❖ WHAT IS DBMS (DATABASE MANAGEMENTSYSTEM)?

DATA: Collection of facts in raw form. Data is a group of unprocessed form. Data does not have any meaning. Data is raw material for *data* processing.

INFORMATION: It is a group of processed data. It has a self define meaning. When data is interpreted and processed to determine its true meaning then it is called Information. For example : Data: 100
Information : Rs. 100

Data	Information
• Data refers to raw facts that have no specific meaning.	• Information refers to processed data that has a purpose and meaning.
• The word 'data' is derived from the Latin word 'datum', which means 'something that is given'.	• The word 'information' is derived from the Latin word 'informatiō', which means 'formation or conception'.
• The data is independent of the information.	• Information is dependent on data.
• Data or raw data is not enough to make a decision.	• The information is sufficient to help make a decision in the respective context.

DATABASE: A group of information. Organized collection of inter related data. Database System is developed to maintain large bodies of information. Information stored in a database is arranged in a particular order. It is designed, built and populated with data for a specific purpose.

META DATA: Database definition or descriptive information is stored in database in the form of a database catalog or dictionary is called metadata. Data about data is called metadata.

DBMS - It is a software that manages the collection of information by insert, update, delete, create& selecting data.

DBMS provides an interface to perform various operations like database creation, storing data in it, updating data, creating a table in the database and a lot more.

It provides protection and security to the database. In the case of multiple users, it also maintains data consistency.

Database	Collection of Large Amount information
Management	Database Management Activities (Create, Insert, Update, Delete, Select)
System	A program to handle Management Activities

❖ **NEED OF DATABASE MANAGEMENT SYSTEM or**

PRIMARY GOALS OF DBMS ARE:

1. To provide a way to **store and retrieve** database information that is both convenient and efficient.
2. To **manage** large and small bodies of information. It involves defining structures for storage of information and providing mechanism for manipulation of information.
3. It should ensure **safety** of information stored, despite system crashes or attempts at unauthorized access.
4. If data are to be **shared** among several users, then system should avoid possible anomalous results.

ADVANTAGES OF DBMS:

(1) **Reduced data Redundancy (duplicacy).**
By sharing data among the different users using different applications, it reduced the data redundancy. So result is that requires less storage and reduces cost.

(2) **Reduced data inconsistency.**
Redundancy leads inconsistency. So, if redundancy reduced or removed then database is no longer inconsistent.

(3) **Reduced complexity of the organization's information systems environment.**
In DBMS, data stored centrally that means any user from different location can access and view the data. So, decision making process becomes fast. In any organization information is easily available using DBMS. So, overall it reduced the complexity of the organization.

(4) **Restricting Unauthorized Access (Improve Security).**
DBMS provide some high level of security and every user must have some permission to access the database.

(5) **Improve strategic use of corporate data.**
Corporate data means data is arranged in very systematic manner. Using the DBMS all type of corporate data can be easily and well managed.

(6) **Provide Storage for Program Objects.**
When database is created all the required memory is allocated at the time of creation.

(7) **Provide Storage Structure for Efficient Query Processing.**
In DBMS every query has fix format SELECT…FROM…WHERE. When query is executed system defined its own area. In oracle it is known as Cursor.

(8) **Provide Backup &Recovery.**
DBMS programs provide DCL statement which controlling database operation. In oracle ROLLBACK for Undo and COMMIT to save the changes. SAVE Point is also used in oracle.

(9) **Provide multiple User Interfaces.**
DBMS help in placing every data on one place in the form of database so that many users access the same data from different location.

(10) Representing Complex Relationship Among data.

Database is collection of tables and table is collection of data which divide into row and column. Using ER diagram you can represent the data and tables.

❖ DISADVANTAGES OF DBMS

(1) Complexity

A good DBMS is a complex piece of software which the people who are attached with DBMS () must understand all the functionality in detail. Fail to understand the system can lead to take wrong decision.

(2) Hardware Cost

For better performance and volume of data increase it requires additional hardware which may increase the cost of DBMS.

(3) Maintenance Cost

Maintenance means to update/modify the system. Any type of maintenance in DBMS requires more cost.

(4) Security Problem

All DBMS software provides a minimum level of security by default.

(5) Integrity Problem

We cannot apply different business rules in DBMS software. It is possible through application software.

(6) Backup & Recovery Problem

All DBMS software not provides inbuilt backup & recovery facility. When backup and recovery is more important for daily operations so it require purchasing from third party.

(7) Higher Impact of failure problem

❖ VARIOUS TYPES OF DATABASE SYSTEM APPLICATIONS

1. **Banking** – For customer information, accounts, and loans, and banking transactions.[All transactions]

2. **Airlines** – For reservation and schedule information. [Reservations, schedules]

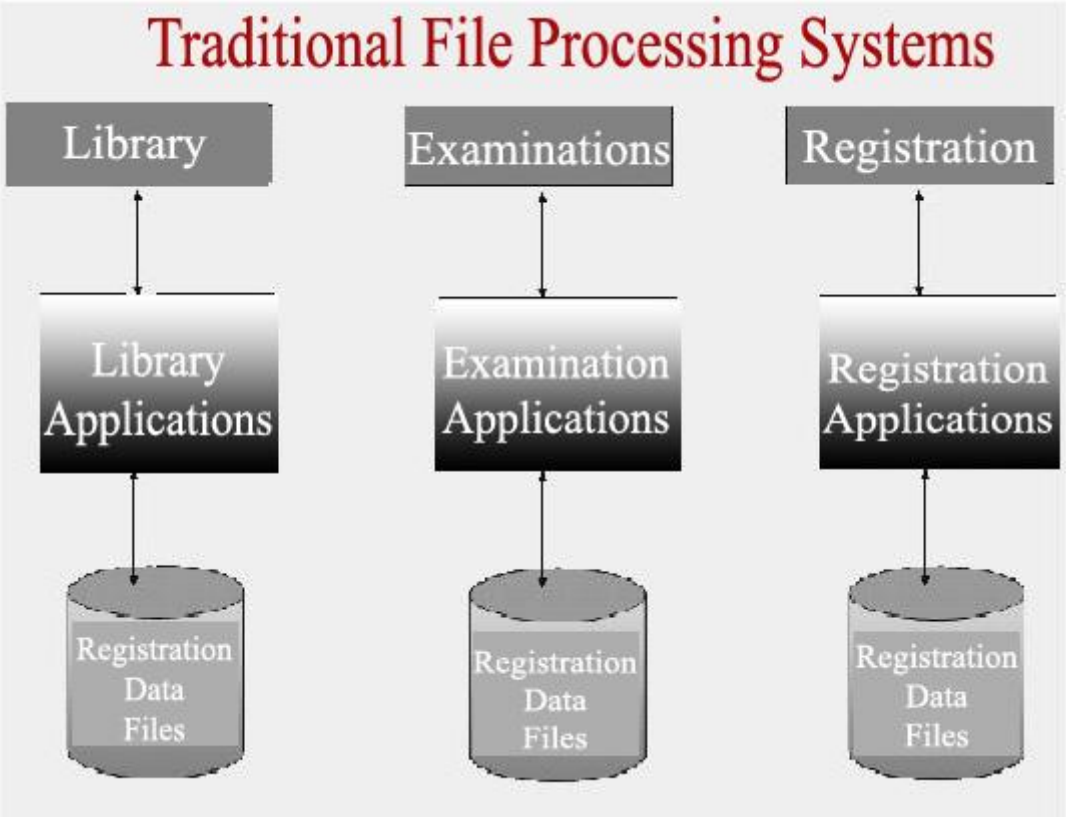
3. **Universities** – For student information, course registrations, and grades. [Registration, grades]

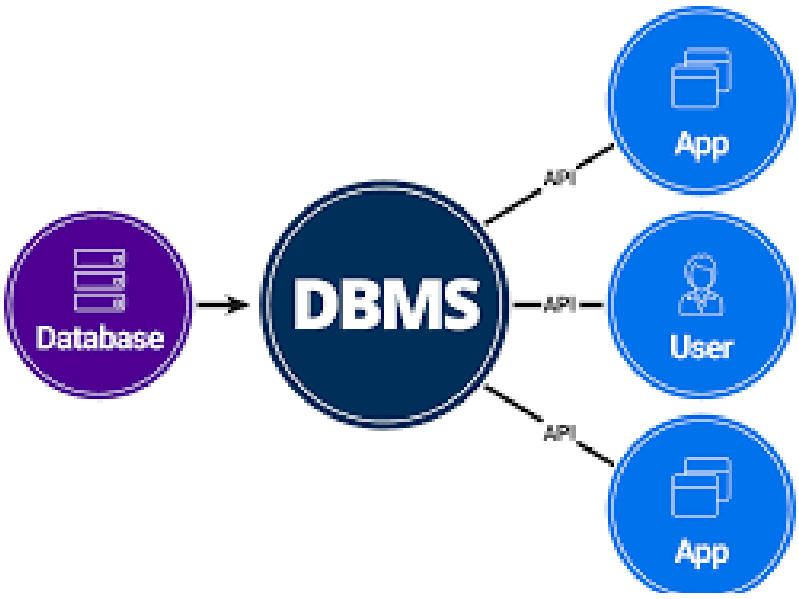
4. **Telecommunication** – For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing formation about communication networks.
5. **Sales** – For customer, product, and purchase information. [Customers, products, purchases]

6. **Manufacturing** – For management of supply chain and for tracking production of items in factories, inventories of items in warehouses/stores, and orders for items. [Production, inventory, orders, supply chain]

7. **Human Resources** – For information about employees, salaries, payroll taxes and benefits, and generation of paychecks. [Employee records, salaries, tax deductions]

❖ **DIFFERENCE BETWEEN DBMS AND FILE-PROCESSING SYSTEM**





DBMS	FILE-PROCESSING SYSTEMS
1. **Redundancies** and inconsistencies in data are reduced due to single file formats and duplication of data is eliminated.	1. Redundancies and inconsistencies in data exist due to single file formats and duplication of data.
2. **Data is easily accessed** due to standard query procedures.	2. Data cannot be easily accessed due to special application programs needed to access data.
3. Isolation/**retrieval of required data is possible** due to common file format, and there are provisions to easily retrieve data	3. Data isolation is difficult due to different file formats and also because new application programs have to be written.
4. **Integrity constraints**, whether new or old, can be created or modified as per need.	4. Introduction of integrity constraints is tedious and again new application programs have to be written.
5. **Atomicity** of updates is possible.	5. Atomicity of updates may not be maintained.
6. **Several users** can access data at the same time i.e concurrently without problems	6. Concurrent accesses may cause problems such as inconsistencies.
7. **Security** features can be enabled in DBMS very easily.	7. It may be difficult to enforce security features.

❖ **List of DBMS Software:**

Oracle	Microsoft Access	SQlite	Progress
DB2	Microsoft SQL Server	Teradata	CSQL
Sybase	Microsoft Visual FoxPro	Ingres	Open Link Virtuoso
Mysql	PostgreSQL	Infomix	

❖ **DBMS and RDBMS terms comparison**

Common Term	**DBMS**	**RDBMS**
Database	Table	Database
Table	Table	Relation
Column	Field	Attribute
Row	Record	Tuple

❖ **DATA INDEPENDENCE: LOGICAL & PHYSICAL**

Data independence is the ability to modify a schema definition in one level without affecting a schema definition in a higher level is called data independence.
There are two types of data independence:

1. Physical data independence

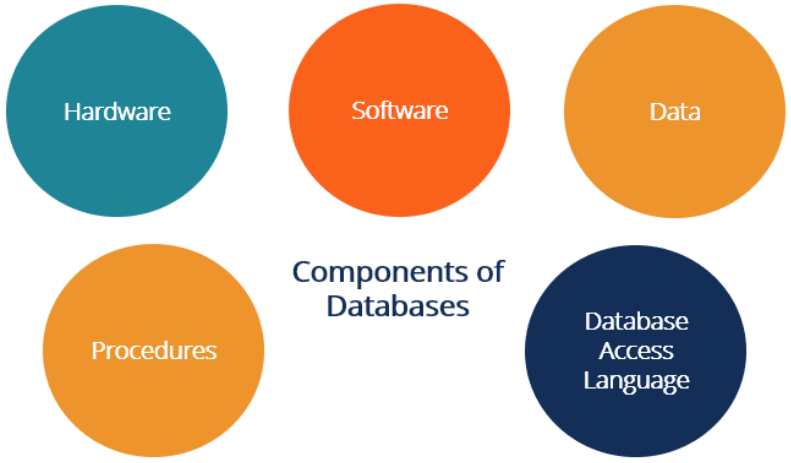
- It is the ability to modify the physical schema without causing application programs to be rewritten.
- Modifications at this level are usually to improve performance.

2. Logical data independence

- It is the ability to modify the conceptual scheme without causing application programs to be rewritten
- It is usually done when logical structure of database is altered. Logical data independence is harder to achieve as the application programs are usually heavily dependent on the logical structure of the data. An analogy is made to abstract data types in programming languages.

❖ COMPONENTS of Database

The five major components of a database are:



1. Hardware

Hardware refers to the physical, electronic devices such as computers and hard disks that offer the interface between computers and real-world systems.

3. Software

Software is a set of programs used to manage and control the database and includes the database software, operating system, network software used to share the data with other users, and the applications used to access the data.

3. Data

Data are raw facts and information that need to be organized and processed to make it more meaningful. Database dictionaries are used to centralize, document, control, and coordinate the use of data within an organization. A database is a repository of information about a database (also called metadata).

4. Procedures

Procedures refer to the instructions used in a database management system and encompass everything from instructions to setup and install, login and logout, manage the day-to-day operations, take backups of data, and generate reports.

5. Database Access Language

Database Access Language is a language used to write commands to access, update, and delete data stored in a database. Users can write commands using Database Access Language before submitting them to the database for execution. Through utilizing the language, users can create new databases, tables, insert data, and delete data.

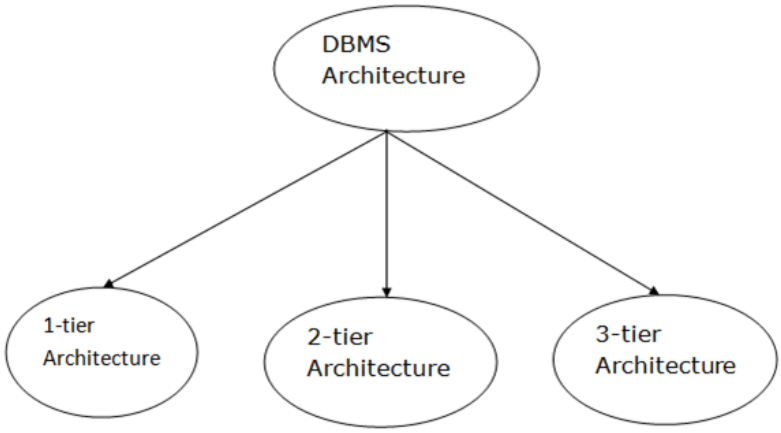
❖ DBMS Architecture

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.

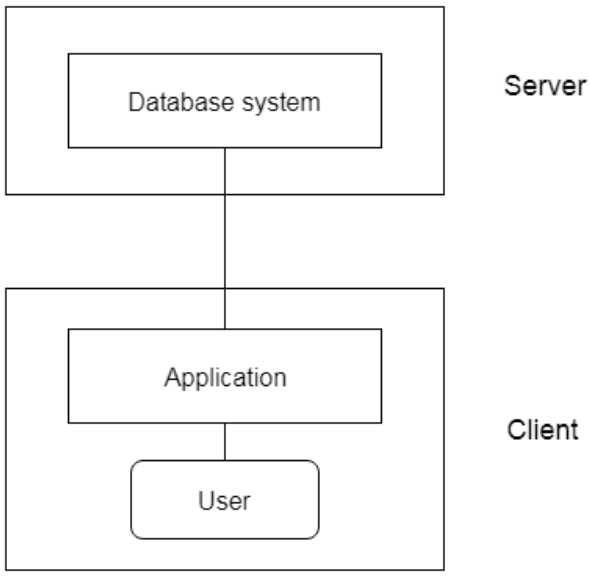


Fig: 2-tier Architecture

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.

❖ **DATABASE MODEL**

A Database model defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system.

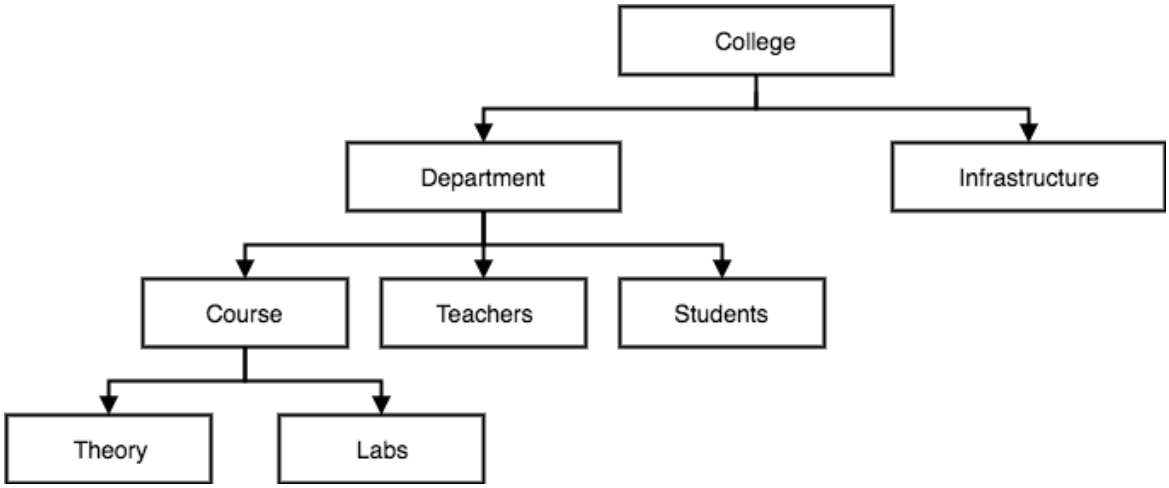
- Hierarchical Model
- Network Model
- Entity-relationship Model
- Relational Model

Hierarchical Model

This database model organizes data into a tree-like-structure, with a single root, to which all the other data is linked. The hierarchy starts from the Root data, and expands like a tree, adding child nodes to the parent nodes.
In this model, a child node will only have a single parent node.

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

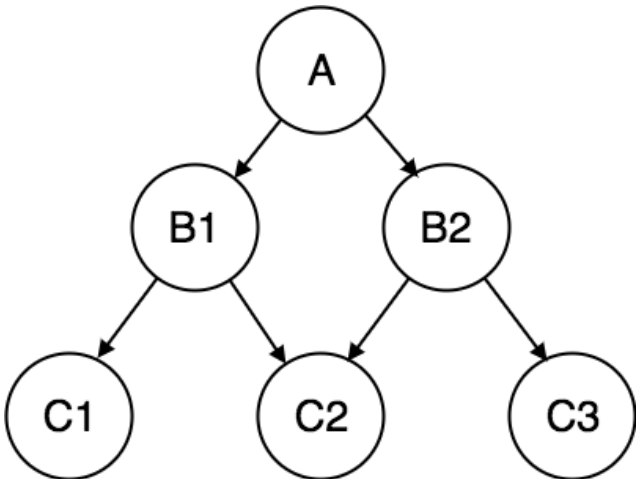
In hierarchical model, data is organized into tree-like structure with one one-to-many relationship between two different types of data, for example, one department can have many courses, many professors and of-course many students.



Network Model

This is an extension of the Hierarchical model. In this model data is organised more like a graph, and are allowed to have more than one parent node.
In this database model data is more related as more relationships are established in this database model. Also, as the data is more related, hence accessing the data is also easier and fast. This database model was used to map many-to-many data relationships.

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



Relational Model

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

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

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

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

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

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

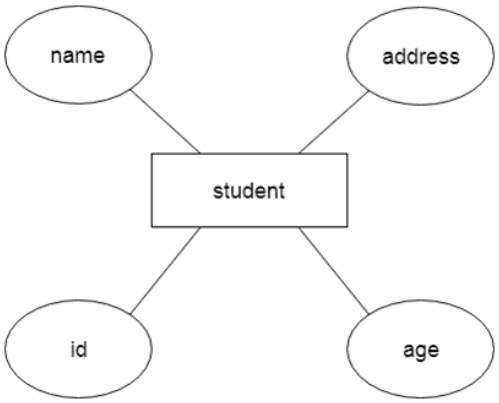
subject_id	name	teacher
1	Java	Mr. J
2	C++	Miss C
3	C#	Mr. C Hash
4	Php	Mr. P H P

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

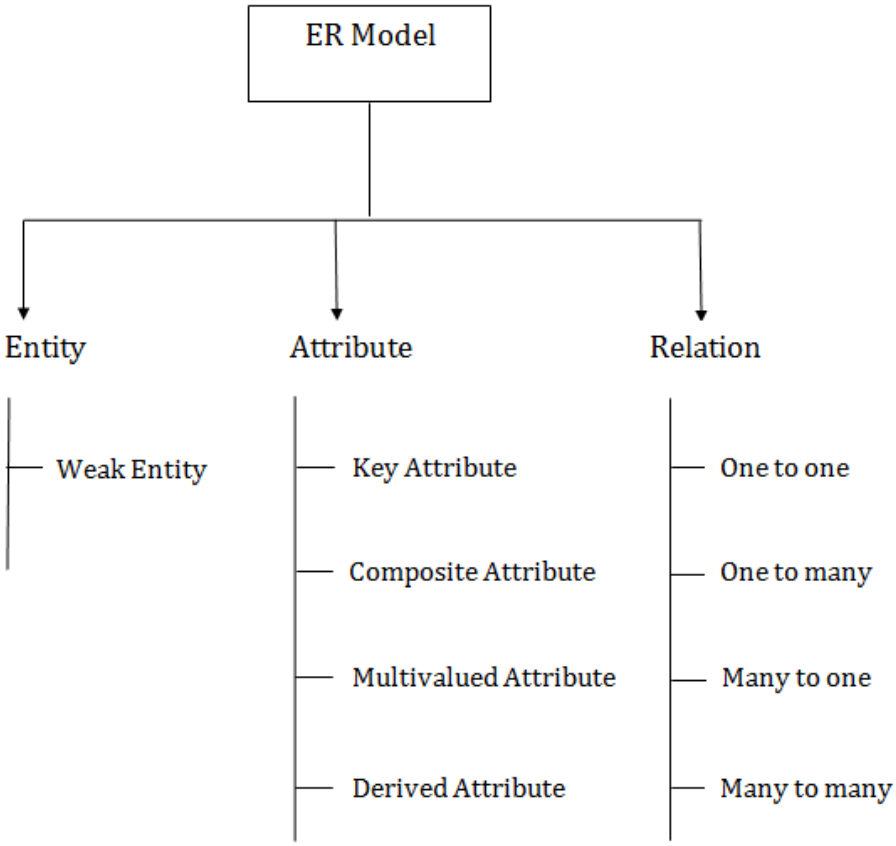
Entity-relationship Model

- 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 modeling, 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.



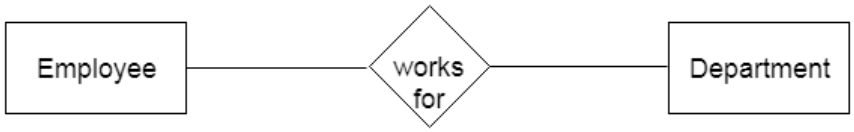
Component of ER Diagram



1. 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.

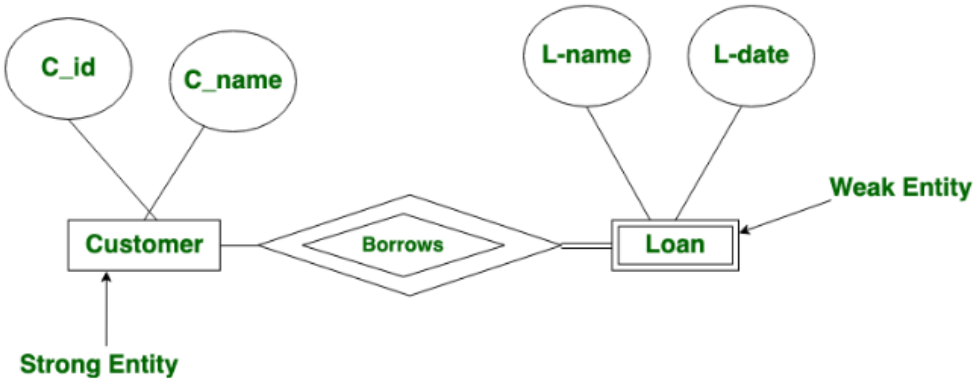


Strong Entity:

A strong entity is not dependent on any other entity in the schema. A strong entity will always have a primary key. Strong entities are represented by a single rectangle. The relationship of two strong entities is represented by a single diamond.
Various strong entities, when combined together, create a strong entity set.

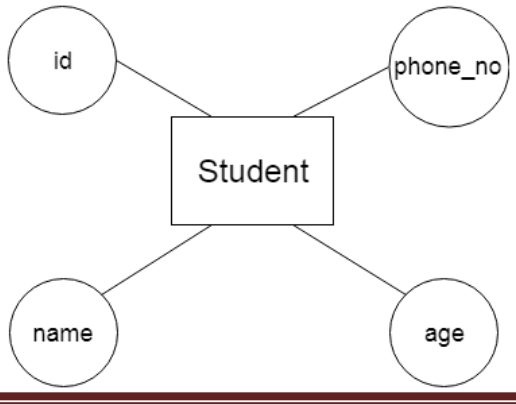
Weak Entity

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.



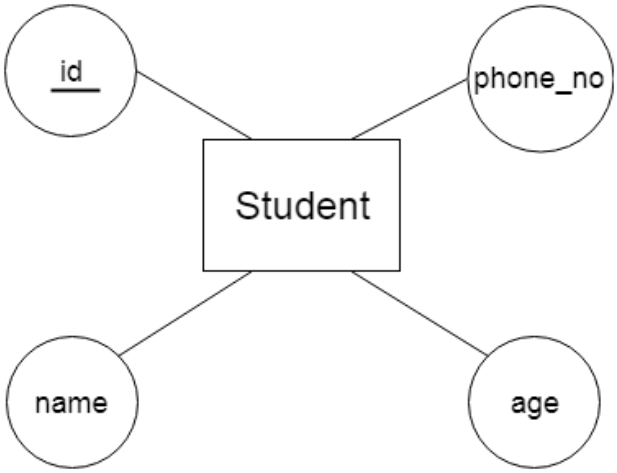
2. Attribute

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

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.

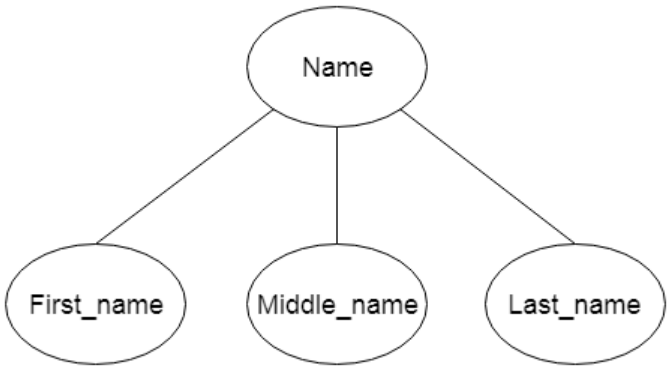


Difference between Strong and Weak Entity:

S.NO	Strong Entity	Weak Entity
1.	Strong entity always has a primary key.	While a weak entity has a partial discriminator key.
2.	Strong entity is not dependent on any other entity.	Weak entity depends on strong entity.
3.	Strong entity is represented by a single rectangle.	Weak entity is represented by a double rectangle.
4.	Two strong entity's relationship is represented by a single diamond.	While the relation between one strong and one weak entity is represented by a double diamond.
5.	Strong entities have either total participation or not.	While weak entity always has total participation.

b. Composite Attribute

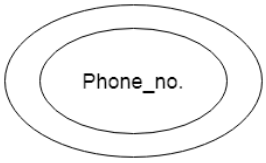
An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.



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.

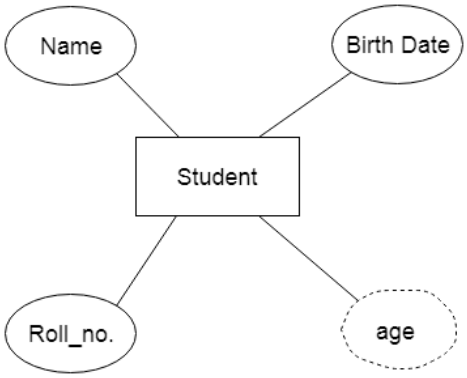
For example, a student can have more than one phone number.



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.

Types of relationship are as follows:

a. One-to-One Relationship

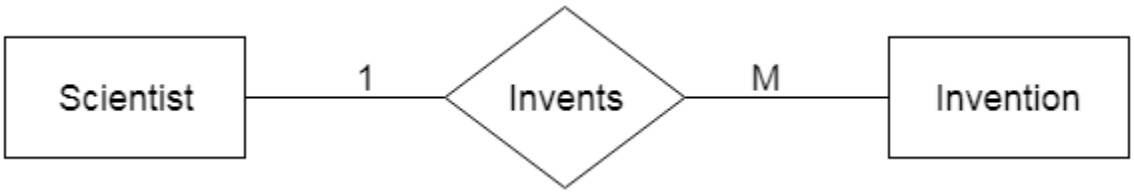
When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.

For example, A female can marry to one male, and a male can marry to one female.

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.

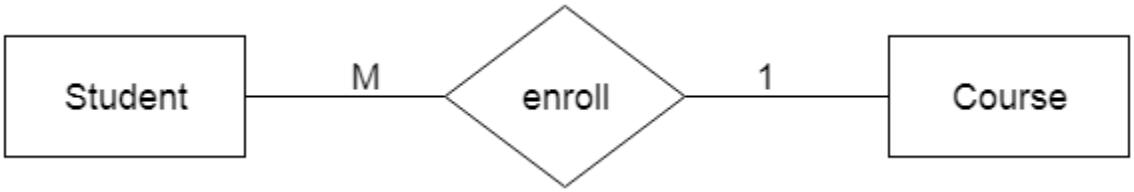
For example, Scientist can invent many inventions, but the invention is done by the only specific scientist.



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.

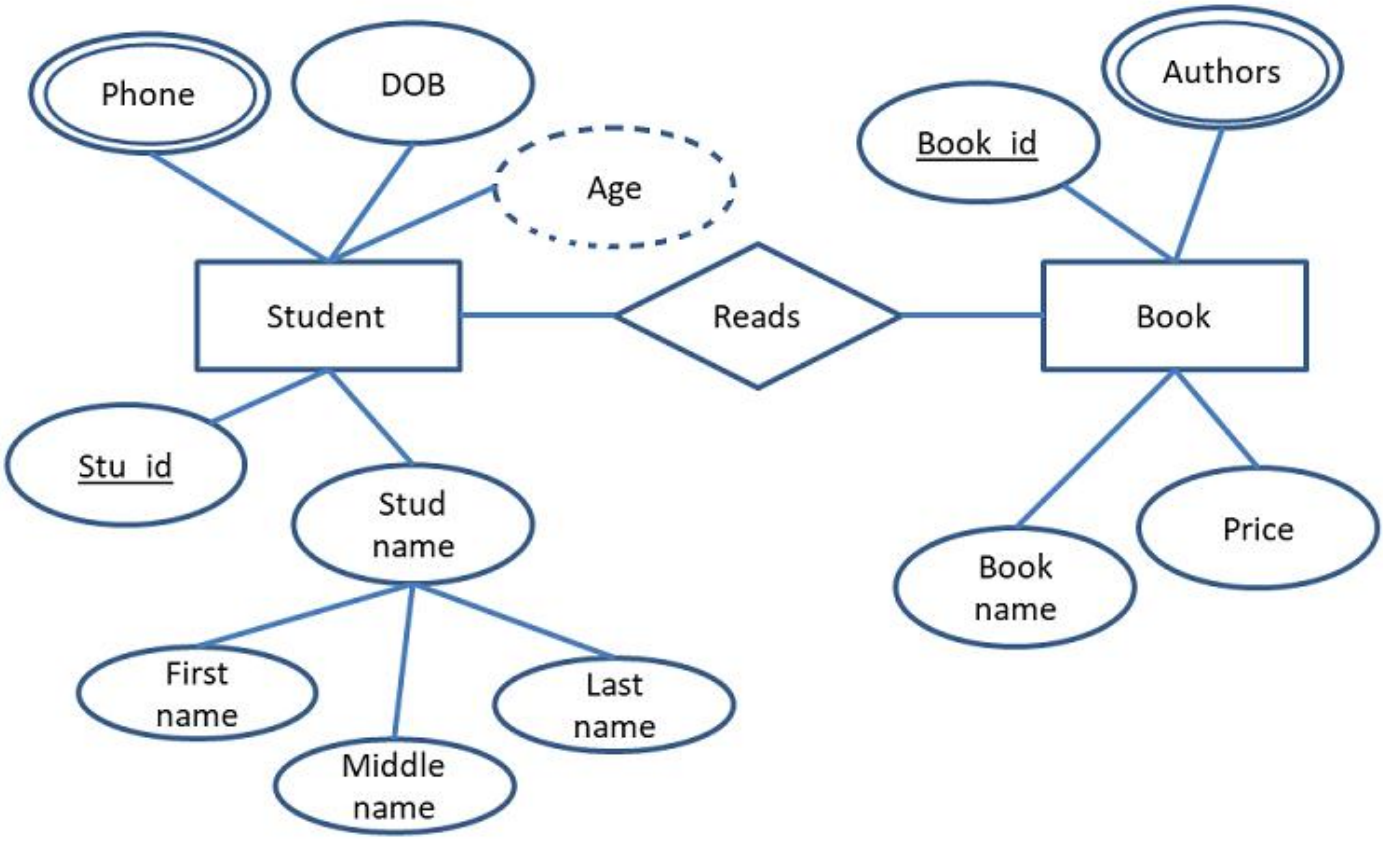
For example, Student enrolls for only one course, but a course can have many students.



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.

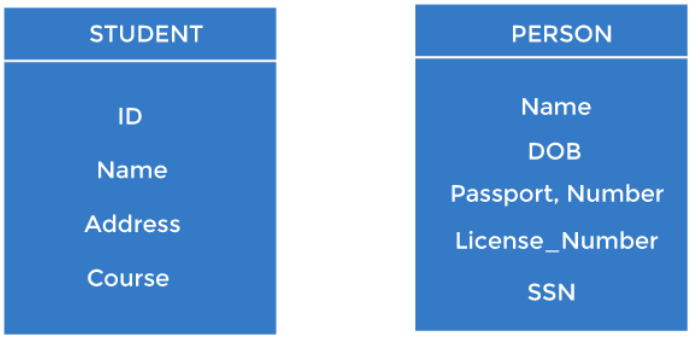

Phone
DOB
Age
Authors
Book_id
Student
Reads
Book
Stu_id
Stud name
Price
Book name
First name
Last name
Middle name

S.No	Hierarchical Data Model	Network Data Model	Relational Data Model
1.	**Relationship** between records is of parent child type.	Relationship between records is expressed in the form of pointers or links.	Relationship between record is represented by a relation that contains a key for each record involved in the relations.
2.	Many-to-many relationship cannot be expressed in this model.	Many-to-many relationship can also be implemented.	Many-to-many relationship can be easily implemented
3.	It is a simple, Straight forward and natural method of implementing record relationships	Record relationship implementation is quite complex due to the use of pointers.	Relationship implementation is very easy though the use of a key or composite key field(s).
4.	This type of model is useful only when there is some hierarchical character in the database.	Network model is useful for representing such records which have many -to-many relationships.	Relational model is useful for representing most of the real world Objects and relationships among them.
5.	In order to represent links among records, pointers are used. Thus relationships among records are physical.	In Network model also the relationship among records are physical.	Relational model does not maintain physical connection records. Data is organized logically in the form of rows columns and stored in table.
6.	Searching for a record is very difficult since one can retrieve a child only after going though its parent record	Searching a record is easy since there are multiple access paths to a data element.	A unique, indexed key field is used to search for a data element.
7.	During updation or deletion process, chance of data inconsistency is involved.	No problem of inconsistency exists in network model because a data element is physically located at just one place.	Data integrity maintaining methods like Normalization process, etc. are adopted for consistency.

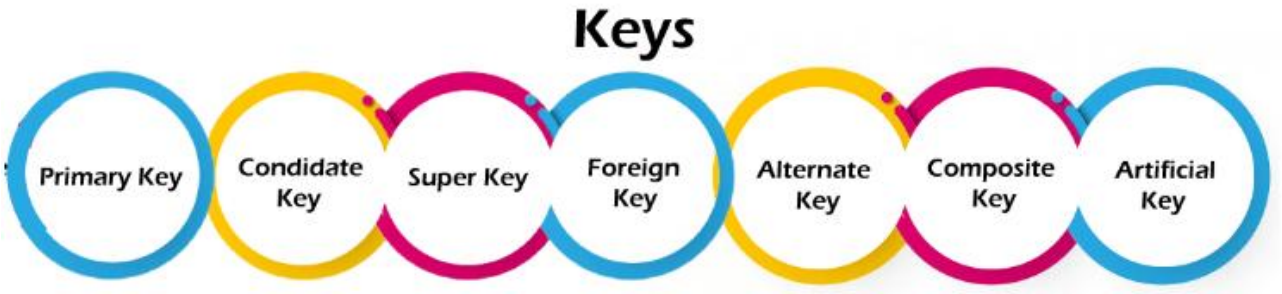
Keys

- 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.

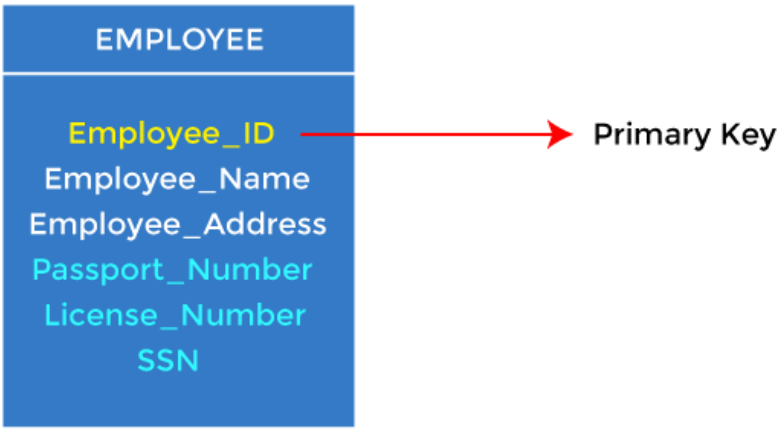


Types of keys:



1. Primary key

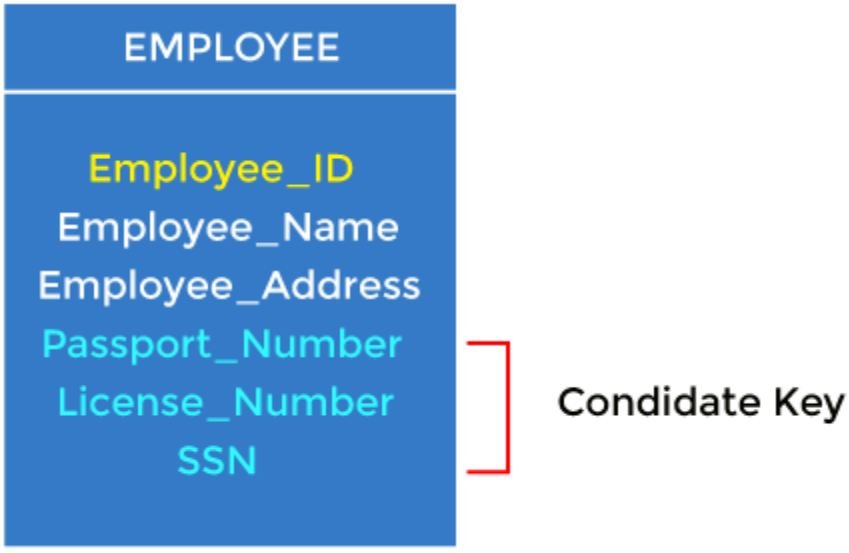
- It is the first key used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys, as we saw in the PERSON table. The key which is most suitable from those lists becomes a primary key.
- In the EMPLOYEE table, ID can be the primary key since it is unique for 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

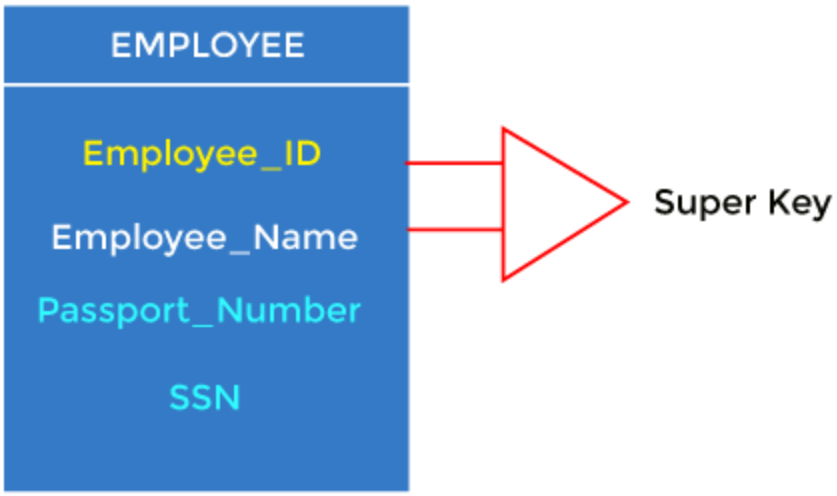
- A candidate key is an attribute or set of attributes that can uniquely identify a tuple.
- Except for the primary key, the remaining attributes are considered a candidate key. The candidate keys are as strong as the primary key.

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.

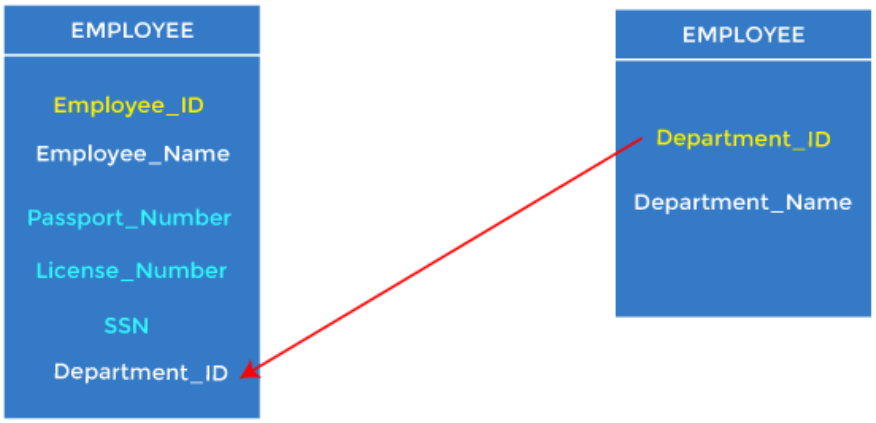


For example: In the above EMPLOYEE table, for(EMPLOEE_ID, EMPLOYEE_NAME), the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key.

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

4. Foreign key

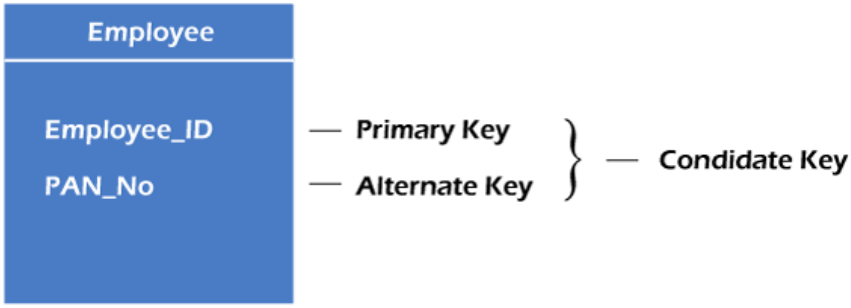
- Foreign keys are the column of the table used to point to the primary key of another table.
- Every employee works in a specific department in a company, and employee and department are two different entities. So we can't store the department's information in the employee table. That's why we link 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

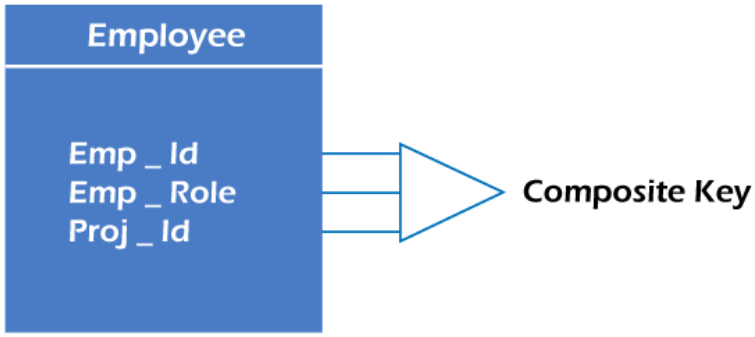
There may be one or more attributes or a combination of attributes that uniquely identify each tuple in a relation. These attributes or combinations of the attributes are called the candidate keys. One key is chosen as the primary key from these candidate keys, and the remaining candidate key, if it 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.

For example, the primary key, which is composed of Emp_ID, Emp_role, and Proj_ID, is large in employee relations. So it would be better to add a new virtual attribute to identify each tuple in the relation uniquely.

Normalization

A large database defined as a single relation may result in data duplication. This repetition of data may result in:

- Making relations very large.
- It isn't easy to maintain and update data as it would involve searching many records in relation.
- Wastage and poor utilization of disk space and resources.
- The likelihood of errors and inconsistencies increases.

So to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that are satisfy desirable properties. Normalization is a process of decomposing the relations into relations with fewer attributes.

What is Normalization?

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
- Normalization divides the larger table into smaller and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

Why do we need Normalization?

The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.

Data modification anomalies can be categorized into three types:

- **Insertion Anomaly:** Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
- **Deletion Anomaly:** The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
- **Updation Anomaly:** The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Dependency:

A dependency is a constraint that applies to or defines the relationship between attributes. It occurs in a database when information stored in the same database table uniquely determines other information stored in the same table.

Database Dependencies and Functional Dependencies

Saying that there is a dependency between attributes in a table is the same as saying that there is a functional dependency between those attributes. If there is a dependency in a database such that attribute B is dependent upon attribute A, you would write this as:

A -> B

Functional Dependency

If the information stored in a table can uniquely determine another information in the same table, then it is called Functional Dependency. Consider it as an association between two attributes of the same relation.

If P functionally determines Q, then

P -> Q

Let us see an example −

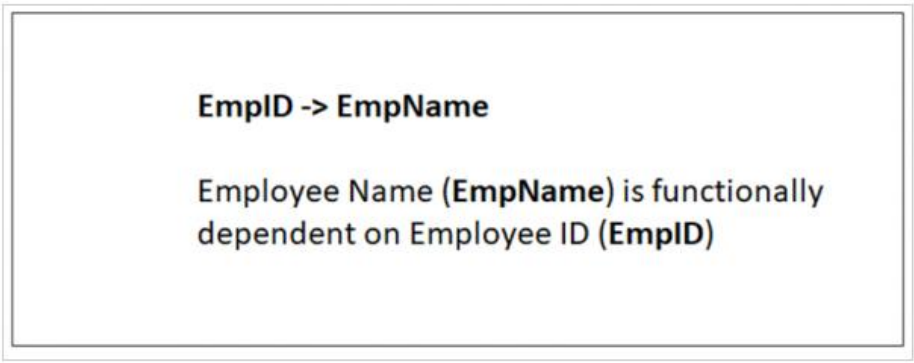
<Employee>

EmpID	EmpName	EmpAge
E01	Amit	28
E02	Rohit	31

In the above table, **EmpName** is functionally dependent on **EmpID** because **EmpName** can take only one value for the given value of **EmpID:**

EmpID -> EmpName

The same is displayed below −



Fully-functionally Dependency

An attribute is fully functional dependent on another attribute, if it is Functionally Dependent on that attribute and not on any of its proper subset.

For example, an attribute Q is fully functional dependent on another attribute P, if it is Functionally Dependent on P and not on any of the proper subset of P.

Let us see an example −

<ProjectCost>

ProjectID	ProjectCost
001	1000
002	5000

<EmployeeProject>

EmpID	ProjectID	**Days** (spent on the project)
E099	001	320
E056	002	190

The above relations states:

EmpID, ProjectID, ProjectCost -> Days

However, it is not fully functional dependent.

Whereas the subset **{EmpID, ProjectID}** can easily determine the **{Days}** spent on the project by the employee.

This summarizes and gives our fully functional dependency −

{EmpID, ProjectID} -> (Days)

Transitive Dependency

When an indirect relationship causes functional dependency it is called Transitive Dependency.

If P -> Q and Q -> R is true, then P-> R is a transitive dependency.

Multivalued Dependency

When existence of one or more rows in a table implies one or more other rows in the same table, then the Multi-valued dependencies occur.

If a table has attributes P, Q and R, then Q and R are multi-valued facts of P.

It is represented by double arrow −

->->

For our example:

P->->Q
Q->->R

In the above case, Multivalued Dependency exists only if Q and R are independent attributes.

Partial Dependency

Partial Dependency occurs when a nonprime attribute is functionally dependent on part of a candidate key.

The 2nd Normal Form (2NF) eliminates the Partial Dependency.

Let us see an example −

<StudentProject>

StudentID	ProjectNo	StudentName	ProjectName
S01	199	Katie	Geo Location
S02	120	Ollie	Cluster Exploration

In the above table, we have partial dependency; let us see how −

The prime key attributes are **StudentID** and **ProjectNo.**

As stated, the non-prime attributes i.e. **StudentName** and **ProjectName** should be functionally dependent on part of a candidate key, to be Partial Dependent.

The **StudentName** can be determined by **StudentID** that makes the relation Partial Dependent.

The **ProjectName** can be determined by **ProjectID**, which that the relation Partial Dependent.

Importance of Dependencies

Database dependencies are important to understand because they provide the basic building blocks used in database normalization, the process of efficiently organizing data in a database.

For example:

- For a table to be in second normal form (2NF), there must be no case of a nonprime attribute in the table that is functionally dependent upon a subset of a candidate key.
- For a table to be in third normal form (3NF), every nonprime attribute must have a nontransitive functional dependency on every candidate key.
- For a table to be in Boyce-Codd Normal Form (BCNF), every functional dependency (other than trivial dependencies) must be on a superkey.
- For a table to be in fourth normal form (4NF), it must have no multivalued dependencies.

Armstrong's Axioms in Functional Dependency in DBMS

• The term Armstrong axioms refer to the sound and complete set of inference rules or axioms, introduced by William W. Armstrong, that is used to test the logical implication of **functional dependencies**. If F is a set of functional dependencies then the closure of F, denoted as F^+, is the set of all functional dependencies logically implied by F. Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

Axioms –

1. **Axiom of reflexivity –**
   If A is a set of attributes and B is subset of A, then A holds B. If $B \subseteq A$ then $A \rightarrow B$ This property is trivial property.
2. **Axiom of augmentation –**
   If $A \rightarrow B$ holds and Y is attribute set, then $AY \rightarrow BY$ also holds. That is adding attributes in dependencies, does not change the basic dependencies. If $A \rightarrow B$, then $AC \rightarrow BC$ for any C.
3. **Axiom of transitivity –**
   Same as the transitive rule in algebra, if $A \rightarrow B$ holds and $B \rightarrow C$ holds, then $A \rightarrow C$ also holds. $A \rightarrow B$ is called as A functionally that determines B. If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

Secondary Rules –

These rules can be derived from the above axioms.

1. **Union –**
   If $A \rightarrow B$ holds and $A \rightarrow C$ holds, then $A \rightarrow BC$ holds. If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$
2. **Composition –**
   If $A \rightarrow B$ and $X \rightarrow Y$ holds, then $AX \rightarrow BY$ holds.
3. **Decomposition –**
   If $A \rightarrow BC$ holds then $A \rightarrow B$ and $A \rightarrow C$ hold. If $X \rightarrow YZ$ then $X \rightarrow Y$ and $X \rightarrow Z$
4. **Pseudo Transitivity –**
   If $A \rightarrow B$ holds and $BC \rightarrow D$ holds, then $AC \rightarrow D$ holds. If $X \rightarrow Y$ and $YZ \rightarrow W$ then $XZ \rightarrow W$.

Types of Normal Forms:

Normalization works through a series of stages called Normal forms. The normal forms apply to individual relations. The relation is said to be in particular normal form if it satisfies constraints.

Following are the various types of Normal forms:

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form.
4NF	A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.
5NF	A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless.

Advantages of Normalization

- Normalization helps to minimize data redundancy.
- Greater overall database organization.
- Data consistency within the database.
- Much more flexible database design.
- Enforces the concept of relational integrity.

Disadvantages of Normalization

- You cannot start building the database before knowing what the user needs.
- The performance degrades when normalizing the relations to higher normal forms, i.e., 4NF, 5NF.
- It is very time-consuming and difficult to normalize relations of a higher degree.
- Careless decomposition may lead to a bad database design, leading to serious problems.

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar

12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Example:

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:

1. {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010

333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Boyce Codd normal form (BCNF)

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

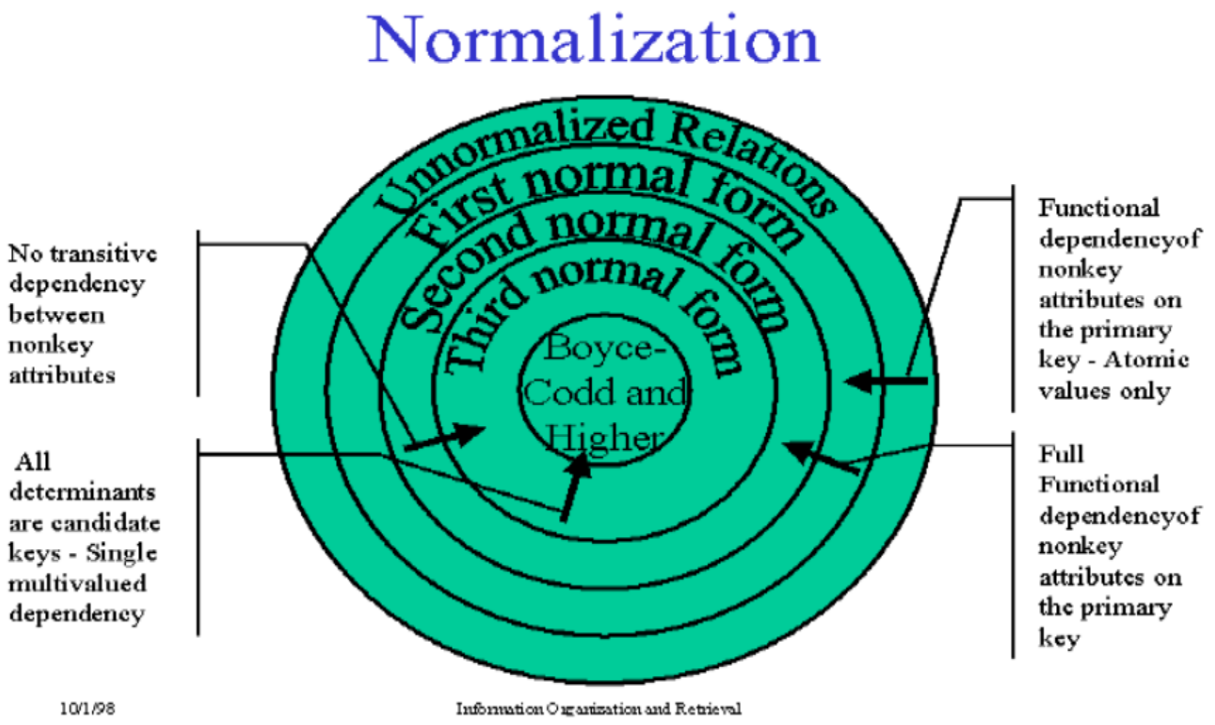
EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283

264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}



Now, this is in BCNF because left side part of both the functional dependencies is a key.