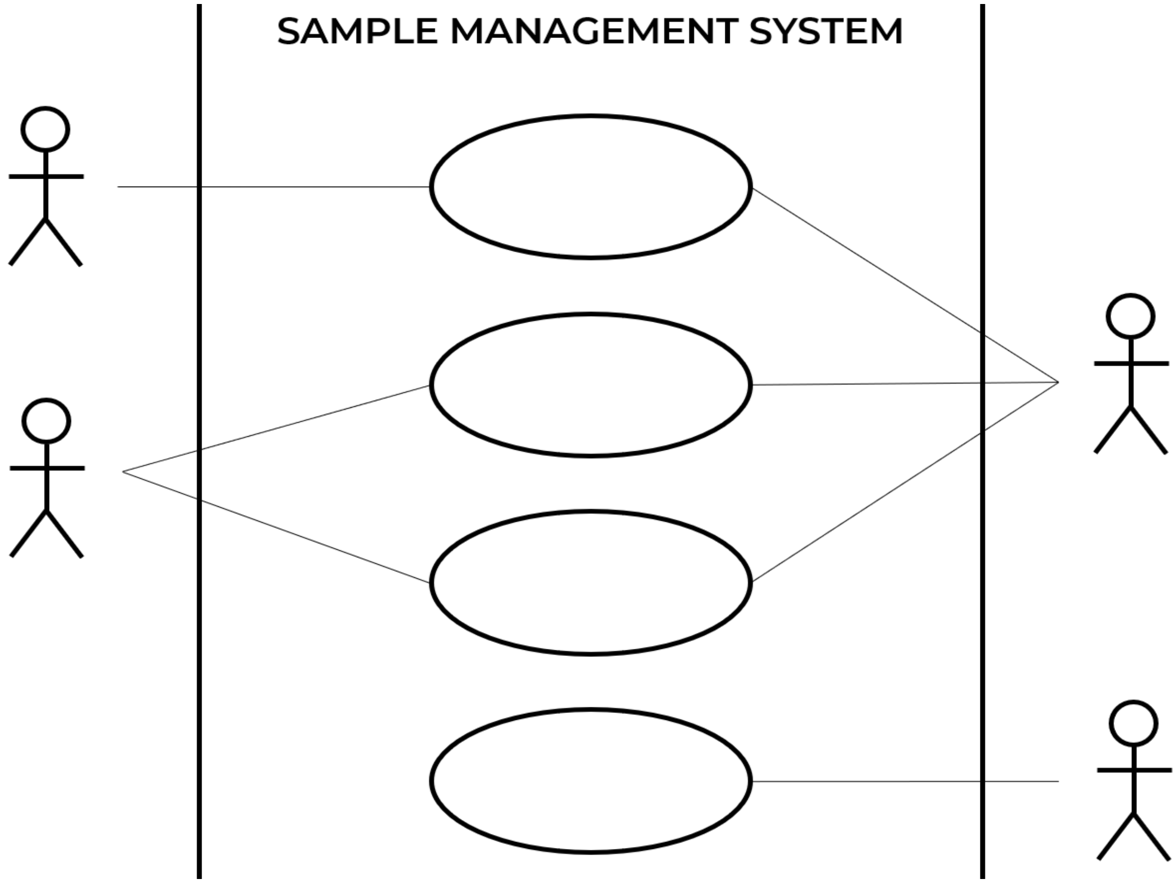
Note Taking Assignment: DMDD Lectures

Yash Revadekar -0027387766

Use Case diagram in software engineering.

A use case diagram is a means to condense information about a system and the users within it. The interactions between various system elements are typically represented graphically.It is used to represent dynamic behavior of a system.

Always focus on the problems and not the tools.
Sample Structure of an Use Case Diagram is shown below:



We can use a Lucidchart website for all the diagram creations.

UseCase Diagram Components:

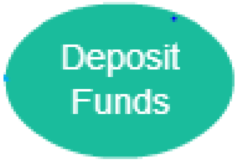
- Use cases: Horizontally shaped ovals that represent the different uses that a user might have.
- Actors: Stick figures that represent the people actually employing the use cases.
- Associations: A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.
- System boundary boxes: A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system.
- Packages: A UML shape that allows you to put different elements into groups. Just as with

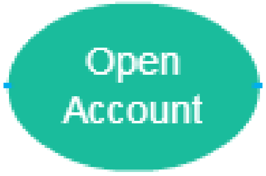
component diagrams, these groupings are represented as file folders.

Use Case Diagram of Banking Application

1. Use Case

   A use case in a use case diagram is a visual representation of a distinct business functionality in a system. Use cases is shown in oval shaped.



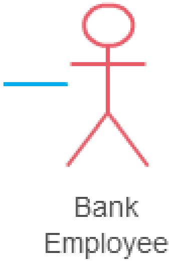


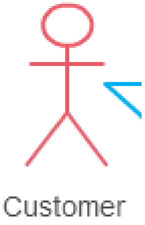
2. Actors

   An actor portrays any entity (or entities) that perform certain roles in a given system. The different roles the actor represents are the actual business roles of users in a given system.

   Types of actors:

   1. Primary: initiates the use of the system (should be at the left of the system)
   2. Secondary: reactory (on the right)

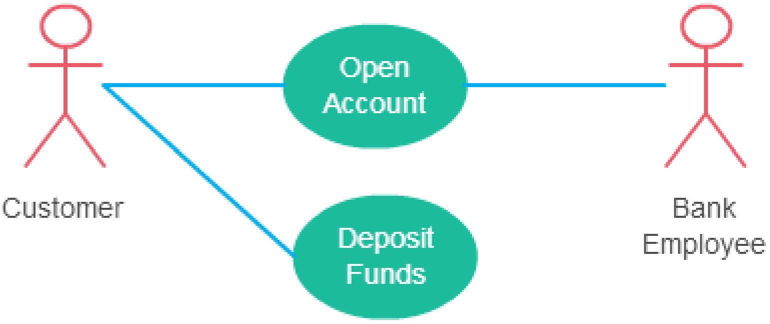




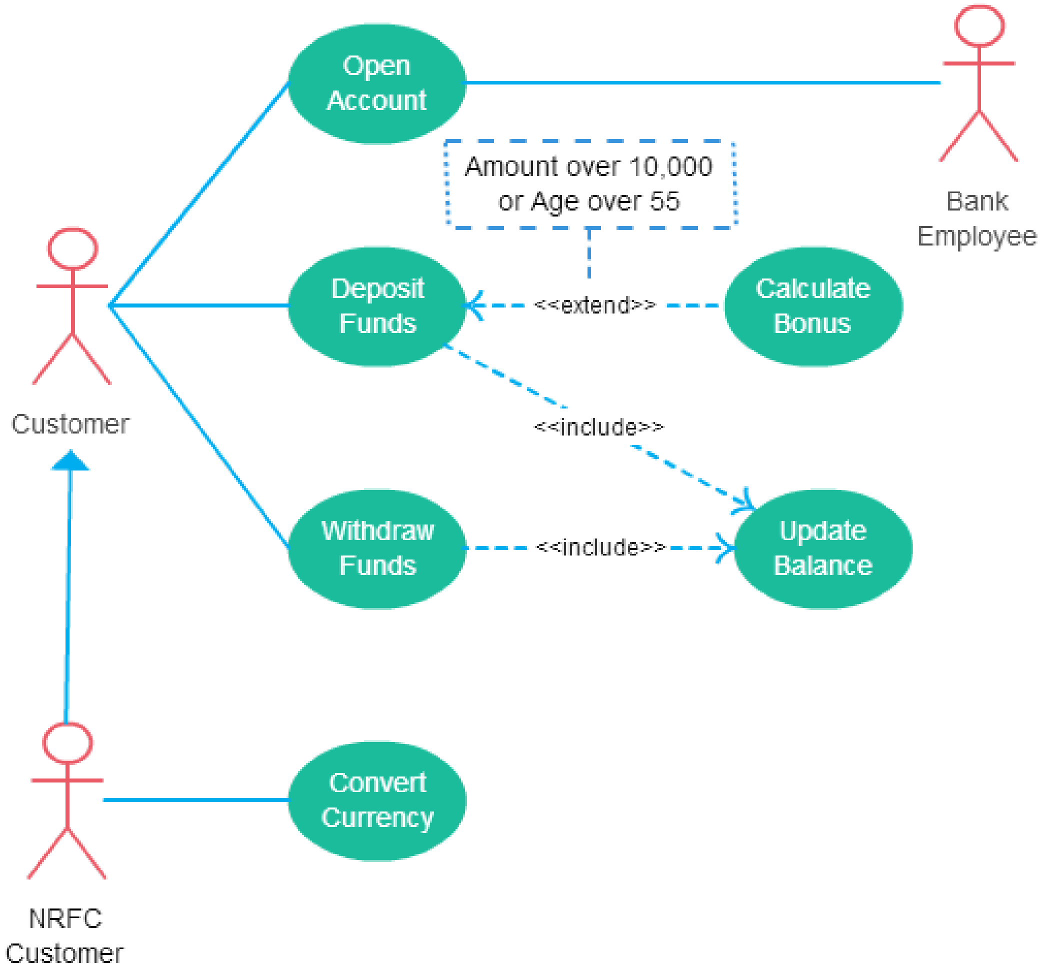
3. Associations

Use cases share different kinds of relationships. A relationship between two use cases is basically a dependency between the two use cases.

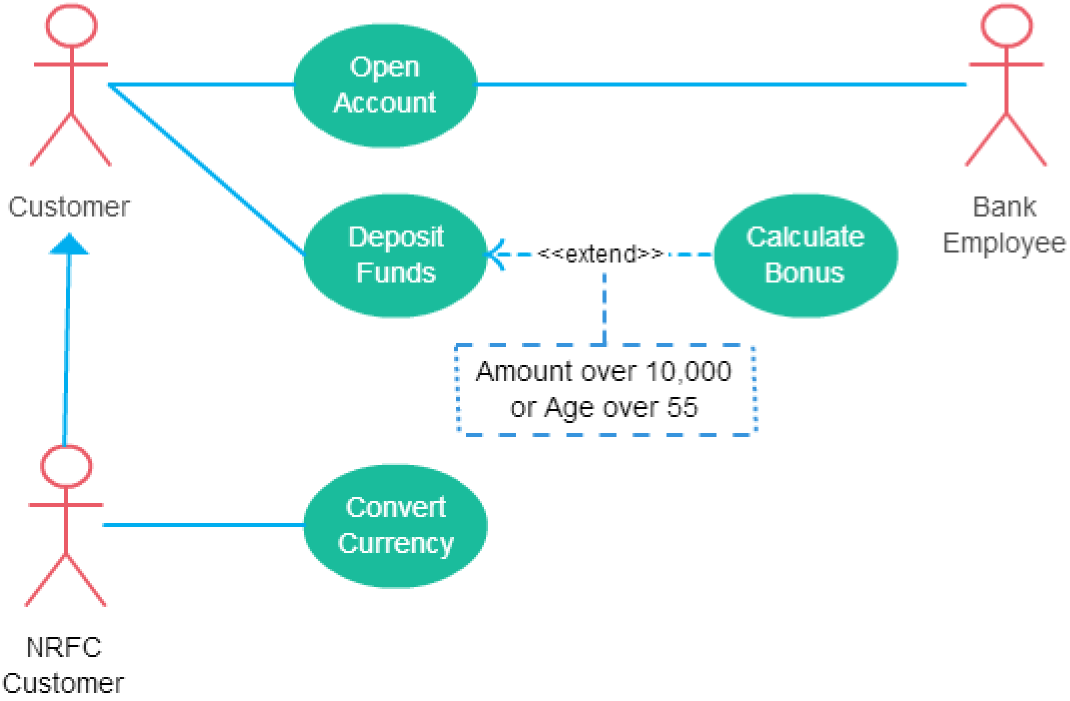
Association: This one is straightforward and present in every use case diagram.



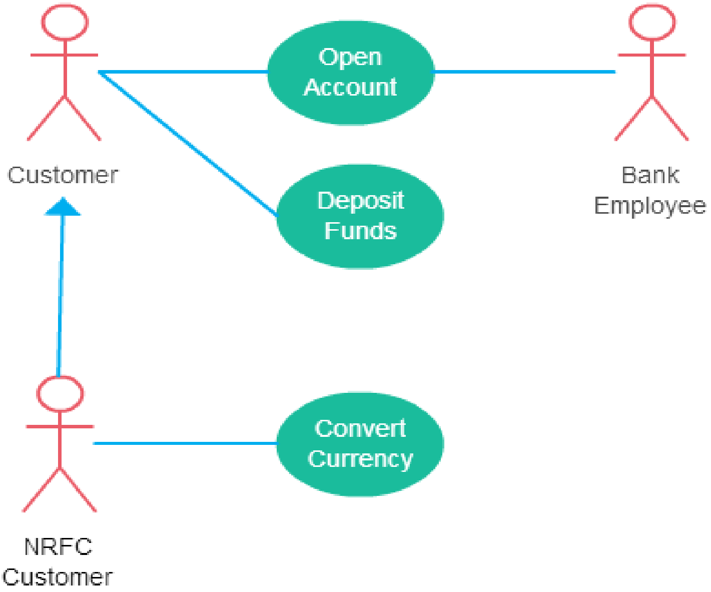
Include: When a use case is depicted as using the functionality of another use case in a diagram, this relationship between the use cases is named as an *include* relationship.



Extend: In an extended relationship between two use cases, the child use case adds to the existing functionality and characteristics of the parent use case.



Generalizations: A generalization relationship is also a parent-child relationship between use cases. The child use case in the generalization relationship has the underlying business process meaning, but is an enhancement of the parent use case.



4. System Boundaries:

   A system boundary defines the scope of what a system will be. A system cannot have infinite functionality. So, it follows that use cases also need to have definitive limits defined. A system boundary of a use case diagram defines the limits of the system. The system boundary is shown as a rectangle spanning all the use cases in the system.

Class diagram in software engineering.

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

The standard class diagram is composed of three sections:

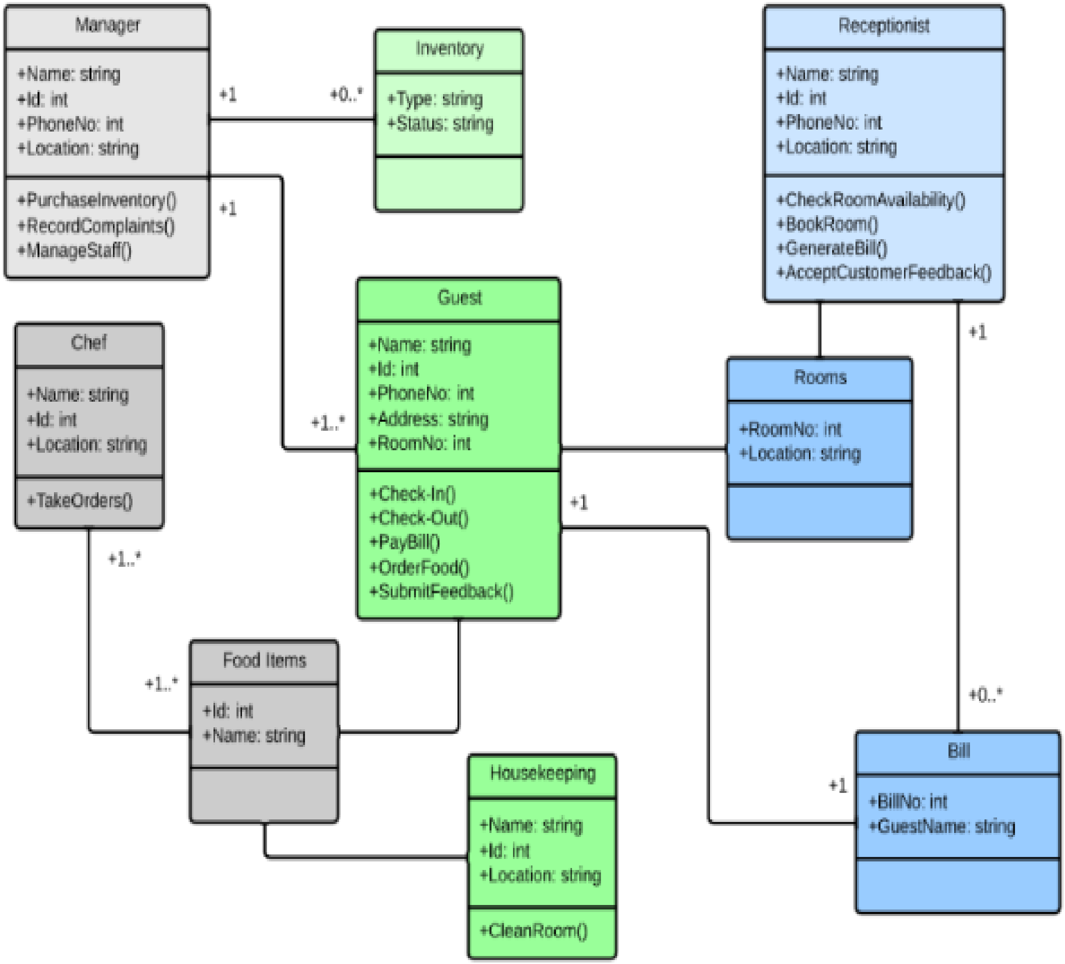
- Upper section: Contains the name of the class. This section is always required, whether you are talking about the classifier or an object.
- Middle section: Contains the attributes of the class. Use this section to describe the qualities of the class. This is only required when describing a specific instance of a class.
- Bottom section: Includes class operations (methods). Displayed in list format, each operation takes up its own line. The operations describe how a class interacts with data

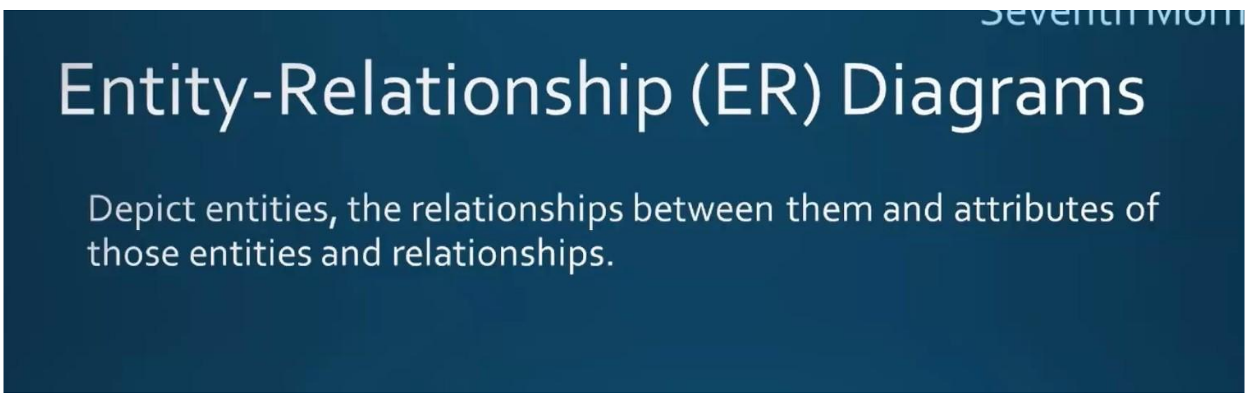
- Classes: A template for creating objects and implementing behavior in a system. In UML, a class represents an object or a set of objects that share a common structure and behavior. They're represented by a rectangle that includes rows of the class name, its attributes, and its operations. When you draw a class in a class diagram, you're only required to fill out the top row—the others are optional if you'd like to provide more detail.
  - Name: The first row in a class shape.
  - Attributes: The second row in a class shape. Each attribute of the class is displayed on a separate line.
  - Methods: The third row in a class shape. Also known as operations, methods are displayed in list format with each operation on its own line.
- Signals: Symbols that represent one-way, asynchronous communications between active objects.
- Data types: Classifiers that define data values. Data types can model both primitive types and enumerations.
- Packages: Shapes designed to organize related classifiers in a diagram. They are symbolized with a large tabbed rectangle shape.
- Interfaces: A collection of operation signatures and/or attribute definitions that define a cohesive set of behaviors. Interfaces are similar to classes, except that a class can have an instance of its type, and an interface must have at least one class to implement it.
- Enumerations: Representations of user-defined data types. An enumeration includes groups of identifiers that represent values of the enumeration.
- Objects: Instances of a class or classes. Objects can be added to a class diagram to represent either concrete or prototypical instances.
- Artifacts: Model elements that represent the concrete entities in a software system, such as

documents, databases, executable files, software components, etc.

- Interactions:
- Inheritance: The process of a child or sub-class taking on the functionality of a parent or superclass, also known as generalization.
- Bidirectional association: The default relationship between two classes. Both classes are aware of each other and their relationship with the other. This association is represented by a straight line between two classes.
- A slightly less common relationship between two classes. One class is aware of the other and interacts with it. Unidirectional association is modeled with a straight connecting line that points an open arrowhead from the knowing class to the known class.

Example of class diagram:



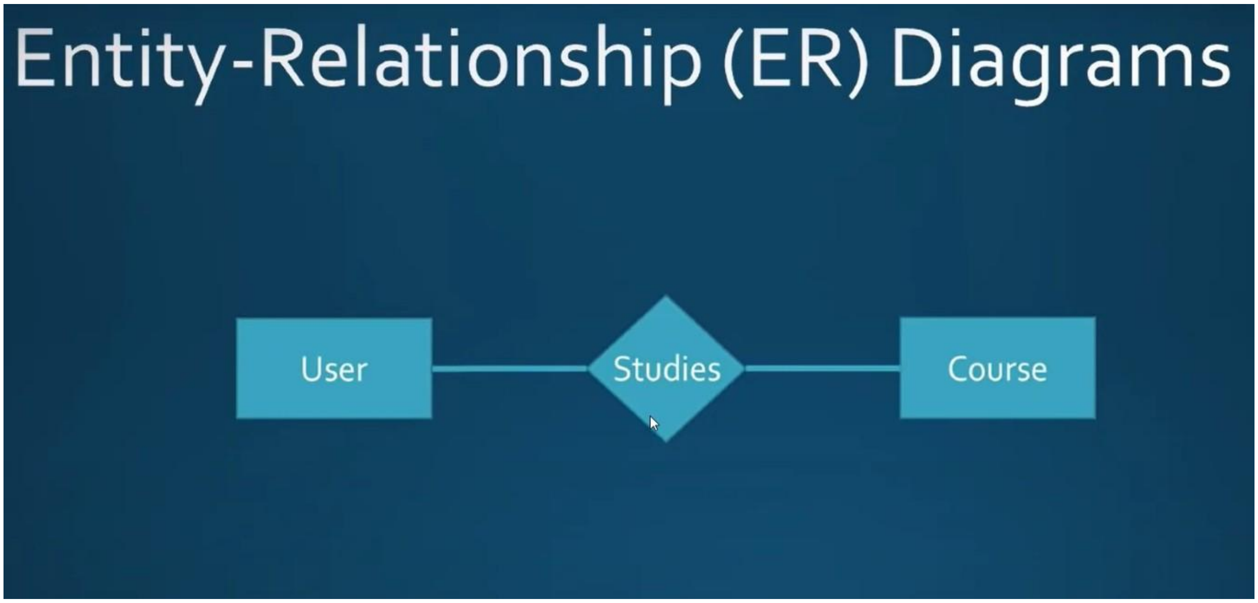


Seventh Mom

Entity-Relationship (ER) Diagrams

Depict entities, the relationships between them and attributes of those entities and relationships.

- A relationship is always between two rectangles:



Entity-Relationship (ER) Diagrams

User

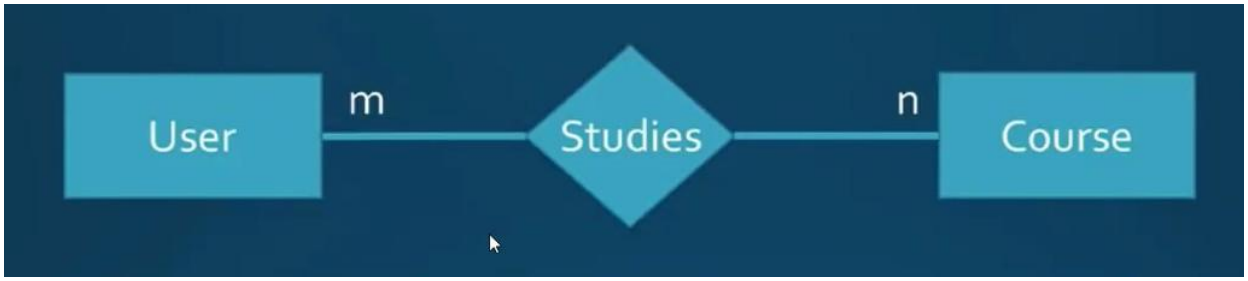
Studies

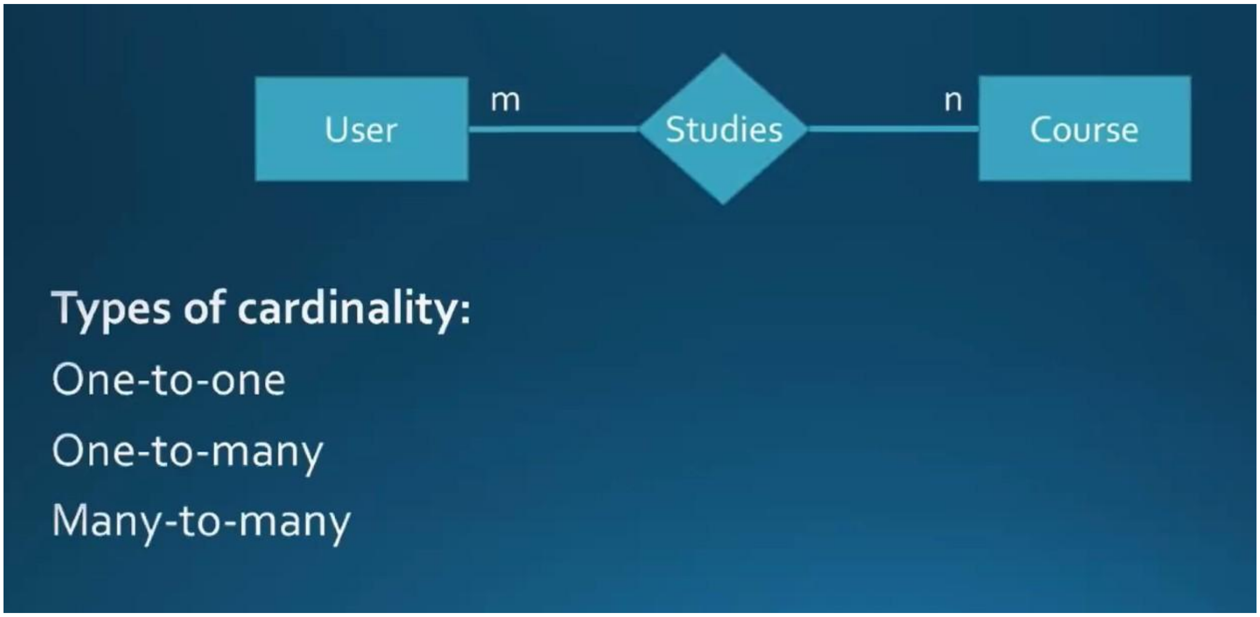
Course

Types of Cardinality:

- One - To - One
- One - To - Many
- Many - To - Many

- Cardinality: Any no of users can study any no of courses.



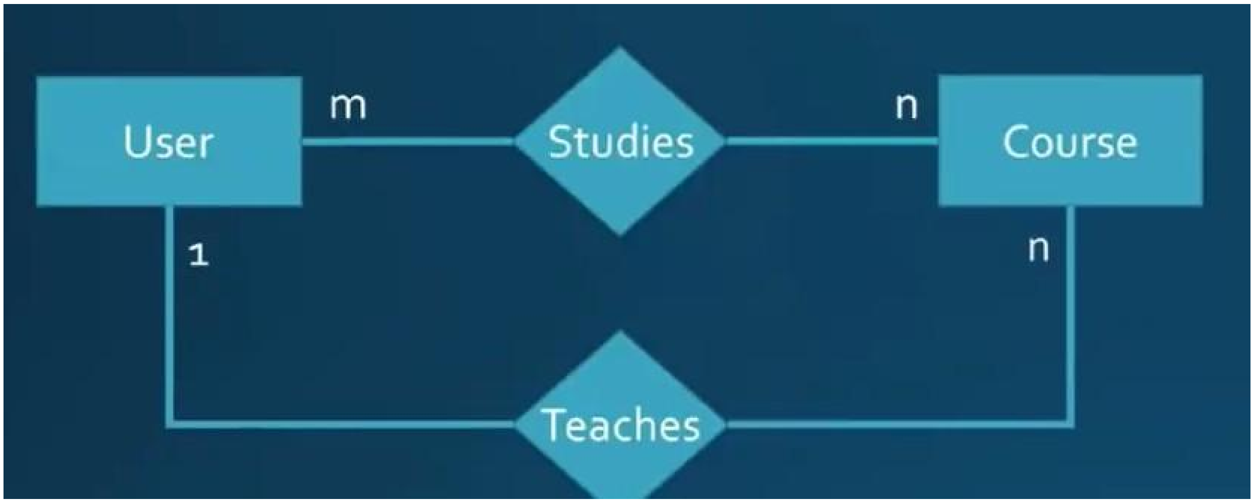


Types of cardinality:

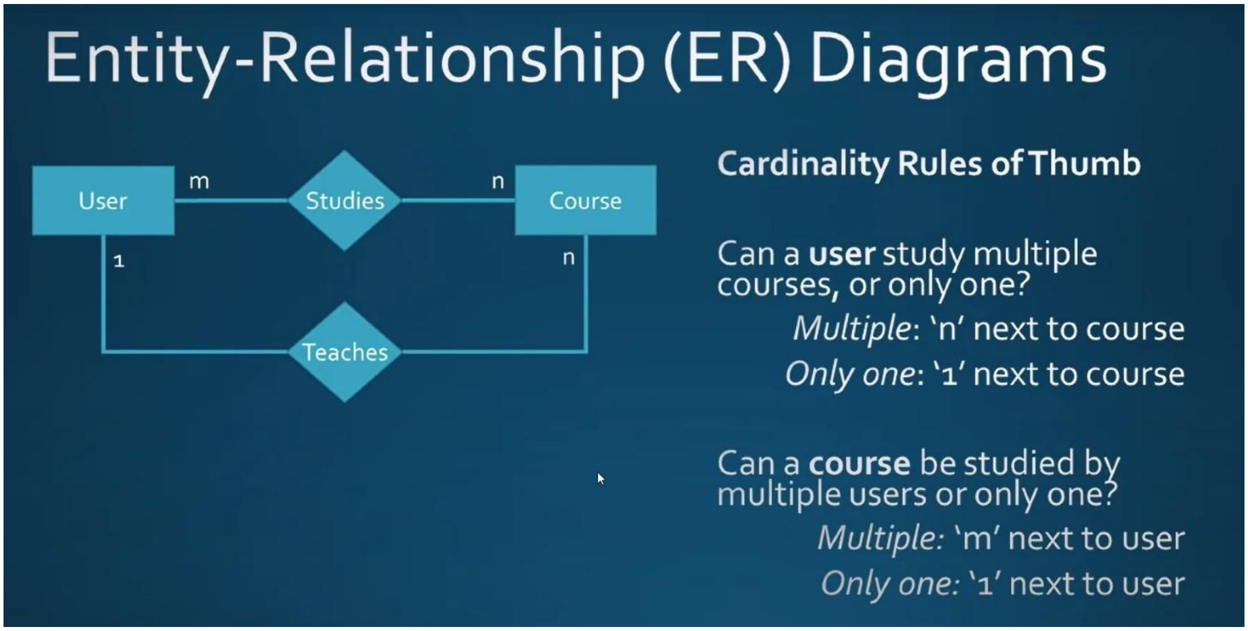
One-to-one

One-to-many

Many-to-many



- Entity - Relationship (ER) Diagrams:



Entity-Relationship (ER) Diagrams

Cardinality Rules of Thumb

Can a **user** study multiple courses, or only one?

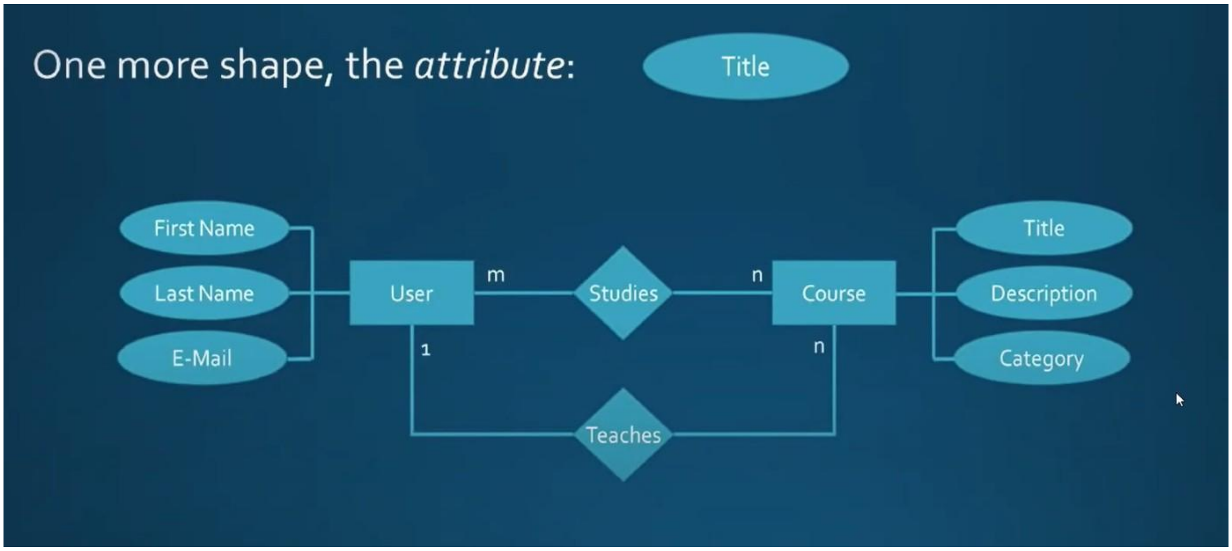
Multiple: 'n' next to course

Only one: '1' next to course

Can a **course** be studied by multiple users or only one?

Multiple: 'm' next to user

Only one: '1' next to user



One more shape, the *attribute*: