UNIT –II

DBMS DATABASE MODELS

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

- Hierarchical Model
- Network Model
- Entity-relationship Model
- Relational Model

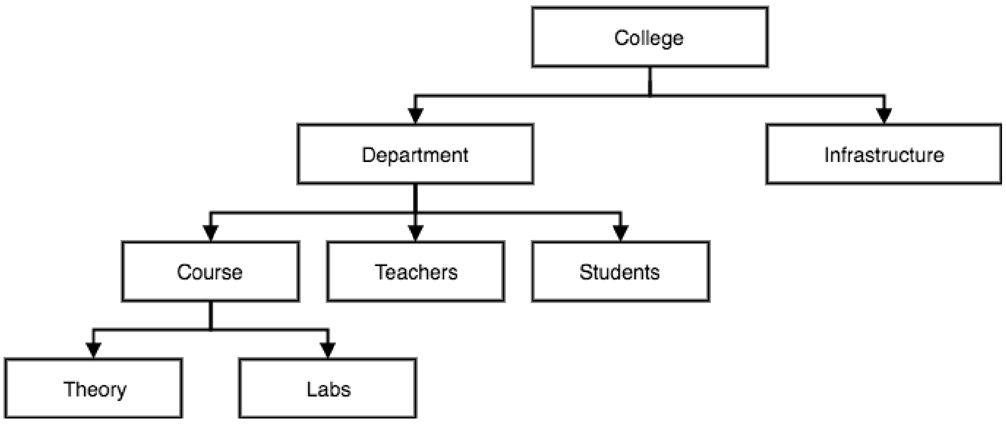
Hierarchical Model

This database model organises data into a tree-like-structure, with a single root, to which all the other data is linked. The heirarchy starts from the **Root** data, and expands like a tree, adding child nodes to the parent nodes.

In this model, a child node will only have a single parent node.

This model efficiently describes many real-world relationships like index of a book, recipes etc.

In hierarchical model, data is organised into tree-like structure with one one-to-many relationship between two different types of data, for example, one department can have many courses, many professors and of-course many students.

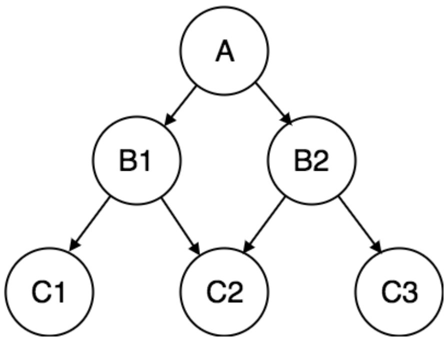


Network Model

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

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

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



Entity-relationship Model

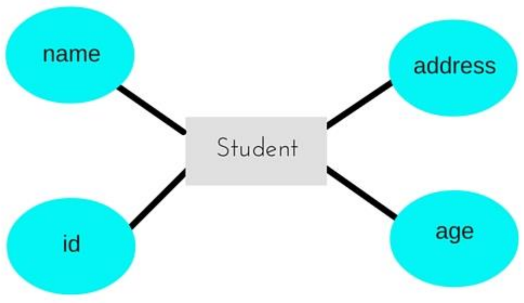
In this database model, relationships are created by dividing object of interest into entity and its characteristics into attributes.

Different entities are related using relationships.

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

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

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



Relational Model

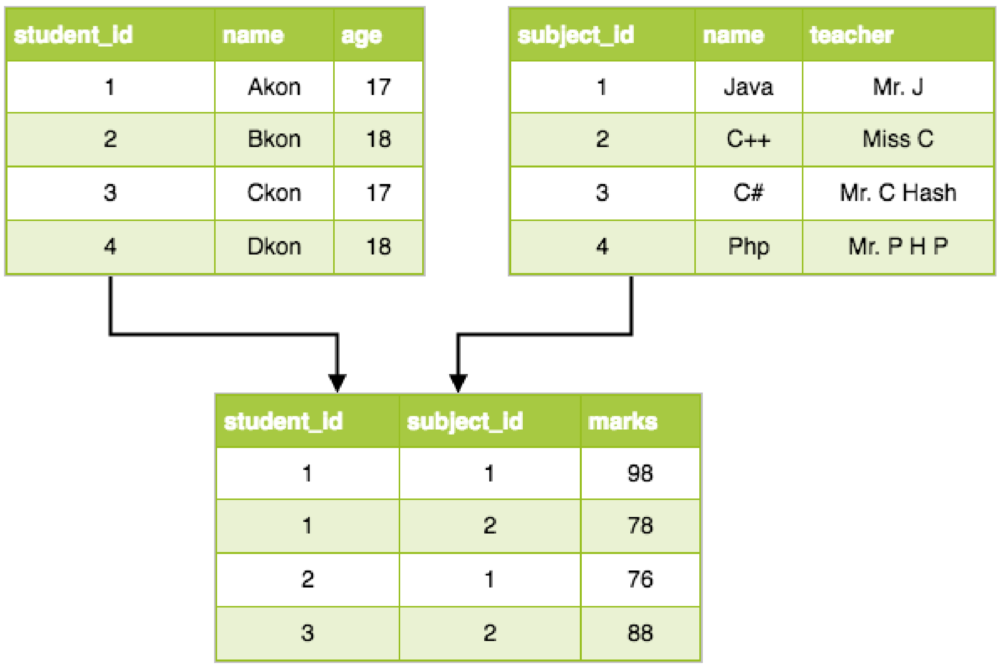
In this model, data is organised in two-dimensional **tables** and the relationship is maintained by storing a common field.

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

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

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

In the coming tutorials we will learn how to design tables, normalize them to reduce data redundancy and how to use Structured Query language to access data from tables.



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

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

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

ER (ENTITY REATIONSHIP) DATAMODEL - BASIC CONCEPTS

The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases.

Entity

An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

An **entity set** is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

Attributes

Entities are represented by means of their properties, called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes.

There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Types of Attributes

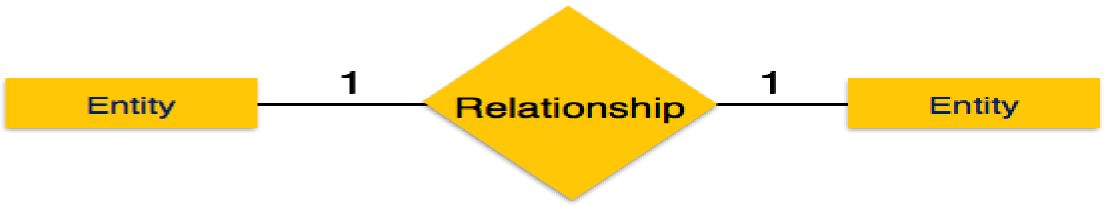
- **Simple attribute** − Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.
- **Composite attribute** − Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.
- **Derived attribute** − Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.
- **Single-value attribute** − Single-value attributes contain single value. For example − Social_Security_Number.
- **Multi-value attribute** − Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

Relationship

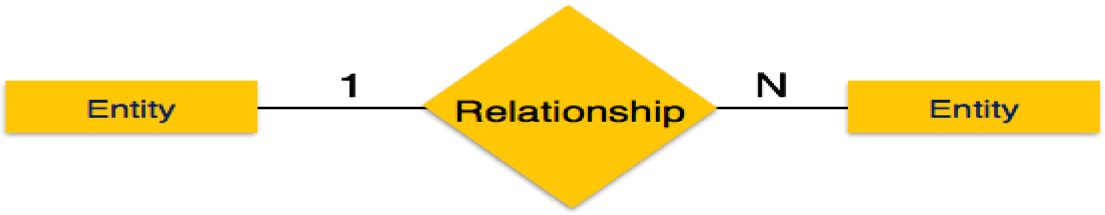
The association among entities is called a relationship. For example, an employee **works_at** a department, a student **enrolls** in a course. Here, Works_at and Enrolls are called relationships.

A relationship where two entities are participating is called a **binary relationship**. Cardinality is the number of instance of an entity from a relation that can be associated with the relation.

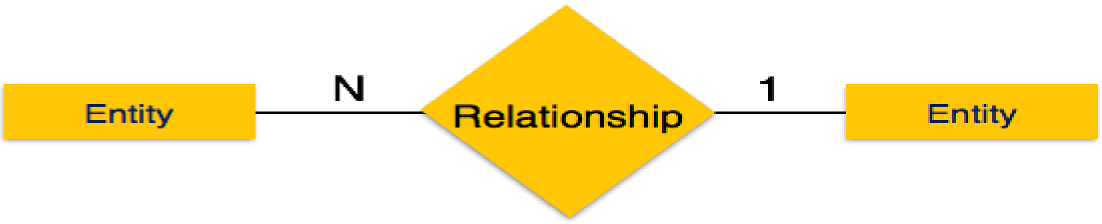
- **One-to-one** − When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.



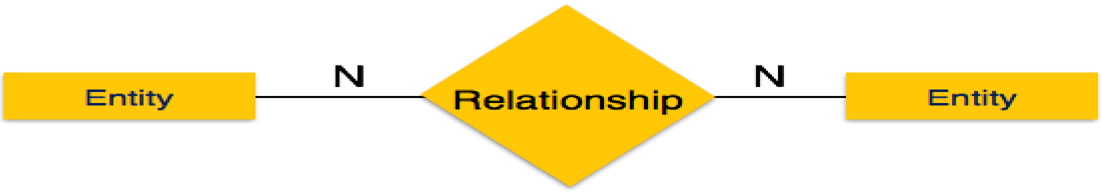
- **One-to-many** − When more than one instance of an entity is associated with a relationship, it is marked as '1:N'. The following image reflects that only one instance of entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts one-to-many relationship.



- **Many-to-one** – When more than one instance of entity is associated with the relationship, it is marked as 'N:1'. The following image reflects that more than one instance of an entity on the left and only one instance of an entity on the right can be associated with the relationship. It depicts many-to-one relationship.



- **Many-to-many** – The following image reflects that more than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts many-to-many relationship.



Relationship Set

A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called **descriptive attributes**.

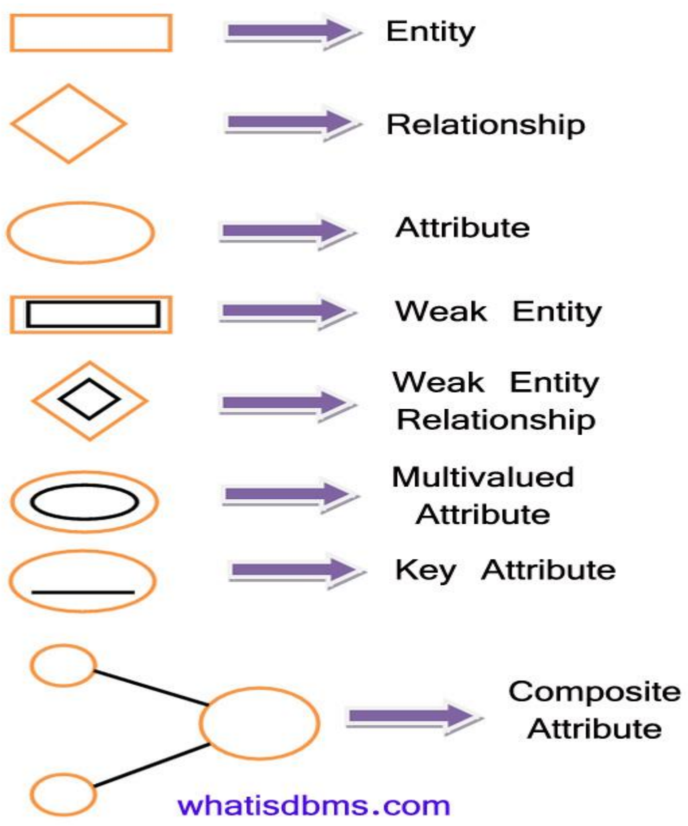
Degree of Relationship

The number of participating entities in a relationship defines the degree of the relationship.

- Binary = degree 2

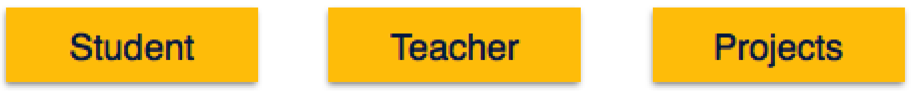
E-R DIAGRAM -REPRESENTATION

Let us now learn how the ER Model is represented by means of an ER diagram. Any object, for example, entities, attributes of an entity, relationship sets, and attributes of relationship sets, can be represented with the help of an ER diagram.

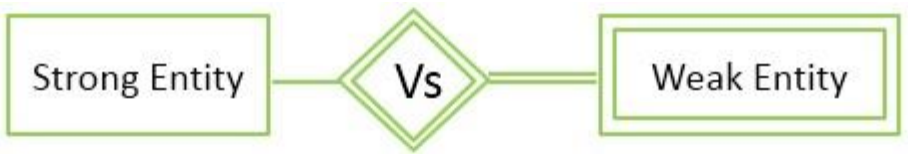


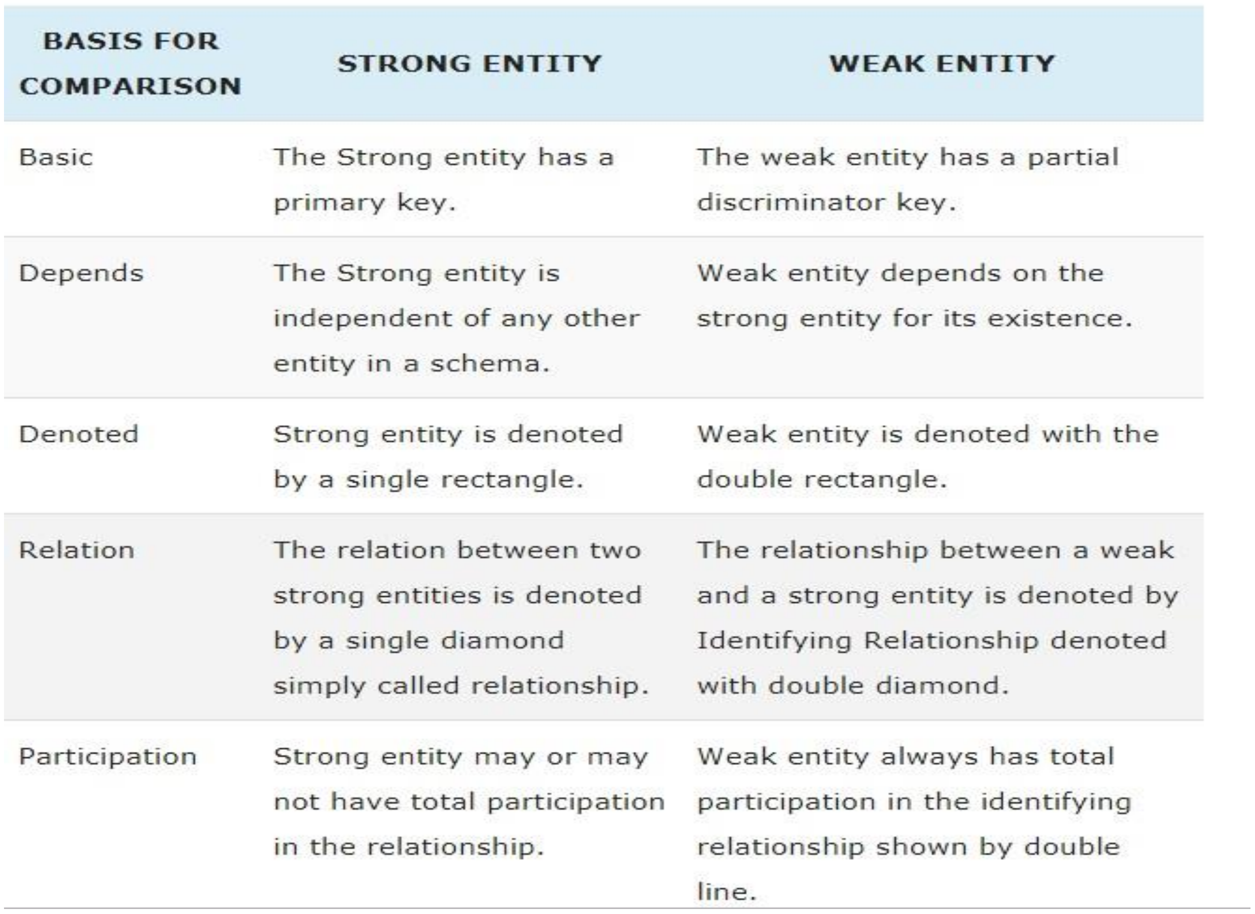
Entity

Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.



Difference between Strong and Weak Entity

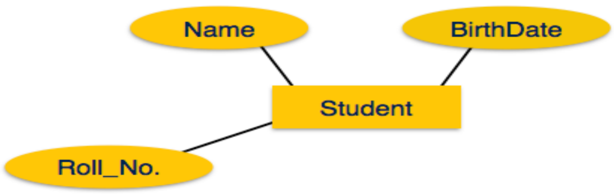




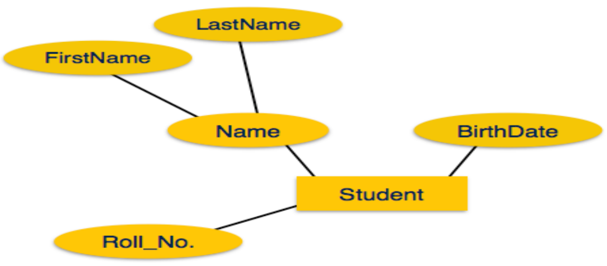
BASIS FOR COMPARISON	STRONG ENTITY	WEAK ENTITY
Basic	The Strong entity has a primary key.	The weak entity has a partial discriminator key.
Depends	The Strong entity is independent of any other entity in a schema.	Weak entity depends on the strong entity for its existence.
Denoted	Strong entity is denoted by a single rectangle.	Weak entity is denoted with the double rectangle.
Relation	The relation between two strong entities is denoted by a single diamond simply called relationship.	The relationship between a weak and a strong entity is denoted by Identifying Relationship denoted with double diamond.
Participation	Strong entity may or may not have total participation in the relationship.	Weak entity always has total participation in the identifying relationship shown by double line.

Attributes

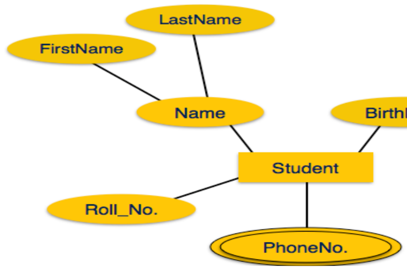
Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).



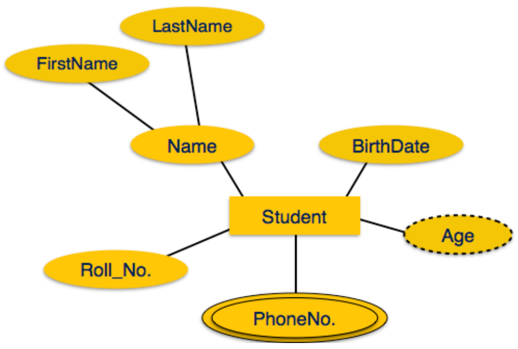
If the attributes are **composite**, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.



Multivalued attributes are depicted by double ellipse.



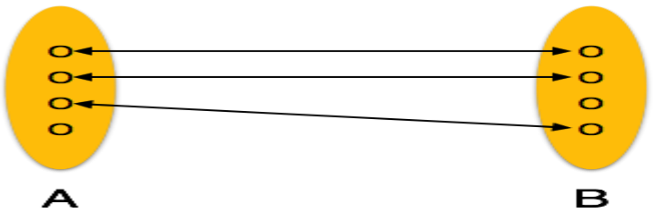
Derived attributes are depicted by dashed ellipse.



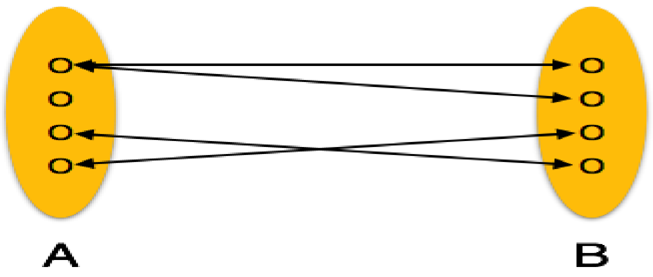
Mapping Cardinalities

Cardinality defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set.

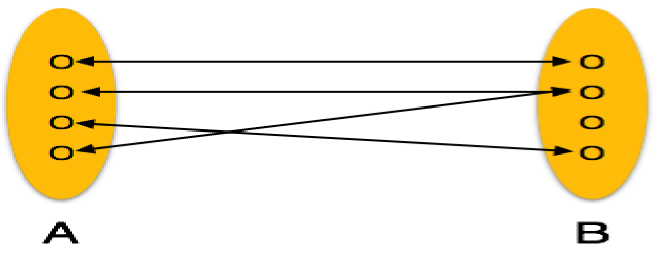
- **One-to-one** − One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



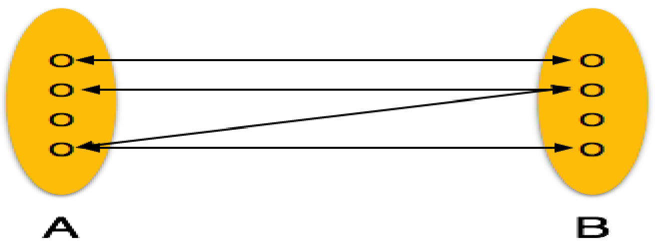
- **One-to-many** − One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.



- **Many-to-one** − More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.



- **Many-to-many** – One entity from A can be associated with more than one entity from B and vice versa.



CONSTRAINTS IN DBMS

✓ Constraints that limit the possible combination of entities that participate in corresponding relationship set

✓ Constraints enforce limits to the data or type of data that can be inserted/updated/deleted from a table. The whole purpose of constraints is to maintain the **data integrity** during an update/delete/insert into a table. In this, we will learn several types of constraints that can be created in DBMS.

✓ **Types of constraints**

- NOT NULL
- UNIQUE
- DEFAULT
- CHECK
- Key Constraints (keys)
- Domain constraints
- Mapping constraints

NOT NULL:

NOT NULL constraint makes sure that a column does not hold NULL value. When we don’t provide value for a particular column while inserting a record into a table, it takes NULL value by default. By specifying NULL constraint, we can be sure that a particular column(s) cannot have NULL values.

UNIQUE:

UNIQUE Constraint enforces a column or set of columns to have unique values. If a column has a unique constraint, it means that particular column cannot have duplicate values in a table.

DEFAULT:

The DEFAULT constraint provides a default value to a column when there is no value provided while inserting a record into a table.

CHECK:

This constraint is used for specifying range of values for a particular column of a table. When this constraint is being set on a column, it ensures that the specified column must have the value falling in the specified range.

DOMAIN CONSTRAINTS:

Each table has certain set of columns and each column allows a same type of data, based on its data type. The column does not accept values of any other data type.
Domain constraints are **user defined data type**

MAPPING CARDINALITY (Mapping constraints):

One to One: An entity of entity-set A can be associated with at most one entity of entity-set B and an entity in entity-set B can be associated with at most one entity of entity-set A.

One to Many: An entity of entity-set A can be associated with any number of entities of entity-set B and an entity in entity-set B can be associated with at most one entity of entity-set A.

Many to One: An entity of entity-set A can be associated with at most one entity of entity-set B and an entity in entity-set B can be associated with any number of entities of entity-set A.

Many to Many: An entity of entity-set A can be associated with any number of entities of entity-set B and an entity in entity-set B can be associated with any number of entities of entity-set A

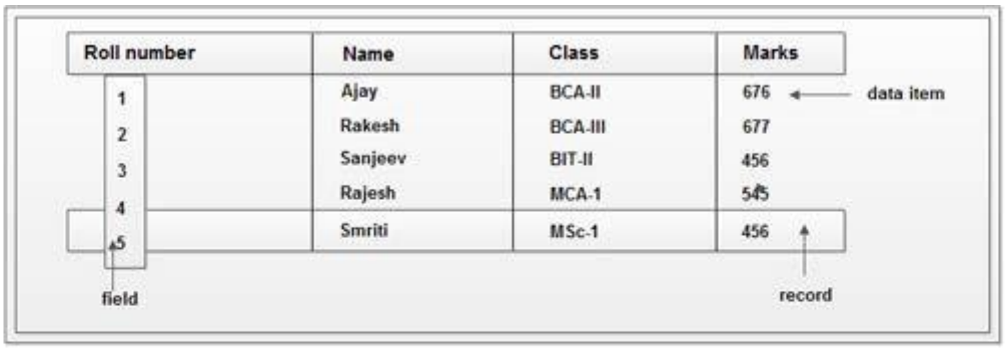
KEY CONSTRAINTS (keys)

What is Key?

A Key is a data item that exclusively identifies a record. In other words, key is a set of column(s) that is used to uniquely identify the record in a table. It is used to fetch or retrieve records / data-rows from data table according to the condition/requirement. Key provide several types of constraints like column can't store duplicate values or null values. Keys are also used to generate relationship among different database tables or views.

A database consists of tables, which consist of records, which further consist of fields.

This below figure provides an example of a typical table consisting of STUDENT details:



Roll number	Name	Class	Marks
1	Ajay	BCA-II	676
2	Rakesh	BCA-III	677
3	Sanjeev	BIT-II	456
4	Rajesh	MCA-1	545
5	Smriti	MSc-1	456

data item
field
record

Here, table containing five records and each containing four fields.

Table name = **Student**

A row =**a record**

A column = **a field**

A cell = a **data item**; a particular value in a field for a particular record.

A table in a database may be empty, containing no records or contain many millions of them. Similarly, a single record may consist of one or thousands of fields. Each field in turn contains a single data item. In order to uniquely identify each record of the table, there must be some field or combination of fields, which should have only unique values.

For example: In above table more than one student may have the same Name, Class and Marks but they must have the different Roll numbers. So, we can distinguish one record from the other with the help of Roll_number column. Here, the column Roll_number that is used to uniquely identify each record of the table is called as Key Field

Key *is a set of one or more columns whose combined values are unique among all occurrences in a given table. A key is the relational means of specifying uniqueness. In patient database, a patient number could be used as a key field to uniquely identify each* *patient's record in the doctors' patient file. The patient's name could not be a key field as there may be more than one patient with the same name. There must be at least one key field in each table. Sometimes a record may contain more than one key field. For example, the doctor's patient's file may contain both a patient number and a National Insurance number for each patient. Both of these are key fields.*

Types of Keys

Every key which has the property of uniqueness can be distinguished as following:

- Candidate Key
- Super Key
- Primary Key
- Foreign Key
- Composite key
- Artificial keys
- Alternate keys

Candidate key: Candidate keys are those attributes of a relation, which have the properties of· uniqueness and irreducibility, Candidate keys are defined as the minimal set of fields which can uniquely identify each record in a table. A super key with no redundant attribute is known as candidate key It is an attribute or a set of attributes that can act as a Primary Key for a table to uniquely identify each record in that table.whose explanation is given below:

Let R be a relation. By definition, the set of all attributes of R has the uniqueness property, meaning that, at any given time, no two tuples in the value of R at that time are duplicates of one another. As in the case of the STUDENT relation, for example, the subset containing just attribute Roll number has that property. These facts constitute the intuition behind the definition of candidate key.

Let K be a set of attributes of relation R. Then K is a candidate key for R .if and only if it possesses both of the following properties:

a) Uniqueness: No legal value of R ever contains two distinct tuples with the same value for K.

b) Irreducibility: No proper subset of K has the uniqueness property.

Note that every relation does have at least one candidate key. The uniqueness property of such keys is self-explanatory. Irreducibility property means if a candidate key is a composite key (consists of more than one attribute) then no individual attribute of candidate key, which participate into it, is unique. For example, if the combination of (Name, Class) is unique, then it can be identified as the candidate key if and only if Name and Class individually are not unique. Since a null value is not guaranteed to be unique, no component of a candidate key is allowed to be null. There can be any number of candidate keys in a table.

Super Key: A super key has the uniqueness property but not necessarily the irreducibility property. A super key is a set of one of more columns (attributes) to uniquely identify rows in a table. A candidate key is a special case of a super key.

For example, if Roll_number is unique in relation STUDENT then, the set of attributes

(Roll_number, Name, Class) is a super key for a relation STUDENT, these set of attributes are also unique, but this combination of keys (composite key) is not having the property of irreducibility because Roll_number which is one subset of the composite key is also unique itself. Thus, this composite key is called as super key because it has the property of uniqueness but not the irreducibility.

Consider a relation of Patient in which Patient_number is unique. Then, Patient_number is a candidate key and (Patient number, Patient name) is a super key. Thus, we can say that "A superset of a candidate key is a super key."

Primary Key: The *primary key* is an attribute or a set of attributes that uniquely identify a specific instance of an entity. Every entity in the data model must have a primary key whose values uniquely identify instances of the entity. Sometimes a record may contain more than one key field.

For example, the doctor's patient's file may contain both a patient number and a National Insurance number for each patient. Both of these are key fields. We therefore choose one of them and call it the primary key field. Primary key cannot contain any Null value because we cannot uniquely identify multiple Null values.

Properties of Primary Key

To qualify as a primary key for an entity, an attribute must have the following properties:

• **Stable:** The value of a primary key must not change or should not become null Through out the life of an entity. A stable primary key helps to keep the model stable.

For example: if we consider a patient record, the value for the primary key (Patientnumber) must not change with time as would happen with the age field.

• **Minimal:** The primary key should be composed of the minimum number of fields that ensures the occurrences are unique.

• **Definitive:** A value must exist for every record at creation time. Because an entity occurrence cannot be substantiated unless the primary key value also exists.

• **Accessible:** Anyone who wants to create, read or delete a record must be able to see the Primary key value.

Once candidate keys are identified, choose one aw' only one, primary key for each.

Candidate keys, which are not chosen as the primary key, are known as Alternate Keys.

An example of an entity that could have several possible primary keys is Employee. Let's assume that for each employee in an organization there are three candidate keys: Employee ID, Social Security Number, and Name. Name is the least desirable candidate. 'While it might work for a small department where it would be unlikely that two people would have exactly the same name, it would not work for a large organization that had hundreds or thousands of employees. Moreover, there is the possibility that an employee's name could change because of marriage. Employee ID would be a good candidate as long as each employee was assigned a unique identifier at the time of hire. Social Security would work best since every employee is required to have one before being hired.

The primary key of any table is any candidate key of that table which the database designer arbitrarily designates as "primary". The primary key may be selected for convenience, comprehension, performance, or any other reasons.

Alternate Key

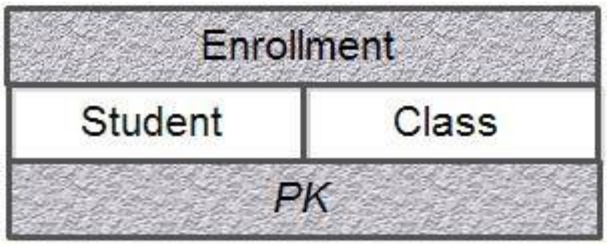
The alternate keys of any table are simply those candidate keys, which are not currently selected as the primary key.

Exactly one of those candidate keys is chosen as the primary key and the remainders, if any, are then called alternate keys. An alternate key is a *function* of all candidate keys *minus* the primary key.

The primary key is unique and often programmers assign a primary key just to obtain uniqueness. If this is the only requirement of the primary key, then there is a danger that at some point someone else may eventually change the primary key (for example, to a system assigned number) and lose the original enforcement of uniqueness on what had been the primary key earlier.

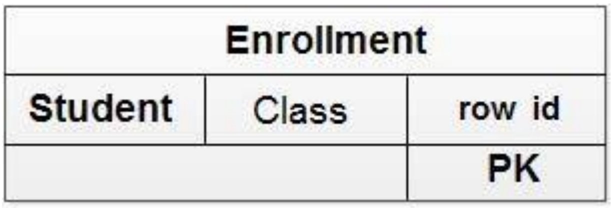
Let look a table below which holds student class enrollment:

This table has a compound primary key



Enrollment	
Student	Class
PK	

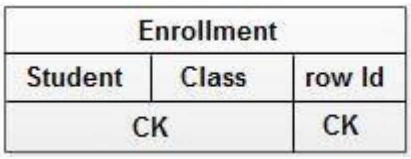
Here, the designer knows that each student-class pair will be unique; i.e., each student may enroll in each class only once. Several years later a new DBA decides that it is inefficient to use two columns for the primary key where one would do. He/She adds a "row id" column and makes it the primary key by loading it with a system counter.



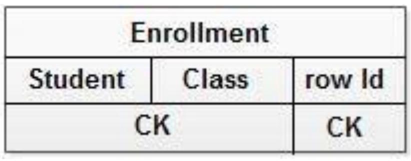
Enrollment		
Student	Class	row id
		PK

This is fine as far as an identity for each row. But now nothing prevents a student from Enrolling in the same class multiple times. This happened because the data model did not retain a candidate key property on the two original columns when the primary key was changed. Therefore the new DBA had no direct way of knowing (other than text notes somewhere) that these two columns must still remain unique, even though they are no longer part of the primary key.

Notice here how the model could have handled this automatically, if it had captured candidate keys in the first place and then generated alternate keys as a function of those candidates not in the primary key. The original two columns remain unique even after they are no longer primary.



Enrollment		
Student	Class	row Id
CK		CK



Enrollment		
Student	Class	row Id
CK		CK

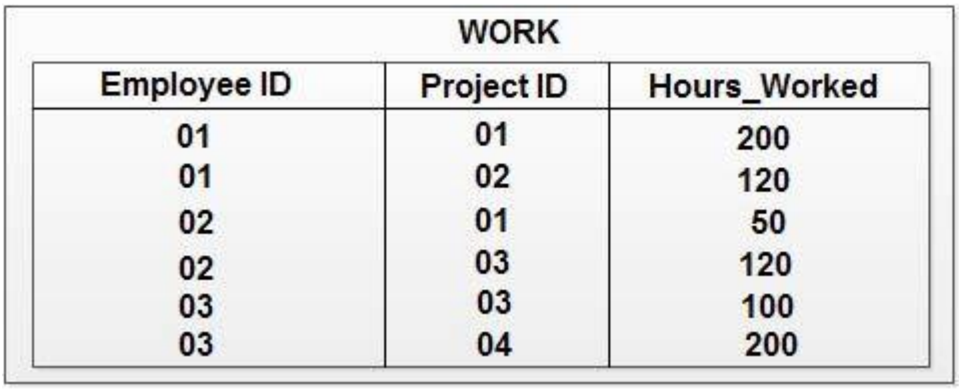
So, do not confuse a candidate key specification of uniqueness with the arbitrary selection of primary key.

Composite Keys

Sometimes it requires more than one attribute to uniquely identify an entity. A primary key that is made up of more than one attribute is known as a *composite key.*

Table below shows an example of a composite key. Each instance of the entity Work can be uniquely identified only by a composite key composed of Employee ID and Project ill,

Example of Composite Key



WORK

Employee ID	Project ID	Hours_Worked
01	01	200
01	02	120
02	01	50
02	03	120
03	03	100
03	04	200

Artificial Keys

An *artificial key* is one that has no meaning to the business or organization.

Artificial keys are permitted when

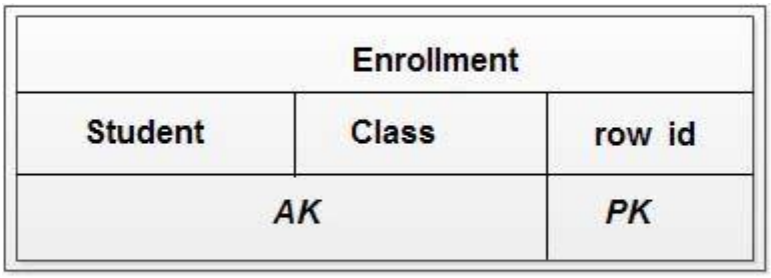
I) no attribute has all the primary key properties, or

2) The primary key is large and complex.

For Example: Enrollment table from business point of view has (student, c1ass) combination as primary key. But this primary key is large and complex, so DBA decides to add row_id column as primary key. Here, row_id can also be called as Artificial Key.

Foreign Keys

Foreign keys are the attributes of a table, which refers to the primary key of some another table. Foreign Keys permit only those values, which appears in the primary key of the table to which it refers or may be null. Foreign keys are used to link together two or more different tables which have some form of relationship with each other. The foreign key is a reference to the tuple of a table from which it was taken, this tuple being called the Referenced or Target tuple. The table containing the referenced tuple will be called as Target table.



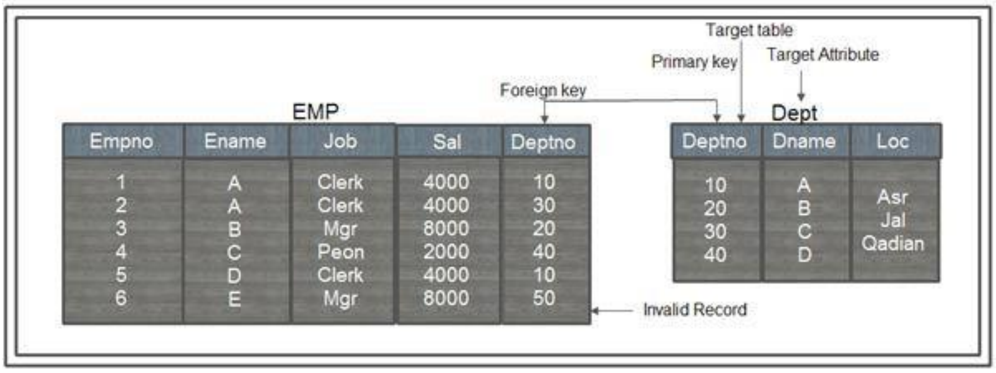
Enrollment		
Student	Class	row id
AK		*PK*

Thus, *a foreign key* is an attribute that completes a relationship by identifying the parent entity. Foreign keys provide a method for maintaining integrity in the data (called referential integrity) and for navigating between different instances of an entity. A foreign key must support every relationship in the model. The matter of integrity of foreign keys is referred to as Referential Integrity.

Foreign keys values should always be matched by corresponding primary key values.

For example:

Consider the below, containing two tables (EMP, DEPT)



EMP

Empno	Ename	Job	Sal	Deptno
1	A	Clerk	4000	10
2	A	Clerk	4000	30
3	B	Mgr	8000	20
4	C	Peon	2000	40
5	D	Clerk	4000	10
6	E	Mgr	8000	50

Dept

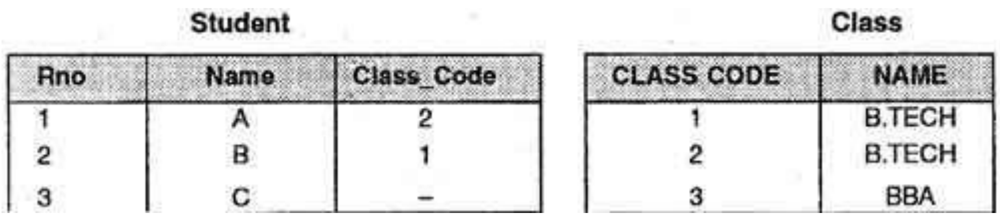
Deptno	Dname	Loc
10	A	Asr
20	B	Jal
30	C	Qadian
40	D	

If we try to insert <u>information</u> of employee with deptno 50, then this is an invalid information, because there is no deptno 50 exists in the company(as shown in table DEPT).Then this invalid information should be prevented from insertion, which would only be possible if deptno of EMP table refer the deptno of DEPT table. It means that only those values are permitted in deptno of EMP table, which appears in the deptno attribute of the DEPT table. Thus, we can say that deptno of EMP table is the foreign key which refers the primary key deptno of DEPT table. Thus, we can insert the empno 6 with any deptno from 10, 20, 30 and 40.

Null may also be permitted in the deptno of EMP table. Here, deptno of DEPT table

IS Target attribute and DEPT is the target table.

Consider another example:



Student

Rno	Name	Class_Code
1	A	2
2	B	1
3	C	–

Class

CLASS CODE	NAME
1	B.TECH
2	B.TECH
3	BBA

Here, college has three valid classes with class code 1, 2 and 3. The class code (foreign key) of student table refers class_code of class table.

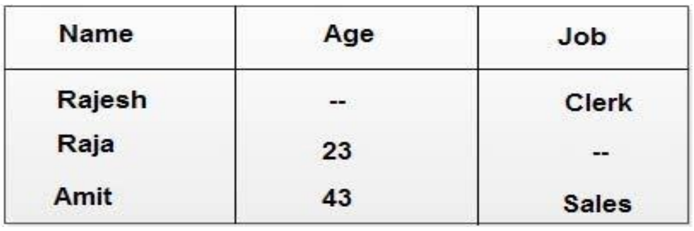
RELATIONAL INTEGRITY RULES (Integrity Constraints)

There are two important integrity rules, which are constraints or restrictions that apply to all instances of the database. The two principal rules for the operational model are known as entity integrity and referential integrity. Before we define these terms, it is necessary to understand the concept of nulls.

Null or Unknown Value

Null represents a value for an attribute that is currently unknown or is not applicable for this tuple. A null can be taken to mean the logical value 'unknown'. It can ~ean that a value is not applicable to a particular tuple, or it could merely mean that. no value has yet been supplied. Nulls are a way to deal with incomplete or exceptional data. However a null is not the same as a zero numeric value or a text string filled with spaces; zeros and spaces are values, but a null represents the absence of a value. Therefore, nulls should be treated differently from other values.

For Example: Consider the following table



Name	Age	Job
Rajesh	--	Clerk
Raja	23	--
Amit	43	Sales

There is no entry for Rajesh's age in the table, obviously not because he does not have one, but simply because it was unknown to the organization at the point of time the 'snapshot' was taken. Raja might have just joined the organization and has not yet been assigned a job; therefore there is no value for the attribute 'job' which can be assigned to his row in the table. In time, it is to be expected that Rajesh's age will be determined and Raja will be assigned a job. A snapshot of the database at that time would reveal values in the currently blank places in the table.

Difference between Null and Not Applicable

In order to understand above concept Let us consider the following example:

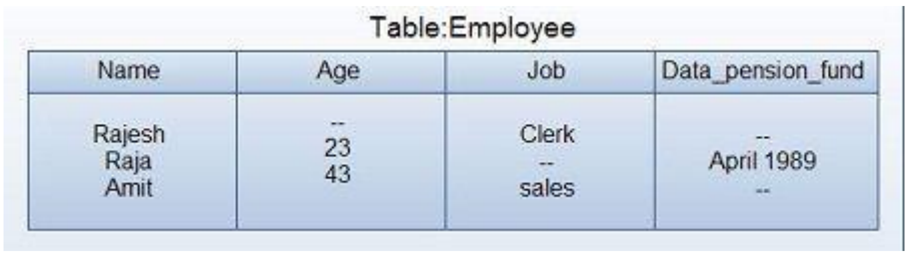


Table:Employee

Name	Age	Job	Data_pension_fund
Rajesh	--	Clerk	--
Raja	23	--	April 1989
Amit	43	sales	--

In this table the attribute 'date-pension fund' has been added which indicates the date when an employee joined the company pension fund. There are blank places for this as well as the original blanks under 'age' and 'job', but there could be different reasons for the blanks in the 'date-pension fund' column. The blank against Rajesh might indicate simply that, as for his age, the information about him is as yet incomplete. The reason for the blank against admit on the other hand, might be that he is a part time employee and, as such, ineligible to join the fund. In the first case, the information was simply unknown, in the second case the information was irrelevant to that particular employee. Where blanks occur in a relational table simply because the relevant facts are unknown, they are referred to as nulls. Thus the above table might be more properly represented as:

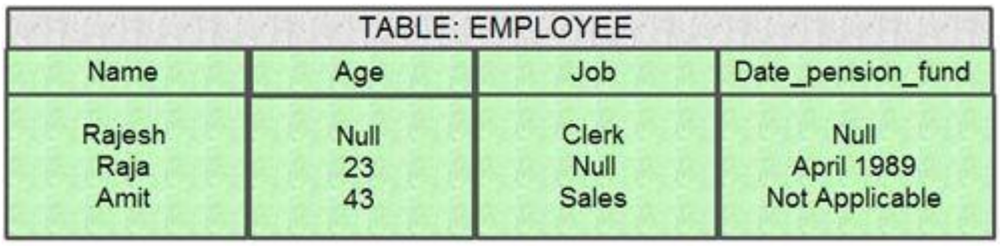
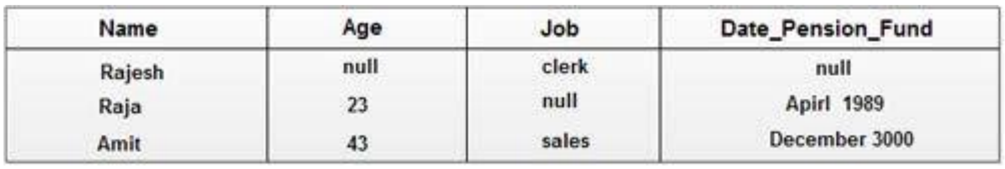


TABLE: EMPLOYEE

Name	Age	Job	Date_pension_fund
Rajesh	Null	Clerk	Null
Raja	23	Null	April 1989
Amit	43	Sales	Not Applicable

The problem with this is that the date values in the 'date-pension fund' column and the legend 'not applicable' in the same column are not syntactically or semantically compatible. This leads to practical difficulties. The problem can be resolved in this particular instance simply by having a 'nonsense' date to indicate non-applicability as in:



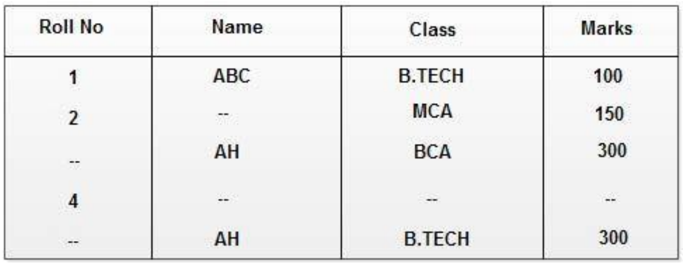
Name	Age	Job	Date_Pension_Fund
Rajesh	null	clerk	null
Raja	23	null	Apirl 1989
Amit	43	sales	December 3000

But not every case is so simple and it is very easy to confuse 'missing' with no applicable information. The important thing to note about nulls is that they are not values. It is not possible, for example, to say that one null equals another or is greater than another and so on.

Entity Integrity rule

Entity Integrity rule states that in a base relation, value of attribute of a primary key cannot be null. Here, a base relation is a relation that corresponds to an entity in the conceptual schema. By definition, a primary key is a minimal identifier that is used to identify tuples uniquely. This means that no subset of the primary key is sufficient to provide unique identification to tuples. If we allow a null for any part of a primary key, we are implying that not all the attributes are needed to distinguish between tuples, which contradicts the definition of the primary key. For example, as branch No is the primary key of the Branch relation, we should not be able to insert a tuple into the Branch relation with a null for the branch_No attribute.

In order to understand the concept of primary key and null, consider the following of STUDENT database.



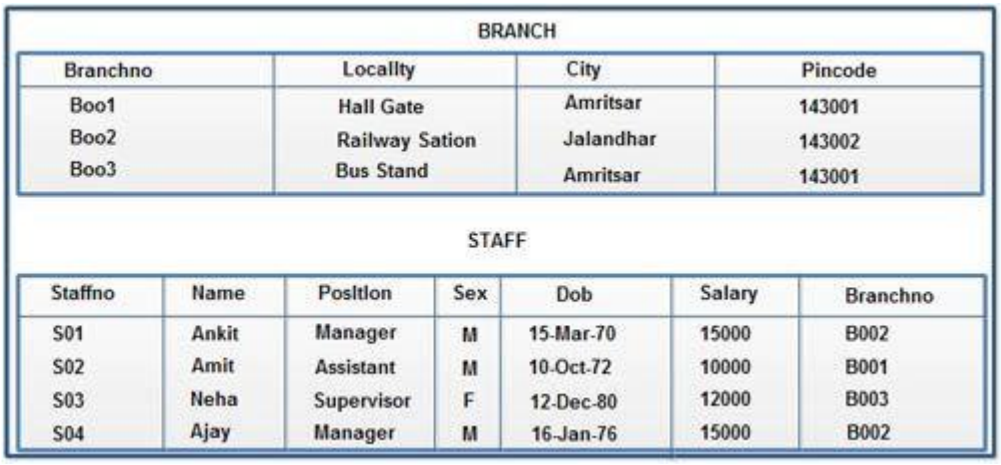
Roll No	Name	Class	Marks
1	ABC	B.TECH	100
2	--	MCA	150
--	AH	BCA	300
4	--	--	--
--	AH	B.TECH	300

Now record 3 and record 5 are not distinguishable as there Rollno is Null and it is not possible to exactly locate one of the students as he is having common name, class and marks with some other student. Record number 3 and 5 violates the rule of primary key because the Rollno is primary key, all the value of attributes are unique but their primary key entries are NULL and this violates the integrity rule 1, so these records are not allowed in RDBMS.

Referential Integrity

Referential Integrity rule states that, if a foreign key exists in a relation, either the foreign key value must match a candidate key value of some tuple in its home relation or the foreign key value must be wholly null.

Consider the following database to understand the referential integrity rule:



BRANCH

Branchno	Locality	City	Pincode
Boo1	Hall Gate	Amritsar	143001
Boo2	Railway Sation	Jalandhar	143002
Boo3	Bus Stand	Amritsar	143001

STAFF

Staffno	Name	Position	Sex	Dob	Salary	Branchno
S01	Ankit	Manager	M	15-Mar-70	15000	B002
S02	Amit	Assistant	M	10-Oct-72	10000	B001
S03	Neha	Supervisor	F	12-Dec-80	12000	B003
S04	Ajay	Manager	M	16-Jan-76	15000	B002

Here, branch_No in the Staff relation is a foreign key targeting the branch_No attribute in the· home relation, Branch. It should not be possible to create a staff record with branch number B025, unless there is already a record for branch number B025 in the Branch relation. However, we should be able to create a new staff record with a null branch number, to cater for the situation where a new member of staff has joined the company but has not yet been assigned to particular branch office.

Enterprise Constraints

Enterprise Constraints are additional rules specified by the users or database administrators of a database. It is also possible for users to specify additional constraints that the data must satisfy. For example, in case of STUDENT database if an upper limit of 300 has been placed upon the marks attribute of STUDENT database, then the user must be able to specify it and expect the DBMS to enforce it. In this case, it should not be possible to add a new student or update a record, whose marks are greater than 300.

EER MODEL (EXTENDED E-R MODEL/ENHANCED E-R MODEL)

EER is a high-level data model that incorporates the extensions to the original ER model.

It is a diagrammatic technique for displaying the following concepts

- *Sub Class and Super Class*
- *Specialization and Generalization*
- *Union or Category*
- *Aggregation*

These concepts are used when the comes in EER schema and the resulting schema diagrams called as EER Diagrams.

Features or EER Model

- EER creates a design more accurate to database schemas.
- It reflects the data properties and constraints more precisely.
- It includes all modeling concepts of the ER model.
- Diagrammatic technique helps for displaying the EER schema.
- It includes the concept of specialization and generalization.
- It is used to represent a collection of objects that is union of objects of different of different entity types.

A. Sub Class and Super Class

- Sub class and Super class relationship leads the concept of Inheritance.

1. Super Class

- Super class is an entity type that has a relationship with one or more subtypes.
- An entity cannot exist in database merely by being member of any super class.

  For example: Shape super class is having sub groups as Square, Circle, Triangle.

2. Sub Class

- Sub class is a group of entities with unique attributes.
- Sub class inherits properties and attributes from its super class.

  For example: Square, Circle, Triangle are the sub class of Shape super class.

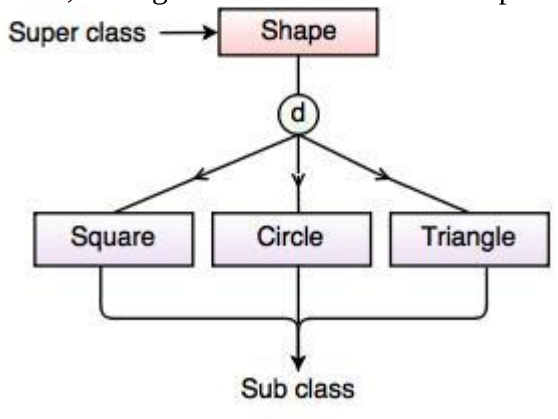


Fig. Super class/Sub class Relationship

B. Specialization and Generalization

1. Generalization

- Generalization is the process of generalizing the entities which contain the properties of all the generalized entities.
- It is a bottom approach, in which two lower level entities combine to form a higher level entity.
- Generalization is the reverse process of Specialization.
- It defines a general entity type from a set of specialized entity type.
- It minimizes the difference between the entities by identifying the common features.

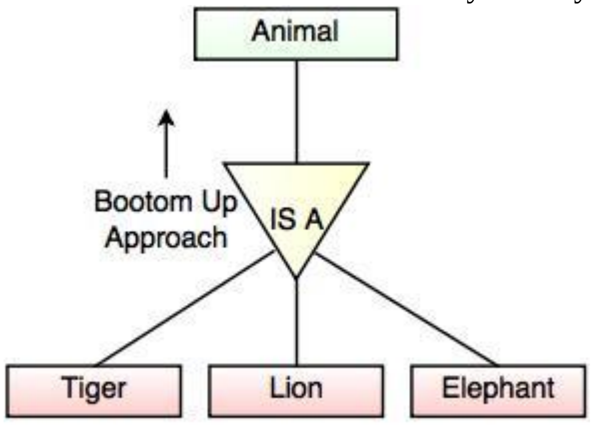


Fig. Generalization

2. Specialization

- Specialization is a process that defines a group entities which is divided into sub groups based on their characteristic.
- It is a top down approach, in which one higher entity can be broken down into two lower level entity.

- It maximizes the difference between the members of an entity by identifying the unique characteristic or attributes of each member.
- It defines one or more sub class for the super class and also forms the superclass/subclass relationship.

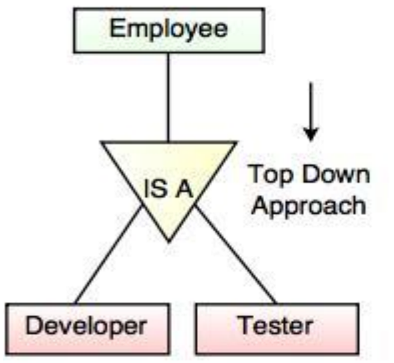


Fig. Specialization

C. Category or Union

- Category represents a single super class or sub class relationship with more than one super class.
- It can be a total or partial participation.

  For example Car booking, Car owner can be a person, a bank (holds a possession on a Car) or a company. Category (sub class) → Owner is a subset of the union of the three super classes → Company, Bank, and Person. A Category member must exist in at least one of its super classes.

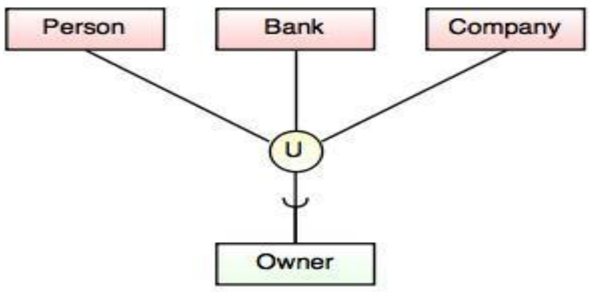


Fig. Categories (Union Type)

D. Aggregation

- Aggregation is a process that represent a relationship between a whole object and its component parts.
- It abstracts a relationship between objects and viewing the relationship as an object.
- It is a process when two entity is treated as a single entity.

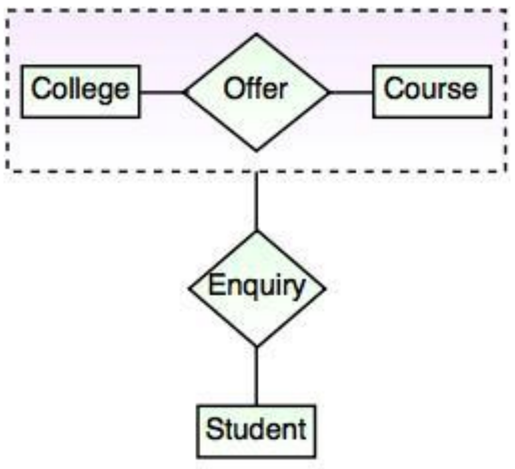


Fig. Aggregation

In the above example, the relation between College and Course is acting as an Entity in Relation with Student.

RELATIONAL DATA MODEL

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

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

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

Concepts:

- **Domains and attributes :** ***(refer ER data model)***
- **Keys:** ***(refer ER data model)***
- **Tuples:** ***(refer ER data model)***
- **Integrity rules:** ***(refer ER data model)***

RELATIONAL ALGEBRA

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

1. Basic operations:

i. Selection ( )
ii. Projection ( )
iii. Cross-product ( )
iv. Set-difference ( )
v. Union ( )

2. Additional operations:

i. Intersection
ii. Join
iii. Renaming

Selection Operation (σ)

It selects tuples that satisfy the given predicate from a relation.

Notation − σp(r)

Here σ stands for selection predicate and r stands for relation and p is a propositional logic formula/select condition which may use connectors like and, or, and not.

$\sigma_{\text{predicate}}$(R): This selection operation functions on a single relation R and describes a relation which contains only those tuples of R that satisfy the specified condition (predicate).

Example:

σ_{teacher} = "database"(Names)

Output - It selects tuples from names where the teacher is 'database.'

Project Operation (∏)

The Projection operation works on a single relation R and defines a relation that contains a vertical subset of R, extracting the values of specified attributes and eliminating duplicates.

Produce a list of salaries for all staff, showing only the staffNo, fName, lName, and

salary details.

ΠstaffNo, fName, lName, salary(Staff)

In the below-mentioned example, the Projection operation defines a relation that contains only the designated Staff attributes staffNo, fName, lName, and salary, in the specified order. The result of this operation is shown in the figure below

Union Operation

For R ∪ S, The union of two relations R and S defines a relation that contains all the tuples of R, or S, or both R and S, duplicate tuples being eliminated. R and S must be union-compatible.

For a union operation to be applied, the following rules must hold −

- r and s must have the same quantity of attributes.
- Attribute domains must be compatible.
- Duplicate tuples get automatically eliminated.

Set difference

For R − S The Set difference operation defines a relation consisting of the tuples that are in relation R, but not in S. R and S must be union-compatible.

Example:

$\prod_{\text{writer}}$ (Nobels) − $\prod_{\text{writer}}$ (papers)

Cartesian product

For R × S, the Cartesian product operation defines a relation that is the concatenation of every tuple of relation R with every tuple of relation S.

Example:

$\sigma_{\text{writer}} = _{\text{'gauravray'}}$(Articles X Notes)

Joins

Join is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.We will briefly describe various join types in the following sections.

Theta θ Join

Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol **θ**.

Notation R1 $\bowtie_{\theta}$ R2

R1 and R2 are relations having attributes *A*1, *A*2, . . , *An* and *B*1, *B*2, . . , *Bn* such that the attributes don't have anything in common, that is R1 ∩ R2 = Φ. Theta join can use all kinds of comparison operators.

Student				**Subjects**	
SID	**Name**	**Std**		**Class**	**Subject**
101	Alex	10		10	English
102	Maria	11		10	English
				11	Music
				11	sports

Student_Detail − $\text{STUDENT} \bowtie_{\text{Student.Std = Subject.Class}} \text{SUBJECT}$

Student_detail

SID	**Name**	**Std**	**Class**	**Subject**
101	Alex	10	10	Math
101	Alex	10	10	English
102	Maria	11	11	Music
102	Maria	11	11	Sports

Equijoin

When Theta join uses only **equality** comparison operator, it is said to be equijoin. The above example corresponds to equijoin.

Natural Join (⋈)

Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between two relations. In addition, the attributes must have the same name and domain.

Natural join acts on those matching attributes where the values of attributes in both the relations are same.

Courses				**HoD**	
CID	**Course**	**Dept**		**Dept**	**Head**
CS01	Database	CS		CS	Alex
ME01	Mechanics	ME		ME	Maya
EE01	Electronics	EE		EE	Mira

Courses ⋈ HoD

Dept	**CID**	**Course**	**Head**
CS	CS01	Database	Alex
ME	ME01	Mechanics	Maya
EE	EE01	Electronics	Mira

Outer Joins

Theta Join, Equijoin, and Natural Join are called inner joins. An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation. Therefore, we need to use outer joins to include all the tuples from the participating relations in the resulting relation. There are three kinds of outer joins − *left outer join, right outer join, and full outer join*.

Left Outer Join(R ⟕ S)

All the tuples from the Left relation, R, are included in the resulting relation. If there are tuples in R without any matching tuple in the Right relation S, then the S-attributes of the resulting relation are made NULL.

Left			**Right**	
A	**B**		**C**	**D**
100	Database		100	Alex
101	Mechanics		102	Maya
102	Electronics		104	Mira

Courses ⟕ HoD

Left		Right	
A	**B**	**C**	**D**
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya

Right Outer Join: (R ⟖ S)

All the tuples from the Right relation, S, are included in the resulting relation. If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made NULL.

Courses ⟖ HoD

Left		Right	
A	**B**	**C**	**D**
100	Database	100	Alex
102	Electronics	102	Maya
---	---	104	Mira

Full Outer Join: (R ⟗ S)

All the tuples from both participating relations are included in the resulting relation. If there are no matching tuples for both relations, their respective unmatched attributes are made NULL.

Courses ⟗ HoD

Left		Right	
A	**B**	**C**	**D**
101	Mechanics	---	---
102	Electronics	102	Maya
---	---	104	Mira