UNIT- 1

Introduction to DBMS

- Introduction to Database Management Systems,
- Purpose of Database Systems,
- Database-System Applications
- Data Abstraction and Database System Structure.

DBMS Environment

- Hardware
  - Client-server architecture
- Software
  - dbms, os, network, application
- Data
  - Schema, subschema, table, attribute
- People
  - Data administrator & database administrator
  - Database designer: logical & physical
  - Application programmer
  - End-user: naive & sophisticated
- Procedure
  - Start, stop, log on, log off, back up, recovery

History of Database Systems

- First generation
  - Hierarchical model
    - Information Management System (IMS)
  - Network model
    - Conference on Data System Languages (CODASYL)
    - Data Base Task Group (DBTG)
  - Limitation
    - Complex program for simple query
    - Minimum data independence
    - No theoretical foundation
- Second generation
  - Relational model
    - E. R. Codd
    - DB2, Oracle
  - Limitation
    - Limited data modeling
- Third generation
  - Object-relational DBMS
  - Object-oriented DBMS

DBMS

A database management system (DBMS) refers to the technology for creating and managing databases. DBMS is a software tool to organize (create, retrieve, update, and manage) data in a database.

WHY USE DBMS?

- To develop software applications In less time.
- Data independence and efficient use of data.
- For uniform data administration.
- For data integrity and security.
- For concurrent access to data, and data recovery from crashes.
- To use user-friendly declarative query language.

Where is a Database Management System (DBMS) being Used?

- **Airlines**: reservations, schedules, etc
- **Telecom**: calls made, customer details, network usage, etc
- **Universities**: registration, results, grades, etc
- **Sales**: products, purchases, customers, etc
- **Banking**: all transactions etc

Advantage of DBMS

A DBMS manages data and has many benefits. These are:

- **Data independence:** Application programs should be as free or independent as possible from details of data representation and storage. DBMS can supply an abstract view of the data for insulating application code from such facts.
- **Efficient data access:** DBMS utilizes a mixture of sophisticated concepts and techniques for storing and retrieving data competently. This feature becomes important in cases where the data is stored on external storage devices.
- **Data integrity and security:** If data is accessed through the DBMS, the DBMS can enforce integrity constraints on the data.
- **Data administration:** When several users share the data, integrating the administration of data can offer significant improvements. Experienced professionals understand the nature of the data being managed and can be responsible for organizing the data representation to reduce redundancy and make the data to retrieve efficiently.

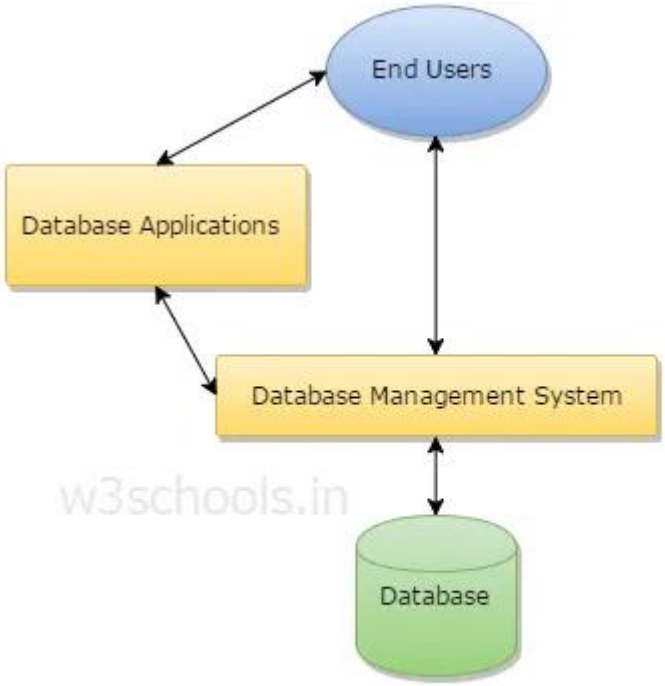
Components of DBMS

Users: Users may be of any kind such as DB administrator, System developer, or database users.

Database application: Database application may be Departmental, Personal, organization's and / or Internal.

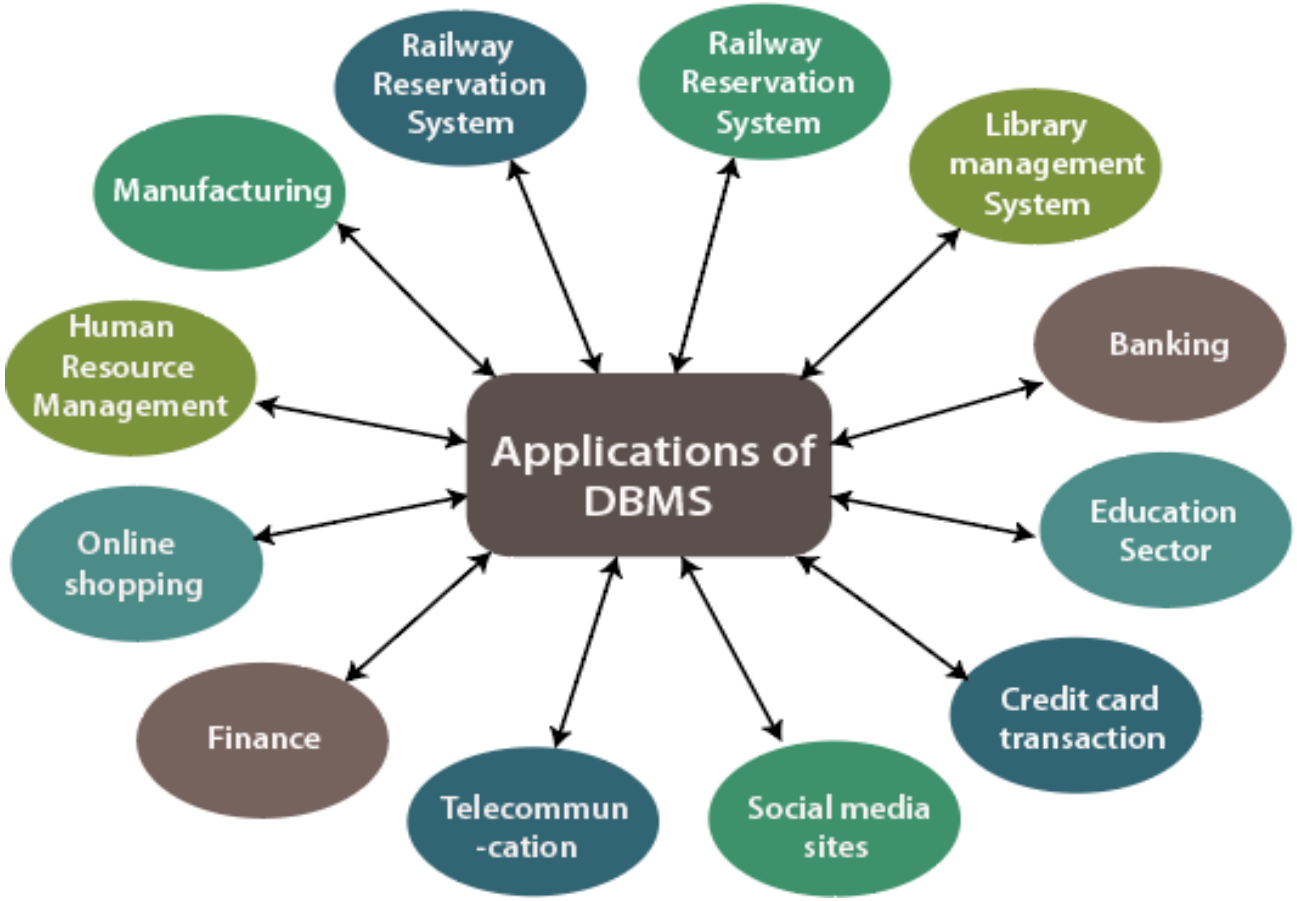
DBMS: Software that allows users to create and manipulate database access,

Database: Collection of logical data as a single unit.



Components of a Database Management System

Applications of DBMS: There are various fields where a database management system is used. Following are some applications which make use of the database management system:



Data Abstraction in DBMS

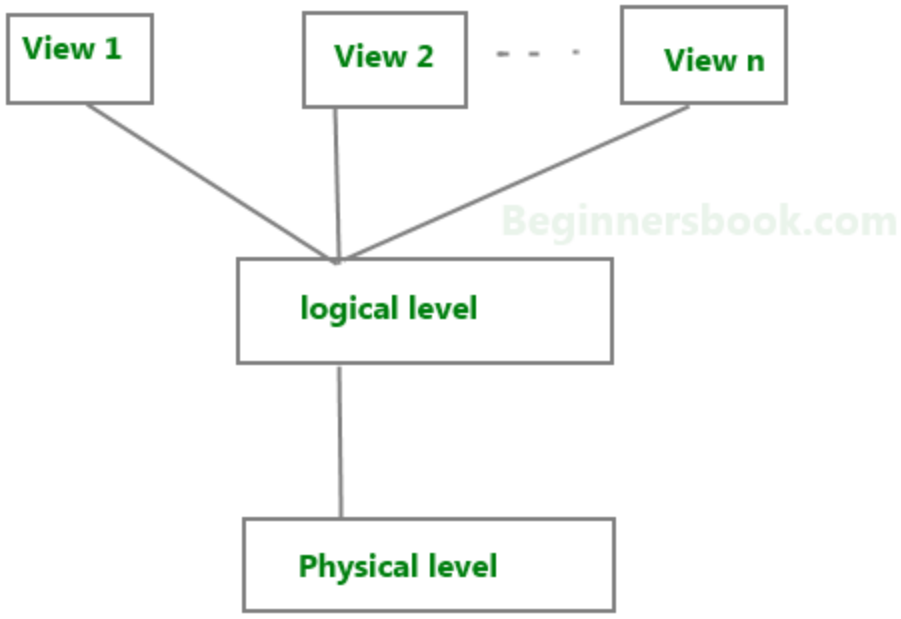
- Database systems are made-up of complex data structures. To ease the user interaction with database, the developers hide internal irrelevant details from users. This process of hiding irrelevant details from user is called data abstraction.

We have three levels of abstraction:

Physical level: This is the lowest level of data abstraction. It describes how data is actually stored in database. You can get the complex data structure details at this level.

Logical level: This is the middle level of 3-level data abstraction architecture. It describes what data is stored in database.

View level: Highest level of data abstraction. This level describes the user interaction with database system.



Three Levels of data abstraction

Example:
Let's say we are storing customer information in a customer table. At **physical level** these records can be described as blocks of storage (bytes, gigabytes, terabytes etc.) in memory. These details are often hidden from the programmers.

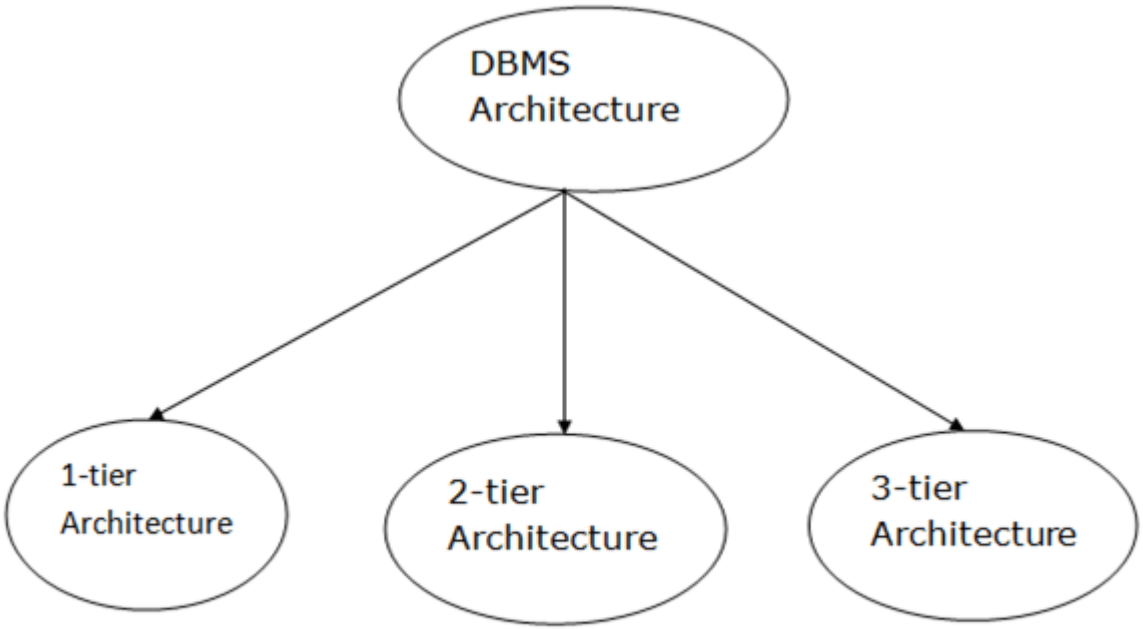
At the **logical level** these records can be described as fields and attributes along with their data types, their relationship among each other can be logically implemented. The programmers generally work at this level because they are aware of such things about database systems.

At view level, user just interact with system with the help of GUI and enter the details at the screen, they are not aware of how the data is stored and what data is stored; such details are hidden from them.

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



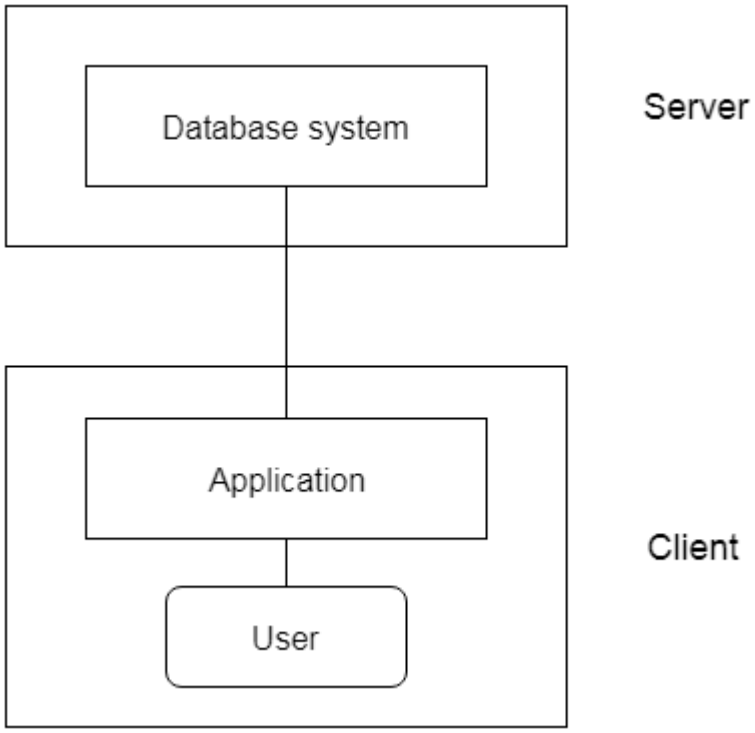
Database architecture can be seen as a single tier or multi-tier. But logically, database architecture is of two types like: **2-tier architecture** and **3-tier architecture**.

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.

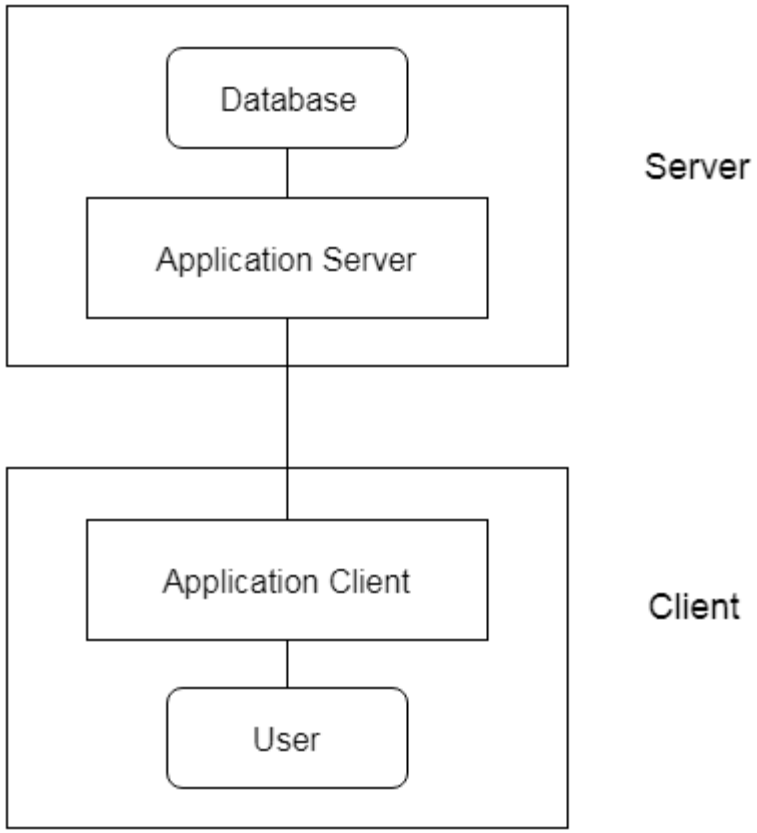
2-Tier Architecture

- 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.



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.



- Structure of relational databases,
- Domains, Relations,
- Relational algebra – fundamental operators and syntax,
- relational algebra queries,
- tuple relational calculus.
- Database Design and ER model

Structure of relational databases

- **A *relational database*** is a type of database. It uses a structure that allows us to identify and access data *in relation* to another piece of data in the database. Often, data in a relational database is organized into tables.
- **Tables: Rows and Columns**
- Tables can have hundreds, thousands, sometimes even millions of rows of data. These rows are often called *records*.
- Tables can also have many *columns* of data. Columns are labelled with a descriptive name (say, age for example) and have a specific *data type*.
- For example, a column called age may have a type of INTEGER (denoting the type of data it is meant to hold).

name	age	country
Natalia	11	Iceland
Ned	6	New York
Zenas	14	Ireland
Laura	8	Kenya

What is a Relational Database Management System (RDBMS)?

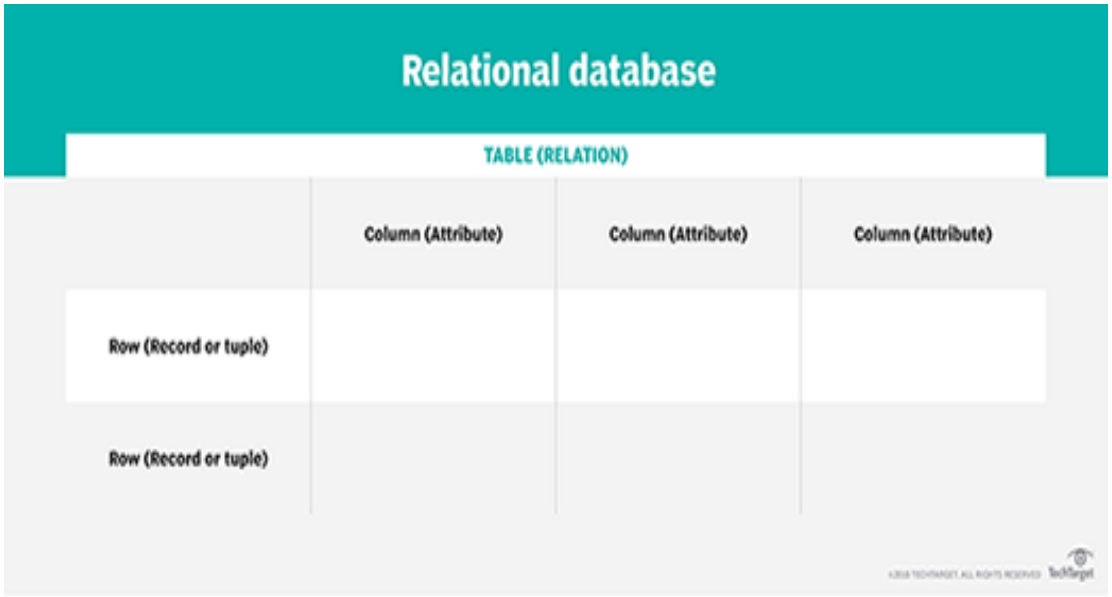
- A relational database management system (RDBMS) is a program that allows you to create, update, and administer a relational database. Most relational database management systems use the SQL language to access the database.

- **How does a relational database work?**

- The data tables used in a relational database store information about related objects. Each row holds a record with a unique identifier -- known as a key -- and each column contains the attributes of the data. Each record assigns a value to each feature, making relationships between data points easy to identify.
- The standard user and application program interface (API) of a relational database is the Structured Query Language. SQL code statements are used both for interactive queries for information from a relational database and for gathering data for reports. Defined data integrity rules must be followed to ensure the relational database is accurate and accessible.

- **What is the structure of a relational database model?**

Each table, sometimes called a *relation,* in a relational database contains one or more data categories in columns or *attributes*. Each row, also called a *record* or *tuple*, contains a unique instance of data -- or *key* -- for the categories defined by the columns. Each table has a unique primary key that identifies the information in a table. The relationship between tables can be set via the use of foreign keys -- a field in a table that links to the primary key of another table.



- For example, a typical business order entry database would include a table that describes a customer with columns for name, address, phone number and so forth. Another table would describe an order, including information like the product, customer, date and sales price.
- A user can get a database report showing the data they need. For example, a branch office manager might want a report on all customers that bought products after a certain date. A financial services manager in the same company could, from the same tables, obtain a report on accounts that need to be paid.
- When creating a relational database, users define the domain of possible values in a data column and constraints that may apply to that data value. For example, a domain of possible customers could allow up to 10 possible customer names, but it is limited in one table to allowing only three of these customer names to be specifiable.
- Two constraints relate to data integrity and the primary and foreign keys:
- **Entity integrity** ensures that the primary key in a table is unique and the value is not set to null.
- **Referential integrity** requires that every value in a foreign key column will be found in the primary key of the table from which it originated.

What are the advantages of relational databases?

- **Categorizing data**. Database administrators can easily categorize and store data in a relational database that can then be queried and filtered to extract information for reports. Relational databases are also easy to extend and aren't reliant on physical organization. After the original database creation, a new data category can be added without having to modify the existing applications.
- **Accuracy.** Data is stored just once, eliminating data duplication in storage procedures.
- **Ease of use.** Complex queries are easy for users to carry out with SQL, the main query language used with relational databases.
- **Collaboration.** Multiple users can access the same database.
- **Security.** Direct access to data in tables within an RDBMS can be limited to specific users.

Domains, Relations,

- Each *column of a relational table* is defined on a domain. A ***domain*** is a set of values. A *datatype*, which is also defined for each column, consists of a domain of values and a set of operations on those values.
- A **domain** *D* is a set of atomic values. By **atomic** we mean that each value in the domain is indivisible as far as the formal relational model is concerned. A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn. It is also useful to specify a name for the domain, to help in interpreting its values. Some examples of domains follow:
  - phone_numbers. The set of ten-digit phone numbers.
  - Names: The set of character strings that represent names of persons.
- The preceding are called *logical* definitions of domains. A **data type** or **format** is also specified for each domain. For example, the data type for the domain
- Usa_phone_numbers can be declared as a character string of the form (*ddd*)*ddddddd*, where each *d* is a numeric (decimal) digit and the first three digits form a valid telephone area code.

- **Attribute:** It contains the name of a column in a particular table. Each attribute Ai must have a domain, dom(Ai).
- Each **attribute** *Ai* is the name of a role played by some domain *D* in the relation schema *R*.

  STUDENT(Name, Ssn, Home_phone, Address, Office_phone, Age, Gpa)
- Using the data type of each attribute, the definition is sometimes written as:

  STUDENT(Name: string, Ssn: string, Home_phone: string, Address: string, Office_phone: string, Age: integer, Gpa: real)

- **Tables** − In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represents records and columns represent the attributes.
- **Tuple** − A single row of a table, which contains a single record for that relation is called a tuple.
- **Relation instance** − A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.
- **Relation schema** − A relation schema describes the relation name (table name), attributes, and their names.
- **Relation key** − Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.
- **Attribute domain** − Every attribute has some pre-defined value scope, known as attribute domain.

Relational algebra – fundamental operators and syntax

- The relational algebra consists of a set of operations that take one or two relations as input and produce a new relation as their result. The relational calculus uses predicate logic to define the result desired without giving any specific algebraic procedure for obtaining that result.
- The relational algebra defines a set of operations on relations, paralleling the usual algebraic operations such as addition, subtraction or multiplication, which operate on numbers. Just as algebraic operations on numbers take one or more numbers as input and return a number as output, the relational algebra operations typically take one or two relations as input and return a relation as output.

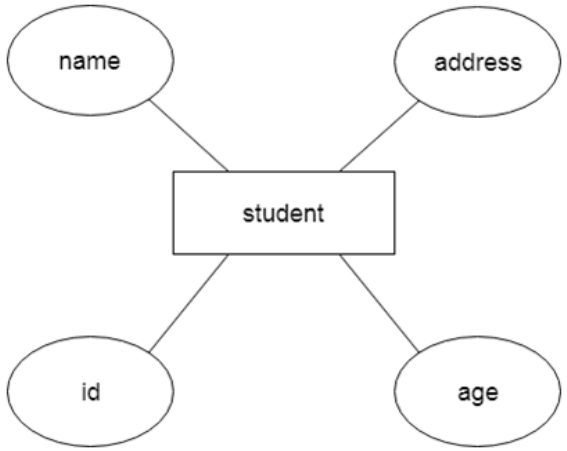
Symbol (Name)	Example of Use
σ	$\sigma_{\text{salary}>=85000}(instructor)$
(Selection)	Return rows of the input relation that satisfy the predicate.
Π	$\Pi_{ID,salary}(instructor)$
(Projection)	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
$\bowtie$	$instructor \bowtie department$
(Natural join)	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
$\times$	$instructor \times department$
(Cartesian product)	Output all pairs of rows from the two input relations (regardless of whether or not they have the same values on common attributes)
$\cup$	$\Pi_{name}(instructor) \cup \Pi_{name}(student)$
(Union)	Output the union of tuples from the two input relations.

What is Database Design?

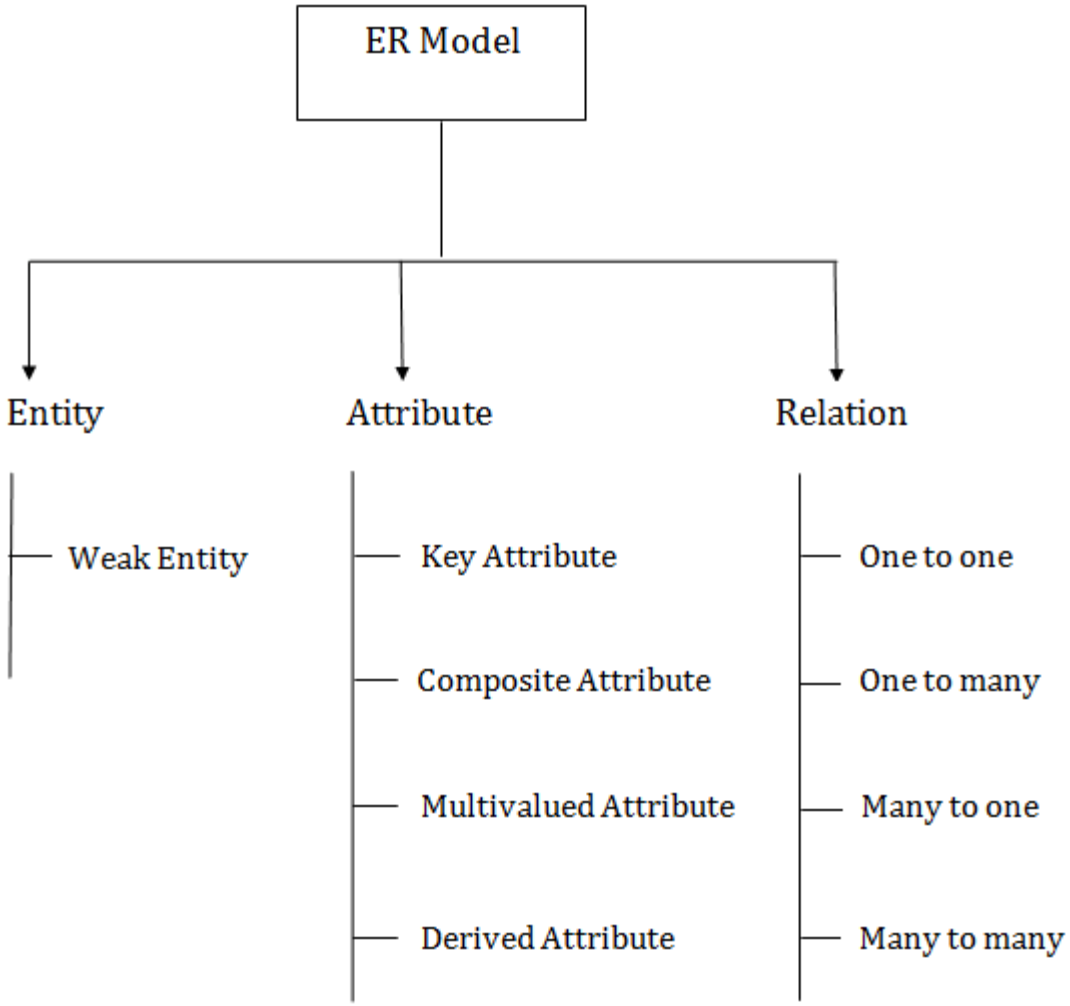
- **Database Design** is a collection of processes that facilitate the designing, development, implementation and maintenance of enterprise data management systems. Properly designed database are easy to maintain, improves data consistency and are cost effective in terms of disk storage space. The database designer decides how the data elements correlate and what data must be stored.
- The main objectives of database design in DBMS are to produce logical and physical designs models of the proposed database system.
- The logical model concentrates on the data requirements and the data to be stored independent of physical considerations. It does not concern itself with how the data will be stored or where it will be stored physically.
- The physical data design model involves translating the logical DB design of the database onto physical media using hardware resources and software systems such as database management systems (DBMS).

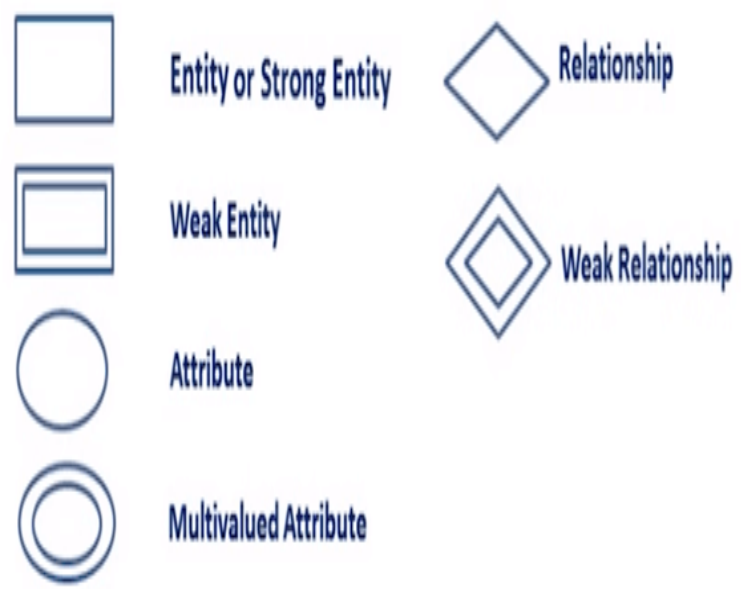
ER model

- ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
- In ER modelling, the database structure is portrayed as a diagram called an entity-relationship diagram.
- **For example,** Suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc and there will be a relationship between them.



- Component of ER Diagram





- **Entity Relationship Diagram Symbols & Notations** mainly contains three basic symbols which are rectangle, oval and diamond to represent relationships between elements, entities and attributes. There are some sub-elements which are based on main elements in ERD Diagram. ER Diagram is a visual representation of data that describes how data is related to each other using different ERD Symbols and Notations.

  Following are the main components and its symbols in ER Diagrams:

  Rectangles: This Entity Relationship Diagram symbol represents entity types

  Ellipses : Symbol represent attributes

  Diamonds: This symbol represents relationship types
- **Lines:** It links attributes to entity types and entity types with other relationship types
- **Primary key:** attributes are underlined
- **Double Ellipses:** Represent multi-valued attributes

Entity Name

Entity

Person,place,object,event or concept about which data is to be maintained

Example: Car, Student

Attribute

Property or characteristic of an entity

Example: Color of car Entity Name of Student Entity

Relation

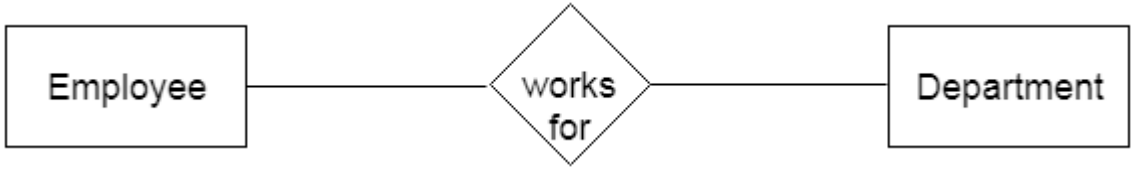
Verb Phrase

Association between the instances of one or more entity types

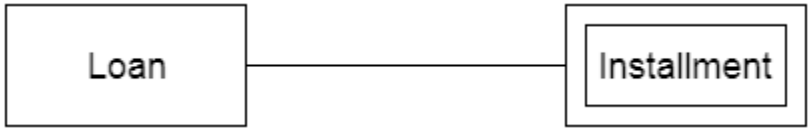
Example: Blue Car Belongs to Student Jack

Entity:

- An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.
- Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity.

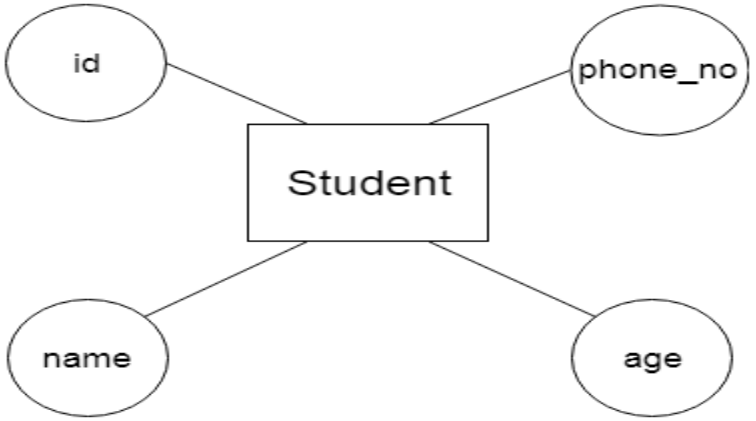


An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle



Attribute:

- The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.
- **For example,** id, age, contact number, name, etc. can be attributes of a student.



a. Key Attribute

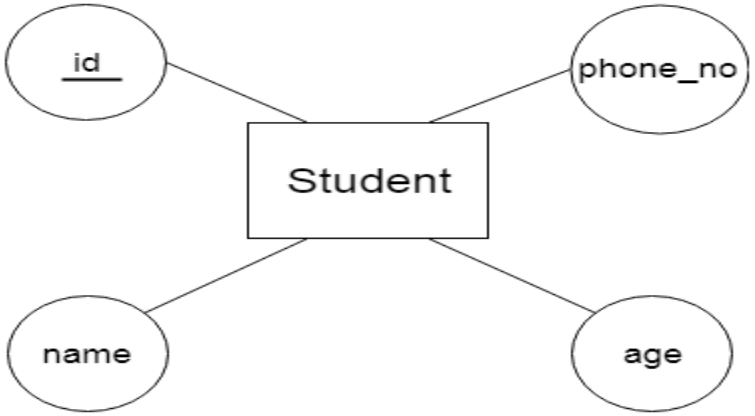
b. Composite Attribute

c. Multivalued Attribute

d. Derived Attribute

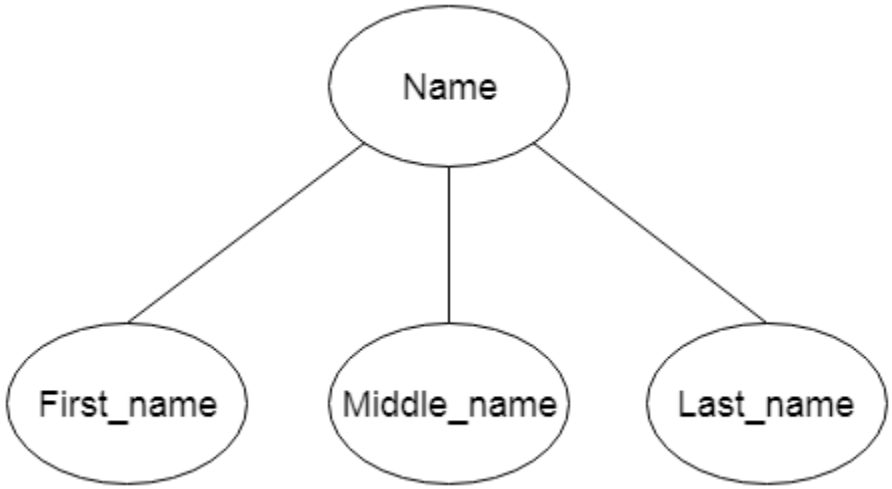
a. Key Attribute

- The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.



b. Composite Attribute

- An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.



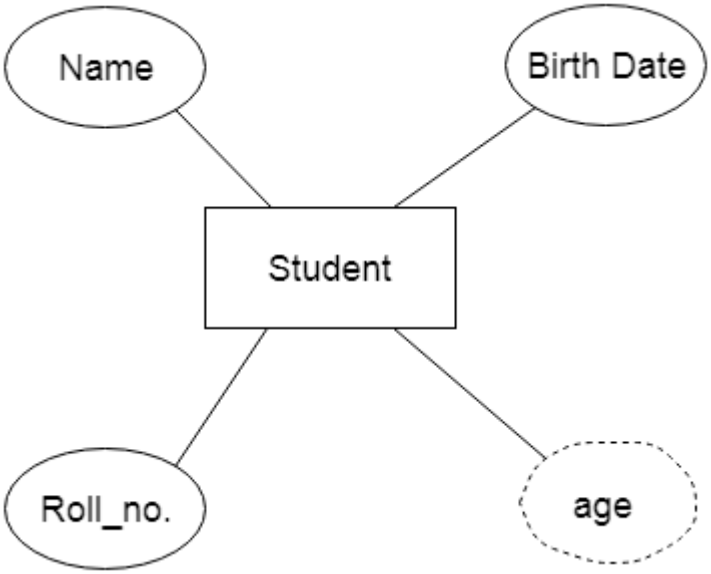
c. Multivalued Attribute

An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.

For example, a student can have more than one phone number.

D. Derived Attribute

- An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.
- **For example,** A person's age changes over time and can be derived from another attribute like Date of birth.



3. Relationship

- A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.

a. One-to-One Relationship

b. One-to-many relationship

c. Many-to-one relationship

d. Many-to-many relationship

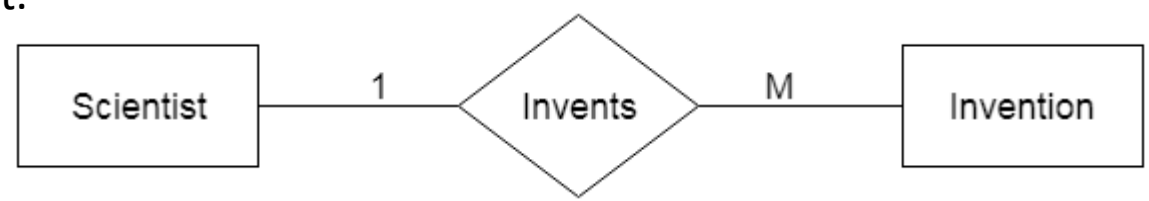
a. One-to-One Relationship

- When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.
- For example, a person has only one passport and a passport is given to one person.

b. One-to-many relationship

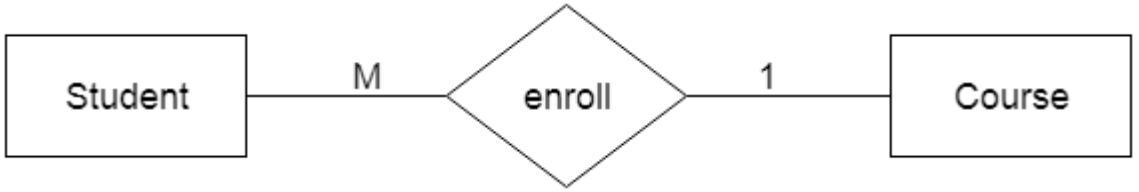
When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a one-to-many relationship.

For example, Scientist can invent many inventions, but the invention is done by the only specific scientist.



c. Many-to-one relationship

- When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.
- **For example,** Student enrolls for only one course, but a course can have many students.

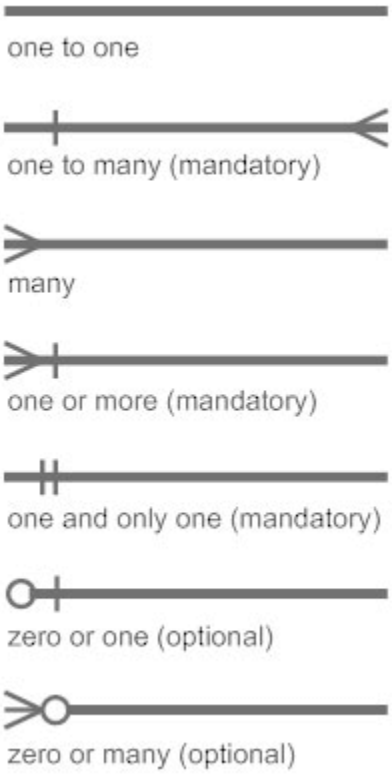


d. Many-to-many relationship

When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.

For example, Employee can assign by many projects and project can have many employees.

Notation of ER diagram

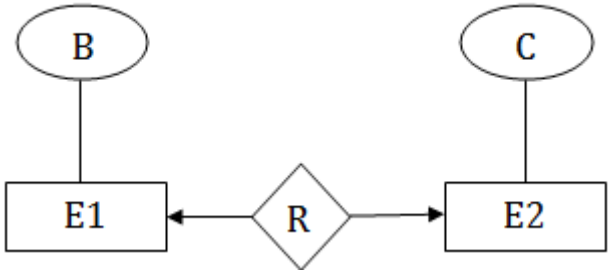


Mapping Constraints

- A mapping constraint is a data constraint that expresses the number of entities to which another entity can be related via a relationship set.
- It is most useful in describing the relationship sets that involve more than two entity sets.
- For binary relationship set R on an entity set A and B, there are four possible mapping cardinalities. These are as follows:
  - One to one (1:1)
  - One to many (1:M)
  - Many to one (M:1)
  - Many to many (M:M)

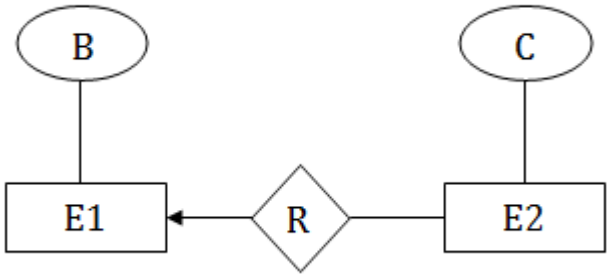
One-to-one

- In one-to-one mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with at most one entity in E1.



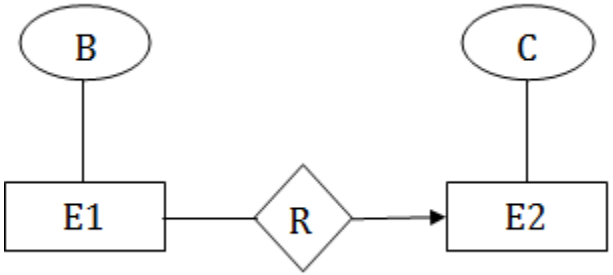
One-to-many

- In one-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with at most one entity in E1.



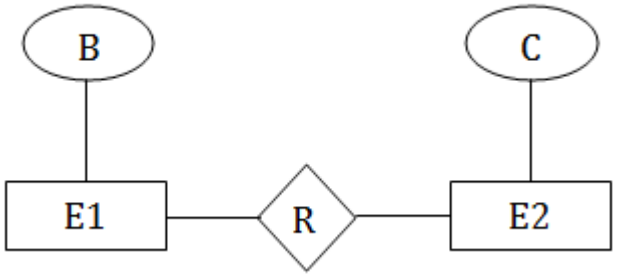
Many-to-one

- In one-to-many mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with any number of entities in E1.



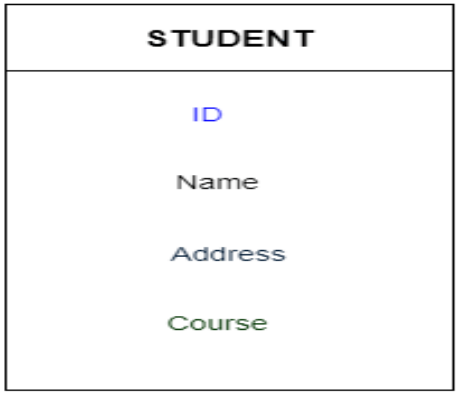
Many-to-many

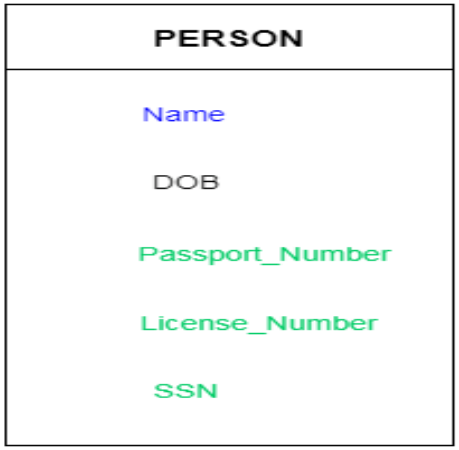
- In many-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with any number of entities in E1.



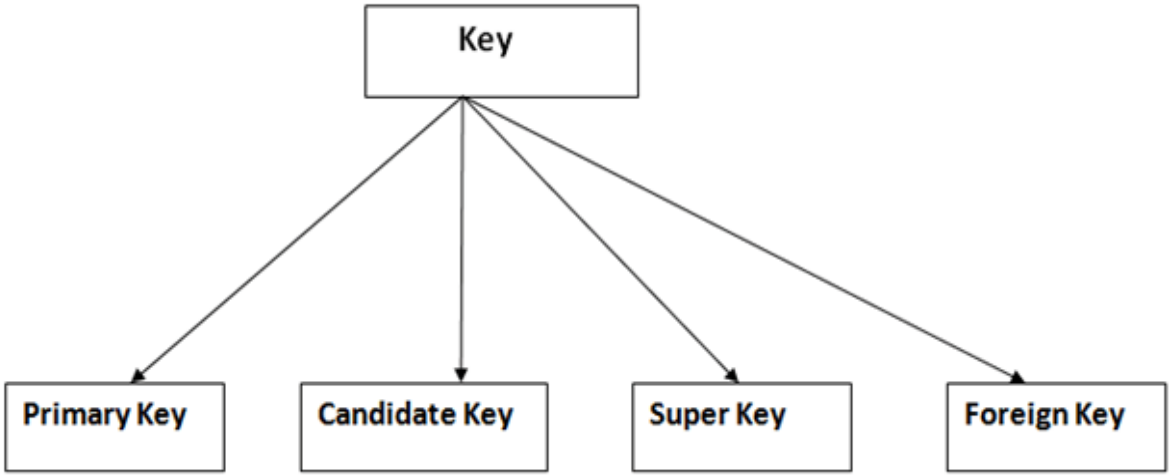
Keys

- 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:** In Student table, ID is used as a key because it is unique for each student. In PERSON table, passport_number, license_number, SSN are keys since they are unique for each person.



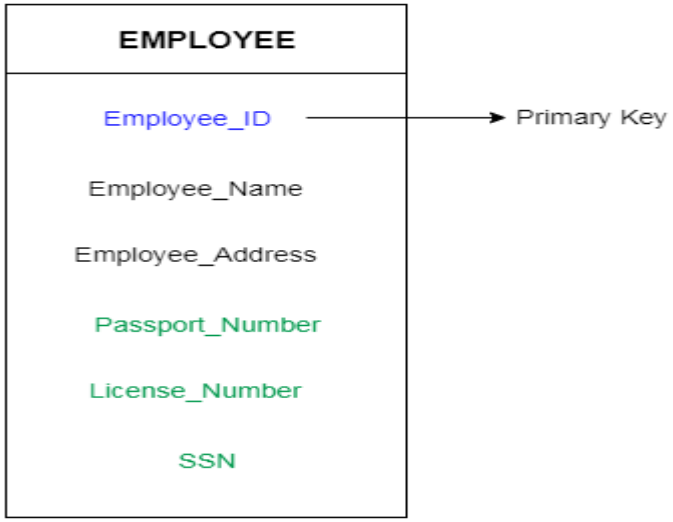


Types of key:



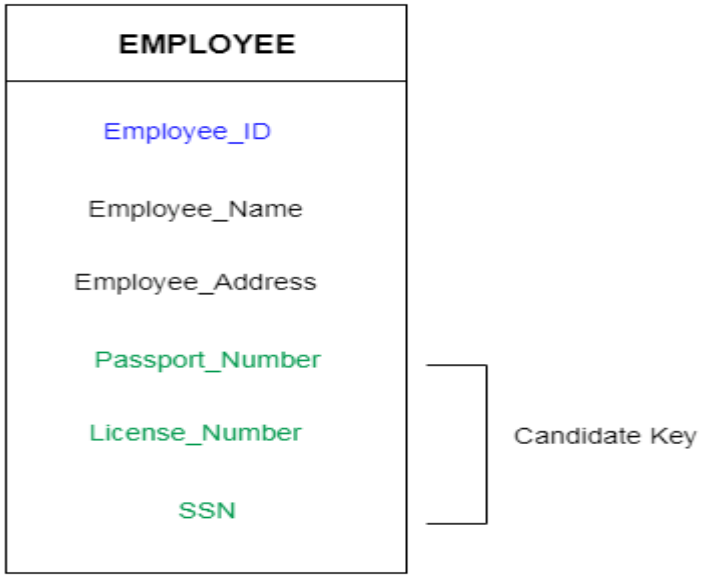
Primary key

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



Candidate key

- A candidate key is an attribute or set of an attribute which can uniquely identify a tuple.
- The remaining attributes except for primary key are considered as 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. Rest of the attributes like SSN, Passport_Number, and License_Number, etc. are considered as a candidate key.

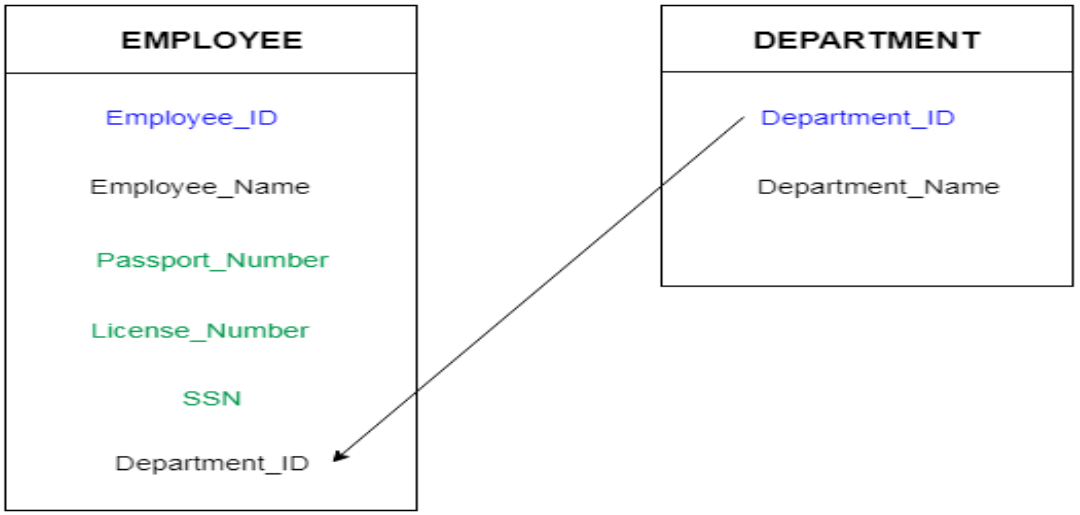


3. Super Key

- Super key is a set of an attribute which can uniquely identify a tuple. 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 which is used to point to the primary key of another table.
- In a company, every employee works in a specific department, and employee and department are two different entities. So we can't store the information of the department 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.
- Now in the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related



- **How to Create an Entity Relationship Diagram (ERD)**
- Now in this ERD Diagram Tutorial, we will learn how to create an ER Diagram. Following are the steps to create an ER Diagram:
- Steps to Create an ER Diagram Let's study them with an Entity Relationship Diagram Example: In a university, a Student enrolls in Courses. A student must be assigned to at least one or more Courses. Each course is taught by a single Professor. To maintain instruction quality, a Professor can deliver only one course

Step 1) Entity Identification

We have three entities

Student

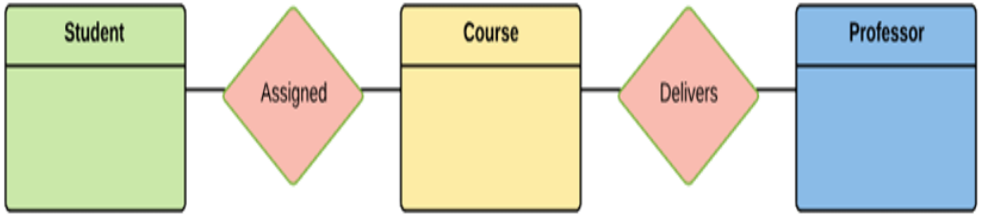
Course

Professor

Step 2) Relationship Identification

We have the following two relationship

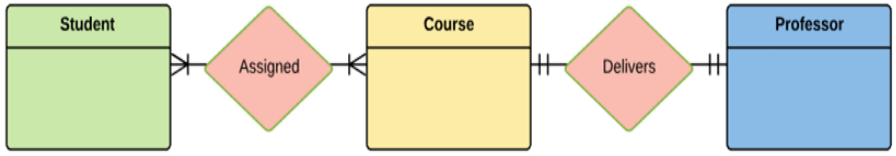
- The student is **assigned** a course
- Professor **delivers** a course



Step 3) Cardinality Identification

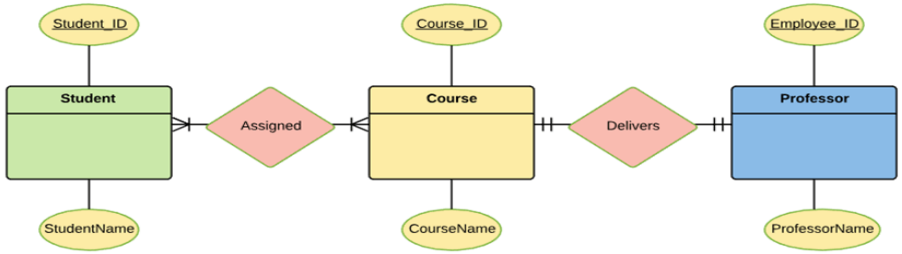
For them problem statement we know that,

- A student can be assigned **multiple** courses
- A Professor can deliver only **one** course



Step 4) Identify Attributes

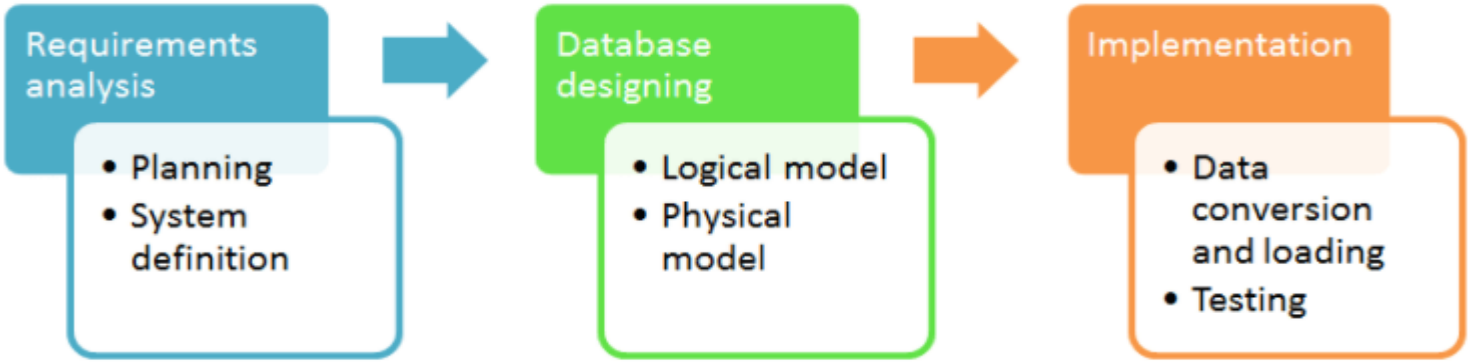
You need to study the files, forms, reports, data currently maintained by the organization to identify attributes



Step 5) Create the ERD Diagram

A more modern representation of Entity Relationship Diagram Example

Database Design Process



Requirement Analysis

First of all, the planning has to be done on what are the basic requirements of the project under which the design of the database has to be taken forward. Thus, they can be defined as:-

Planning - This stage is concerned with planning the entire DDLC (Database Development Life Cycle). The strategic considerations are taken into account before proceeding.

System definition - This stage covers the boundaries and scopes of the proper database after planning.

Database Designing

- The next step involves designing the database considering the user-based requirements and splitting them out into various models so that load or heavy dependencies on a single aspect are not imposed. Therefore, there has been some model-centric approach and that's where logical and physical models play a crucial role.
- **Physical Model** - The physical model is concerned with the practices and implementations of the logical model.
- **Logical Model** - This stage is primarily concerned with developing a model based on the proposed requirements. The entire model is designed on paper without any implementation or adopting DBMS considerations.

Implementation

- The last step covers the implementation methods and checking out the behaviour that matches our requirements. It is ensured with continuous integration testing of the database with different data sets and conversion of data into machine understandable language. The manipulation of data is primarily focused on these steps where queries are made to run and check if the application is designed satisfactorily or not.

- **Data conversion and loading** - This section is used to import and convert data from the old to the new system.
- **Testing** - This stage is concerned with error identification in the newly implemented system. Testing is a crucial step because it checks the database directly and compares the requirement specifications.

Database Design Process

- The process of designing a database carries various conceptual approaches that are needed to be kept in mind. An ideal and well-structured database design must be able to:
- Save disk space by eliminating redundant data.
- Maintains data integrity and accuracy.
- Provides data access in useful ways.
- Comparing Logical and Physical data models.

Logical

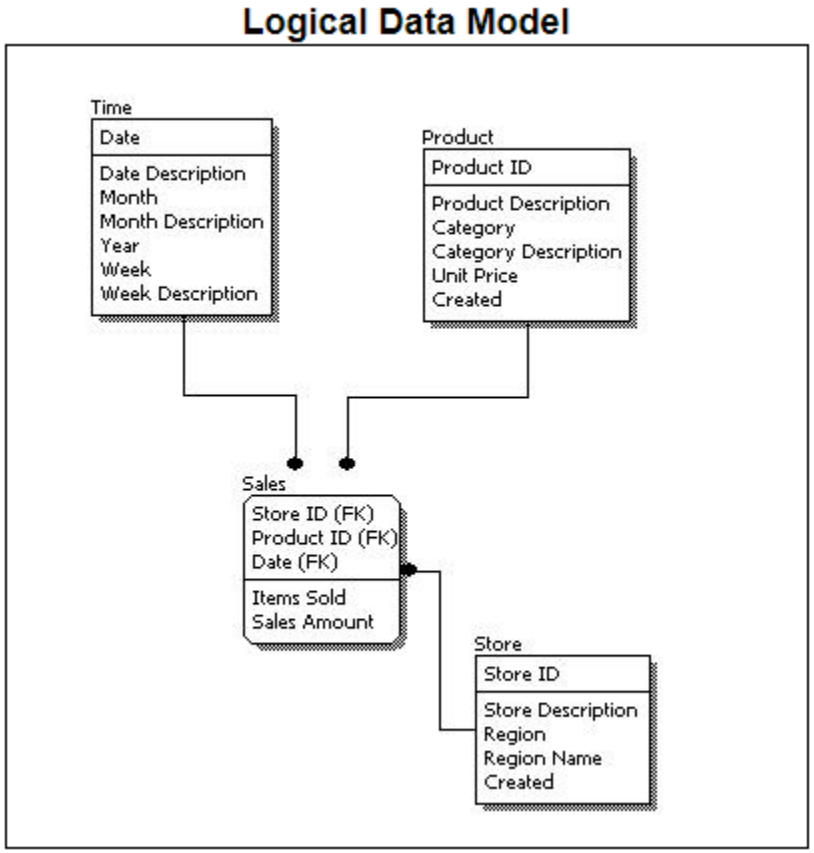
- A logical data model generally describes the data in as many details as possible, without having to be concerned about the physical implementations in the database. Features of logical data model might include:
- All the entities and relationships amongst them.
- Each entity has well-specified attributes.
- The primary key for each entity is specified.
- Foreign keys which are used to identify a relationship between different entities are specified.
- Normalization occurs at this level.

- A logical model can be designed using the following

1. Specify all the entities with primary keys.
2. Specify concurrent relationships between different entities.
3. Figure out each entity attributes
4. Resolve many-to-many relationships.
5. Carry out the process of normalization.

- Also, one important factor after following the above approach is to critically examine the design based on requirement gathering. If the above steps are strictly followed, there are chances of creating a highly efficient database design that follows the native approach.
- To understand these points, see the image below to get a clear picture.

If we compare the logical data model as shown in the figure above with some sample data in the diagram, we can come up with facts that in a conceptual data model there are no presence of a primary key whereas a logical data model has primary keys for all of its attributes. Also, logical data model the cover relationship between different entities and carries room for foreign keys to establish relationships among them.



Physical

A Physical data mode generally represents how the approach or concept of designing the database. The main purpose of the physical data model is to show all the **structures** of the table including the **column name, column data type, constraints, keys(primary and foreign)**, and the relationship among tables. The following are the features of a physical data model:

1.Specifies all the columns and tables.

2.Specifies foreign keys that usually define the relationship between tables.

3.Based on user requirements, de-normalization might occur.

4.Since the physical consideration is taken into account so there will straightforward reasons for difference than a logical model.

5.Physical models might be different for different RDBMS. For example, the data type column may be different in MySQL and SQL Server.

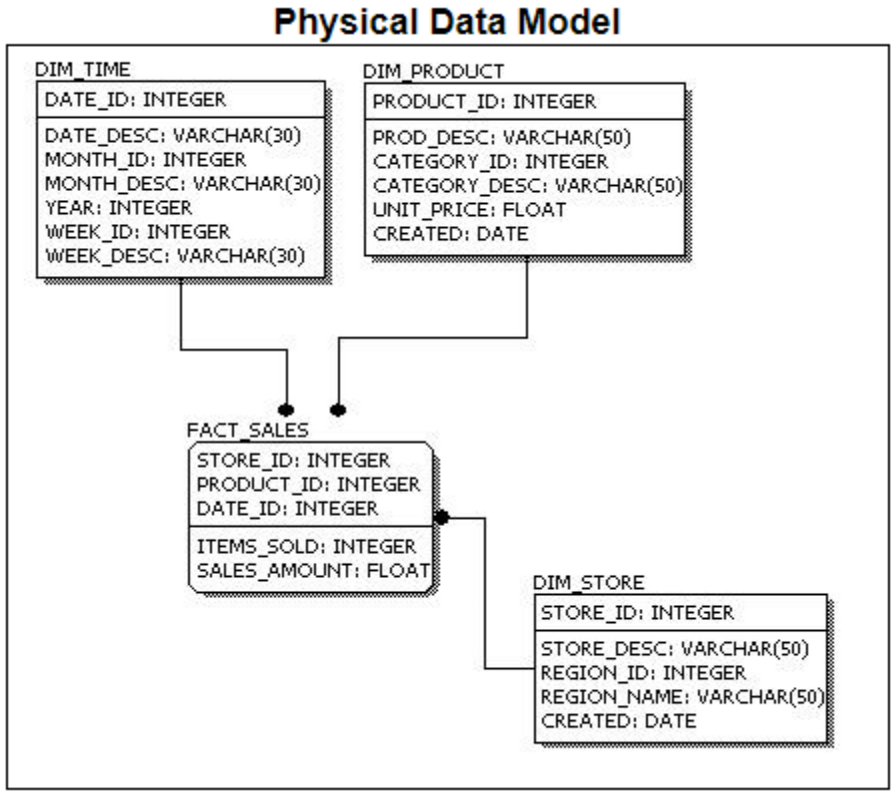
While designing a physical data model, the following points should be taken into consideration:

1.Convert the entities into tables.

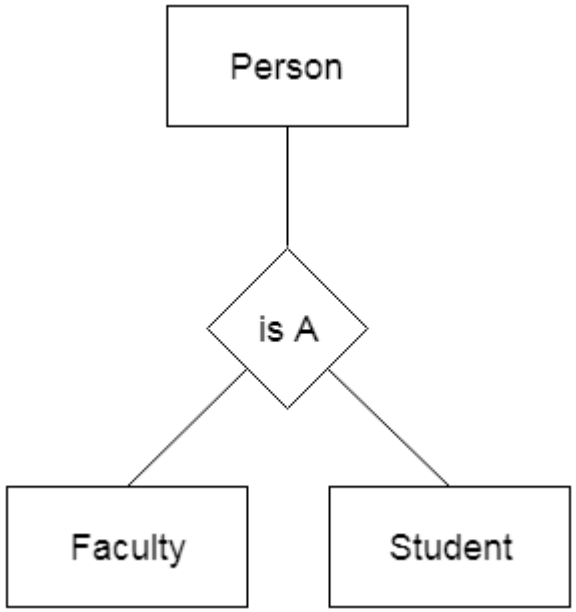
2.Convert the defined relationships into foreign keys.

3.Convert the data attributes into columns.

4.Modify the data model constraints based on physical requirements.

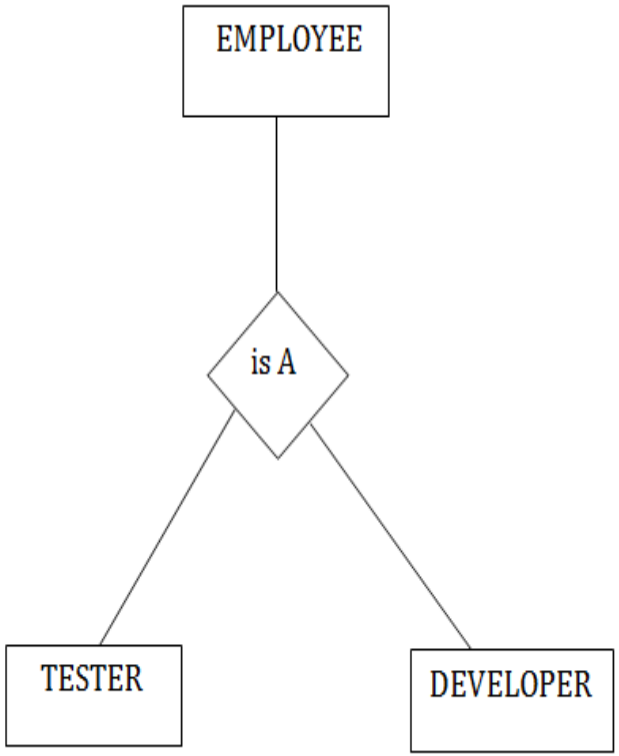


- Glossary
- **Entity** - An entity in the database can be defined as abstract data that we save in our database. For example, a customer, products.
- **Attributes** - An attribute is a detailed form of data consisting of entities like length, name, price, etc.
- **Relationship** - A relationship can be defined as the connection between two entities or figures. For example, a person can relate to multiple persons in a family.
- **Foreign key** - It acts as a referral to the Primary Key of another table. A foreign key contains columns with values that exist only in the primary key column they refer to.
- **Primary key** - A primary key is the pointer of records that is unique and not null and is used to uniquely identify attributes of a table.
- **Normalization** - A flexible data model needs to follow certain rules. Applying these rules is called normalizing**.**



Generalization

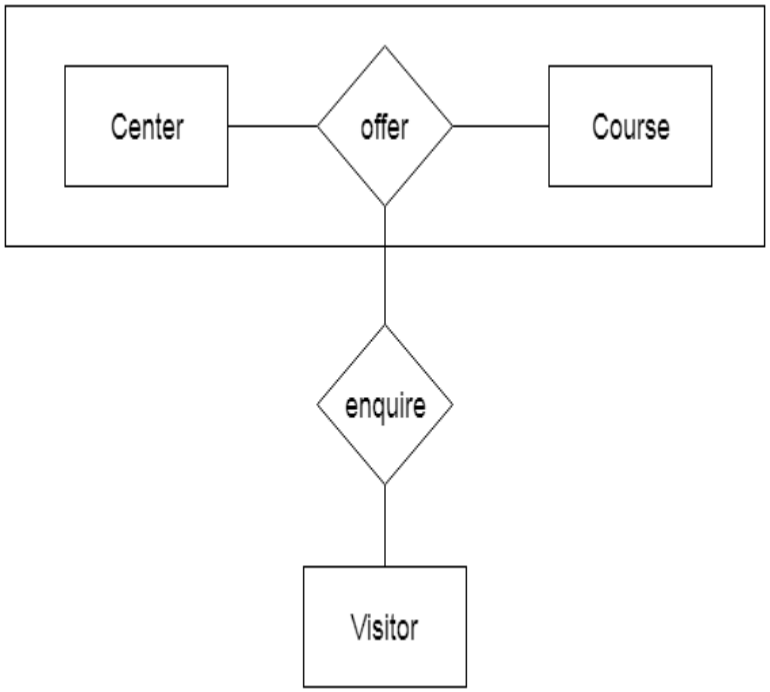
- Generalization is like a bottom-up approach in which two or more entities of lower level combine to form a higher level entity if they have some attributes in common.
- In generalization, an entity of a higher level can also combine with the entities of the lower level to form a further higher level entity.
- Generalization is more like subclass and superclass system, but the only difference is the approach. Generalization uses the bottom-up approach.
- In generalization, entities are combined to form a more generalized entity, i.e., subclasses are combined to make a superclass.
- **For example,** Faculty and Student entities can be generalized and create a higher level entity Person.



Specialization

- Specialization is a top-down approach, and it is opposite to Generalization. In specialization, one higher level entity can be broken down into two lower level entities.
- Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics.
- Normally, the superclass is defined first, the subclass and its related attributes are defined next, and relationship set are then added.
- **For example:** In an Employee management system, EMPLOYEE entity can be specialized as TESTER or DEVELOPER based on what role they play in the company.

Aggregation

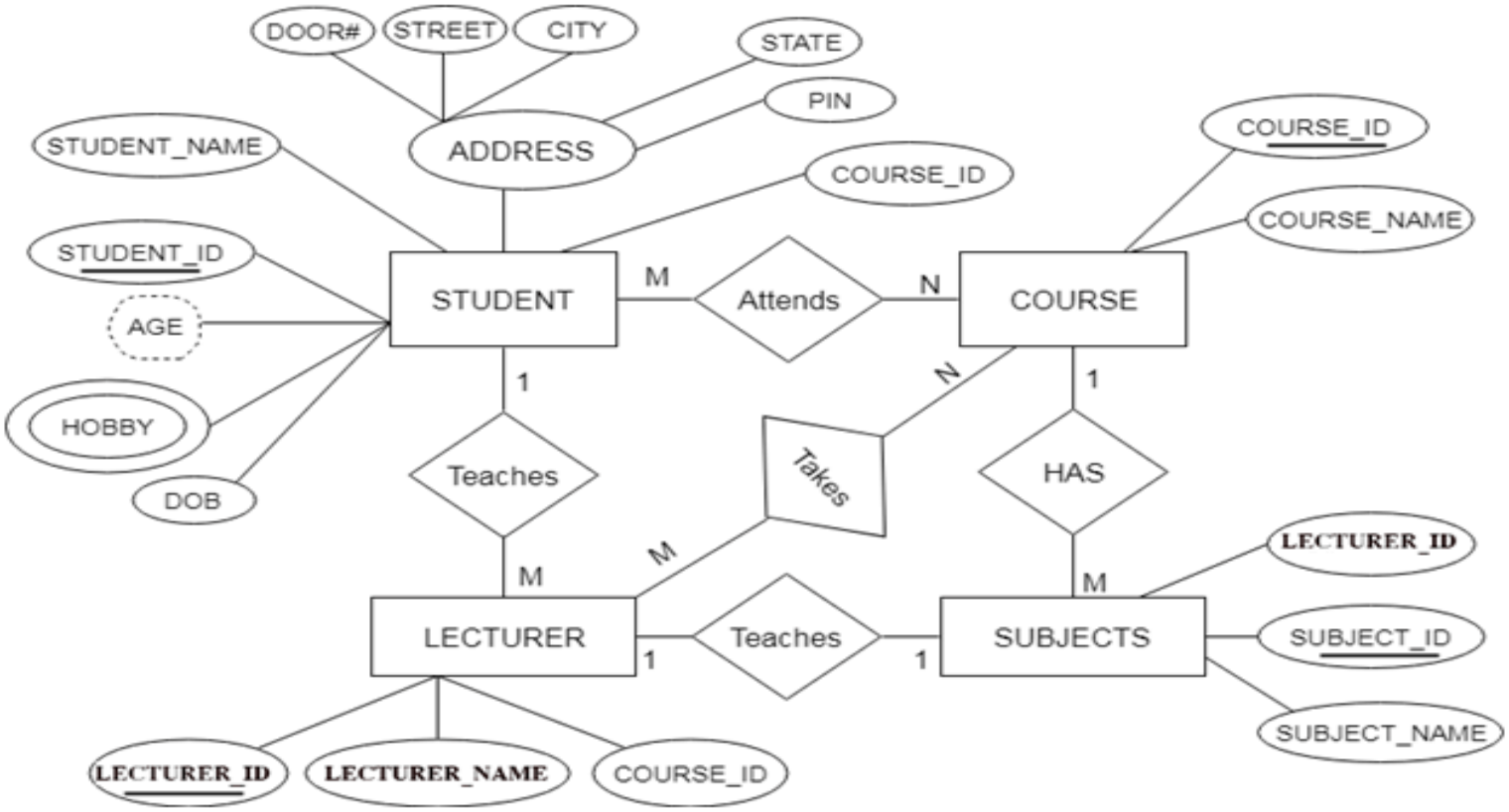


- In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity.

For example: Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor. In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will ask the enquiry about both.

Reduction of ER diagram to Table

- The database can be represented using the notations, and these notations can be reduced to a collection of tables.
- In the database, every entity set or relationship set can be represented in tabular form.



- There are some points for converting the ER diagram to the table:
- **Entity type becomes a table.**
- In the given ER diagram, LECTURE, STUDENT, SUBJECT and COURSE forms individual tables.
- **All single-valued attribute becomes a column for the table.**
- In the STUDENT entity, STUDENT_NAME and STUDENT_ID form the column of STUDENT table. Similarly, COURSE_NAME and COURSE_ID form the column of COURSE table and so on.
- **A key attribute of the entity type represented by the primary key.**
- In the given ER diagram, COURSE_ID, STUDENT_ID, SUBJECT_ID, and LECTURE_ID are the key attribute of the entity.
- **The multivalued attribute is represented by a separate table.**
- In the student table, a hobby is a multivalued attribute. So it is not possible to represent multiple values in a single column of STUDENT table. Hence we create a table STUD_HOBBY with column name STUDENT_ID and HOBBY. Using both the column, we create a composite key.
- **Composite attribute represented by components.**
- In the given ER diagram, student address is a composite attribute. It contains CITY, PIN, DOOR#, STREET, and STATE. In the STUDENT table, these attributes can merge as an individual column.
- **Derived attributes are not considered in the table.**
- In the STUDENT table, Age is the derived attribute. It can be calculated at any point of time by calculating the difference between current date and Date of Birth.
- Using these rules, you can convert the ER diagram to tables and columns and assign the mapping between the tables. Table structure for the given ER diagram is as below: