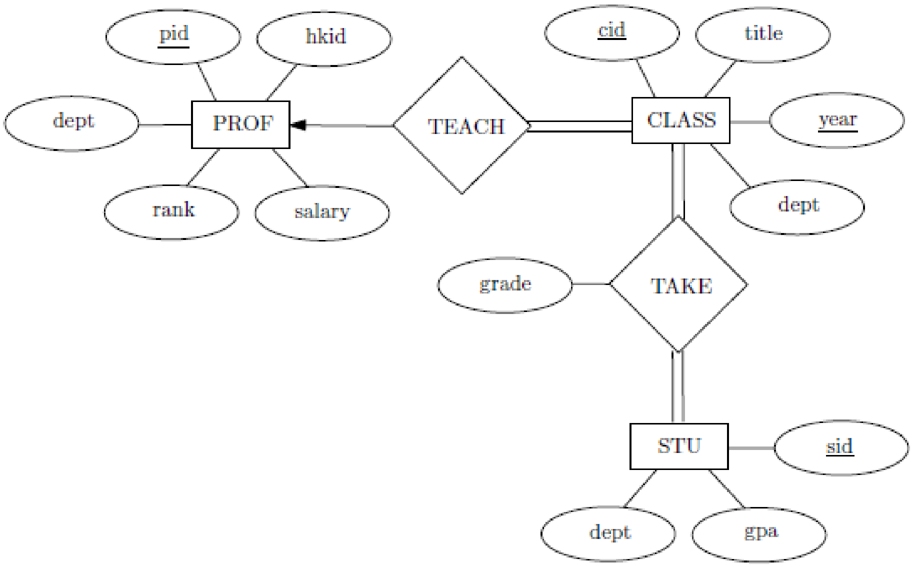
Note: The primary key of the partial participant will become the foreign key of the total participant
So the equivalent relational schema are given below

Employee(Emp_code, Skill, Emp_Name)

Department(Dep_Code, Dep_Name, Location, mgr_Emp_code)

Note: mgr_emp_code references Emp_code in Employee

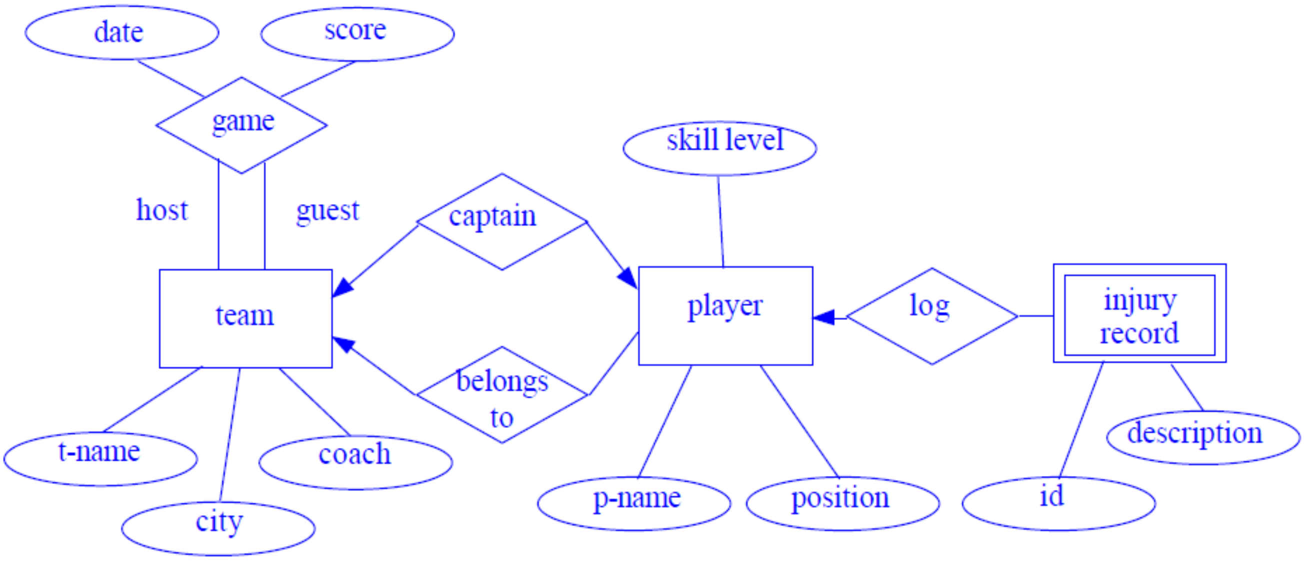
4.



PROF(pid , hkid , dept , rank , salary) note primary key is pid
CLASS(cid, year, title, dept pid) ***note foreign key pid references primary key pid in prof, primary key is(cid+year)***
STU(sid, dept, gpa) note : primary key is sid
TAKE(cid, year, sid, grade) : note primary key is (cid+year +sid)

5. Suppose you are given the following requirements for a simple database for the National Cricket League (NCL):
   - the NCL has many teams,
   - each team has a name, a city, a coach, a captain, and a set of players,
   - each player belongs to only one team,
   - each player has a name, a position (such as *left bowler* or *batsman*), a skill level, and a set of injury records,
   - a team captain is also a player,
   - a game is played between two teams (referred to as host_team and guest_team) and has a date (such as *May 11th, 1999*) and a score (such as *4 to 2*).

   Construct a clean and concise ER diagram for the NHL database using the Chen notation as in your textbook. List your assumptions and clearly indicate the cardinality mappings as well as any role indicators in your ER diagram.



***UML Class Diagram**

UML stands for unified modeling language. It is a graphical language for modeling a system structure and its behavior. UML is not any programming language, it is a set of diagrams that can be used to specify, construct, visualize and document the system design.

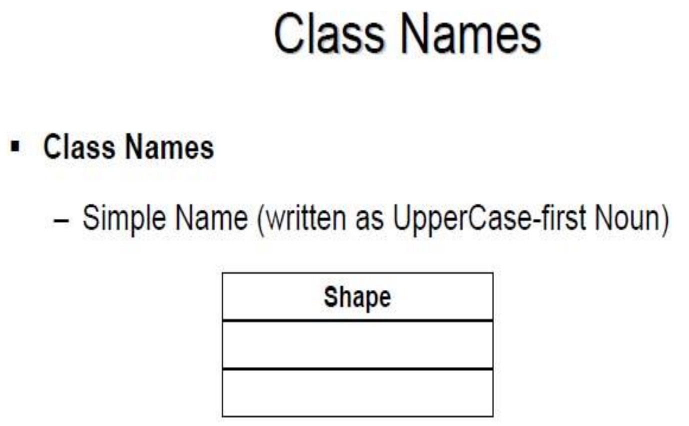
Class diagram is one of the UML diagrams which are used to show the structure of the system. These diagrams are composed of classes and their relationships. A UML class describes a set of objects that share the same attributes, operations, and relationship. Class diagrams may specify both the conceptual **[what]** and implementation **[how]** details of the system. Class diagrams represent structural and **not** behavioral relationships that exist among system entities

The following points should be remembered while drawing a class diagram:

- The name of the class diagram should be meaningful to describe the aspect of the system.
- Each element and their relationships should be identified in advance.
- Responsibility (attributes and methods) of each class should be clearly identified.
- For each class minimum number of properties should be specified. Because unnecessary properties will make the diagram complicated.

Basic Terms:

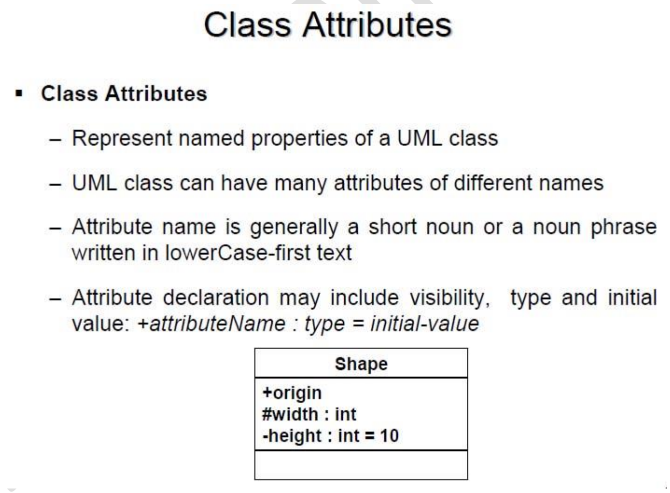
1. Class represented by rectangle.
2. A relationship between two classes is drawn by a line.



Class Names

- **Class Names**
  - Simple Name (written as UpperCase-first Noun)

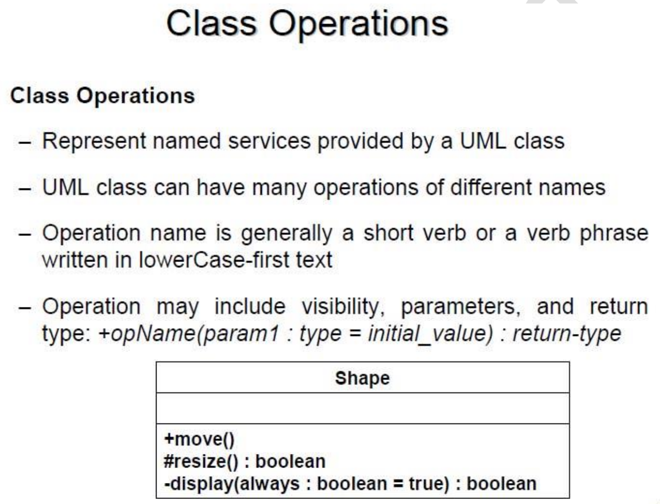
Fig1



Class Attributes

- **Class Attributes**
  - Represent named properties of a UML class
  - UML class can have many attributes of different names
  - Attribute name is generally a short noun or a noun phrase written in lowerCase-first text
  - Attribute declaration may include visibility, type and initial value: *+attributeName : type = initial-value*

Fig 2

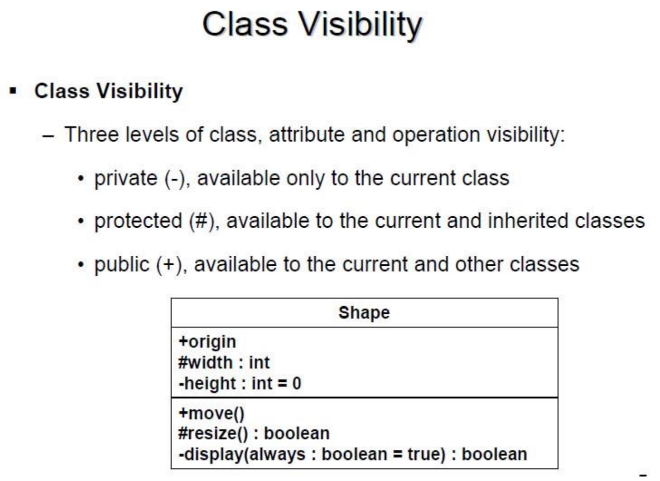


Class Operations

Class Operations

- Represent named services provided by a UML class
- UML class can have many operations of different names
- Operation name is generally a short verb or a verb phrase written in lowerCase-first text
- Operation may include visibility, parameters, and return type: *+opName(param1 : type = initial_value) : return-type*

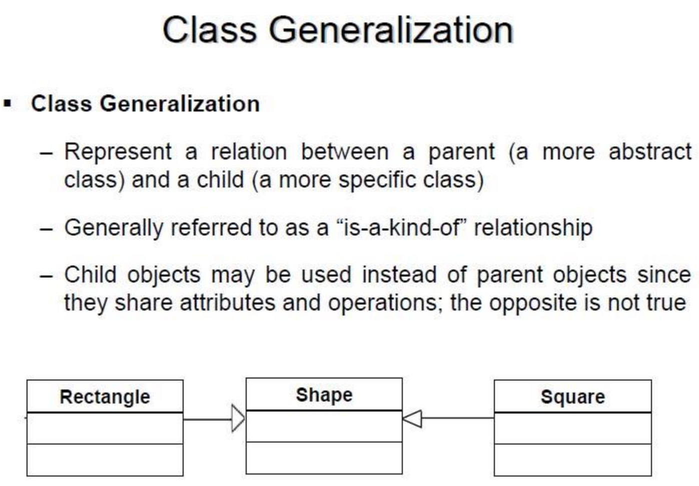
Fig 3



Class Visibility

- **Class Visibility**
  - Three levels of class, attribute and operation visibility:
    - private (-), available only to the current class
    - protected (#), available to the current and inherited classes
    - public (+), available to the current and other classes

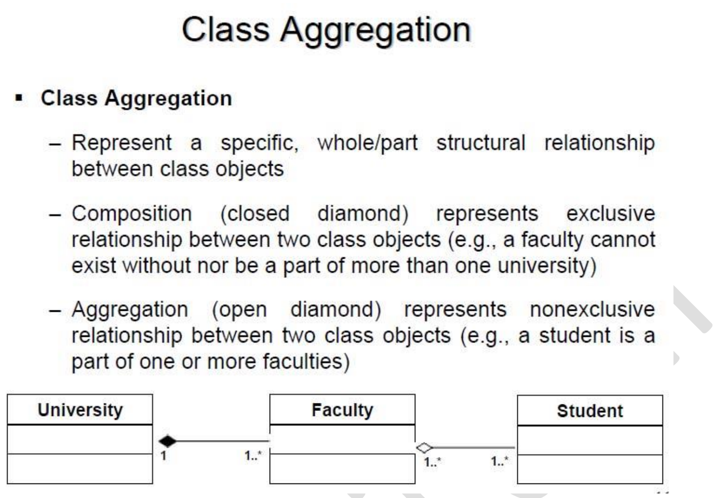
Fig4



Class Generalization

- **Class Generalization**
  - Represent a relation between a parent (a more abstract class) and a child (a more specific class)
  - Generally referred to as a "is-a-kind-of" relationship
  - Child objects may be used instead of parent objects since they share attributes and operations; the opposite is not true

Fig 5



Class Aggregation

- **Class Aggregation**
  - Represent a specific, whole/part structural relationship between class objects
  - Composition (closed diamond) represents exclusive relationship between two class objects (e.g., a faculty cannot exist without nor be a part of more than one university)
  - Aggregation (open diamond) represents nonexclusive relationship between two class objects (e.g., a student is a part of one or more faculties)

Fig 6

<u>Association, aggregation and composition in class diagram</u>

An association represents a family of links. Associations are normally represented as a line.

Aggregation is a variant of the "has a" association relationship; aggregation is more specific than association. It is an association that represents a part-whole or part-of relationship. As a type of association, an aggregation can be named and have the same adornments that an association can. However, an aggregation may not involve more than two classes. In UML, it is graphically represented as a *hollow* diamond shape on the containing class end of the tree with a single line that connects the contained class to the containing class.

Composition is a stronger variant of the "owns a" association relationship; composition is more specific than aggregation .*Composition* usually has a strong *life cycle dependency* between instances of the container class and instances of the contained class(es): If the container is destroyed, normally every instance that it contains is destroyed as well. The UML graphical representation of a composition relationship is a *filled* diamond shape on the containing class end of the tree of lines that connect contained class(es) to the containing class.

Example:

Some 'common sense' examples of aggregations could perhaps include a team being an aggregate of its players, a village of its inhabitants, a company of its workforce or a society and its membership. These are all examples of whole-part, or 'has-a' [4], relationships, and could be modeled using aggregation.

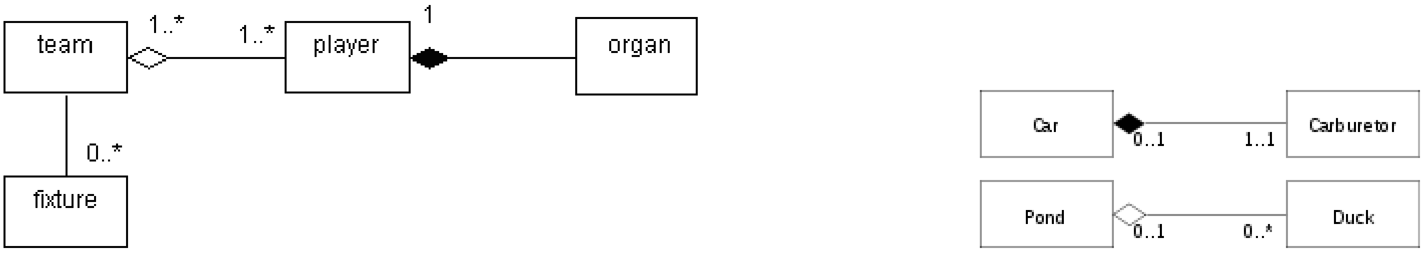


Figure 1: association, aggregation and composition.

Similarly, some common sense examples of compositions could be a person and their limbs, heart, brain, and so forth, a stadium and its pitch and seats, a wall and the bricks from which it is built, and so on. In each of these cases, the composite is composed of its parts. You could consider writing a class diagram for the composite showing the parts as attributes. The attribute notation denotes composition, and so this would seem to make sense.

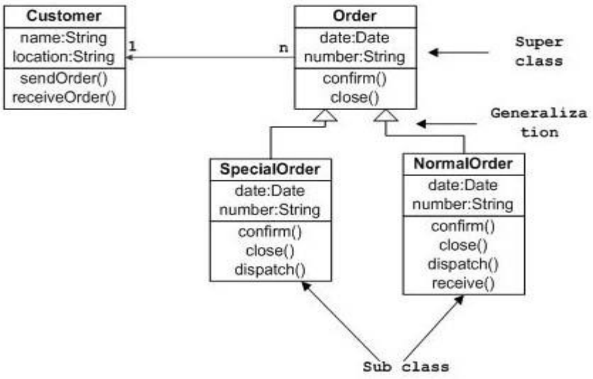
Solved Example:

1. **Class Diagram for order management System**

- First of all *Order* and *Customer* are identified as the two elements of the system and they have a *one to many* relationship because a customer can have multiple orders.

- We would keep *Order* class is an abstract class and it has two concrete classes (inheritance relationship) *SpecialOrder* and *NormalOrder*.
- The two inherited classes have all the properties as the *Order* class. In addition they have additional functions like *dispatch ()* and *receive ()*.

So the following class diagram has been drawn considering all the points mentioned above:



Use of Class Diagrams

Class diagram is a static diagram and it is used to model static view of a system. The static view describes the vocabulary of the system. Class diagram is also considered as the foundation for component and deployment diagrams. Class diagrams are not only used to visualize the static view of the system but they are also used to construct the executable code for forward and reverse engineering of any system. Generally UML diagrams are not directly mapped with any object oriented programming languages but the class diagram is an exception. Class diagram clearly shows the mapping with object oriented languages like Java, C++ etc. So from practical experience class diagram is generally used for construction purpose. So in a brief, class diagrams are used for:

- Describing the static view of the system.
- Showing the collaboration among the elements of the static view.
- Describing the functionalities performed by the system.
- Construction of software applications using object oriented languages.

Assignment-2: Draw E-R Diagram for library management system.