DBMS

- **Data** is nothing but facts and statistics stored or free flowing over a network, generally it's raw and unprocessed.
- For example: When you visit any website, they might store you IP address, that is data, in return they might add a cookie in your browser, marking you that you visited the website, that is data, your name, it's data, your age, it's data.
- Data becomes **information** when it is processed, turning it into something meaningful.

Database

- A **Database** is a collection of related data organised in a way that data can be easily accessed, managed and updated. Database can be software based or hardware based, with one sole purpose, **storing data**.

- During early computer days, data was collected and stored on tapes, which were mostly write-only, which means once data is stored on it, it can never be read again. They were slow and bulky, and soon computer scientists realised that they needed a better solution to this problem.

- A **DBMS** is a software that allows creation, definition and manipulation of database, allowing users to store, process and analyse data easily.
- DBMS provides us with an interface or a tool, to perform various operations like creating database, storing data in it, updating data, creating tables in the database and a lot more.
- DBMS also provides protection and security to the databases. It also maintains data consistency in case of multiple users.

- Here are some examples of popular DBMS used these days:
- MySql
- Oracle
- SQL Server
- IBM DB2
- PostgreSQL
- Amazon SimpleDB (cloud based) etc.
-

- characteristics:
- **Data stored into Tables:**
- Data is never directly stored into the database. Data is stored into tables, created inside the database.
- DBMS also allows to have relationships between tables which makes the data more meaningful and connected.

- **Reduced Redundancy:** In the modern world hard drives are very cheap, but earlier when hard drives were too expensive, unnecessary repetition of data in database was a big problem. But DBMS follows **Normalisation** which divides the data in such a way that repetition is minimum.

- **Data Consistency:** On Live data, i.e. data that is being continuously updated and added, maintaining the consistency of data can become a challenge. But DBMS handles it all by itself.

- **Support Multiple user and Concurrent Access:** DBMS allows multiple users to work on it(update, insert, delete data) at the same time and still manages to maintain the data consistency.

- **Query Language:** DBMS provides users with a simple Query language, using which data can be easily fetched, inserted, deleted and updated in a database.

- **Security:** The DBMS also takes care of the security of data, protecting the data from un-authorised access. In a typical DBMS, we can create user accounts with different access permissions, using which we can easily secure our data by restricting user access.

- DBMS supports **transactions**, which allows us to better handle and manage data integrity in real world applications where multi-threading is extensively used.

Advantages of DBMS

- Segregation of application program.
- Minimal data duplicity or data redundancy.
- Easy retrieval of data using the Query Language.
- Reduced development time and maintenance need.
- With Cloud Datacenters, we now have Database Management Systems capable of storing almost infinite data.
- Seamless(all in one) integration into the application programming languages which makes it very easier to add a database to almost any application or website.

Disadvantages of DBMS

- It's Complexity
- Except MySQL, which is open source, licensed DBMSs are generally costly.
- They are large in size.

- Components of DBMS
- The database management system can be divided into five major components, they are:
- Hardware
- Software
- Data
- Procedures
- Database Access Language

- DBMS Components: Hardware
- When we say Hardware, we mean computer, hard disks, I/O channels for data, and any other physical component involved before any data is successfully stored into the memory.
- When we run Oracle or MySQL on our personal computer, then our computer's Hard Disk, our Keyboard using which we type in all the commands, our computer's RAM, ROM all become a part of the DBMS hardware.
-

- DBMS Components: Software
- This is the main component, as this is the program which controls everything.
- The DBMS software is more like a wrapper around the physical database, which provides us with an easy-to-use interface to store, access and update data.

- The DBMS software is capable of understanding the Database Access Language and interpret it into actual database commands to execute them on the DB.
-

- DBMS Components:
- Data
- Data is that resource, for which DBMS was designed. The motive behind the creation of DBMS was to store and utilise data.
- In a typical Database, the user saved Data is present and **meta data** is stored.
- **Metadata** is data about the data. This is information stored by the DBMS to better understand the data stored in it
- .
- **For example:** When I store my **Name** in a database, the DBMS will store when the name was stored in the database, what is the size of the name, is it stored as related data to some other data, or is it independent, all this information is metadata.
-

- DBMS Components:
- Procedures
- Procedures refer to general instructions to use a database management system. This includes procedures to setup and install a DBMS, To login and logout of DBMS software, to manage databases, to take backups, generating reports etc.

- DBMS Components:
- Database Access Language
- Database Access Language is a simple language designed to write commands to access, insert, update and delete data stored in any database.
- A user can write commands in the Database Access Language and submit it to the DBMS for execution, which is then translated and executed by the DBMS.
- User can create new databases, tables, insert data, fetch stored data, update data and delete the data using the access language.

- Users
- **Database Administrators:** Database Administrator or DBA is the one who manages the complete database management system. DBA takes care of the security of the DBMS, it's availability, managing the license keys, managing user accounts and access etc.

- **Application Programmer or Software Developer:** This user group is involved in developing and designing the parts of DBMS.

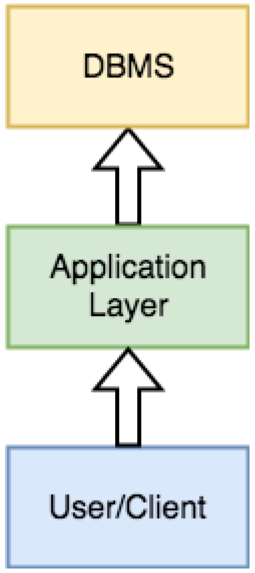
- **End User:** These days all the modern applications, web or mobile, store user data. How do you think they do it? Yes, applications are programmed in such a way that they collect user data and store the data on DBMS systems running on their server. End users are the one who store, retrieve, update and delete data.
-

DBMS Architecture

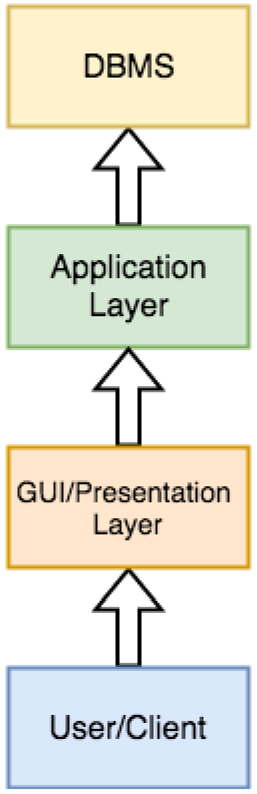
- A Database Management system is not always directly available for users and applications to access and store data in it. A Database Management system can be **centralised**(all the data stored at one location), **decentralised**(multiple copies of database at different locations) or **hierarchical**, depending upon its architecture.
- **1-tier DBMS** architecture also exist, this is when the database is directly available to the user for using it to store data. Generally such a setup is used for local application development, where programmers communicate directly with the database for quick response.
- Database Architecture is logically of two types:
- 2-tier DBMS architecture
- 3-tier DBMS architecture

- 2-tier DBMS Architecture
- 2-tier DBMS architecture includes an **Application layer** between the user and the DBMS, which is responsible to communicate the user's request to the database management system and then send the response from the DBMS to the user.
- An application interface known as **ODBC**(Open Database Connectivity) provides an API that allow client side program to call the DBMS. Most DBMS vendors provide ODBC drivers for their DBMS.

- Such an architecture provides the DBMS extra security as it is not exposed to the End User directly. Also, security can be improved by adding security and authentication checks in the Application layer too.



- 3-tier DBMS Architecture
- 3-tier DBMS architecture is the most commonly used architecture for web applications.

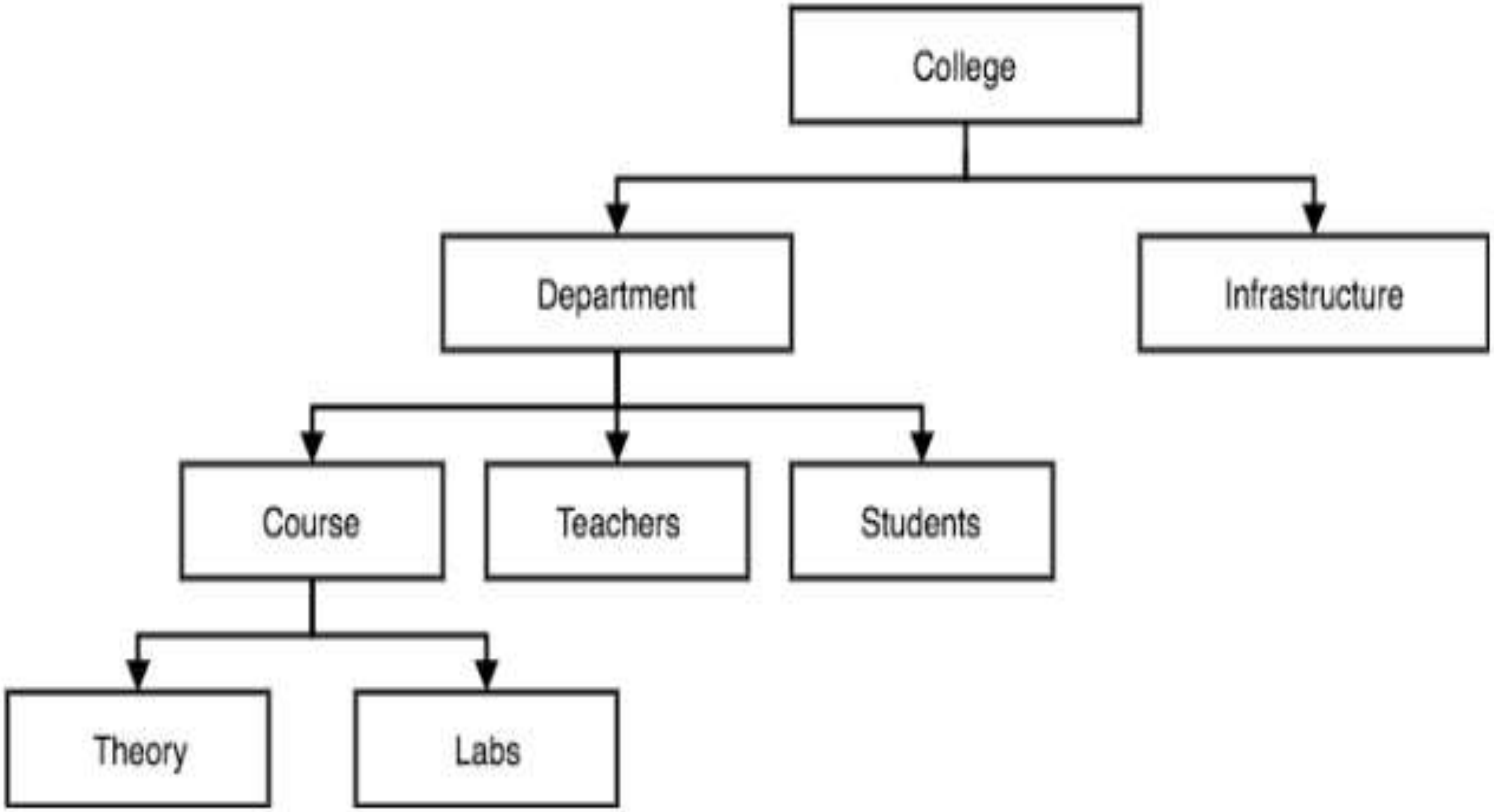


- It is an extension of the 2-tier architecture. In the 2-tier architecture, we have an application layer which can be accessed programmatically to perform various operations on the DBMS. The application generally understands the Database Access Language and processes end users requests to the DBMS.
- In 3-tier architecture, an additional Presentation or GUI Layer is added, which provides a graphical user interface for the End user to interact with the DBMS.
- For the end user, the GUI layer is the Database System, and the end user has no idea about the application layer and the DBMS system.
- If you have used **MySQL**, then you must have seen **PHPMyAdmin**, it is the best example of a 3-tier DBMS architecture.

- DBMS Database Models
- A Database model defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system. While the **Relational Model** is the most widely used database model, there are other models too:
- Hierarchical Model
- Network Model
- Entity-relationship Model
- Relational Model

Hierarchical Model

- This database model organises data into a tree-like-structure, with a single root, to which all the other data is linked.
- The heirarchy starts from the **Root** data, and expands like a tree, adding child nodes to the parent nodes.
- In this model, a child node will only have a single parent node.
- This model efficiently describes many real-world relationships like index of a book, recipes etc.
- In hierarchical model, data is organised into tree-like structure with one one-to-many relationship between two different types of data, for example, one department can have many courses, many professors and of-course many students.
-

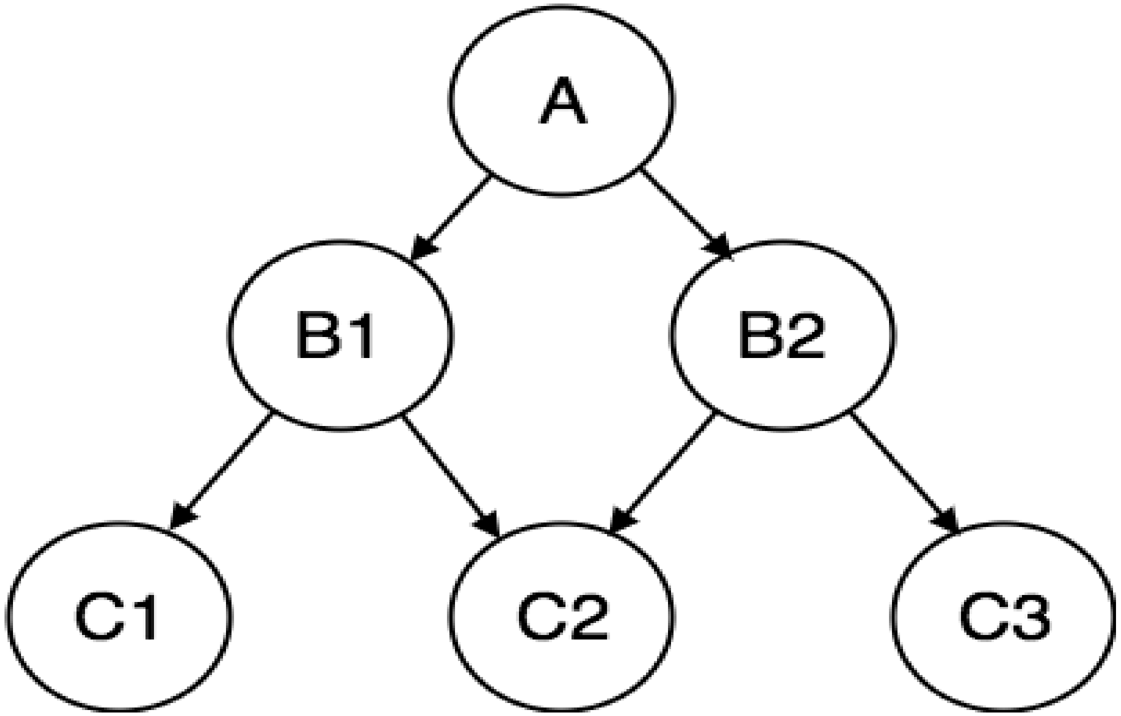

College
Department
Infrastructure
Course
Teachers
Students
Theory
Labs

Network Model

This is an extension of the Hierarchical model. In this model data is organised more like a graph, and are allowed to have more than one parent node.

In this database model data is more related as more relationships are established in this database model. Also, as the data is more related, hence accessing the data is also easier and fast. This database model was used to map many-to-many data relationships.

This was the most widely used database model, before Relational Model was introduced.


$$\begin{tikzcd} & & A \arrow[dl] \arrow[dr] & & \\ & B1 \arrow[dl] \arrow[dr] & & B2 \arrow[dl] \arrow[dr] & \\ C1 & & C2 & & C3 \end{tikzcd}$$

Entity-relationship Model

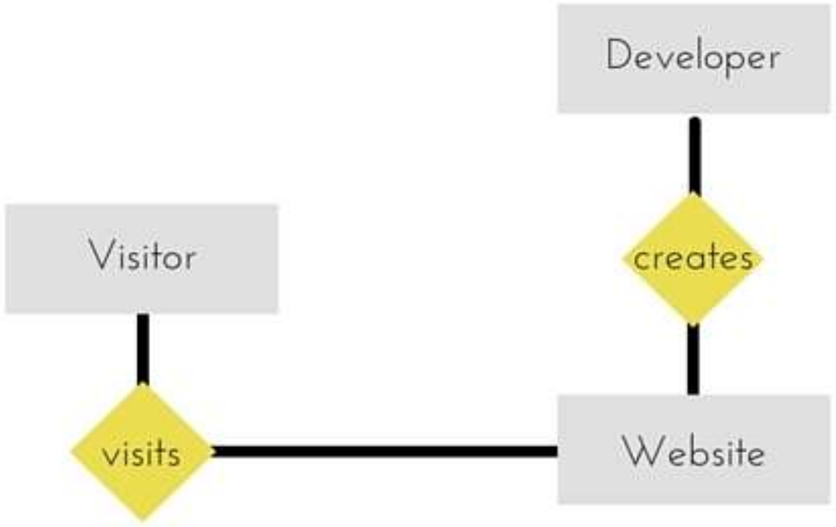
- In this database model, relationships are created by dividing object of interest into entity and its characteristics into attributes.

- Different entities are related using relationships.
- E-R Models are defined to represent the relationships into pictorial form to make it easier for different stakeholders to understand.

- This model is good to design a database, which can then be turned into tables in relational model(explained below).

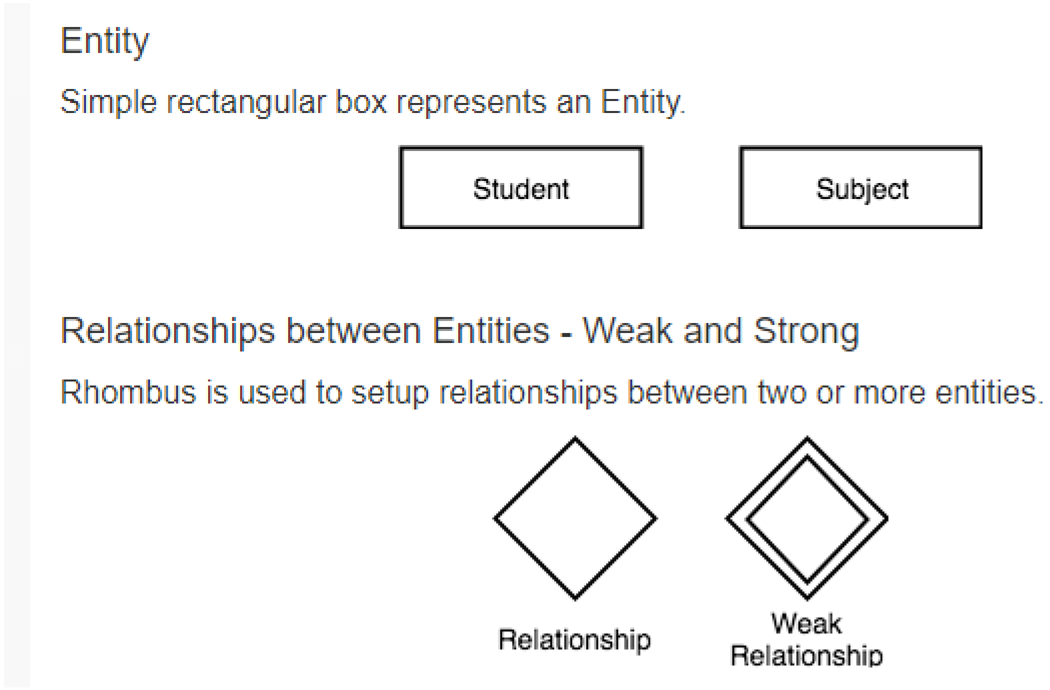
- Let's take an example, If we have to design a School Database, then **Student** will be
  an **entity** with **attributes** name, age, address etc.
  As **Address** is generally complex, it can be
  another **entity** with **attributes** street name, pincode, city etc, and there will be a relationship between them.

- Relationships can also be of different types
- Working with ER Diagrams to know the details of relationships ….
- ER Diagram is a visual representation of data that describes how <u>data is related to </u>each other.
- In ER Model, we disintegrate data into entities, attributes and setup relationships between entities, all this can be represented visually using the ER diagram.


Developer
creates
Visitor
visits
Website

Components of ER Diagram

- Entitiy, Attributes, Relationships etc form the components of ER Diagram and there are defined symbols and shapes to represent each one of them.
- Let's see how we can represent these in our ER Diagram.



Entity

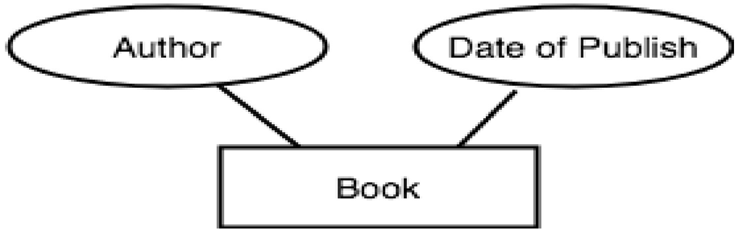
Simple rectangular box represents an Entity.

Relationships between Entities - Weak and Strong

Rhombus is used to setup relationships between two or more entities.

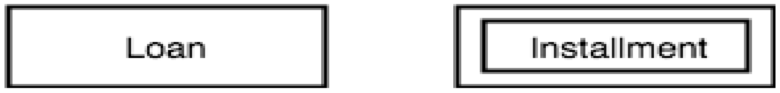
Attributes for any Entity

Ellipse is used to represent attributes of any entity. It is connected to the entity.



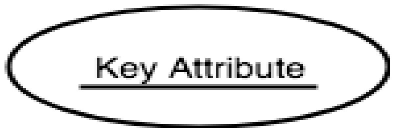
Weak Entity

A weak Entity is represented using double rectangular boxes. It is generally connected to another entity.



Key Attribute for any Entity

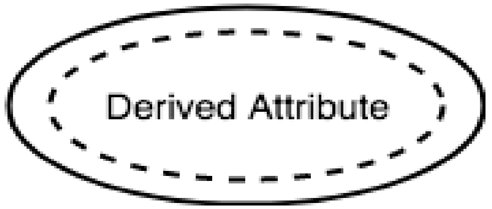
To represent a Key attribute, the attribute name inside the Ellipse is underlined.



Derived Attribute for any Entity

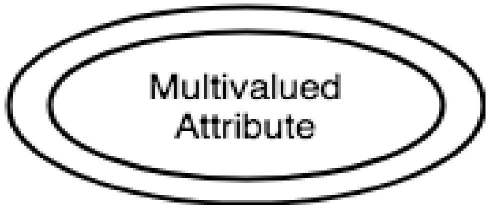
Derived attributes are those which are derived based on other attributes, for example, age can be derived from date of birth.

To represent a derived attribute, another dotted ellipse is created inside the main ellipse.



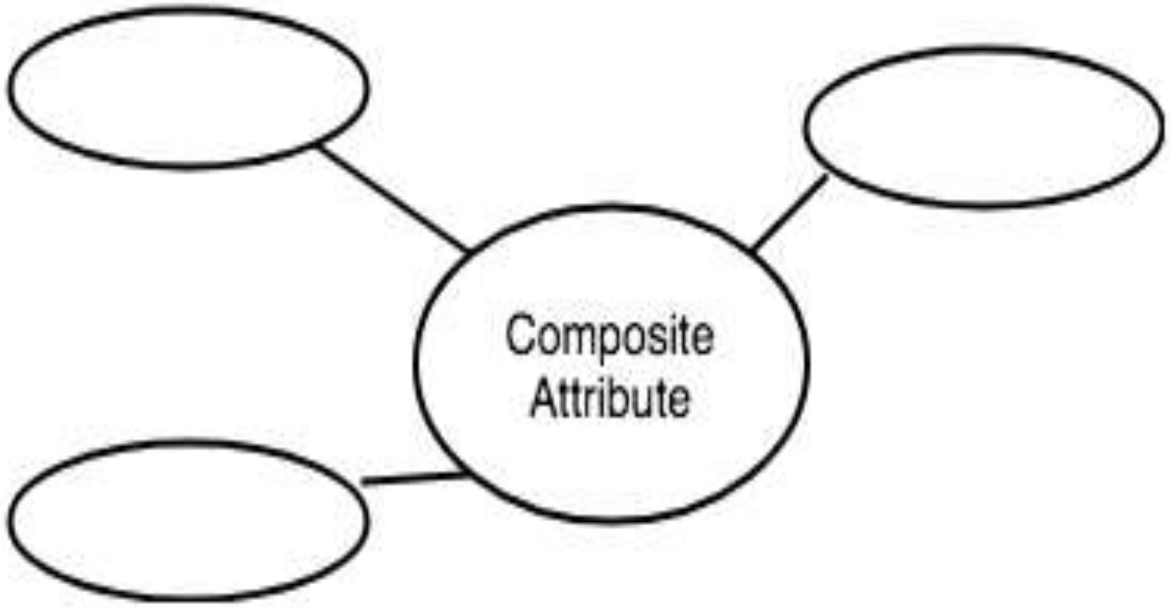
Multivalued Attribute for any Entity

Double Ellipse, one inside another, represents the attribute which can have multiple values.



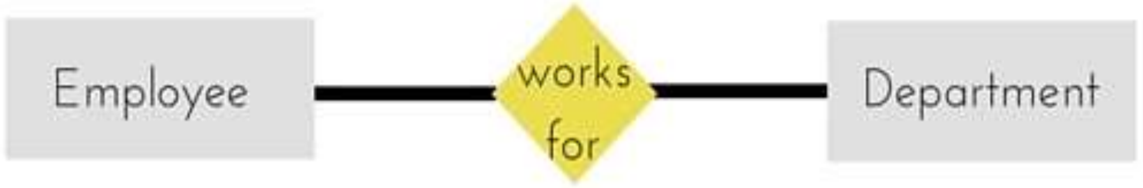
Composite Attribute for any Entity

A composite attribute is the attribute, which also has attributes.



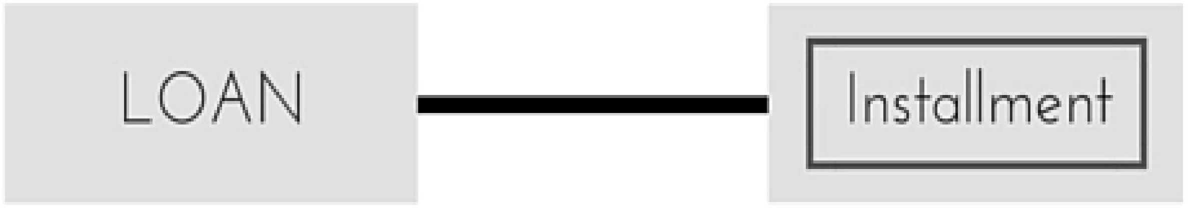
ER Diagram: Entity

An **Entity** can be any object, place, person or class. In ER Diagram, an **entity** is represented using rectangles. Consider an example of an Organisation- Employee, Manager, Department, Product and many more can be taken as entities in an Organisation.



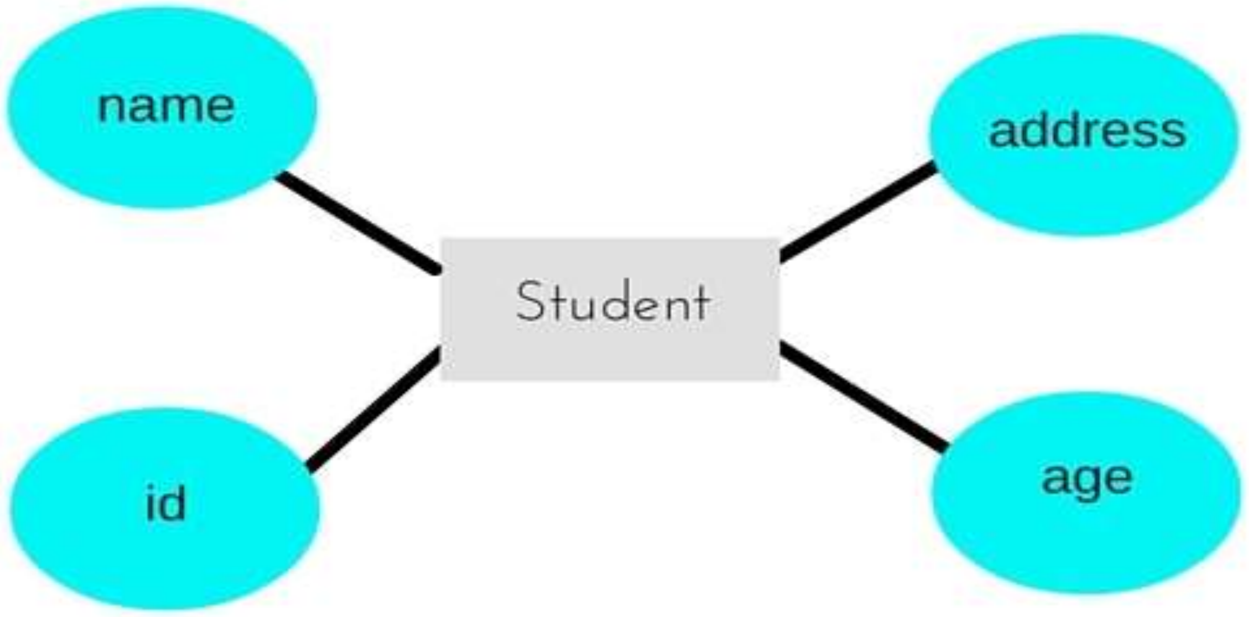
ER Diagram: Weak Entity

Weak entity is an entity that depends on another entity. Weak entity doesn't have anay key attribute of its own. Double rectangle is used to represent a weak entity.



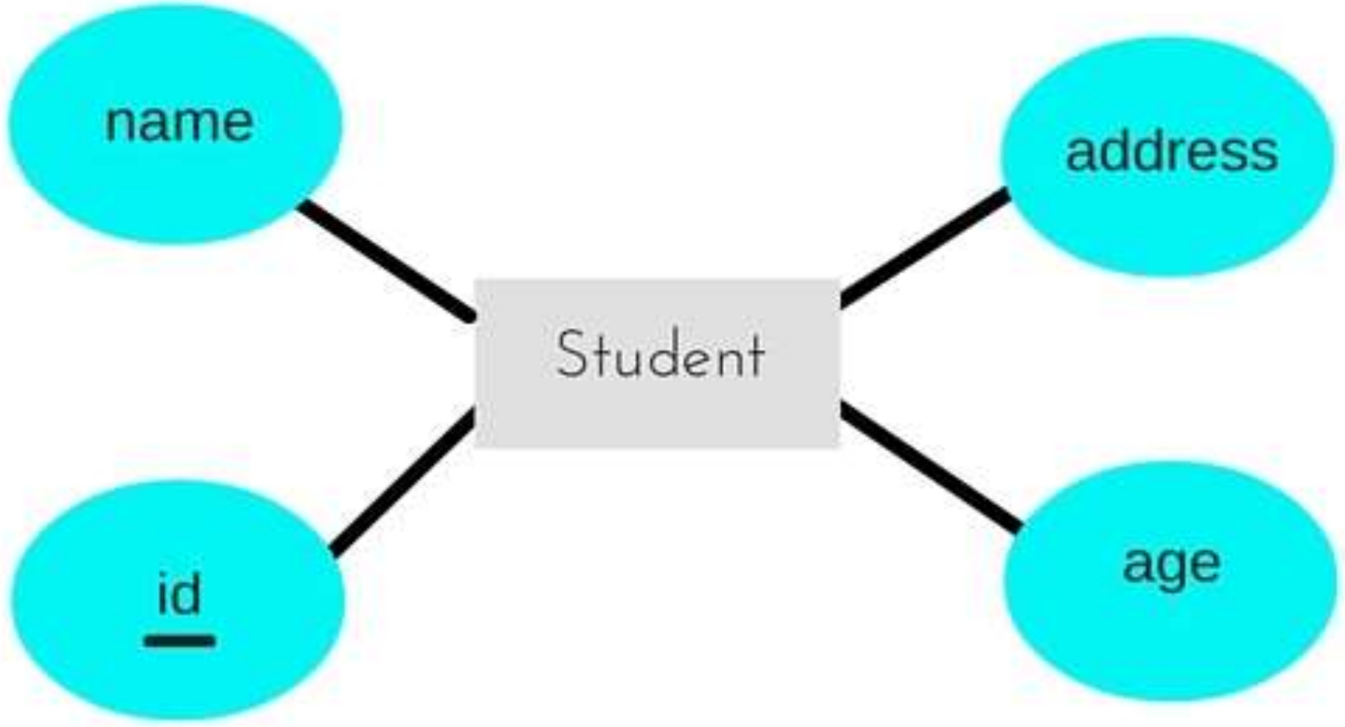
ER Diagram: Attribute

An **Attribute** describes a property or characterstic of an entity. For example, **Name**, **Age**, **Address** etc can be attributes of a **Student**. An attribute is represented using eclipse.



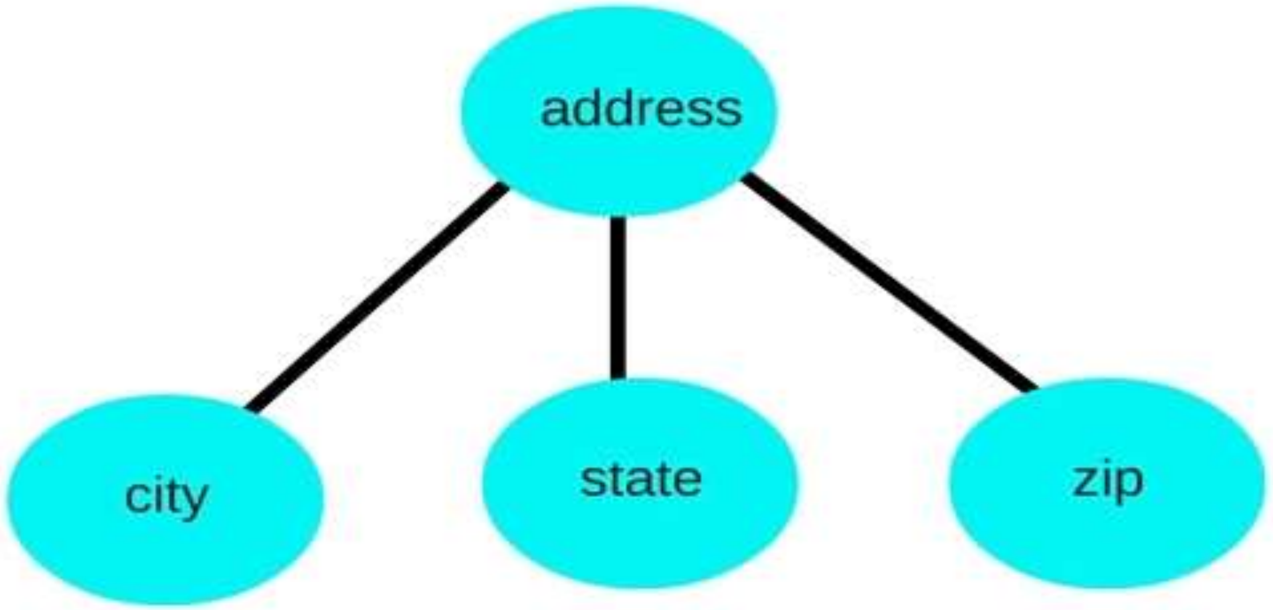
ER Diagram: Key Attribute

Key attribute represents the main characterstic of an Entity. It is used to represent a Primary key. Ellipse with the text underlined, represents Key Attribute.



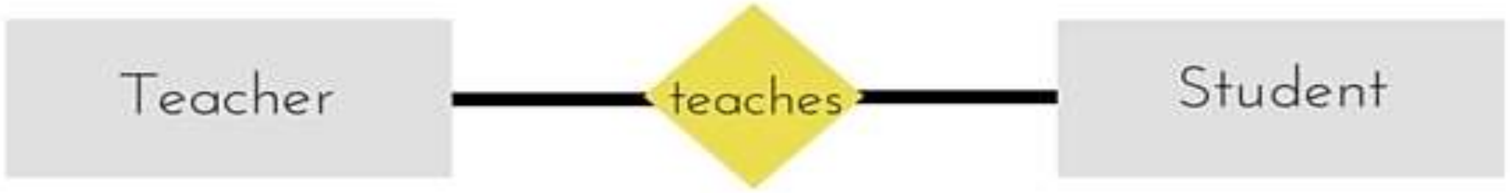
ER Diagram: Composite Attribute

An attribute can also have their own attributes. These attributes are known as **Composite** attributes.



ER Diagram: Relationship

A Relationship describes relation between **entities**. Relationship is represented using diamonds or rhombus.



There are three types of relationship that exist between Entities.

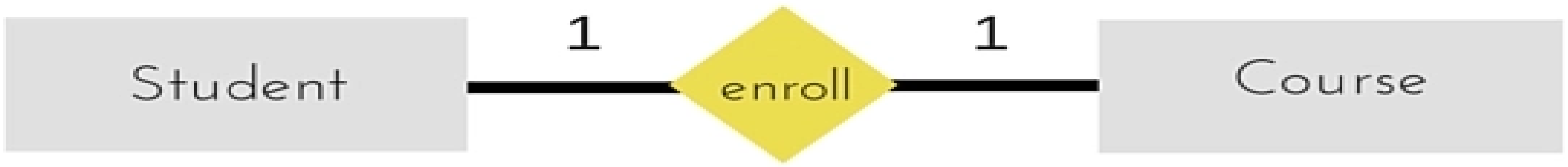
1. Binary Relationship
2. Recursive Relationship
3. Ternary Relationship

ER Diagram: Binary Relationship

Binary Relationship means relation between two Entities. This is further divided into three types.

One to One Relationship

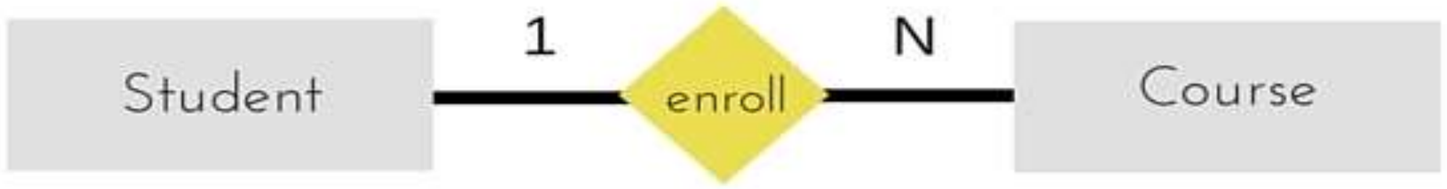
This type of relationship is rarely seen in real world.



The above example describes that one student can enroll only for one course and a course will also have only one Student. This is not what you will usually see in real-world relationships.

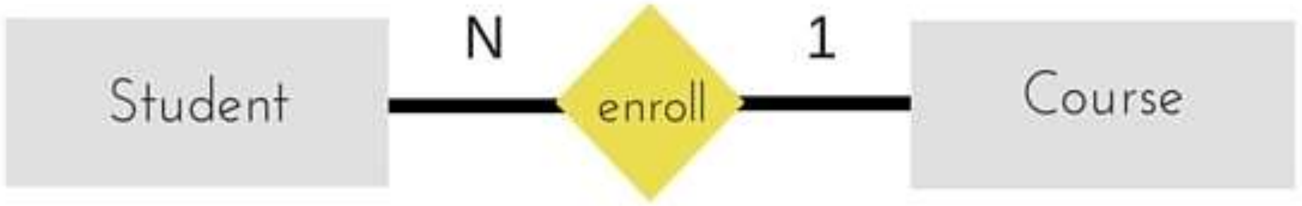
One to Many Relationship

The below example showcases this relationship, which means that 1 student can opt for many courses, but a course can only have 1 student. Sounds weird! This is how it is.

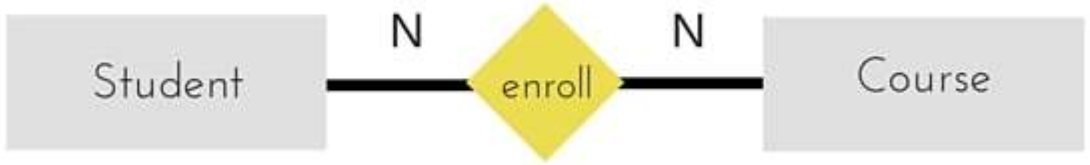


Many to One Relationship

It reflects business rule that many entities can be associated with just one entity. For example, Student enrolls for only one Course but a Course can have many Students.



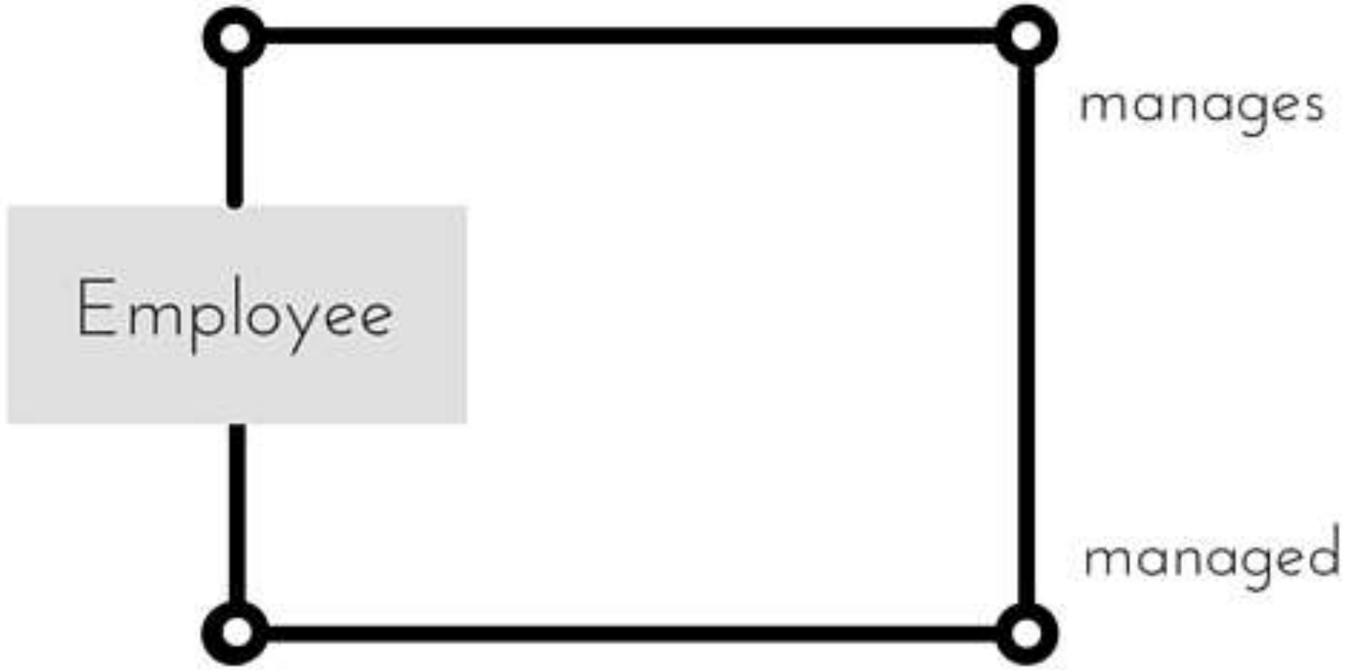
Many to Many Relationship



The above diagram represents that one student can enroll for more than one courses. And a course can have more than 1 student enrolled in it.

ER Diagram: Recursive Relationship

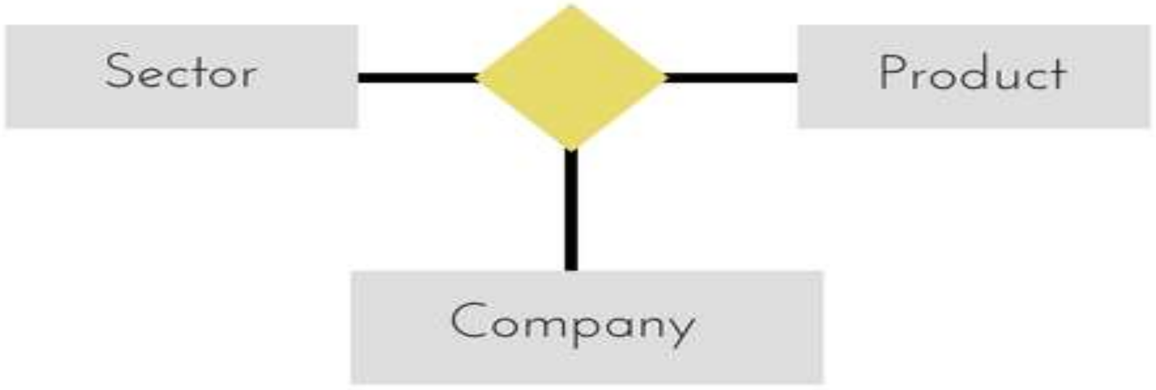
When an Entity is related with itself it is known as **Recursive** Relationship.



ER Diagram: Ternary Relationship

Relationship of degree three is called Ternary relationship.

A Ternary relationship involves three entities. In such relationships we always consider two entites together and then look upon the third.



- The above relationship involves 3 entities.
- Company operates in Sector, producing some Products.

- For example, in the diagram above, we have three related entities, **Company**, **Product** and **Sector**. To understand the relationship better or to define rules around the model, we should relate two entities and then derive the third one.
- A **Company** produces many **Products**/ each product is produced by exactly one company.
- A **Company** operates in only one **Sector** / each sector has many companies operating in it.
- Considering the above two rules or relationships, we see that although the complete relationship involves three entities, but we are looking at two entities at a time.

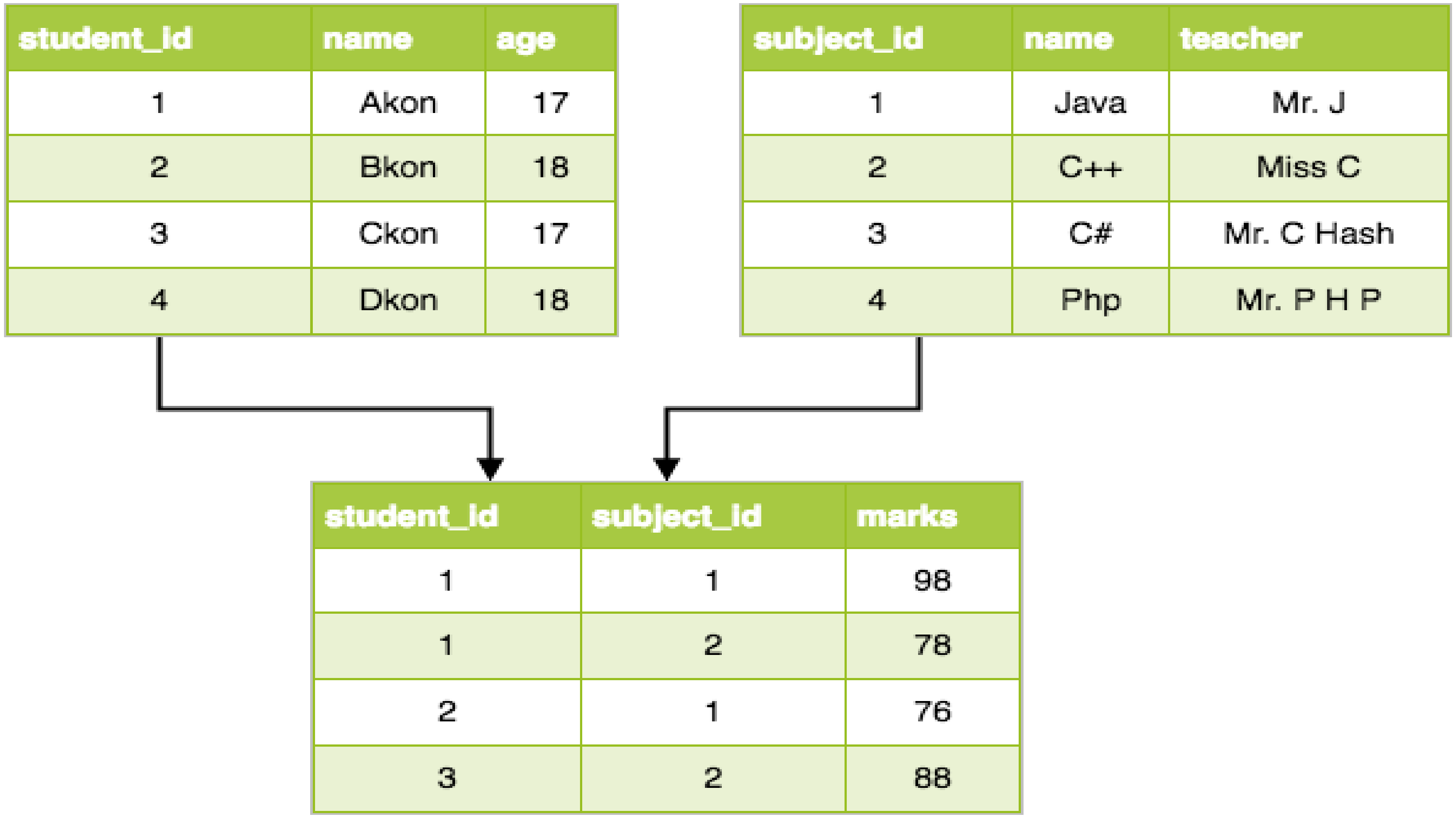
Relational Model

In this model, data is organised in two-dimensional **tables** and the relationship is maintained by storing a common field.

This model was introduced by E.F Codd in 1970, and since then it has been the most widely used database model, infact, we can say the only database model used around the world.

The basic structure of data in the relational model is tables. All the information related to a particular type is stored in rows of that table.

Hence, tables are also known as **relations** in relational model.



student_id	name	age
1	Akon	17
2	Bkon	18
3	Ckon	17
4	Dkon	18

subject_id	name	teacher
1	Java	Mr. J
2	C++	Miss C
3	C#	Mr. C Hash
4	Php	Mr. P H P

student_id	subject_id	marks
1	1	98
1	2	78
2	1	76
3	2	88

Basic Concepts of ER Model in DBMS

- Entity-relationship model is a model used for design and representation of relationships between data.
- The main data objects are termed as Entities, with their details defined as attributes, some of these attributes are important and are used to identity the entity, and different entities are related using relationships.
- In short, to understand about the ER Model, we must understand about:
- Entity and Entity Set
- What are Attributes? And Types of Attributes.
- Keys
- Relationships

Entity and Entity Set

- Considering the above example, **Student** is an entity, **Teacher** is an entity, similarly, **Class**, **Subject** etc are also entities.
- An Entity is generally a real-world object which has characteristics and holds relationships in a DBMS.

- <u>If a Student is an Entity, then the complete dataset of all the students will be the **Entity Set**</u>

Attributes

- If a Student is an Entity, then student's **roll no.**, student's **name**, student's **age**, student's **gender** etc will be its attributes.

- An attribute can be of many types, here are different types of attributes defined in ER database model:

- **Simple attribute:** The attributes with values that are atomic and cannot be broken down further are simple attributes. For example, student's **age**.

- **Composite attribute:** A composite attribute is made up of more than one simple attribute.
- For example, student's **address** will contain**, house no.**, **street name**, **pincode** etc.

- **Derived attribute:** These are the attributes which are not present in the whole database management system, but are derived using other attributes. For example, *average age of students in a class*.

- **Single-valued attribute:** As the name suggests, they have a single value.

- **Multi-valued attribute:** And, they can have multiple values.

Keys

- If the attribute **roll no.** can uniquely identify a student entity, amongst all the students, then the attribute **roll no.** will be said to be a key.

- Following are the types of Keys:
- Super Key
- Candidate Key
- Primary Key

- Keys are very important part of Relational database model.
- They are used to establish and identify relationships between tables and also to <u>uniquely identify any record </u>or row of data inside a table.

- A Key can be a single attribute or a group of attributes, where the combination may act as a key.

Need of a key

- In real world applications, number of tables required for storing the data is huge, and the different tables are related to each other as well.
- Also, tables store a lot of data in them. Tables generally extends to thousands of records stored in them, unsorted and unorganised.
- Now to fetch any particular record from such dataset, you will have to apply some conditions, but what if there is duplicate data present and every time you try to fetch some data by applying certain condition, you get the wrong data. How many trials before you get the right data?
- To avoid all this, **Keys** are defined to easily identify any row of data in a table.

Let's take a simple **Student** table, with fields student_id, name, phone and age.

student_id	name	phone	age
1	Akon	9876723452	17
2	Akon	9991165674	19
3	Bkon	7898756543	18
4	Ckon	8987867898	19
5	Dkon	9990080080	17

Super Key

- **Super Key** is defined as a set of attributes within a table that can uniquely identify each record within a table. Super Key is a superset of Candidate key.
- In the table defined above super key would

- include student_id, **(student_id**, name), phone etc.

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

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

- Similarly, phone number for every student will be unique, hence again, phone can also be a key.
- So they all are super keys.

Candidate Key

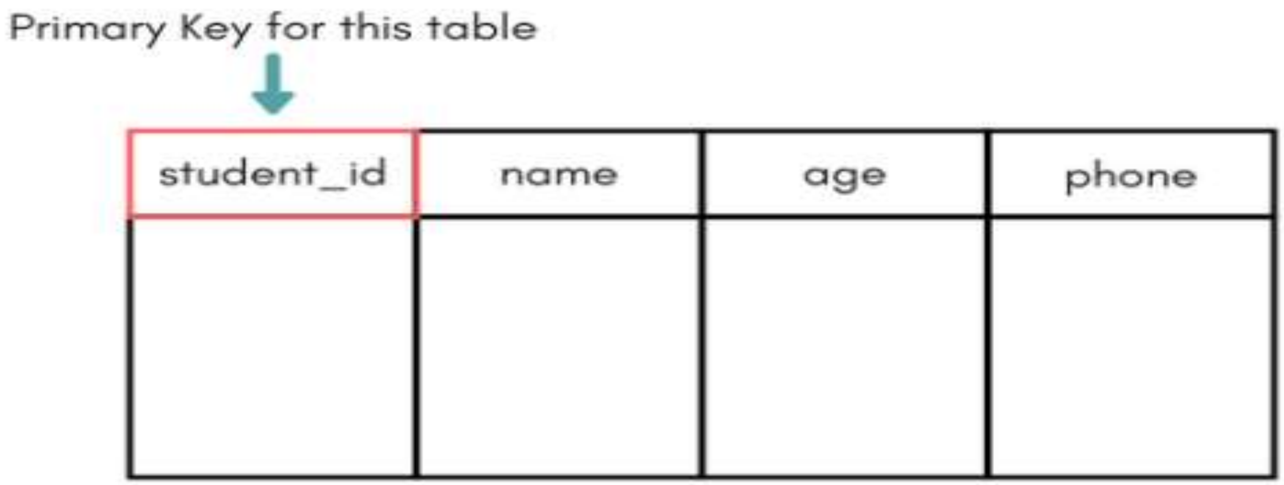
- Candidate keys are defined as the minimal set of fields which can uniquely identify each record in a table.

- It is an attribute or a set of attributes that can act as a Primary Key for a table to uniquely identify each record in that table. There can be more than one candidate key.

- In our example, student_id and phone both are candidate keys for table **Student**.
- A candiate key can never be NULL or empty. And its value should be unique.
- There can be more than one candidate keys for a table.
- A candidate key can be a combination of more than one columns(attributes).

Primary Key

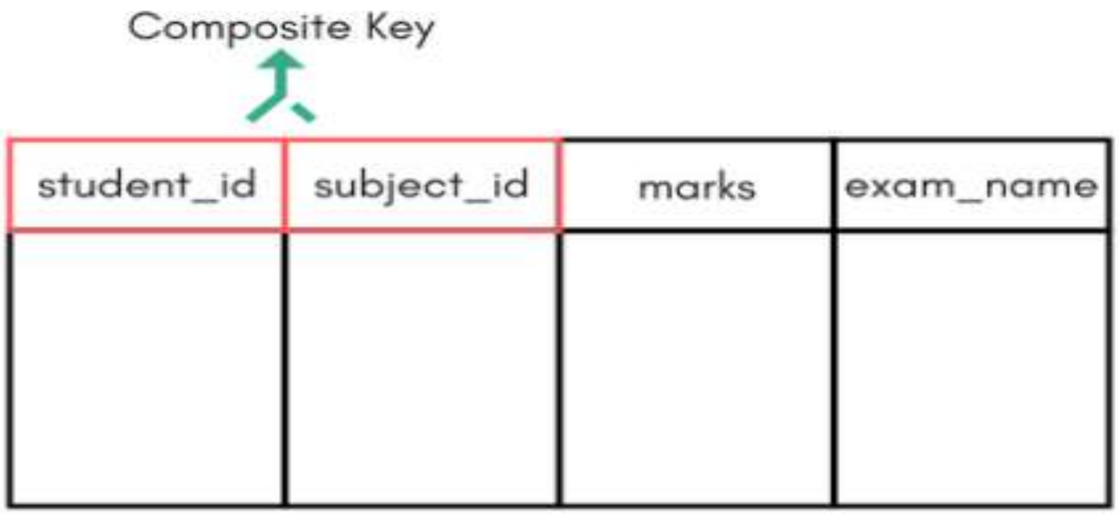
- Primary key is a candidate key that is most appropriate to become the main key for any table. It is a key that can uniquely identify each record in a table.



For the table **Student** we can make the `student_id` column as the primary key.

Composite Key

Key that consists of two or more attributes that uniquely identify any record in a table is called **Composite key**. But the attributes which together form the **Composite key** are not a key independentely or individually.



Score Table - To save scores of the student for various subjects.

In the above picture we have a **Score** table which stores the marks scored by a student in a particular subject.

In this table `student_id` and `subject_id` together will form the primary key, hence it is a composite key.

Secondary or Alternative key

The candidate key which are not selected as primary key are known as secondary keys or alternative keys.

Non-key Attributes

Non-key attributes are the attributes or fields of a table, other than **candidate key** attributes/fields in a table.

Non-prime Attributes

Non-prime Attributes are attributes other than **Primary Key attribute(s)**..

NORMALIZATION

- Database Normalization is a technique of organizing the data in the database.
- Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition) and
- undesirable characteristics like Insertion, Update and Deletion Anomalies(differences).
- It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.
- Normalization is used for mainly two purposes,
- 1.Eliminating redundant(useless) data.
- 2.Ensuring data dependencies make sense i.e data is logically stored.

Problems Without Normalization

- If a table is not properly normalized and have data redundancy then it will not only eat up extra memory space
- but will also make it difficult to handle and update the database, without facing data loss.
- Insertion, Updation and Deletion Anomalies are very frequent if database is not normalized.
- To understand these anomalies let us take an example of a **Student** table.

rollno	name	branch	hod	office_tel
401	Akon	CSE	Mr. X	53337
402	Bkon	CSE	Mr. X	53337
403	Ckon	CSE	Mr. X	53337
404	Dkon	CSE	Mr. X	53337

- In the table above, we have data of 4 Computer Sci. students. As we can see, data for the fields branch, hod(Head of Department) and office_tel is repeated for the students who are in the same branch in the college, this is **Data Redundancy**.

- Insertion Anomaly
- Suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted, or else we will have to set the branch information as **NULL**.
- Also, if we have to insert data of 100 students of same branch, then the branch information will be repeated for all those 100 students.
- These scenarios are nothing but **Insertion anomalies**.

- Updation Anomaly
- What if Mr. X leaves the college? or is no longer the HOD of computer science department? In that case all the student records will have to be updated, and if by mistake we miss any record, <u>it will lead to data inconsistency</u>. This is Updation anomaly.
-

- Deletion Anomaly
- In our **Student** table, two different informations are kept together, Student information and Branch information. Hence, at the end of the academic year, if student records are deleted, we will <u>also lose the branch information.</u> This is Deletion anomaly.

- Normalization Rule
- Normalization rules are divided into the following normal forms:

- 1. First Normal Form
- 2 . Second Normal Form
- 3. Third Normal Form
- 4. BCNF
- 5. Fourth Normal Form

First Normal Form (1NF)

- For a table to be in the First Normal Form, it should follow the following 4 rules:

1. It should only have single(atomic) valued attributes/columns.
2. Values stored in a column should be of the same domain
3. All the columns in a table should have <u>unique names.</u>
4. And the order in which data is stored, does not matter.

rules for First Normal Form

- The first normal form expects you to follow a few simple rules while designing your database, and they are:
-

  Rule 1: Single Valued Attributes
- Each column of your table should be single valued which means they should not contain multiple values. We will explain this with help of an example later, let's see the other rules for now.

- Rule 2: Attribute Domain should not change
- This is more of a "Common Sense" rule. In each column the values stored must be of the same kind or type.

- **For example:** If you have a column dob to save date of births of a set of people, then you cannot or you must not save 'names' of some of them in that column along with 'date of birth' of others in that column. It should hold only 'date of birth' for all the records/rows.

- Rule 3: Unique name for Attributes/Columns
- This rule expects that each column in a table should have a unique name. This is to avoid confusion at the time of retrieving data or performing any other operation on the stored data.
- If one or more columns have same name, then the DBMS system will be left confused.

- Rule 4: Order doesn't matters
- This rule says that the order in which you store the data in your table doesn't matter.

Time for an Example

Although all the rules are self explanatory still let's take an example where we will create a table to store student data which will have student's roll no., their name and the name of subjects they have opted for.
Here is our table, with some sample data added to it.
Our table already satisfies 3 rules out of the 4 rules, as all our column names are unique, we have stored data in the order we wanted to and we have not inter-mixed different type of data in columns.
But out of the 3 different students in our table, 2 have opted for more than 1 subject. And we have stored the subject names in a single column. But as per the 1st Normal form each column must contain atomic value.

roll_no	name	subject
101	Akon	OS, CN
103	Ckon	Java
102	Bkon	C, C++

How to solve this Problem?
It's very simple, because all we have to do is break the values into atomic values.
Here is our updated table and it now satisfies the First Normal Form.
By doing so, although a few values are getting repeated but values for the subject column are now atomic for each record/row.
Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

roll_no	name	subject
101	Akon	OS
101	Akon	CN
103	Ckon	Java
102	Bkon	C
102	Bkon	C++

- Second Normal Form (2NF)
- For a table to be in the Second Normal Form,
- It should be in the First Normal form.
- And, <u>it should not have Partial Dependency</u>.

What is Dependency?
Let's take an example of a **Student** table with columns student_id, name, reg_no(registration number), branch and address(student's home address).
In this table, student_id is the primary key and will be unique for every row, hence we can use student_id to fetch any row of data from this table

student_id	name	reg_no	branch	address

Even for a case, where student names are same,

if we know the `student_id` we can easily fetch the correct record.

student_id	name	reg_no	branch	address
10	Akon	07-WY	CSE	Kerala
11	Akon	08-WY	IT	Gujarat

- Hence we can say a **Primary Key** for a table is the column or a group of columns(composite key) which can uniquely identify each record in the table.

- I can ask from branch name of student with student_id **10**, and I can get it. Similarly, if I ask for name of student with student_id **10** or **11**, I will get it.
-
- So all I need is student_id and every other column **depends** on it, or can be fetched using it.

- This is **Dependency** and we also call it **Functional Dependency**.

What is Partial Dependency?

- Now that we know what dependency is, we are in a better state to understand what partial dependency is.
- For a simple table like Student, a single column like student_id can uniquely identfy all the records in a table.
- But this is not true all the time. So now let's extend our example to see if more than 1 column together can act as a primary key.

Let's create another table for **Subject**, which will
have subject_id and subject_name fields
and subject_id will be the primary key.

subject_id	subject_name
1	Java
2	C++
3	Php

Now we have a **Student** table with student information and
another table **Subject** for storing subject information.

Let's create another table **Score**, to store the **marks** obtained
by students in the respective subjects.
We will also be saving **name of the teacher** who teaches that subject alon

score_id	student_id	subject_id	marks	teacher
1	10	1	70	Java Teacher
2	10	2	75	C++ Teacher
3	11	1	80	Java Teacher

- in the score table we are saving the **student_id** to know which student's marks are these and **subject_id** to know for which subject the marks are for.
- Together, student_id + subject_id forms a **Candidate Key** for this table, which can be the **Primary key**.

- See, if I ask you to get me marks of student with student_id 10, can you get it from this table? No, because you don't know for which subject. And if I give you subject_id, you would not know for which student. Hence we need student_id + subject_id to uniquely identify any row.
- But where is Partial Dependency?
- Now if you look at the **Score** table, we have a column names teacher which is only dependent on the subject, for Java it's Java Teacher and for C++ it's C++ Teacher & so on.

- Now as we just discussed that the primary key for this table is a composition of two columns which is student_id & subject_id but the teacher's name only depends on subject, hence the subject_id, and has nothing to do with student_id.

- This is **Partial Dependency**, where an attribute in a table depends on only a part of the primary key and not on the whole key.

How to remove Partial Dependency?
There can be many different solutions for this, but out objective is to remove teacher's name from Score table.

The simplest solution is to remove columns teacher from Score table and add it to the Subject table. Hence, the Subject table will become:

subject_id	subject_name	teacher
1	Java	Java Teacher
2	C++	C++ Teacher
3	Php	Php Teacher

And our Score table is now in the second normal form, with no partial dependency.

score_id	student_id	subject_id	marks
1	10	1	70
2	10	2	75
3	11	1	80