ER Models

What is Entity Relationship Diagram in DBMS?

To give a user view of how the data in a given database has a logical relationship amongst one another. With various units involved in the same like –

- Entities
- Attributes
- Various Relationship
- etc

Unit components of Entity Relationship Diagram

Entity

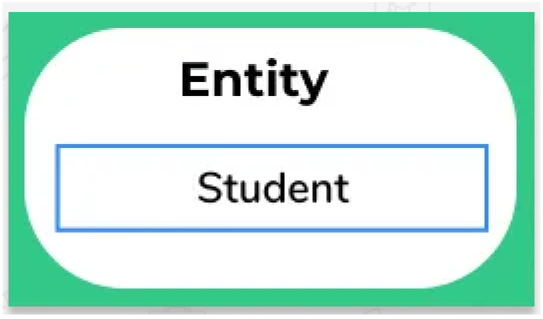
Entity is a real object representation in an Entity Relationship diagram. For example if we want to create a database for a college in that case students studying in the college will be considered as an entity.

It can be anything like –

-
    - Teachers
    - Courses
    - Buildings
    - Classrooms etc

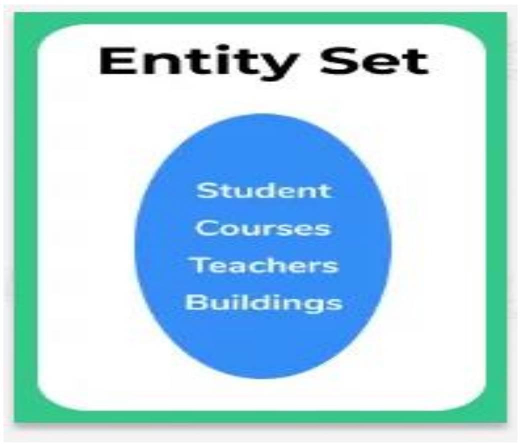
The way of representing an entity is as follows –

Representation : A very simple Rectangular box.



Entity Set
A set of all entities together is called an entity set.

Representation : A vertical Oval listing all entities.



Attributes

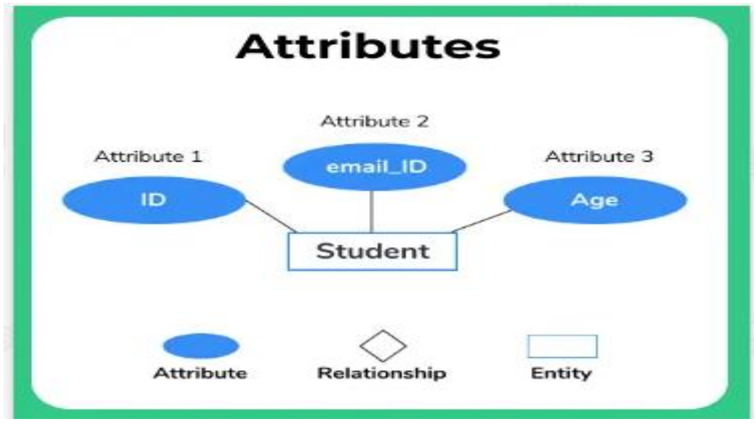
Attributes are the properties of any given entity. For example for student entity the following will be the attributes –

-
    - ID
    - email ID
    - Age
    - Address
    - Phone Number etc

Representation : A horizontal Oval.

There are the following types of attributes –

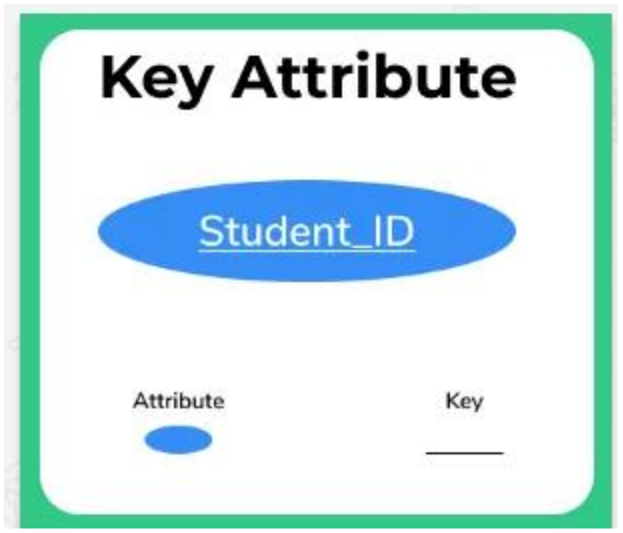
-
    - Key attribute
    - Composite attribute
    - Multivalued attribute
    - Derived attribute



Key Attribute
A key attribute is the one which will uniquely identify and associate identity for an entity.

For example – For a student, his roll number or his studentID maybe its key attribute. For course offered by university, the course code like IT101 will be its key attribute.

Representation : It is same as Horizontal oval for an attribute. But, has an underline below the same.



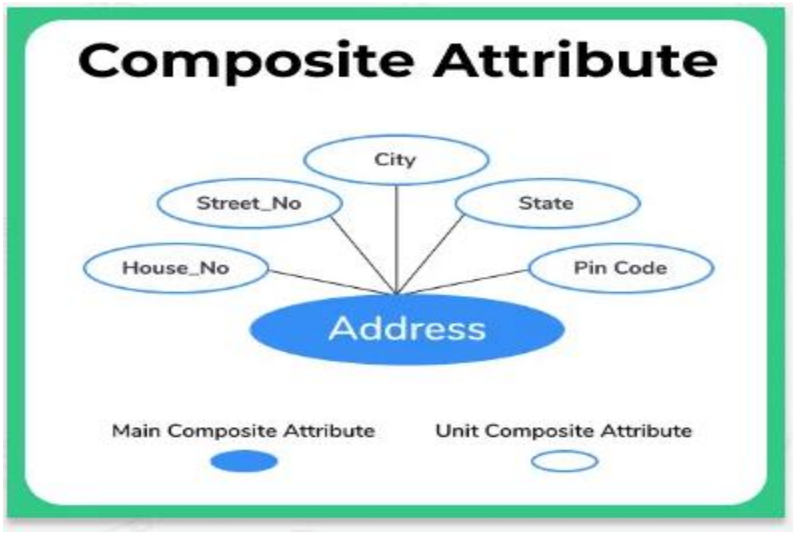
Composite Attribute
As clear from its name a composite attribute composes of multiple units of attributes forming a larger one.

For example – Address may have following composite attributes –

-
    - House Number

- Street
- City
- State
- Country
- Pin Code

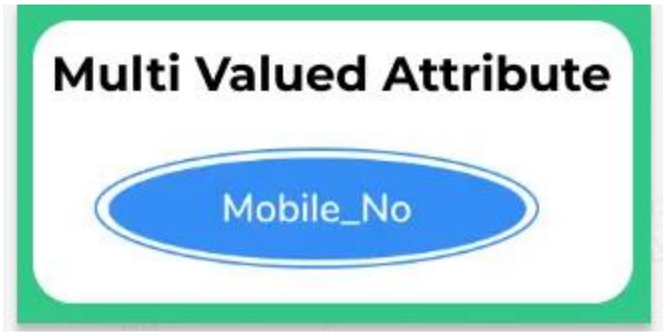
Representation : Horizontal ovals mapped with further composite vertical ovals



<u>Multivalued Attribute</u>

A multivalued attribute is one, which may have more than one value for example. The phone number for student maybe multivalued as student may have one or more than 1 phone numbers.

Representation : Double Horizontal Oval

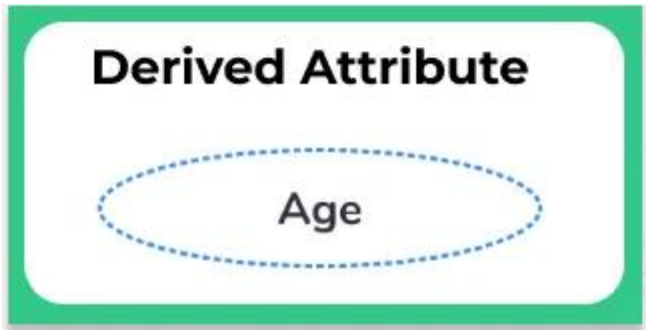


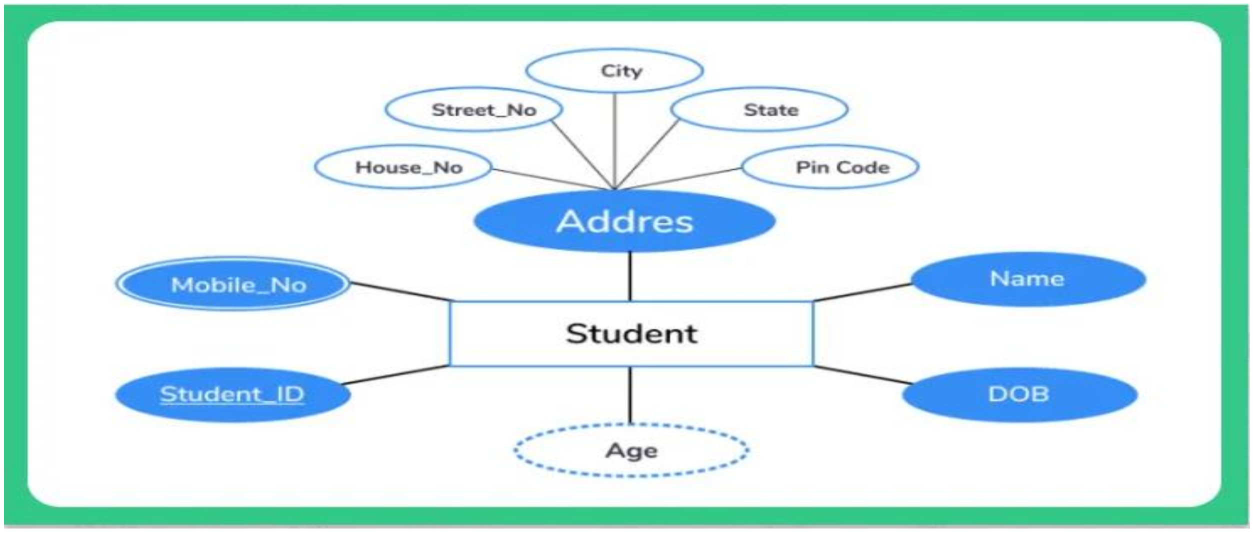
<u>Derived Attribute</u>

A derived attribute as the name suggests is the one that can be derived or calculated with the help of other attribute present itself.

For example – The age of the student can be calculated from date of birth present as an attribute.

Representation : Horizontal Dashed Oval.





Relationship

The association between different entities that are existing in a database is depicted by relationship. For example in a university there may be thousands of students, which have enrolled in a few subjects each year.

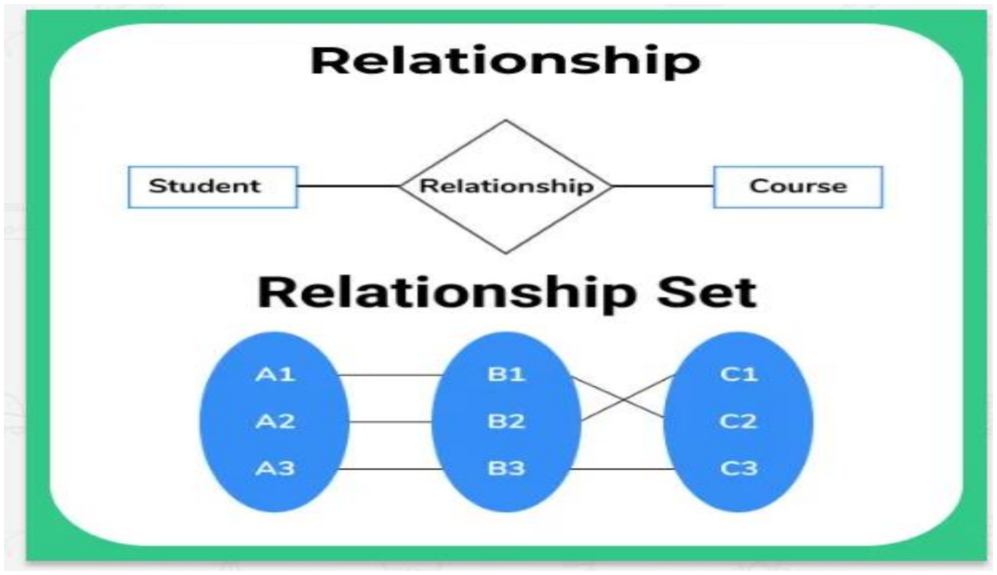
So student entity is related to course entity by enrolment relationship.

Representation : Diamond box with entities connected to its edges as depicted in the image.

There are various types of relationships than can exist in a database which are –

-
  - Unary
  - Binary
  - n-ary

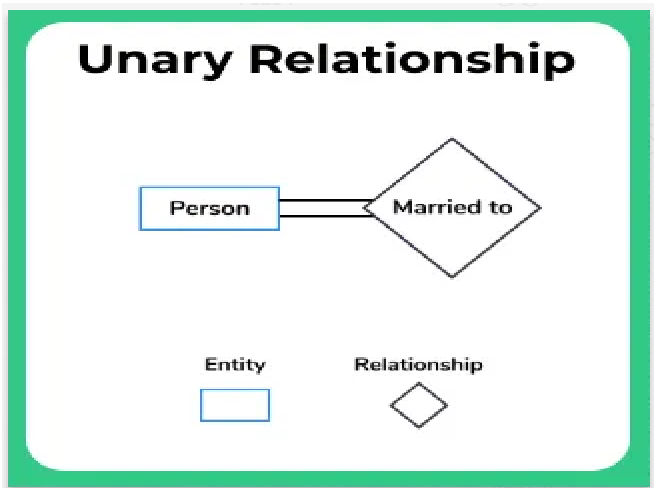
Relationship set is mapping of different entity set to the others with the help of connected lines are depicted in the image.



Unary Relationship
Unary relationship is where there is only entity which is related to itself.

For example –

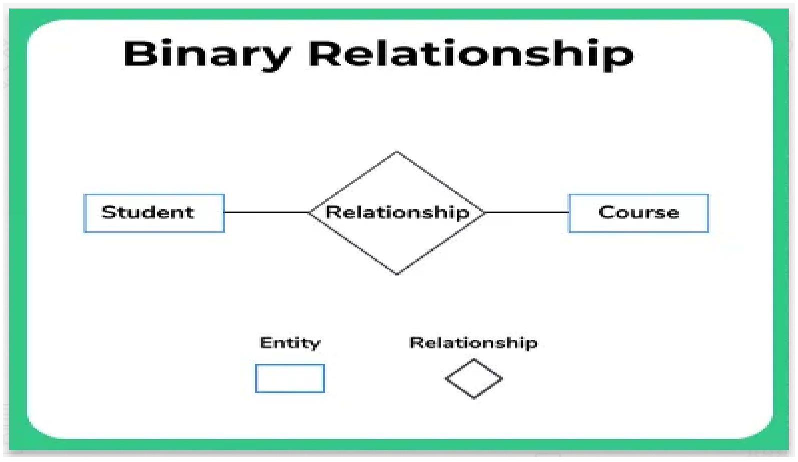
An entity person is related to itself, in relationship as married to.



Binary Relationship
Binary relationship is when there are 2 different entities associated with some relationship with one another.

For example –

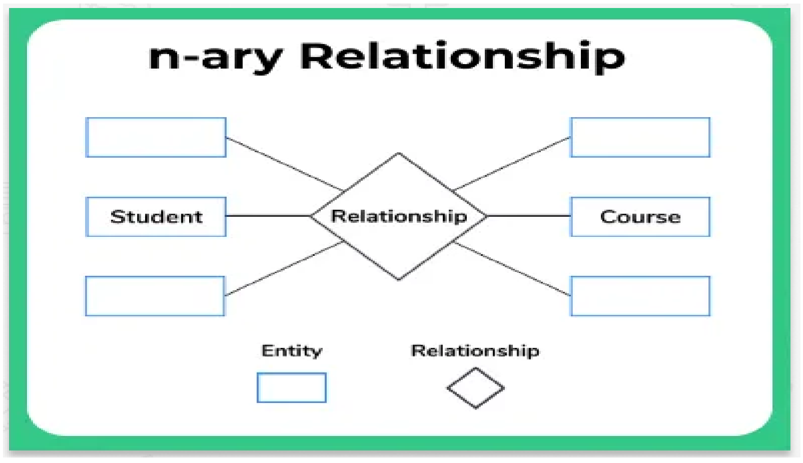
Students enrolled in a course



n-ary Relationship
n-ary relationship is when there is complex relationship amongst various entities. These are generally avoided by programmers to keep database architecture simple

For example –

Men eating animal, Animals eating animals, Men eating Plants, Animal eating plants etc etc. All of them are bound by eating relationship.



Cardinality

For a given entity sites, the number of entities that can participate in a relationship with another entity set is called cardinality.

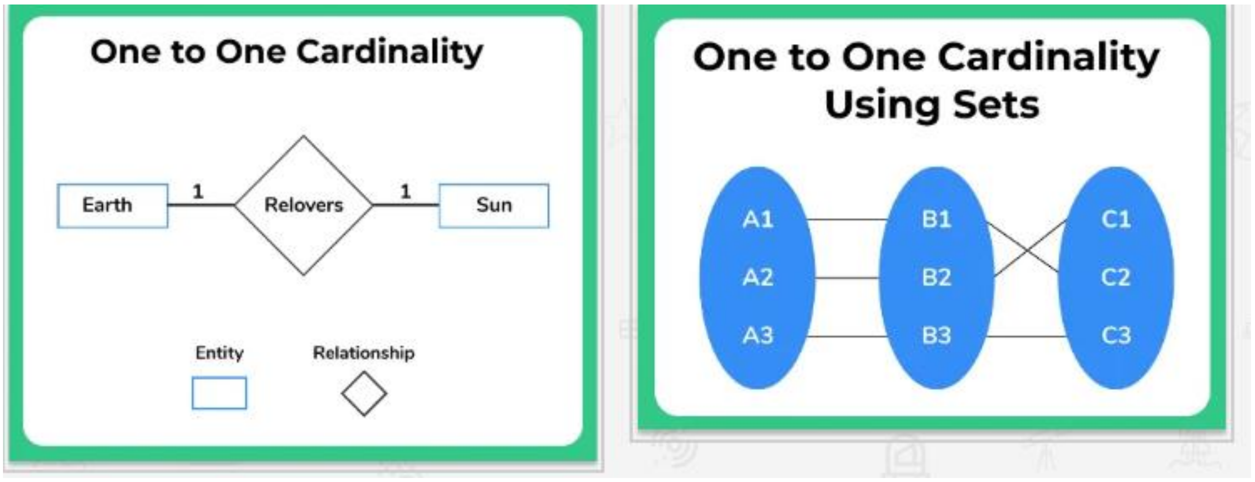
One to One
When there is only one entity at a given instance participating in a relationship or association with only one different entity.

For example –

Man married woman.

At a given time only one man can marry one woman at a given time. So it is one to one relationship.

Another example would be earth revolves around the sun



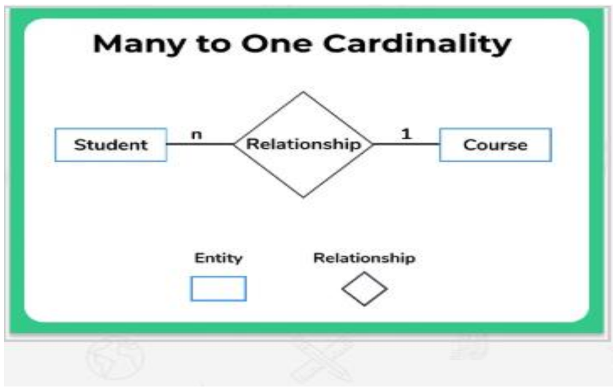
Many to One
When there are multiple entities in a given entity set in relationship with only one entity of another or same entity set.

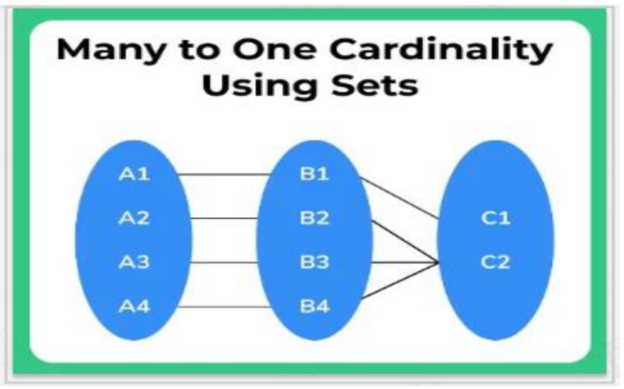
Example –

Assuming there is a rule that student can only enrol in 1 course this semester.

Students enrol in course.

There may be n number of students while the course offered is only 1.

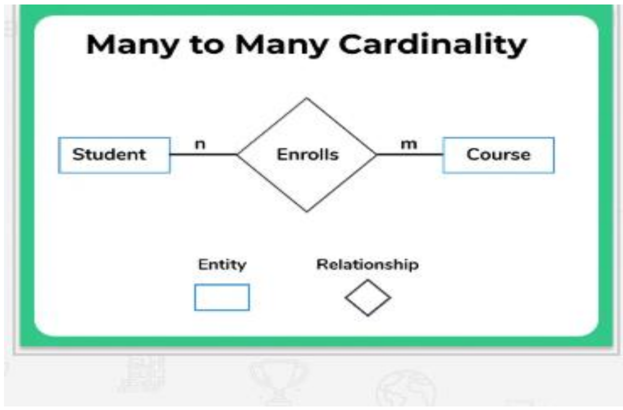


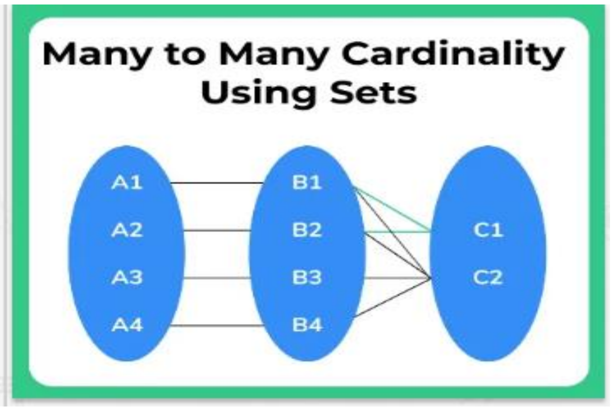


Many to Many
Imagine there is no restriction in number of courses that can be enrolled in, then.

Many students can register in many different courses.

n students may register in m different courses.





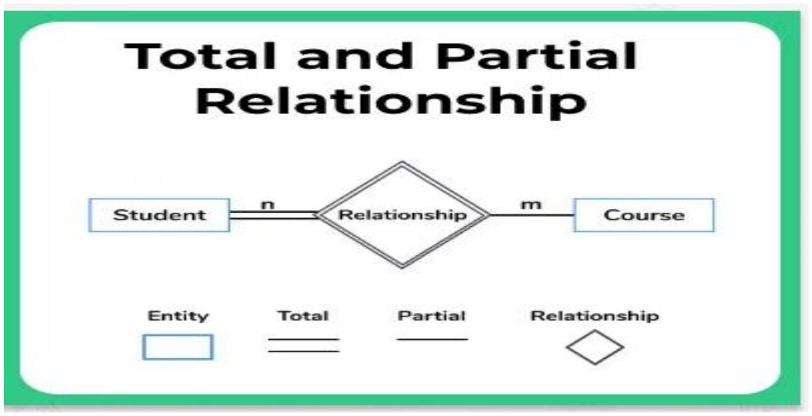
Participation Constraint

Total Participation
All the entities in the entity set must participate in the relationship. For example the university forces, you to necessarily enrol in atleast 1 course, in that case it will be called total participation that student end.

It is represented by a double line joining entity and relationship.

Partial Participation
It is when there maybe cases when some entities may and some entities may not participate in a relationship.

This is represented by single line joining entity and relationship.

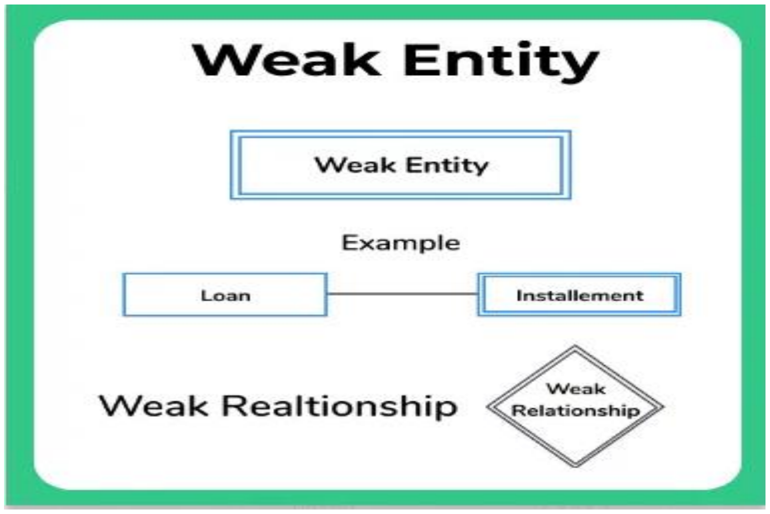


Weak Entity and Weak Relationship

Weak entities are the one which may not have their own attributes and depend on other entity for their existence. For example the monthly instalment is dependent upon the loan entity and its attributes. The participation for weak entity type is always total.

This is represented by a double rectangle as represented in the image.

Weak relationship which is represented by double diamond connects relationship between a strong entity and a weak entity.



Attributes in DBMS

ATTRIBUTES IN DBMS

- Attributes define what type of data is stored in a database table
- For example ***student table stores the details of Student name, roll no, marks, attendance, etc*** **all these details corresponding to the properties or attributes** of the student table
- There are 6 types of attributes
    - Simple attributes
    - Composite attributes
    - Single valued attributes
    - Multi valued attributes
    - Derived attributes
    - Key attributes

Simple Attributes

Simple attributes are those **attributes which can not be divided further.**

Example :

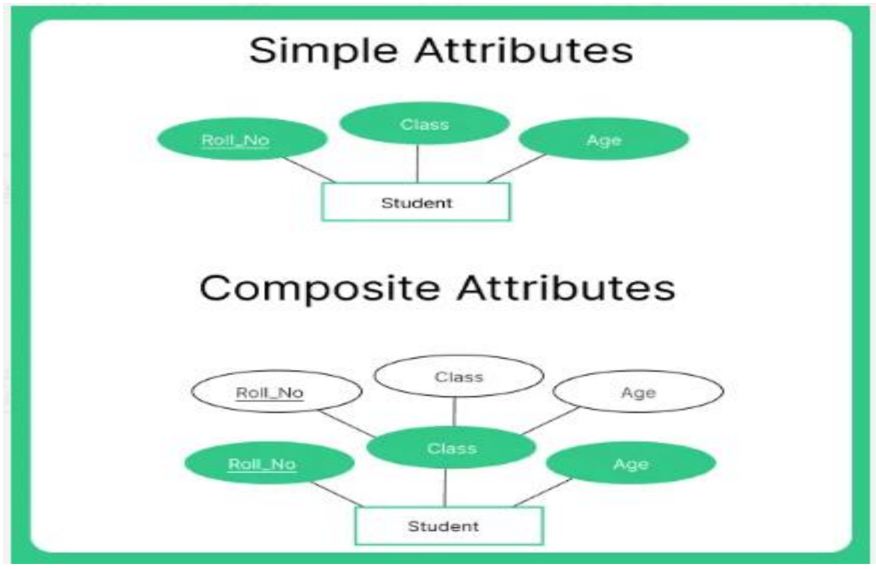
All the attributes are simple attributes as they can not be divided further.

Composite Attributes

Composite attributes are those **attributes which are composed of many other simple attributes.**

Example :

The attributes "Name" and "Address" are composite attributes as they are composed of many other simple attributes.



<u>Single Valued Attributes</u>

Single valued attributes are those **attributes which can take only one value for a given entity from an entity set.**

Example :

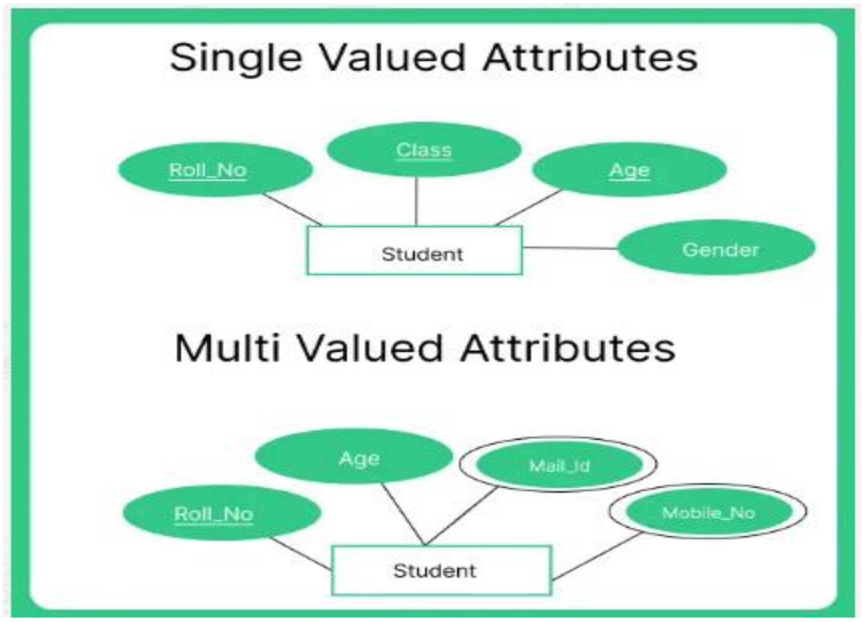
All the attributes are single valued attributes as they can take only one specific value for each entity.

<u>Multi Valued Attributes</u>

Multi valued attributes are those **attributes which can take more than one value for a given entity from an entity set.**

Example :

The attributes “Mob_no” and “Email_id” are multi valued attributes as they can take more than one values for a given entity.



<u>Derived Attributes</u>

Derived attributes are those **attributes which can be derived from other attribute(s).**

Example :

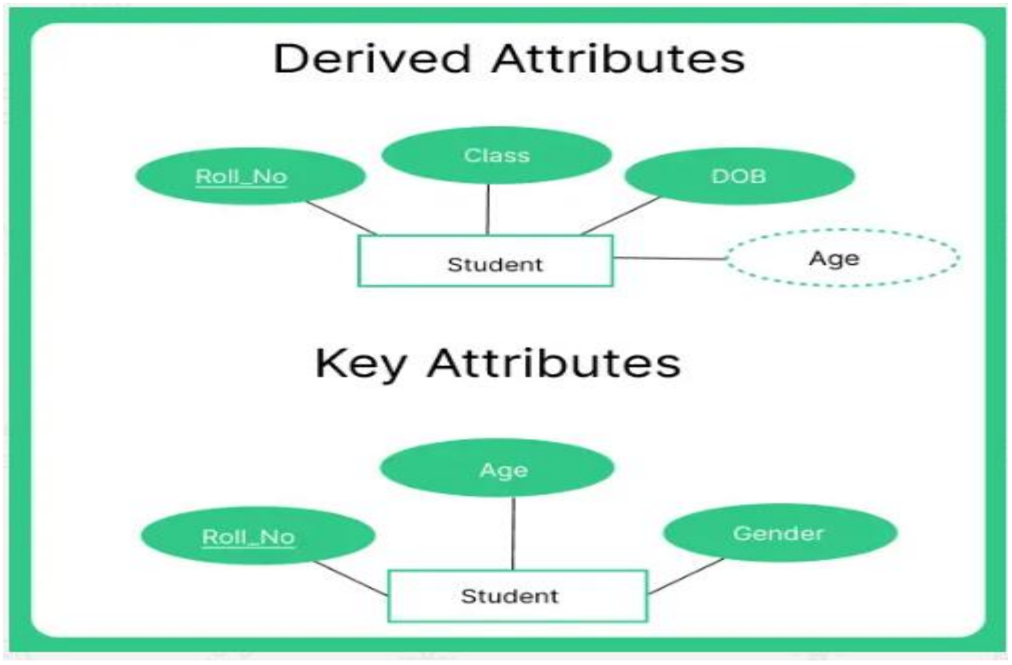
The attribute "Age" is a derived attribute as it can be derived from the attribute "DOB".

<u>Key Attribute</u>

Key attributes are those **attributes which can identify an entity uniquely in an entity set.**

Example :

The attribute "Roll_no" is a key attribute as it can identify any student uniquely.

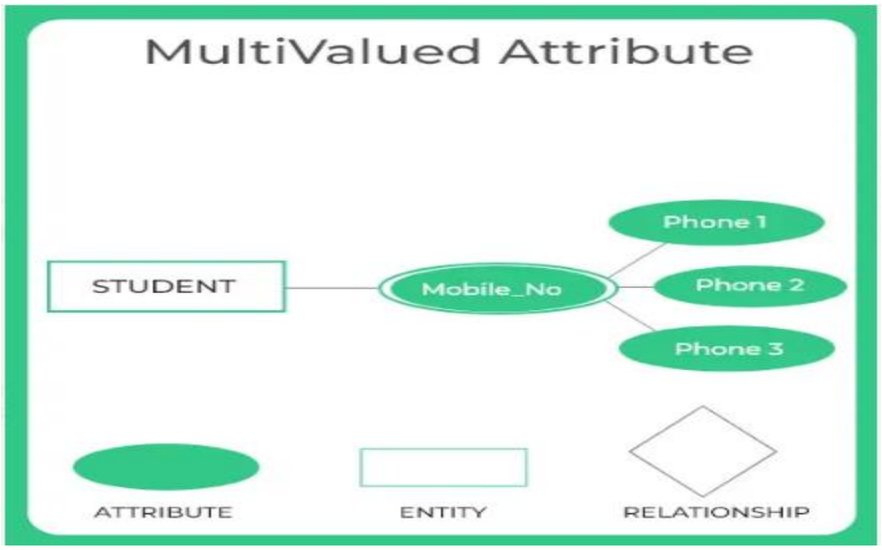


Multivalued Attributes in DBMS

MULTIVALUED ATTRIBUTES IN DBMS

- Attributes define what type of data is stored in a database table for example student table stores details of Student name, roll no, marks, attendance, etc all these details corresponding the properties or attributes of the student table.
- A multivalued attributed in DBMS(Data Base Management System) is an attribute that can have multiple value for a single intance of an entity.
- For example, a person may have multiple phone numbers, so the “Phone Number” attribute for that person would be a multivalued attribute.
- In relational databases, multivalued attributes are typically represented by creating a separate table for the attribute, with a one-to-many relationship between the main entity and the new table.

Multi-valued attribute DefinitionA multivalued attribute is capable of storing more than one value in a single attribute i.e nothing but it can hold multiple values for the single attribute.



Example for Multivalued attribute

Example 1

- In a Real-world scenario, ***a person can have more than one phone number*** hence “***phone number attribute”*** will become a multivalued attribute

Example 2

- A person may have more than one address .hence address may become a multi-valued attribute

Representation of multivalued attribute

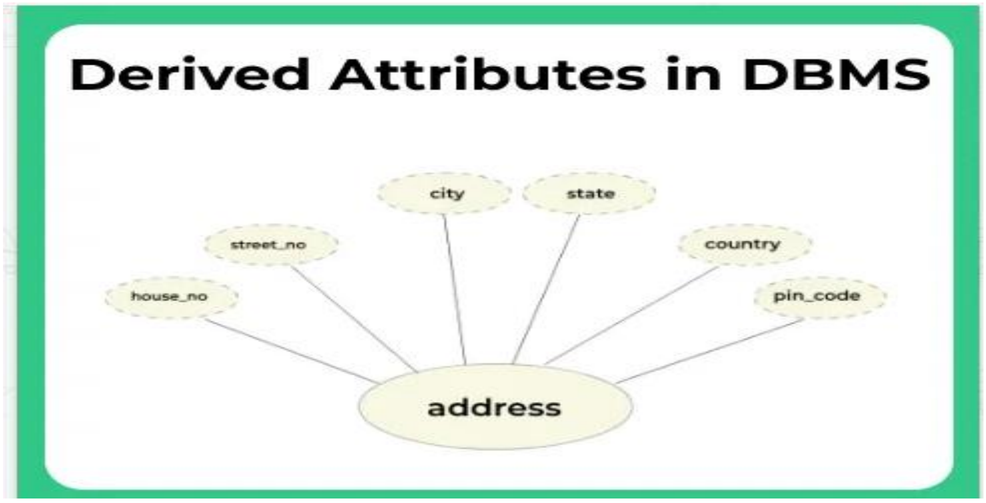
A multivalued attribute is represented using ***a double oval*** **with the name of the attribute inside the oval.**

Derived attribute in DBMS

A derived attribute as the name suggests is the one that can be
derived or calculated with the help of other attributes present themselves.

For example – The age of the student can be calculated from
‘date of the birth present as an attribute.

Representation: Horizontal Dashed Oval



<u>Definition of a derived attribute</u>

- A derived attribute data is derived (copied) from the attribute of another table
- The derived attribute value is always dynamic.

<u>Example of a derived attribute</u>

If you need to copy the age data from another table to the student table then and age attribute will become derived attribute in the student table

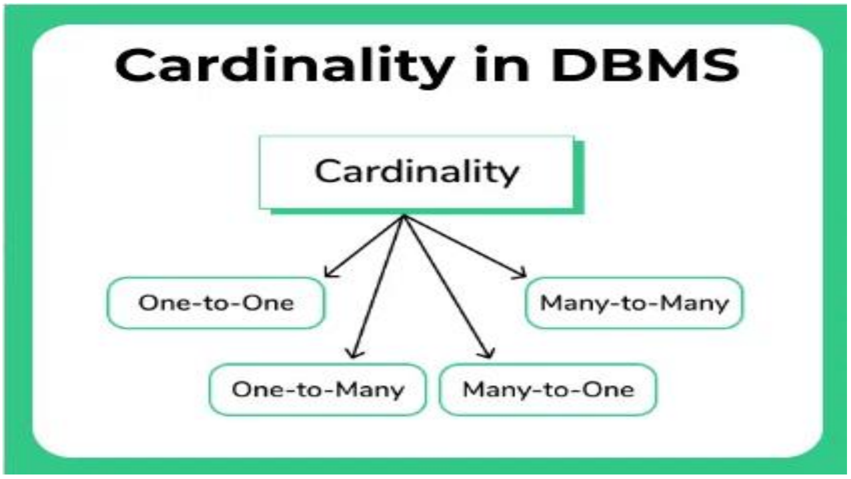
<u>Representation of derived attribute</u>

In the ER diagram, the derived attribute is represented by a **dashed oval** with a name inside the dashed oval

Cardinality in DBMS
Cardinality

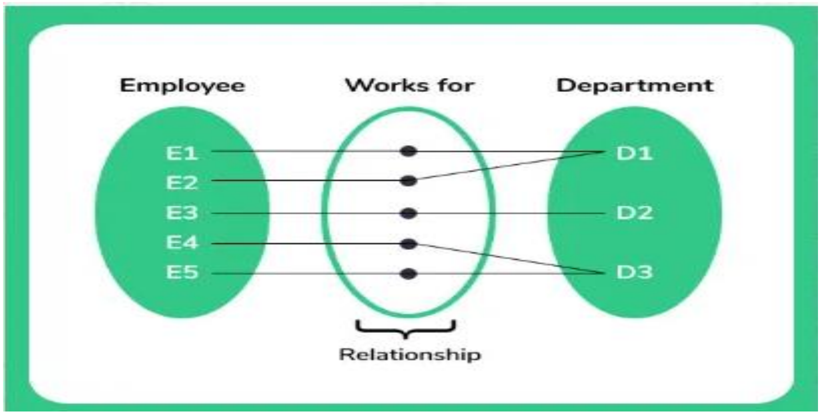
- In terms of SQL, **Cardinality** can be defined as the distinctiveness of data values stored in a column.
  - Higher the cardinality, more is the number of unique data in the columns.
  - Lower cardinality indicates there is a higher number of repeated values in the column.
- In terms of Data Models, Cardinality also refers to the relationship between two tables. The Relationship between the tables can be as-
  - One-to-One.

    - One-to-Many.
    - Many-to-one.
    - Many-to-Many.



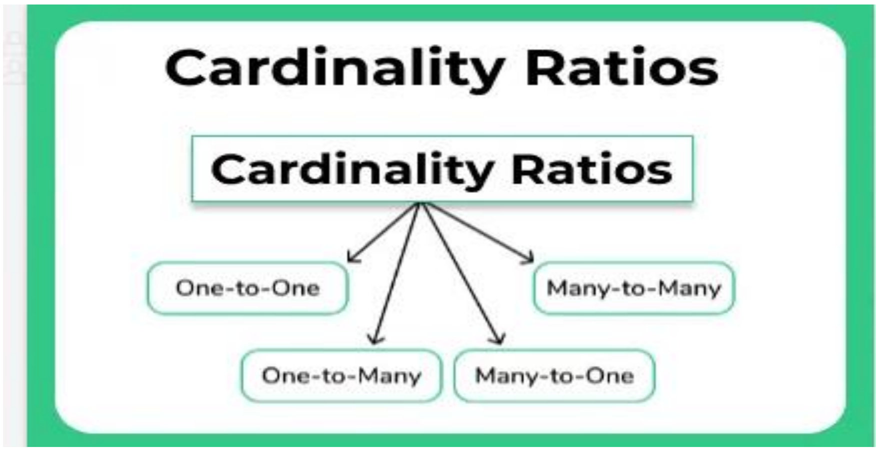
Understanding the concept

- Here we consider data values as shown in figure.
- The figure shows a *Relationship* between an Employee and The Department he works in.
- As learned previously there are two Entities here : *Employee* and *Department*
- ‘Employee’ is related to ‘Department’ by the relationship ‘*Works For’*.
- Now let us understand the concept of Cardinality here.
    - The relationship of **D1** in *Department* to **E1** and **E2** in *Employee* can be defined as **One-to-Many.**
    - Similarly, The Relationship of **E3** in *Employee* to **D2** in *Department* can be defined as **One-to-One.**



Cardinality Ratio in DBMS

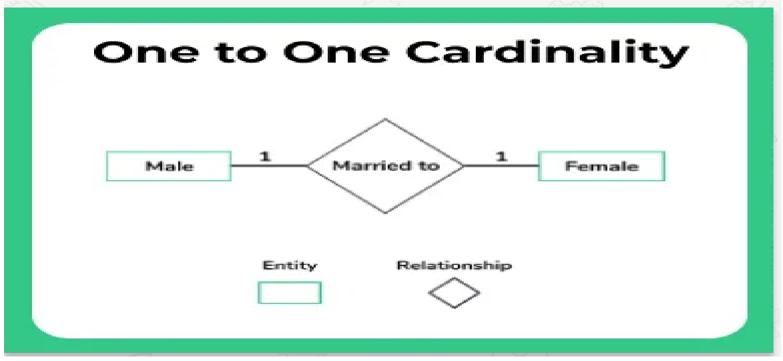
- Cardinality tells how many times the entity of an entity set participates in a relationship
- There are 4 types of cardinalities
  - One to one
  - One to many
  - Many to one
  - Many to many



1. One to one cardinality

- When a single instance of an entity is associated with a single instance of another entity, then it is called a one to one cardinality
- Here each entity of the entity set participates only once in the relationship.

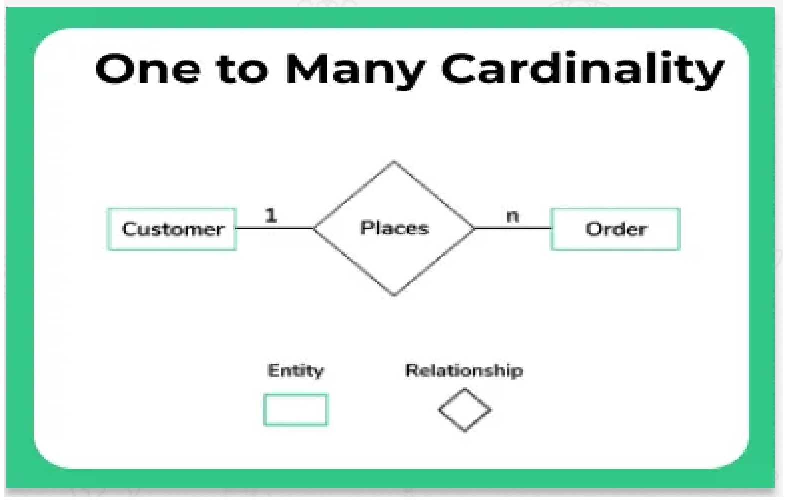
Example for one to one cardinality : Assume that the only male can be married to only one female and one female can be married to only one male, this can be viewed as one to one cardinality



2. One-to-Many cardinality

- When is a single instance of an entity is associated with more than one instance of another entity then this type of relationship is called one to many relationships
- Here entities in one entity set can take participation in any number of times in relationships set and entities in another entity set can take participation only once in a relationship set.

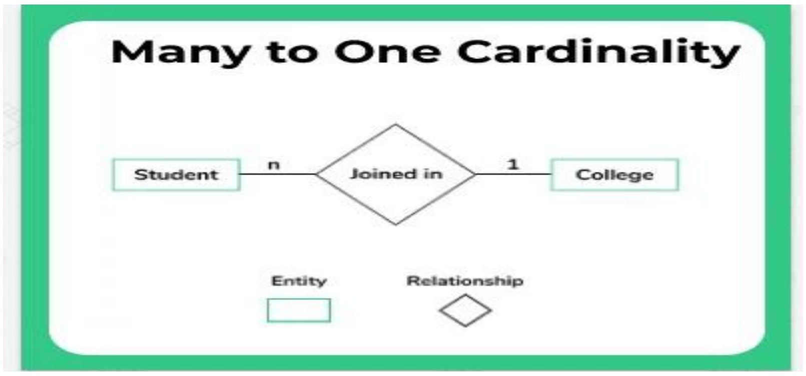
Example for one to many cardinalities For example in the Real-world, many students can study in a single college but the student cannot apply to more than one college at the same time



3. Many-to-one cardinality
When entities in one entity set can participate only once in a relationship set and entities in another entity can participate more than once in the relationship set, then such type of cardinality is called many-to-one

Example for many to one cardinality In real time a student takes only one course but a single course can be taken by any number of students. so many to one relationship is observed here

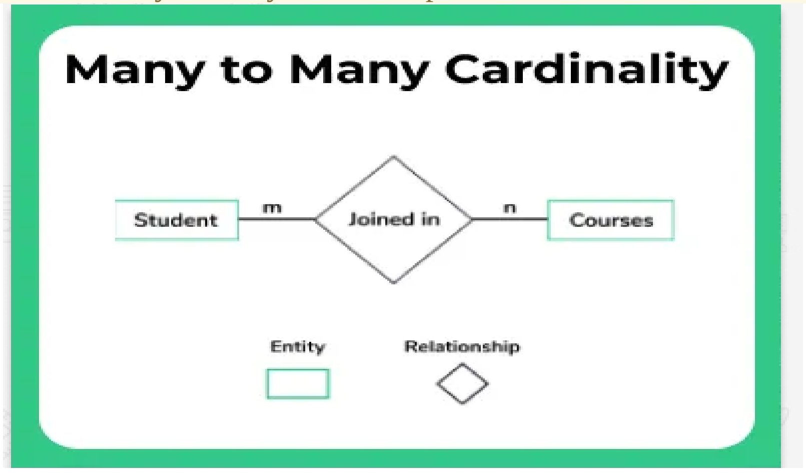
It means that one course can be taken by any number of students but only one course can be allotted to one student



4. Many-to-many cardinality

- Here more than one instance of an entity is associated with more than one instance of another entity then it is called many to many relationships
- In this cardinality, entities in all entity sets can **take participate any number of times in the relationship** cardinality is many to many.

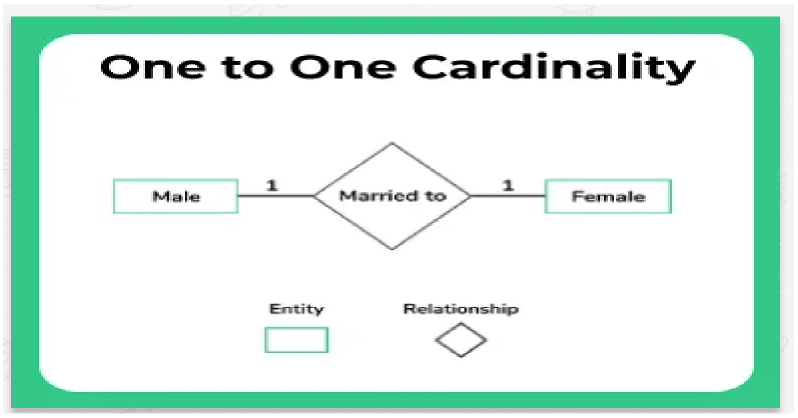
Example for many to many cardinality In the Real world assume that a student can take more than one course and the single course can be taken by any number of students this relationship will be many to many relationship



One to One Relationship Cardinality in DBMS
One to One Relationship Cardinality in DBMS

Cardinality tells ***how many times the entity of an entity set participates in a relationship***

There are four different types of cardinalities **one to one, many to one, one to many, many to many,** here will discuss how one to one cardinality is observed.



One to One cardinality

- When a single instance of an entity is associated with a single instance of another entity, then it is called as one to one cardinality
- Here each entity of the entity set participate only once in the relationship

Example for one to one cardinality

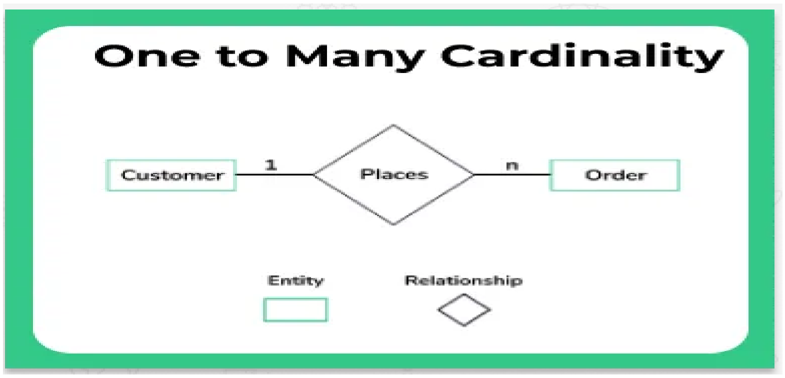
Assume that the ***only male can be married to only one female and one female can be married to only one male,*** this can be viewed as one to one cardinality.

One to Many Relationship Cardinality in DBMS

One to Many Relationship Cardinality in DBMS

Cardinality tells how many times the entity of an entity set participates in a relationship

There are four different types of cardinalities one to one, many to one, one to many, many to many' here will discuss how many to one cardinality is observed



One-to-Many cardinality

- When is a single instance of an entity is associated with more than one instance of another entity then this type of relationship is called one to many relationships
- Here entities in one entity set can take participation in any number of times in relationships set and entities in another entity set can take participation only once in a relationship set

<u>Example for one to many cardinalities</u>

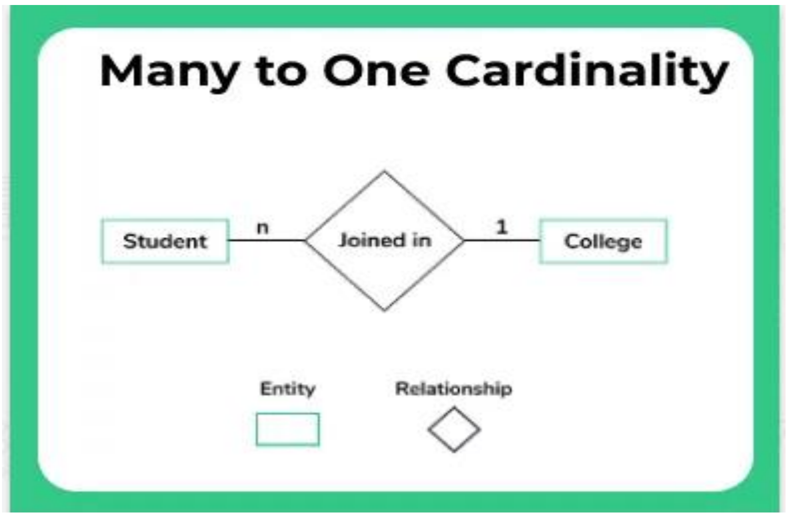
For example in the Real world, ***many students can study in a single college but the student cannot apply to more than one college at the same time***

Many to One Relationship Cardinality in DBMS

Many to One Relationship Cardinality in DBMS

Cardinality tells how many times the entity of an entity set participates in a relationship

There are four different types of cardinalities one to one, many to one, one to many, many to many' he will discuss how many to one cardinality how is observed



Many-to-one cardinality
When entities in one entity set can participate only once in a relationship set and entities in another entity can participate more than once in the relationship set, then such type of cardinality is called many-to-one.

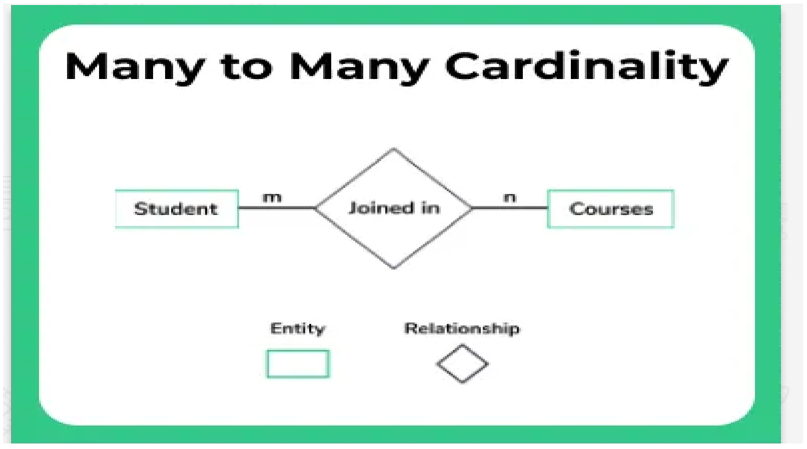
Example for many to one cardinality

- In real time a student takes only one course but a single course can be taken by any number of students. so many to one relationship is observed here
- It means that ***one course can be taken by any number of students but only one course can be allotted to one student***

Many to many Relationship Cardinality in DBMS
Many to many Relationship Cardinality in DBMS

Cardinality tells how many times the entity of an entity set participates in a relationship

There are four different types of cardinalities one to one, many to one, one to many, many to many.he will discuss how many to many cardinality how is observed.



<u>Many-to-many cardinality</u>

- Here more than one instance of an entity is associated with more than one instance of another entity then it is called many to many relationships
- In this cardinality, entities in all entity sets can **take participate any number of times in the relationship** cardinality is many to many.

<u>Example for many to many cardinality</u>

- In the Real world assume that a ***student can take more than one course and the single course can be taken by any number of students*** this relationship will be many to many relationship

Relationships in DBMS

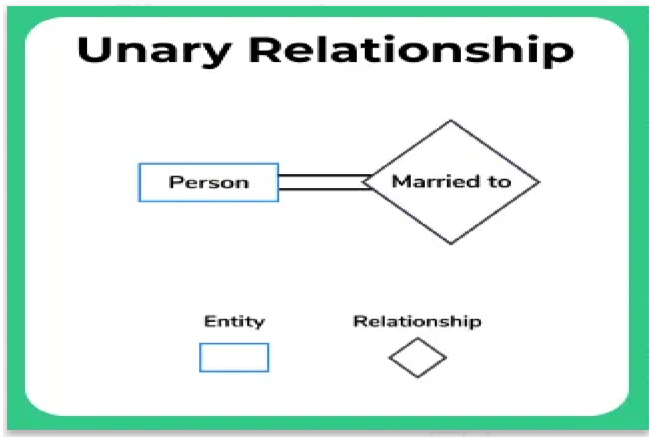
While most websites say that the following are types of relationships –

- One to One
- Many to One
- Many to Many

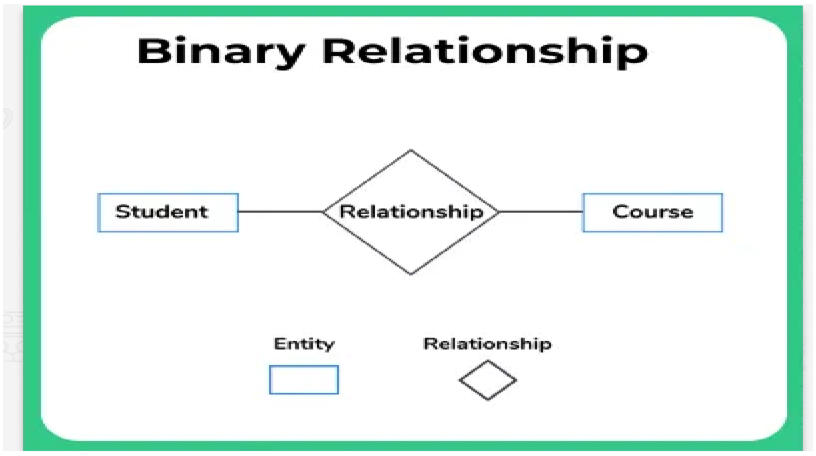
The above information is not true, the above is called Cardinality. The relationship is of the following types –

- Unary Relationship
- Binary
- n-ary

Unary Relationship
It is a case when an entity is related to itself. For example an entity type called person will be married to a person itself.



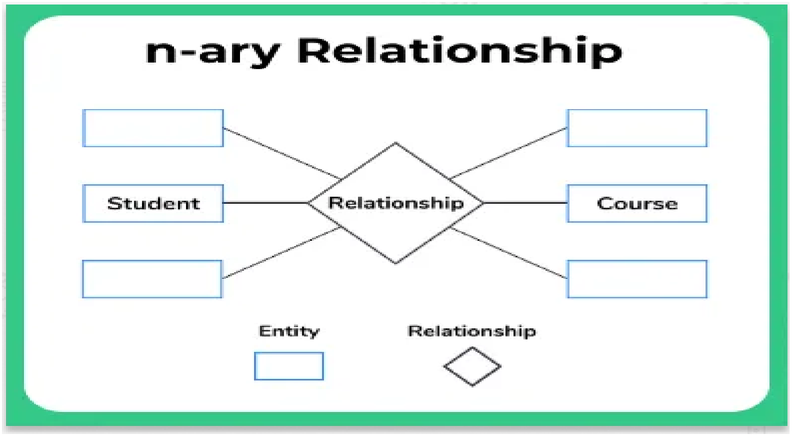
Binary Relationship
It is when there are two entities which different entities are related to one another like a student is related to another entity called courses, as enroll type of relationship.



n-ary Relationship
n-ary relationship is when there is complex relationship amongst various entities. These are generally avoided by programmers to keep database architecture simple

For example –

Men eating animal, Animals eating animals, Men eating Plants, Animal eating plants etc etc. All of them are bound by eating relationship.



Cardinality

All websites confused relationships as cardinality. While cardinality refers to number of participating instances in a relationship, which can be anyUnary, Binary or n-ary. Following, are the cardinality in relationships in DBMS.

- **One to one**
- **Many to One**
- **Many to many**

One to One Cardinality in Relationship

In such cases only, a unique row in Table A will be related to only and only one unique row in Table B. We can take example of marriage between man and wife.

Many to One Cardinality in Relationship

In such cases many rows in table A are related to only one unique row of Table B. For example course enrollment relationship between students and University. For example many students can enroll in only one single university.

One to Many Cardinality in Relationship

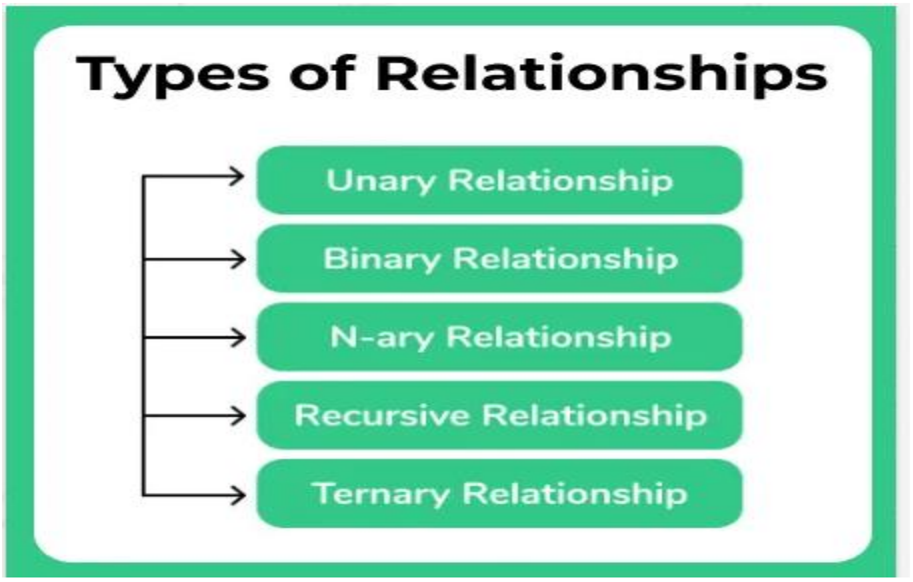
In such cases a single unique row in table A are related to only multiple rows of Table B. For example Company gives employment to employees.

Many to Many Cardinality in Relationship

In such cases many rows in table A are related to many rows of Table B. For example student course enrollment relationship between students and Courses offered by university. As many students can enroll for many courses.

Types of Relationships in DBMS
Types of Relationships in DBMS

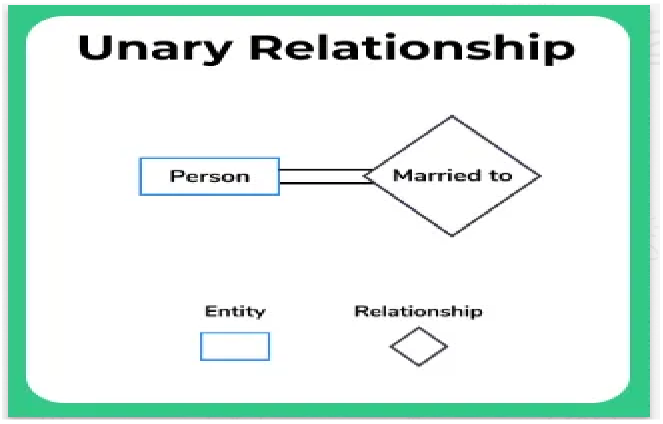
- The relationship also ***shows the different entity sets that are participating in a relationship,*** these relationships are very much useful for analyzing the design process of the system.
- There are 5 types of relationships
    - Unary Relationship
    - Binary Relationship
    - n-ary Relationship
    - Recursive Relationship
    - Ternary Relationship



Unary relationship
When there is only one entity set participating in a relationship then such type of relationship is called unary relationship

Example of unary relationship

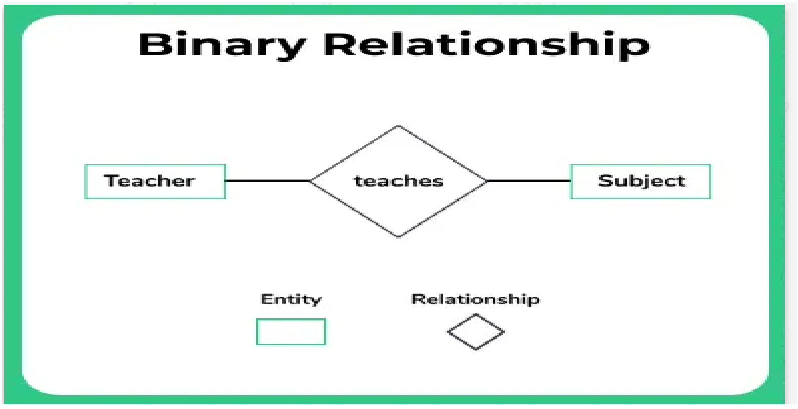
For example, a person has ***only one passport and only one passport is given to only one person*** and hence unary relationship is observed



Binary relationship
When there are exactly two entity sets participating in a relationship then such type of relationship is called binary relationship

Example of binary relationship

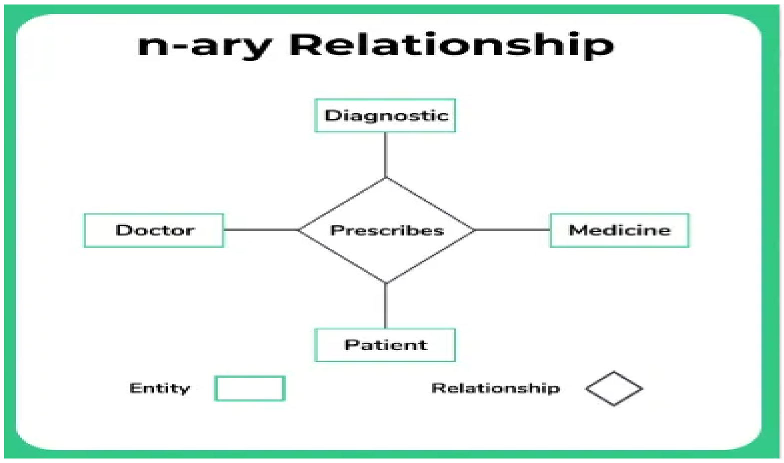
For example, a ***teacher teaches a subject*** here 2 entities are teacher and subject for the   relationship teacher teaches subject



N-ary relationship
When a large number of entity sets are participating in a relationship, then such type of relationship is called an n-ary relationship

Example of n-ary Relationship

In the real world, a patient goes to a doctor and doctor prescribes the medicine and diagnosis to the patient, ***four entities Doctor, patient and medicine, diagnostics are involved in the relationship "prescribes"***

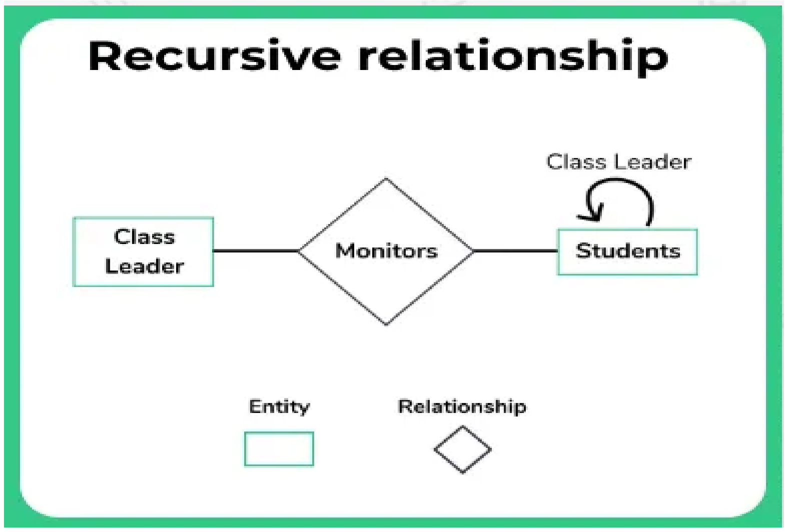


Recursive relationship

A Recursive relationship is nothing but, simply an entity is having a relationship with self

<u>**Example of recursive Relationship**</u>

-
  - The ***person who is a supervisor for many other employees also come under employees category***
  - Student can be a class monitor and handle other students ***but a person who is working as a class leader is itself a student of the class*** and hence a class monitor has a recursive relationship of entity student



Ternary relationship
When there are exactly three entity sets participating in a relationship then such type of relationship is called ternary relationship

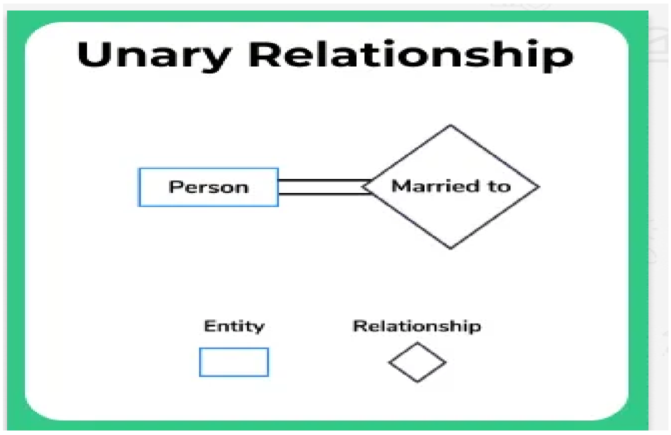
Example of Ternary Relationship

In the real world, a patient goes to a doctor and doctor prescribes the medicine to the patient, ***three entities Doctor, patient and medicine are involved in the relationship "prescribes"***

Unary Relationship in DBMS

- A relationship represents the association between two are more entities
- The relationship also ***shows the different entity sets that are participating in a relationship,*** these relationships very much useful analyzing the design process of the system

We have observed **unary, binary n-ary, recursive, ternary** relationships in a database design schema. Here we will discuss how unary relationship exists.



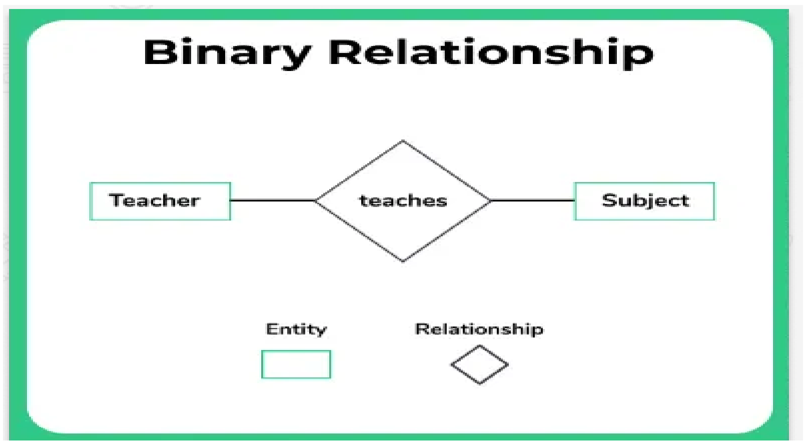
Unary relationship
When there is only one entity set participating in a relationship then such type of relationship is called unary relationship

Example of unary relationship
For example, a person has ***only one passport and only one passport is given to only one person*** and hence unary relationship is observed

Binary Relationship in DBMS
Binary Relationship in DBMS

- A relationship represents the association between two are more entities
- The relationship also ***shows the different entity sets that are participating in a relationship,*** these relationships very much useful analyzing the design process of the system

We have observed **unary, binary n-ary, recursive, ternary** relationships in a database design schema. Here we will discuss how binary relationship exists.



Binary relationship
When there are exactly two entity sets participating in a relationship then such type of relationship is called binary relationship

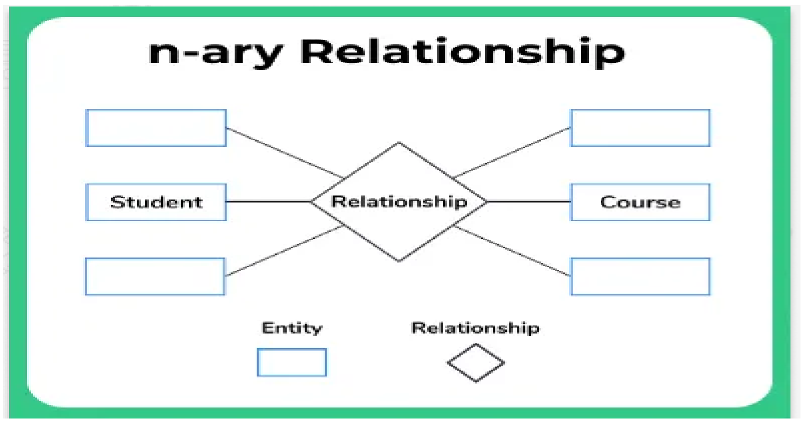
Example of binary relationship
For example, a ***teacher teaches a subject*** here 2 entities are teacher and subject for the relationship teacher teaches subject

N-ary Relationship in DBMS
N-ary Relationship in DBMS

- A relationship represents the association between two are more entities

- The relationship also ***shows the different entity sets that are participating in a relationship,*** these relationships very much useful analyzing the design process of the system

We have observed **unary, binary n-ary, recursive, ternary** relationships in a database design schema. Here we will discuss how n-ary relationship exists.



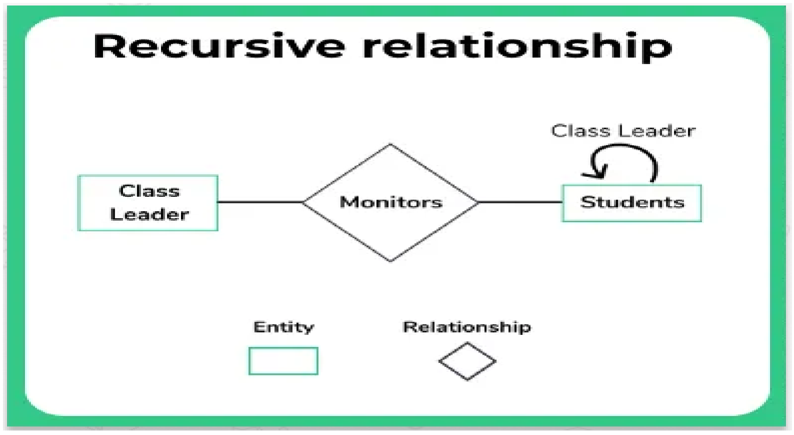
N-ary relationship
When a large number of entity sets are participating in a relationship, then such type of relationship is called an n-ary relationship

<u>Example of n-ary Relationship</u>
In the real world, a patient goes to a doctor and doctor prescribes the medicine and diagnosis to the patient, ***four entities Doctor, patient and medicine, diagnostics are involved in the relationship "prescribes"***

Recursive Relationship in DBMS
Recursive Relationship in DBMS

- A relationship represents the association between two are more entities
- The relationship also ***shows the different entity sets that are participating in a relationship,*** these relationships very much useful analyzing the design process of the system

We have observed **unary, binary n-ary, recursive, ternary** relationships in a database design schema. Here we will discuss how recursive relationship exists



<u>Recursive relationship</u>

- Relationship between two entities of the same type is called recursive relationship,
- A Recursive relationship is nothing but, simply an entity is having a relationship with self

<u>Example of recursive Relationship</u>

- The ***person who is a supervisor for many other employees also come under employees category***
- Student can be a class monitor and handle other students ***but a person who is working as a class leader is itself a student of the class*** and hence a class monitor has a recursive relationship of entity student

Ternary Relationship in DBMS

- A relationship represents the association between two are more entities
- The relationship also ***shows the different entity sets that are participating in a relationship,*** these relationships very much useful analyzing the design process of the system

We have observed **unary, binary n-ary, recursive, ternary** relationships in a database design schema. Here we will discuss how ternary relationship exists.

<u>Ternary relationship</u>
When there are exactly three entity sets participating in a relationship then such type of relationship is called ternary relationship

<u>Example of Ternary Relationship</u>

In the real world, a patient goes to a doctor and doctor prescribes the medicine to the patient, ***three entities Doctor, patient and medicine are involved in the relationship "prescribes"***

Relationship Cardinality in DBMS

Cardinality Relationship for Data Models

There are the following four types of Cardinality Relationship in DBMS –

1. One to One
2. One to Many
3. Many to One
4. Many to Many

One to One

A relationship between a husband and wife can be considered as one to one, where The Entities are Husband and Wife and the Relationship is Marriage This means, if you put this information in a DBMS table, a particular row of one table only associates itself with single and unique row of another table.

One to Many

A relationship between a Company (1) and Employees (n) can be considered as one to many, where The Entities are company and employees and the Relationship is pays. This means, if you put this information in a DBMS table, a particular row of one table associates itself with multiple rows of another table.

Many to One

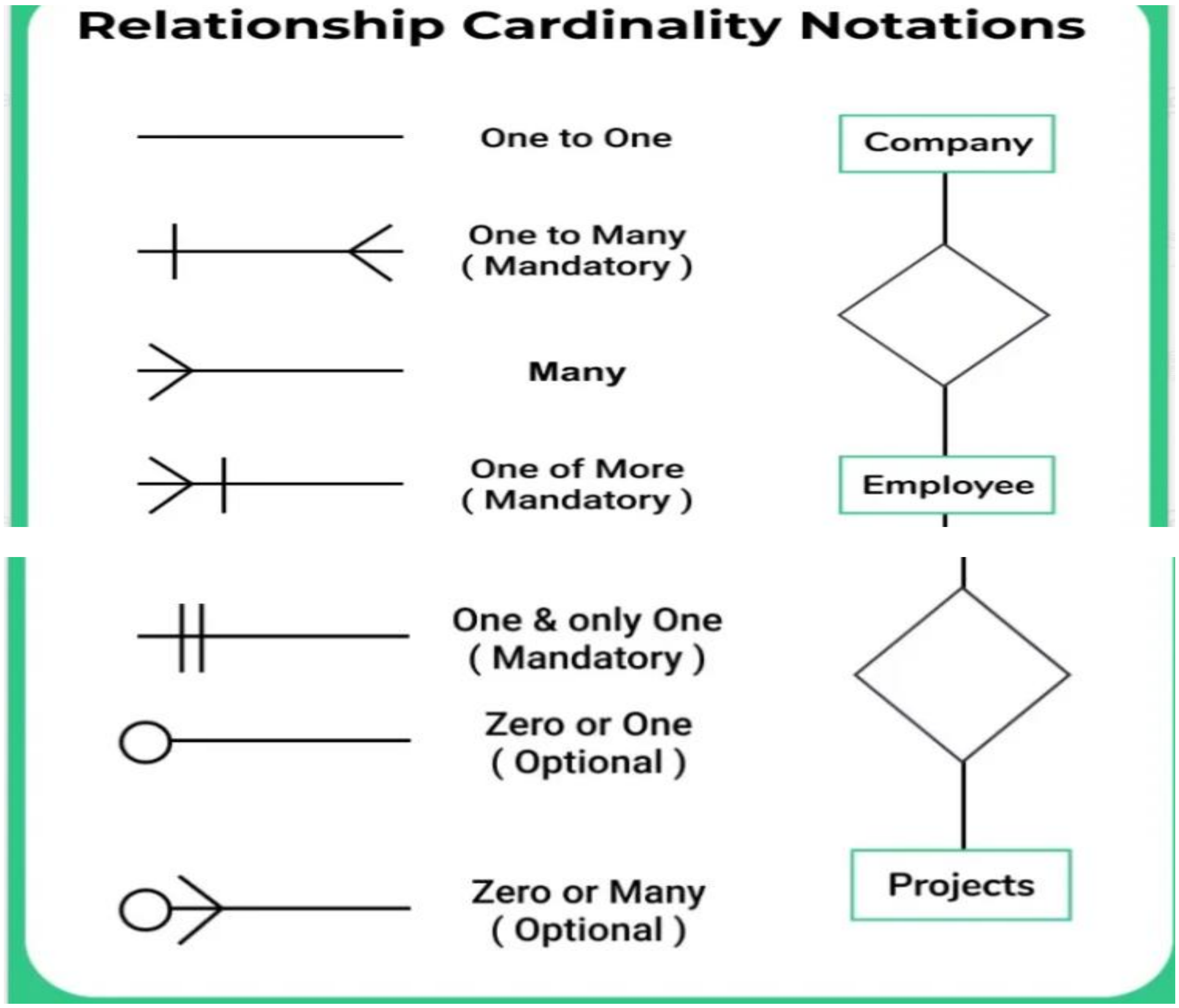
A relationship between a Students and University can be considered as one to one, where The Entities are Students and University and the Relationship is study. This means, if you put this information in a DBMS table, multiple rows of student table associates itselves with single row of University table.

Many to Many

A relationship between a students and courses can be considered as many to many, where The Entities are students and courses and the Relationship is enroll. This means, if you put this information in a DBMS table, a multiple rows of one table associates itself with multiple rows of another table.

Cardinality Notations

- **New Notations (Kaushik Style) –** These follow 1, n, m structure only and are in fashion of being used.
- **Old Notations (Bachman Style)** – These follow pictorial representation of relationship without using 1, n, m. You can find examples below, however, these are unpopular these days.


Relationship Cardinality Notations
One to One
One to Many (Mandatory)
Many
One of More (Mandatory)
One & only One (Mandatory)
Zero or One (Optional)
Zero or Many (Optional)
Company
Employee
Projects

Cardinality Relationship for Query Optimization

In terms of this, the system with less redundancy that is less duplicacy has high cardinality. For example one table with all unique entities in a column will have higher cardinality than the one with duplicate values.

Participation Constraints in DBMS

In a Relationship, Participation constraint specifies the presence of an entity when it is related to another entity in a relationship type. It is also called the minimum cardinality constraint.

This constraint **specifies the number of instances of an entity that are participating in the relationship type.**

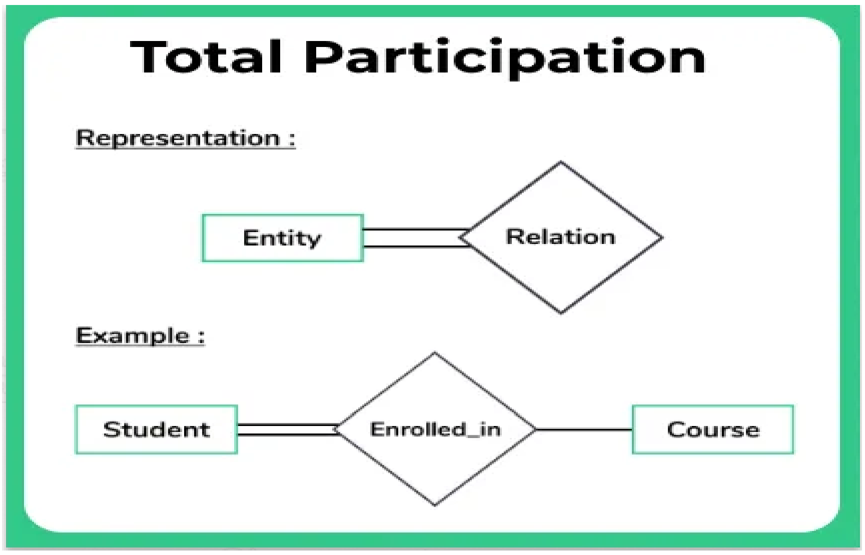
- There are two types of Participation constraint:
    - Total participation
    - Partial participation

Total participation constraint

- It specifies that **each entity present in the entity set must mandatorily participate in at least one relationship instance of that relationship set**,for this reason, it is also called as mandatory participation
- It is **represented using a double line** between the entity set and relationship set

Example of total participation constraint

-
    - It specifies that each student must be enrolled in at least one course where the **"student" is the entity set** and **relationship "enrolled in" signifies total participation**
    - It means that **every student must have enrolled at least in one course**

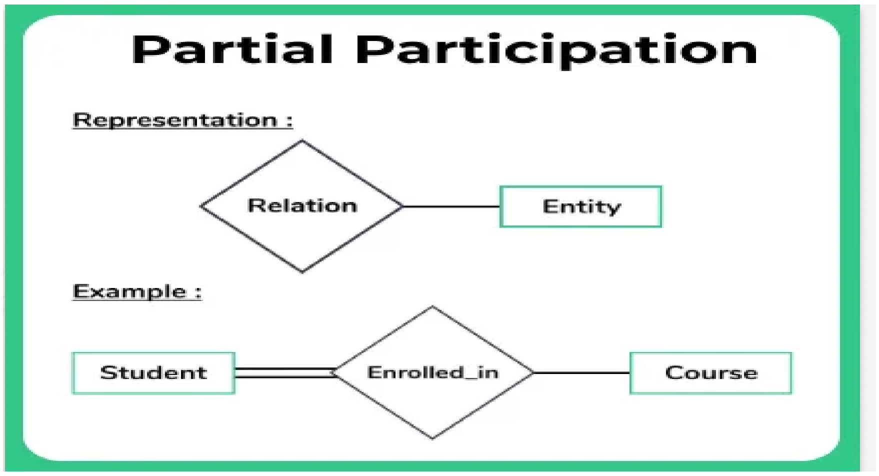


Partial participation

- It specifies that **each entity in the entity set may or may not participate in the relationship instance of the relationship set**, is also called as optional participation
- It is represented using a **single line between the entity set and relationship set** in the ER diagram

<u>**Example of partial participation**</u>

A **single line between the entities i.e courses and enrolled in a relationship signifies the partial participation,**which means there might be some courses where enrollments are not made i.e enrollments are optional in that case



Weak Entity and Strong Entity in DBMS

Strong entity

- Strong Entity is independent of any other entity in the schema

Example – A student entity can exist without needing any other entity in the schema or a course entity can exist without needing any other entity in the schema

- A Strong entity is nothing but an entity set having a primary key attribute or a table that consists of a primary key column

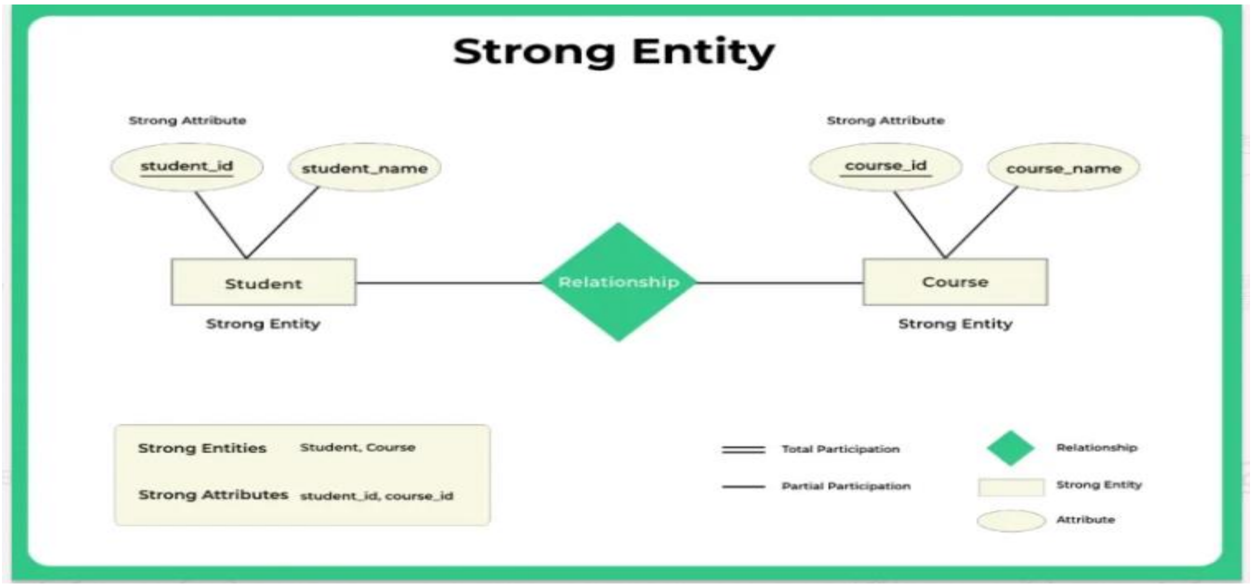
Representation

-
  - The **strong entity** is represented by a **single rectangle**.
  - **The relationship between two strong entities** is represented by a **single diamond.**

NoteThe primary key of the strong entity is represented by underlining it (Called as Strong Attribute)

Examples for the strong entity

-
  - Consider the ER diagram which consists of **two entities student and course**
  - **Student entity is a strong entity** because it consists of a **primary key** called student id which is enough for accessing each record uniquely
  - In the same way, the course entity contains of course ID attribute which is capable of uniquely accessing each row.



Weak entity

- A weak entity is an entity set that does not have sufficient attributes for Unique Identification of its records.

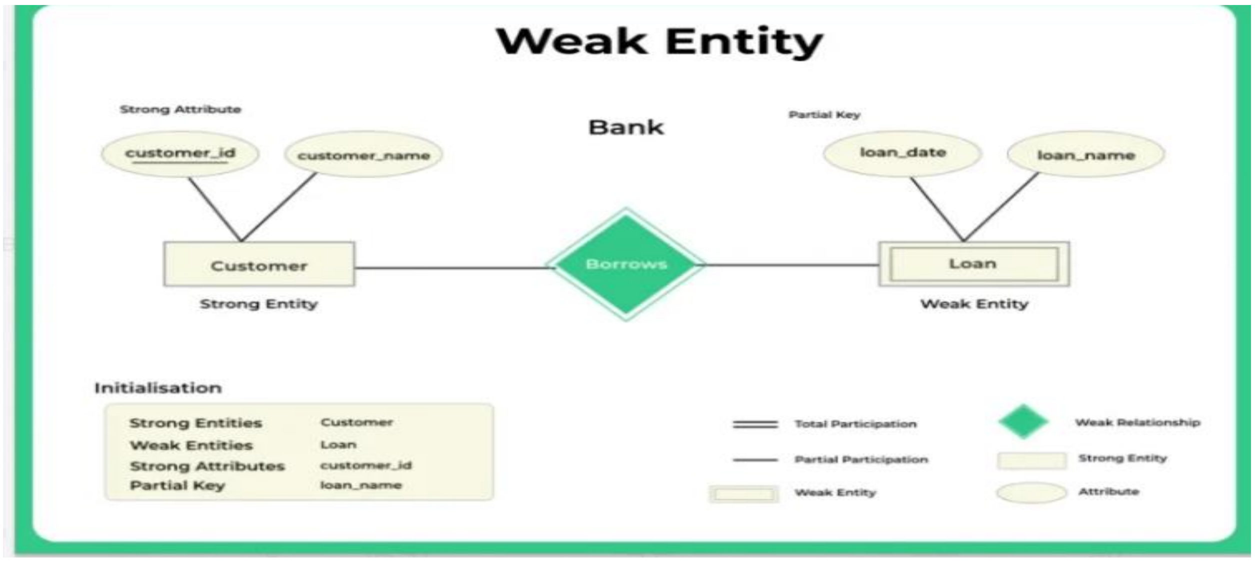
Example 1 – A loan entity can not be created for a customer if the customer doesn't exist

Example 2 – A dependents list entity can not be created if the employee doesn't exist

- Simply a weak entity is nothing but an entity that does not have a primary key attribute
- It **contains** a partial key called a **discriminator which helps in identifying a group of entities from the entity set**
- **A discriminator** is represented by underlining with a **dashed line**

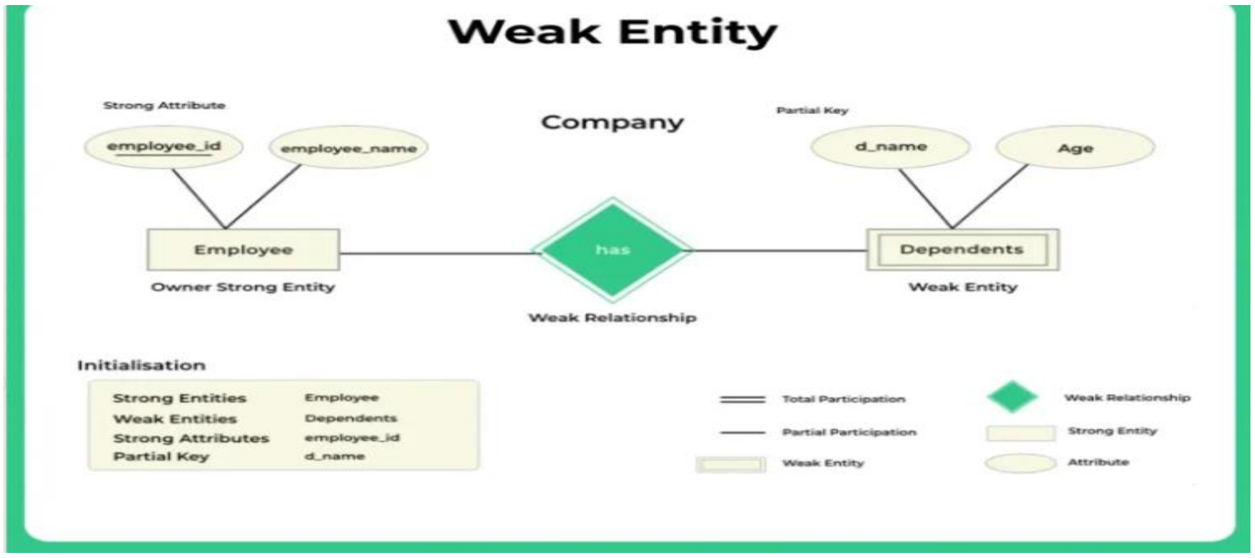
Representation

-
    - A **double rectangle** is used for representing a **weak entity set**
    - The **double diamond** symbol is used for representing the **relationship between a strong entity and a weak entity** which is known as identifying relationship



Example for weak entity

-
    - In the ER diagram, we have **two entities Employee and Dependents.**
    - **Employee is a strong entity** because it has a **primary key** attribute called Employee number (Employee_No) which is capable of uniquely identifying all the employee.
    - Unlike Employee, **Dependents is weak entity** because it **does not have any primary key .**
    - D_Name along with the Employee_No can uniqly identfy the records of Depends. So here the D_Name (Depends Name) is partial key.



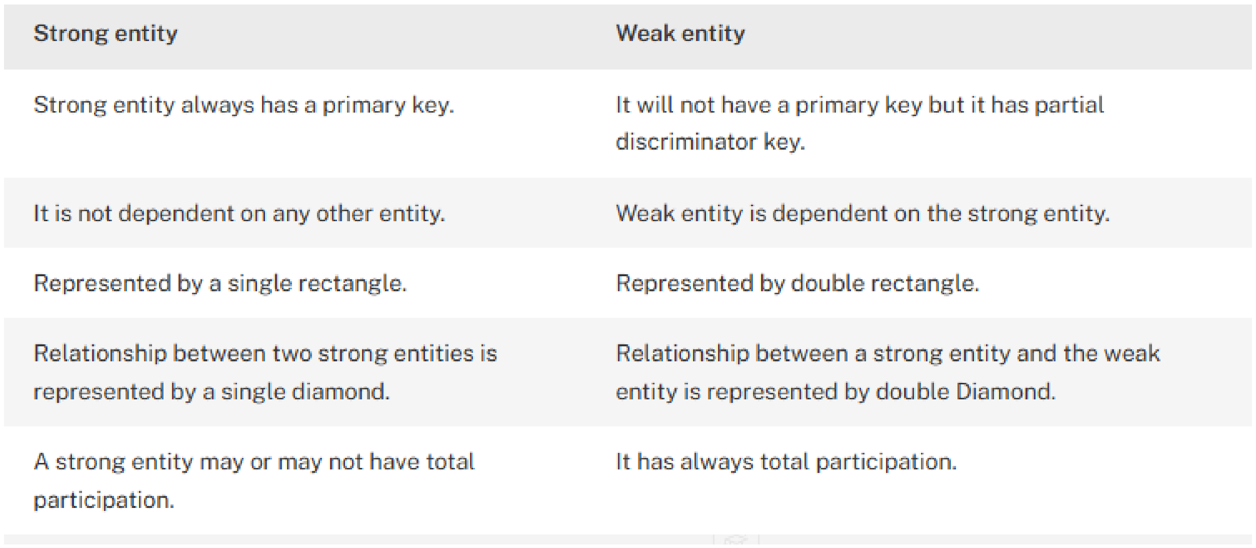
ImportantA lot of you can also argue that a course entity can not be created if student hasn't taken it so course would be weak entity.

This would be wrong way to think as -

If student s1 doesn't exist then courses c1, c2 ... others can exist as they may have taken courses or college chooses to have one course that is taken by none of student this semester.

But, if customer c1 doesn't exist then loan L1 for him can not exist at all as loan L1 is uniquely dependent on existence of customer C1

Strong entity vs Weak entity



Strong entity	Weak entity
Strong entity always has a primary key.	It will not have a primary key but it has partial discriminator key.
It is not dependent on any other entity.	Weak entity is dependent on the strong entity.
Represented by a single rectangle.	Represented by double rectangle.
Relationship between two strong entities is represented by a single diamond.	Relationship between a strong entity and the weak entity is represented by double Diamond.
A strong entity may or may not have total participation.	It has always total participation.

Generalization, Specialization and Aggregation in DBMS
What is the need for Generalization, Specialization and Aggregation in DBMS ?

With large databases it was realised that the ER model was become a little more complex and inconvenient to use.

Thus by programmers there were some additions in the ER model were suggested to reduce down on the complexity of the program, some new concepts were added which were –

1. Generalisation
2. Specialisation
3. Aggregation

We will learn more about them below –

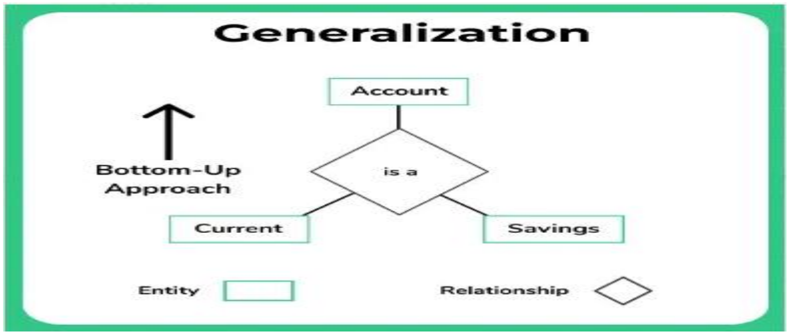
What is Generalization
In generalization we combine lower level entities to form a higher level entity. Thus its clear that it follows a bottom up approach.

Example –

In a bank there are two different types of accounts – Current and Savings, combine to form a super entity Account.

It thus follows system like classes, like super-classes and sub-classes right ?

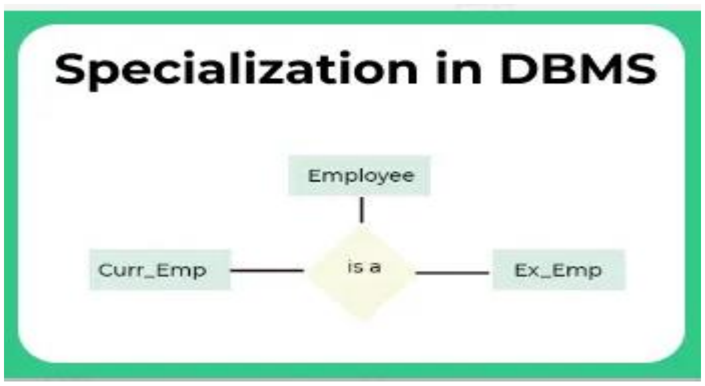
It may also be possible that the higher level entity may also combine with further entity to form a one more higher level entity.



What is Specialization
While generalization may follow a bottom up approach. **Specialization** is opposite to that, it follows a top down approach rather.

Example –

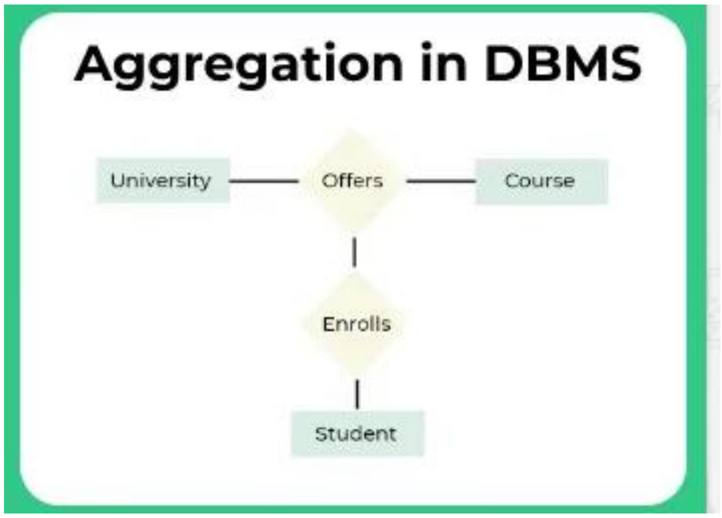
Employee may be decomposed to further as current employee entity and ex employee entity.



What is Aggregation
Aggregation is simply when we would consider two different entities as a single entity together.

Example –

University offering course can be considered a same entity, when viewed from a student entity perspective.



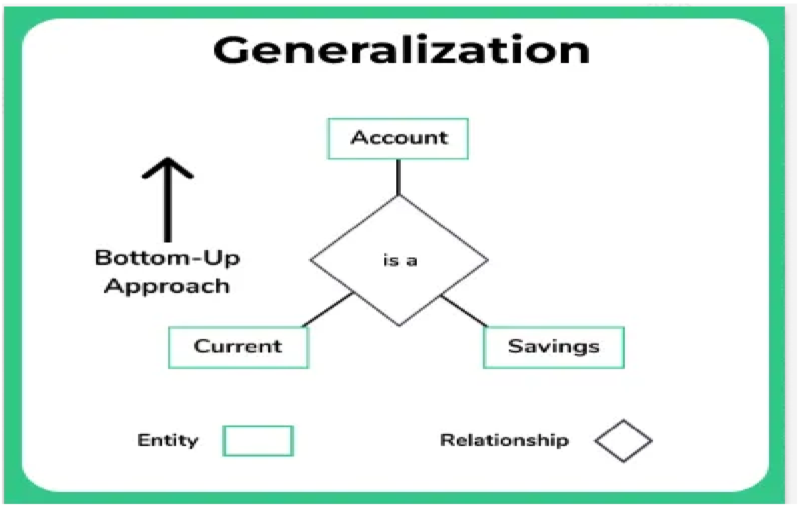
Generalization in DBMS
Generalization

Generalization is the term that is frequently used for the design process of any relational schema.

- **Definition:** Generalization is a bottom-up design process in which two or more entities of lower level or combined to form a higher level entity if they have some common attributes or properties.
- In these two or more entities integrated to form a generalized entity that sub-classes are combined to make a super-class.
- Generalization is viewed as a Reverse process of specialization.

Example:

Student, teacher, workers, doctor,…….are generalized as person.



Design steps for Generalization

- Define some entities with attributes.
- Identify common attributes between these entities and create a super-entity, so that all the remaining entities in some way related to the super-entity.
- Add Relationship between entities.

<u>Examples for Generalization</u>

- For example, we have entities as a teacher, workers, students, households…so on.

- These entities have some most common requirements name, salary, role, etc role of these categories can be generalized as a super common entity as citizens.

Conclusion:

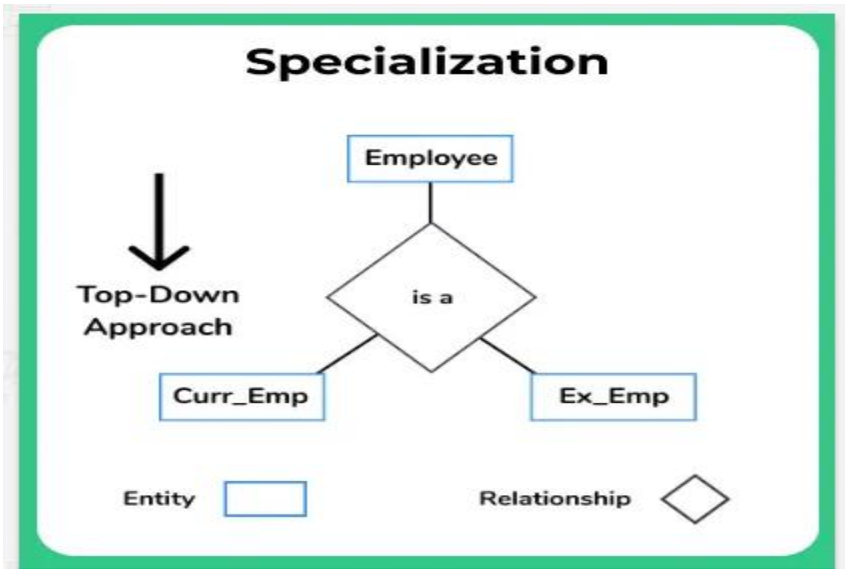
- Generalization is a ***bottom-up design approach.***
- Common properties from sub-entities are generalized to a super-entity.
- generalization ***is always applied to a group of entities and if over-designed it seems to reduce the schema size.***

Specialization in DBMS
Specialization

- **Definition**: Specialization is a top-down design approach where one higher level entity can be broken into two or more lower level entities so that subsets of entities that *share some distinguishing characteristics can be identified*
- The specialization can be viewed as a ***reverse design process for generalization***

Example: All types of employees in a company how common requirements or functionalities like name salary etc

The specialization is always done on a single entity *over design it increases the schema size*



Design steps for Specialization

- Generally, superclasses are defined first
- Define subclass and its related attributes
- IS-A Relationship is linked between these entities

Example for Specialization

- Consider an entity employee which consists of attributes name and salary, all types of employees like technician, engineer, accountant [sub entities] have different roles in the company, but all employees working in a company have some common functionalities or requirements like salary, id iD, name, etc i.e these properties are common to all the employees
- Identified common properties are inherited from parent entity (employee) to submit entities(tester, accountant, engineer)

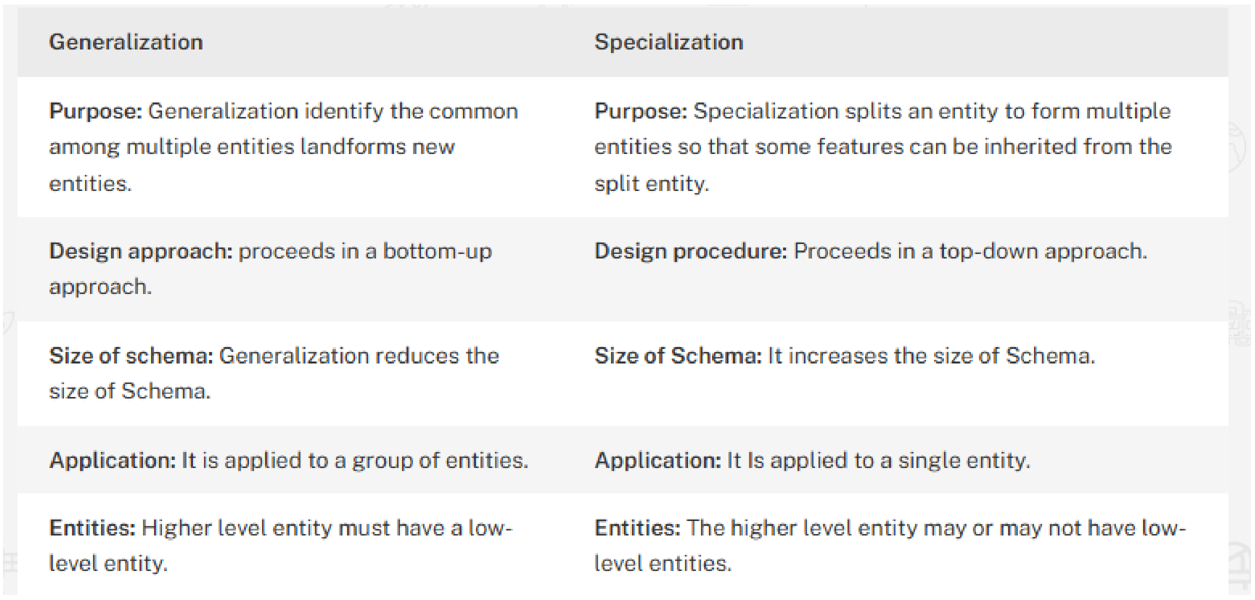
Conclusion:

- The main Idea behind specialization is the ***sharing of common properties between entities***
- Specialisation is a ***top-down*** design process

Generalization V/S Specialization

Both generalization specialization are design procedures and are equally important in designing a schema

Let’s understand the key differences between the two.



Generalization	Specialization
Purpose: Generalization identify the common among multiple entities landforms new entities.	**Purpose:** Specialization splits an entity to form multiple entities so that some features can be inherited from the split entity.
Design approach: proceeds in a bottom-up approach.	**Design procedure:** Proceeds in a top-down approach.
Size of schema: Generalization reduces the size of Schema.	**Size of Schema:** It increases the size of Schema.
Application: It is applied to a group of entities.	**Application:** It Is applied to a single entity.
Entities: Higher level entity must have a low-level entity.	**Entities:** The higher level entity may or may not have low-level entities.

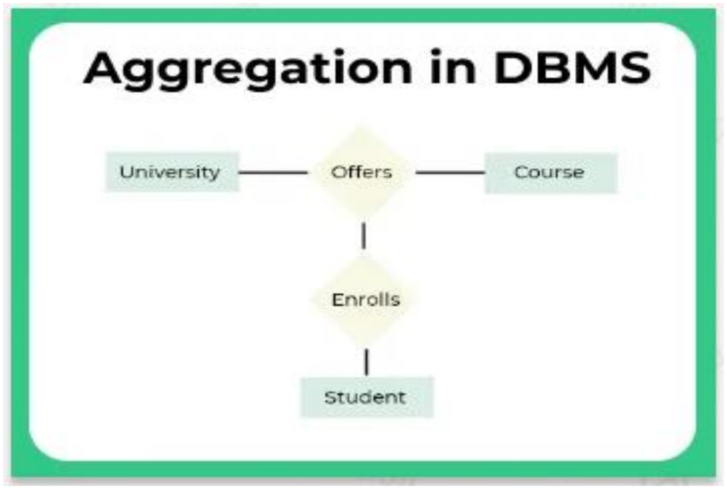
Aggregation in DBMS

What is Aggregation ?

- Aggregation is a design strategy in which the relationship is modelled between a collection of entities and another relationship.
- Simply it is used when **we need Express a relationship among other relationships.**

Example:

In Real-world situation for example if students visit a coaching institute then he shows interest not only to inquire about the course alone or not only just coaching centre, he will definitely enquire the details about both the coaching institute and the details of the concerned course



Design steps for aggregation

- Define entities and their attributes
- Add a relationship between these entities
- Define another entity so that the relationship can be established between the existing relationship and this entity

<u>Why aggregation</u>

- Aggregation is a process of compiling information on an object thereby abstracting higher-level object
- In SQL we need to find the sum of salaries of all the employees working in an organization or to find the highest-paid employee from all branches of the organization etc.

When using data in the form of numerical values, the following operations can be used to perform DBMS aggregation:

- **Average (AVG):** This function provides the mean or average of the data values.
- **Sum:** This provides a total value after the data values have been added.
- **Count:** This provides the number of records.
- **Maximum (Max):** This function provides the maximum value of a given set of data.
- **Minimum (Min):** This provides the minimum value of a given set of data.
- **Standard deviation (std dev):** This provides the dispersion or variation of the sets of data. Let's take a simple example of a database of student marks. If the standard deviation is high, it means the average is obtained by lower number of students than usual, and the lowest and highest marks are higher.

Characteristics of Aggregation
Aggregation is used when the DBMS has the following characteristics.

- **Many trivial entities:** A DBMS may consist of many entities that are not significant enough to provide meaningful information. In such a case, the trivial entities can be combined into one complex entity through aggregation. For example, many trivial entities called *rooms* can be combined to form a single entity called *hotel*.
- **One trivial entity:** Aggregation is also needed if a DBMS has a single trivial entity that should be used for multiple operations. In this case, the trivial entity is used to form relationships with other entities. This may lead to many aggregation entities depending on the operations required. For example, an employee in an organization may be given an insurance policy that covers his dependants. The entity *dependants* is a trivial entity because it cannot exist without the entity *employee*.
- **Inapplicable entity-model relationship:** The entity-model relationship cannot be applied to certain entities within the system. These specific entities can be combined with other entities to allow the application of the entity-model relationship in the entire

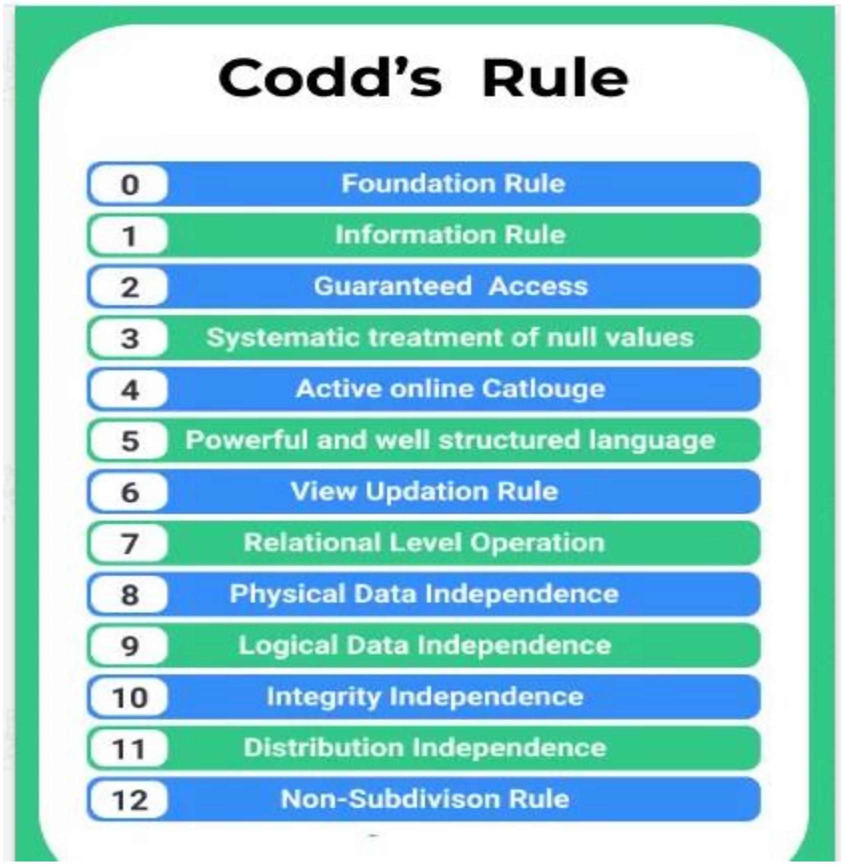
system. This ensures that all the entities in the system are utilized. For example, the entity-model relationship for students can only be applied if students enroll in a class. The entity *grade* can only be formed if the relationship *enroll* exists.

Relational Database Model

EF Codd’s Rules in DBMS

EF Codd’s rules in DBMS

- ***EF Codd*** is a computer scientist who first ***outlined the relational model*** which now became the most popular and one and only database model
- Codd Proposed 13 rules (listed from 0 to 12) popularly known as ***codd’s 12 rules*** which are used as a yardstick to test the quality of relational database management system(RDBMS)
- Till now none of the commercial product has followed all the 13 rules ***even Oracle has followed 8.5 out of 13***



Codd's Rule

0	Foundation Rule
1	Information Rule
2	Guaranteed Access
3	Systematic treatment of null values
4	Active online Catlouge
5	Powerful and well structured language
6	View Updation Rule
7	Relational Level Operation
8	Physical Data Independence
9	Logical Data Independence
10	Integrity Independence
11	Distribution Independence
12	Non-Subdivison Rule

Rule 0: Foundation rule

- If a system is said to be an RDBMS then the ***database should be managed using only relational capabilities***

Rule 1: Information rule

- All the ***data*** including metadata ***must be stored in some cell of the table*** in the form of rows and columns .

Rule 2: Guaranteed access

- Each ***data element*** in a table ***must be accessed through a combination of table name + primary key (row)+ attribute(column)***
- Example : emp+empid+ename,sal
- Strictly the ***data must not be accessed via a pointer .***

Rule 3: Systematic treatment of null values

- ***Null values*** represent different situations it may be ***missing data or not applicable or no value situation.***
- Null values ***must be handled consistently*** and also ***primary key must not be null*** and any expression on null must give null.

Rule 4: Active online catalog

- Database dictionary is a catalog which ***shows structural description of the complete database*** and it must be stored online
- This rule states that a database dictionary ***must be governed by the same rules and same query language as used for general database***.

Rule 5: Powerful and well-structured language

- The database should be accessible through a language **which supports definition, manipulation and all transaction management activities**, such a language is called structured language
- For example, SQL, ***if database uses a different language for data access and manipulation then it is a violation of the rule.***

Rule 6: View updation rule

- Difference ***views*** created for different purposes ***should be automatically updated*** by the system itself.

Rule 7: Relational level operation

- Operations like ***insert, delete and update operations must be supported*** at each level of relation even though it might be a ***nested relation or a complex relation***
- ***Set operations*** like union, intersection, minus ***must be supported.***

Rule 8: Physical data independence

- Any ***change in the physical location*** of the table ***should not reflect the change at the application level***
- Example: If you rename or move a file from one disk to another then it should not affect the application.

Rule 9: Logical data independence

- If there are any ***changes done to the logical structure*** of the database table, then ***users view of data should not be changed***
- If the ***table is split into two tables,*** then a ***new view should give result as the join of these two tables*** but this rule is very difficult to satisfy.

Rule 10: Integrity independence

- Database table should ***design itself on integrity*** rather than using external programs.
- It should use ***primary keys, check constants triggers***, etc which makes our DBMS independent of the front end application.

Rule 11: Distribution Independence

- Data distribution over various geographical locations over a network should not reflect the end-user i.e ***you should feel that all the data is stored in a single place***
- This rule ***laid the foundation for the distributed database.***

Rule 12: Non-Subversion Rule

- Any access given to the ***data that is present in the lowest level must not give a chance to authenticate constraints and change data*** ,this can be achieved through some kind of encryption techniques.

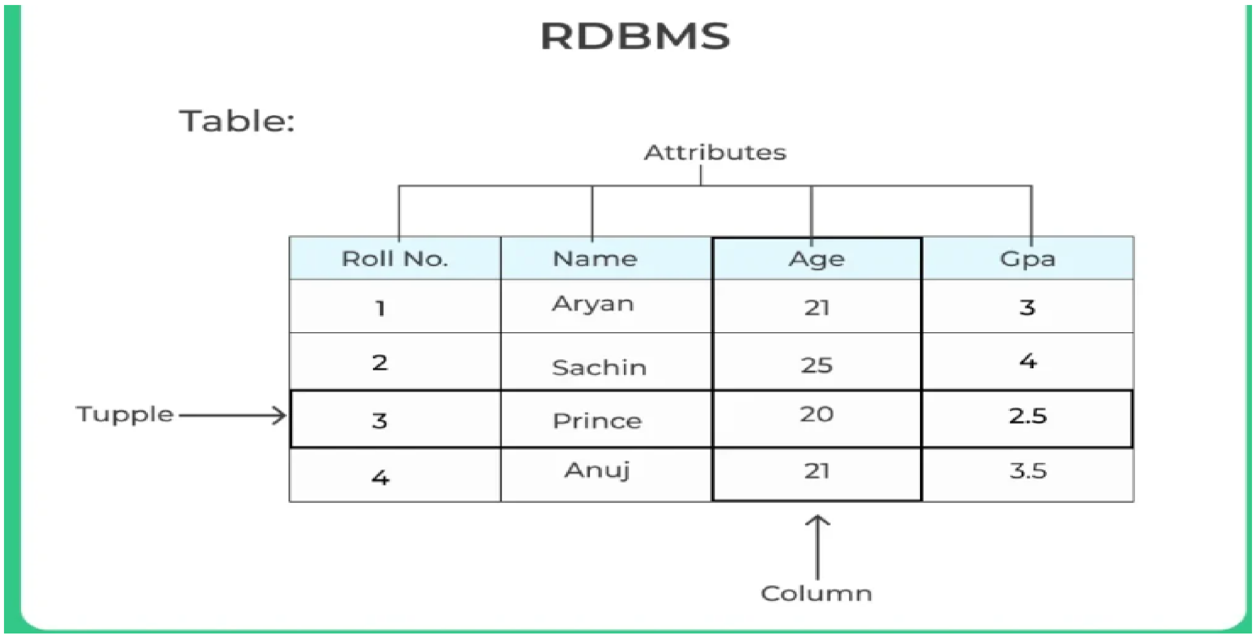
RDBMS
RDBMS?

RDBMS – Relational Database Management System, is a software system or collection of various programs that work together on a relational datamodel database and offer various integrated entities like – Database administration, Data definition, creation, updation etc. Some examples of RDBMS are –

- SQL
- mySQL
- IBM DB2
- Oracle
- Microsoft Access

RDBMS was introduced by E. F. Codd.

The basic structure of RDBMS revolves around tables like –



Roll No.	Name	Age	Gpa
1	Aryan	21	3
2	Sachin	25	4
3	Prince	20	2.5
4	Anuj	21	3.5

Column or Attribute
Column is all the values for a particular attribute for a table. This information is stored vertically.

For example – for students all the names as shown in the image will be the column.

Table
The data in RDBMS systems are stored inside tables.

•

  - In the first row, each column contains the attribute names of an entity
  - The next set of rows contain the data for an entity set object

Record or Row
The each row containing the data about a single entity in the entity set is called the record. This is also known as tuple as well.

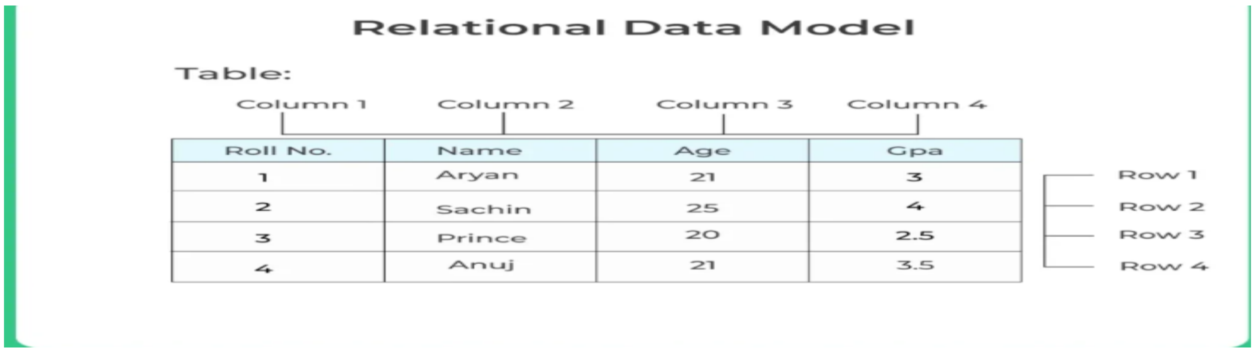
Example –

For students studying in a university the entity is student and information about roll number 1 is stored in the a row which has values of all information like name, age, gpa year etc.

Field
Fields are nothing but the list of all possible attributes for the table. Roll_No, name, age, GPA, year all are fields.

Relational Data Model
Realational Data Model

The basic idea between this relational data model is simple two-dimensional tables, also called as relations which consist of rows and columns.


Relational Data Model
Table:
Column 1
Column 2
Column 3
Column 4
Row 1
Row 2
Row 3
Row 4

Roll No.	Name	Age	Gpa
1	Aryan	21	3
2	Sachin	25	4
3	Prince	20	2.5
4	Anuj	21	3.5

Terms used in the relational model

Relation:

- Representing data in the form of a table consisting of rows and columns and relation both convey the same meaning.
- Student is a relation which consists of student records.

Schema:

- Schema is nothing but the organization of data in a table i.e in what order the data is logically aligned
- STUDENT relation can be represented as STUDENT (STUD_ID, NAME, PHONE, STATE, STUD_AGE).

Instance:

- Set of unique values present in the table at present is known as relational instance.

Attribute:

- An attribute defines the properties of a table that means what type of data that a table is storing ,these are nothing but a view of data in the form of columns
- In the relation student STUD_ID, NAME, PHONE, STATE, STUD_AGE correspond to the attributes of the relation.

Domain :

- A domain is nothing but the set of possible values that are allowed for a column in a database table
- For example, the age column in the student table is allowed to have values between 21 - 32.

Tuple :

- It is nothing but the row of the database table
- The student relation is having 4 tuples or rows.

Null values :

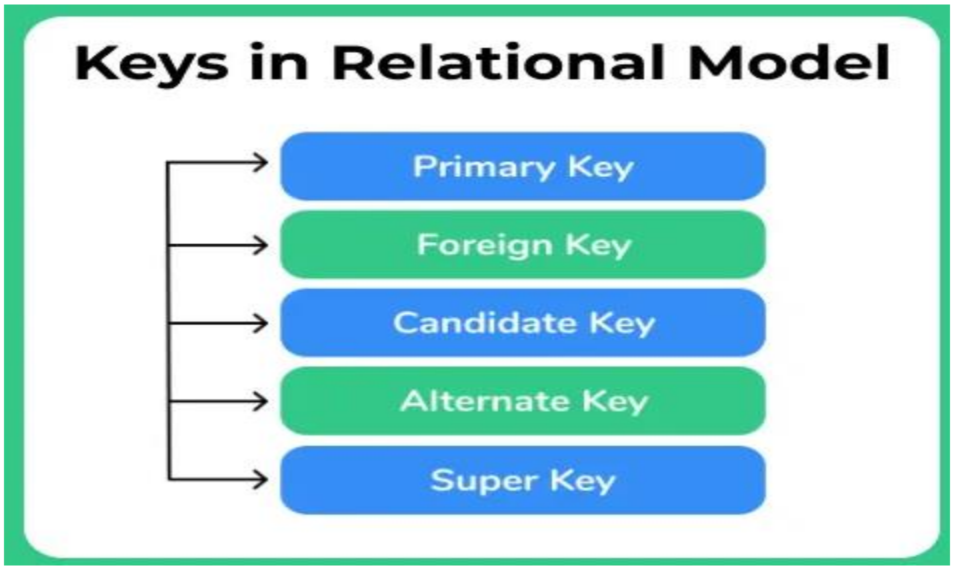
- There are certain situations where data may be unknown, missing or undefined which are represented by using this NULL
- A null value is different from zero and any operation on null value will result in null.

Advantages of the relational model

- Data integrity for accuracy and consistency
- No data redundancy
- Access control and integrity in the form of constraints which enables validation before entering and accessing the data
- Provides high security
- Supports to store any types(numbers, characters, date, images, audio, text files)
- Data can be managed and used by several users at a time
- Data can be shared across several platforms.

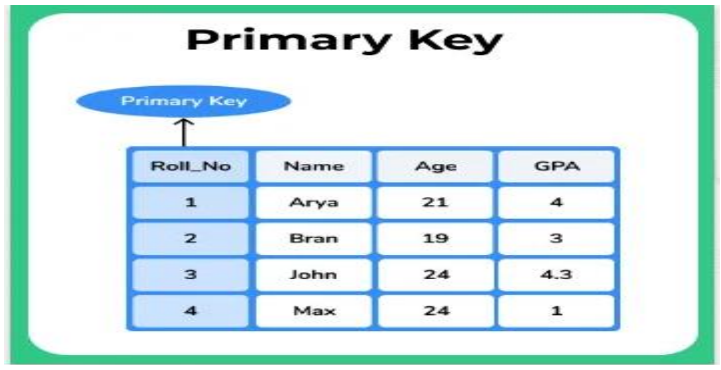
Keys in Relational Model

- Keys ensure the data integrity and consistency and helps to access a record uniquely
- The relational Model has the following keys
    - Primary Key
    - Foreign Key
    - Candidate Key
    - Alternate Key
    - Super Key



Primary Key

A primary key is a constraint in a table which uniquely identifies each row record in a database table by enabling one or more the column in the table as primary key.


Primary Key
Primary Key

Roll_No	Name	Age	GPA
1	Arya	21	4
2	Bran	19	3
3	John	24	4.3
4	Max	24	1

reating a primary key

A particular column is made as a primary key column by using the primary key keyword followed with the column name.

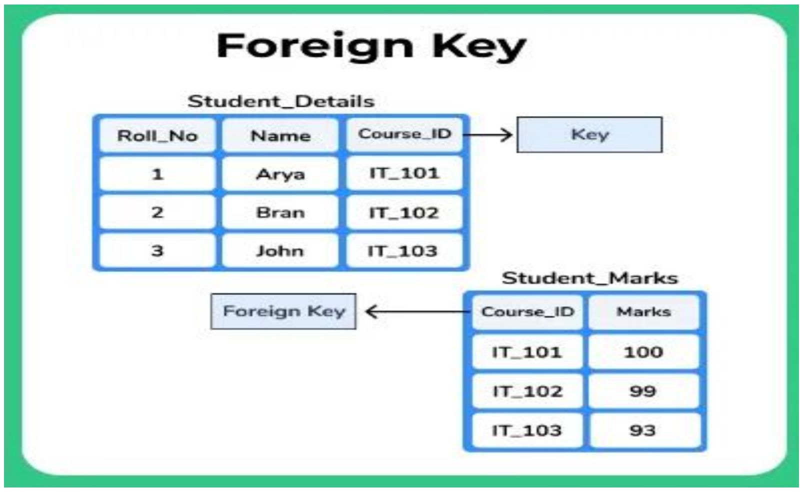
```
CREATE TABLE EMP (
  ID   INT     ,
  NAME VARCHAR (20) ,
  AGE  INT  ,
  COURSE VARCHAR(10) ,
  PRIMARY KEY (ID)
);
```

- Here we have used the primary key on ID column then ID column must contain unique values i.e ***one ID cannot be used for another student***.
- If you try to ***enter duplicate value while inserting in the row you are displayed with an error***
- Hence ***primary key will restrict you to maintain unique values and not null values in that particular column***

Foreign Key

- The foreign key constraint is a column or list of columns which points to the primary key column of another table

- The main purpose of the foreign key is only those values are allowed in the present table that will match to the primary key column of another table.



Example to create a foreign key

Reference Table

```
CREATE TABLE CUSTOMERS1(
  ID   INT ,
 NAME VARCHAR (20) ,
 COURSE VARCHAR(10) ,
  PRIMARY KEY (ID)
);
```

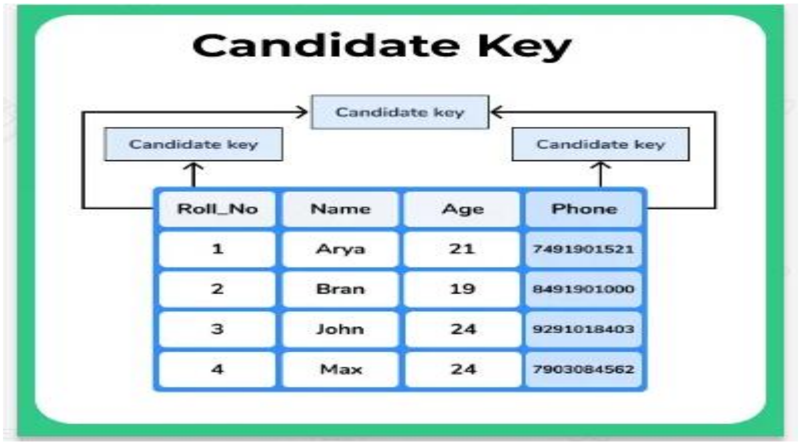
Child Table

```
CREATE TABLE CUSTOMERS2(
  ID   INT ,
  MARKS INT,
  REFERENCES CUSTOMERS1(ID)
);
```

Candidate Key

- Candidate keys are selected from the set of super keys, the only thing that you should remember while selecting candidate keys is, it should not have any redundant attitude

- Definition of candidate key: Super key with no redundant attributes known as candidate key i.e should not contain any column that contains duplicate data.



Roll_No	Name	Age	Phone
1	Arya	21	7491901521
2	Bran	19	8491901000
3	John	24	9291018403
4	Max	24	7903084562

<u>Example for Candidate Key</u>

- Consider the student table with columns [ID,NAME,PHONE]
- First, identify all the super keys present in the table, then eliminate the super keys that contain a column with duplicate data. Then the remaining superkeys that are left or nothing but candidate keys

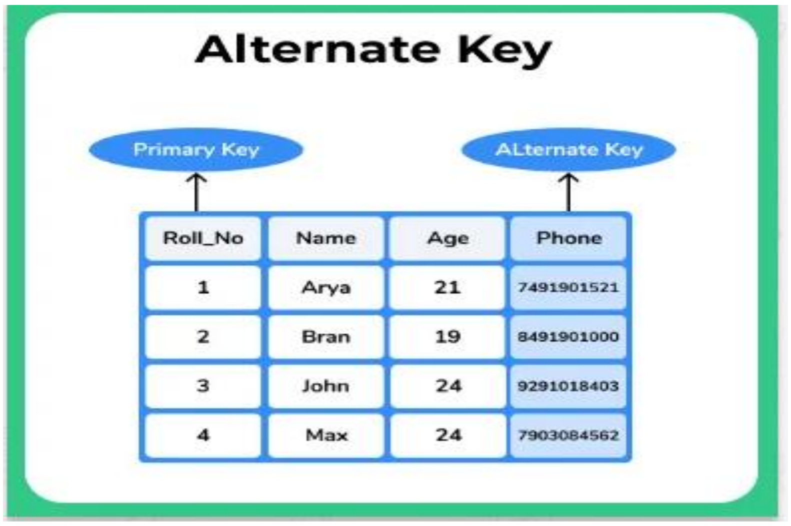
1. {Id}: **ID column will contain all unique values** hence ID column is a candidate key
2. {phone}: As ***no two students have the same phone number, it is not redundant*** data column and hence phone column is a candidate key
3. {Id, phone}: As both ***ID and phone are unique for all students*** this combination is a valid candidate key
4. {Id, Name}: This combination is not a candidate key because ***name column may have duplicate values***
5. {Id, phone, Name}: This combination is not a candidate key because ***name column may have duplicate values***
6. {Name, Phone}: This combination is not a candidate key because ***name column may have duplicate values***

Candidate keys available is the table student

- {Id}
- {phone}
- {Id, phone}

Alternate Key

Alternate keys are columns present in the table which are not selected as primary keys but still, they have all the capabilities to be used as a primary key is called alternate key.



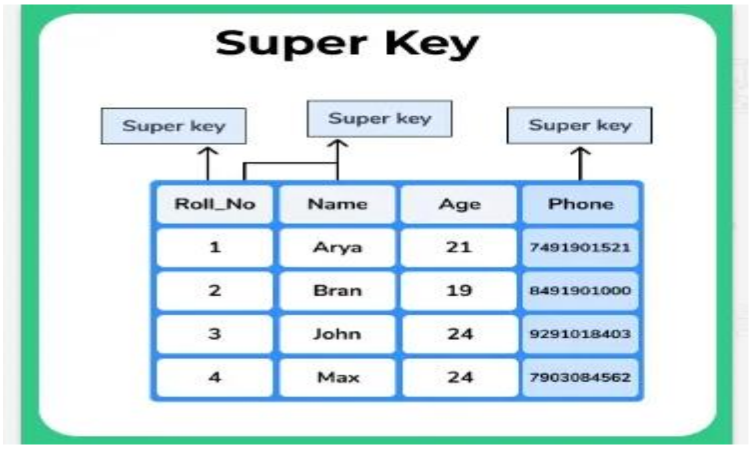
Roll_No	Name	Age	Phone
1	Arya	21	7491901521
2	Bran	19	8491901000
3	John	24	9291018403
4	Max	24	7903084562

<u>Examples for an alternate key</u>

- Example if a table student contain four columns [ID,NAME,PHONE,AGE]
- Among these four columns ***ID is used as a primary key because no two students will have the same Id*** and capable of uniquely accessing a student record in the table
- In the same way ***phone attribute, no two students will have same phone number***. Hence phone column is an alternate key because it is not been selected as a primary key, even if it is having the capability to be selected as a primary key

Super Key

- A super key is a group of single or multiple keys which uniquely identifies rows in a table.
- A Super key ***may have additional attributes that are not needed for unique identification.***



Roll_No	Name	Age	Phone
1	Arya	21	7491901521
2	Bran	19	8491901000
3	John	24	9291018403
4	Max	24	7903084562

<u>Example of the super key</u>
Consider the student table with columns [ID,NAME,PHONE]

-
    - [ID]: As ***no two students will have the same Id,*** it will help to uniquely access the student details, **hence it is a super key**Now we will identify all the Super Keys present in the table that is the set of attributes that will help to uniquely access a record
    - [NAME, ID]: Even if more than one student has the same name then ***the combination name will help to recognize the record, as student id will break the tie*** and this is combination is a super key
- [PHONE]: As ***no two students will have the same phone number,*** a phone number will help to uniquely access the student details and hence ***phone number is a super key***

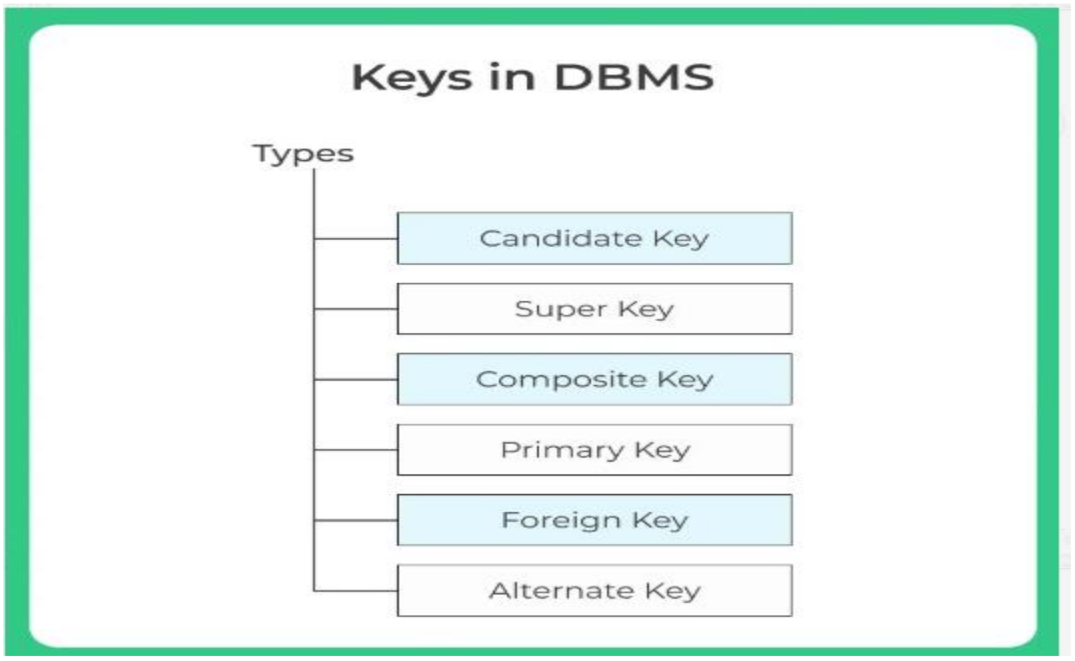
Keys in DBMS
Keys in DBMS?

Keys Identify any given tuple or record uniquely and also provide necessary functionality between various tables, in sql there are various types of keys, some of which are –

- Primary Key
- Foreign Key
- Candidate Key
- Super Key
- Alternate Key
- Composite Key

We also use keys as they surely help us in enforcing identity & integrity in the relationship.

Let’s have a look further to understand how each of them work –

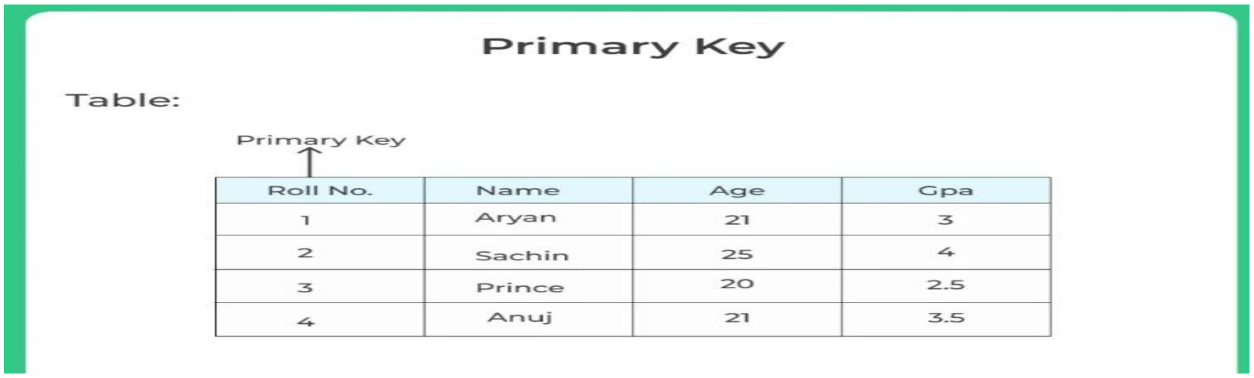


Primary Key in DBMS
What is a Primary Key?

A Primary Key is a constraint in a table which uniquely identifies each row record in a database table by enabling one or more the column in the table as primary key.

Primary Key in DBMS

Sometimes you need to maintain each and every row or record in a database table as unique. This purpose is served by using a primary key on that particular column


Primary Key
Table:
Primary Key

Roll No.	Name	Age	Gpa
1	Aryan	21	3
2	Sachin	25	4
3	Prince	20	2.5
4	Anuj	21	3.5

A primary key has the following properties

- A primary key column must contain unique values
- Null values not allowed for the primary key column
- Only ***o***ne primary key is allowed for a table.

Examples for primary keys

- Student ID in student table is a primary key because no two students will have the same Id
- Employee ID in employee table is a primary key because no two employees to have the same Id.
- Age, name, phone numbers will not make sense to be used as primary key columns because more than one person can have same age name our phone numbers, etc.

Creating a primary key

A particular column is made as a primary key column by using the primary key keyword followed with the column name

```
CREATE TABLE EMP
(
  Stu_ID   INT
  Stu_Name VARCHAR (20)
  Stu_Age  INT
  PRIMARY KEY (Stu_ID)
);
```

- Here we have used the primary key on ID column then ID column must contain unique values i.e one ID cannot be used for another student.
- If you try to enter duplicate value while inserting in the row you are displayed with an error
- Hence primary key will restrict you to maintain unique values and not null values in that particular column.

Students Table

Stu_ID	Stu_Name	Stu_Age
101	Steve	23
102	John	24
103	Robert	28
104	Steve	29
105	Carl	29

Primary key with more than one attribute

- Consider a table with three attitudes customer ID, product ID, product quantity
- Customer ID needs to be entered for each time the customer purchases an order hence customer ID appears more than once in the customer ID table hence it cannot be served as the primary key i.e it failed to uniquely identify a record
- Example customer ID 66 has placed two orders hence customer ID appeared 66 two times in the customer ID column

Customer_ID	Product_ID	Order_Quantity
66	9023	10
67	9023	15
68	9031	20

69	9031	18
66	9111	50

- Product ID and product quality cannot be declared as the primary key because of more than one customer purchase same product and the same quantity.
- In this situation all three attributes fail to serve as a primary key .Hence combination of these attribute can be used as a primary key
- For example [customer ID, product ID] can be used as the primary key table customers this combination helps to uniquely access records of customer

Example

```
Create table ORDER
(
   Customer_ID int ,
   Product_ID int ,
   Order_Quantity int ,
   Primary key (Customer_ID, Product_ID)
)
```

- While choosing a set of attributes for a primary key, we always choose the minimal set that has a minimum number of attributes.
- For example, if there are two sets that can identify a row in the table, the set that has a minimum number of attributes should be chosen as the primary key.

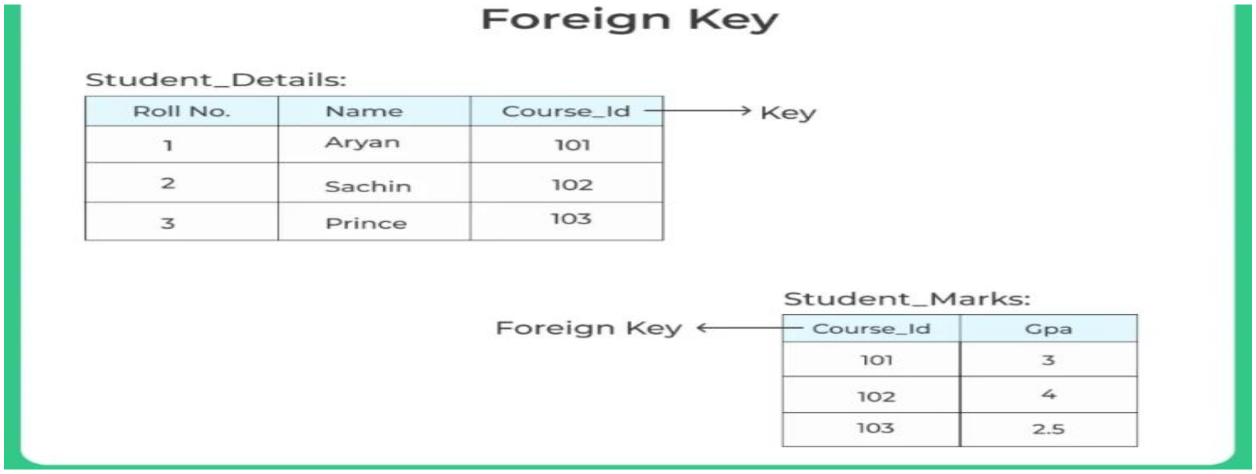
Foreign Key in DBMS
What is a Primary Key?

The foreign key constraint is a column or list of columns which points to the primary key column of another table.

Foreign Key in DBMS

- Sometimes you need to maintain and restrict only Same data in a table that is exactly the same column data of another table, this purpose is served by using a Foreign Key.

- The main purpose of the foreign key is only those values are allowed in the present table that will match to the primary key column of another table.



Roll No.	Name	Course_Id
1	Aryan	101
2	Sachin	102
3	Prince	103

Course_Id	Gpa
101	3
102	4
103	2.5

Example to create a foreign key

Reference Table

```
CREATE TABLE CUSTOMERS1(
  ID   INT ,
 DEPT VARCHAR (20)
  PRIMARY KEY (ID)
);
```

Child Table

```
CREATE TABLE CUSTOMERS2(
  ID   INT ,
  ADDRESS VARCHAR (20)
  REFERENCES CUSTOMERS1(ID)
);
```

CUSTOMERS1 table:

ID	DEPT
65	Dairy

66	Snacks
67	Snacks

<u>CUSTOMERS2 table:</u>

ID	ADDRESS
65	Hyderabad
66	Chennai
67	Hyderabad

- ID column in the customers1 table is used as a foreign key in the customers2 table which means all the ID values in customers2 must exist in the customers1 table
- An ID value that is not present in customers1 table is not allowed to be entered in the customers2 table ID column
- ID column of the customers1 table contains values as 65 66 67 now ID column in customers2 table must contain only these values that is 65 66 67 ,if the user enters other than this values in the ID column of the customers2 table it will raise an error because customers1 table id column is a foreign key in the customers2 table
- Hence we can observe that a link is maintained between two tables that is if you want to enter any data in the foreign key column table then we must add the data in the primary key column of the parent table if it is not present

Note:

- The column or list of the column that is used as foreign key in the present table must be a primary key in another table

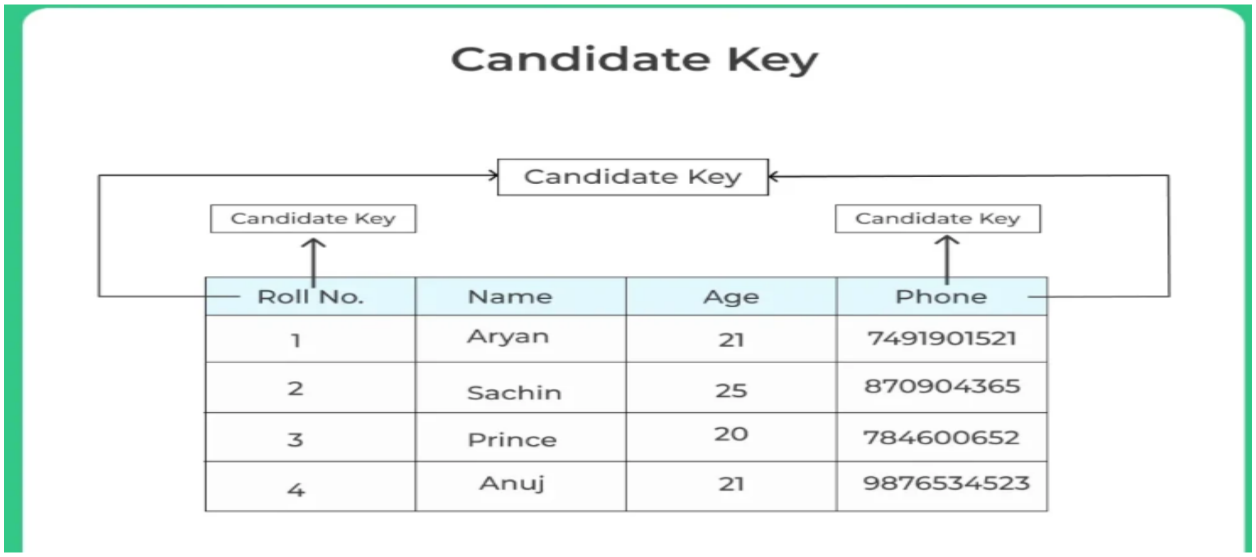
- The structure and data type of a PRIMARY KEY column of one table which used as a FOREIGN KEY in another table must be the same
- The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

Why foreign key ?

- A foreign key is used to prevent activities that would destroy the link between tables
- A foreign key prevents invalid data being inserted into the foreign key column because it restricts the user to enter only those values that are present in the primary key of another table.

Candidate Key in DBMS
Candidate Key in DBMS

- Definition of candidate key: Super key with no redundant attributes known as candidate key i.e should not contain any column that contains duplicate data.
- Hence candidate key also called as minimal super key.



Roll No.	Name	Age	Phone
1	Aryan	21	7491901521
2	Sachin	25	870904365
3	Prince	20	784600652
4	Anuj	21	9876534523

A candidate satisfies the following properties

- A candidate key column must be unique i.e all the columns that are involved in the candidate key also must be unique
- A candidate key may have more than one attribute
- A candidate key column will not contain any null value
- Candidate key always selects a minimum column combination that helps to uniquely identify the record.

Example for Candidate Key

Consider the following table student

ID	NAME	PHONE
66	Elena	9789578575
67	Haley	8758273875
68	Alex	7847326558
69	Elena	7923857563

First, identify all the super keys present in the table, then eliminate the super keys that contain a column with duplicate data. Then the remaining superkeys that are left or nothing but candidate keys

1. {Id}: ID column will contain all unique values hence ID column is a candidate key
2. {phone}: As no two students have the same phone number, it is not a redundant data column and hence phone column is a candidate key
3. {Id, phone}: As both ID and phone are unique for all students this combination is a valid candidate key
4. {Id, Name}: This combination is not a candidate key because the name column may have duplicate values

5. {Id, phone, Name}: This combination is not a candidate key because the name column may have duplicate values
6. {Name, Phone}: This combination is not a candidate key because the name column may have duplicate values

Candidate keys available in the table student

- {Id}
- {phone}
- {Id, phone}

Candidate key versus super key

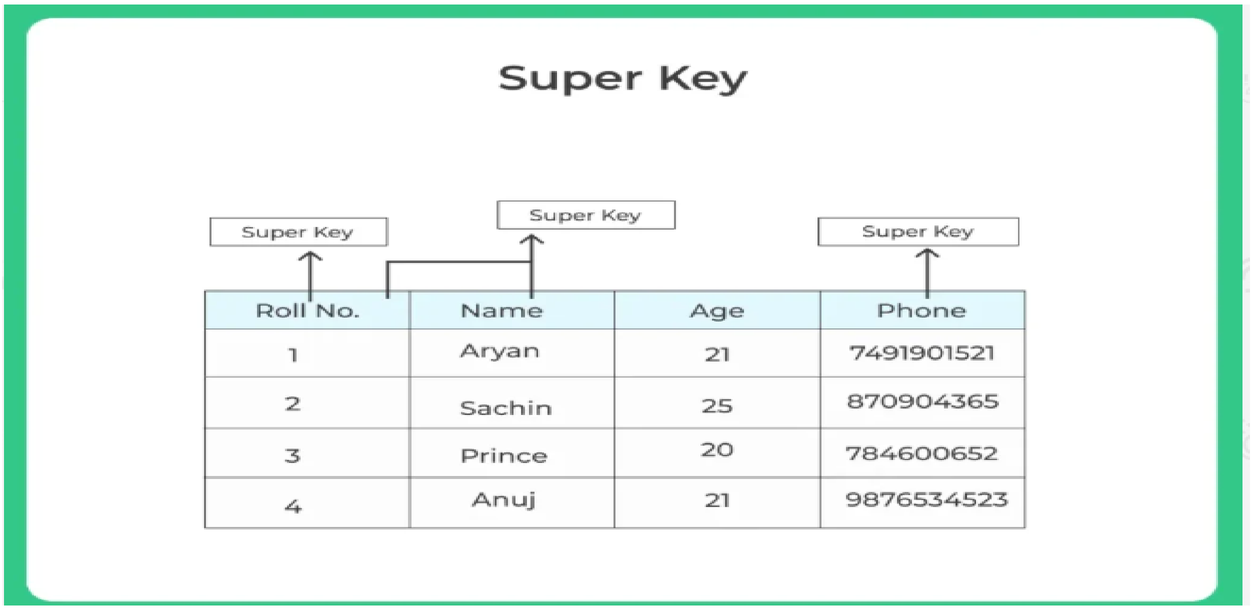
- First of all, you need to understand that all candidate keys are super keys this is because candidate keys are selected from super keys
- Look for those keys from which we can remove any columns that contain duplicate data. In the above example, we have not chosen {Id, name} as candidate key because {Id} alone can identify a unique row in the table, and {Name} column is redundant.

Super Key in DBMS

Super Key in DBMS

- A super key is a group of single or multiple keys which uniquely identifies rows in a table.
- A Super key may have additional attributes not needed for unique identification.

More detailed DefinitionA super key as a set of those keys that identify a row or a tuple uniquely. The word super denotes the superiority of a key. Thus, a super key is the superset of a key known as a Candidate key. It means a candidate key is obtained from a super key only



Roll No.	Name	Age	Phone
1	Aryan	21	7491901521
2	Sachin	25	870904365
3	Prince	20	784600652
4	Anuj	21	9876534523

Example of the super key

Consider the following table student

ID	NAME	PHONE	AGE
66	John	7832973477	23
67	Judy	9732867464	21
68	Elena	8923826476	25
69	John	7326247844	23

The possible set of SuperKeys –

[ID]:

As*no two students will have the same Id***,** it will help to uniquely access the student details, hence it is a super key. Now we will identify all the Super Keys present in the table that is the set of attributes that will help to uniquely access a record

[NAME, ID]:

Even if more than one student has the same name then the combination name will help to recognize the record, as student id will break the tie and this is combination is a super key

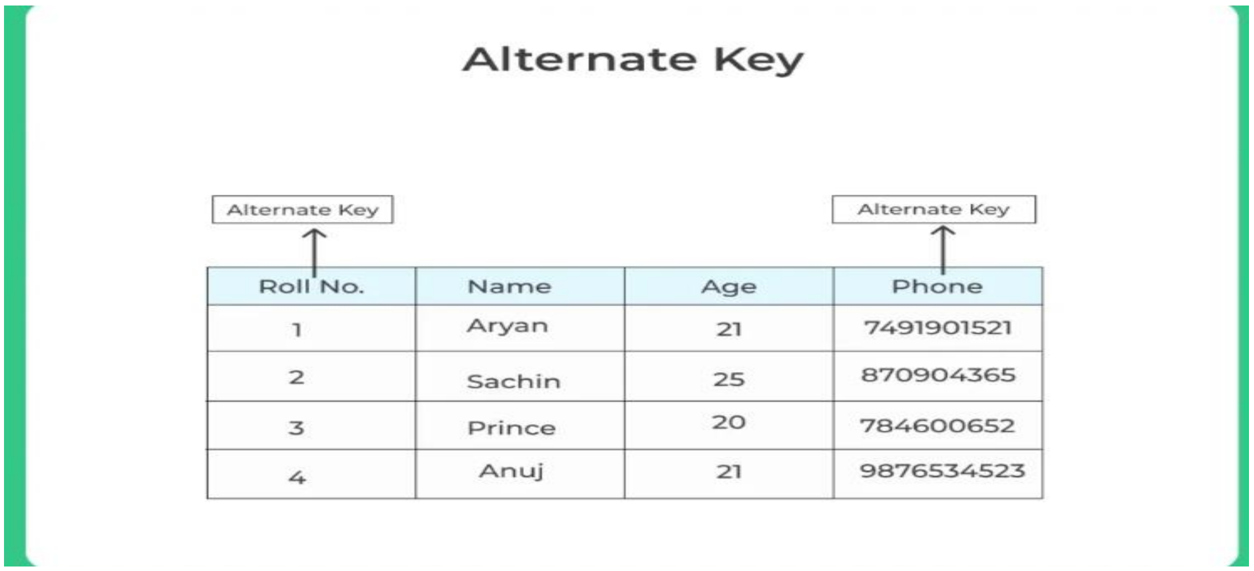
[PHONE]:

As no two students will have the same phone number, a phone number will help to uniquely access the student details and hence phone number is a super key

Alternate Key in DBMS
Alternate keys are columns present in the table which are not selected as primary keys but still, they have all the capabilities to be used as a primary key is called alternate keys.

Alternate Key in DBMS

Sometimes more than one column is having the capability to uniquely identify a row but they might not be selected as primary keys this type of key on nothing but Alternate Key.



Roll No.	Name	Age	Phone
1	Aryan	21	7491901521
2	Sachin	25	870904365
3	Prince	20	784600652
4	Anuj	21	9876534523

Examples for an alternate key
Example if a table student contain four columns

[ID,NAME,PHONE,AGE]

Consider the following table student

ID	NAME	PHONE	AGE
66	John	7832973477	23
67	Judy	9732867464	21

68	Elena	8923826476	25
69	John	7326247844	23

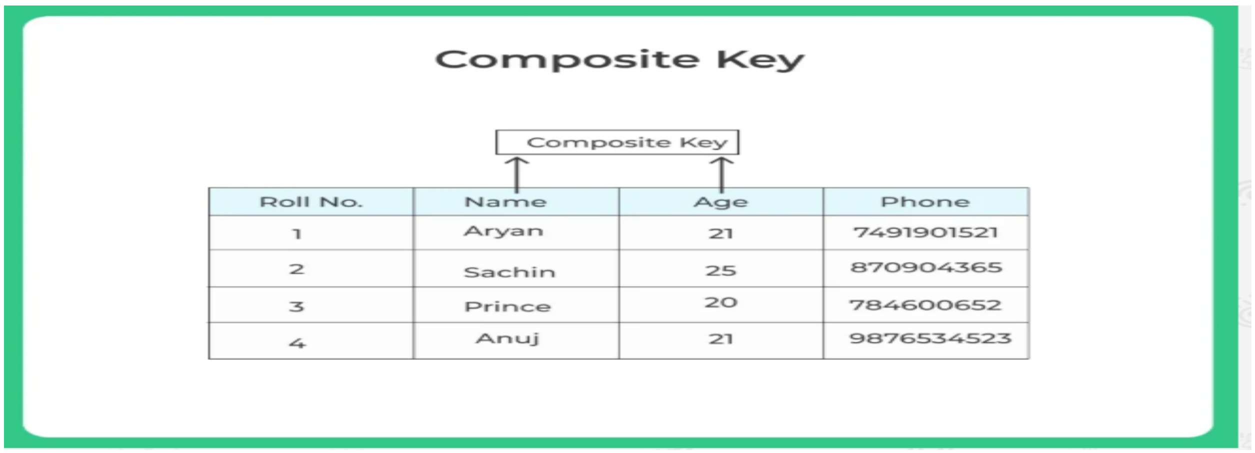
- Among these four columns ID is used as a primary key because no two students will have the same Id and capable of uniquely accessing a student record in the table
- In the same way phone attribute, no two students will have same phone number.Hence phone column is an alternate key because it is not been selected as a primary key, even if it is having the capability to be selected as a primary key

Composite Key in DBMS

Composite key is a candidate key, which is combination of two or more attributes of a table to uniquely identify occurrence an entity.

Composite Key in DBMS

A Composite Key is a combination of one or more attributes. If a single column alone fails to be served as a primary key then combination columns would help to uniquely access a record from table such type of keys or nothing but composite keys.


Composite Key
Composite Key

Roll No.	Name	Age	Phone
1	Aryan	21	7491901521
2	Sachin	25	870904365
3	Prince	20	784600652
4	Anuj	21	9876534523

Example

If a table contains three columns [name, address, course] individually feel to access the record uniquely combination of either [name, course],[name, address],[course, address] would help to access records uniquely

- Consider a table with three attitudes customer ID, product ID, product quantity

- Customer ID needs to be entered for each time the customer purchases an order hence customer ID appears more than once in the customer ID table hence it cannot be served as the primary key i.e it failed to uniquely identify a record
- Example customer ID 66 has placed two orders hence customer ID appeared 66 two times in the customer ID column

customer ID	product ID	product quantity
66	9023	10
67	9023	15
68	9031	20
69	9031	18
66	9111	50

- Product ID and product quality cannot be declared as the primary key because of more than one customer purchase same product and the same quantity
- In this situation all three attributes fail to serve as a primary key .Hence combination of these attribute can be used as a primary key
- For example [customer ID, product ID] can be used as the primary key table customers this combination helps to uniquely access records of customer.

```
Create table ORDER
(
   Customer_ID int ,
   Product_ID int ,
   Order_Quantity int ,
   Primary key (Customer_ID, Product_ID)
)
```

- While choosing a set of attributes for a primary key, we always choose the minimal set that has a minimum number of attributes.

- For example, if there are two sets that can identify a row in the table, the set that has a minimum number of attributes should be chosen as the primary key.

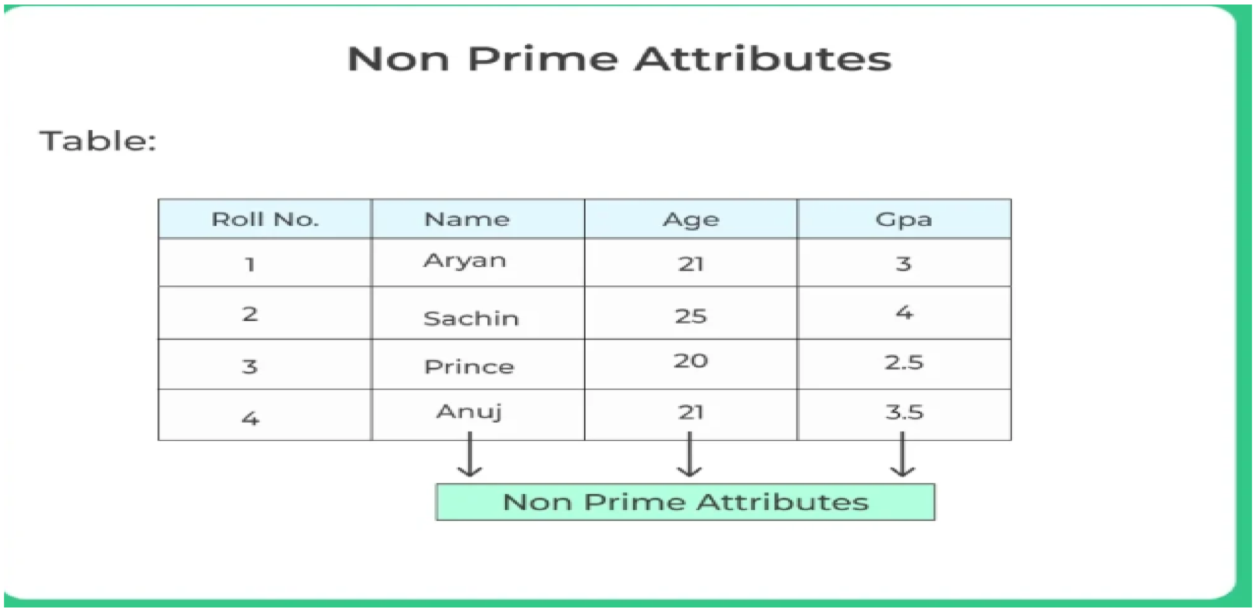
Non Prime Attributes in DBMS

What are Attributes?

Attributes are the descriptive properties that each entity in an Entity Set possesses. Each attribute has a distinct domain or set of values from which it can derive its values.

Non Prime Attributes in DBMS

Attributes of the relation which does not exist in any of the possible candidate keys of the relation, such attributes are called non prime attributes. Non prime attributes also called as Non Key attributes.



Non Prime Attributes

Table:

Roll No.	Name	Age	Gpa
1	Aryan	21	3
2	Sachin	25	4
3	Prince	20	2.5
4	Anuj	21	3.5

Examples :
Consider the following table:

ID	NAME	AGE	CITY	PHONE_NO
67	Luke	22	Delhi	8353355366
68	Haley	19	Noida	7323598665

69	Luke	21	Mumbai	8656864365

- [ID] : Every individual will have a ***unique ID*** so it is a prime attribute.
- [NAME] : Two different persons ***might have same name***, so it is non prime attribute
- [AGE] : Two different persons ***might have same age***, so it is non prime attribute
- [CITY] : Two different persons ***might be living in the same city***, so it is non prime attribute
- [PHONE_NO] : ***No two persons will have same phone number***, so it is not a non prime attribute

SQL Constraints

Default in DBMS

DEFAULT IN DBMS

- Suppose in real time you are having a column in a table where the majority of the data values ***that column is frequently repeated*** and in that case why should you enter the same value, again and again, each time while entering a row
- For example, if Hyderabad is the city for all employees in the company then a city column with Hyderabad value is the ***default column in the table that is you need not enter this value you whenever you insert a new row.***

DEFAULT clause

- Default clause in SQL is used to add default data to the columns
- When a column is specified as default with some value then all the rows will use the same value i.e each and every time while entering the data we need not enter that value
- But ***default column value can be customized*** i.e it can be overridden when inserting a data for that row based on the requirement.

Example for DEFAULT clause

The following SQL sets a DEFAULT value for the "city" column when the "emp" table is created:

My SQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE emp (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'hyderabad'
);
```

Upon using a select query on the emp table following data is displayed

ID	LastNmae	FirstName	Age	City
66	Dunphy	Alex	21	hyderabad
67	Tucker	Lily	22	hyderabad
68	Dunphy	Luke	22	hyderabad
69	Pritchett	Alex	23	hyderabad

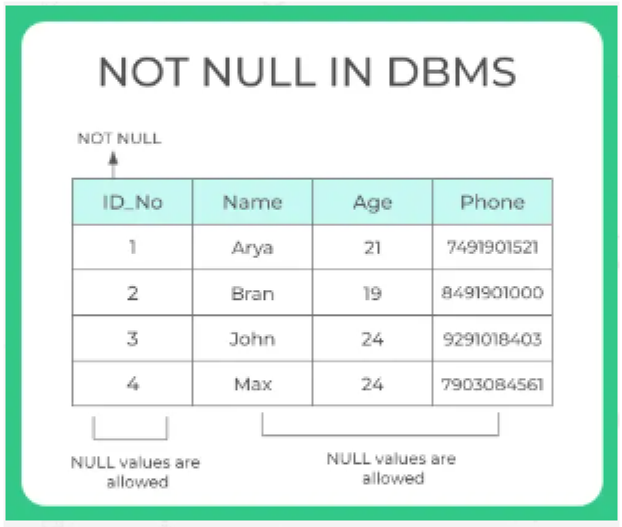
- You can see that all the values of the ***city column are same i.e Hyderabad*** this is because we have set city column as default column with a default value of Hyderabad
- As a result, whenever you insert a new row each time you need not enter a value for this default column that is ***entering a column value for a default column is optional and if you don't enter the same value is considered that is used in the default clause***

Not Null in DBMS

NOT NULL IN DBMS

- Null value is different from zero value or a field that contain spaces
- Null represents a record where data may be missing data or data for that record may be optional
- Not all constraints prevents a column to contain null values

- Once ***not null is applied to a particular column, you cannot enter null values to that column*** and restricted to maintain only some proper value other than null
- A ***not-null constraint cannot be applied at table level***



ID_No	Name	Age	Phone
1	Arya	21	7491901521
2	Bran	19	8491901000
3	John	24	9291018403
4	Max	24	7903084561

<u>Syntax</u>

```
CREATE TABLE table_name
(
  column1 datatype(size),
  column2 datatype(size),
  column3 datatype(size) NOT NULL ,
);
```

<u>Example</u>

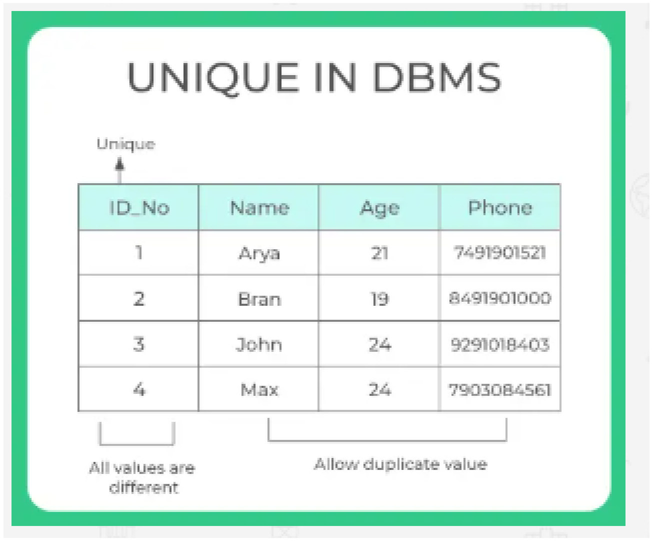
```
CREATE TABLE STUDENT
(
  ID   INT            NOT NULL,
  NAME VARCHAR (20)     NOT NULL,
  AGE  INT            NOT NULL,
  ADDRESS  CHAR (25) ,
  SALARY   DECIMAL (18, 2),
  PRIMARY KEY (ID)
);
```

- In the above example, we have applied not null on three columns ID, name and age which means ***whenever a record is entered using insert statement all three columns should contain a value other than null***
- We have two other columns address and salary, ***where not null is not applied*** which means that ***you can leave the row as empty or use null value while inserting the record into the table***

Unique in DBMS

UNIQUE IN DBMS

- Sometimes we need to maintain only unique data in the column of a database table, this is possible by using a unique constraint
- Unique constraint ensures that all values in a column are unique
- If we unique constraint is applied to a column then all the values present the column must be unique i.e if you try to enter a duplicate data then will be shown an error
- A unique key can be applied to any number of columns in a table.


UNIQUE IN DBMS
Unique
All values are different
Allow duplicate value

ID_No	Name	Age	Phone
1	Arya	21	7491901521
2	Bran	19	8491901000
3	John	24	9291018403
4	Max	24	7903084561

Syntax to create a unique constraint

```
CREATE TABLE table_name(
   column1 datatype(size) UNIQUE,
   column2 datatype(size) ,
   column3 datatype(size) ,
);
```

Example

```
CREATE TABLE Persons (
    ID int UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
);
```

In the above example, ***as we have used unique constraint on ID column we are not supposed to enter the data that is already present***, simply no two ID values are same.

Combination of not null and unique

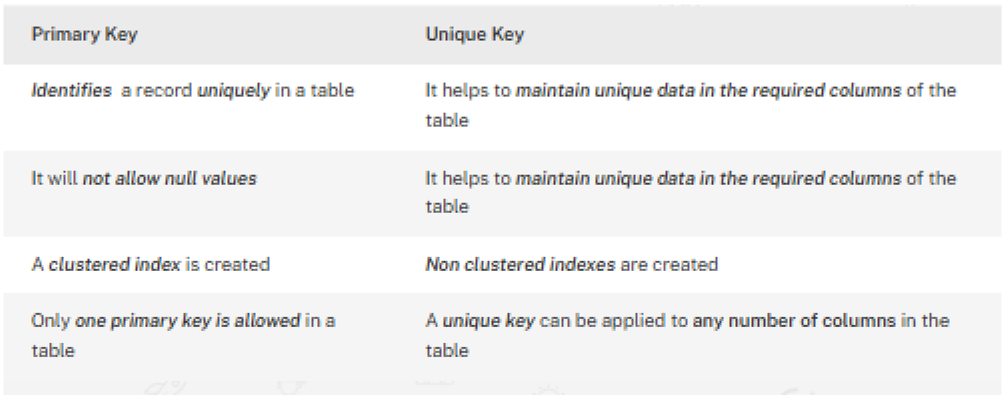
- You ***can use multiple constraints on a single column***
- Whenever you apply both ***not null and unique constraints to a column it restricts to have only unique data values and also show it prevents null values at the same time.***

Example

```
    ID int NOT NULL UNIQUE
```

Primary key vs Unique key

- Both primary key and unique key will not allow duplicate values but the difference is that, the ***primary key will not accept null values*** where as ***unique key allows null values on the column***
- Simply only one unique + not null= primary key



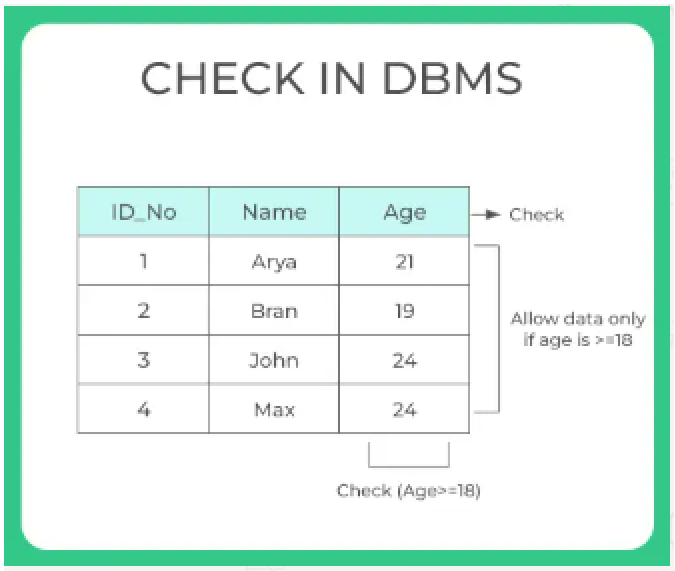
Primary Key	Unique Key
Identifies a record *uniquely* in a table	It helps to *maintain unique data in the required columns* of the table
It will *not allow null values*	It helps to *maintain unique data in the required columns* of the table
A *clustered index* is created	*Non clustered indexes* are created
Only *one primary key is allowed* in a table	A *unique key* can be applied to **any number of columns** in the table

CHECK IN DBMS

- Suppose in real-time if you want to give access to an application only if the age entered by the user is greater than 18 this is done at the back-end by using a check constraint
- Check constraint ensures that the data entered by the user for that column is within the range of values or possible values specified.

Syntax for check constraint

```
CREATE TABLE STUDENT (
    column1 datatype(size),
    column2 datatype(size),
    column3 datatype(size),
   CHECK (condition)
);
```



ID_No	Name	Age
1	Arya	21
2	Bran	19
3	John	24
4	Max	24

Example for check constraint

```
CREATE TABLE STUDENT (
   ID int ,
   Name varchar(255) ,
   Age int,
   CHECK (Age>=18)
);
```

- As we have used a ***check constraint as (Age>=18)*** which means ***values entered by the user for this age column while inserting the data must be less than or equal to 18*** otherwise an error is shown
- Simply, the only possible values that the ***age column will accept is [0 -17].***

Check constraint at table level

```
CREATE TABLE student {
  ID int ,
  Name varchar(255) ,
  Program interface specificationAge int,
  City varchar(255),
  CONSTRAINT CHK_Person CHECK (Age>=18 AND City='hyderabad')
);
```

- You ***can apply check constraint to more than one column at a time***
- In the above example we have used two conditions with the check constraint ***Age>=18 AND City=' Hyderabad'*** , as a result, both the columns age and city will be checked while data is entered i.e the ***only possible values that are accepted for the column age are 0 -17 and city must be only Hyderabad***

Key Constraints in DBMS

CHECK IN DBMS

- Constraints or nothing but the rules that are to be followed while entering data into columns of the database table
- Constraints ensure that data entered by the user into columns must be within the criteria specified by the condition

- For example, if you want to maintain only unique IDs in the employee table or if you want to enter only age under 18 in the student table etc
- We have 5 types of key constraints in DBMS
    - NOT NULL: ensures that the specified ***column doesn't contain a NULL value.***
    - UNIQUE : ***provides a unique/distinct values*** to specified columns.
    - DEFAULT: ***provides a default value to a column*** if none is specified.
    - CHECK :***checks for the predefined conditions before inserting*** the data inside the table.
    - PRIMARY KEY: it ***uniquely identifies a row*** in a table.
    - FOREIGN KEY: ensures ***referential integrity*** of the relationship

Not Null

- Null represents a record where data may be missing data or data for that record may be optional
- Once ***not null is applied to a particular column, you cannot enter null values to that column*** and restricted to maintain only some proper value other than null
- A ***not-null constraint cannot be applied at table level***

Example

```
CREATE TABLE Orders (
   OrderID int NOT NULL,
   OrderNumber int NOT NULL,
   PersonID int,
   PRIMARY KEY (OrderID),
   FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

- In the above example, we have applied not null on three columns ID, name and age which means ***whenever a record is entered using insert statement all three columns should contain a value other than null***

- We have two other columns address and salary, ***where not null is not applied*** which means that ***you can leave the row as empty or use null value while inserting the record into the table***

Unique

- Sometimes we need to maintain only unique data   in the column of a database table, this is possible by using a unique constraint
- Unique constraint ensures that all values in a column are unique

Example

```
CREATE TABLE Persons (
   ID int UNIQUE,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
);
```

In the above example, ***as we have used unique constraint on ID column we are not supposed to enter the data that is already present***, simply no two ID values are same

DEFAULT

- Default clause in SQL is used to add default data to the columns
- When a column is specified as default with some value then all the rows will use the same value i.e each and every time while entering the data we need not enter that value
- But ***default column value can be customized*** i.e it can be overridden when inserting a data for that row based on the requirement.

Example for DEFAULT clause

The following SQL sets a DEFAULT value for the “city” column when the “emp” table is created:

My SQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE emp (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   City varchar(255) DEFAULT 'hyderabad'
);
```

- As a result, whenever you insert a new row each time you need not enter a value for this default column that is ***entering a column value for a default column is optional and if you don't enter the same value is considered that is used in the default clause***

Check

- Suppose in real-time if you want to give access to an application only if the age entered by the user is greater than 18 this is done at the back-end by using a check constraint
- Check constraint ensures that the data entered by the user for that column is within the range of values or possible values specified.

Example for check constraint

```
CREATE TABLE STUDENT (
   ID int ,
   Name varchar(255) ,
   Age int,
   CHECK (Age>=18)
);
```

- As we have used a ***check constraint as (Age>=18)*** which means ***values entered by the user for this age column while inserting the data must be less than or equal to 18*** otherwise an error is shown
- Simply, the only possible values that the ***age column will accept is [0 -17]***

Primary Key

A primary key is a constraint in a table that uniquely identifies each row record in a database table by enabling one or more the columns in the table as the primary key.

Creating a primary key

A particular column is made as a primary key column by using the primary key keyword followed with the column name

```
CREATE TABLE EMP (
  ID   INT
  NAME VARCHAR (20)
  AGE  INT
  COURSE VARCHAR(10)
  PRIMARY KEY (ID)
);
```

- Here we have used the primary key on ID column then ID column must contain unique values i.e ***one ID cannot be used for another student***.
- If you try to ***enter duplicate value while inserting in the row you are displayed with an error***
- Hence ***primary key will restrict you to maintain unique values and not null values in that particular column***

Foreign Key

- The foreign key a constraint is a column or list of columns that points to the primary key column of another table
- The main purpose of the foreign key is only those values are allowed in the present table that will match the primary key column of another table.

Example to create a foreign key

Reference Table

```

CREATE TABLE CUSTOMERS1(
   ID   INT ,
   NAME VARCHAR (20) ,
   COURSE VARCHAR(10) ,
  PRIMARY KEY (ID)
);
```

Child Table

```
CREATE TABLE CUSTOMERS2(
  ID   INT ,
  MARKS INT,
  REFERENCES CUSTOMERS1(ID)
);
```

DOMAIN CONSTRAINTS IN DBMS

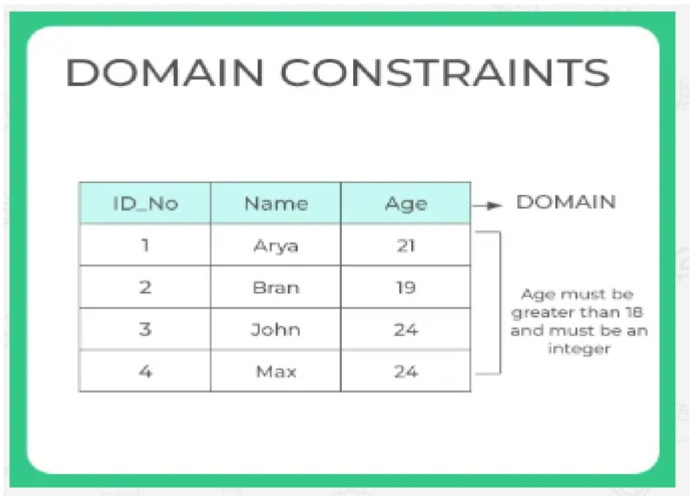
- DBMS table is viewed as a combination of rows and columns.

For example, if you are having a column called month and you want ***only [Jan, Feb, March….dec] as values allowed to be entered for that particular column which is referred to as domain for that particular column.***

Or For Example, in the age column, it must only accept students greater than 18 age or roll number should be a positive number.

Definition :
Domain constraint ensures two things

-
  - It makes sure that the data value entered for that particular column matches with the data type defined for that column
  - It shows that the constraints((NOT NULL / UNIQUE / PRIMARY KEY / FOREIGN KEY / CHECK / DEFAULT)) ***impose on that column not fulfilled or not***



ID_No	Name	Age
1	Arya	21
2	Bran	19
3	John	24
4	Max	24

Syntax for domain constraints Domain Constraint = data type check for the column + Constraints (NOT NULL / UNIQUE / PRIMARY KEY / FOREIGN KEY / CHECK / DEFAULT)

Example :

For example, we want to create a table “student” with “stu_id” field having a value greater than 100, can create a domain and table like this:

Constraint Creation Template

```
create domain domain_name int
constraint constraint_name
check(write codition here);
```

Constraint Creation Example

```
create domain age_constraint int
constraint age_test
check(value > 18);
```

Table Example

```
create table student (
student_id PRIMARY KEY,
student_name varchar(30),
student_age age_constraint int
);
```

Example 2

Constraint Creation Example

```
create domain roll_no_constraint int
constraint roll_test
```

```
check(value > 0);
```

Table Example

```
create table student (
RollNo roll_no_constraint PRIMARY KEY,
StudentName varchar(30),
StudentPhone int
);
```

MAPPING CONSTRAINTS IN DBMS

Mapping constraints defines how many entities can be related to another entity to a relationship. It is very much useful for identifying relationships that are involved with more than one relationship

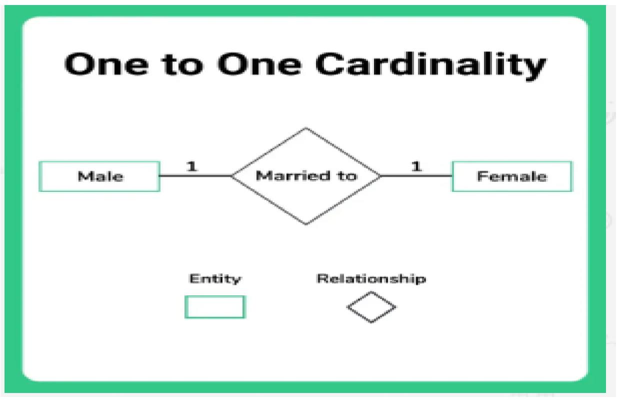
- Simple binary relationship with two entity sets then 4 possible mapping cardinalities to exist as follows
    - **One to one (1:1)**
    - **One to many (1:M)**
    - **Many to one (M:1)**
    - **Many to Many (M M)**

One to one cardinality

- When a single instance of an entity is associated with a single instance of another entity, then it is called as one to one cardinality
- Here each entity of the entity set participate only once in the relationship

Example for one to one cardinality

Assume that the ***only male can be married to only one female and one female can be married to only one male,*** this can be viewed as one to one cardinality

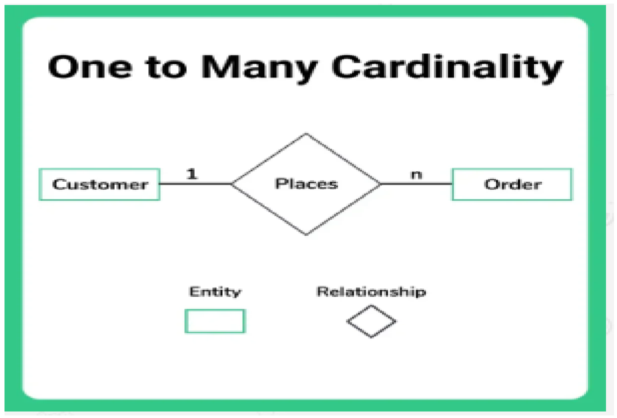


One-to-Many cardinality

- When is a single instance of an entity is associated with more than one instance of another entity then this type of relationship is called one to many relationships
- Here entities in one entity set can take participation in any number of times in relationships set and entities in another entity set can take participation only once in a relationship set

Example for one to many cardinalities

For example in the Real world, ***many students can study in a single college but the student cannot apply to more than one college at the same time***

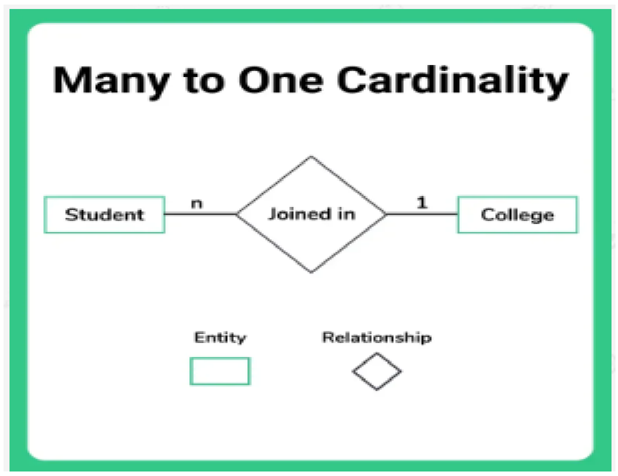


Many-to-one cardinality

When entities in one entity set can participate only once in a relationship set and entities in another entity can participate more than once in the relationship set, then such type of cardinality is called many-to-one

Example for many to one cardinality

- In real time a student takes only one course but a single course can be taken by any number of students. so many to one relationship is observed here
- It means that ***one course can be taken by any number of students but only one course can be allotted to one student***

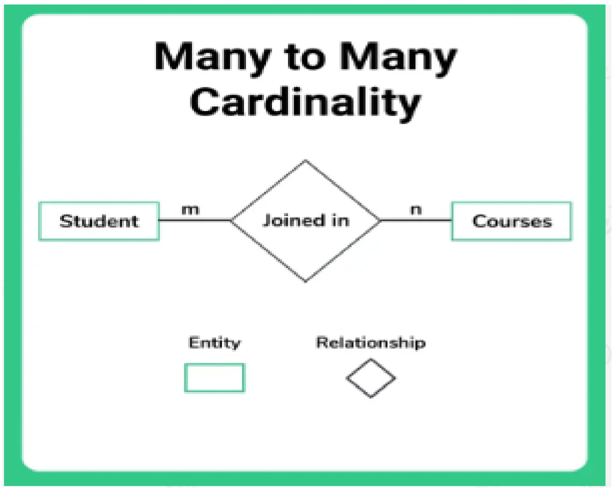


Many-to-many cardinality

- Here more than one instance of an entity is associated with more than one instance of another entity then it is called many to many relationships
- In this cardinality, entities in all entity sets can **take participate any number of times in the relationship** cardinality is many to many.

Example for many to many cardinality

In the Real world assume that a ***student can take more than one course and the single course can be taken by any number of students*** this relationship will be many to many relationship



ER to Relational Model Conversion

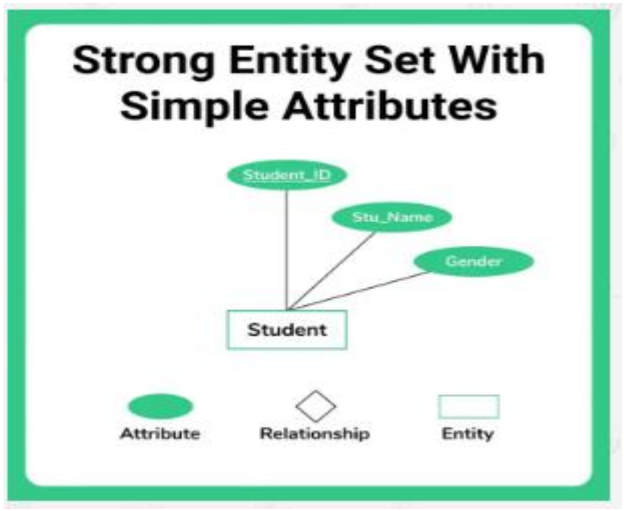
- Conversion is nothing but converting ER diagram to tabular form.
- This is done because ***tables can be easily implemented by RDBMS like MySQL, Oracle*** etc.

Certain rules are to be followed to convert ER diagram to table.They are

Rule-01: For Strong Entity Set With Only Simple Attributes

In relational model ***only one table is required*** to represent a strong entity set with simple attributes. In this

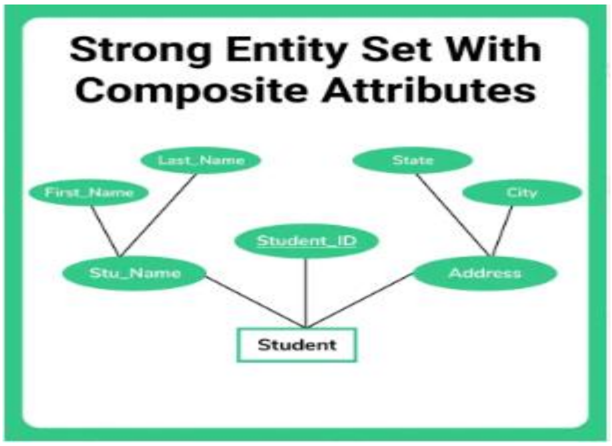
- ***Attributes are taken as columns*** of the table
- ***Key attribute*** is declared as ***primary key of the table***



Rule-02: For Strong Entity Set With Composite Attributes

In relational model ***only one table is required*** to represent a strong entity set with composite attributes. In this

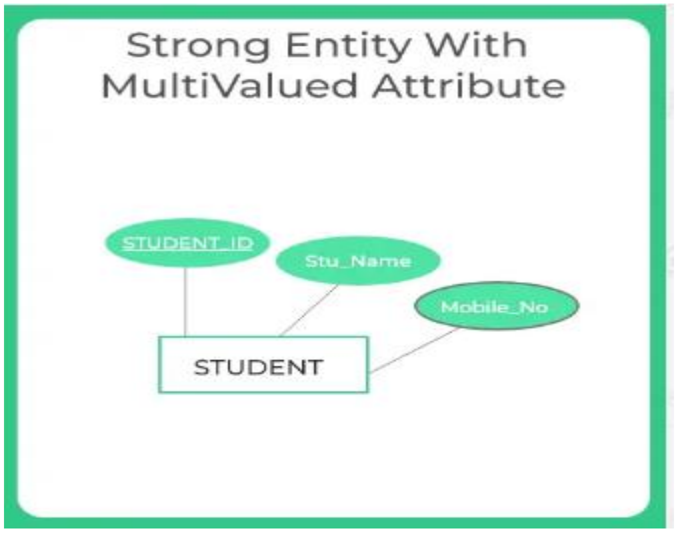
- ***Simple attributes are taken as columns*** of the table and
- ***Simple attributes of the composite attributes are considered as columns*** but not composite attributes themselves
- ***Key attribute*** is declared as ***primary key*** of the table



Rule-03: For Strong Entity Set With Multi Valued Attributes

In relational model ***two tables are required*** to represent a strong entity set with multi-valued attributes.

- ***One table*** with columns as ***primary key and multi-valued attributes*** and
- ***One table*** with columns as ***primary key and other simple attributes***



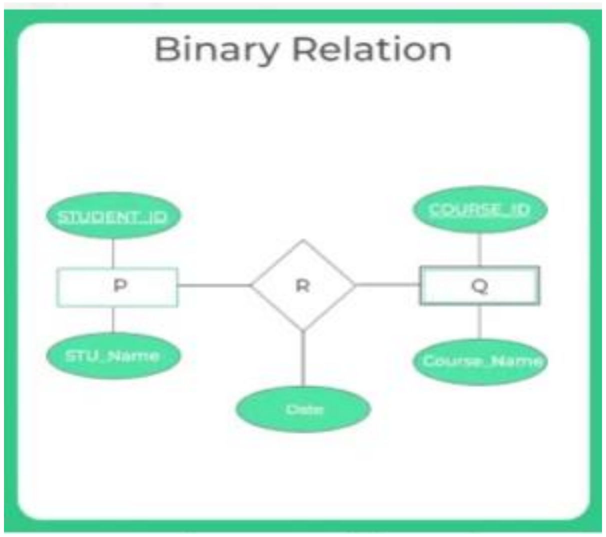
Rule-04: Translating Relationship Set into a Table

Only ***one table is required*** to represent a relationship set with columns as

- ***Key attributes of each participating entity set as primary keys***
- Attributes of the relationship if any.

For suppose ,if we consider a relationship set with 2 entity sets , we require totally 3 tables to represent the whole ER diagram

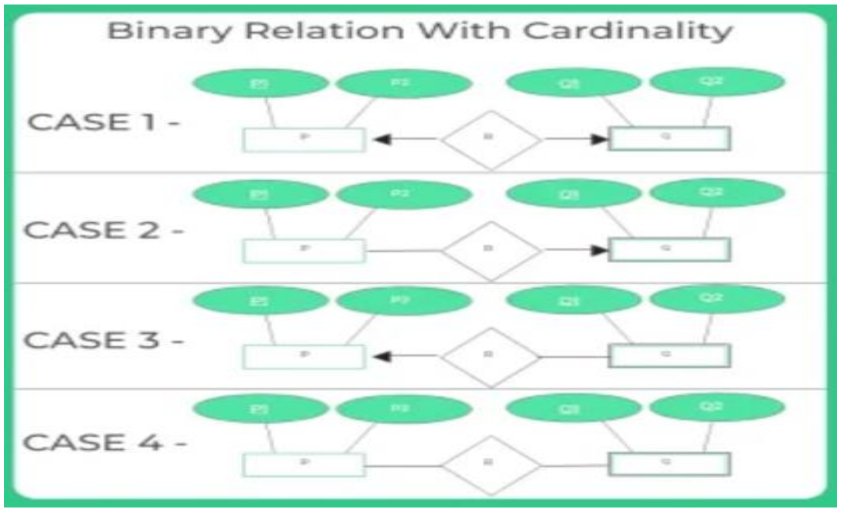
- ***One table for first entity set*** with columns as its attributes
- ***Other table for second entity set*** with columns as its attributes
- ***Another table for relationship set with columns as primary keys of both entity sets and attributes of relationship set***



Rule-05: For Binary Relationships With Cardinality Ratios

In this model four cases are possible. They are :

- **Case-1:** Binary relationship with cardinality ratio 1:1
- **Case-2:** Binary relationship with cardinality ratio m:1
- **Case-3:** Binary relationship with cardinality ratio 1:n
- **Case-4:** Binary relationship with cardinality ratio m:n



Case-1: For Binary Relationship With Cardinality Ratio 1:1

In this case ***two tables are required*** , we can combine the relationship set with either of the entity sets.

- First possible way
    1. PR (p1 , p2 , q1)
    2. Q (q1 , q2)
- Second possible way
    1. P(p1 , p2)
    2. QR (p1 , q1 , q2)

Case-2: For Binary Relationship With Cardinality Ratio m:1

In this model ***two tables are required*** to represent the ER diagram. They are

- PR (p1 , p2 , q1)
- Q (q1 , q2)

Case-3: For Binary Relationship With Cardinality Ratio 1:n

In this model ***two tables are required*** to represent the ER diagram. They are

- P(p1 , p2)
- QR (p1, q1 , q2)

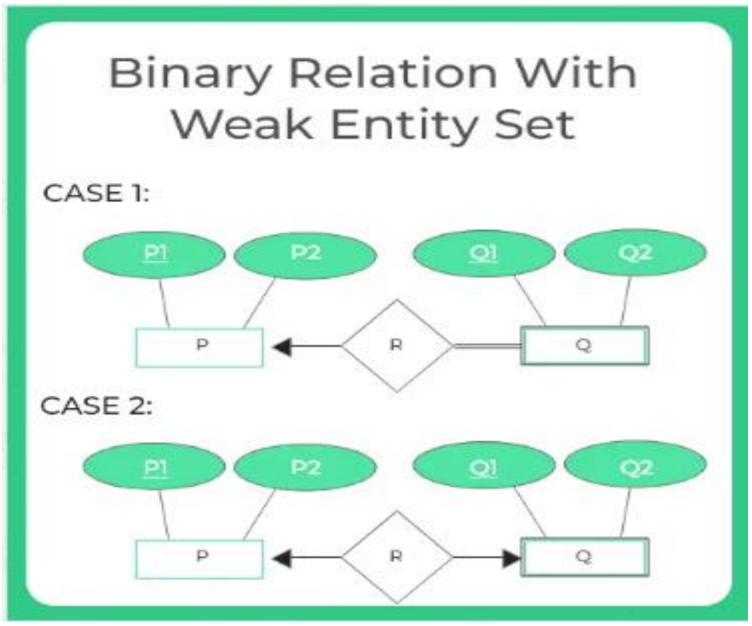
Case-4: For Binary Relationship With Cardinality Ratio m:n

In this model ***three tables are required*** to represent the ER diagram. They are

- P (p1 , p2)
- R (p1 , q1)
- Q (q1 , q2)

Rule-06: For Binary Relationship With Both Cardinality Constraints and Participation Constraints

- In this model ***foreign key acquires NOT NULL costraint because of total participation constraint***
- Rule 5 is followed to implement cardinality constraints.
- In this two cases are possible :



<u>Case-01: For Binary Relationship With Cardinality Constraint and Total Participation Constraint From One Side</u>

- Here we will ***combine the entity set Q and relationship set R as cardinality ratio is 1:n.***
- Then, two tables will be required-
    1. P(p1 , p2)
    2. QR (p1 , q1 , q2)
- Foreign key ***p1 has acquired NOT NULL constraint because of total participation constraint***

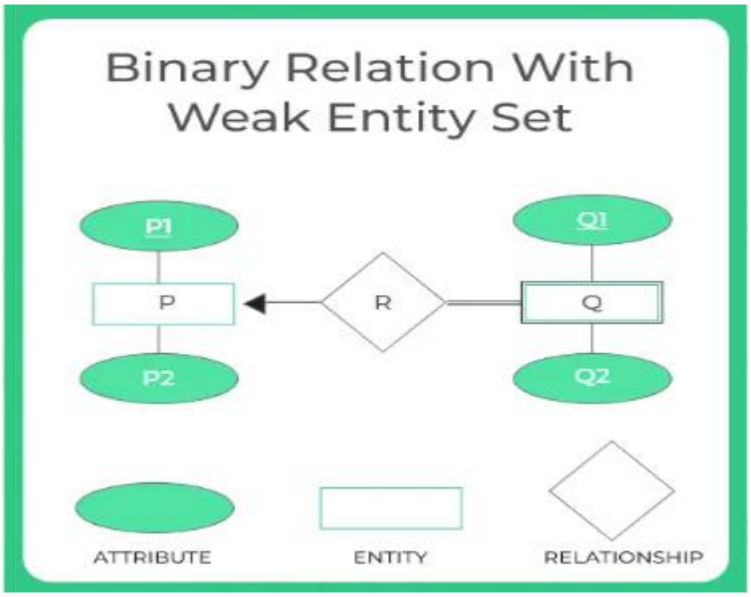
Case-02: For Binary Relationship With Cardinality Constraint and Total Participation Constraint From Both Sides

Only ***one table is required to represent a binary relationship when there is a key constraint from both the sides of entity set with total participation.*** It is

- PRQ(p1 , p2, q1 , q2)

Rule-07: For Binary Relationship With Weak Entity Set

- Whenever a ***weak entity set appears***, it will have ***identifying relationship set with total participation constraint.***
- This model ***requires two tables*** to represent the ER diagram. They are
  1. P (p1 , p2)
  2. QR (p1 , q1 , q2)



Relational Calculus

Relational Calculus in DBMS

What is Relational Calculus?

Using a non procedural query language, unlike the procedural one used in relational algebra. In other words it only provides the information about description of the query but not detailed methods on how to do it.

-
  - Relational Calculus focusses upon mathematical predicate calculus
  - Relational Algebra focusses on mathematic algebra

Forms of relational Calculus –

1.
   1. Tuple Relational Calculus (TRC)
   2. Domain Relational Calculus (DRC)

Tuple Relational Calculus (TRC)

1. In tuple calculus we find tuples which are true for a given condition.
2. The predicate must be true for a tuple
3. Result obtained maybe more than 1 tuple.

Use – For relational calculus.

Notation – {t| P(t)}

Breakdown –

1.
   1. t represents tuples returned as results
   2. P for Predicate i.e. conditions for results

Other relevant notations –

-
  - ∈ – represents Belongs to
  - ∃ – called an existential quantifier represents there is at least one
  - r – means relation
  - (∨) – OR
  - (∧) – AND

- ○ (¬) – NOT

Example 1 :

{t | Employee (e) and e.SALARY > 100000}

- Now, this represents results which will be returned as tuple t.
- Predicate here is – Employee (e) and e.SALARY > 100000
- Will return tuples for all the employees which have salary greater than 100000.

We can also write this as –

{t | t ∈ Employee (e) and e.SALARY > 100000}

This just represents that the tuple t belongs to relation Employee and we're using this to be on the safe side.

Example 2 :

{t| ∃ s ∈ Salary(t.emp_ID = s.emp_ID ∧ s.Salary >= 100000)}

It will result in the emp_Id for each employee that has his/her salary greater or equal to 10000.

Domain Relational Calculus (DRC)

While in tuple relationship calculus we did relational mathematics based on the tuple results and predicates. In domain relational calculus, however, we do it based on the domains of the attributes.

Use – For relational calculus.

Notation – { $c_1, c_2, ..., c_n$ | F($c_1, c_2, ... ,c_n$)}

Breakdown –

1. $c_1, c_2, ..., c_n$ represents domain of attributes(columns)
2. F for Predicate i.e. condition for results

Example :

{< Fname, Emp_ID > | ∈ Employee ∧ Salary > 10000}

The result here will be returning the Fname and Emp_ID values for all the rows in the employee table where salary is greater than 10000.

TUPLE RELATIONAL CALCULUS IN DBMS

- Relational algebra specifies procedures and methods to fetch data hence is called as a procedural query language ,whereas relational calculus is ***a non procedural query language focuses on just fetching data rather than how the query will work and how data will be fetched***
- Simply relational calculus is nothing but focusing on what to do rather than focusing on how to do

Relational calculus is present in two formats

- Tuples relational calculus(TRC)
- Domain relational calculus(DRC)

Tuple Relational Calculus (TRC) in DBMS

Relational calculus peoples are filtered based on a condition

Syntax:

```
{ T | Condition }
```

Relation part

- Here t represents the tuple variable which is nothing but representing that it is a table
- It can be any variable but for understanding we use variable ***t which stands for the table as per our context***
- For example, if ***our table is Student***, we would put it as ***Student(T)***

Condition part

Condition is *specified using this dot variable* the common and column we need to operate

Example 1

```
T.age > 22
```

- where ***T is our tuple variable and age is a column that is used for filtering*** the records in the relation
- Now combine both relational and conditional part and see how the ***final quotation will look like T.name | Student(T) AND T.age > 21***
- It is a relational calculus which results in names of students from the table ***student for having age greater than 17***

Example 2

```
{t | Employee (e) and e.SALARY > 100000}
```

- Now, this represents results which will be returned as tuple t.
- Predicate here is – Employee (e) and e.SALARY > 100000 will ***return tuples for all the employees who have a salary greater than 100000.***

Example 3

```
{ t | Employee (e) and Salary(t.emp_ID = s.emp_ID ∧ s.Salary >= 100000)}
```

- It will ***result in the employeeId for each employee that has his/her salary greater or equal to 10000.***

Domain Relational Calculus in DBMS

- Relational algebra specifies procedures and methods to fetch data hence is called as a procedural query language ,whereas relational calculus is ***a non procedural query language focuses on just fetching data rather than how the query will work and how***

data will be fetched.

- Simply relational calculus is nothing but focusing on what to do rather than focusing on how to do.

Relational calculus is present in two formats

- Tuples relational calculus(TRC)
- Domain relational calculus(DRC)

DRC in DBMS

- In Domain relational calculus filtering of records is done based on the domain of the attributes rather than tuple values
- A domain is nothing but the set of allowed values in the column of a table

Syntax:

```
{ c1, c2, c3, ..., cn | F(c1, c2, c3, ... ,cn)}
```

where, ***c1, c2… etc represents the domain of attributes***(columns) and ***F represents the formula*** including the condition for fetching the data.

Example 1

```
{< name, age > | ∈ Student ∧ age <21}
```

Again, the above query will return the ***names and ages of the students in the table Student who not greater than 21 years old***

Example 2

```
{< Fname, Emp_ID > | ∈ Employee ∧ Salary > 10000}
```

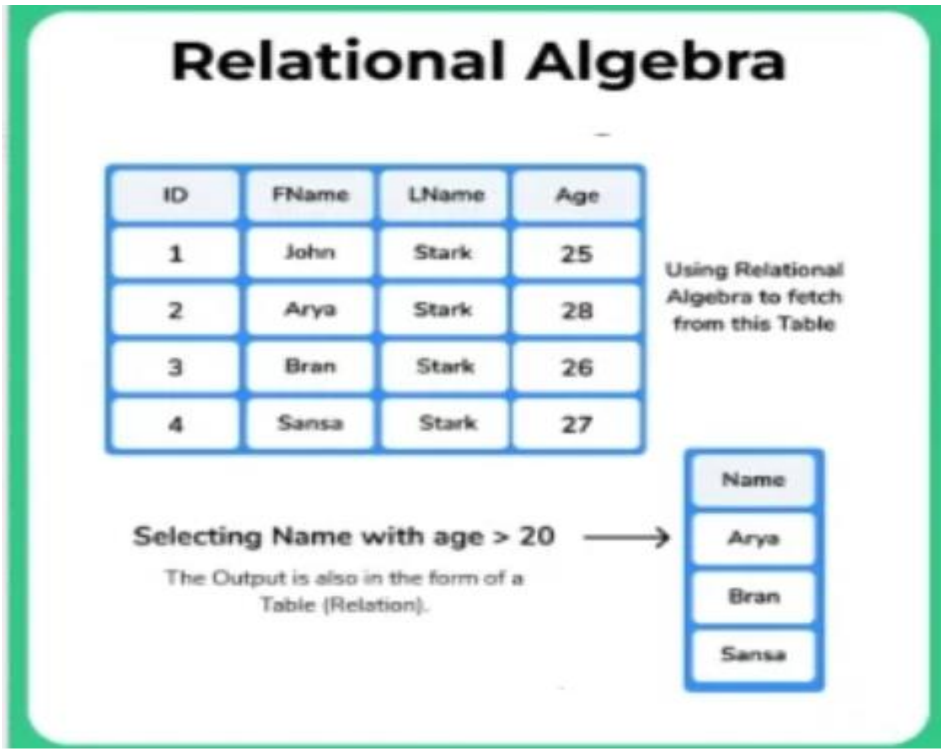
The result here will be returning the ***Fname and Emp_ID values for all the rows in the employee table where salary is greater than 10000.***

Relational Algebra in DBMS
Relational Algebra in DBMS is a query language which is procedural in nature, both of its input and output are relations. The theoretical foundations of relational databases and SQL is provided by Relational Algebra.

Relational Algebra in DBMS

Following operations can be applied via relational algebra –

1. Select
2. Project
3. Union
4. Set Different
5. Cartesian product
6. Rename


Relational Algebra
ID
FName
LName
Age
1
John
Stark
25
2
Arya
Stark
28
3
Bran
Stark
26
4
Sansa
Stark
27
Using Relational Algebra to fetch from this Table
Selecting Name with age > 20
The Output is also in the form of a Table (Relation).
Name
Arya
Bran
Sansa

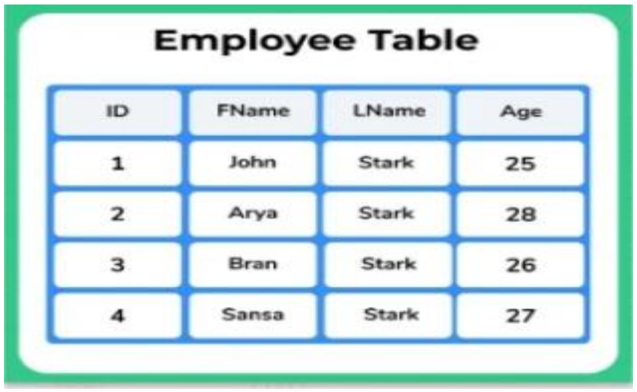
Select Operation (σ)
<u>Use –</u> Fetching rows (tuples) from a table, which satisfied a given condition.

Notation – $\sigma_p(r)$

Breakdown –

1. σ represents select predicate
2. rfor relation
3. pfor proposition logics like – − =, ≠, ≥, < , >, ≤. with us of connectors like OR, AND, or NOT

Like we used in the earlier example in the image above, of selecting rows for people who had age > 25 and give results in Fname format.

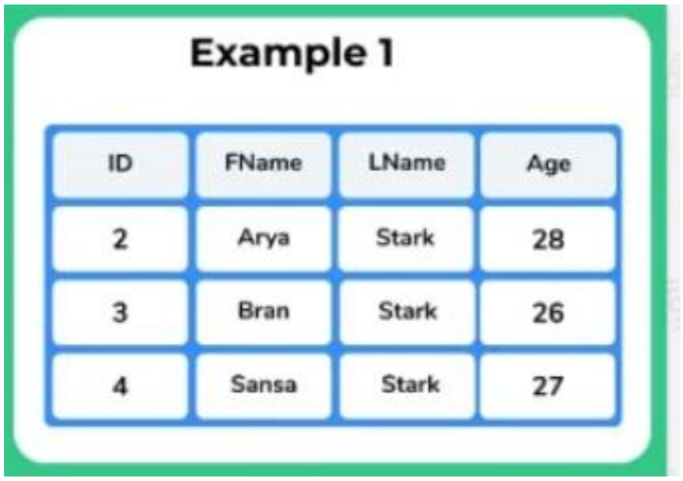


Employee Table

ID	FName	LName	Age
1	John	Stark	25
2	Arya	Stark	28
3	Bran	Stark	26
4	Sansa	Stark	27

Example 1

- Query – σ_{age} > 25 (Student)
    - σ(select predicate)
    - r(relation) – Employee
    - p(proposition logic) – age >25
    - Result – returning the list of students with age greater than 25.
    - In SQL – select * from Student where age > 25



Example 1

ID	FName	LName	Age
2	Arya	Stark	28
3	Bran	Stark	26
4	Sansa	Stark	27

Project Operation (∏)
Use – Fetching on specific columns from a table

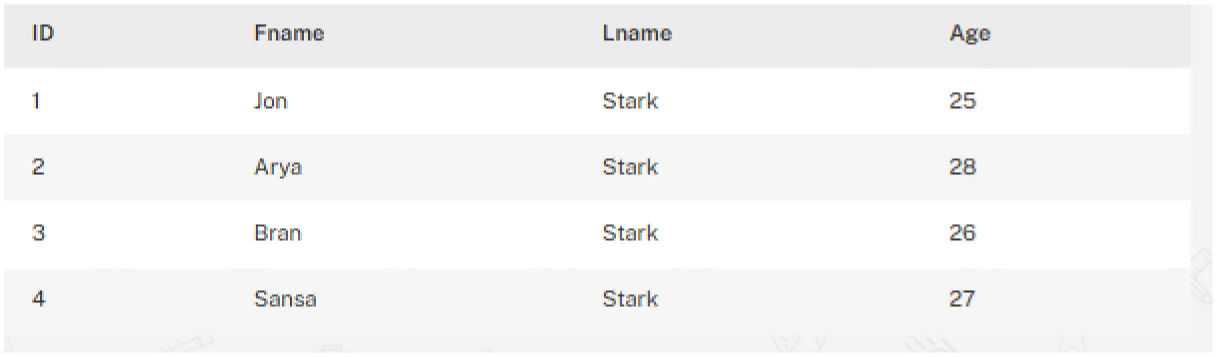
Notation – $\prod_{A1,A2,An}(r)$

Breakdown –

1. ∏ represents Project predicate
2. rfor relation
3. A1, A2, A3for selection from columns for projection

Example 1

- Query$\sigma_{Lname = 'Stark'}$ (Student)
    - σ(select predicate)
    - r(relation) – Employee
    - p(proposition logic) – Lname = ‘Stark’
    - Result – returning the list of students Lname = ‘Stark’
    - In SQL – select * from Student where Lname = ‘Stark’



ID	Fname	Lname	Age
1	Jon	Stark	25
2	Arya	Stark	28
3	Bran	Stark	26
4	Sansa	Stark	27

Union Operation (∪)
Union Operation performs as expected, it essentially finds the union of the tables included in the union i.e finds only the unique rows/tuples from multiple tables, removing the duplications.

Use – Fetching union rows (tuples), i.e unique rows (tuples) from multiple tables removing the duplications

Notation – A(∪)B

Breakdown –

1. (∪) represents Union Operation
2. A and B are the tables

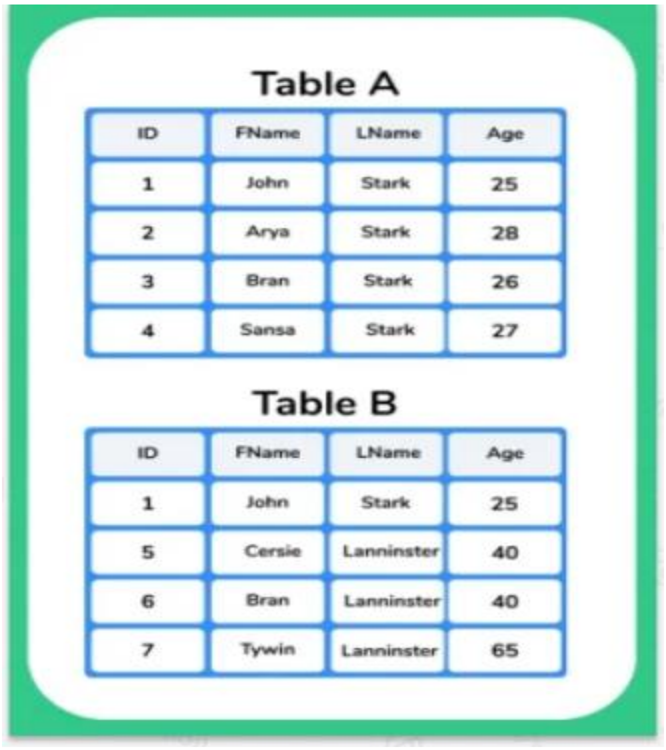


Table A

ID	FName	LName	Age
1	John	Stark	25
2	Arya	Stark	28
3	Bran	Stark	26
4	Sansa	Stark	27

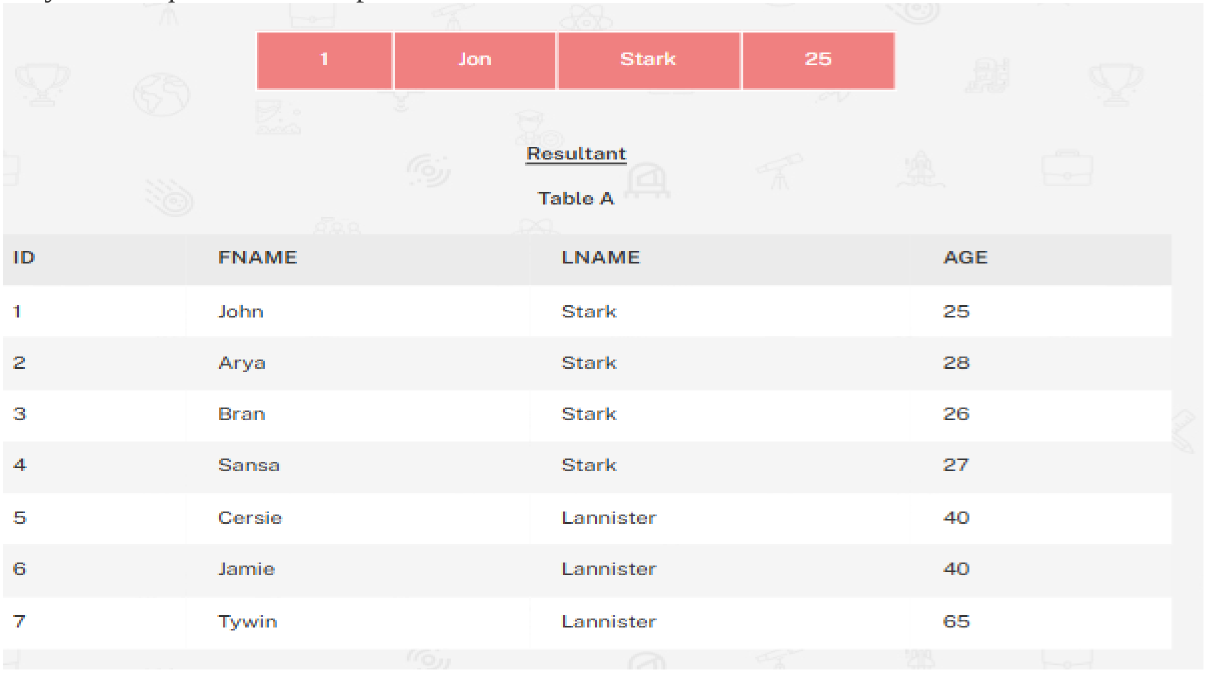
Table B

ID	FName	LName	Age
1	John	Stark	25
5	Cersie	Lanninster	40
6	Bran	Lanninster	40
7	Tywin	Lanninster	65

Example 1

- QueryA(∪)B
  - (∪)(Union Operation)
  - A and BTable
  - In SQL – SELECT * FROM A UNION SELECT * FROM B;

The following table is removed as it is present in both table A and table B and union operation only has unique and non duplicates.



1	Jon	Stark	25

Resultant

Table A

ID	FNAME	LNAME	AGE
1	John	Stark	25
2	Arya	Stark	28
3	Bran	Stark	26
4	Sansa	Stark	27
5	Cersie	Lannister	40
6	Jamie	Lannister	40
7	Tywin	Lannister	65

Intersection Operation (∩)
Intersection operation works simply by helping to find the rows (tuples) that are common i.e. exists in both (all) the tables involved in the intersection operation.

Use – Fetching union rows (tuples), i.e common rows (tuples) from multiple tables and only the rows that exist in both (all) tables involved in the operation.

Notation – A(∩)B

Breakdown –

1. (∩) represents Intersection Operation

2. A and B are the tables

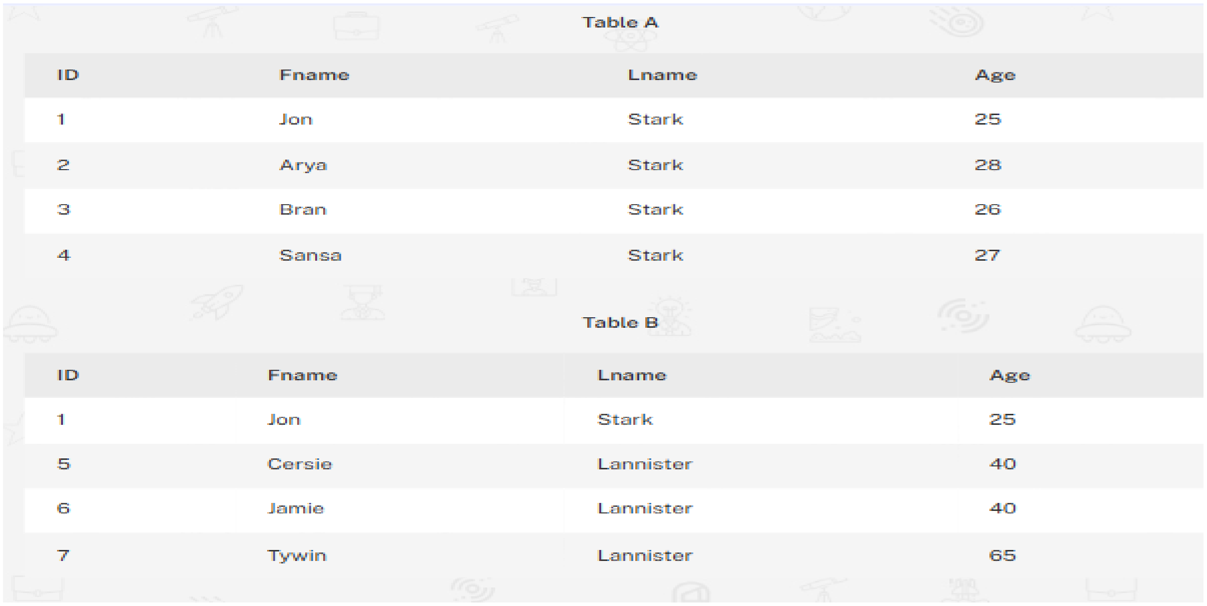


Table A

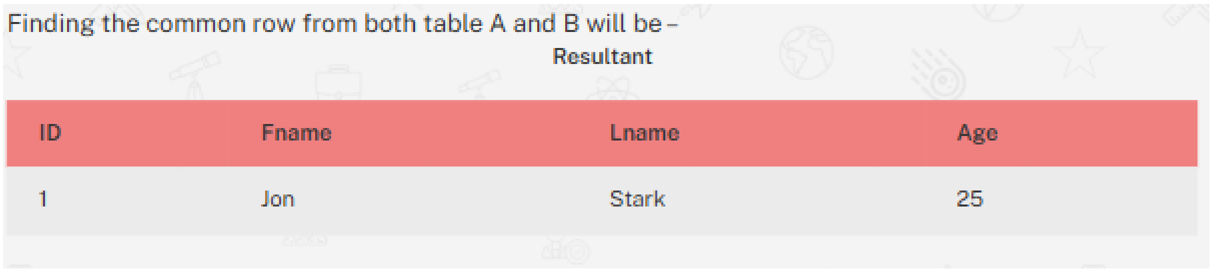
ID	Fname	Lname	Age
1	Jon	Stark	25
2	Arya	Stark	28
3	Bran	Stark	26
4	Sansa	Stark	27

Table B

ID	Fname	Lname	Age
1	Jon	Stark	25
5	Cersie	Lannister	40
6	Jamie	Lannister	40
7	Tywin	Lannister	65

Example 1

- QueryA(∩)B
    -
        - A and BTables
        - (∩)(Intersection Operation)
        - In SQL – SELECT * FROM A INTERSECT SELECT * FROM B;



Finding the common row from both table A and B will be –

Resultant

ID	Fname	Lname	Age
1	Jon	Stark	25

$\prod_{\text{player}}$ (Cricket) ∪ $\prod_{\text{player}}$ (Football)

The following gives the result for selecting the people who either play only cricket or only play football or play both of them.

Set Difference (−)

Use – Fetching rows from which are present in relation but not the other one. Example players who play cricket but don't play football.

Notation – A - B

Breakdown –

1. - represents Set Difference Operation
2. A and B are the tables (relations)

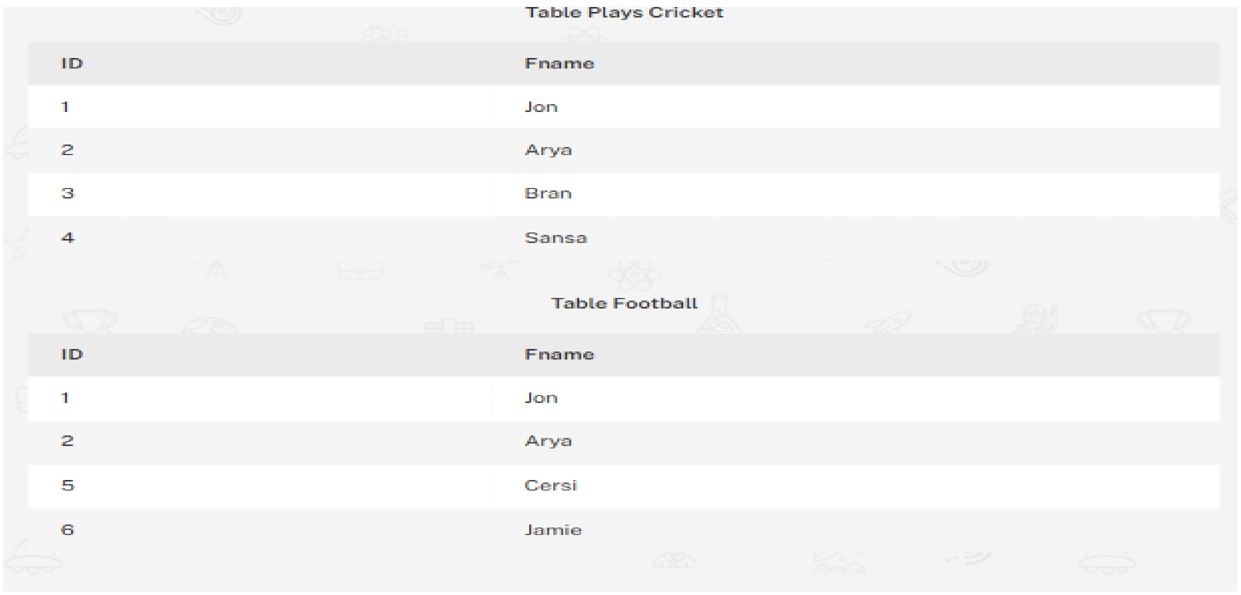


Table Plays Cricket

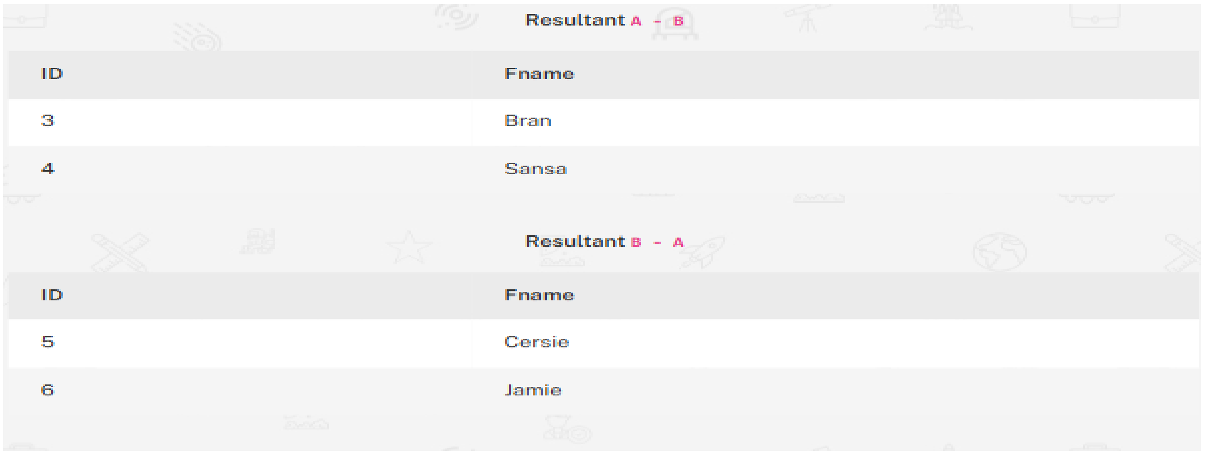
ID	Fname
1	Jon
2	Arya
3	Bran
4	Sansa

Table Football

ID	Fname
1	Jon
2	Arya
5	Cersi
6	Jamie

Example 1

- QueryA - B
    -
    - A and BTables A: Plays Cricket B: Plays Football
    - -(Intersection Operation)
    - In SQL – SELECT * FROM A INTERSECT SELECT * FROM B;


Resultant A – B

ID	Fname
3	Bran
4	Sansa

Resultant B – A

ID	Fname
5	Cersie
6	Jamie

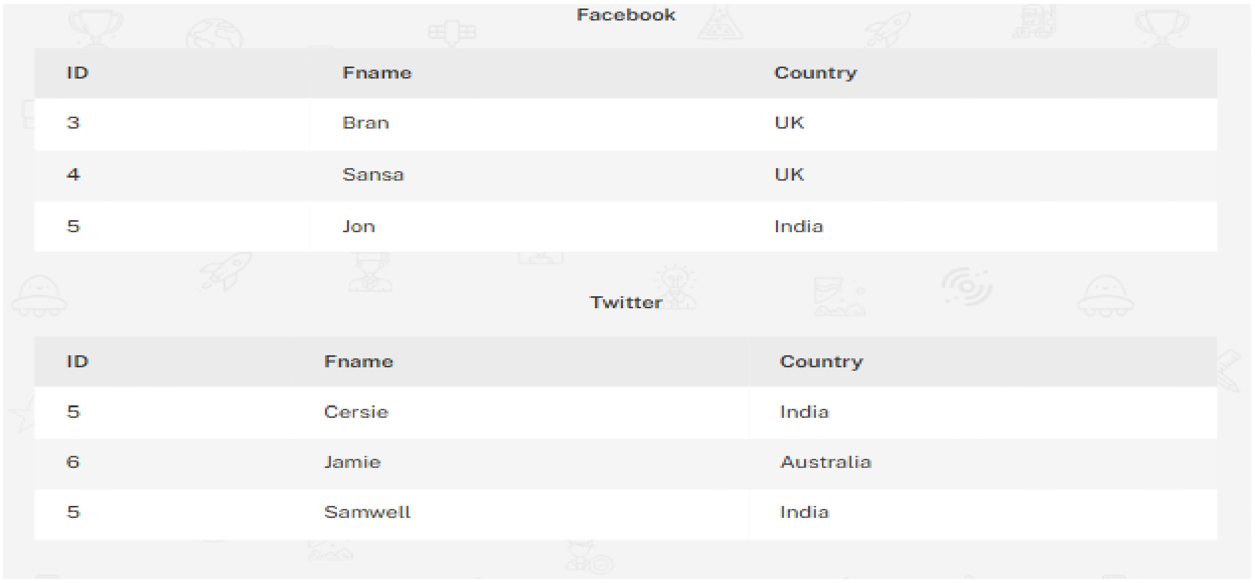
Cartesian Product (X)

Use – Merging columns from two different relations, i.e. combining them together. It is mostly not a suitable and meaningful data representation if we just calculate the cartesian product alone. To get meaningful data we must follow the same by subsequent operations. We will understand more about this in detail.

Notation – A X B

Breakdown –

1. X represents Cartesian Product
2. A and B are the tables (relations)


Facebook

ID	Fname	Country
3	Bran	UK
4	Sansa	UK
5	Jon	India

Twitter

ID	Fname	Country
5	Cersie	India
6	Jamie	Australia
5	Samwell	India

Imaging that we have list of all Facebook and Twitter users and we want to find out the unique list of people from both of them who live in India.

$\sigma_{Fname = 'Country'}$(Facebook X Twitter)

Result

Fname
Jon
Samwell

Rename Operation (ρ)

Use – When we find the result of a some relational operation they are just displayed !. However, we may want to store them in a new relation with a name that we can further use. For the same we use rename operation.

Don't get confused with the name, the name might give idea that it is done to rename an existing table. However, this is used to store results of relation in a new named table.

Notation – ρ_x (E)

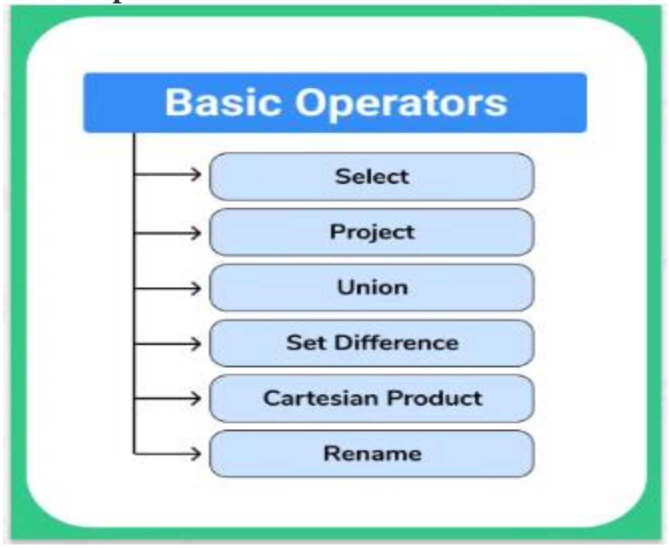
Breakdown –

1. ρ i.e. rho represents rename operation.
2. x is the new name of the resultant table,
3. E is the expression used for result.

Use – ρ_{users} (E =A (∩) B)

The results of expression will be stored in new table names as users.

Basic Operators in DBMS



Relational Algebra in DBMS

- Relational algebra is a widely used procedural query language.
- Relational algebra collects instances of relations as input and gives occurrences of relations as output by using various operations.
- Relational algebra operations are performed recursively on a relation.
- The output of relational algebra operations is a new relation, which can be formed from one or more input relations.

Basic Operators in Relational Algebra:
There are 6 basic operations in Relational Algebra. They are

- Select (σ)
- Project (∏)
- Union (∪)
- Set Difference (-)
- Cartesian product (X)
- Rename (ρ)

Select (σ)

- The SELECT operation is ***used for selecting a subset of the tuples according to a given selection condition.***
- This operation is ***denoted by symbol sigma(σ).***

<u>Syntax:</u>

$\sigma_p(r)$

- σ denotes ***Select operation***
- p denotes the ***condition***
- r is the ***relation/table name***

<u>Example 1:</u>

$\sigma_{\text{course = "Java"}}$ (Student)

Output– Selects tuples from Student where course = ‘Java’.

<u>Example 2</u>

$\sigma_{\text{course = "Java" and age = "20"}}$(Student)

Output – Selects tuples from Student where the course is ‘Java’ and age is ’20’.

<u>Example 3</u>

$\sigma_{\text{salary > 50000}}$ (Employee)

Output – Selects tuples from Employee where salary is greater than 50000

Projection(π)

- The projection method ***defines a relation that contains a vertical subset of Relation.***
- It is ***represented by symbol pi(π)***

Example:

Consider the following table:

Stu_ID	Stu_Name	City
1	Priya	Hyderabad
2	Anjali	Mumbai
3	Rahul	Delhi
4	Rishi	Chennai

Here, the projection of Stu_Name and City will give

$\Pi_{\text{Stu\_Name,City}}$ (Students)

Stu_Name	City
Priya	Hyderabad
Anjali	Mumbai
Rahul	Delhi
Rishi	Chennai

-

Note:

For the next 4 operations

Consider table ‘A’ as:

Attribute1	Attribute2
1	1
1	2

Consider table ‘B’ as:

Attribute1	Attribute2
1	1
1	3

Union operation (υ)

- The operation A ∪ B ***includes all tuples that are in tables A or in B***.
- Even ***if a tuple is present in both table A and table B*** the ***result will contain that tuple only once***.

For a union operation to be valid, the following conditions must hold –

- A and B must have the ***same number of attributes.***
- Attribute ***domains need to be compatible***.
- ***Duplicate tuples*** should be automatically ***removed***.

A ∪ B gives

Attribute1	Attribute2
1	1
1	2
1	3

Set Difference (-)

- It is ***represented by symbol ‘-‘ .***
- Operation A – B, is a relation which includes all tuples ***that are in A but not in B***.

For a set difference operation to be valid, the following conditions must hold –

- The ***attribute name of A has to match with the attribute name in B.***
- The two-operand relations ***A and B should be either compatible or Union compatible.***

Example
A-B

Attribute1	Attribute2
1	2

Cartesian product(X)

- Cartesian product is ***helpful to merge columns from two relations.***
- This operation ***is meaningful only when it is followed by other operations***.

Example – Cartesian product

$\sigma_{\text{Attribute 2} = \text{'1'}}$ (A X B)

Output – The above example shows all rows from relation A and B whose Attribute 2 has value 1

Attribute1	Attribute2
1	1
1	1

Rename (ρ)

- Rename operation can be used ***to rename a relation or an attribute of a relation.***
- It is ***represented by symbol 'ρ'.***

Syntax:

```
ρ(new_relation_name, old_relation_name)
```

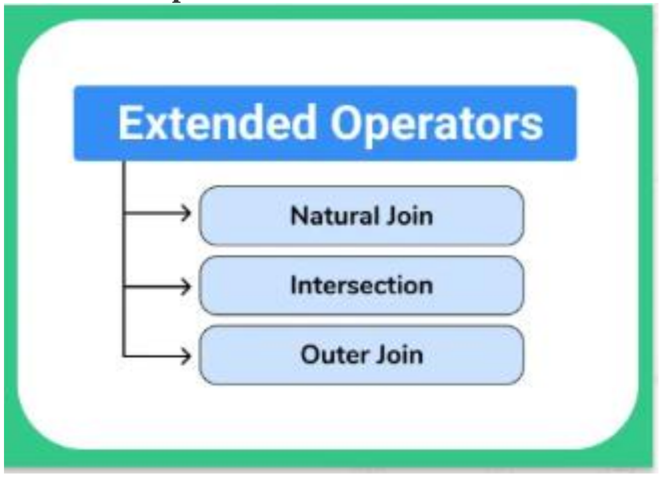
Example:

```
ρ(Att_2, ∏(Attribute2)(B))
```

Output:

Attribute2
1
3

Extended Operators in DBMS



Relational Algebra in DBMS

- Relational algebra is a widely used procedural query language.
- Relational algebra collects instances of relations as input and gives occurrences of relations as output by using various operations.
- Relational algebra operations are performed recursively on a relation.
- The output of relational algebra operations is a new relation, which can be formed from one or more input relations.

Extended Operators in Relational Algebra:

Extended operators in Relational algebra are:

- Natural Join (⋈)
- Left, Right, Full outer join (⟕, ⟖, ⟗)
- Intersection (∩)

Learn more about Relational Algebra here on this page.

Natural Join (⋈)

- JOIN operation also allows ***joining variously related tuples from different relations.***
- Natural join can only be performed ***if there is a common attribute (column) between the relations.***
- The ***name and type of the attribute must be same.***

Example:
Consider the following two tables:

Table C

Num	Square
2	4
3	9

Table D

Num	Cube
2	8
3	27

C ⋈ D

Num	Square	Cube
2	4	8
3	9	27

Outer Join

- In an outer join, ***along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.***
- There are ***three types of outer joins***. They are

- Left Outer Join
- Right Outer Join
- Full Outer Join

- Consider the following 2 Tables :

Table A

Num	Square
2	4
3	9
4	16

Table B

Num	Cube
2	8
3	27
5	125

Left Outer Join(A ⟕ B)

- In the left outer join, ***the operation allows keeping all tuple in the left relation.***
- If there is ***no matching tuple is found in the right relation***, then the attributes of the right relation in the join ***results are filled with null values***.

Example:

A ⟕ B

Num	Square	Cube
2	4	8
3	9	27
4	16	–

<u>Right Outer Join(A ⟖ B)</u>

- In the right outer join, ***the operation allows keeping all tuple in the right relation.***
- If there is ***no matching tuple is found in the left relation***, then the attributes of the left relation in the join ***results are filled with null values***.

<u>Example:</u>

A ⟖ B

Num	Square	Cube
2	4	4
3	9	9
5	–	125

<u>Full Outer Join: (A ⟗ B)</u>

In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.

A ⟗ B

Num	Square	Cube
2	4	8
3	9	27
4	16	–
5	–	125

Intersection (∩)

- An intersection is ***defined by the symbol ∩***
- A ∩ B defines a relation consisting of ***a set of all tuple that is in both A and B.*** However, A and B must be union-compatible.

<u>Example:</u>

Consider 2 tables –

Table A :

Attribute1	Attribute2
1	1
1	2

Table B :

Attribute1	Attribute2
1	1
1	3

A ∩ B

Attribute1	Attribute2
1	1