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

Data becomes **information** when it is processed, turning it into something meaningful. Like, based on the cookie data saved on user's browser, if a website can analyse that generally men of age 20-25 visit us more,

What is a Database?

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

What is DBMS?

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

DBMS also provides protection and security to the databases. It also maintains data consistency in case of multiple users.

Here are some examples of popular DBMS used these days:

- MySql
- Oracle

- SQL Server
- IBM DB2
- PostgreSQL

Characteristics of Database Management System

A database management system has following characteristics:

1. **Data stored into Tables:** Data is never directly stored into the database. Data is stored into tables, created inside the database. DBMS also allows to have relationships between tables which makes the data more meaningful and connected. You can easily understand what type of data is stored where by looking at all the tables created in a database.
2. **Reduced Redundancy:** In the modern world hard drives are very cheap, but earlier when hard drives were too expensive, unnecessary repetition of data in database was a big problem. But DBMS follows **Normalisation** which divides the data in such a way that repetition is minimum.
3. **Data Consistency:** On Live data, i.e. data that is being continuosly updated and added, maintaining the consistency of data can become a challenge. But DBMS handles it all by itself.
4. **Support Multiple user and Concurrent Access:** DBMS allows multiple users to work on it(update, insert, delete data) at the same time and still manages to maintain the data consistency.
5. **Query Language:** DBMS provides users with a simple Query language, using which data can be easily fetched, inserted, deleted and updated in a database.
6. **Security:** The DBMS also takes care of the security of data, protecting the data from un-authorised access. In a typical

DBMS, we can create user accounts with different access permissions, using which we can easily secure our data by restricting user access.

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

Advantages of DBMS

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

There are the following differences between DBMS and File systems:

Basis	DBMS Approach	File System Approach
Meaning	DBMS is a collection of data. In DBMS, the user is not required to write the procedures.	The file system is a collection of data. In this system, the user has to write the procedures for managing the database.

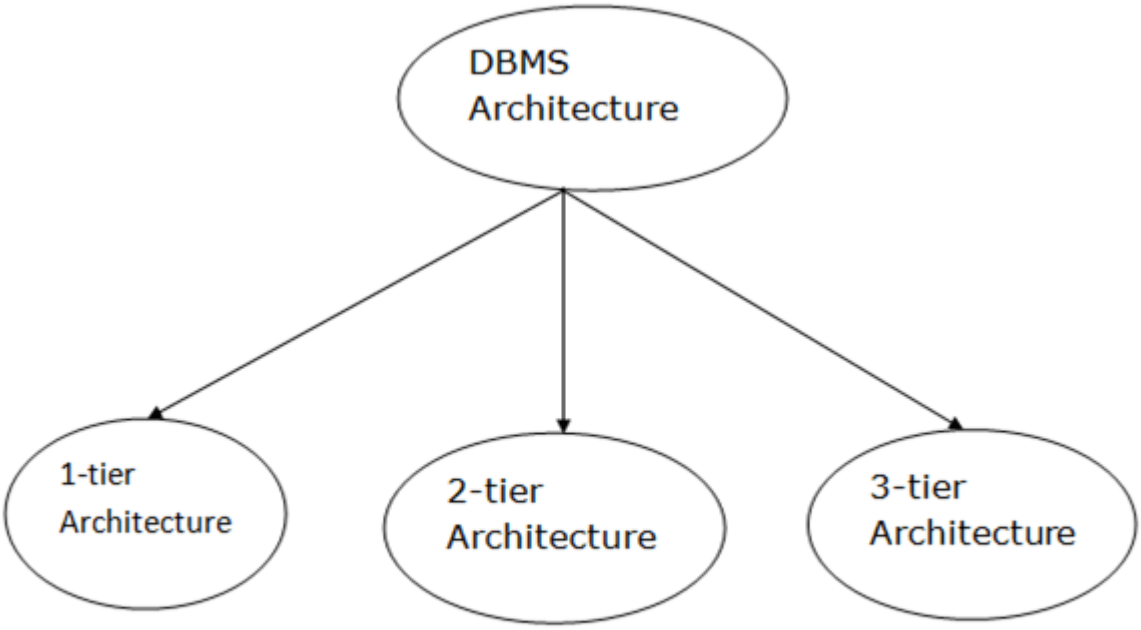
Sharing of data	Due to the centralized approach, data sharing is easy.	Data is distributed in many files, and it may be of different formats, so it isn't easy to share data.
Data Abstraction	DBMS gives an abstract view of data that hides the details.	The file system provides the detail of the data representation and storage of data.
Security and Protection	DBMS provides a good protection mechanism.	It isn't easy to protect a file under the file system.
Recovery Mechanism	DBMS provides a crash recovery mechanism, i.e., DBMS protects the user from system failure.	The file system doesn't have a crash mechanism, i.e., if the system crashes while entering some data, then the content of the file will be lost.
Manipulation Techniques	DBMS contains a wide variety of sophisticated techniques to store and retrieve the data.	The file system can't efficiently store and retrieve the data.
Concurrency Problems	DBMS takes care of Concurrent access of data using some form of locking.	In the File system, concurrent access has many problems like redirecting the file while deleting some information or updating some information.
Where to use	Database approach used in large systems which interrelate many files.	File system approach used in large systems which interrelate many files.
Cost	The database system is expensive to design.	The file system approach is cheaper to design.
Data Redundancy and Inconsistency	Due to the centralization of the database, the problems of data redundancy and inconsistency are controlled.	In this, the files and application programs are created by different programmers so that there exists a lot of duplication of data which may lead to inconsistency.
Structure	The database structure is complex to design.	The file system approach has a simple structure.

Data Independence	In this system, Data Independence exists, and it can be of two types. o Logical Data Independence o Physical Data Independence	In the File system approach, there exists no Data Independence.
Integrity Constraints	Integrity Constraints are easy to apply.	Integrity Constraints are difficult to implement in file system.
Data Models	In the database approach, 3 types of data models exist: o Hierarchal data models o Network data models o Relational data models	In the file system approach, there is no concept of data models exists.
Flexibility	Changes are often a necessity to the content of the data stored in any system, and these changes are more easily with a database approach.	The flexibility of the system is less as compared to \the DBMS approach.
Examples	Oracle, SQL Server, Sybase etc.	Cobol, C++ etc.

DBMS Architecture

- The DBMS design depends upon its architecture. The basic client/server architecture is used to deal with a large number of PCs, web servers, database servers and other components that are connected with networks.
- The client/server architecture consists of many PCs and a workstation which are connected via the network.
- DBMS architecture depends upon how users are connected to the database to get their request done.

Types of DBMS Architecture



Database architecture can be seen as a single tier or multi-tier. But logically, database architecture is of two types like: **2-tier architecture** and **3-tier architecture**.

1-Tier Architecture

- In this architecture, the database is directly available to the user. It means the user can directly sit on the DBMS and uses it.
- Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.
- The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.

2-Tier Architecture

- The 2-Tier architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like: **ODBC**, **JDBC** are used.
- The user interfaces and application programs are run on the client-side.

- The server side is responsible to provide the functionalities like: query processing and transaction management.
- To communicate with the DBMS, client-side application establishes a connection with the server side.

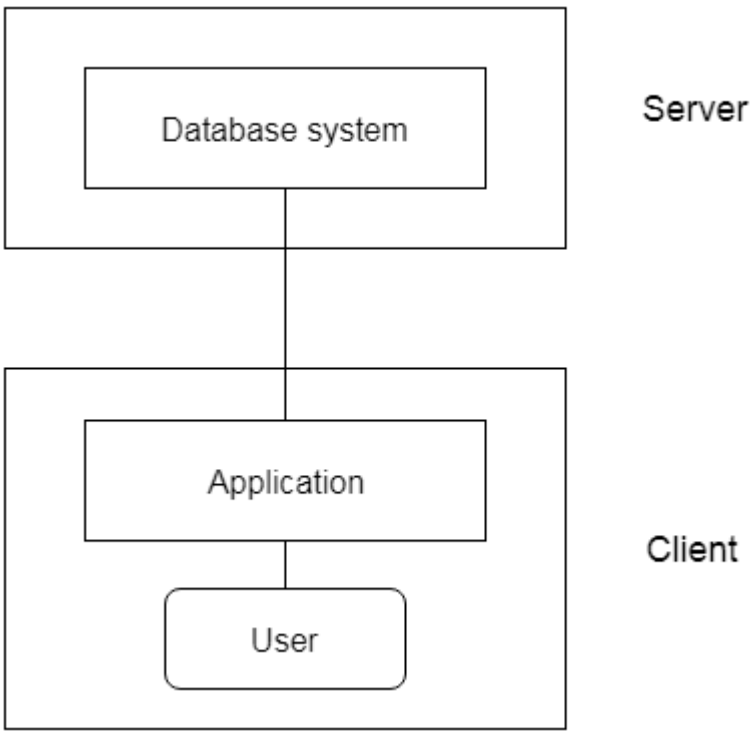


Fig: 2-tier Architecture

3-Tier Architecture

- The 3-Tier architecture contains another layer between the client and server. In this architecture, client can't directly communicate with the server.
- The application on the client-end interacts with an application server which further communicates with the database system.
- End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.
- The 3-Tier architecture is used in case of large web application.

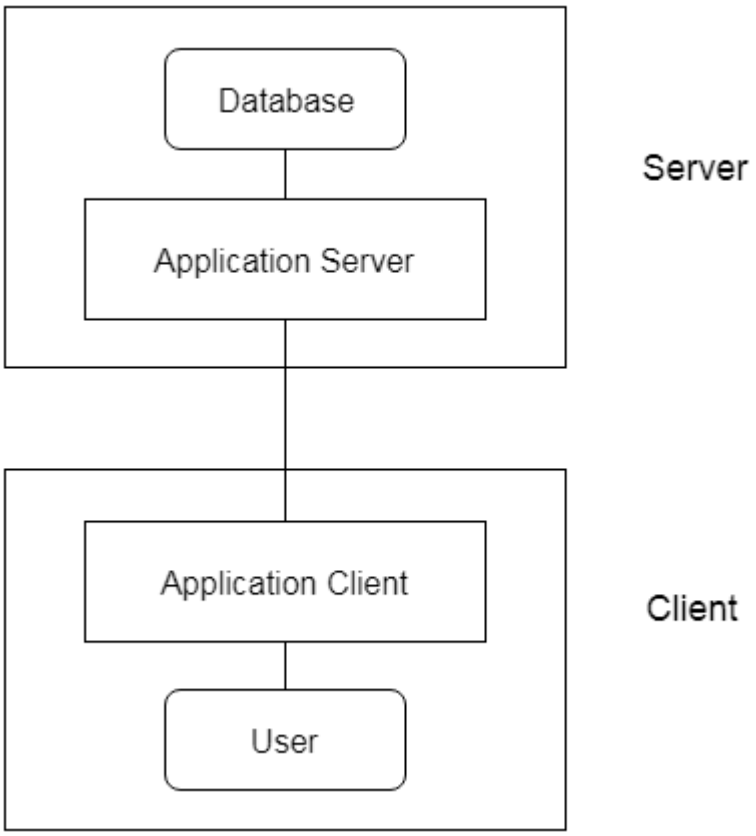


Fig: 3-tier Architecture

Data model Schema and Instance

- The data which is stored in the database at a particular moment of time is called an instance of the database.
- The overall design of a database is called schema.
- A database schema is the skeleton structure of the database. It represents the logical view of the entire database.
- A schema contains schema objects like table, foreign key, primary key, views, columns, data types, stored procedure, etc.
- A database schema can be represented by using the visual diagram. That diagram shows the database objects and relationship with each other.
- A database schema is designed by the database designers to help programmers whose software will interact with the database. The process of database creation is called data modeling.

A schema diagram can display only some aspects of a schema like the name of record type, data type, and constraints. Other aspects can't be specified through the schema diagram. For example, the given figure neither show the data type of each data item nor the relationship among various files.

In the database, actual data changes quite frequently. For example, in the given figure, the database changes whenever we add a new grade or add a student. The data at a particular moment of time is called the instance of the database.

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

Data Independence

- Data independence can be explained using the three-schema architecture.
- Data independence refers characteristic of being able to modify the schema at one level of the database system without altering the schema at the next higher level.

There are two types of data independence:

1. Logical Data Independence

- Logical data independence refers characteristic of being able to change the conceptual schema without having to change the external schema.
- Logical data independence is used to separate the external level from the conceptual view.
- If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.
- Logical data independence occurs at the user interface level.

2. Physical Data Independence

- Physical data independence can be defined as the capacity to change the internal schema without having to change the conceptual schema.
- If we do any changes in the storage size of the database system server, then the Conceptual structure of the database will not be affected.
- Physical data independence is used to separate conceptual levels from the internal levels.
- Physical data independence occurs at the logical interface level.

Database Language

- A DBMS has appropriate languages and interfaces to express database queries and updates.
- Database languages can be used to read, store and update the data in the database.

Types of Database Language

1. Data Definition Language

- **DDL** stands for **D**ata **D**efinition **L**anguage. It is used to define database structure or pattern.
- It is used to create schema, tables, indexes, constraints, etc. in the database.
- Using the DDL statements, you can create the skeleton of the database.

- Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Here are some tasks that come under DDL:

- **Create:** It is used to create objects in the database.
- **Alter:** It is used to alter the structure of the database.
- **Drop:** It is used to delete objects from the database.
- **Truncate:** It is used to remove all records from a table.
- **Rename:** It is used to rename an object.

These commands are used to update the database schema that's why they come under Data definition language.

2. Data Manipulation Language

DML stands for **D**ata **M**anipulation **L**anguage. It is used for accessing and manipulating data in a database. It handles user requests.

- **Select:** It is used to retrieve data from a database.
- **Insert:** It is used to insert data into a table.
- **Update:** It is used to update existing data within a table.
- **Delete:** It is used to delete all records from a table.
- **Merge:** It performs UPSERT operation, i.e., insert or update operations.
- **Call:** It is used to call a structured query language or a Java subprogram.
- **Explain Plan:** It has the parameter of explaining data.
- **Lock Table:** It controls concurrency.

3. Data Control Language

- **DCL** stands for **D**ata **C**ontrol **L**anguage. It is used to retrieve the stored or saved data.
- The DCL execution is transactional. It also has rollback parameters.

  (But in Oracle database, the execution of data control language does not have the feature of rolling back.)

Here are some tasks that come under DCL:

- **Grant:** It is used to give user access privileges to a database.

- **Revoke:** It is used to take back permissions from the user.

There are the following operations which have the authorization of Revoke:

CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

4. Transaction Control Language

TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

Here are some tasks that come under TCL:

- **Commit:** It is used to save the transaction on the database.
- **Rollback:** It is used to restore the database to original since the last Commit.

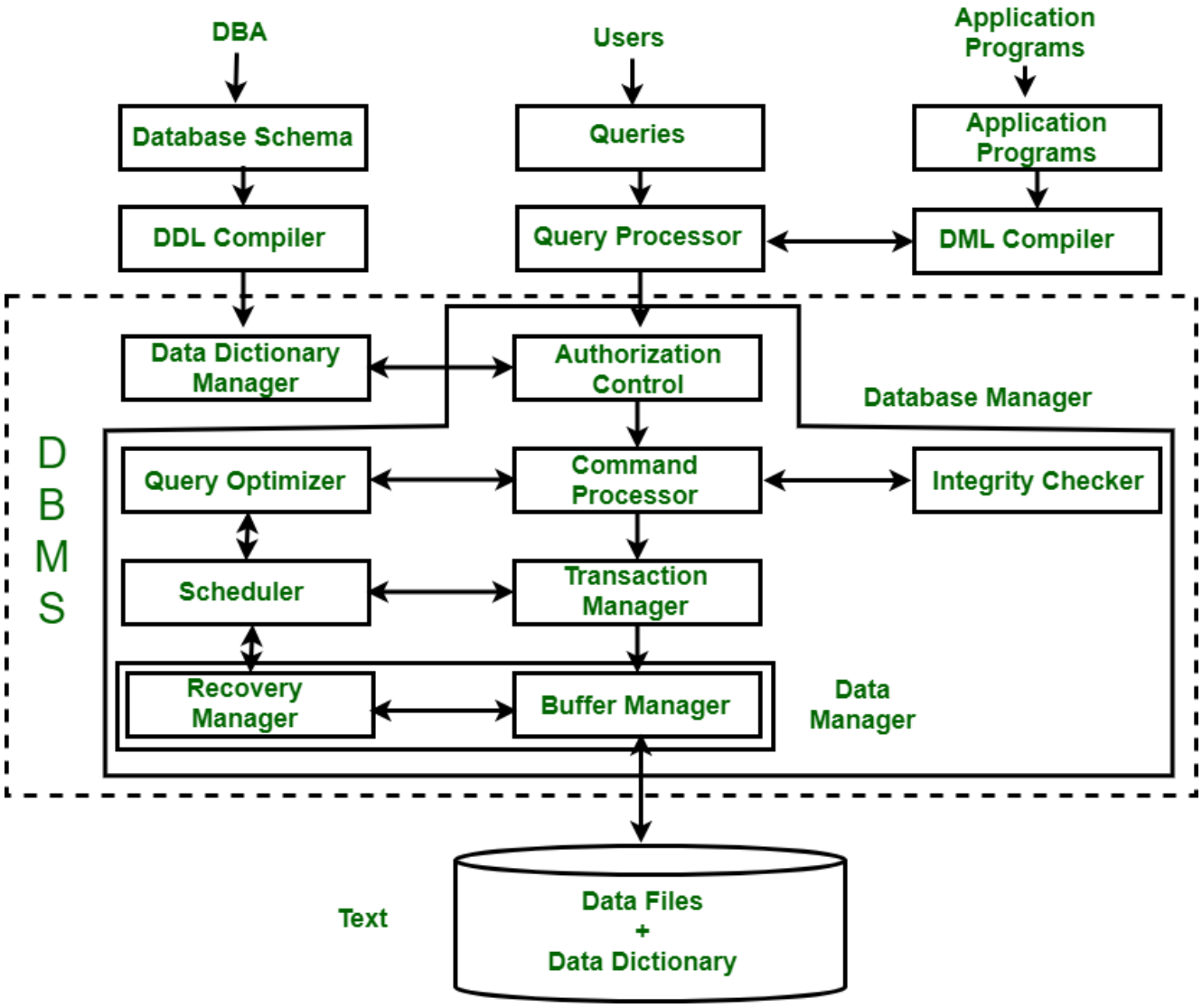
Structure of Database Management System

Database Management System (DBMS) is a software that allows access to data stored in a database and provides an easy and effective method of –

- Defining the information.
- Storing the information.
- Manipulating the information.
- Protecting the information from system crashes or data theft.
- Differentiating access permissions for different users.

Please be note that the Structure of Database Management System is also referred to as ***Overall System Structure*** *or* ***Database Architecture*** *but it is different from the* ***tier architecture*** *of Database****.***

The database system is divided into three components: Query Processor, Storage Manager, and Disk Storage. These are explained as following below.



Architecture of DBMS

1. Query Processor :
It interprets the requests (queries) received from end user via an application program into instructions. It also executes the user request which is received from the DML compiler.
Query Processor contains the following components –

- **DML Compiler –**
  It processes the DML statements into low level instruction (machine language), so that they can be executed.

- **DDL Interpreter –**
  It processes the DDL statements into a set of table containing meta data (data about data).

- **Embedded DML Pre-compiler –**
  It processes DML statements embedded in an application program into procedural calls.

- **Query Optimizer –**
  It executes the instruction generated by DML Compiler.

2. Storage Manager :
Storage Manager is a program that provides an interface between the data stored in the database and the queries received. It is also known as Database Control System. It maintains the consistency and integrity of the database by applying the constraints and executes the DCL statements. It is responsible for updating, storing, deleting, and retrieving data in the database.
It contains the following components –

- **Authorization Manager –**
  It ensures role-based access control, i.e,. checks whether the particular person is privileged to perform the requested operation or not.

- **Integrity Manager –**
  It checks the integrity constraints when the database is modified.

- **Transaction Manager –**
  It controls concurrent access by performing the operations in a scheduled way that it receives the transaction. Thus, it ensures that the database remains in the consistent state before and after the execution of a transaction.

- **File Manager –**
  It manages the file space and the data structure used to represent information in the database.

- **Buffer Manager –**
  It is responsible for cache memory and the transfer of data between the secondary storage and main memory.

3. Disk Storage: It contains the following components –

- **Data Files –**
  It stores the data.

- **Data Dictionary –**
  It contains the information about the structure of any database object. It is the repository of information that governs the metadata.

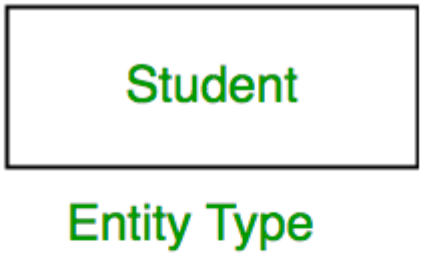
- **Indices –**
  It provides faster retrieval of data item.

Introduction of ER Model

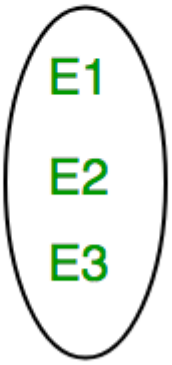
ER Model is used to model the logical view of the system from data perspective which consists of these components:

Entity, Entity Type, Entity Set –
An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.

An Entity is an object of Entity Type and set of all entities is called as entity set. e.g.; E1 is an entity having Entity Type Student and set of all students is called Entity Set. In ER diagram, Entity Type is represented as:



Entity Type



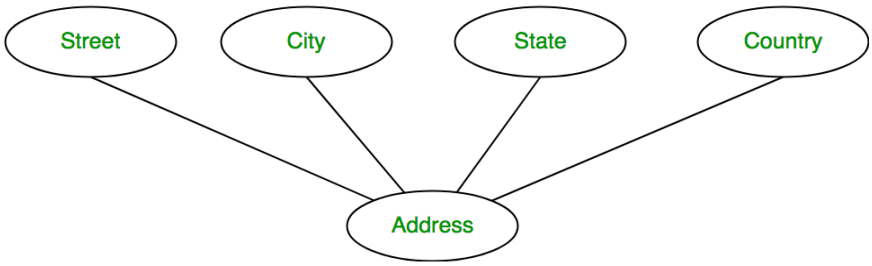
Entity Set

Attribute(s):
Attributes are the **properties which define the entity type**. For example, Roll_No, Name, DOB, Age, Address, Mobile_No are the attributes which defines entity type Student. In ER diagram, attribute is represented by an oval.

1. Key Attribute –
The attribute which **uniquely identifies each entity** in the entity set is called

key attribute.For example, Roll_No will be unique for each student. In ER diagram, key attribute is represented by an oval with underlying lines.

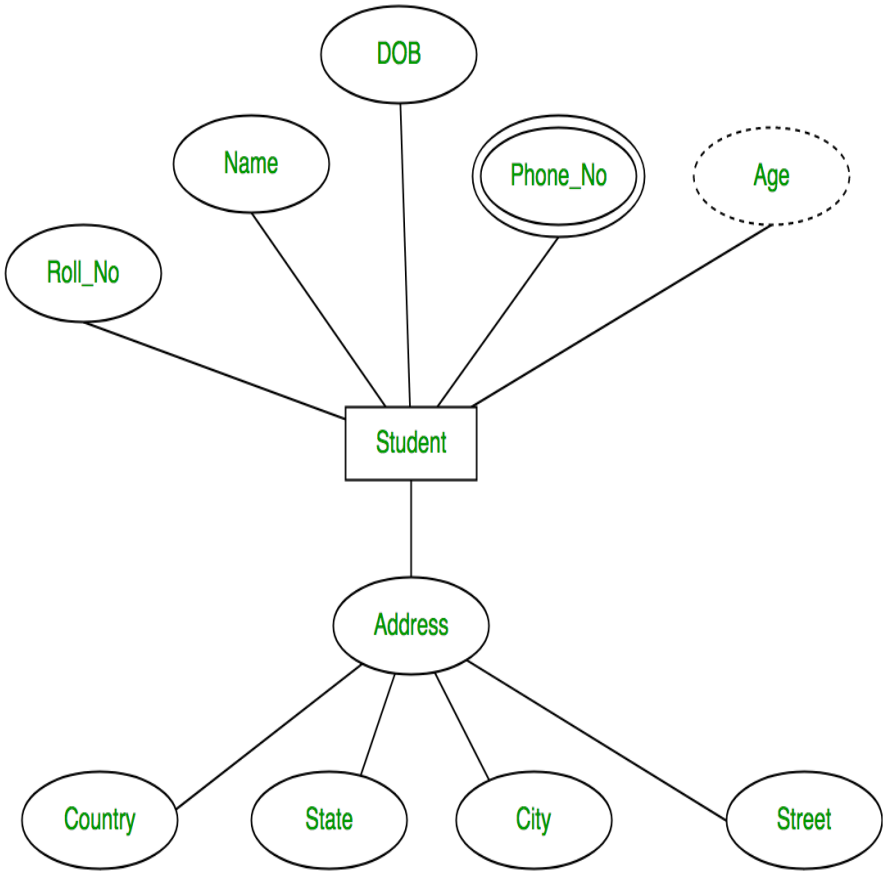
2. Composite Attribute –
An attribute **composed of many other attribute** is called as composite attribute. For example, Address attribute of student Entity type consists of Street, City, State, and Country. In ER diagram, composite attribute is represented by an oval comprising of ovals.



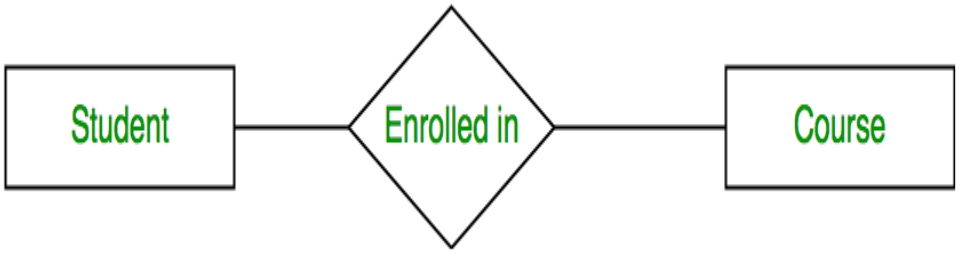
3. Multivalued Attribute –
An attribute consisting **more than one value** for a given entity. For example, Phone_No (can be more than one for a given student). In ER diagram, multivalued attribute is represented by double oval.

4. Derived Attribute –
An attribute which can be **derived from other attributes** of the entity type is known as derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, derived attribute is represented by dashed oval.

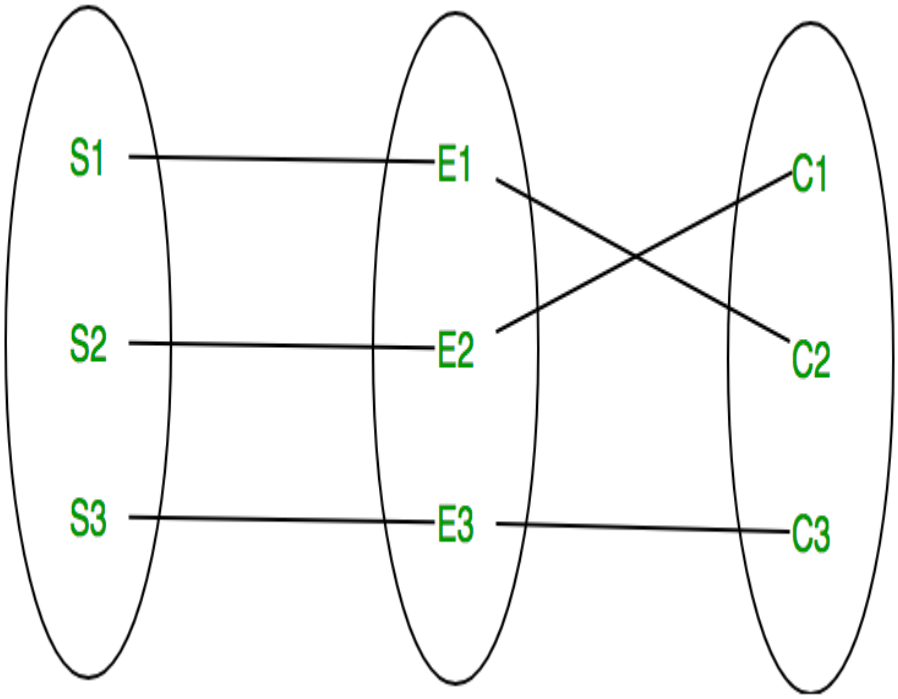
The complete entity type **Student** with its attributes can be represented as:



Relationship Type and Relationship Set:
A relationship type represents the **association between entity types**. For example,'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, relationship type is represented by a diamond and connecting the entities with lines.

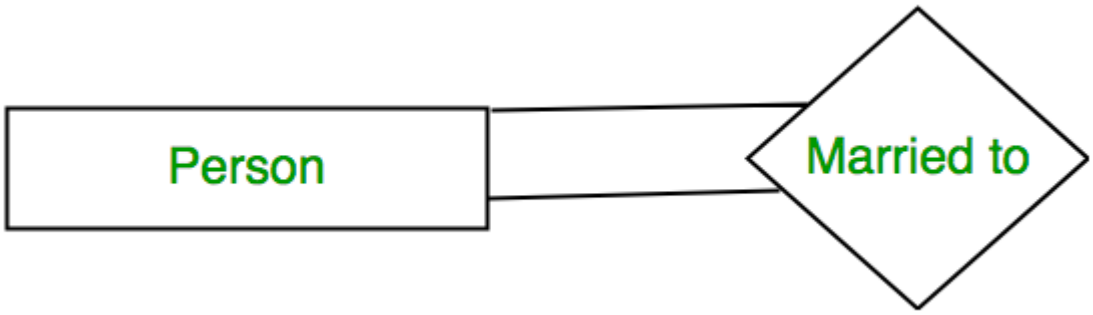


A set of relationships of same type is known as relationship set. The following relationship set depicts S1 is enrolled in C2, S2 is enrolled in C1 and S3 is enrolled in C3.

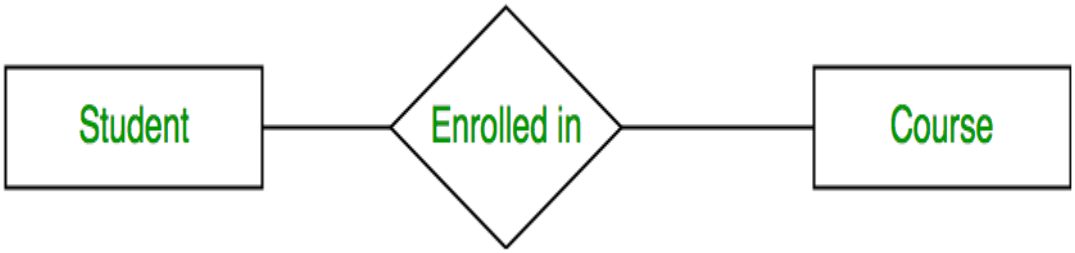


Degree of a relationship set:
The number of different entity sets **participating in a relationship** set is called as degree of a relationship set.

1. Unary Relationship –
When there is **only ONE entity set participating in a relation**, the relationship is called as unary relationship. For example, one person is married to only one person.

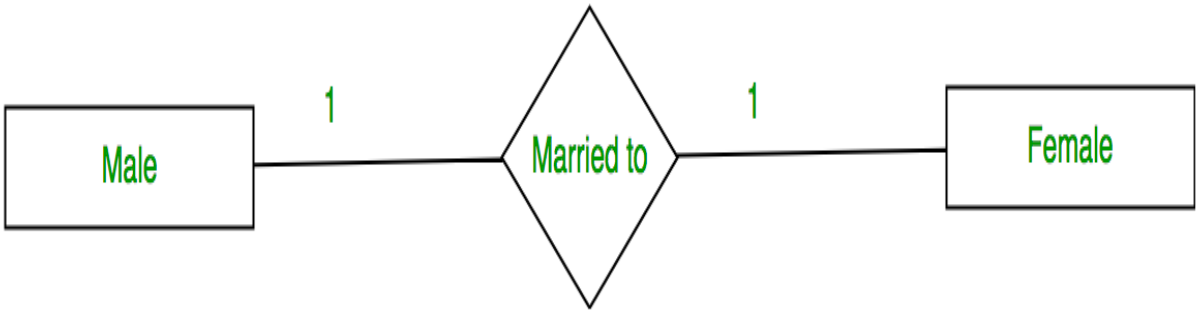


2. Binary Relationship –
When there are **TWO entities set participating in a relation**, the relationship is called as binary relationship.For example, Student is enrolled in Course.

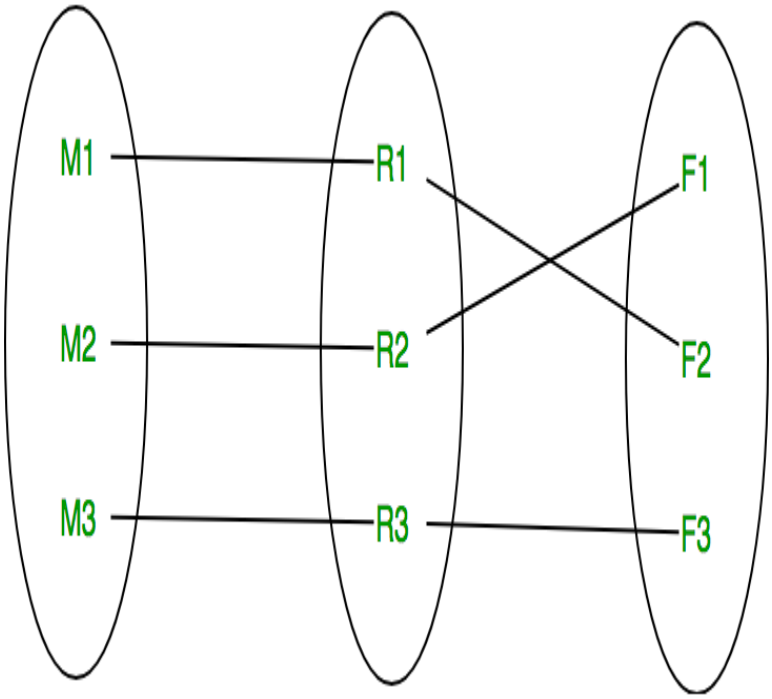


3. n-ary Relationship –
When there are n entities set participating in a relation, the relationship is called as n-ary relationship.

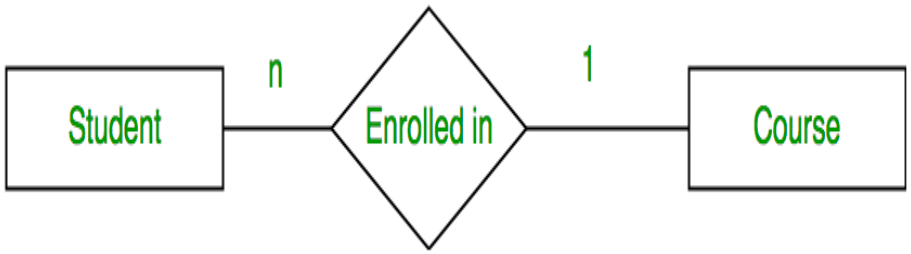
Cardinality:
The **number of times an entity of an entity set participates in a relationship** set is known as cardinality. Cardinality can be of different types:
1. One to one – When each entity in each entity set can take part **only once in the relationship**, the cardinality is one to one. Let us assume that a male can marry to one female and a female can marry to one male. So the relationship will be one to one.



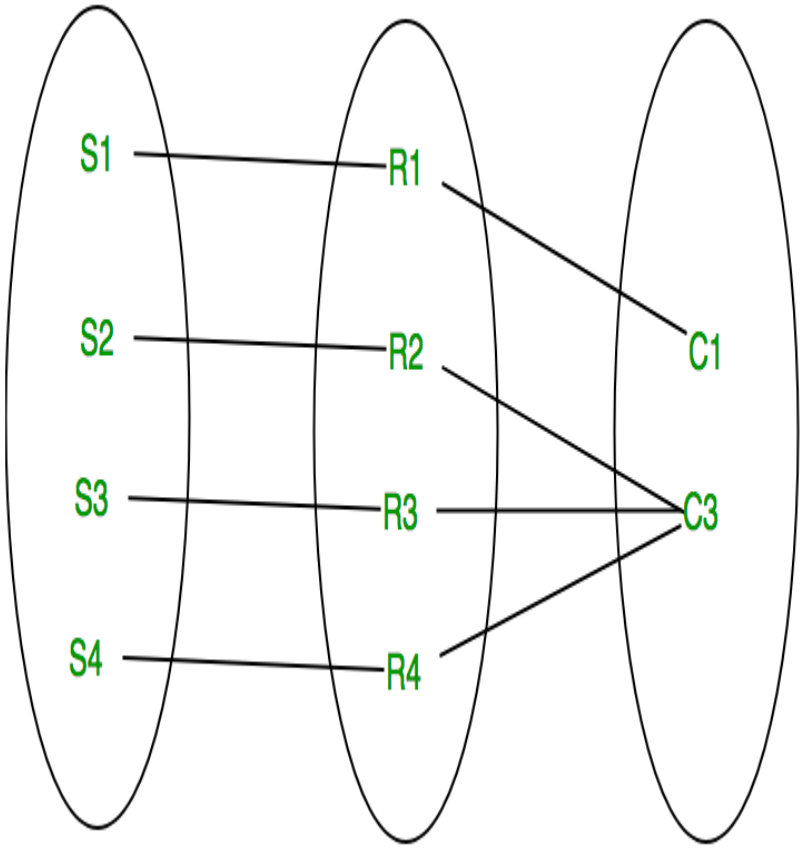
Using Sets, it can be represented as:



2. Many to one – When entities in one entity set **can take part only once in the relationship set and entities in other entity set can take part more than once in the relationship set,** cardinality is many to one. Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student, there will be only one course.

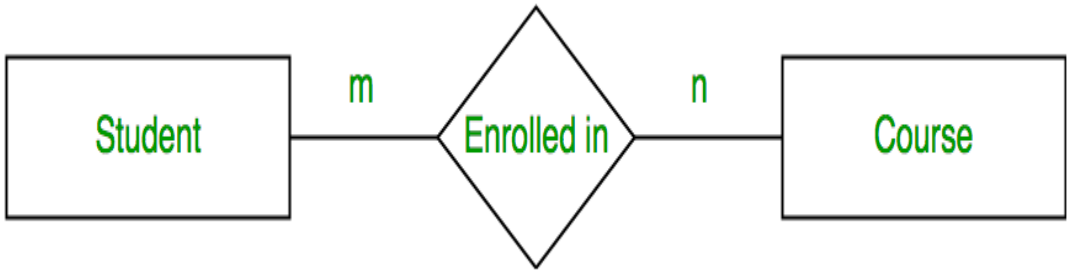


Using Sets, it can be represented as:

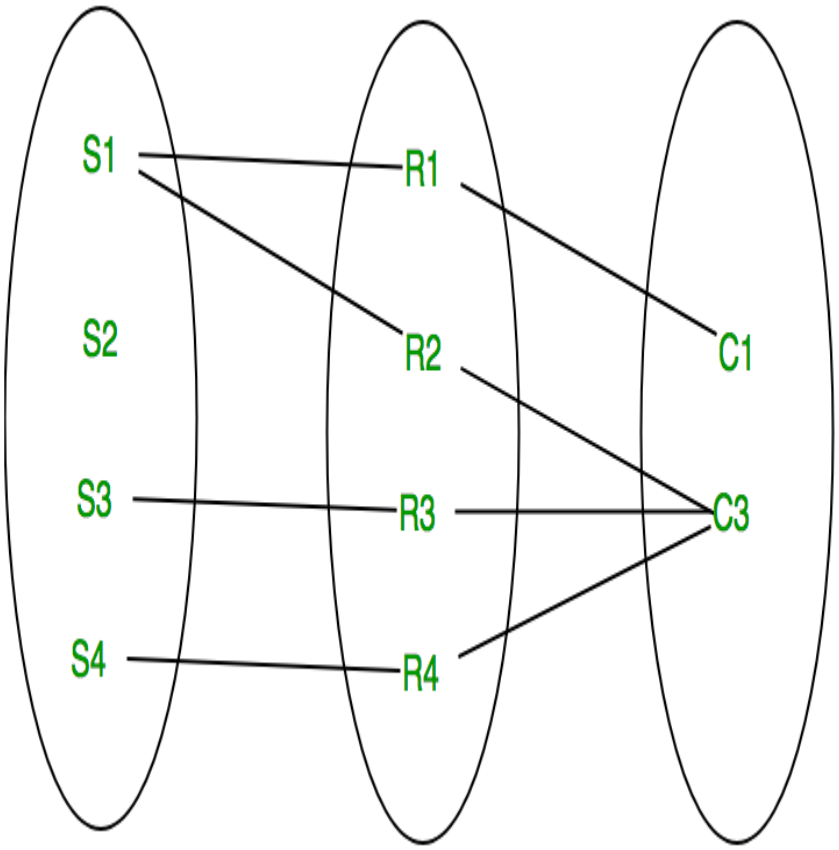


In this case, each student is taking only 1 course but 1 course has been taken by many students.

3. Many to many – When entities in all entity sets can **take part more than once in the relationship** cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.

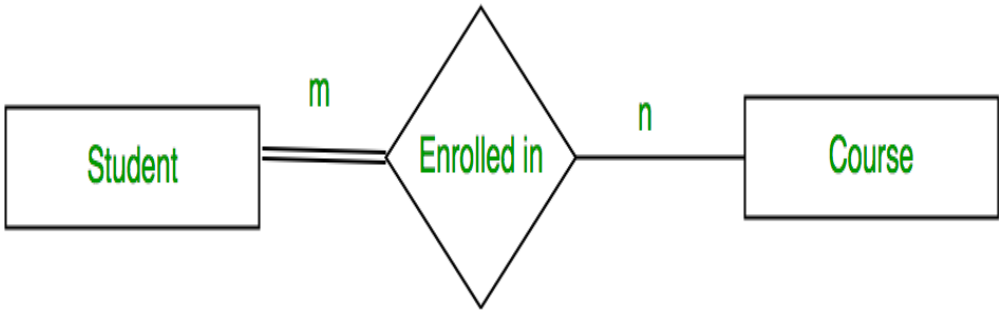


Using sets, it can be represented as:

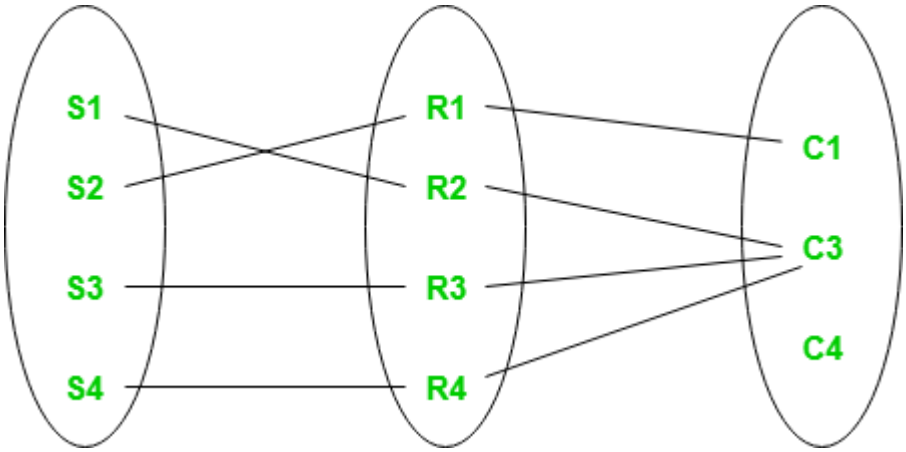


In this example, student S1 is enrolled in C1 and C3 and Course C3 is enrolled by S1, S3 and S4. So it is many to many relationships.

Participation Constraint:
Participation Constraint is applied on the entity participating in the relationship set.
1. Total Participation – Each entity in the entity set **must participate** in the relationship. If each student must enroll in a course, the participation of student will be total. Total participation is shown by double line in ER diagram.
2. Partial Participation – The entity in the entity set **may or may NOT participat**e in the relationship. If some courses are not enrolled by any of the student, the participation of course will be partial.
The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation.



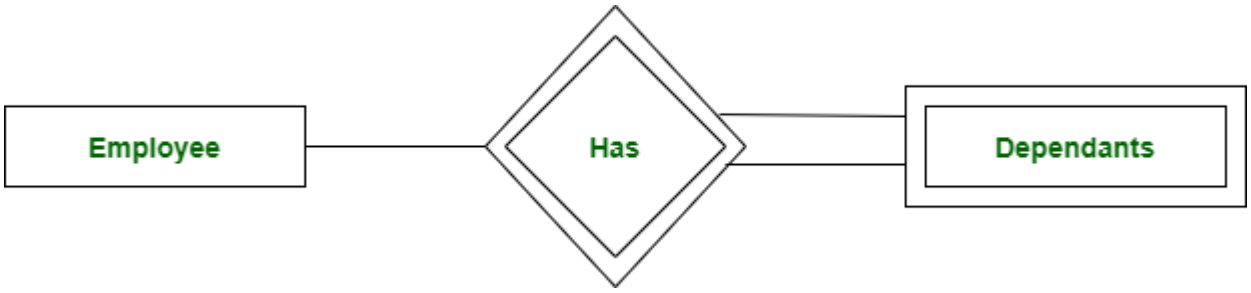
Using set, it can be represented as,



Every student in Student Entity set is participating in relationship but there exists a course C4 which is not taking part in the relationship.

Weak Entity Type and Identifying Relationship:
As discussed before, an entity type has a key attribute which uniquely identifies each entity in the entity set. But there exists **some entity type for which key attribute can't be defined**. These are called Weak Entity type. For example, A company may store the information of dependents (Parents, Children, Spouse) of an Employee. But the dependents don't have existence without the employee. So Dependent will be weak entity type and Employee will be Identifying Entity type for Dependent.

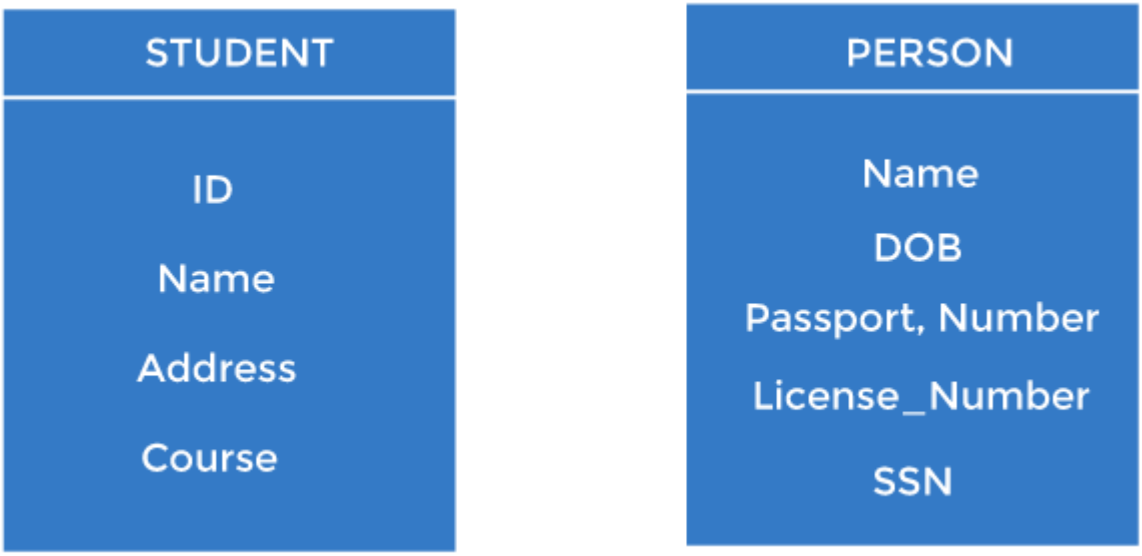
A weak entity type is represented by a double rectangle. The participation of weak entity type is always total. The relationship between weak entity type and its identifying strong entity type is called identifying relationship and it is represented by double diamond.



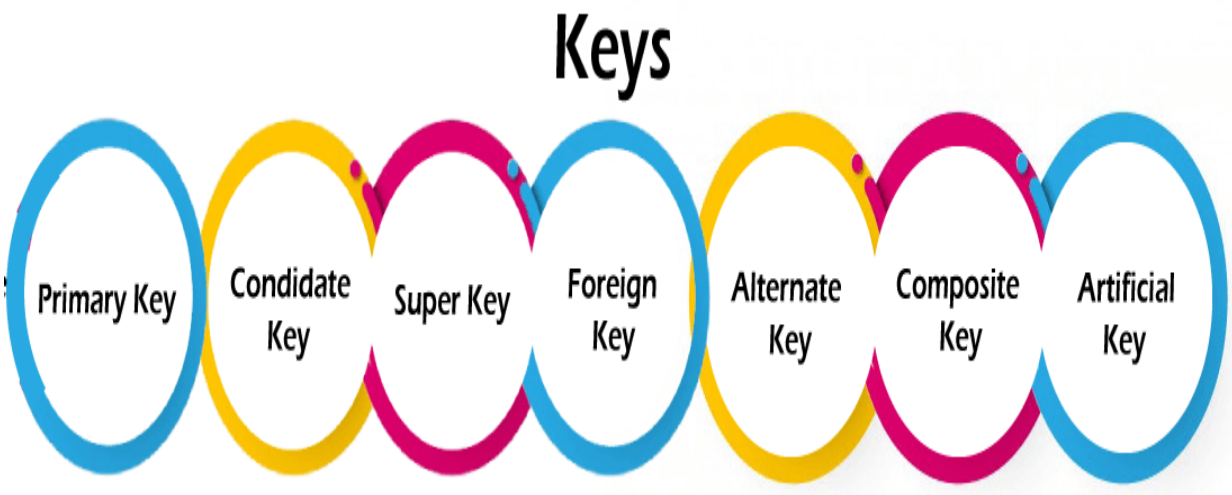
Keys

- Keys play an important role in the relational database.
- It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

For example, ID is used as a key in the Student table because it is unique for each student. In the PERSON table, passport_number, license_number, SSN are keys since they are unique for each person.

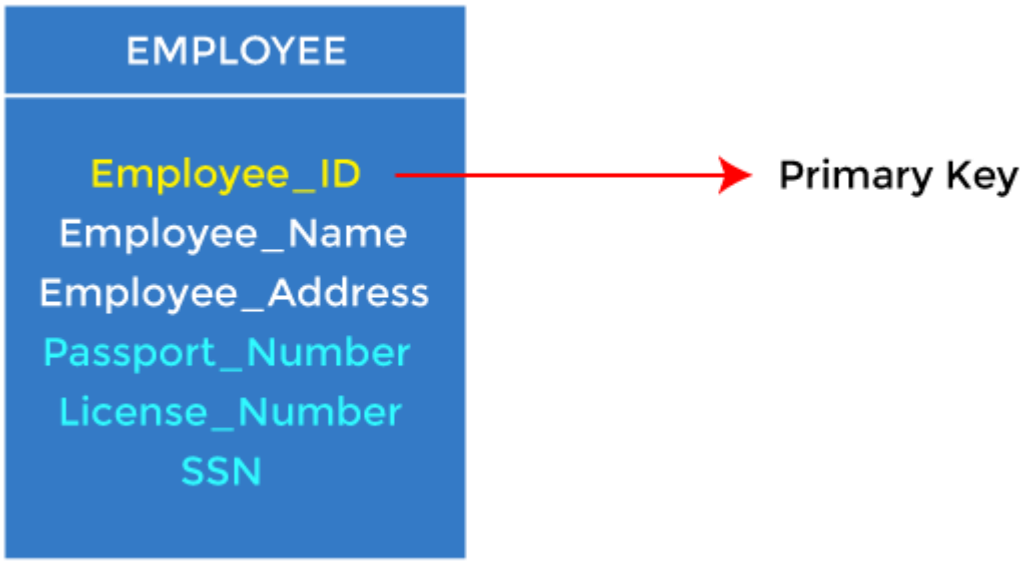


Types of keys:



1. Primary key

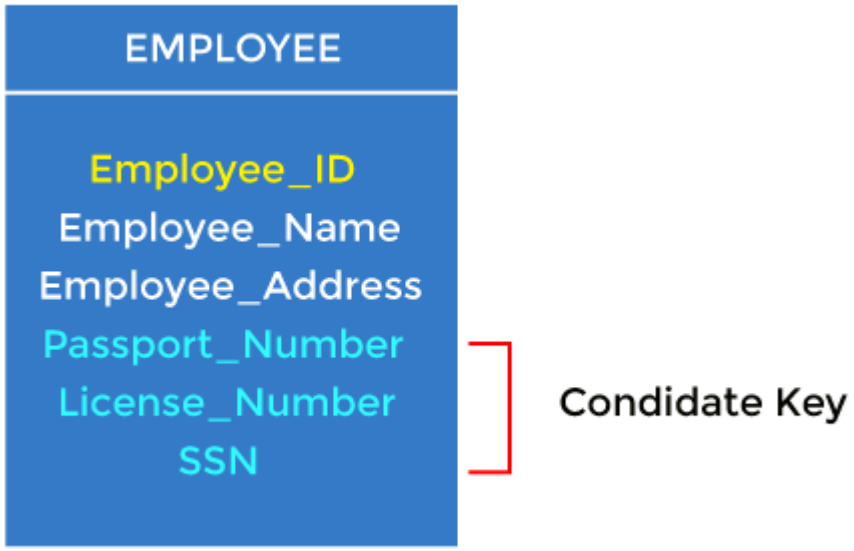
- It is the first key used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys, as we saw in the PERSON table. The key which is most suitable from those lists becomes a primary key.
- In the EMPLOYEE table, ID can be the primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License_Number and Passport_Number as primary keys since they are also unique.
- For each entity, the primary key selection is based on requirements and developers.



2. Candidate key

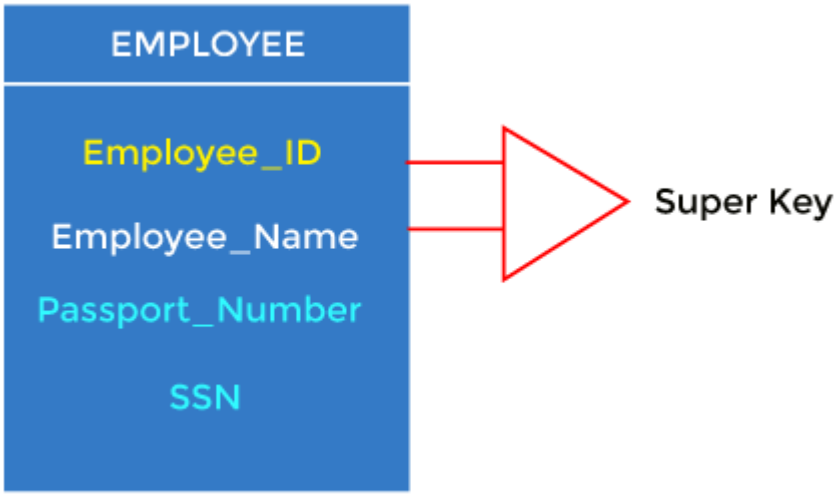
- A candidate key is an attribute or set of attributes that can uniquely identify a tuple.
- Except for the primary key, the remaining attributes are considered a candidate key. The candidate keys are as strong as the primary key.

For example: In the EMPLOYEE table, id is best suited for the primary key. The rest of the attributes, like SSN, Passport_Number, License_Number, etc., are considered a candidate key.



3. Super Key

Super key is an attribute set that can uniquely identify a tuple. A super key is a superset of a candidate key.

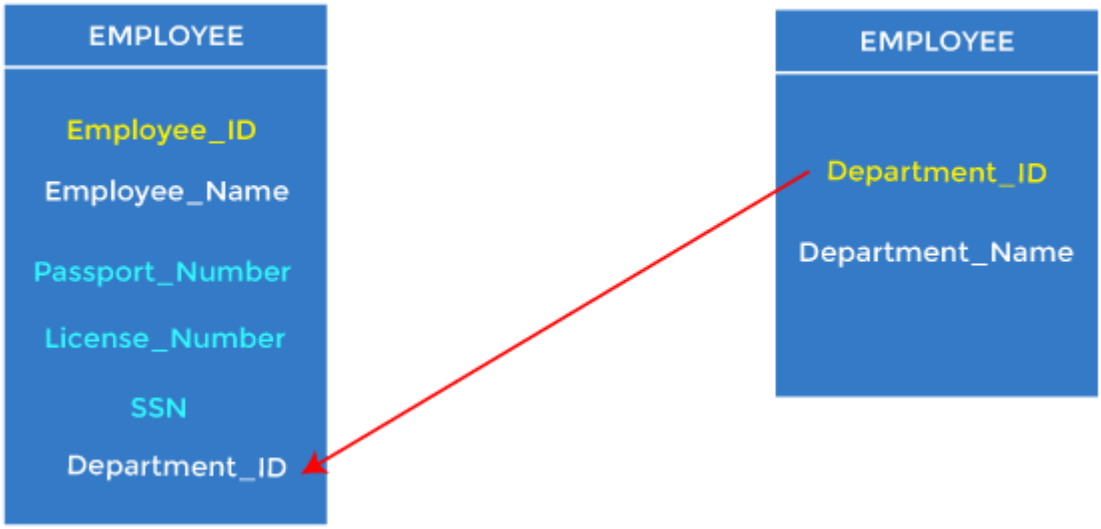


For example: In the above EMPLOYEE table, for(EMPLOEE_ID, EMPLOYEE_NAME), the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key.

The super key would be EMPLOYEE-ID (EMPLOYEE_ID, EMPLOYEE-NAME), etc.

4. Foreign key

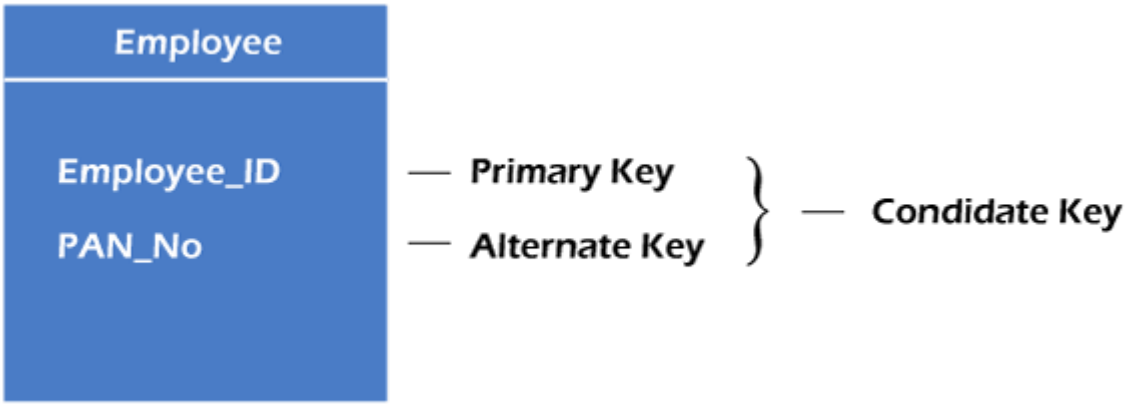
- Foreign keys are the column of the table used to point to the primary key of another table.
- Every employee works in a specific department in a company, and employee and department are two different entities. So we can't store the department's information in the employee table. That's why we link these two tables through the primary key of one table.
- We add the primary key of the DEPARTMENT table, Department_Id, as a new attribute in the EMPLOYEE table.
- In the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.



5. Alternate key

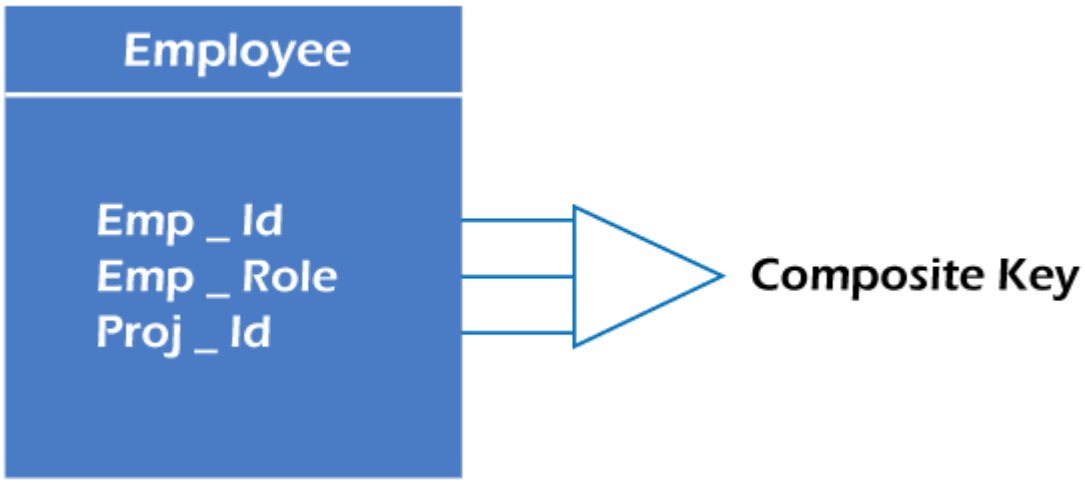
There may be one or more attributes or a combination of attributes that uniquely identify each tuple in a relation. These attributes or combinations of the attributes are called the candidate keys. One key is chosen as the primary key from these candidate keys, and the remaining candidate key, if it exists, is termed the alternate key. **In other words,** the total number of the alternate keys is the total number of candidate keys minus the primary key. The alternate key may or may not exist. If there is only one candidate key in a relation, it does not have an alternate key.

For example, employee relation has two attributes, Employee_Id and PAN_No, that act as candidate keys. In this relation, Employee_Id is chosen as the primary key, so the other candidate key, PAN_No, acts as the Alternate key.

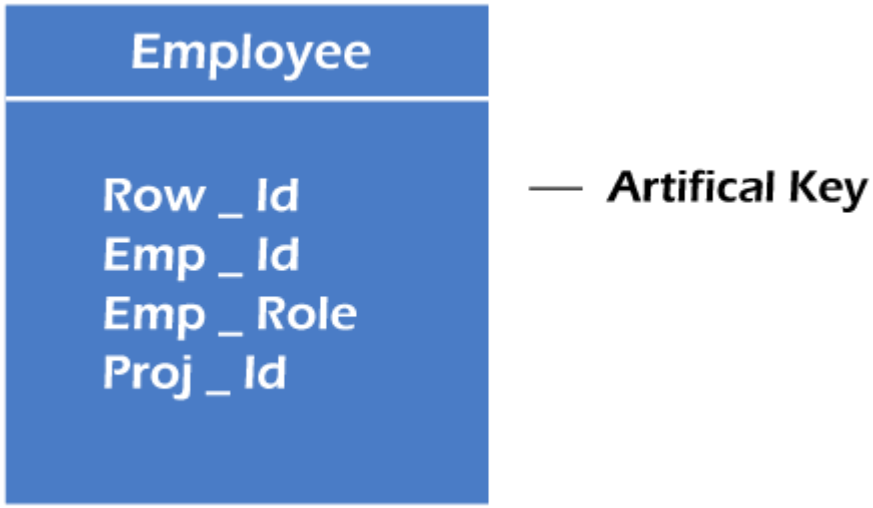


6. Composite key

Whenever a primary key consists of more than one attribute, it is known as a composite key. This key is also known as Concatenated Key.



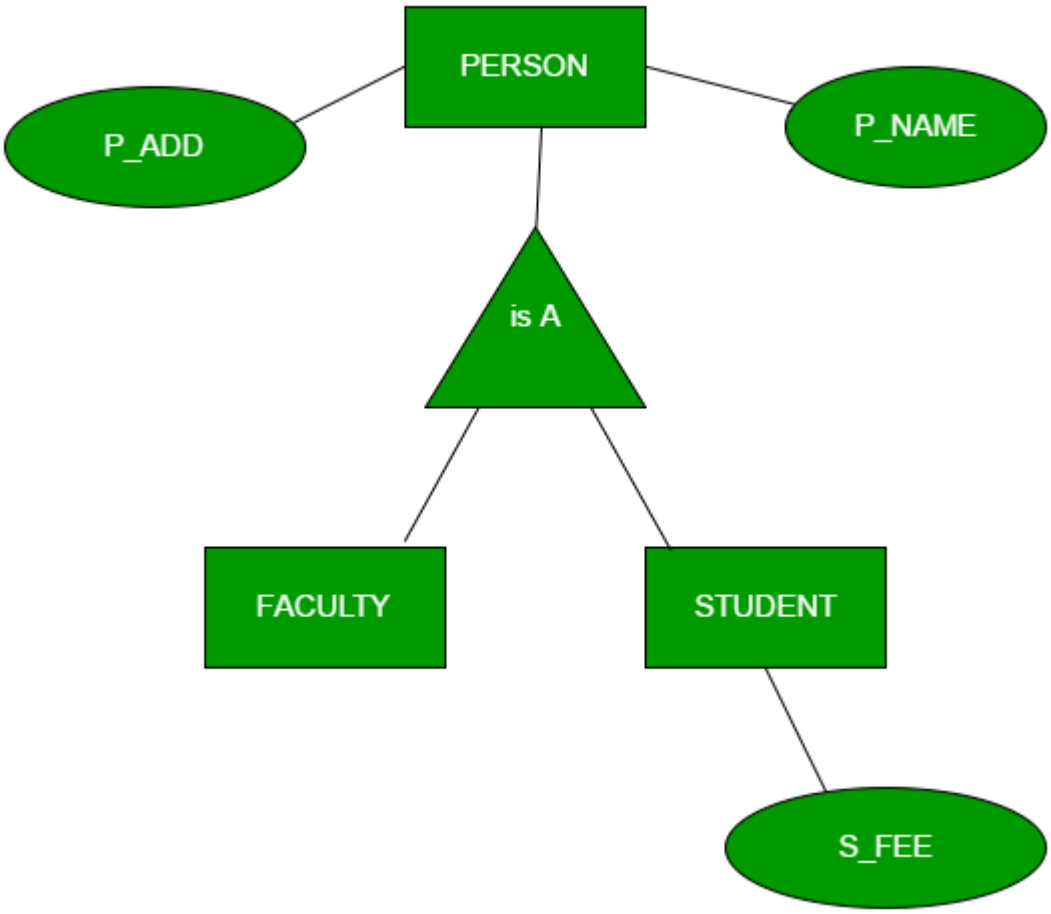
For example, in employee relations, we assume that an employee may be assigned multiple roles, and an employee may work on multiple projects simultaneously. So the primary key will be composed of all three attributes, namely Emp_ID, Emp_role, and Proj_ID in combination. So these attributes act as a composite key since the primary key comprises more than one attribute.



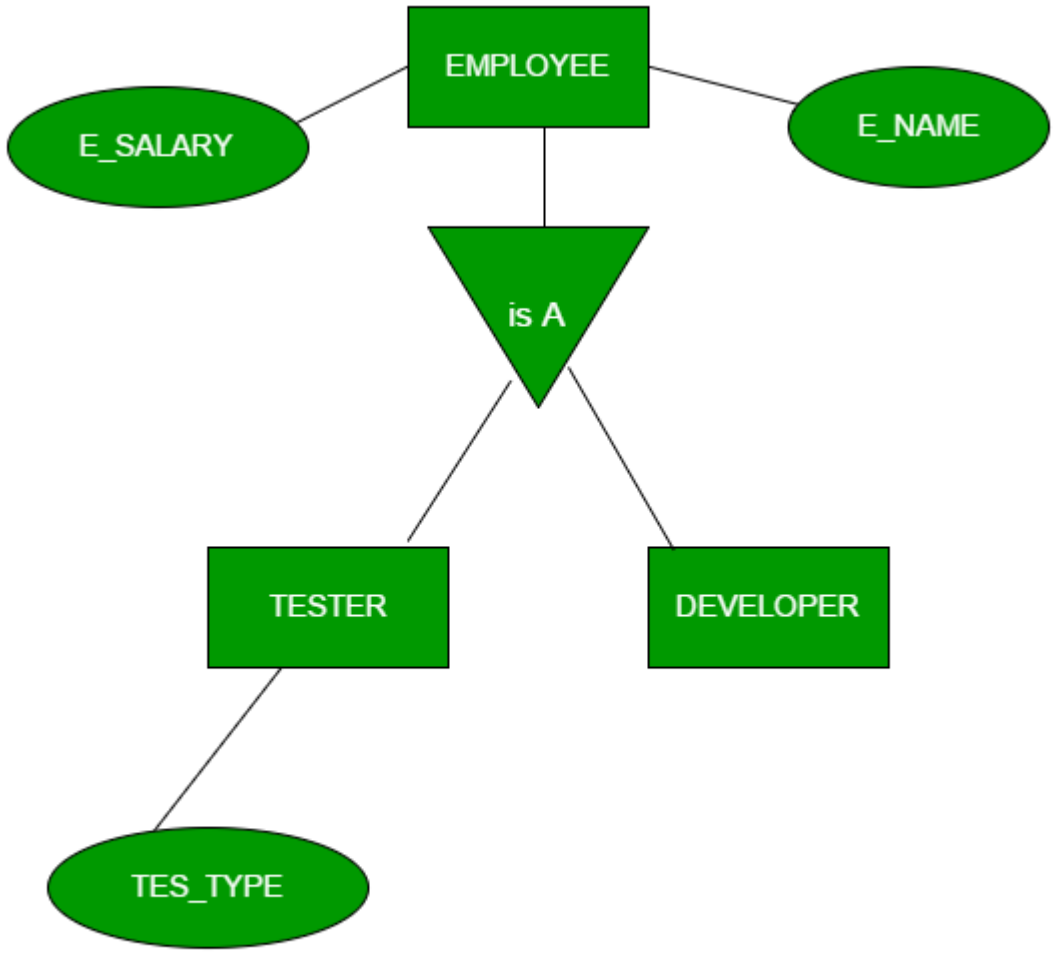
7. Artificial key

The key created using arbitrarily assigned data are known as artificial keys. These keys are created when a primary key is large and complex and has no relationship with many other relations. The data values of the artificial keys are usually numbered in a serial order.

Generalization –
Generalization is the process of extracting common properties from a set of entities and create a generalized entity from it. It is a bottom-up approach in which two or more entities can be generalized to a higher level entity if they have some attributes in common. For Example, STUDENT and FACULTY can be generalized to a higher level entity called PERSON as shown in Figure 1. In this case, common attributes like P_NAME, P_ADD become part of higher entity (PERSON) and specialized attributes like S_FEE become part of specialized entity (STUDENT).

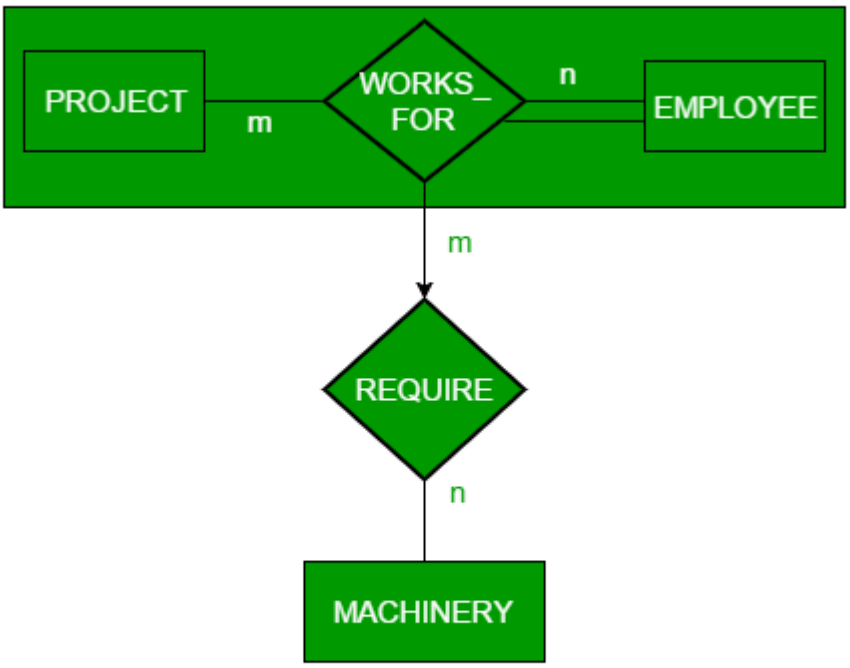


Specialization –
In specialization, an entity is divided into sub-entities based on their characteristics. It is a top-down approach where higher level entity is specialized into two or more lower level entities. For Example, EMPLOYEE entity in an Employee management system can be specialized into DEVELOPER, TESTER etc. as shown in Figure 2. In this case, common attributes like E_NAME, E_SAL etc. become part of higher entity (EMPLOYEE) and specialized attributes like TES_TYPE become part of specialized entity (TESTER).



Specialization

Aggregation –
An ER diagram is not capable of representing relationship between an entity and a relationship which may be required in some scenarios. In those cases, a relationship with its corresponding entities is aggregated into a higher level entity. Aggregation is an abstraction through which we can represent relationships as higher level entity sets.
For Example, Employee working for a project may require some machinery. So, REQUIRE relationship is needed between relationship WORKS_FOR and entity MACHINERY. Using aggregation, WORKS_FOR relationship with its entities EMPLOYEE and PROJECT is aggregated into single entity and relationship REQUIRE is created between aggregated entity and MACHINERY.



Aggregation

Conversion of ER diagrams to tables in DBMS

Conversion of ER diagrams to tables

Follow the steps given below for the conversion of the ER diagrams to tables in the database management system (DBMS) −

Step 1 − Conversion of strong entities

- For each strong entity create a separate table with the same name.
- Includes all attributes, if there is any composite attribute divided into simple attributes and has to be included.
- Ignore multivalued attributes at this stage.
- Select the p key for the table.

Step 2 − Conversion of weak entity

- For each weak entity create a separate table with the same name.
- Include all attributes.
- Include the P key of a strong entity as foreign key is the weak entity.
- Declare the combination of foreign key and decimator attribute as P key from the weak entity.

Step 3 − Conversion of one-to-one relationship

- For each one to one relation, say A and B modify either A side or B side to include the P key of the other side as a foreign key.
- If A or B is having total participation, then that should be a modified table.

- If a relationship consists of attributes, include them also in the modified table.

Step 4 − Conversion of one-to-many relationship

- For each one to many relationships, modify the M side to include the P key of one side as a foreign key.
- If relationships consist of attributes, include them as well.

Step 5 − Conversion of many-many relationship

- For each many-many relationship, create a separate table including the P key of M side and N side as foreign keys in the new table.
- Declare the combination of foreign keys as P for the new table.
- If relationships consist of attributes, include them also in the new table.

Step 6 − Conversion of multivalued attributes

- For each multivalued attribute create a separate table and include the P key of the present table as foreign key.
- Declare the combination of foreign key and multivalued attribute as P keys.

Step 7 − Conversion of n-ary relationship

- For each n-ary relationship create a separate table and include the P key of all entities as foreign key.
- Declare the combination of foreign keys as P key.