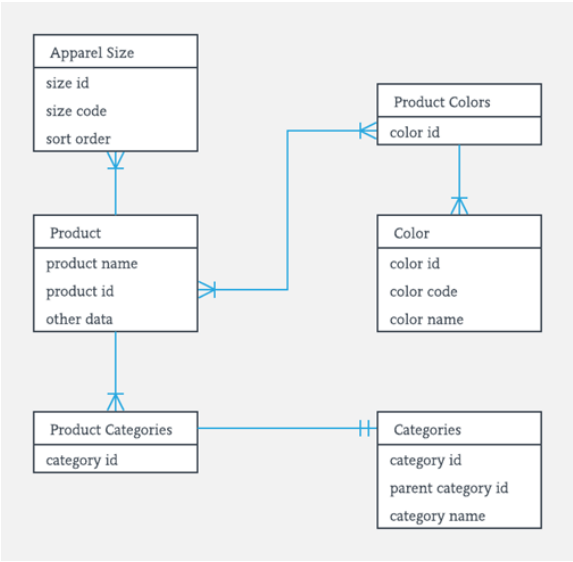
Mahatma Gandhi Mission's College of Engineering and Technology
Kamothe, Navi Mumbai

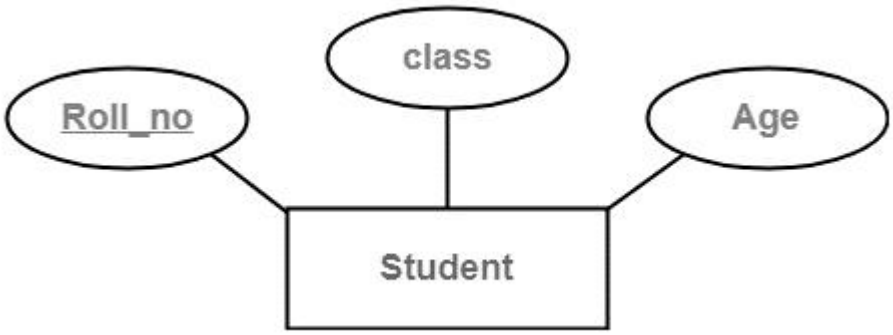
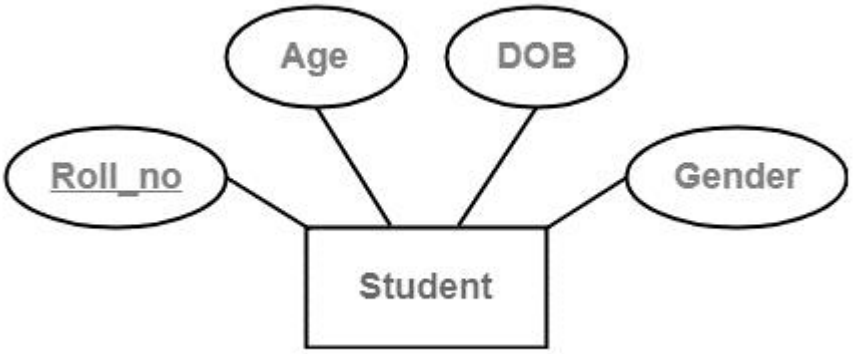
MAHATMA GANDHI MISSION

Assignment -2

Subject-DBMS Div-A & B Sem-IV Class –SE (AC Year 2021-22)

Q.No	Question
Q1.Fill in the blanks	
a)	Entities are two types 1)tangible2)intangible
b)	Real world objects in ER diagram is called as entity.
c)	Properties of entities are called as relationships.
d)	Teacher teaches to whole class is one to many type of relation.
e)	In ER diagram is developed by Peter Chen.
Q2. Choose Correct Options	
a)	Which of the following gives a logical structure of the database graphically? a) Entity-relationship diagram b) Entity diagram c) Database diagram d) Architectural representation
b)	2. The entity relationship set is represented in E-R diagram as a) Double diamonds b) Undivided rectangles c) Dashed lines d) Diamond
c)	An entity set that does not have sufficient attributes to form a primary key is termed a _____ a) Strong entity set b) Variant set c) Weak entity set d) Variable set
d)	If you were collecting and storing information about your music collection, an album would be considered a(n) _____ a) Relation b) Entity c) Instance d) Attribute
e)	Weak entity set is represented as a) Underline b) Double line c) Double diamond d) Double rectangle
Q3. state whether the following statements are true or false (Give Reasons)	
a)	ER was developed by Chen in 1980

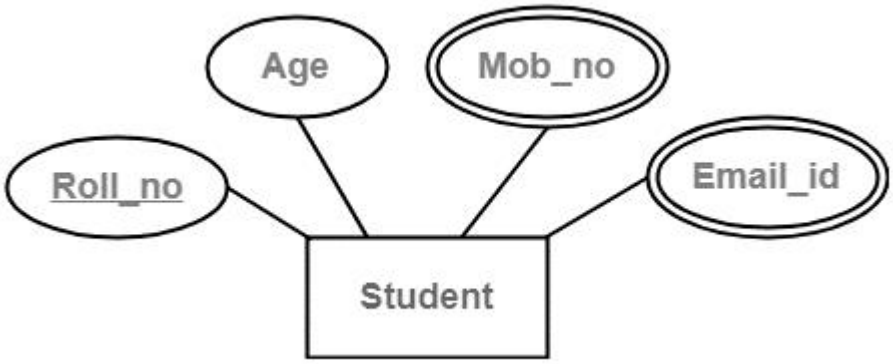
	A. True B. False
b)	An ERs purpose is to support a user's perception of the data and conceal the technical aspects associated with database design. A. True B. False
c)	An Attribute is a property of an entity or a relationship type A. True B. False
Q4. Name the following or define or design the following	
a) **Ans:**	Define ER diagram. **ER Diagram** stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities, attributes and relationships. ER Diagrams contain different symbols that use rectangles to represent entities, ovals to define attributes and diamond shapes to represent relationships. At first look, an ER diagram looks very similar to the flowchart. However, ER Diagram includes many specialized symbols, and its meanings make this model unique. The purpose of ER Diagram is to represent the entity framework infrastructure.  Entity Relationship Diagram Example

b) **Ans:**	List types of attributes. The different types of attributes are as follows − Composite attribute It can be divided into smaller sub parts, each sub part can form an independent attribute. For example − Name FirstName MiddelName LastName Simple or Atomic attribute Attributes that cannot be further subdivided are called atomic attributes. For example − Phone number PIN code  Single valued Attribute Attributes having a single value for a particular item is called a single valued attribute. For example: Room Number  Multi-valued Attribute

Attribute having a set of values for a single entity is called a multi-valued attribute.

For example −

```
e-mail
Tel.No
Hobbies
```

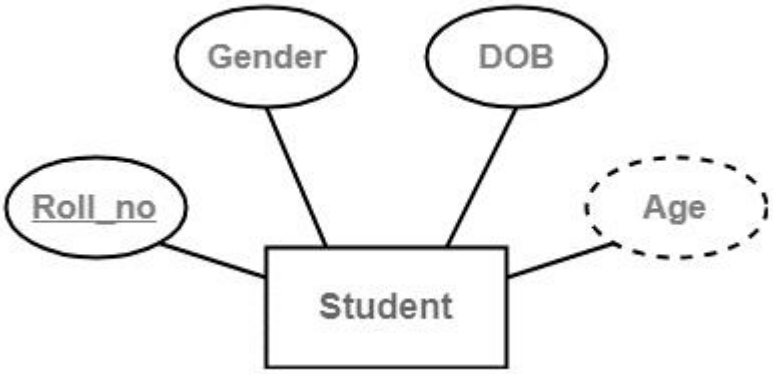


Derived Attributes or stored Attributes

When one attribute value is derived from the other is called a derived attribute.

For example: Age can be derived from date of birth, where,

- Age is the derived attribute.
- DOB is the stored attribute.

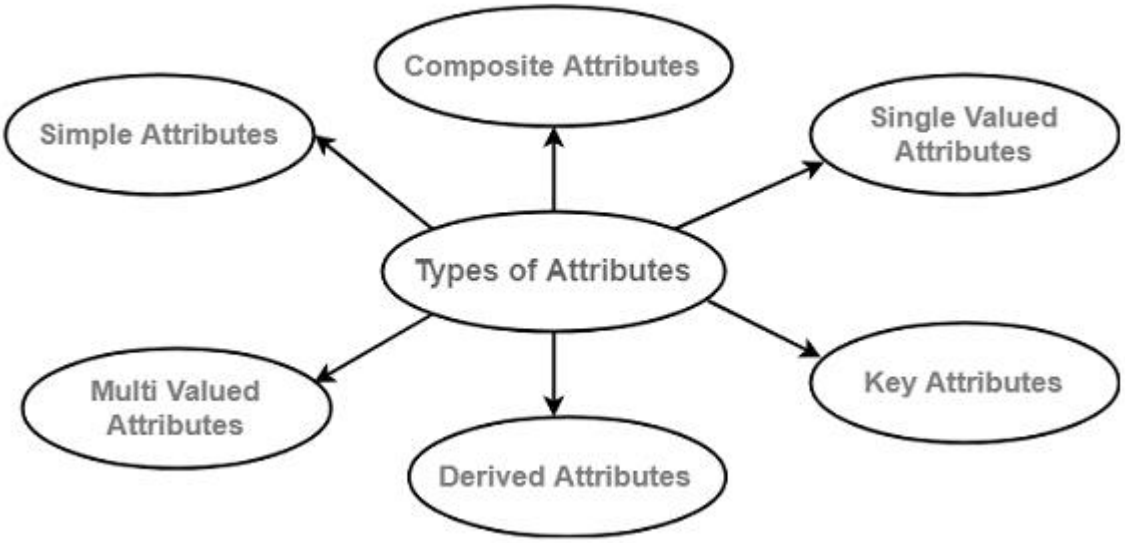


Complex Attribute

Nesting of composite and multi-valued attributes forms a complex attribute.

For example

If a person has more than one house and each house has more than one phone. Then, that attribute phone is represented as a complex attribute.



c)
Ans:

List the types of Entities

Strong Entity

The strong entity has a primary key. Weak entities are dependent on strong entity. Its existence is not dependent on any other entity.

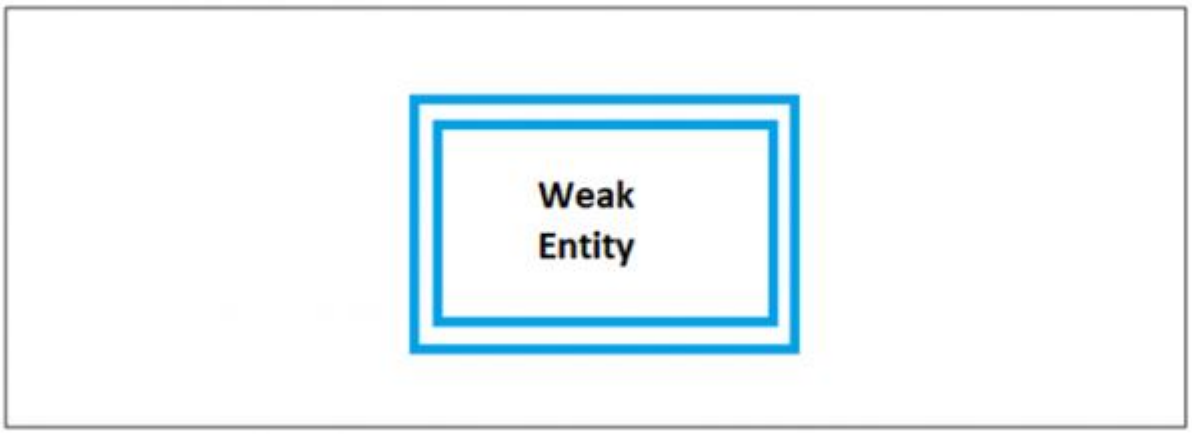
Strong Entity is represented by a single rectangle −

Continuing our previous example, **Professor** is a strong entity here, and the primary key is **Professor_ID.**

Weak Entity

The weak entity in DBMS do not have a primary key and are dependent on the parent entity. It mainly depends on other entities.

Weak Entity is represented by double rectangle −



Continuing our previous example, **Professor** is a strong entity, and the primary key is **Professor_ID**. However, another entity is **Professor_Dependents**, which is our Weak Entity.

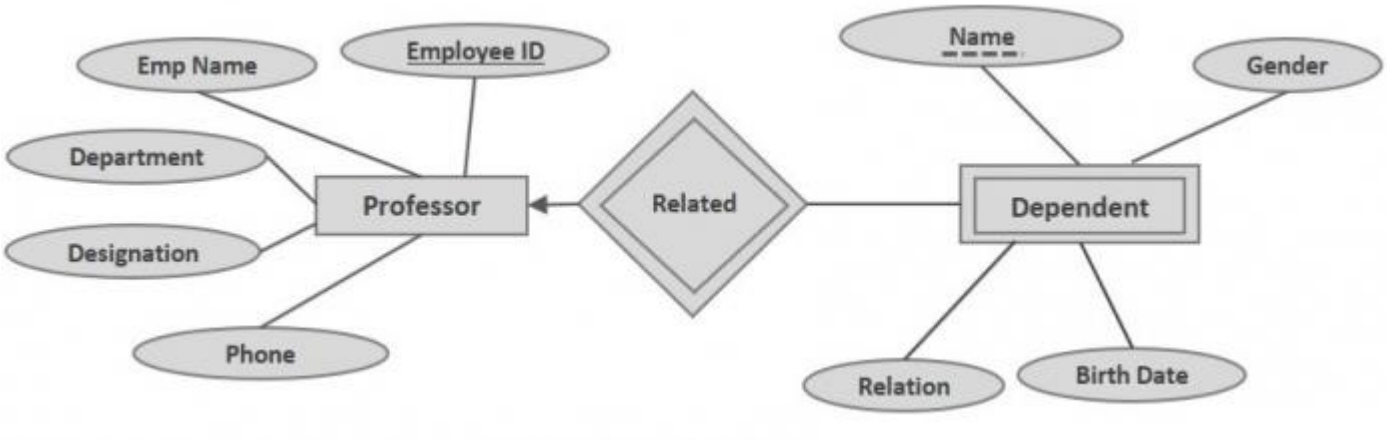
<Professor_Dependents>

Name	**DOB**	**Relation**

This is a weak entity since its existence is dependent on another entity **Professor**, which we saw above. A Professor has Dependents.

Example of Strong and Weak Entity

The example of a strong and weak entity can be understood by the below figure.



The Strong Entity is **Professor**, whereas **Dependent** is a Weak Entity.

ID is the primary key (represented with a line) and the Name in **Dependent** entity is called **Partial Key** (represented with a dotted line).

Q5. Answer the following questions in brief (20 to 30 words)

a) Ans: Explain the features of ER diagram?

ER Diagrams are composed of entities, relationships and attributes. They also depict cardinality, which defines relationships in terms of numbers. Here's a

glossary:

Entity

A definable thing—such as a person, object, concept or event—that can have data stored about it. Think of entities as nouns. Examples: a customer, student, car or product. Typically shown as a rectangle.

Entity type: A group of definable things, such as students or athletes, whereas the entity would be the specific student or athlete. Other examples: customers, cars or products.

Entity set: 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.

Entity categories: Entities are categorized as strong, weak or associative. A **strong entity** can be defined solely by its own attributes, while a **weak entity** cannot. An associative entity associates entities (or elements) within an entity set.

Entity keys: Refers to an attribute that uniquely defines an entity in an entity set. Entity keys can be super, candidate or primary. **Super key:** A set of attributes (one or more) that together define an entity in an entity set. **Candidate key:** A minimal super key, meaning it has the least possible number of attributes to still be a super key. An entity set may have more than one candidate key. **Primary key:** A candidate key chosen by the database designer to uniquely identify the entity set. **Foreign key:** Identifies the relationship between entities.

Relationship

How entities act upon each other or are associated with each other. Think of relationships as verbs. For example, the named student might register for a course. The two entities would be the student and the course, and the relationship depicted is the act of enrolling, connecting the two entities in that way. Relationships are typically shown as diamonds or labels directly on the connecting lines.

Recursive relationship: The same entity participates more than once in the relationship.

Attribute

A property or characteristic of an entity. Often shown as an oval or circle.

Descriptive attribute: A property or characteristic of a relationship (versus of an entity.)

Attribute categories: Attributes are categorized as simple, composite, derived, as well as single-value or multi-value. **Simple:** Means the attribute value is atomic and can't be further divided, such as a phone number. **Composite:** Sub-attributes spring from an attribute. **Derived:** Attributed is calculated or otherwise derived from another attribute, such as age from a birthdate.

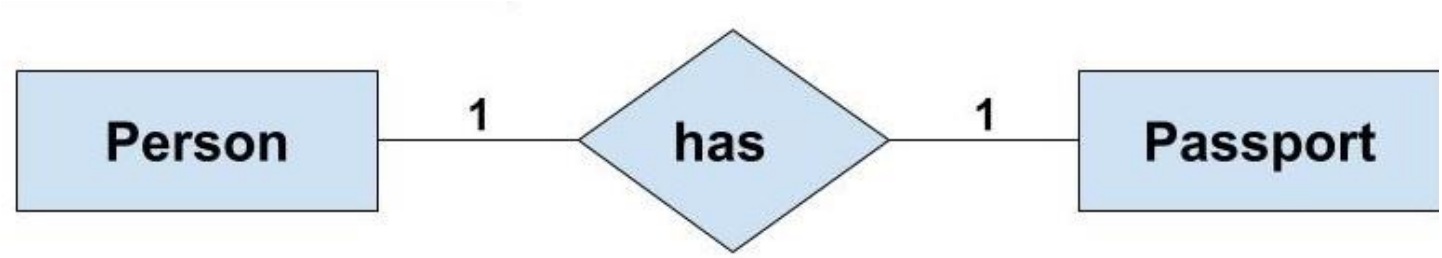
Multi-value: More than one attribute value is denoted, such as multiple phone numbers for a person.

Single-value: Just one attribute value. The types can be combined, such as: simple single-value attributes or composite multi-value attributes.

Cardinality

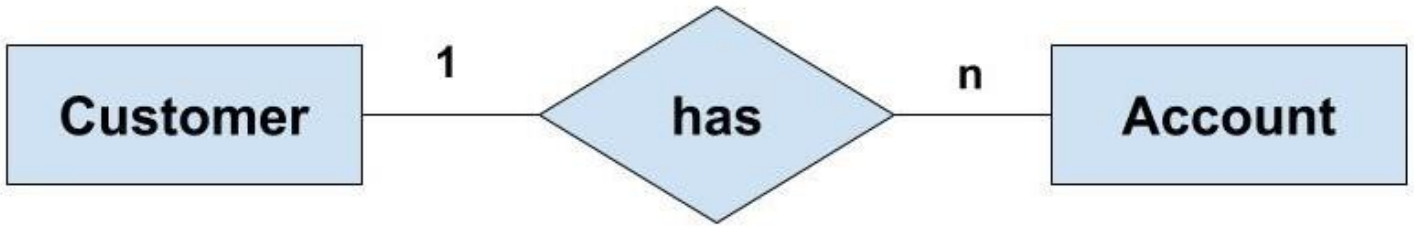
Defines the numerical attributes of the relationship between two entities or entity sets. The three main cardinal relationships are one-to-one, one-to-many, and many-many. A **one-to-one example** would be one student associated with one mailing address. A **one-to-many example (or many-to-one, depending on the relationship direction):** One student registers for multiple courses, but all those courses have a single line back to that one student. **Many-to-many example:** Students as a group are associated with multiple faculty members, and faculty members in turn are associated with multiple students.

Cardinality views: Cardinality can be shown as look-across or same-side,

	depending on where the symbols are shown. **Cardinality constraints:** The minimum or maximum numbers that apply to a relationship.
b) **Ans:**	What are the types of relationships ## *One-to-One Relationship* Such a relationship exists when each record of one table is related to only one record of the other table. ***For example,*** If there are two entities 'Person' (Id, Name, Age, Address)and 'Passport'(Passport_id, Passport_no). So, each person can have only one passport and each passport belongs to only one person.  Such a relationship is not very common. However, such a relationship is used for security purposes. In the above example, we can easily store the passport id in the 'Person' table only. But, we make another table for the 'Passport' because Passport number may be sensitive data and it should be hidden from certain users. So, by making a separate table we provide extra security that only certain database users can see it. ## *One-to-Many or Many-to-One Relationship* Such a relationship exists when each record of one table can be related to one or more than one record of the other table. This relationship is the most common relationship found. A one-to-many relationship can also be said as a

many-to-one relationship depending upon the way we view it.

For example, If there are two entity type ‘Customer’ and ‘Account’ then each ‘Customer’ can have more than one ‘Account’ but each ‘Account’ is held by only one ‘Customer’. In this example, we can say that each Customer is associated with many Account. So, it is a one-to-many relationship. But, if we see it the other way i.e many Account is associated with one Customer then we can say that it is a many-to-one relationship.

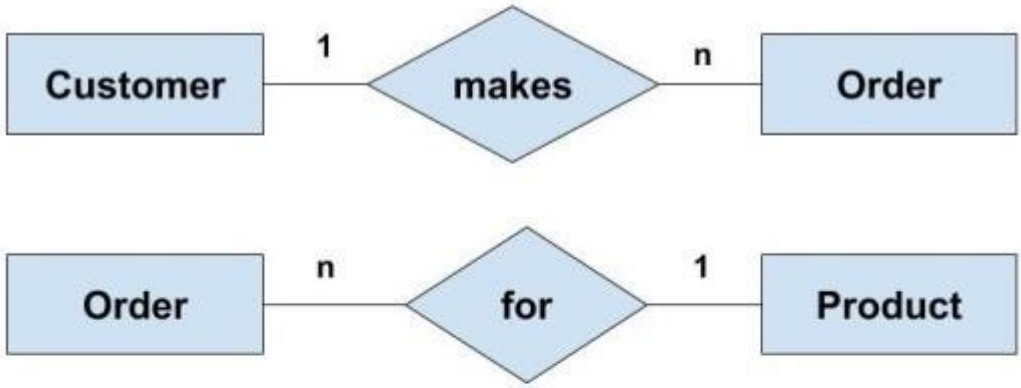


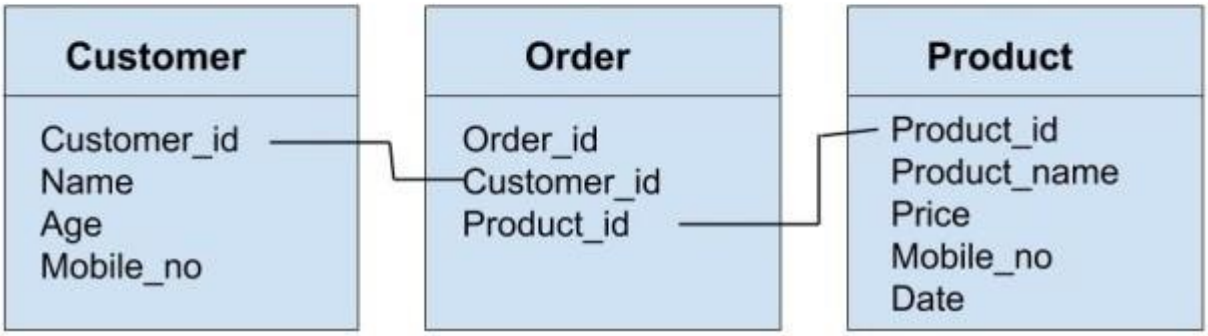
Many-to-Many Relationship

Such a relationship exists when each record of the first table can be related to one or more than one record of the second table and a single record of the second table can be related to one or more than one record of the first table. A many-to-many relationship can be seen as a two one-to-many relationship which is linked by a 'linking table' or 'associate table'. The linking table links two tables by having fields which are the primary key of the other two tables. We can understand this with the following example.

Example: If there are two entity type ‘Customer’ and ‘Product’ then each customer can buy more than one product and a product can be bought by many different customers.

Now, to understand the concept of the linking table here, we can have the ‘Order’ entity as a linking table which links the ‘Customer’ and ‘Product’ entity. We can break this many-to-many relationship in two one-to-many relationships. First, each ‘Customer’ can have many ‘Order’ whereas each ‘Order’ is related to only one ‘Customer’. Second, each ‘Order’ is related only one Product wheres there can many orders for the same Product.





In the above concept of linking can be understood with the help of taking into consideration all the attributes of the entities 'Customer', 'Order' and 'Product'. We can see that the primary key of both 'Customer' and 'Product' entity are included in the linking table i.e 'Order' table. These key act as foreign keys

while referring to the respective table from the 'Order' table.

c)

Ans:

Difference between specialization and Generalization

GENERALIZATION	SPECIALIZATION
Generalization works in Bottom-Up approach.	Specialization works in top-down approach.
In Generalization, size of schema gets reduced.	In Specialization, size of schema gets increased.
Generalization is normally applied to group of entities.	We can apply Specialization to a single entity.
Generalization can be defined as a process of creating groupings from various entity sets	Specialization can be defined as process of creating subgrouping within an entity set
In Generalization process, what actually happens is that it takes the union of two or more lower-level entity sets to produce a higher-level entity sets.	Specialization is reverse of Generalization. Specialization is a process of taking a subset of a higher level entity set to form a lower-level entity set.
Generalization process starts with the number of entity sets and it creates high-level entity with the help of some common features.	Specialization process starts from a single entity set and it creates a different entity set by using some different features.
In Generalization, the difference and similarities between lower entities are ignored to form a higher entity.	In Specialization, a higher entity is split to form lower entities.
There is no inheritance in Generalization.	There is inheritance in Specialization.

. Q6. Answer the following questions in brief (50 to 70 words)

a)

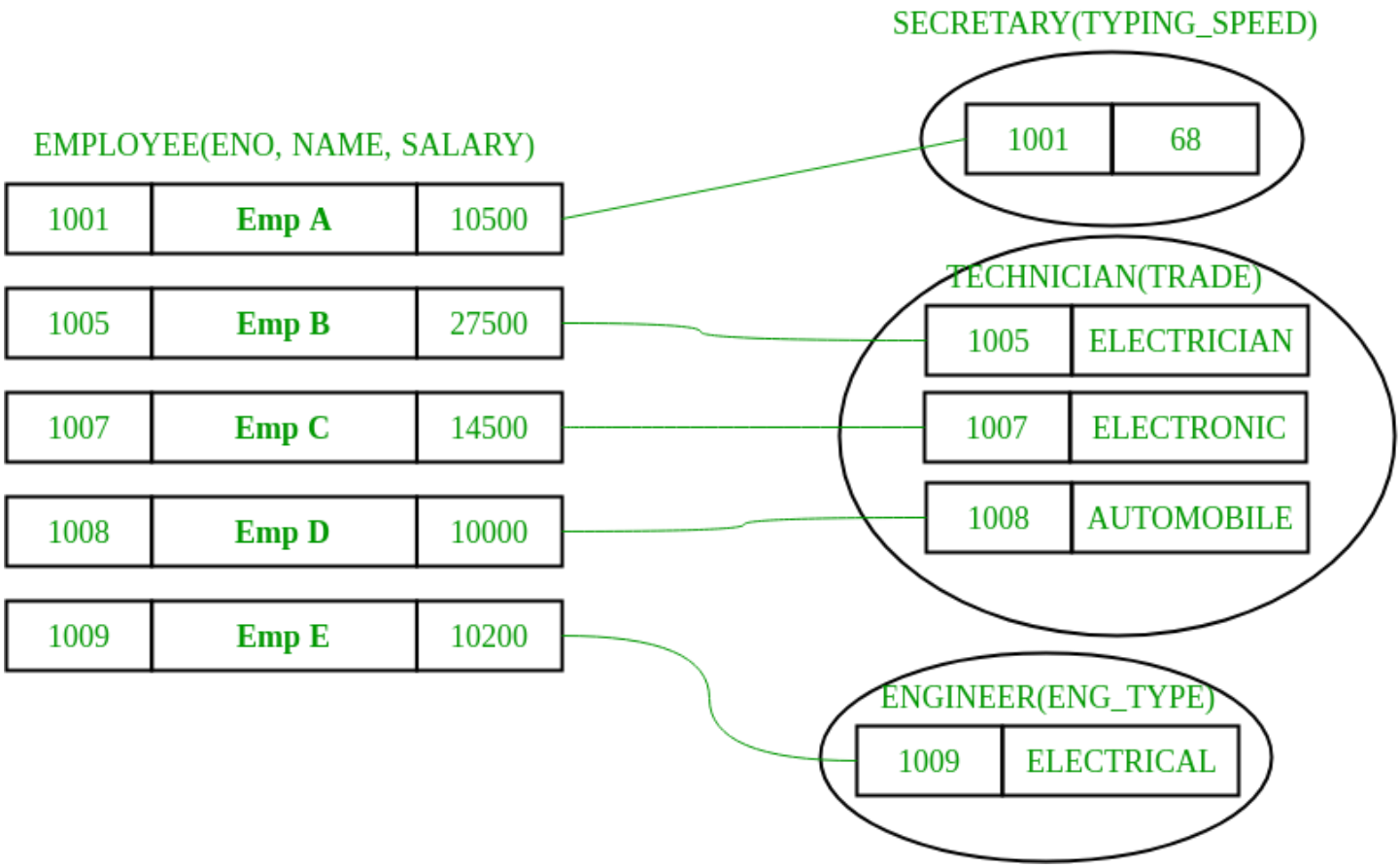
Ans:

Explain EER diagram with properties

Enhanced entity-relationship diagrams are advanced database diagrams very similar to regular ER diagrams which represent requirements and complexities of complex databases.
It is a diagrammatic technique for displaying the Sub Class and Super Class; Specialization and Generalization; Union or Category; Aggregation etc.

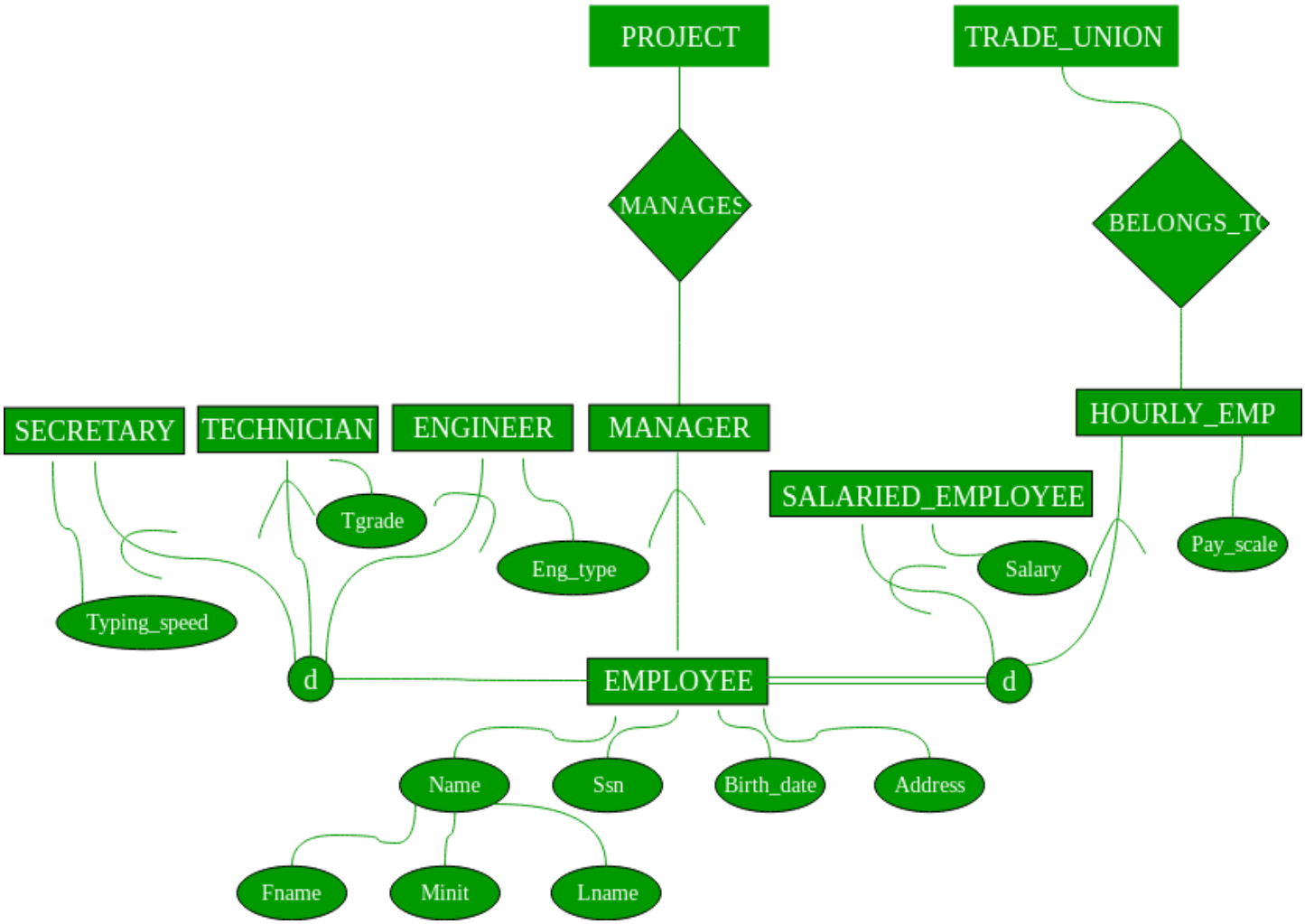
Generalization and Specialization –
These are very common relationships found in real entities. However, this kind of relationship was added later as an enhanced extension to the classical ER model. **Specialized** classes are often called **subclass** while a **generalized class** is called a superclass, probably inspired by object-oriented programming. A sub-class is best understood by **"IS-A analysis"**. Following statements hopefully makes some sense to your mind "Technician IS-A Employee", "Laptop IS-A Computer".
An entity is a specialized type/class of another entity. For example, a Technician is a special Employee in a university system Faculty is a special class of Employee. We call this phenomenon generalization/specialization. In the example here Employee is a generalized entity class while the Technician and Faculty are specialized classes of Employee.

Example – This example instance of **"sub-class"** relationships. Here we have four sets of employees: Secretary, Technician, and Engineer. The employee is super-class of the rest three sets of individual sub-class is a subset of Employee set.



- An entity belonging to a sub-class is related to some super-class entity. For instance emp, no 1001 is a secretary, and his typing speed is 68. Emp no 1009 is an engineer (sub-class) and her trade is "Electrical", so forth.
- Sub-class entity "inherits" all attributes of super-class; for example, employee 1001 will have attributes eno, name, salary, and typing speed.

Enhanced ER model of above example –



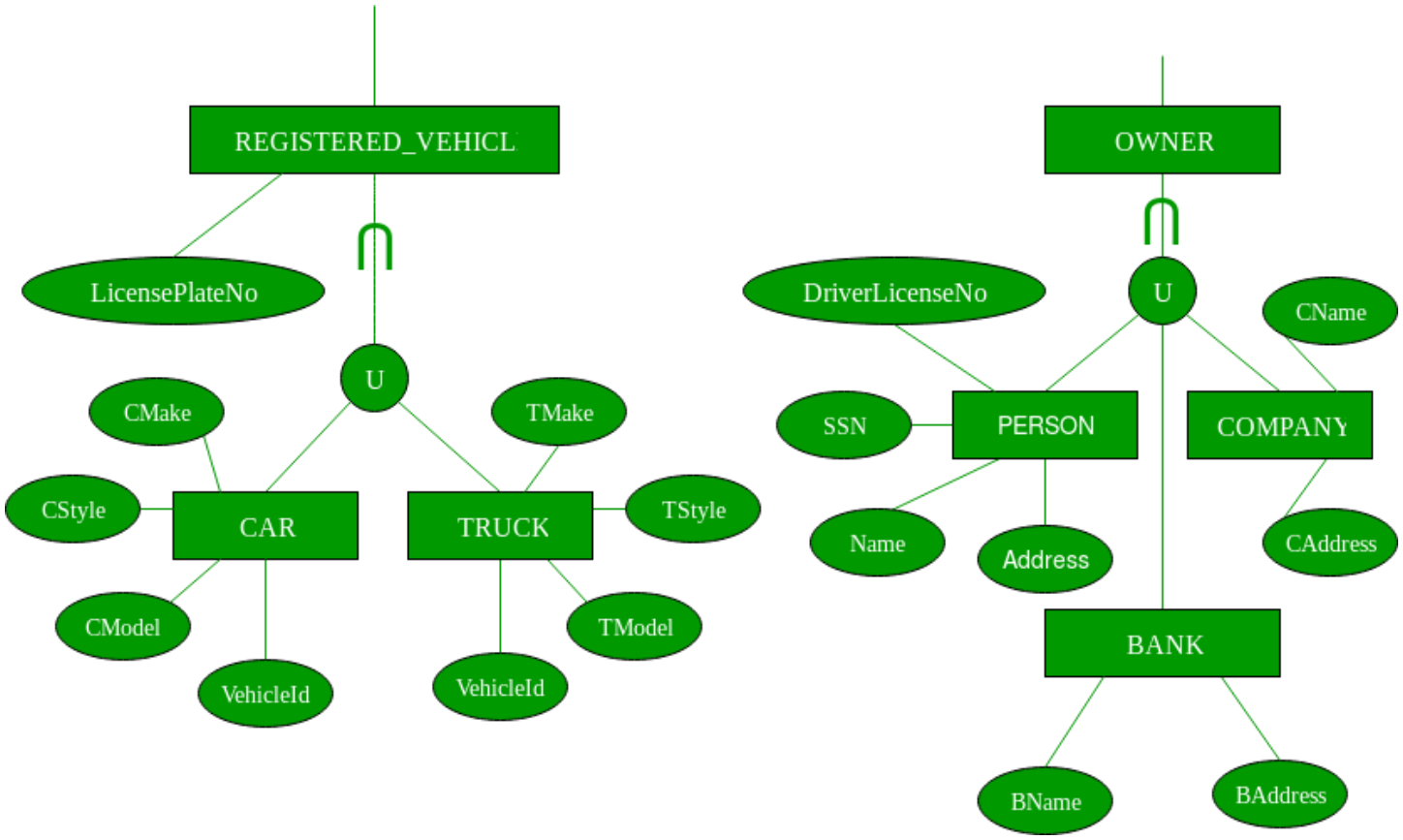
Constraints – There are two types of constraints on the “Sub-class” relationship.

1. **Total or Partial –** A sub-classing relationship is total if every super-class entity is to be associated with some sub-class entity, otherwise partial. Sub-class “job type based employee category” is partial sub-classing – not necessary every employee is one of (secretary, engineer, and technician), i.e. union of these three types is a proper subset of all employees. Whereas other sub-classing “Salaried Employee AND Hourly Employee” is total; the union of entities from sub-classes is equal to the total employee set, i.e. every employee necessarily has to be one of them.
2. **Overlapped or Disjoint –** If an entity from super-set can be related (can occur) in multiple sub-class sets, then it is overlapped sub-classing, otherwise disjoint. Both the examples: job-type based and salaries/hourly employee sub-classing are disjoint.

Note – These constraints are independent of each other: can be “overlapped and total or partial” or “disjoint and total or partial”. Also, sub-classing has transitive property.

Multiple Inheritance (sub-class of multiple superclasses**) –**

An entity can be a sub-class of multiple entity types; such entities are sub-class of multiple entities and have multiple super-classes; Teaching Assistant can subclass of Employee and Student both. A faculty in a university system can be a subclass of Employee and Alumnus.

	In multiple inheritances, attributes of sub-class are the union of attributes of all super-classes. **Union –** • Set of Library Members is **UNION** of Faculty, Student, and Staff. A union relationship indicates either type; for example, a library member is either Faculty or Staff or Student. • Below are two examples show how **UNION** can be depicted in ERD – Vehicle Owner is UNION of PERSON and Company, and RTO Registered Vehicle is UNION of Car and Truck. 
b) **Ans:**	Draw and explain Database Architecture. **Database Architecture** is a representation of DBMS design. It helps to design, develop, implement, and maintain the database management system. A DBMS architecture allows dividing the database system into individual components that can be independently modified, changed, replaced, and altered. It also helps to understand the components of a database. A Database stores critical information and helps access data quickly and securely. Therefore, selecting the correct Architecture of DBMS helps in easy and efficient data management. • Types of DBMS Architecture • 1-Tier Architecture • 2-Tier Architecture

- 3-Tier Architecture

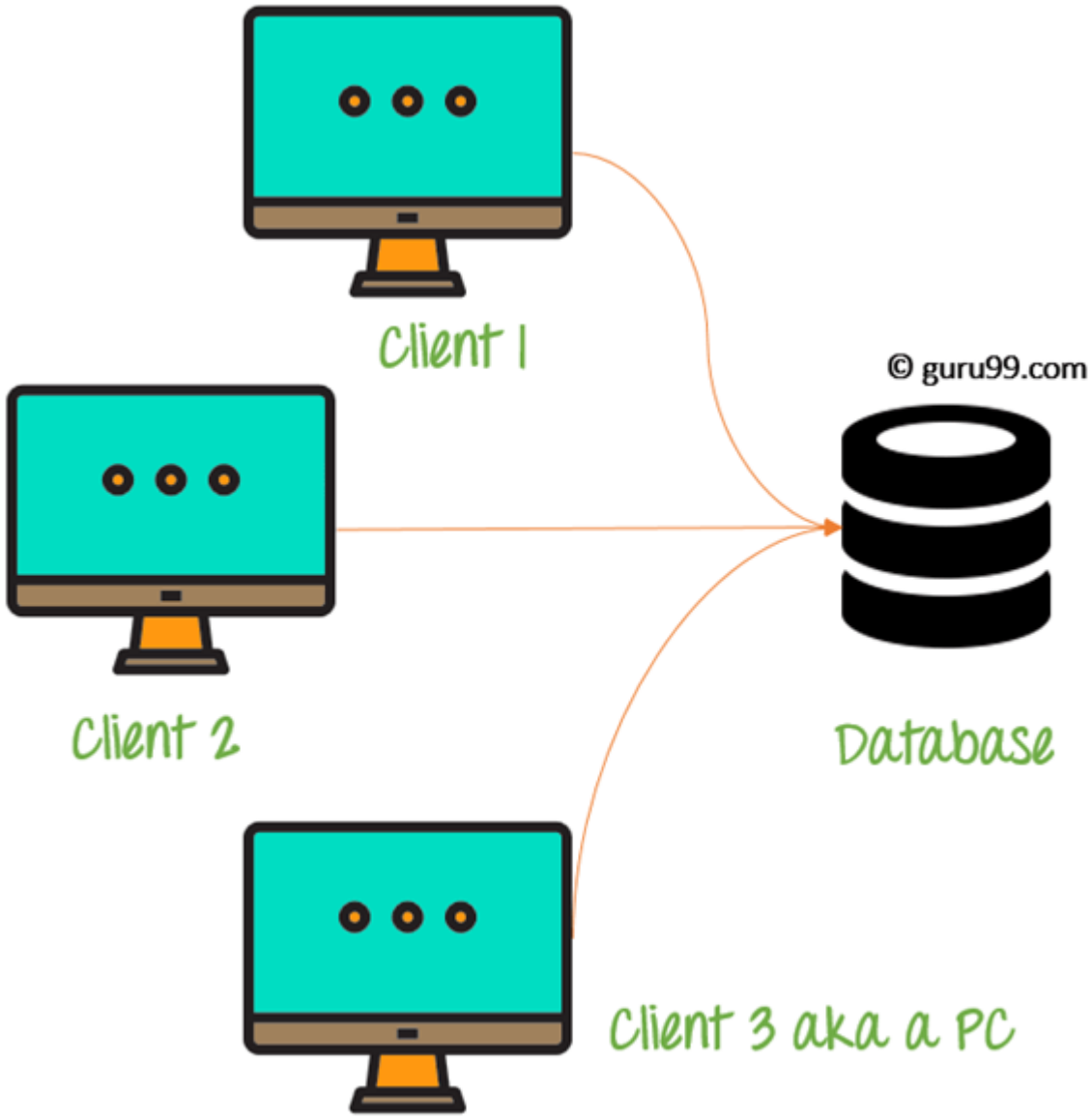
1-Tier Architecture

1 Tier Architecture in DBMS is the simplest architecture of Database in which the client, server, and Database all reside on the same machine. A simple one tier architecture example would be anytime you install a Database in your system and access it to practice SQL queries. But such architecture is rarely used in production.

1 Tier Architecture Diagram

2-Tier Architecture

A **2 Tier Architecture** in DBMS is a Database architecture where the presentation layer runs on a client (PC, Mobile, Tablet, etc.), and data is stored on a server called the second tier. Two tier architecture provides added security to the DBMS as it is not exposed to the end-user directly. It also provides direct and faster communication.



2 Tier Architecture Diagram

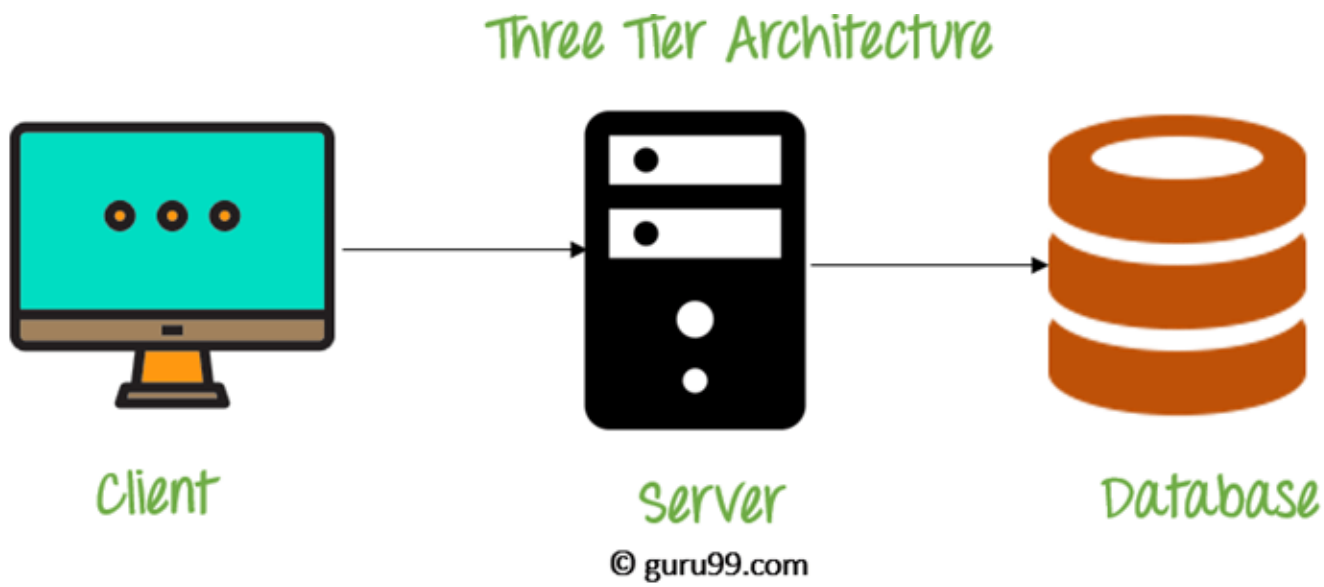
In the above 2 Tier client-server architecture of database management system, we can see that one server is connected with clients 1, 2, and 3.

Two Tier Architecture Example:

A Contact Management System created using MS- Access.

3-Tier Architecture

A **3 Tier Architecture** in DBMS is the most popular client server architecture in DBMS in which the development and maintenance of functional processes, logic, data access, data storage, and user interface is done independently as separate modules. Three Tier architecture contains a presentation layer, an application layer, and a database server.

	3-Tier database Architecture design is an extension of the 2-tier client-server architecture. A 3-tier architecture has the following layers: 1. Presentation layer (your PC, Tablet, Mobile, etc.) 2. Application layer (server) 3. Database Server  3 Tier Architecture Diagram The Application layer resides between the user and the DBMS, which is responsible for communicating the user's request to the DBMS system and send the response from the DBMS to the user. The application layer(business logic layer) also processes functional logic, constraint, and rules before passing data to the user or down to the DBMS.

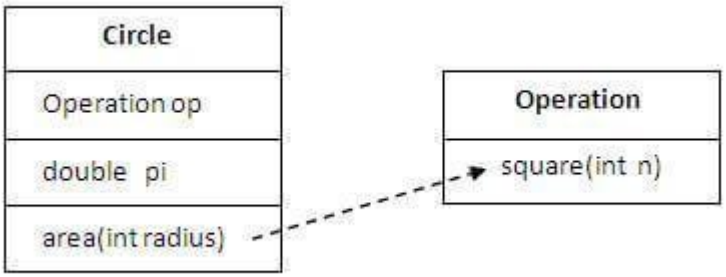
c)
 	
Ans:
	Explain Aggregation in details
	
	If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.
	
	Consider a situation, Employee object contains many informations such as id, name, emailld etc. It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc. as given below.
	
	1. **class** Employee{
	2. **int** id;
	3. String name;
	4. Address address;//Address is a class
	5. ...
	6. }

In such case, Employee has an entity reference address, so relationship is Employee HAS-A address.

Why use Aggregation?

- For Code Reusability.

Simple Example of Aggregation



In this example, we have created the reference of Operation class in the Circle class.

```
1. class Operation{
2.  int square(int n){
3.   return n*n;
4.  }
5. }
6.
7. class Circle{
8.  Operation op;//aggregation
9.  double pi=3.14;
10.
11. double area(int radius){
12.   op=new Operation();
13.   int rsquare=op.square(radius);//code reusability (i.e. delegates the method call).
14.   return pi*rsquare;
15. }
16.
17.
18.
19. public static void main(String args[]){
20.   Circle c=new Circle();
```

```
21.    double result=c.area(5);
22.    System.out.println(result);
23.  }
24. }
```

When use Aggregation?

- Code reuse is also best achieved by aggregation when there is no is-a relationship.
- Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

Understanding meaningful example of Aggregation

In this example, Employee has an object of Address, address object contains its own informations such as city, state, country etc. In such case relationship is Employee HAS-A address.

Address.java

```
1.  public class Address {
2.  String city,state,country;
3.
4.  public Address(String city, String state, String country) {
5.      this.city = city;
6.      this.state = state;
7.      this.country = country;
8.  }
9.
10. }
```

Emp.java

```
1.  public class Emp {
2.  int id;
3.  String name;
4.  Address address;
5.
6.  public Emp(int id, String name,Address address) {
7.      this.id = id;
8.      this.name = name;
9.      this.address=address;
10. }
11.
```

```
12. void display(){
13. System.out.println(id+" "+name);
14. System.out.println(address.city+" "+address.state+" "+address.country);
15. }
16.
17. public static void main(String[] args) {
18. Address address1=new Address("gzb","UP","india");
19. Address address2=new Address("gno","UP","india");
20.
21. Emp e=new Emp(111,"varun",address1);
22. Emp e2=new Emp(112,"arun",address2);
23.
24. e.display();
25. e2.display();
26.
27. }
28. }
```

Q7. Think and Answer

a)

What is type of cardinality of relations?

Ans:

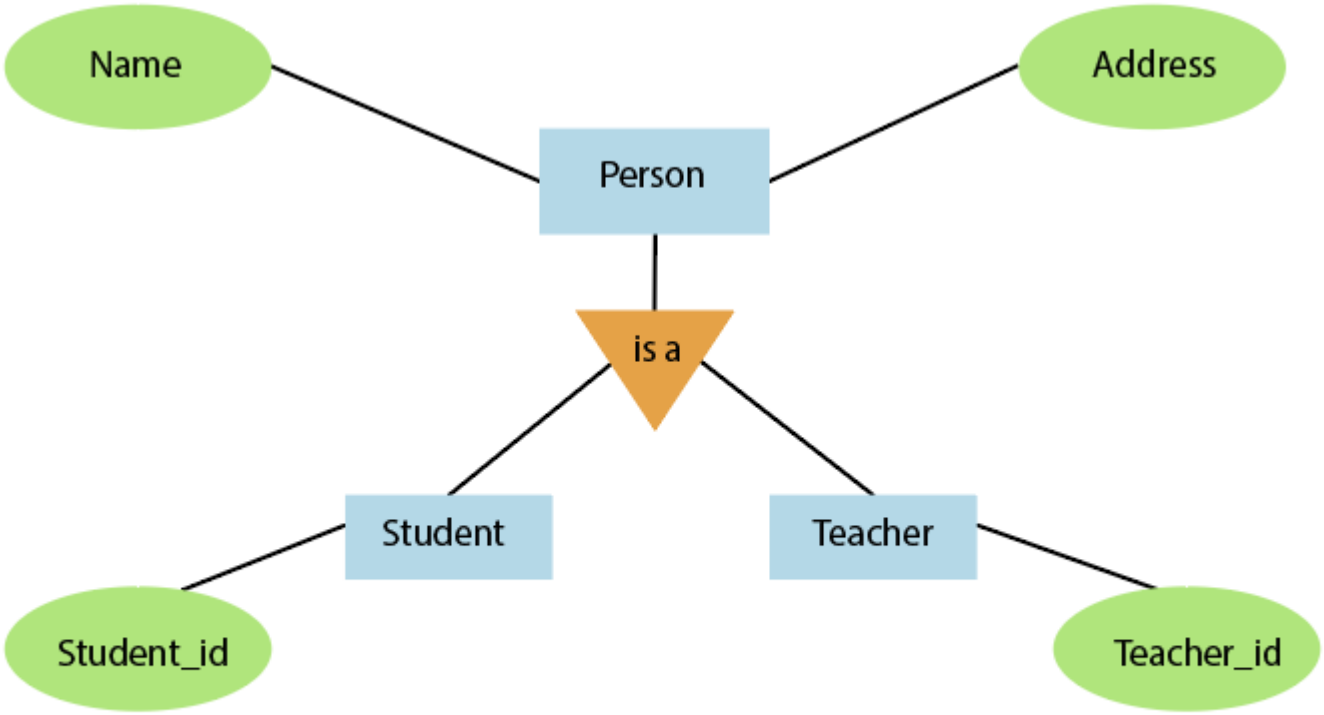
The cardinality of the relationship means having unique or multiple instances per value for the joining field between two tables. The most common type of cardinality is **one-to-many or many-to-one** which happens between fact and dimension tables. However, you can find one-to-one relationships too.

b)

Give the example of Generalization and Aggregation

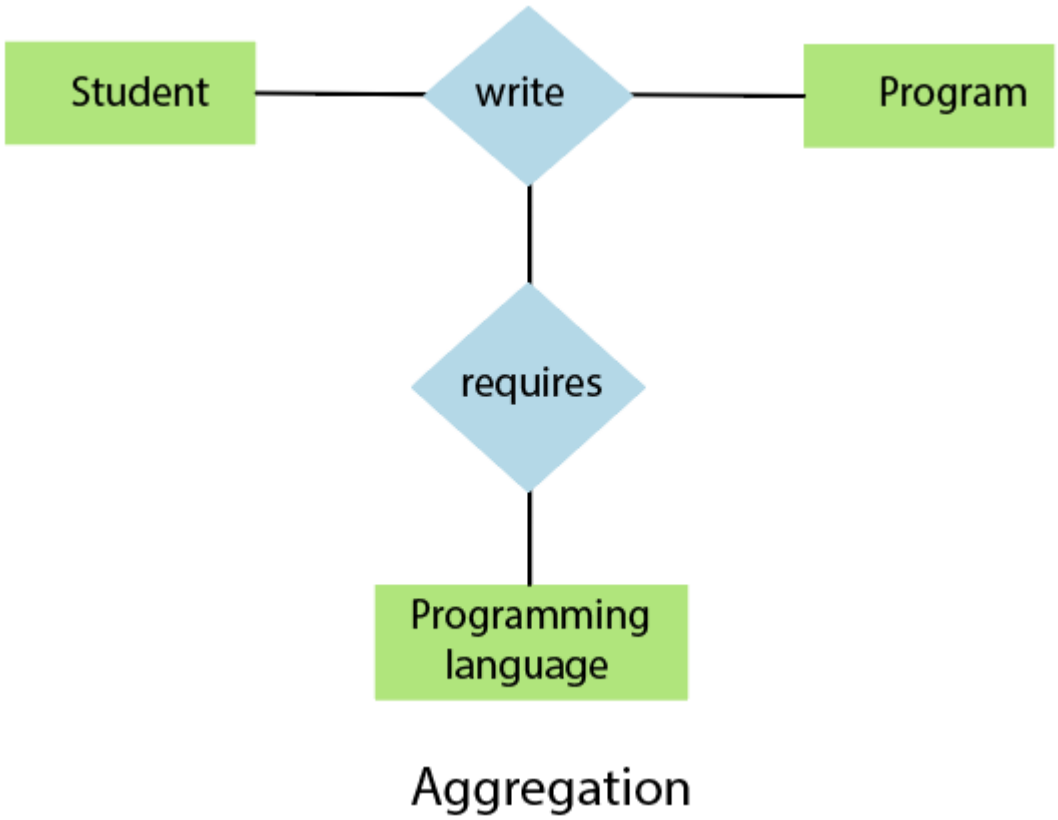
Ans:

According to the below diagram, there are two entities, namely **teacher** and **student**. The teacher entity contains attributes such as **teacher_id, name, and address** and student entity include **student_id, name, and address**. Both entities can be combined to create a higher-level entity **person**. The address and name are common to both the entities. The teacher entity has its attribute teacher_id, and the student has its attribute student_id. The entities teacher and student are generalized further into the person entity.
A lower-level entity is called a subclass, and the higher-level entity is called a superclass. So, the **person** entity is the superclass of two subclasses **teacher** and **student**.



Generalisation

Example:
The **student** and **program** are two entity set and related through a relationship set **write**. Suppose, a student who writes any program must require a programming language installed in their system. So, there will be another relationship set **requires**. You need to connect the relationship set **requires** with an entity set **programming language** and relationship set **write.** But, we can connect only entity set to a relationship set.
One relationship set cannot be connected with another relationship set; for this, we need aggregation. Here, the **write** (relationship set) will treat like a **higher-level entity** and can be associated with a relationship set **requires**. So, aggregation is needed when you express a relationship set with another relationship set.

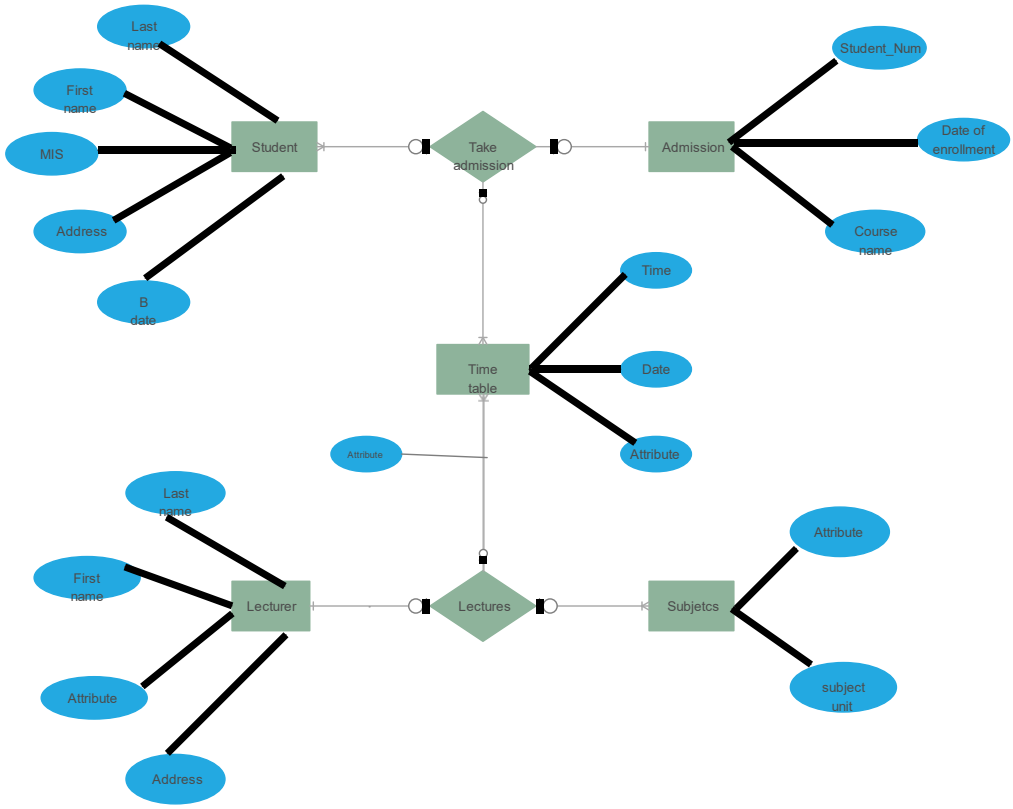


Aggregation

Q8. **My Ideas**

a) Draw college management system ER diagram which includes aggregation generalization and specialization

Ans:



b) **Ans:**	Draw Payroll system ER diagram which includes aggregation generalization and specialization