Introduction to Database Management System

What is data?

Any raw and unprocessed fact that we can record is known as data.
Example - New Delhi, India
In the above example New Delhi, India could be an address of a person or capital of a country but we are unable to analyse anything meaningful from it, hence it is data.

What is information?

When we process the data to get meaningful facts, it is called information.
Example - New Delhi is the capital of India.
Here we are getting some meaningful facts about New Delhi, hence it is information.

Difference between data and information

Data	Information
Raw and unorganised facts is Data	Data that is meaningful is Information
Data is not helpful in decision making.	Information helps in decision making
999999999 is data	A person's phone number is 999999999 is information.
2000 is data	I was born in 2000 is information

What is a database?

A Database is a collection of related data organised in a way that data can be easily accessed, managed and updated.
Example - Let us consider Facebook. It needs to store, update and show related data of members, activities, messages etc. Here we can use the database to do all these operations efficiently.

Database Management System-

A Database Management System or DBMS is a system that allows creation, definition and manipulation of databases, allowing users to store, process and analyse data easily.

DBMS provides users with an interface or a tool, to perform various operations like creating a database, storing data in it, updating data, creating tables in the database and a lot more.

MySQL, PostgreSQL, Microsoft Access, Oracle, MongoDB, Cassandra etc are all examples of DBMS.

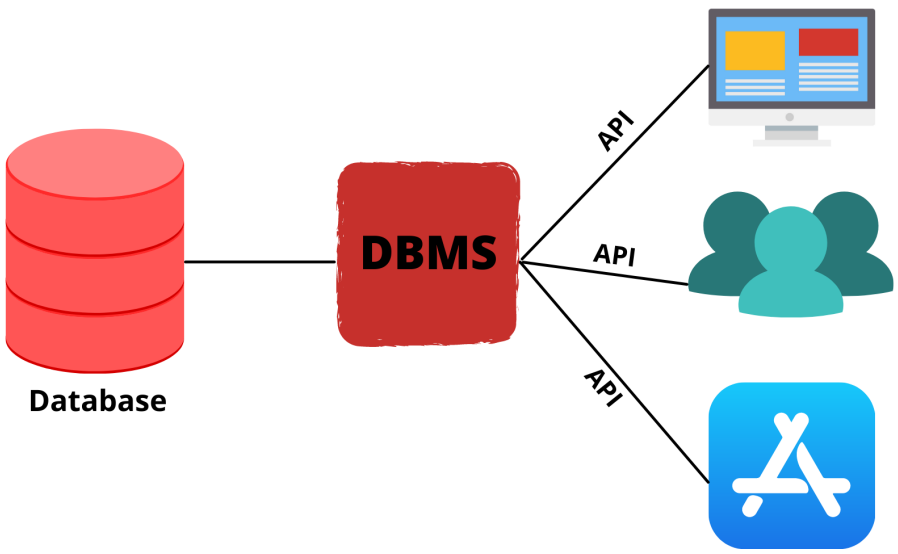


Figure: Database access through DBMS

What are File Systems ?

File System is a way of naming the files and storing them in a storage medium. File Systems helps in organizing the data and allows to retrieve the files easily when needed.

Due to the increase in data, the need for File Systems is also increasing. Different file systems are available for different operating systems.

Features of File Systems:

1. Data is stored as isolated data files and entities.
2. It costs less as compared to a database.

Client Server Architecture -

The main goal of client server architecture is to define specialized servers with specific functionalities.
Client - It is host (computer) i.e. capable of receiving information or using a particular service from the host. It provides user interface capabilities and local processing of requests.

Servers - Server is a remote computer which provides information or access to particular services. It provides services to client machines.

So basically the Client requests what is needed and the server serves it as long as it's present in the database.

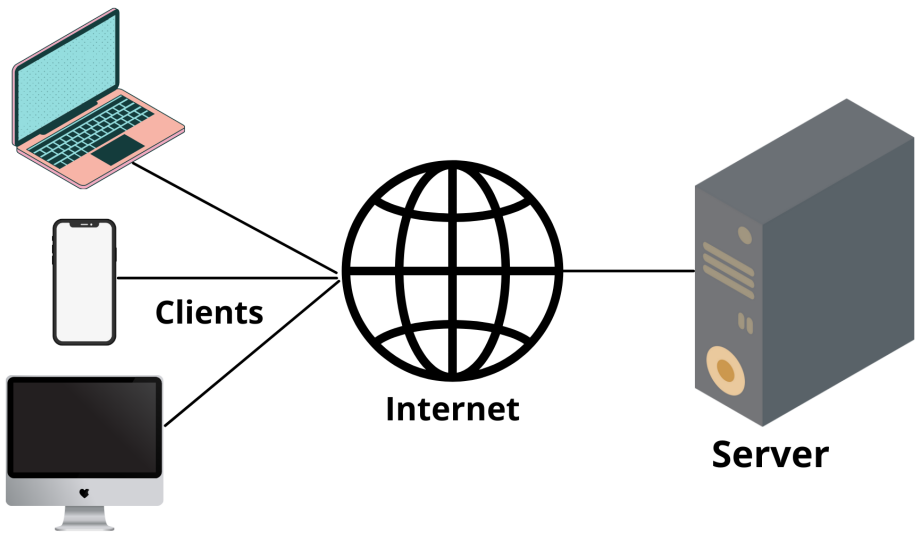


Figure: Client server architecture

Type of Client Server Architecture -

It could be further classified as

Tier 1 Architecture -

In this type of architecture Client, Server and Database all reside in a single machine.

Example of one tier architecture would be anytime you install a database in your system and access it to practice SQL queries.

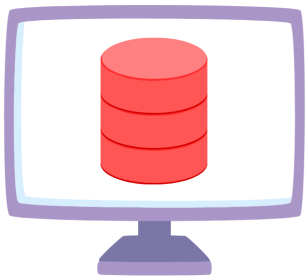


Figure: 1 tier architecture

Tier 2 Architecture -

In this architecture client reside in one machine and Server and database in another. It provides security to the DBMS as it is not exposed to the end-user directly. In this architecture multiple users can request from the same database server.

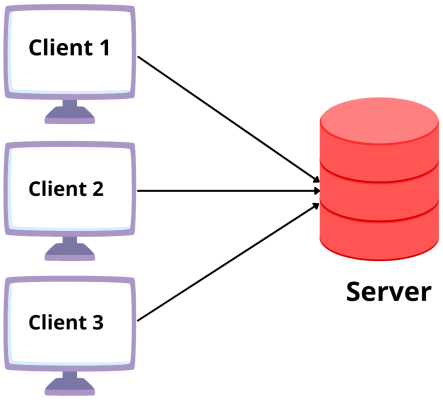


Figure: 2 Tier Architecture

Tier 3 Architecture -

In this architecture Client (User), Server and database all three reside in different machines. It is an extension of two tier architecture. Server resides between the client/user and database which is responsible for communicating the user's request to the DBMS system and sending the response from the DBMS to the user.

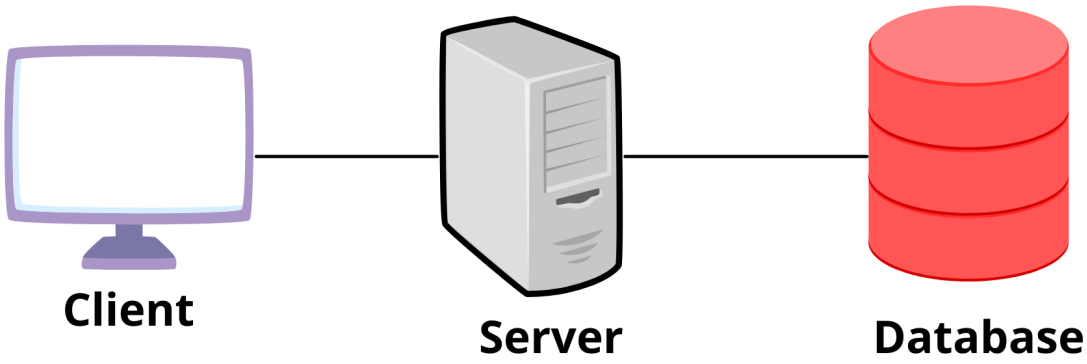


Figure: 3 Tier Architecture

Need for DBMS

Database System-

In DBMS we have users who write queries using some query language for example SQL (Structured Query Language).
These queries are processed by the DBMS (eg, MySQL, Oracle etc).
After queries are processed, based on results DBMS software accesses the stored data.

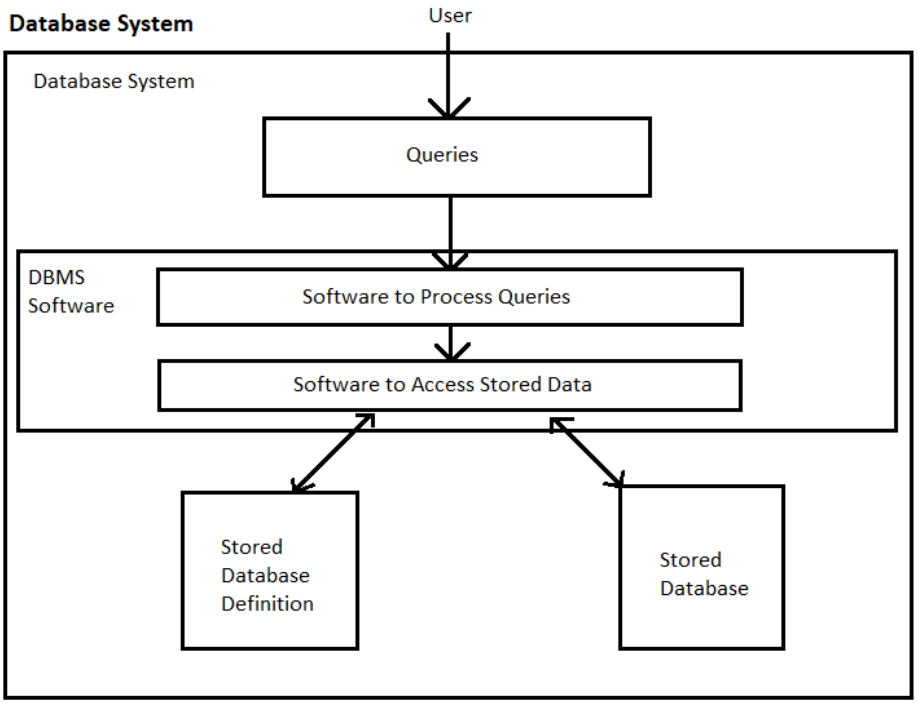


Figure : Database system

DBMS vs File System -

DBMS and File system both contain a collection of information and a set of programs by which we can access that data. Before DBMS we were using the file system approach.

Difference between DBMS vs File System -

DBMS	File System
DBMS is a software that helps in managing databases.	File systems help in managing files and organising them on a storage
The data is stored in databases.	The data is stored in files.
It coordinates both the physical and the logical access to the data.	It coordinates only the physical access.
It is designed to coordinate multiple users at the same time.	A file can be accessed by two programs concurrently only if both programs have read-only access to the file.
It gives an abstract view of data and hides the details.	The file system provides the detail of the data representation and storage of data.
Data inconsistency and redundancy could be reduced easily in DBMS.	It is hard to reduce data redundancy and inconsistency in the file system.
It provides a mechanism for data backup and data recovery.	If data is lost, we may not be able to recover it.

Advantages of DBMS -

- **Reduced redundancy**

In a traditional file system each user group maintains its own files which leads to data redundancy. This duplication of data or redundancy leads to wastage of storage space, and also inconsistency in data. In DBMS there are various methods(Normalization) by which we can remove data redundancy.

- **Data inconsistency**

Data inconsistency occurs when different versions of same data appear in different places. The chances of Data inconsistency get reduced a lot when the database is properly designed. Since in file system the different versions of same data may be present in different files and since there is no mechanism to handle it, file systems are more prone to data inconsistency.

- **Backup and Recovery**

The backup and recovery subsystem provides recovery options in case of system failure.
Example - You are transferring money from A's account to B's account, in case of system failure if the transaction got interrupted, the backup and recovery subsystem will try to resume the transaction or it'll restore it to previous state when transaction got started.

- **Integrity Constraints**

DBMS enforces integrity constraints to our data so that our database has only valid entries.
Example - If we are inserting the age of people in our database, then with the help of integrity constraints we'll make sure that it is a valid age i.e. it should not be negative.

- **Restricting Unauthorized Access**

DBMS allows multiple users with different access permissions to share a single database. DBMS provides a security and authorization subsystem to ensure this which is used by Database Administrator (DBA).

Data Models

Data model in DBMS is a concept that describes how data is stored, connected, accessed and updated in a database management system. It defines the logical design and structure of the database.

The importance of Data model is -

1. It provides a better understanding of the data and can be used by database developers to create a physical database.
2. Data Model helps to define the relational tables, primary and foreign keys and stored procedures.
3. A data model helps design the database at the conceptual, physical and logical levels.

Data Models are classified into three types –

- Conceptual Data Model
- Representational Data Model
- Physical Data Model

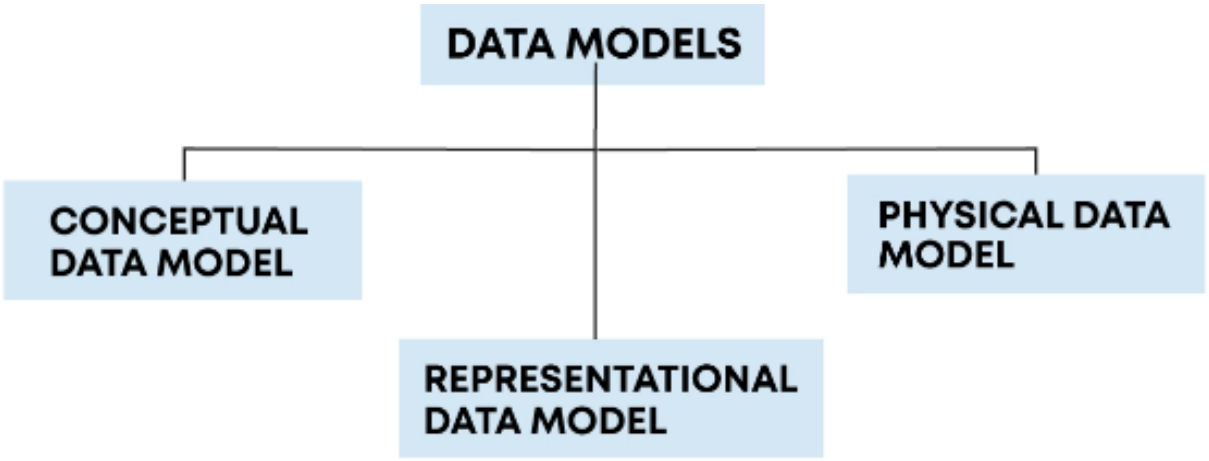


Figure: Classification of Data Models

Conceptual Data Model

Conceptual data model describes the database at a very high level

- It is used in the requirement gathering process and for representational purposes.
- By conceptual data model we get to know what we want to build and how our data will look like.
- It also helps to know what kind of database we should use.

One of the examples of conceptual data models is Entity Relationship Diagram.

Conceptual data models known as Domain models as they create a common structure for all stakeholders by establishing basic concepts and scope.

Representational Data Model

The Representation data model is used to represent only the logical part of the database.

- It focuses on the design part of the database.
- It shows only the part of information that you want to show at a specific level as it removes or hides the data for a specific kind of user.
- It represents how data is stored in a relational database.

A popular representational model is the Relational model.

Physical Data Model

Physical Data Model contains relationships between tables that address cardinality and nullability of the relationships.

- It describes a database-specific implementation of the data model.
- Physical data model shows all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships between tables.

Schemas

A database schema is like a blueprint or an architecture of exactly how our data will look. The schema doesn't hold the data itself, but instead it describes the data and how it may be related to other tables or models.

Schemas structures the unstructured data into structured data. Schemas are helpful in designing database management systems.

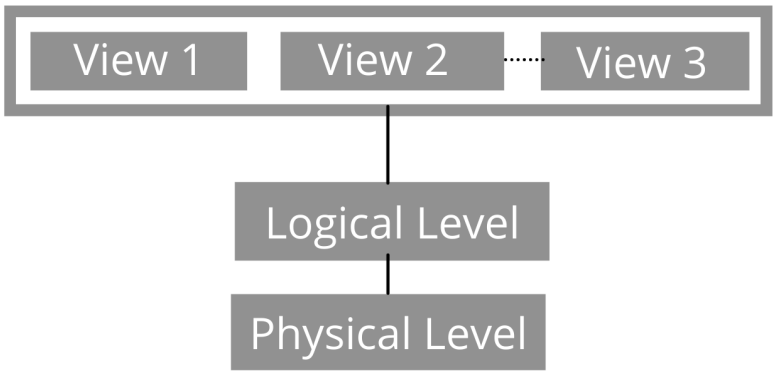


Figure: Schemas

DATABASE SCHEMA TYPES:

There are three main database schema types namely view, logical, physical database schema which define different parts of the schema:

Physical Schema- The physical database schema represents how the actual data is stored on disk storage. That is the actual code that will be used to create the structure of your database.

Logical schema- A logical database schema represents how the data is organized in tables. It explains how attributes from tables are linked to each other.For creating a logical schema, we use ER Model technique.

View Schema - the view schema describes the database design at view level which describes how the user interacts with the database

Advantages of Database Schemas:

1. A database schema can be easily transferred to another user.
2. It enables the transfer of database objects between schemas.

3. Data can be managed independent of physical storage.

Difference between Database and Database Schema:

DATABASE	DATABASE SCHEMA
The database contains interrelated data	Database schema is a structural view of data
Contains the data	Does not contain any data of its own
Data Changes	Schema does not change

Instances:

Instance of a database is defined as the data or collection of information stored in a database at a particular moment of time.

- Data changes frequently in the case of an instance.
- Whenever a new record or data is added in the database, data will be updated.
- It contains a snapshot of the database.

DataModels define the variable declaration in a table that belongs to a particular database. The value of variables which are defined at a particular moment of time are called instances.

Three Schema Architecture

The Three Schema Architecture tells how the data is represented in a database. The Three Schema Architecture is also known as three-level architecture or ANSI/ SPARC architecture.

This architecture is used to divide the database into 3 levels to separate the user and the physical database. This helps in providing abstraction to the users who may not require all the details.It basically hides the physical storage details from the user.

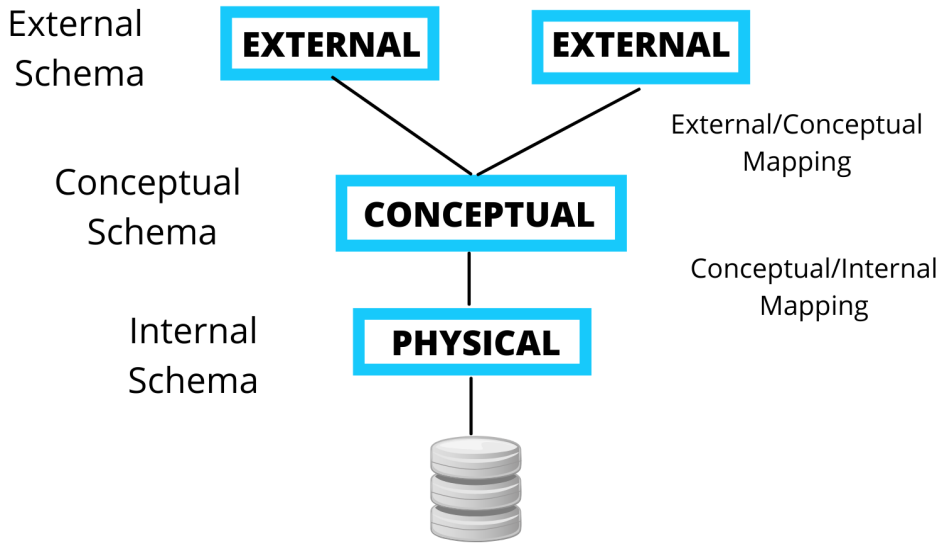


Figure: Three Schema Architecture

1. Physical Layer

The Physical Layer contains the internal Schema which defines the physical storage structure of the database. The internal schema is the low-level representation of the database. The internal Schema tells us what data is stored in the database and how?.

Major points related to physical layer are:

a. How data is stored in a database.
b. It shows the actual representation of the database.

2. Conceptual Layer

Conceptual Layer contains the conceptual Schema. It is designed to represent the organization of data as relations and records. If a change is to be made in the conceptual schema it should not affect the existing external schema.

Major points related to the Conceptual Layer are:

a. It describes the structure of Schema for the community or set users.
b. It hides the Physical Structure/Schema.
c. Representational Model is an example of Conceptual Schema.

3. External Layer

External layer is the highest level in the three level architecture and is also known as the view level. It is used as a view for a number of users due to this we can have multiple external levels. It only shows the relevant database content to the users in the form of views and hides the rest of the data. Same database is used to provide different sets of data to different sets of users.

Major points related to the External Layer are:

a. To provide access control.
b. It handles the way the users visualize the database according to their need.

Data Independence

Data Independence in DBMS is the property to modify the definition of a schema in a level without affecting the definition of schema in other levels.
There are two levels of Data Independence:

1. Physical Data Independence
2. Logical Data Independence

Physical Data Independence

Physical Data Independence is the property to modify the Physical Schema without changing the Logical and Conceptual level.

- Physical Data Independence can be used to change the storage device from one to another without affecting the Logical level.

- This allows us to provide a logical description without the need to specify any physical structures of the database.
- It also separates the conceptual level from the other internal levels.

Logical Data Independence

Logical Data Independence is the property to modify the Conceptual Schema without changing the external level.

- In Logical Data Independence ,any change done at the logical level the external view of the data remains unchanged.
- It is used to change the definition of data.

ER Models

Entity Relationship Model is the graphical representation of database design. It is a high level data model that describes data in terms of entities, attributes and relationships. It shows the relationship between entity sets.

ER Diagram is the blueprint of our database.

For Example:
If we have to design a Bank's Loan Database storing the customer's data as well the data related to loans that have been given to each customer.
It's ER-diagram with few of its attributes will look like:

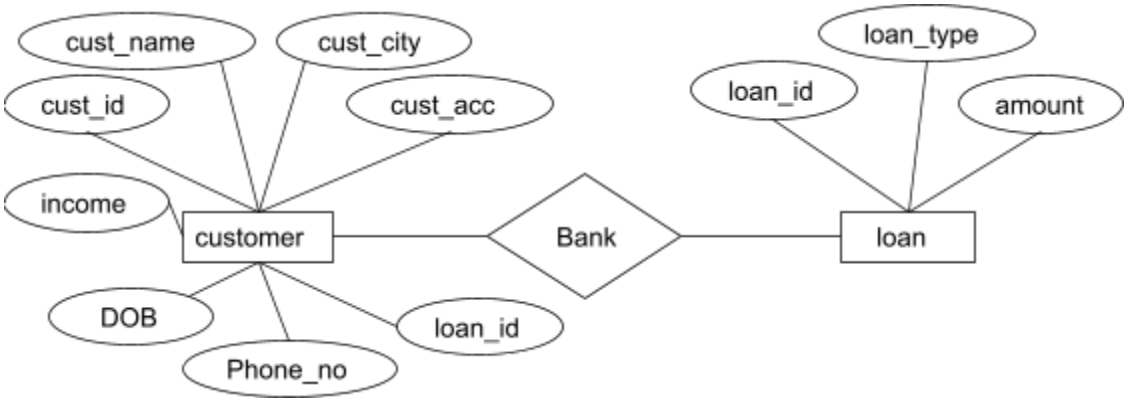


Figure: ER – Diagram 1

Keywords

1. **Entity**
   Entity is a real world object,class, person or place. Entities can be uniquely identified. It has a physical existence.
   It is represented by a rectangle, like in the above diagram customer and loan are the entities.

Example –
In ER Diagram -1 any customer who has taken a Loan from the bank could be an entity.

2. **Entity Set**
   Set of entities of the same type at a particular point of time is known as the entity set.
   Example –
   In ER Diagram -1 customer and loan are the entity sets..

3. **Attributes**
   Properties of the entity are known as attributes. Each attribute has some value. An entity can contain multiple attributes.
   They are represented by eclipse.
   In ER – Diagram 1 cust_id, cust_name, cust_city, cust_acc, dob, phone_no, loan_id and income are attributes of customer entity set.

Example

In the above ER Diagram - 1 customer and loan are entities.

- cust_id, cust_name, cust_city, cust_acc, dob, phone_no, loan_id and income are attributes of customer entity.
- loan_id, loan_type, amount are attributes of the loan entity.
- Bank is a relationship set between customer and loan, as a customer of the bank will take loan from his/her bank only.

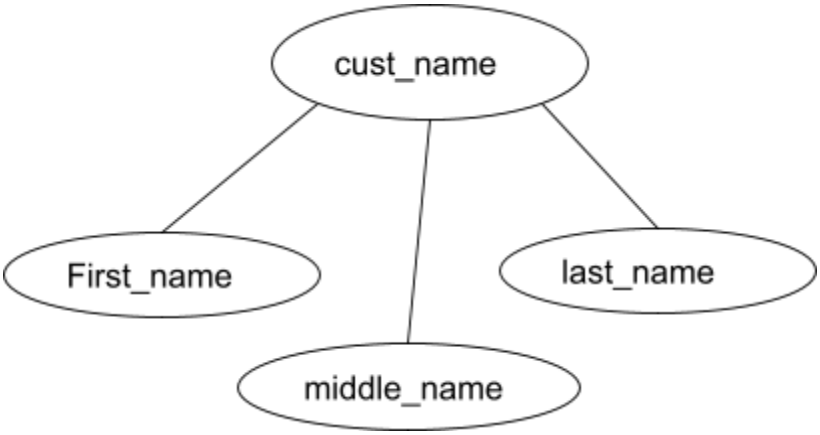
Types of attributes

Based on Composition

1. **Simple Attributes -** If an attribute cannot be divided further then it is known as a simple attribute.
   Example – In ER – Diagram 1 cust_acc is a simple attribute.

2. **Composite Attributes -** If any attribute can be divided into further parts then it is known as a composite attribute.

Example – In ER – Diagram 1 cust_name is a composite attribute because it is further divided into Fist_name, Middle_name and Last_name.



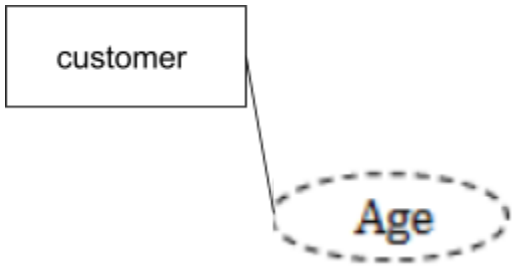
Based on count of values that can be stored

1. **Single Valued Attribute -** If a particular attribute has only one value then it is known as a single valued attribute.
   Example – cust_id is a single valued attribute because a customer could have only one name.
2. **Multi Valued Attribute -** If an attribute has more than one value then it is known as a multivalued attribute.
   It is represented by a double oval.
   Example – Phone_no can be a multivalued attribute because a customer can have multiple contact numbers.

Based on how the attribute value can be stored

1. **Stored Attribute -** The initial information which we save in our database is known as a stored attribute.
   Example – In ER diagram 1 DOB is a stored attribute.

2. **Derived Attribute -** The value which we can derive from stored attributes is known as derived attribute.
   Example - Age is a derived attribute because it can be derived from DOB.

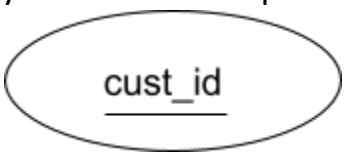


3. **Complex Attribute -** If an attribute has multivalued and composite components in it then it is called complex attribute.
   Example - In a person table a person can have many residents i.e. multivalued and each resident is a composite attribute which can be further divided into house number and city i.e. composite. So address is a complex attribute.

Null Values - It represents something which is unknown.
Example - If we have to insert the income of a customer but it is unknown then we may insert NULL there.

Keys in ER diagram

An attribute or set of attributes which can uniquely identify a record is known as keys.
Example - In the customer table, cust_id is a key attribute because it can identify each customer record uniquely and it can be represented as

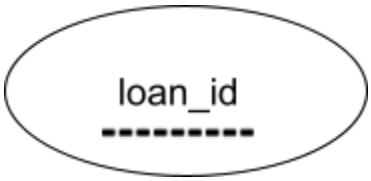


Types of Keys in ER Model –

1. **Primary Key –** This is an attribute or set of attributes that can uniquely identify each entity in the entity set. In ER – diagram 1 cust_id can be the primary key in the customer table.

2. **Foreign Key –** Whenever there is some relationship between two entities there must be some common attribute between them. This common attribute must be the primary key of an entity set and will become foreign key of another entity set.
   In ER diagram 1 loan_id is foreign key in the customer entity set and the discriminator or Partial key(won't call it a Primary key) in the loan entity set because it is maintaining the relationship between student and course entity set.

Note: Primary key for Loan entity here will be: (cust_id, Loan_id.)
(Primary key of a weak entity set = It's Partial Key/Discriminator + Primary key of strong entity set)
These discriminators or partial key is represented in ER-Diagram with a dashed underlining.



Strong and Weak Entity Set

- An entity set which is not dependent on any other entity set in the schema is known as a strong entity set i.e. a strong entity set will always have a primary key. It is represented with a rectangle.

- A weak entity set is usually dependent on a strong entity set to ensure its existence and it does not have any primary key rather contains a discriminator or a partial key to differentiate between the records present in the weak entity set table. It is represented with a double rectangle.
  It needs to have participation.

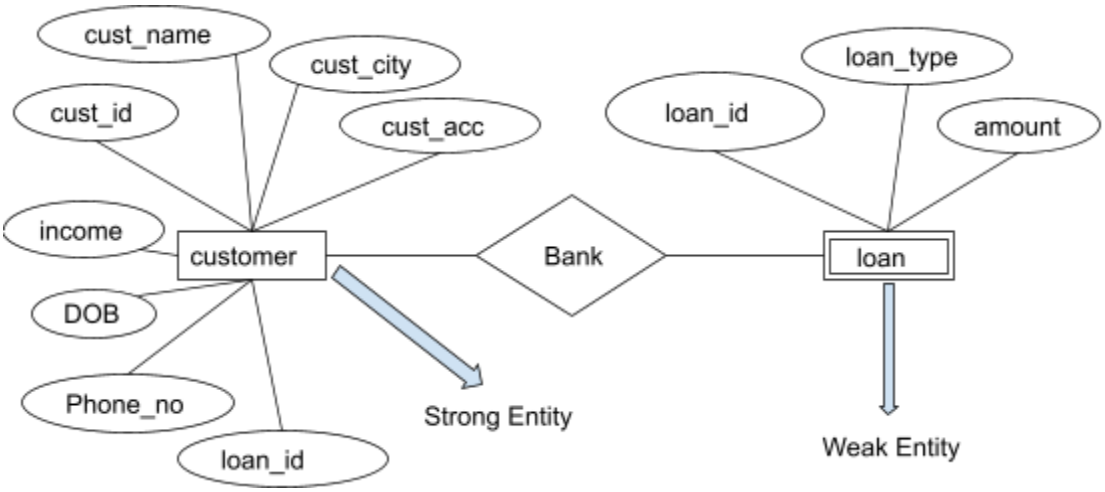
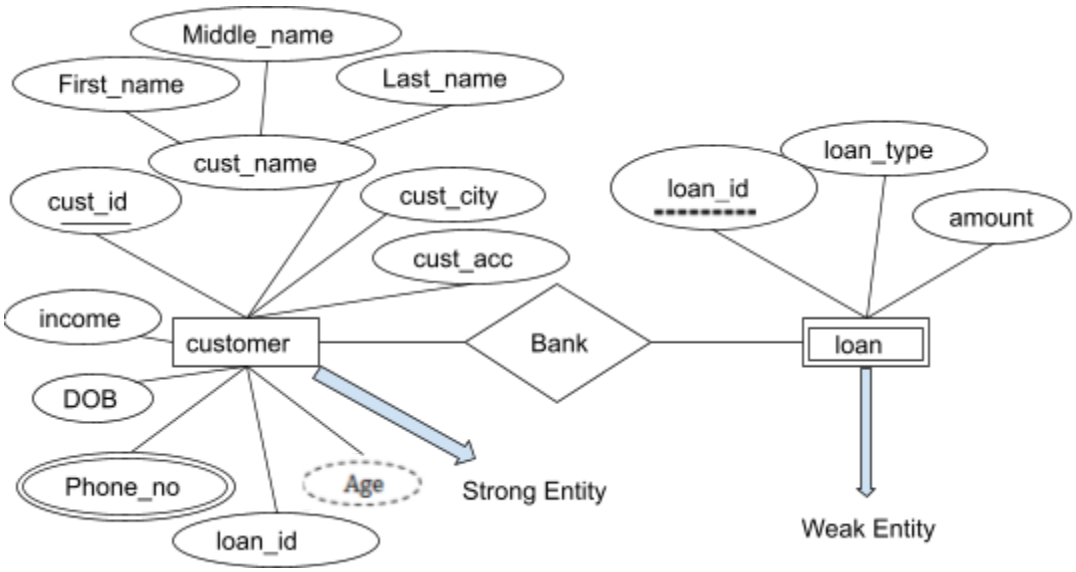


Figure: ER – Diagram 2

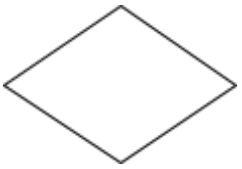
In above ER Diagram customer is strong entity set and Loan is weak entity set because without a customer there is no existence of loan and loan entity set has total participation i.e. every loan must have at least one customer.

The Final ER-Diagram we get by defining all the things we learnt till now,



Relationships

It defines how two or more entities are connected with each other. It is an association among two or more entities.
It is represented by a diamond operator in the ER diagram.

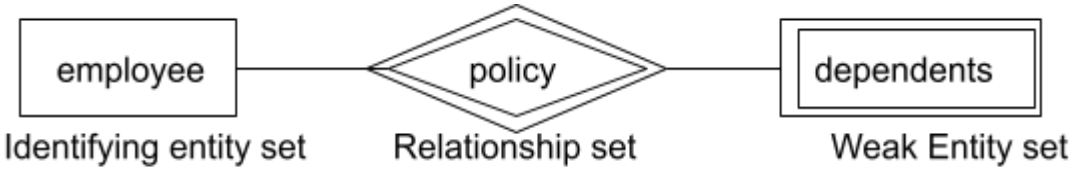


It is of two types:

1. **Strong Relationship**
   The relationship between two independent entities is a strong relationship.
2. **Weak Relationship**
   When a relationship exists between a weak entity and its owner entity, then it is called a weak relationship.

Example
Assume we have an Insurance database and in that we have two entities Employee and dependents. In this both entities will be connected with **Policy** relationships.

Treat policy as a special relationship that provides extra information (an attribute like 'id') required to identify entities in dependents uniquely
An entity in employee has a **policy** covering his dependents



A weak entity set is one whose existence is dependent on another entity, called its identifying entity set (or owner entity set)

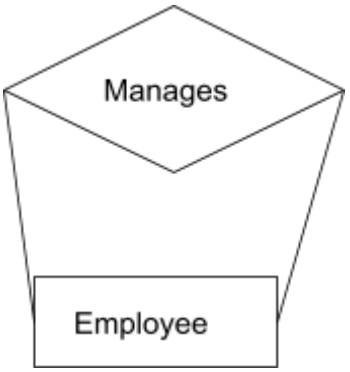
In ER model different types of geometrical shapes are used to represent different things.

Entity Sets	Rectangle
Weak Entity Sets	Double Rectangle
Links attributes to entity set and entity sets to relationship set	Lines
Total participation of an entity in a relationship set	Double Lines
Relationship	Diamond
Weak Relationship	Double Diamond
Attributes	Ellipse
Multivalued attributes	Double Ellipse
Derived attributes	Dashed Ellipse

Degree of Relationship

Number of entities participating in a single relationship is known as the degree of that relationship.
Relationships can be classified according to how many entities are participating into a particular relationship.

1. **Unary Relationship**
   It exists when only one entity participates in a relationship.
   Example

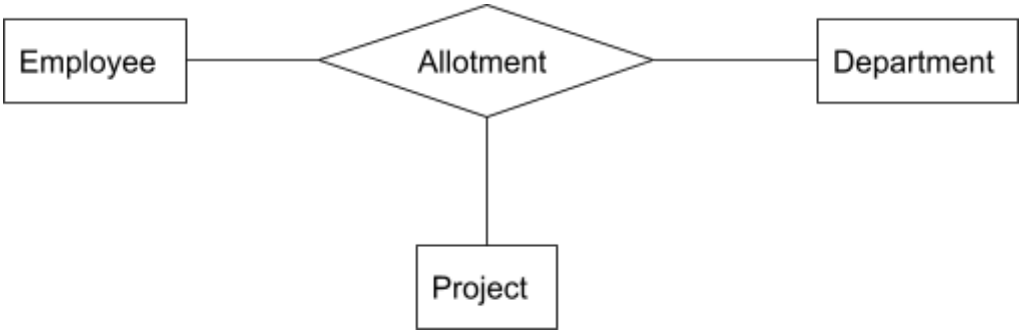


In the above example, an employee could be a manager also, so the manager manages other employees, and all types of employees are in a single entity. Therefore the relationship here is unary relationship.

2. **Binary Relationship**
   When two entities participate in a relationship then it is known as binary relationship. (degree two)
   Example

In the above example ‘Enrolled in’ relationship is connected only with two entities, therefore, ‘Enrolled in’ relationship is binary.

3. **Ternary Relationship**
   It exists when there is an association among three entities. (degree three)



In the above example Allotment is connected with three entities, therefore the relationship here is ternary.

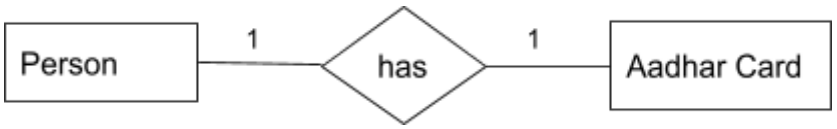
Note: Relationships between more than two entity sets are rare. Most relationships are binary.

Cardinality Ratio

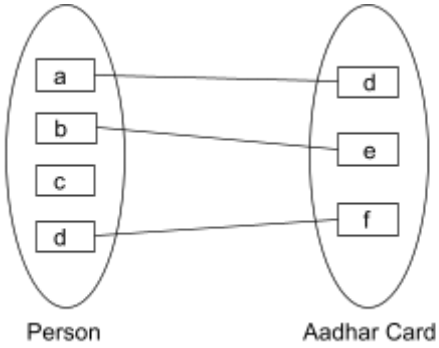
In binary relationship cardinality ratio represents how many times the entity/instance of an entity set can participate in a relationship.
OR
It refers to the number of entities to which another entity can be associated through a relationship set.

For Binary Relationship set is of four types: –

1. **One to One –**
   If only one instance of an entity is associated with only one instance of another entity then it is one to one relationship.

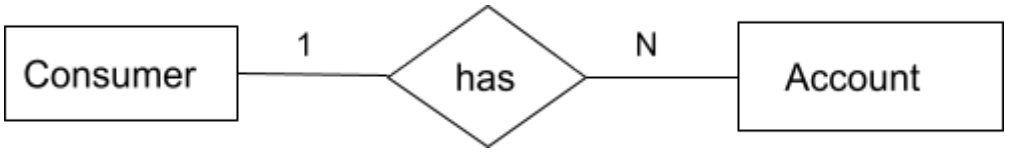


   In the above example one person could only have one Aadhar Card, therefore it is one to one relation.
   To further visualise it:-

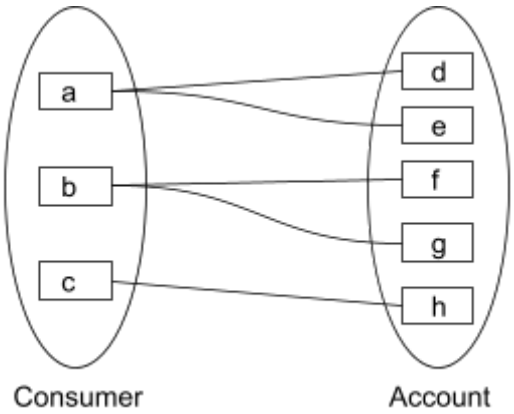


   An entity in Person is associated with at most one entity in Aadhar Card and an entity in Aadhar Card is associated with at most one entity in Person.

2. **One to Many –**
   If a single instance of an entity set is associated with more than one instance of another entity then it is one to many relationships.

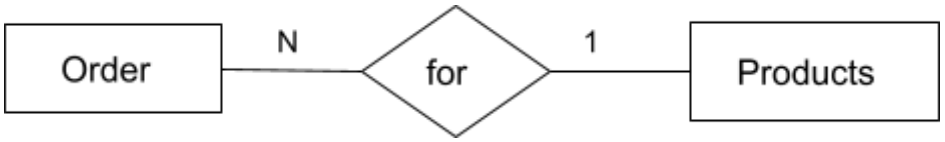


In the above example one customer can have multiple bank accounts. So the relationship here is one to many.
To further visualise it:-

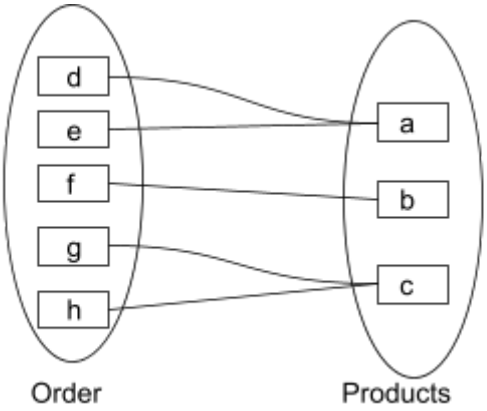


An entity in Consumer is associated with zero or more entities in Account. An entity in Account is associated with at most one entity in Consumer.

3. **Many to One –**
   If more than one instance of an entity is associated with a single instance of another entity then it is called many to one relationship.



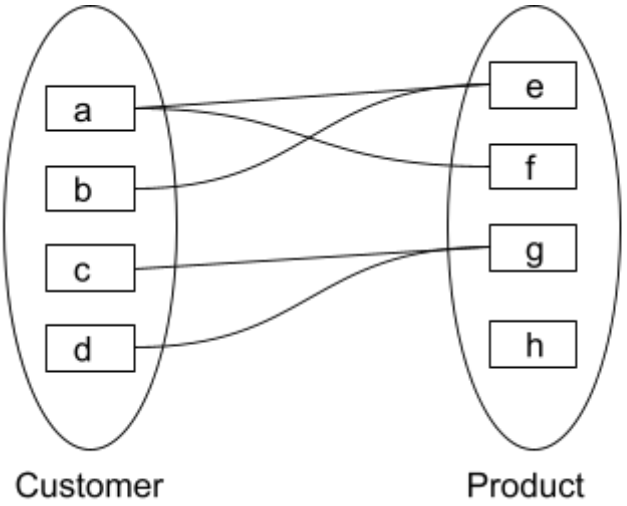
In the above example one product could have multiple orders, so that relationship here is many to one.
To further visualise it:-



An entity in Order is associated with at most one entity in Product. An entity in Product is associated with one or more entities in Order.

4. **Many to Many –**
   If more than one instance of an entity set is associated with more than one instance of another entity then it is many to many relationship.

In the above example one customer can place many orders and one product can be ordered by multiple customers. Therefore the relationship here is many to many.
To further visualise it:

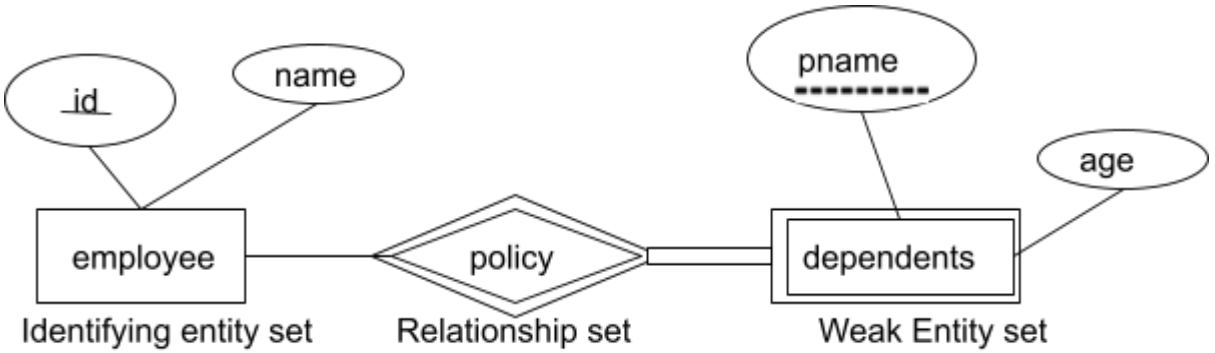


An entity in Customer is associated with one or more entities in Product. An entity in Product is associated with zero or more entities in Customer.

Participation Constraints

It is also called minimum cardinality constraint.

1. **Partial Participation –**
   - It specifies that not all entities are involved in the relationship instance.
   - It is shown by using a single line between the entity set and relationship set in the ER diagram.
2. **Total Participation –**
   - It specifies that each entity must be involved in at least one relationship instance, therefore it is also called as mandatory participation.
   - It is shown by using a double line between the entity set and relationship set



In In this example, An entity in employee has a policy covering his dependents
Here, a given employee's dependents are likely to have unique names.
And In dependents, each entity is distinct but enough attributes are not present to identify a dependent uniquely.
So here we Treat policy as a special relationship that provides extra information (id) required to identify entities in dependents uniquely.
Note1: pname could be made unique – but conceptually, a dependent is still dependent on an employee, so not a good way to model

Note2: Here, id in employee is a primary key and pname in dependents is a partial key/discriminator.
Therefore, dependents have total participation because it can not exist without an employee entity and the employee has partial participation.

Some Terminologies related to ER models

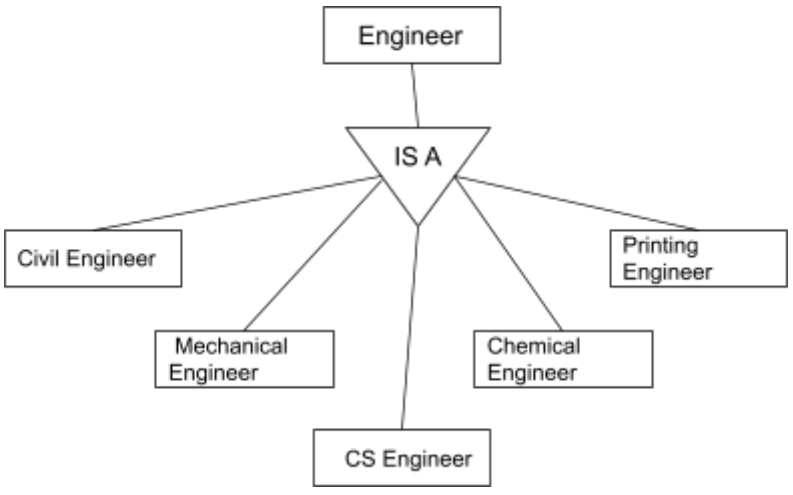
When complexity of data increases, it becomes harder to use the traditional ER model for data modeling.
So, there are some enhancements in the traditional ER model to make it able to handle complex applications better. We also may call it Enhanced ER-model.
These are -

- Specialization
- Generalization
- Aggregation

1. Specialization

In layman language, Specialization is a process of mastering a specific domain (refer to example given below).
But, with respect to ER Model, specialization is the procedure to splitting up the entities into further sub entities on the basis of their functionalities, specialities and features.
These sub-designation of entities are distinctive from other entities in the set.

To define it in a more technical way, Specialisation is a process which follows a Top-to-Down approach.
In this a higher-level entity is specialized in two or more low level entities. (we design sub-entities for a given entity)



Initially there was an Engineer named single entity. But sometimes we don't need all the engineers present in the world for necessary work. We need Someone specialized in a specific domain.For example, a CS engineer can not be asked to build a bridge or building. So Engineers can be further divided into Civil Engineer, Mechanical Engineer, CS engineer, Chemical Engineer, Printing Engineer and many more.

Why Specialization?

One of the features of Specialization in a data model is that certain attributes may only be applicable to a few entities of the Parent Set but not to all. In order to group the entities to which these attributes are applicable, a sub-group is defined. A large number of the attributes of the members of the sub-group may still be shared with other members of the Parent Entity set.

Note: In Specialisation, schema size increases.

2. Generalization

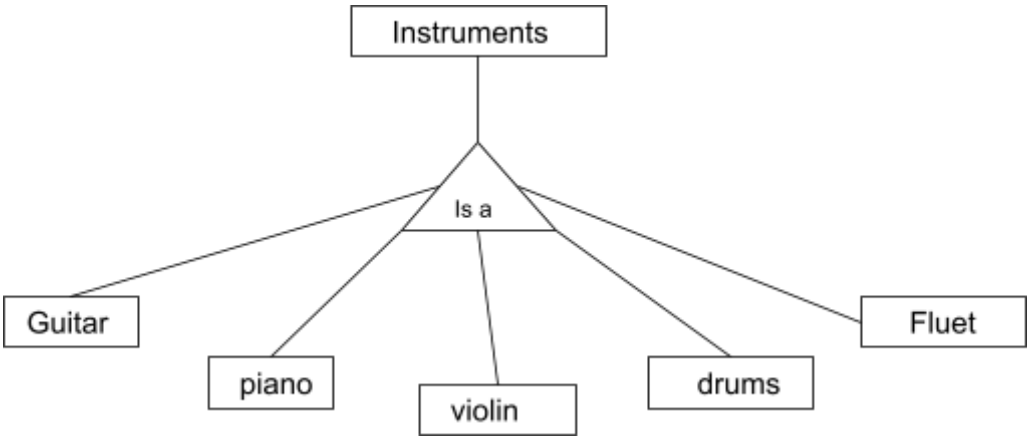
In simple terms, Generalization is the reverse process of specialization. In generalization, the sub entities are combined together resulting in the formation of a parent entity set on the basis of some common features in such a way that the new entity thus formed contains all the features of the sub entities.

Generalization is a process which follows a Bottom-to-Up approach.

So to define it in technical terms, two or more low-level entities are synthesised to make higher-level entity on the basis of common attributes.

This process involves establishing relationship among tables, by analyzing the attributes belonging to the entities and then making an informed decision to form a high level entity.

The attributes for this high level entity are those attributes which are believed to be most required for each child-entity.



In the diagram above entities like guitar, piano, violin, drums, flute are generalized into a single parent entity, named Instrument.
This generalization could happen because of the common features of all the entities.

Why Generalization?
It optimizes the database, by making it simpler.
Data repetition is also avoided, as now there won't be no need to mention the same attributes or data in different tables/relation.

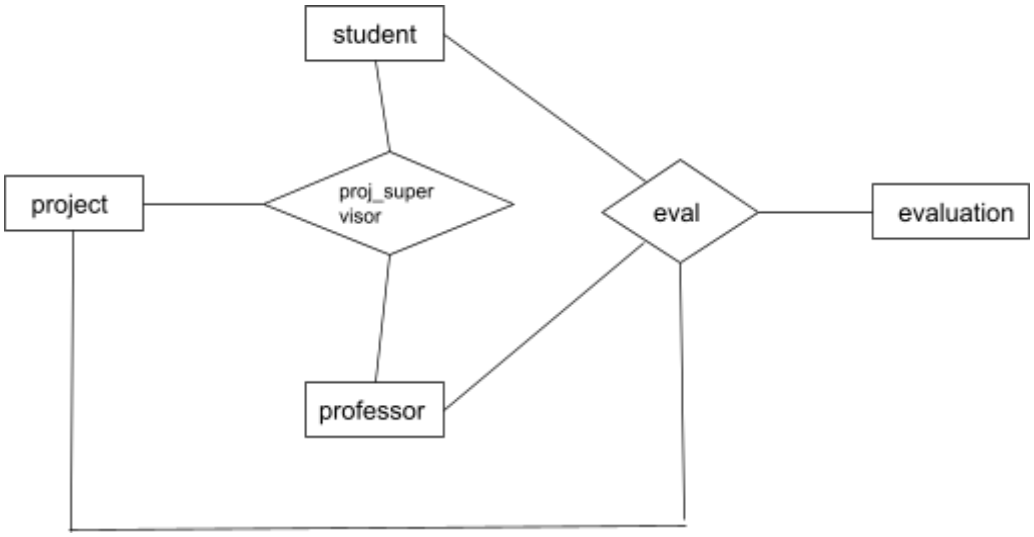
Note: In Generalization, schema size shrinks.

* Both Generalisation and Specialization follow attribute inheritance (i.e. The attributes of the higher-level entity sets are said to be inherited by the lower-level entity sets, similarly vice-versa), they also follow participation inheritance (i.e. if a parent entity participates in a relationship set then its sub-entities will also participate in that relationship set.)

3. Aggregation

Aggregation is used when we need to express a relationship among relationships. It is like abstraction through which relationships are treated as higher-level entities.
In this multiple entities are considered as a single entity and again this single entity has a relationship with another entity. It treats relationships as an abstract entity.

Let’s understand with an example:
In our college times, many of us take up a project under a professor during our summer time or whenever an individual wants, so let’s Consider a ternary relationship proj_supervisor, suppose now the instructor/professor needs to evaluate the projects students have been doing under them.
So we record those evaluations.
Here’s an ER-Model for this:-

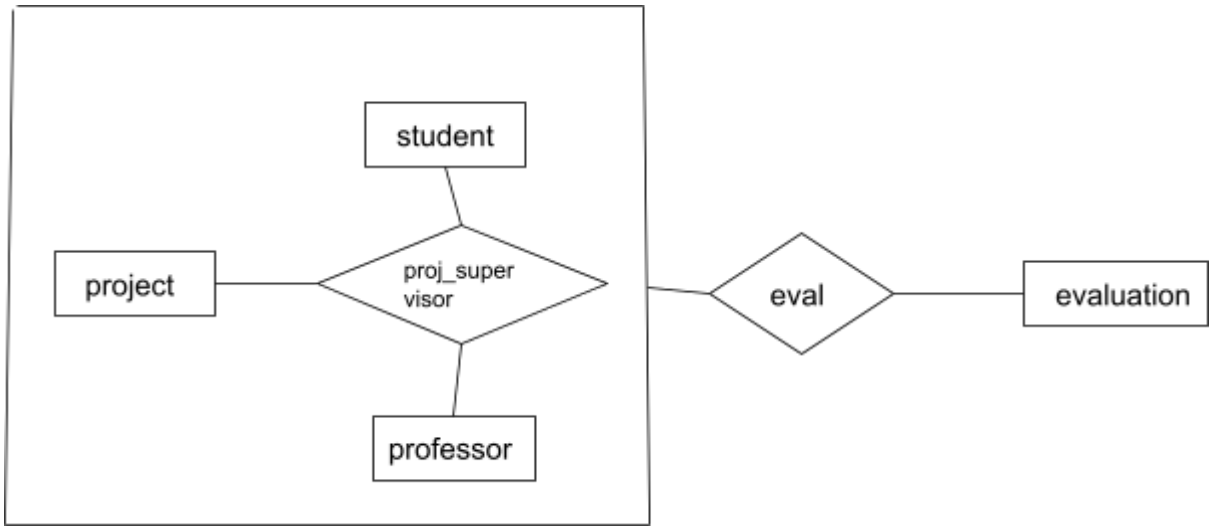


Relationship sets eval and proj_supervisor represent overlapping information.
Every eval relationship corresponds to a proj_supervisor a relationship that is evaluation can only happen if a student is allocated a project under some professor.
However, some proj_supervisor relationships may not correspond to any eval relationships.
I.e. there might not be any student to evaluate, though that student has been allocated a project under some professor. So we can’t discard the proj_supervisor relationship.

Now, We need to have a relationship between relationships.

For solving this problem, we use aggregation.
As, A student is supervised by a particular professor on a particular project. A student, professor, project combination may have an associated evaluation.
Basically, we treat their combination as one entity.

Enhanced ER-model after aggregation:-



Why Aggregation?

Due to ER Model's limitation to represent relationships among relationships.

Although if we try to represent ternary relationships using ER-diagram it might result in a lot of redundancies.

Hence, aggregation is used which is redundancy free.

Relational Model

It was introduced by Ted Codd in 1970. This model was all about organizing the data in the form of tables in row and column. This model has all the properties and capabilities required to process data with efficiency.

The relation in Relational Model is different from relations in ER model, in relational model relation refers to tables in which data is organized in the form of rows and columns.

When we get a problem statement and we have to solve it using databases:

- first we convert it into an ER model for better visualization of the problem statement.
- then we convert it into a relational model and
- the relational model is implemented using the Relational database management system (RDBMS).

Example –

Here ID, Name, Salary, dept_name are attributes for the table/relation named Professors.

ID	Name	Salary	dept_name
4784545	Phunsuk Wangdu	95000	Mechanics
6474736	Farhan Qureshi	90000	Anatomy
4353463	Karan Malhotra	80000	Ortho

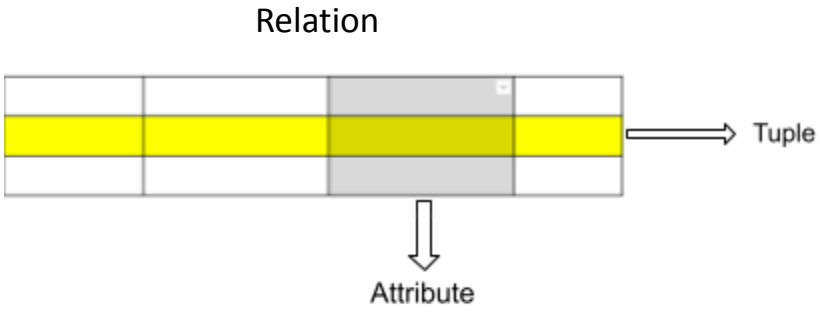
Records present in the above relations are called tuple.

Eg: Below is one of the tuple from above table.

ID	Name	Salalry	dept_name
6474736	Farhan Qureshi	90000	Anatomy

Relational Database Management System (RDBMS) –

A RDBMS is a type of database that stores and provides access to data points that are related to one another. RDBMS stores data in tables or relations. It is based on a relational model i.e. each row is a unique record that is called **tuple**. Column of the table stores attributes. Oracle, MySQL etc are some examples of RDBMS software.



Relation

In relational Model relations are saved in the format of tables. This table stores relations among entities. Rows of tables represent records and columns of table represent attributes.

Tuple

A single row or record of the table is known as tuple.

Relation Instance

A row or tuple of a table is called a relationship instance.

Relation schema

Relational schema defines design and structure of the relation and also the variable declarations in tables. It consists of the relation name, set of attributes, column name.

The relationship schema for above table/relation Professors would be –

Professors (ID, Name, Salary,dept_name)

Degree of table

It is the number of columns in a given relation.

For the above relation, degree = 4.

Cardinality

The total number of rows in a given relation is the cardinality of that relation.

For the above relation, cardinality is 3.

Relation Key

Relation key is a set of attributes of a table which can uniquely identify each row.

Important properties of table

- The values have to be atomic(they can't be broken down The values in each column has to be of same data types further)
- Each row has to be unique in a table. To avoid Redundancy, otherwise will go against the concepts of RDBMS.
- The sequence of column and row is insignificant.
- Each attribute must be unique.

Keys

Super Key

It is a set of one or more attributes which can uniquely identify the row of a table.

In the table shown below–

Stu_id	name	cou_id	Address	phone
496	Kuldeep Daga	101	781 Ashok Nagar, Indore	943523345
267	Ojasv Singh	671	50 M Sector-4,Chandigarh	999436436
367	Sampriti Patel	241	107, GS road, Guwahati	967543322

1. Stu_id,
2. Stu_id, name
3. Stu_id, name, cou_id,
4. Stu_id,name, cou_id, Address
5. Stu_id, name, cou_id, Address, phone

These all could be super key because they can uniquely identify each record of the table.

Candidate Key

It is a minimal subset of super keys. It contains no redundant attribute.As already mentioned it is a minimal set of super keys. Hence, it is selected from the set of super keys given that those selected keys DO NOT have any redundant attributes.

Candidate Key value should not be null.

From the above super key,

Stu_id is candidate key because it can alone identify the other attributes uniquely.

Note: Phone is not considered to be a candidate key, as it can hold NULL value. In case a student doesn't have a phone number.

Primary Key

It is a unique identifier which helps us to identify each and every tuple uniquely. No two rows have the same value for primary key Attribute and the primary key cannot be null. The primary key in the table cannot be changed. It is selected out of all the candidate keys by database admin.

It could be further divided into **Composite Key** and **Compound Key**.

- Primary key which is formed using at least two attributes is known as composite key.

Stu_id	name	cou_id	Address	phone
496	Kuldeep Daga	101	781 Ashok Nagar, Indore	943523345
267	Ojasv Singh	671	50 M Sector-4,Chandigarh	999436436
367	Sampriti Patel	241	107, GS road, Guwahati	967543322

In this table name and address together can uniquely identify the row so it could be a primary key. As there are two attributes together making the primary key, so it is a composite primary key.

- Primary keys which are formed using two foreign keys which have been referenced in some other table.

Stu_id	name	Address	phone
496	Kuldeep Daga	781 Ashok Nagar, Indore	943523345
267	Ojasv Singh	50 M Sector-4,Chandigarh	999436436
367	Sampriti Patel	107, GS road, Guwahati	967543322

Table/Relation - Enrollment:-

Stu_id	cou_id	Enr_num
496	101	0978078068
267	671	9679543000
367	241	0808676542

Table/Relation - Courses:-

cou_id	Stu_id	Cou_name	Enr_num
101	496	physiology	0978078068
671	267	music	9679543000
241	367	Machine Learning	0808676542

In the table/relation Enrollment ,**stud_id is a foreign key** which is referring to the Student relation where stud_id is a primary key.

Similarly, **cou_id is also a foreign key** which is referring to the Courses relation where cou_id is a primary key.

Stud_id and cou_id together is the primary key in the Enrollment relation which are individually foreign keys.

Hence (Stud_id, cou_id) is the compound primary key in the Enrollment Relation.

Alternate Key

All the candidate keys except the primary key are known as alternate keys.

Foreign Key

It creates a relationship between two tables. It is also used to maintain data integrity consistency.

Relation Student:-

Stu_id	name	cou_id	Address	phone
496	Kuldeep Daga	101	781 Ashok Nagar, Indore	943523345
267	Ojasv Singh	671	50 M Sector-4,Chandigarh	999436436
367	Sampriti Patel	241	107, GS road, Guwahati	967543322

Relation Courses:-

cou_id	Stu_id	Cou_name	Enr_num
101	496	physiology	0978078068
671	267	music	9679543000
241	367	Machine Learning	0808676542

In this example **cou_id is foreign key** in the Relation Student because it is referencing the primary key of the Courses table.

Surrogate Key

To uniquely identify the data, generally we give a surrogate key as an integer. It is a virtual key with virtual or no actual reason.

Example

Let’s say there is a rating system on your website. You’re storing name and rating corresponding to that user but there is no unique key in your system.

There is no primary key in our current table because names can be duplicate and also rating too.

To identify each row uniquely we can create a surrogate key as rating_id.

Rating_Id	Customer_Name	Rating
1	Akash	5
2	Anand	4

Integrity Constraints

Integrity constraints are rules that should be applied on database columns to ensure the validity of data. By Guarding against accidental damage to the database.

Integrity constraints make sure that the data insertion, updating and other processes are performed in such a way that the integrity of data is not affected.

With all the authorised changes to the database happening, integrity constraints ensure that there is no loss of data consistency.

All the relations in a Relational Database Model need to follow some rules or constraints to maintain the integrity of data, these constraints are called Relational Integrity Constraints.

There are three type of integrity constraints –

1. Domain Constraints
2. Entity Constraints
3. Referential Constraints

Domain Constraints

The domain integrity constraints restrict the value in the particular attributes of a relation.

Therefore, if in an attribute we want to take email as input, then for ensuring that email is valid, we can use domain constraints.

Before inserting email in the table, checks can be performed for validation.

Let’s take a different example:

We need to ensure that semester student is registering for is one of fall, winter, spring or summer:

Schema design for the Relation section:-

```
CREATE TABLE section (
course_id VARCHAR(8),
sec_id VARCHAR(8),
```

```
semester VARCHAR(6),
year NUMERIC(4,0),
building VARCHAR(15),
room_number VARCHAR(7),
time_slot_id VARCHAR(4),
primary key (course_id, sec_id, semester, year),
check (semester in ('Fall', 'Winter', 'Spring', 'Summer'))
);
```

In purple we have marked the check(P). 'P' is the predicate on which checks are performed.

Entity Constraints

- It puts constraints on Primary key i.e. primary should be unique and does not have NULL value.
- This constraint ensures that every row is uniquely identified in our table by ensuring the first constraint stated above.

Referential Constraints

- It is specified between two relations and helps maintain consistency among the tuples of two relations.
- It ensures that if a value appears in one relation for a given set of attributes also should also appear for a certain set of attributes in another relation.
- Example: If "Prosthodontics" is a department name appearing in one of the tuples in the instructor relation, then there exists a tuple in the department relation for "Prosthodontics".
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2 then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Relation Project:

ProjectID	EmpID	ProjectName
100	1	pro_1
200	2	pro_2
300	3	pro_3
400	5	pro_4

Table Employee:

EmpID	EmpFname	EmpLname	Age	EmailID
1	Riya	Khanna	21	riya@abc.com
2	Sahil	Kumar	32	sahil@abc.com
3	Vishwas	Aanand	24	vishwas@abc.com
4	Harleen	Kaur	27	harleen@abc.com

Now in the above example, EmpId in Relation Project is a Foreign key, and is a primary key in Relation Employee. (marked in blue)
EmpID = 5 (marked as red) in the foreign key of the relation Project is not allowed. Since, EmpID = 5 is not defined in the primary key of the relation Employee.
Hence, leading to Violation of Referential Constraints.

Designing a database

Problem statement – Design order management system that has a database which captures all the orders of all the customers.

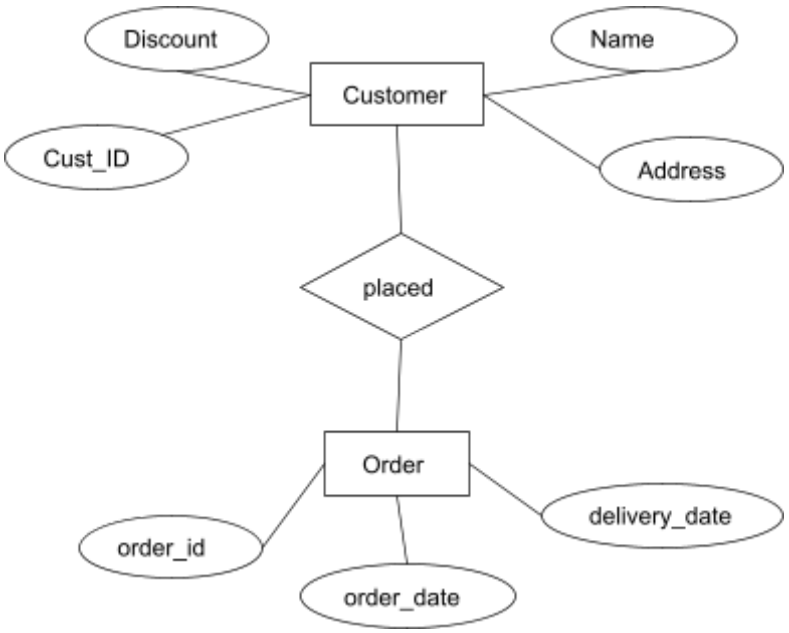
Steps

1. Come up with the idea
2. Convert it to E/R diagram
3. Relational Model
4. Relational Database

ER Diagram

- First we'll try to figure out all the entities.
- There will be two entities: customer and order. The relationship between the entity's customer and order is placed.
- Now we'll try to figure out attributes of both the entities.
- The attributes of the customer could be Customer Id, Name, Address, Discount provided to that customer.
- Attributes of order could be Order_ID, Delivery_Date, Order_Date.

Now after figuring out entities, attributes and relationship between entities, we're good to proceed for the ER diagram.



After that we can create a relational table according to ER Model.

Relation Customer:

Cust_id	Name	Address	Discount	Order_id
8495	Uday	97-B Model Town, Jalandhar, Punjab	30%	323134
4564	Reet	813 Aadarsh villa, Nokha, Rajasthan	55%	523623

Relation Order:

Order_id	Order_date	Delivery_date	Cust_id
323134	25/08/2021	27/08/2021	8495
523623	02/06/2021	05/06/2021	4564

Note: for the above example, In each relation a foreign key is marked with blue and primary is marked with red.

Relational Algebra

It is a procedural query language. It is a step by step result of arriving at a definite result. It collects records of relations as input and gives occurrences of relations as output. Relational Algebra uses different operators to perform queries. The operator may be either unary or binary.

Let us assume we have three tables –

Customer

Cust_Id	Cust_Name	Address
1	Indu	Huda, Panipat
2	Piyush	Jaipur
3	Raj	Noida

Product1

Prod_ID	Prod_Name	Price
B3	Tshirt	505
A1	Shoes	1000
B1	Kurta	500
N3	Watch	2000

Product2

Prod_ID	Prod_Name	Price
B3	Tshirt	505
Z1	Denim	1500

Purchase

Purchase_ID	Cust_ID	Prod_ID
190	1	A1
102	3	N3

Selection (σ)

It is used to select some attributes or some tuples from the table.

Example

$$\sigma_{\text{Cust\_Name="Indu"}} (\text{Customer})$$

It will select a customer whose name is Indu but it'll not display.

Note: This Select is DIFFERENT from MySQL SELECT.

Projection (π)

It is used to print / project attributes or some tuple from the table. Projection removes duplicate data.

Example

$$\prod_{\text{Cust\_Name="Indu"}} (\text{Customer})$$

Output –

Cust_Id	Cust_Name	Address
1	Indu	Huda, Panipat

Union (U)

It performs union between two given relations. Suppose there are two tuples X and Y, the union operation would contain all the tuples that are either in X or Y or both in X and Y. It removes duplicate tuples.

Example

$$\prod_{\text{Prod\_name}} (\text{Product1}) \cup \prod_{\text{Prod\_name}} (\text{Product2})$$

Output –

Prod_Name
Tshirt
Shoes
Kurta
Watch
Denim

Intersection (∩)

It performs an intersection between two given relations. Suppose you have two tuples X and Y, then X intersection Y will contain only those tuples which are common in both X and Y.

Example

$$\prod_{\text{Prod\_name}} (\text{Product1}) \cap \prod_{\text{Prod\_name}} (\text{Product2})$$

Output –

Prod_Name
Tshirt

Set Difference (–)

If you have two tuples X and Y then X – Y contains all the tuples which are present in X but not in Y.

Example

$$\prod_{\text{Prod\_name}} (\text{Product1}) - \prod_{\text{Prod\_name}} (\text{Product2})$$

Output –

Prod_Name
Shoes
Kurta
Watch

Cartesian Product (X)

The cartesian product is an operation used to combine each row in a given table with each row of another table. It is also known as cross product

Example

Customer X Purchase

Output –

Cust_Id	Cust_Name	Address	Purchase_ID	Cust_ID	Prod_ID
1	Indu	Huda, Panipat	190	1	A1
2	Piyush	Jaipur	190	1	A1
3	Raj	Noida	190	1	A1
1	Indu	Huda, Panipat	102	3	N3
2	Piyush	Jaipur	102	3	N3
3	Raj	Noida	102	3	N3

Rename (ρ)

It is used to rename the output relation. Let say you have a tuple named Add and you want to rename it to Address, then you can use Rename operator.

Example

ρ(Product2 , Out_Of_Stock)

This will rename the Product2 to Out_Of_Stock

DBMS

Topic : MySQL Setup

MySQL Installation Guide

Windows

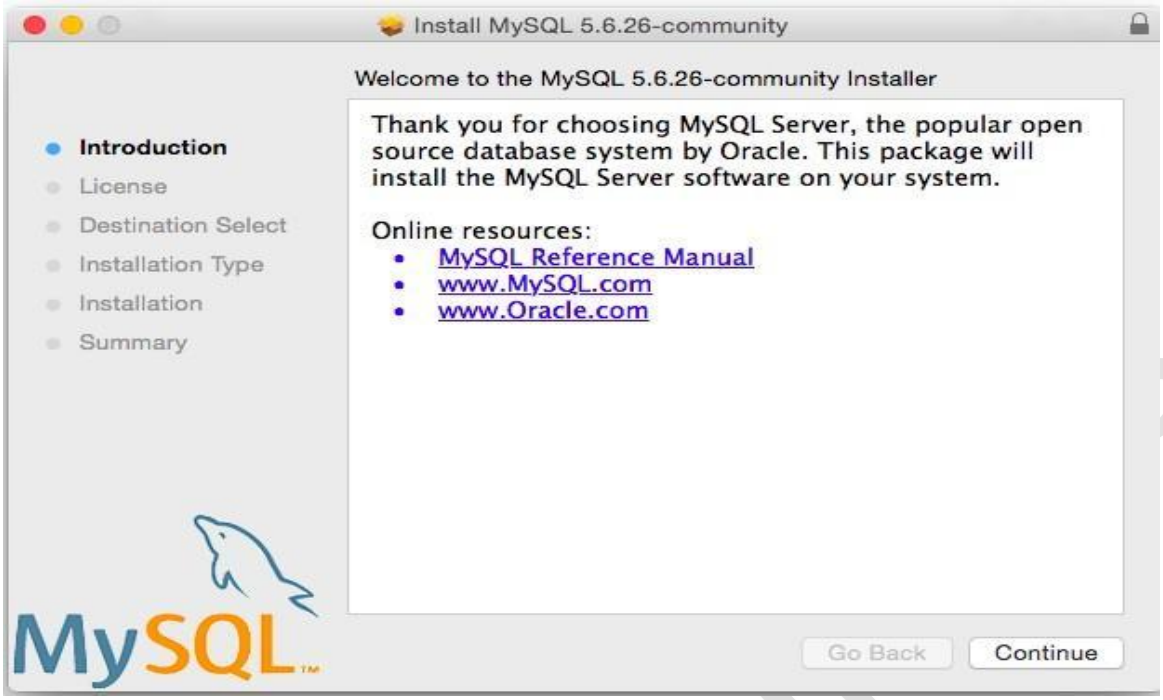
1. Download MySQL Installer from https://dev.mysql.com/downloads/installer/ and execute it. (i.e mysql-installer-community-8.0.26.0.msi)
2. Choose the appropriate Setup Type for your system.
3. Choose "Developer Default" to install MySQL server and other MySQL tools related to MySQL development, helpful tools like MySQL Workbench
4. Complete the installation process by following the instructions. This will install several MySQL products and start the MySQL server. And at last Configure password.
5. Now start it from Startup Menu

Linux

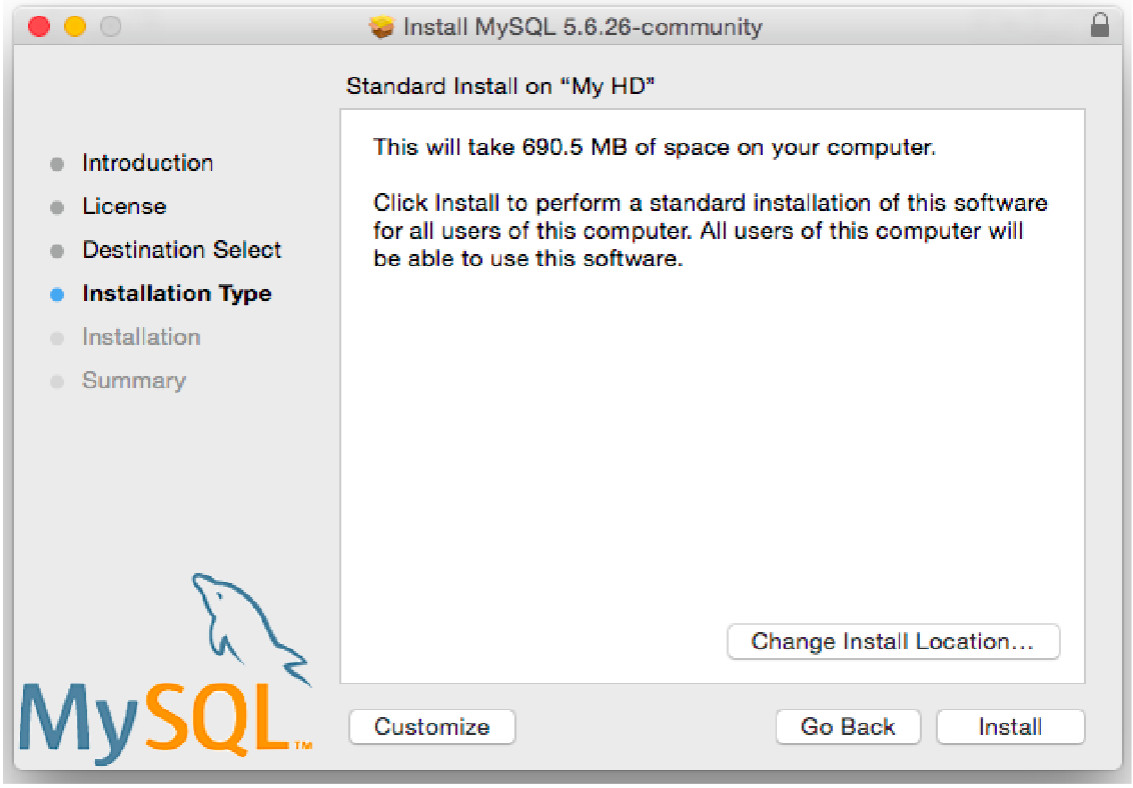
1. Installing MySQL
    a. Update the package index on your server -> sudo apt update
    b. Then install the default package: sudo apt install mysql-server
2. Configuring MySQL
    a. For fresh installations, you'll want to run the included security script
        sudo mysql_secure_installation
    b. Setup Password
    c. Press Y and then ENTER to accept the defaults for all the subsequent questions.
3. Run MySQL
    a. In the terminal run mysql -u root -p
    b. Enter correct Password then play

MacOS

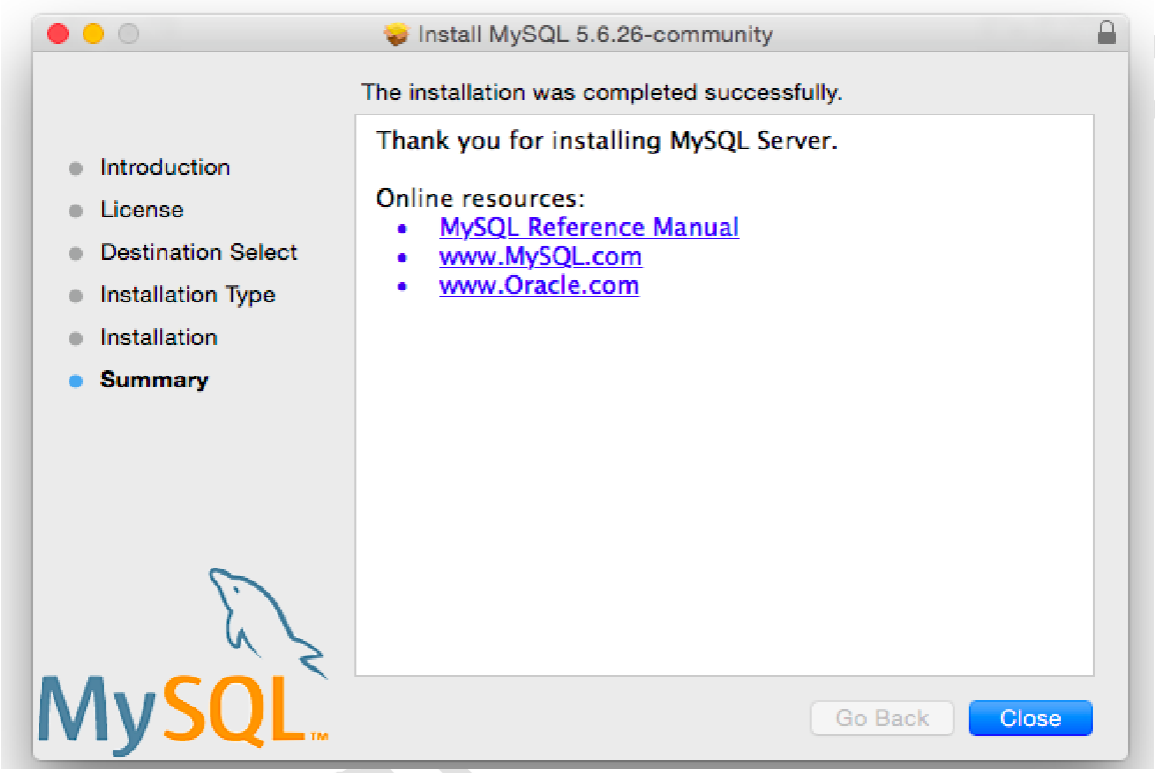
1. Download the disk image (Ædmg†) file from url - https://dev.mysql.com/downloads/mysql/
2. Choose Operating system and then .dmg file
3. Double-click the file to mount the disk image and see its contents.
4. Double-click the MySQL installer package. It will be named according to the MySQL version and the OS X version you have chosen. For example, if you have downloaded the package for MySQL 5.6.44 and OS X 10.8, double-click mysql-5.6.44-osx-10.8-x86_64.pkg.
5. You will be presented with the opening installer dialog. Click Continue to begin installation.



- Click ***Continue*** and then ***Agree*** to continue.
- From the ***Installation Type*** page. Click Install to execute the installation wizard using



8. Click ***Install*** to begin the installation process.
9. Once the installation has been completed successfully, you will be shown an ***Install Succeeded*** message with a short summary. Now, Close the wizard and begin using the MySQL server.



Introduction to SQL

What is SQL?

SQL stands for Structured Query Language. It is used for accessing and manipulating the data.

SQL uses CRUD operations to communicate with the databases. CRUD stands for Create, Read, Update and Delete procedures.

Here is a breakdown of CRUD operations -

- CREATE procedures: Performs the INSERT statement to create a new record.
- READ procedures: Reads the records of the table
- UPDATE procedures: Executes an UPDATE statement on the table based on the specified primary key for a record within the WHERE clause of the statement.
- DELETE procedures: Deletes a specified row in the WHERE clause.

What is RDBMS?

RDBMS stands for **Relational Database Management System**. It is the basis for SQL, and for all modern database systems Ex- MS SQL, MySQL, etc.

Tables are the database objects that are used in RDBMS. Table is a collection of related data entries and it consists of numerous columns and rows.

Table is the simplest form of data storage in a Relational Database. Below is an example of table named Ninjas which contains attributes ID, Ninja's Name and City-

ID	Ninja's Name	City
101	Lokesh Ninja	Kolkata
102	Kuldeep Ninja	Bhopal
103	Ojasv Ninja	Shimla

In this course we will be using an open-source RDBMS i.e MySQL. This database uses Structured Query Language for all CRUD operations and other procedures.

MySQL uses the Client-Server model. In this model a "client" is a front-end application that uses the services provided by a MySQL server. And this whole use of the services takes place through SQL Queries.

Difference between MySQL and SQL :

SQL	MySQL
SQL is a Structured Query Language. It is useful to manage relational databases.	MySQL is an RDBMS to store, retrieve, modify and administrate a database using SQL.
SQL is a query **language**.	MYSQL is used as an **RDBMS** database.
To query and operate database systems.	Allows data handling, storing, modifying, deleting in a tabular format.

Example: Banking System -

Understanding basic MySQL database using Banking System with help of ER Model.

Note:- ER Model :

1. Entity - Real world object (Can be understand as a tables)
2. Attributes - These are the properties of the entity.

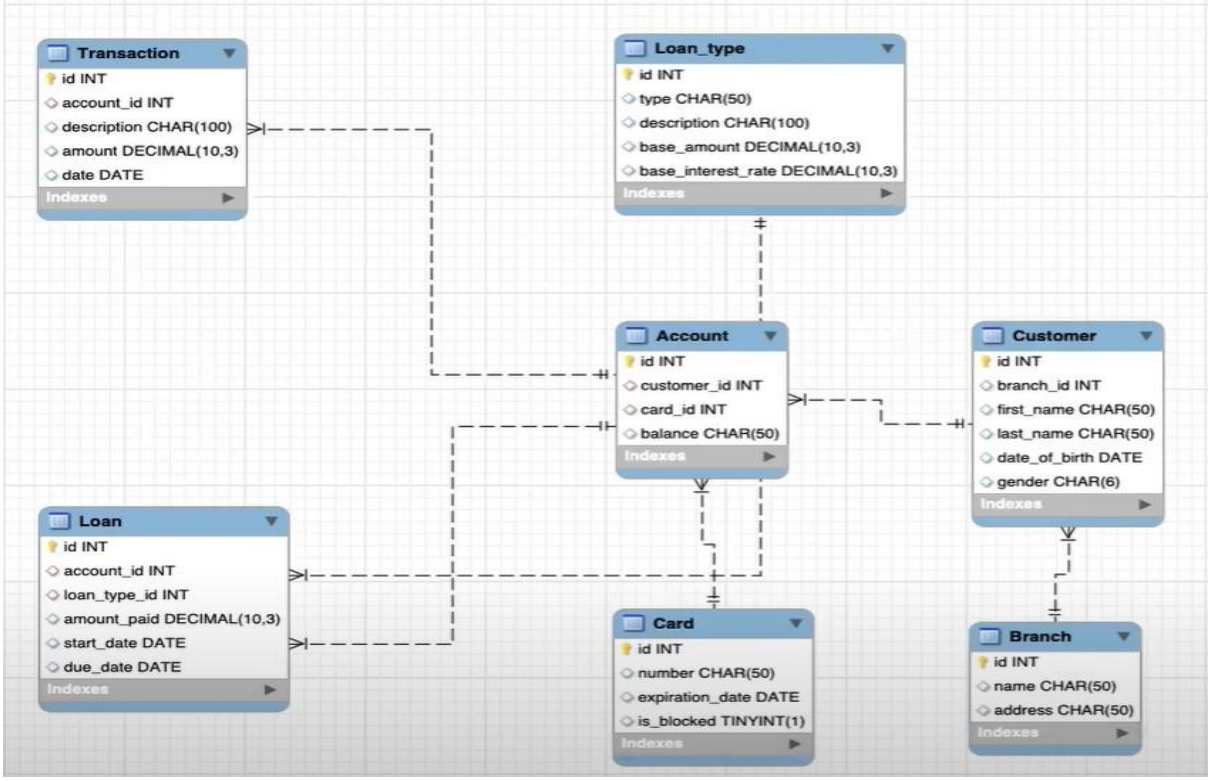
In our Banking system we will be considering below mentioned entities:

1. Customer
2. Account
3. Branches
4. Loan
5. Loan type(Loan_type)
6. Transactions
7. Cards

Customer entity and with attributes(column):-

customer_id	branch_id	first_name	last_name	Gender	D.O.B

In this above customer entity we have used branch_id as the attribute that we can use to set up a relationship with Branch entity. Similarly, in the given ER Model below we can understand the different relationships that exist between different entities.


Transaction
id INT
account_id INT
description CHAR(100)
amount DECIMAL(10,3)
date DATE
Indexes
Loan_type
id INT
type CHAR(50)
description CHAR(100)
base_amount DECIMAL(10,3)
base_interest_rate DECIMAL(10,3)
Indexes
Account
id INT
customer_id INT
card_id INT
balance CHAR(50)
Indexes
Customer
id INT
branch_id INT
first_name CHAR(50)
last_name CHAR(50)
date_of_birth DATE
gender CHAR(6)
Indexes
Loan
id INT
account_id INT
loan_type_id INT
amount_paid DECIMAL(10,3)
start_date DATE
due_date DATE
Indexes
Card
id INT
number CHAR(50)
expiration_date DATE
is_blocked TINYINT(1)
Indexes
Branch
id INT
name CHAR(50)
address CHAR(50)
Indexes

SQL Data Types-

We store data in SQL in the form of tables. Tables are the simplest objects(structures) for data storage that exist in a database.

Below we have given an example of table -

Defining a table using SQL Commands -

```
CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   Ninaja_Name VARCHAR (20)     NOT NULL,
   Ninja_Age  INT               NOT NULL,
   Ninja_Address  CHAR (25) ,
   Ninja_Goal CHAR(25),
   PRIMARY KEY (ID)
);
```

This command will result into creation of a table -

ID	Ninja_Name	Ninja_Age	Ninja_Address	Ninja_Goal

From a SQL command for each column attribute we have defined a specific data type. In case of ID it is INT i.e Integer and similarly in case of Ninja_Name it is VARCHAR i.e a variable length string.

While choosing a data type for a column attribute we have to think of

1. Value we want to represent - like if it is a name(Character data type will be most suitable, VARCHAR) or Age(Numerical data type will be more relevant, INT)
2. Space or Memory that value will occupy - like if we are taking customer feedback we need to store value in a variable character data type, VARCHAR.

Below we have given three main types of data types -

String Data Types -

Datatype	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special

	characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The *size* parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The *size* parameter specifies the maximum column length in bytes.

Numeric Data Types:

Datatype	Description
BIT(*size*)	A bit-value type. The number of bits per value is specified in *size*. The *size* parameter can hold a value from 1 to 64. The default value for *size* is 1.
TINYINT(*size*)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The *size* parameter specifies the maximum display width (which is 255)
INT(*size*)/ INTEGER(*size*)	Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The *size* parameter specifies the maximum display width (which is 255)
FLOAT(*p*)	A floating point number. MySQL uses the *p* value to determine whether to use FLOAT or DOUBLE for the resulting data type. If *p* is from 0 to 24, the data type becomes FLOAT(). If *p* is from 25 to 53, the data type becomes DOUBLE()
DECIMAL(*size*, *d*)	The number of digits after the decimal point is specified in the *d* parameter. The maximum number for *size* is 65. The maximum number for *d* is 30. The default value for *size* is 10. The default value for *d* is 0.

Date and Time Data Types:

Datatype	Description
DATE	Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME	This is a date and time combination in the format: YYYY-MM-DD hh:mm:ss. The supported range starts from '1000-01-01 00:00:00' and end at '9999-12-31 23:59:59'.
TIME	This is a time Format: hh:mm:ss. The supported range is starts from '-838:59:59' and ends at '838:59:59'
TIMESTAMP	TIMESTAMP values are stored as the number of seconds. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC.

SQL Commands

Types of SQL statements/commands -

1. DDL(Data Definition language)
2. DQL/DRL(Data Query Language or Data Retrieve Language)
3. DML(Data Manipulation Language)
4. DCL(Data Control Language)
5. TCL(Transaction Control Language)

DDL(Data Definition Language) -

DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. DDL commands are mentioned below -

- **CREATE** Create TABLE, DATABASE, INDEX or VIEW
- **DROP** Delete TABLE, DATABASE, or INDEX
- **ALTER TABLE** Add/Remove columns from table
- **TRUNCATE** Removes all records from a table.
- **RENAME** Rename an existing object in the table.

DQL(Data Query Language) -

DQL consists of commands that can feasibly retrieve the data from the database using a single command. DQL commands are mentioned below -

- **SELECT** Select data from database.

DML(Data Manipulation Language)-

DML commands are used to make modifications to the database. DQL commands are given below -

- **INSERT** Insert data into a table.
- **UPDATE** Update table data.
- **DELETE** Delete rows from a table.

DCL(Data Control Language)-

DCL commands are used to grant and take back authority from users. DCL commands are given below -

- **GRANT** Access privileges to the database.
- **REVOKE** Withdraws the user's access privileges.

TCL(Transaction Control Language)-

TCL commands are used to manage transactions done in the database. Some of theTCL commands are given below -

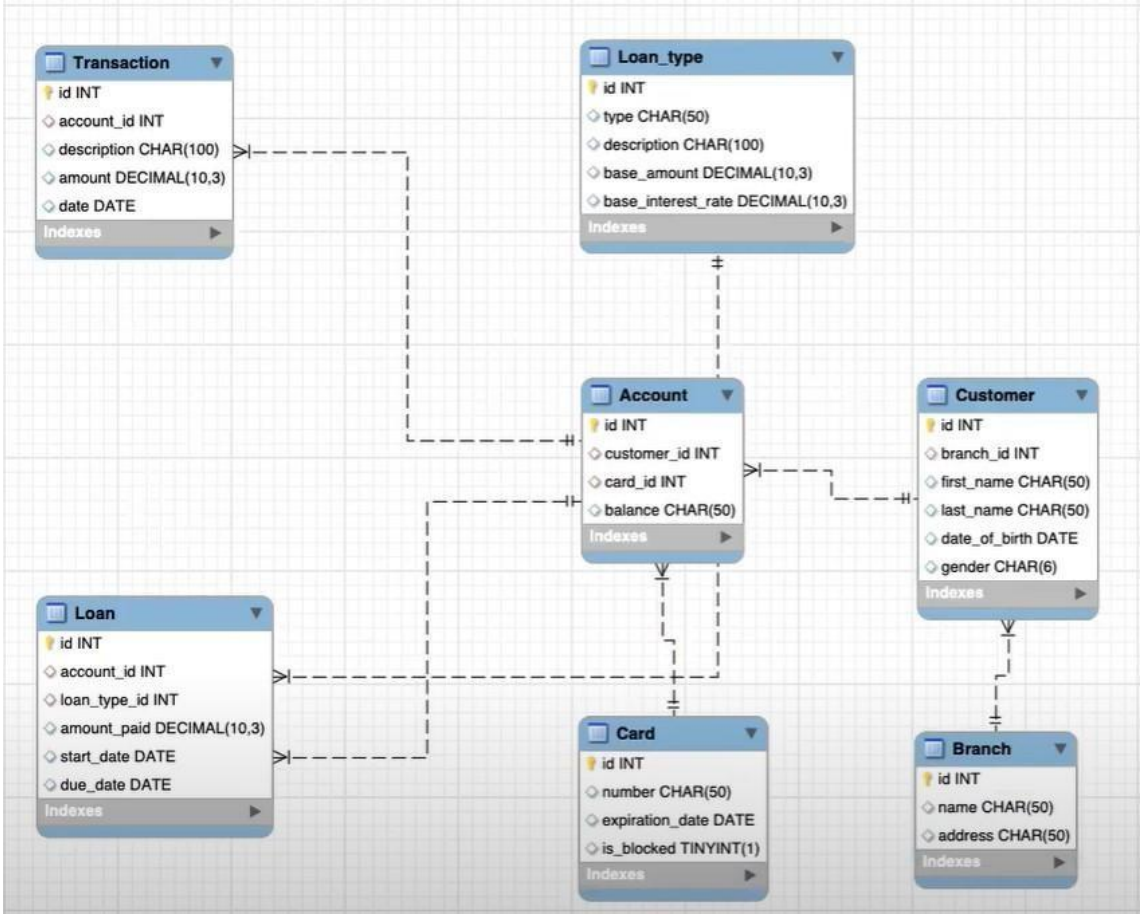
- **BEGIN TRANSACTION** used to begin a transaction.
- **COMMIT** used to apply changes and end transactions.
- **ROLLBACK** used to discard changes and end transactions.
- **SAVEPOINT** points within the groups of transactions in which to ROLLBACK.

Filtering and Sorting Data

Filtering - SQL filters are used to obtain a specific subset of the data items from the database.

The filter is an SQL WHERE clause that provides a set of comparisons that must be true in order for a data item to be returned.

Ex-



For the above ER Diagram,
To get data associated with a particular address we will use a filter. Given below, gives us a branch that has Delhi as an address.

```
SELECT * FROM Branch WHERE address = “Delhi”;
```

Here, address is the key and “Delhi” is the value which makes the key-value pair for the WHERE clause condition.

Ex- For the same ER Model - We want to find a loan with an interest rate of either 25% or 35%.

```
SELECT * FROM Loan_type WHERE base_interest_rate = 25 OR base_interest_rate = 35;
```

In the above examples, we saw how we can use filters i.e., conditionals inside the WHERE clause.

Ex - For the same ER Model - We want to find customer details whose date of birth is before 16th June 2000.

```
SELECT * FROM Customer WHERE date_of_birth < ’16-06-2000’;
```

General Form -

The general form of the **WHERE** clause in our query as a Conditional statement to filter the data is given below:

```
SELECT column_name(s) FROM T_name WHERE conditions;
```

When filtering the strings:

Wildcards:

Symbol	Description
%	Represents zero or more characters
_	Represents a single character

Few examples:

1. 'a%' - Find any value that starts with "a".
2. '%or%' - Finds any values that have "or" in any position.
3. '_r%'- Finds any values that have "r" in the second position.
4. 'a%o'- Finds any values that start with "a" and end with "o"

Ex- Write a SQL query to fetch branch id whose address starts with B,

```
SELECT id FROM Branch WHERE address = “B%”;
```

We use wildcards with LIKE operators. Query: - SELECT column_name(s) FROM T_name WHERE column_name LIKE '%o_r%';

Filtering Operators

1. **IN operator:**

   The IN operator allows specifying multiple values in a WHERE clause.
   Note: - We can use multiple OR operators in place of IN operators as well.
   General Form:

   ```
   SELECT column_name(s) FROM table_name WHERE column_name IN (value1, value2, …);
   ```

   Equivalent-

   ```
   SELECT column_name(s) FROM table_name WHERE column_name = value1 OR column_name = value2 OR column_name = value3, …….;
   ```

2. **NOT IN operator:**

   NOT IN operator is the same as IN operator in syntax and it just negates the conditional i.e., NOT IN operator will return true for only conditionals which are false for IN operator.
   General Form:

   ```
   SELECT column_name(s) FROM table_name WHERE column_name NOT IN (value1, value2, …);
   ```

   The above query will return column_name which is not equal to value1, value2, etc. which is nothing but a negation of what we get from IN operator.

3. **IS NULL:**

   The IS NULL operator is used to test for empty values (NULL values).
   Ex- We want to find the branch ID for which address is NULL, the query is given below –

   ```
   SELECT branch_id FROM branch where address IS NULL;
   ```

   Similarly, we also have IS NOT NULL operator which we use to check for non-empty values.

Ex: We want to find a branch ID for which address is not NULL, a query is given below-

```
SELECT branch_id FROM branch where address IS NOT NULL;
```

Apart from these operators, there are several comparison operators which can be used to filter data. These operators are particularly used when filtering is based on some numeric fields. The various comparison operators are shown in the table below:

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

General Form:

```
SELECT columnList(s) FROM table_name WHERE column_name operator value(s);
```

There are some Logical operators which can also be used to filter data. Following are some logical operators that can be used to filter data along with the syntax of writing the query with the specific operator.

Operator	Meaning
AND	Returns true if both component conditions are true
OR	Returns true if either component condition is true
NOT	Returns true if the following condition is false

- AND requires both conditions to be true:

  SELECT columnList(s) FROM table_name WHERE condition1 AND column_name LIKE '%a_;

- OR requires either condition to be true.

  SELECT columnList(s) FROM table_name WHERE condition1 OR column_name LIKE '%a_;

- NOT operator
  SELECT columnList(s) FROM table_name WHERE column_name NOT IN (value1, value2, …);

Order of precedence is the order in which the filtering conditions will be evaluated. This is one of the important concepts when you are preparing for interviews. The order of precedence of the filtering component of SQL are given below:

Order Evaluated	Operator
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	NOT logical condition
7	AND logical condition
8	OR logical condition

Sorting: -

Sorting is simply done using the ORDER BY keyword which is used to sort the result set in ascending or descending order.

General form:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Ex- Enlist account ID in ascending order.

```
SELECT id FROM account ORDER BY id ASC;
```

Q. What is the difference between Sorting and Filtering?
Ans.
Sorting is arranging the data in ascending or descending order according to the user's needs

Whereas Filtering is used to fetch the data that a user might need for a use case.

Grouping

Grouping in SQL is done using GROUP BY. GROUP BY statement groups rows that have the same values into summary rows, like “find the accounts in each branch”

General form:-
Query:- SELECT column_name(s) FROM T_name WHERE condition GROUP BY column_name(s);

Ex-
table named Ninjas -

ID	Ninja’s Name	City
101	Lokesh Ninja	Kolkata
102	Kuldeep Ninja	Bhopal
103	Raju Ninja	Kolkata
104	Ojasv Ninja	Shimla
105	Abhi Ninja	Bhopal
106	Tarun Ninja	Bhopal

- Find the number of Ninjas in each city ?

SELECT COUNT(ID), City FROM Ninjas GROUP BY City ORDER BY COUNT(ID) DESC;
Output:

COUNT(ID)	City
1	Shimla
2	Kolkata
3	Bhopal

Note: In the above query we have used the aggregate function COUNT().

Now if we want only CITY with COUNT = 2 (lets say), we have to use the HAVING clause.
SELECT COUNT(ID), City FROM Ninjas GROUP BY City HAVING COUNT(ID) = 2;

Output:

COUNT(ID)	City
2	Kolkata

General form:-
Query:- SELECT column_name(s) FROM T_name WHERE condition GROUP BY column_name(s) HAVING condition;

Queries with Tables & Constraints

- **Create table clause:**

Syntax - 1: Both Constraint and Key are declared with column definition.

```
CREATE TABLE <table-name> (
<column-name-1> <data-type>,
<column-name-2> <data-type> CONSTRAINT,
<column-name-3> <data-type> KEY,
<column-name-4> <data-type> CONSTRAINT KEY
);
```

Syntax - 2: Key declared at the end.

```
CREATE TABLE <table-name> (
<column-name-1> <data-type>,
<column-name-2> <data-type>,
<column-name-3> <data-type> CONSTRAINT,
KEY_NAME1(<column-name1>),
KEY_NAME2(<column-name2>)
);
```

Syntax - 3: Both Constraint and Key declared at the end.

```
CREATE TABLE <table-name> (
<column-name-> <data-type>,
<column-name-> <data-type>,
KEY_NAME1(<column-name>),
KEY_NAME2(<column-name>),
CONSTRAINT1(<column-name>)
);
```

- **ALTER table clause:**

The various variations in ALTER table clause are explained in the following.

a. Add column

Syntax:

```
ALTER TABLE <table-name>
ADD <column-name> <data-type>;
```

b. Drop column

Syntax:

ALTER TABLE *<table-name>*
DROP COLUMN *<column-name>*;

c. Modify column

Syntax-1:

ALTER TABLE *<table-name>*
MODIFY COLUMN *<column-name>* *<data-type>*;

Syntax-2:

ALTER TABLE *<table-name>*
ALTER COLUMN *<column-name>* *<data-type>*;

Below is a customer table with details related to the customer:

id	branch_id	first_name	last_name	DOB	gender

This is the table we want to create with Primary Key as Id, below is the query for it -

```
CREATE TABLE customer
(
id INT NOT NULL,
branch_id INT NOT NULL,
first_name VARCHAR(20),
last_name VARCHAR(20),
DOB INT,
gender CHAR(6),
PRIMARY KEY(id)
);
```

In the above query, we have declared id as PRIMARY KEY.

- **What is the Primary key?**

o Primary Key helps us to uniquely identify tuples of a relation. It cannot be NULL and have UNIQUE values.

o Each table has only one primary key. But a primary key can have one or more columns(fields) as part of the Primary Key.

- In case we have multiple fields as Primary key it is called **Composite Key** and all the conditions that apply to a single field Primary Key applies to multiple field Composite Key as well.

- Let's say for the same customer table we want id and branch id as PRIMARY KEY we can use the below-mentioned declaration if we haven't declared the table yet -

```
CREATE TABLE customer
(
 id INT NOT NULL,
 branch_id INT NOT NULL,
 first_name VARCHAR(20),
 last_name VARCHAR(20),
 DOB INT,
 gender CHAR(6),
 PRIMARY KEY(id, branch_id)
);
```

Let's say we have declared table, as mentioned below:

```
CREATE TABLE customer
(
id INT NOT NULL,
branch_id INT NOT NULL,
first_name VARCHAR(20),
last_name VARCHAR(20),
DOB INT,
gender CHAR(6)
);
```

In case we want to make any attribute a Primary Key we need to use ALTER keyword for that as mentioned below -

```
ALTER TABLE CUSTOMER
ADD PRIMARY KEY (ID);
```

Note:- Primary Key column should already have been declared as NOT NULL at the time of creating the table.

Similarly, we can declare the composite primary key (assuming columns or attributes we want to use are declared as NOT NULL) as done below -

```
ALTER TABLE CUSTOMERS
ADD CONSTRAINT Pkey_Custid PRIMARY KEY (ID, NAME);
```

We have learned to declare the primary key and all the conditions associated with column(s) to be the Primary key now let us discuss **how to delete a primary key** - Let us say have declared a table as below mentioned -

```
CREATE TABLE customer
(
 id INT NOT NULL,
 branch_id INT NOT NULL,
 first_name VARCHAR(20),
 last_name VARCHAR(20),
 DOB INT,
 gender CHAR(6),
 PRIMARY KEY(id)
);
```

Now we want to delete id primary key we will use ALTER and DROP keyword as mentioned below -

```
ALTER TABLE CUSTOMERS
DROP PRIMARY KEY;
```

- **What is Foreign Key?**

Foreign Key refers to the Primary Key of another table. Each table can have any number of foreign key(s). Let's us say we have a customer and account table with details mentioned below -

Table : customer

id	branch_id	first_name	last_name	DOB	gender

Table : account

id	balance	customer_id

Declaring customer table -

```
CREATE TABLE customer
(
id INT NOT NULL,
branch_id INT NOT NULL,
first_name VARCHAR(20),
last_name VARCHAR(20),
DOB INT,
gender CHAR(6),
PRIMARY KEY(id)
);
```

Declaring account table with Primary Key and Foreign Key -

```
CREATE TABLE account
(
id INT NOT NULL,
balance INT,
customer_id INT NOT NULL,
PRIMARY KEY(id),
FOREIGN KEY(customer_id) REFERENCES customer(id) );
```

- **What is a UNIQUE constraint?**

UNIQUE constraints make sure that all the values in a column which is declared UNIQUE are different.

Then, what is the difference between UNIQUE and PRIMARY KEY?

UNIQUE	PRIMARY KEY
UNIQUE constraint column need not be a PRIMARY KEY.	PRIMARY KEY constraint automatically has a UNIQUE constraint.
Multiple columns in a table can have a UNIQUE constraint	There is only one PRIMARY KEY in a table

Below we have mentioned an example of a UNIQUE Constraint –

```
CREATE TABLE customer
(
id INT NOT NULL,
name VARCHAR(255) NOT NULL,
UNIQUE(id)
);
```

Using UNIQUE constraint in multiple columns

```
CREATE TABLE customer
(
id INT NOT NULL,
name VARCHAR(255) NOT NULL,
CONSTRAINT UNIQUE(id, name)
);
```

- **What is the CHECK constraint?**

The CHECK is used to put limitations on the value range that we can put in a column.

For example let's say we have the account table as given below,

account_id	balance	customer_id

If we want only customers with a minimum balance of let us say 3000 or more we can use the CHECK constraint to ensure that.

```
CREATE TABLE account
(
account_id INT NOT NULL,
balance INT NOT NULL,
customer_id INT NOT NULL,
CHECK(balance >= 3000)
);
```

- **What is the DEFAULT constraint?**

DEFAULT constraint sets a default value for a column. This default value is added to all the new records unless other value is specified.

Let us say for all the accounts we want to have a balance of 100 when an account is added we can make sure of this by using the DEFAULT constraint

```
CREATE TABLE account
(
account_id INT NOT NULL,
balance INT DEFAULT 100,
customer_id INT NOT NULL,
);
```

We can drop the default constraint by using DROP keyword –

```
ALTER TABLE account
ALTER balance DROP DEFAULT;
```

Also, if suppose we haven't initialised the balance we can later modify as done below-

```
ALTER TABLE account
MODIFY balance DEFAULT 100;
```

Below mentioned table summarises the constraint -

Constraint	Description
CHECK	Determine whether the value is valid or not from a logical expression
FOREIGN KEY	Link between two tables by one specific column of both tables. The specified column in one table must be a PRIMARY KEY and referred by the column of another table known as a FOREIGN KEY.
UNIQUE	Maintains the uniqueness of a column in a table. More than one UNIQUE column can be used in a table.
NOT NULL	column can not contain any NULL value
PRIMARY KEY	Enforces the table to accept unique data for a specific column and is a unique index for accessing the table faster.

- **What is TRUNCATE? How is it different from DROP and DELETE?**

TRUNCATE removes all the records from the table. Below is the given general form -

TRUNCATE TABLE account;

After running this query all the data from the account table will be cleared but the table will still exist.

Whereas in the case of DROP the whole table will be dropped from the database i.e table record and schema will be cleared from the database.

Also, DELETE is used to delete the record of a particular row or conditionals -

DELETE FROM account where account_id = XYZ;

- **Difference between Delete, Drop and Truncate:**

Delete	Drop	Truncate
DML command	DDL command	DDL command
Removes one, some or all the records in the table.	Removes the entire table structure.	Removes all the records from the table.
Is a slow operation	Relatively faster	Fastest of all.

- **ON DELETE CASCADE:**

ON DELETE CASCADE is an option that can be added to the table while creating constraints. It is used to delete rows from the child table automatically when similar rows in the parent table get deleted.

- **ON UPDATE CASCADE:**

ON UPDATE CASCADE is an option that can be added to the table while creating constraints. Suppose there is a situation where we have two tables such that primary key of one table is the foreign key for another table. if we update the primary key of the first table then using the ON UPDATE CASCADE foreign key of the second table automatically get updated.

- **How to rename the table and column?**

Rename a table from cust to customers –

```
ALTER TABLE cust
RENAME TO  customers;
```

Rename column name to the surname in customers table -

```
ALTER TABLE customers
RENAME name TO surname;
```

Information Schema

We have an in-built view in MySQL called information_schema, which holds all details about the constraints on a particular table. INFORMATION_SCHEMA provides access to information about the MySQL server, such as database metadata, database or table names, column data types, and permissions.

General form:

```
SELECT COLUMN_NAME, CONSTRAINT_NAME, REFERENCED_COLUMN_NAME,
REFERENCED_TABLE_NAME
FROM information_schema.KEY_COLUMN_USAGE
WHERE TABLE_NAME = 'table_name';
```

Modifying Data

- Modifying Data mainly comes under Data Manipulation Language in SQL. There are mainly four types of operations we can perform on a table/database.

1. INSERT
2. UPDATE
3. DELETE
4. RENAME

For explaining all of these operations we will use the below-mentioned table- (Assume it to be empty at the beginning)

Table Name - **Student**

Student_id	Student_name	Batch	Course

- **INSERT**

To perform the INSERT operation, we have to use the INSERT INTO statement. There are two ways to use this statement as mentioned below.

Mention both the column names and the values you want to insert -

```
INSERT INTO Student (Student_id, Student_name, Batch, Course)
VALUES (1, Ojasv, 2018, SQL);
```

Output:-

Student_id	Student_name	Batch	Course
1	Ojasv	2018	SQL

Mention only the values of for the columns. Make sure the order of

occurrence of values is the same as the order of columns -

INSERT INTO Student
VALUES (2,Kuldeep, 2018, MongoDB);

Output:-

Student_id	Student_name	Batch	Course
1	Ojasv	2018	SQL
2	Kuldeep	2018	MongoDB

For storing date and time fields in SQL we have the following date fields

1. DATE - format YYYY-MM-DD
2. YEAR - format YYYY or YY
3. DATETIME - format: YYYY-MM-DD HH:MI: SS
4. TIMESTAMP - format: YYYY-MM-DD HH:MI: SS

General form:

INSERT INTO table_name

VALUES (value1,value2...);

Example:

Consider a Table name : **order** with following attributes

o_id	pname	o_date

INSERT function will be shown below:

INSERT INTO order
VALUES ('1', 'table', '2022-01-13');

o_id	pname	o_date
1	table	2022-01-13

- **UPDATE**

To perform update operations on a table/database we have to use UPDATE statement.

General Form -

```
UPDATE TABLE
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Note: Always remember to use WHERE with the UPDATE statement otherwise values will be updated for all the columns.

Ex- UPDATE Kuldeep to Kuldeep Ravaliya and MongoDB to MongoDB & SQL.

Query:

```
UPDATE Student
SET Student_name = 'Kuldeep Ravaliya' AND Course = 'MongoDB & SQL'
WHERE Student_id = 2;
```

Output:

Student_id	Student_name	Batch	Course
1	Ojasv	2018	SQL
2	Kuldeep Ravaliya	2018	MongoDB & SQL

- **Additional Information:**

The functions below can be used in the INSERT and UPDATE clauses to add the current timestamp in the column value.

1. CURDATE(): The CURDATE() function returns the current date value in the 'YYYYMMDD' format or "YYYY-MM-DD' format. It depends on whether a string or numeric value is used in the table function.

   General Form:

   ```
   SELECT CURDATE();
   ```

2. CURTIME(): The CURTIME() returns the current time value in HHMMSS.uuuuuu format or 'HH:MM: SS' format. It depends on whether a string or numeric value is used in the table function.

   General Form:

```
SELECT CURTIME();
```

- **DELETE**

To perform the delete operation on the existing table/database we use the DELETE statement

General form-

```
DELETE FROM table
WHERE condition;
```

Ex-

```
DELETE FROM Students
WHERE Student_name = 'Kuldeep Ravaliya';
```

Output:

Student_id	Student_name	Batch	Course
1	Ojasv	2018	SQL

Note- We can also delete all the records from the table using - DELETE FROM TABLE - this all table rows will be deleted but the table will still be there for use.

- **RENAME**

To perform Rename operation on a table we can use RENAME statement -

Ex- We can change the name of the student table 'Student' to 'Student1' using RENAME statement, syntax for that –

```
RENAME TABLE Student TO Student1;
```

Output:

Student1 -

Student_id	Student_name	Batch	Course
1	Ojasv	2018	SQL

- **REPLACE INTO**

To perform Replace operation on a table we can use REPLACE INTO statement.
General form-

```
REPLACE [INTO] T_name(column_name(s))
VALUES(value_list);
```

Ex-

```
REPLACE INTO Student1(student_name)
VALUES ('1','LOKESH', 2019,'SQL');
```

Output:

Student_id	Student_name	Batch	Course
1	Lokesh	2018	SQL

- **Difference between UPDATE & REPLACE INTO?**

UPDATE	REPLACE INTO
This is used to modify existing data in the table.	This works exactly like INSERT, but if an old row in the table has the same value as a new role for a PRIMARY KEY or a UNIQUE index, the old row is deleted before the new row is inserted.

- **DELETE in Safe Mode** -

By default **safe update mode** is enabled. We can disable it by this command:

```
SET SQL_SAFE_UPDATES = 0;
```

After this command, the DELETE clause can be used.

To enable the safe mode, we can use the following command:

SET SQL_SAFE_UPDATES = 1;

Safe mode exists to disallow update and delete operations without the use of PRIMARY KEY in a WHERE clause. Hence, we can prevent the loss of data with the help of DELETE in Safe Mode.

- **DELETE CASCADE :**
  This clause is used to delete multiple records from more than one table, linked through foreign key.

- **UPDATE CASCADE :**
  This clause is used to update multiple records from more than one table, linked through foreign key.

- **REPLACE :**
  - It is used to update the already present tuple data in a relation.
  - When we use REPLACE query with the help of WHERE clause in PRIMARY KEY column, then the row present will get update.
  - If there is no reference of the primary key, then a new tuple entry will be added in the relation, with updated values.

Joining Tables

- **What are JOINS in SQL?**

JOIN is an operation that exists in SQL that helps us to combine rows from two or more tables based on a related column between them.

- **SQL Aliases –**

Alias is a concept that is used to give temporary names to tables or particular column in a table. They exist till the duration of the query. Alias name can be give through “AS” keyword or without it.

Syntax-1 –

```
SELECT t.col_name
FROM <table-name> t;
```

Syntax-2 –

```
SELECT t.col_name
FROM <table-name> AS t;
```

- **Types of JOIN**:-

1. INNER JOIN - This returns a resulting table that has matching values from both the table or all the tables.

2. OUTER JOIN – We have 3 types of OUTER JOINS -

- **LEFT OUTER JOIN** - This returns a resulting table that all the data from left table and the matched data from the right table
- **RIGHT OUTER JOIN** - This returns a resulting table that all the data from right table and the matched data from the left table
- **FULL OUTER JOIN** - This returns a resulting table that contains all data when there is a match on left or right table data. It can be implemented using UNION operation.

3. CROSS JOIN – This returns all the cartesian product of the data present in both the tables. Hence, all possible variations are reflected in the output.

4. SELF JOIN – It is used to get the output from a particular table, when the same table is joined to itself.

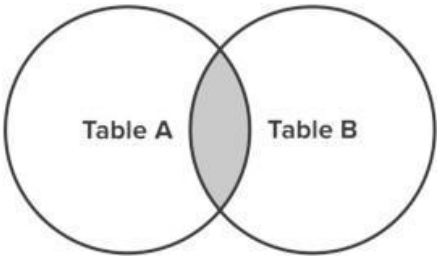
NOTE-1 : It is not compulsory to use "OUTER" keyword in the joins. We can specify the join as LEFT JOIN or LEFT OUTER JOIN, there is no difference or error in the output.

NOTE-2 : Query - ***Table_Name.****

Output – All the data from particular table. This kind of query is used in case of multiple tables.

NOTE-3: The JOINS can be used interchangeably (LEFT OUTER & RIGHT OUTER etc), and a query can be amended to change the particular join into another alternate join, but the output will be the same. There will be no error in doing so.

- **INNER JOIN -**



General form -

```
SELECT *
FROM TableA
INNER JOIN TableB
ON TableA.column1 = TableB.column1;
```

Example:

Table 1:- NINJA table is shown below -

Ninja_ID	Name	CITY
1	Ojasv Ninja	Jaipur
2	Tejas Ninja	Trichy
3	Rejas Ninja	Manipal

Table 2:- ORDERS

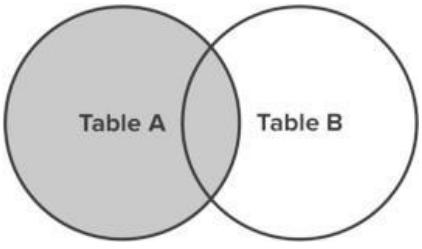
Order_ID	Product_name	Ninja_ID
1	SQL	1
2	WEBDEV	2
3	CP	3
4	ALGO	4

```
SELECT Ninja_ID, Name, City, Product_name
FROM NINJA
INNER JOIN ORDERS
ON NINJA.Ninja_ID = ORDERS.Order_ID;
```

Output:-

Ninja_ID	Name	City	Product_name
1	Ojasv Ninja	Jaipur	SQL
2	Tejas Ninja	Trichy	WEBDEV
3	Rejas Ninja	Manipal	CP

- **LEFT JOIN -**



General form -

```
SELECT *
FROM TableA
LEFT JOIN TableB
ON TableA.column1 = TableB.column1;
```

Example:
Table 1: NINJA table is shown below -

Ninja_ID	Name	CITY
1	Ojasv Ninja	Jaipur
2	Tejas Ninja	Trichy
3	Rejas Ninja	Manipal
5	Kejas Ninja	Lucknow

Table 2: ORDERS

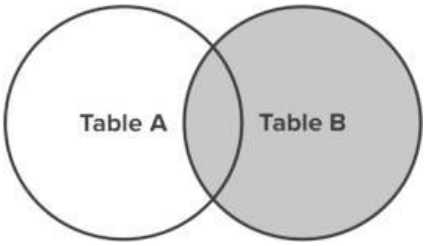
Order_ID	Product_name	Ninja_ID
1	SQL	1
2	WEBDEV	2
3	CP	3
4	ALGO	4

```
SELECT Ninja_ID, Name, City, Product_name
FROM NINJA
INNER JOIN ORDERS
ON NINJA.Ninja_ID = ORDERS.Order_ID;
```

Output:-

Ninja_ID	Name	City	Product_name
1	Ojasv Ninja	Jaipur	SQL
2	Tejas Ninja	Trichy	WEBDEV
3	Rejas Ninja	Manipal	CP
5	Kejas Ninja	Lucknow	NULL

- **RIGHT JOIN -**



General form -

```
SELECT *
FROM TableA
RIGHT JOIN TableB
ON TableA.column1 = TableB.column1;
```

Example:

Table 1: NINJA table is shown below -

Ninja_ID	Name	CITY
1	Ojasv Ninja	Jaipur
2	Tejas Ninja	Trichy
3	Rejas Ninja	Manipal
5	Kejas Ninja	Lucknow

Table 2: ORDERS

Order_ID	Product_name	Ninja_ID
1	SQL	1
2	WEBDEV	2
3	CP	3
4	ALGO	4

Query:

```
SELECT Ninja_ID, Name, City, Product_name
FROM NINJA
RIGHT JOIN ORDERS
ON NINJA.Ninja_ID = ORDERS.Order_ID;
```

Output:

Ninja_ID	Name	City	Product_name
1	Ojasv Ninja	Jaipur	SQL
2	Tejas Ninja	Trichy	WEBDEV
3	Rejas Ninja	Manipal	CP
NULL	NULL	NULL	ALGO

- **FULL JOIN -**

General form -

```
SELECT * FROM TableA
LEFT JOIN TableB ON TableA.column1 = TableB.column2
UNION
SELECT * FROM TableA
RIGHT JOIN TableB ON TableA.column1 = TableB.column2
```

Example:

Table 1: NINJA

Ninja_ID	Name	CITY
1	Ojasv Ninja	Jaipur
2	Tejas Ninja	Trichy
3	Rejas Ninja	Manipal
5	Kejas Ninja	Lucknow

Table 2: ORDERS

Order_ID	Product_name	Ninja_ID
1	SQL	1
2	WEBDEV	2
3	CP	3
4	ALGO	4

Query:

```
SELECT Ninja_ID, Name, City, Product_name
FROM NINJA
LEFT JOIN ORDERS  ON NINJA.Ninja_ID = ORDERS.Order_ID
UNION
SELECT Ninja_ID, Name, City, Product_name
FROM NINJA
RIGHT JOIN ORDERS  ON NINJA.Ninja_ID = ORDERS.Order_ID;
```

Output:

Ninja_ID	Name	City	Product_name
1	Ojasv Ninja	Jaipur	SQL
2	Tejas Ninja	Trichy	WEBDEV
3	Rejas Ninja	Manipal	CP
3	Kejas Ninja	Lucknow	NULL
NULL	NULL	NULL	ALGO

- **CROSS JOIN & FULL JOIN -**

The CROSS join produces the cartesian product of all the data present in both tables. Hence, all possible combinations or variations will be reflected, also it doesn't have ON clause, as we don't require any condition as we are joining everything to everything. A FULL JOIN is a combination of LEFT JOIN and RIGHT JOIN and gives the value output as per the WHERE clause condition.

- **CROSS JOIN –**

General form -

```
SELECT *
FROM TableA
CROSS JOIN TableA;
```

Example:
Table 1: NINJA

Ninja_ID	Name	CITY
1	Ojasv Ninja	Jaipur
2	Tejas Ninja	Trichy
3	Rejas Ninja	Manipal
5	Kejas Ninja	Lucknow

Table 2: ORDERS

Order_ID	Product_name	Ninja_ID
1	SQL	1
2	WEBDEV	2
3	CP	3
4	ALGO	4

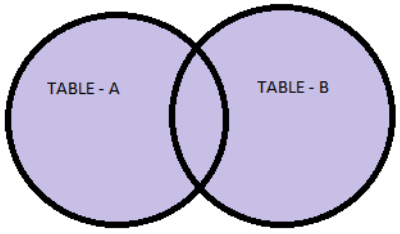
Query:

```
SELECT NINJA.Ninja_ID, NINJA.Name, NINJA.City, ORDERS.Product_name
FROM NINJA
CROSS JOIN ORDERS;
```

Output:

Ninja_ID	Name	City	Product_name
1	Ojasv Ninja	Jaipur	SQL
2	Tejas Ninja	Trichy	SQL
3	Rejas Ninja	Manipal	SQL
5	Kejas Ninja	Lucknow	SQL
1	Ojasv Ninja	Jaipur	WEBDEV
2	Tejas Ninja	Trichy	WEBDEV
3	Rejas Ninja	Manipal	WEBDEV
5	Kejas Ninja	Lucknow	WEBDEV
1	Ojasv Ninja	Jaipur	CP
2	Tejas Ninja	Trichy	CP
3	Rejas Ninja	Manipal	CP
5	Kejas Ninja	Lucknow	CP
1	Ojasv Ninja	Jaipur	ALGO
2	Tejas Ninja	Trichy	ALGO
3	Rejas Ninja	Manipal	ALGO
5	Kejas Ninja	Lucknow	ALGO

- **SELF JOIN -**



General form -

```
SELECT A.Col_1, B.Col_2
FROM TableA  A, TABLEA  B
WHERE A.COL_NAME = B.COL_NAME
AND <condition>;
```

Example:

Table 1: NINJA table is shown below -

Ninja_ID	Name	CITY
1	Ojasv Ninja	Jaipur
2	Tejas Ninja	Trichy
3	Rejas Ninja	Manipal

```
SELECT a.Ninja_ID, b.Name
FROM NINJA a, NINJA b
WHERE a.Ninja_ID = b.Ninja_ID
AND a.City = “Jaipur”
```

Output:-

Ninja_ID	Name
1	Ojasv Ninja

Set Operations

In SQL, Set operations are used to combine multiple SELECT statements. There are mainly four types of set operations in SQL -

1. Union
2. Intersect
3. Minus

- **What is the difference between JOIN and Set Operations(UNION)?**

- **UNION -**
  This operator is used to combine two or more more SELECT statements.

  General Form -
  SELECT column(s) FROM
  table1 UNION
  SELECT column(s) FROM table2;
  Ex- Below we have two tables

  Table - **Ninja**

Ninja_id	Ninja_Name	City
1	Saksham	Dehradun
2	Sachin	Vadodara
3	Utkarsh	Varanasi

Table - **Course**

Course_id	Course_Name	City
1	Database	Lucknow
2	OS	Mumbai
3	Networks	Delhi
4	ML	Varanasi

```
SELECT City FROM Ninja
UNION
SELECT City FROM Course;
```

Output:

City
Dehradun
Vadodara
Varanasi
Lucknow
Mumbai
Delhi

What if we replace UNION with INNER JOIN in the above query?

```
SELECT City FROM Ninja
INNER JOIN
SELECT City FROM Course;
```

Output:

City
Dehradun
Vadodara
Varanasi
Lucknow
Mumbai
Delhi
Varanasi

As you can see there is recurrence in case of INNER JOIN. Hence, let us understand the difference between JOIN and Set Operations.

- **What is the difference between JOIN and Set Operations(UNION)?**

JOIN	UNION(Set Operations)
JOIN Combines multiple table depending upon the matched condition	UNION(& other Set Operations) Combines is the resulting set from two or more SELECT statements
Combination is column-wise	Combination is row-wise
Datatypes of corresponding columns from each table could be different.	Datatypes of corresponding columns from each table should be the same.
May or maynot return distinct row(s)	Returns distinct row(s)
From each table the number of column(s) selected may or maynot be the same.	From each table the number of column(s) selected should be the same.

Note:- Above difference is valid for other set operations and join(inner, left, right) as well.

- **INTERSECT-**

This operator returns the common values of the tables.

Ex- Below we have two tables -

Ninja -

Ninja_id	Ninja_Name	City
1	Saksham	Dehradun
2	Sachin	Vadodara
3	Utkarsh	Varanasi

Course-

Course_id	Course_Name	City
1	Database	Lucknow
2	OS	Mumbai
3	Networks	Delhi
4	ML	Varanasi

```
SELECT DISTINCT City
FROM Ninja
INNER JOIN Course
USING(City);
```

Output:

City
Varanasi

- **MINUS-**

  This operator returns the distinct row from the first table that does not occur in the second table.

  Ex- Below we have two tables

  Table- **Ninja**

Ninja_id	Ninja_Name	City
1	Saksham	Dehradun
2	Sachin	Vadodara
3	Utkarsh	Varanasi

  Table - **Course**

Course_id	Course_Name	City
1	Database	Lucknow
2	OS	Mumbai
3	Networks	Delhi
4	ML	Varanasi

  Query:

```
SELECT City FROM Ninja
LEFT JOIN Course USING(City)
WHERE Course.City IS NULL;
```

Output:

City
Dehradun
Vadodara

Join vs Subqueries

What do you mean by Joins?

JOIN is an operation that exists in SQL that helps us to combine rows from two or more tables based on a related column between them. When many tables exist in the FROM clause of a query, a join is done. Any column from any of these tables can be selected from the query's choose list. A cartesian product is created if the join condition is missing or faulty.

Types of JOINS:

1. **INNER JOIN**: This returns a resulting table that has matching values from both the table and all the tables.
2. **LEFT JOIN**: This returns a resulting table that all the data from the left table and the matched data from the right table.
3. **RIGHT JOIN**: This returns a resulting table that all the data from the right table and the matched data from the left table.
4. **FULL JOIN**: This returns a resulting table that contains all data when there is a match on left or right table data.

What do you mean by Subquery?

Subqueries is a way of using queries within a particular query. It is accomplished using the parenthesis. The inner query is always executed first, followed by the execution of the outer query. Inner query results in a single value or set of values as an output. A subquery is a SELECT statement that is nested within another SQL statement's clause. They can be quite handy for selecting rows from a table based on data from the same or another table. A subquery is used to return data that will be utilized as a condition in the main query to further limit the data that may be retrieved. WHERE clause, SELECT clause, and FROM clause are the SQL clauses where the subquery can be used. Let us say we have a query Q2 and its data will be used by query Q1, then we can say that Q2 will be the Inner query of Main query Q1. Also, this query Q2 is called sub-query.

Join vs Subqueries

Joins	Subquery
Joins execute faster than subqueries	Subqueries execute slower than joins
In most cases, the retrieval time of query using joins is usually faster than subqueries.	There are very few scenarios in which retrieval time of query using subquery will be faster
Joins maximize the calculation burden on the database	Subquery keeps the responsibility of calculation on the user.
Joins are difficult to implement and understand	Subqueries are easier to understand and implement than joins
In Joins, we cannot use the result of other queries.	Subquery helps us to use the result of another query in the outer query.

Table 1 Join vs Subqueries

Transaction Control Language in SQL

Managing of Transactions in the database is done using TCL.

- **What is a Transaction?**

A transaction is a sequential group of queries, statements, or operations such as update, select, insert or delete to perform as a one single work unit that can be committed or rolled back.

Let us understand this using an example –

BEGIN TRANSACTION T1;

COMMIT;

SAVEPOINT S1;

```
SELECT * FROM Customers;

INSERT INTO Customers(id) VALUES (1);

UPDATE Customer SET name ='RAJU' WHERE id = 1;
```

Before we jump into the individual description let us understand what we mean by transaction using the above query example. As you can see in the above query (which is in a box), we call a transaction. We begin a transaction using the TCL command BEGIN TRANSACTION after we write all queries we use the TCL command COMMIT to record all the statements in the transaction (query inside the box). At last we save our current progress to save point S1.

Note- The box we have used is just for representational purposes.

There are mainly four type of TCL Command -

1. BEGIN TRANSACTION
2. COMMIT
3. ROLLBACK
4. SAVEPOINT

- **BEGIN TRANSACTION -**

This TCL command basically starts the transaction.

General form:

BEGIN TRANSACTION transaction_name;

- **COMMIT TRANSACTION -**

This TCL command COMMIT makes changes to the transaction.

General form:

BEGIN TRANSACTION transaction_name;

- **ROLLBACK TRANSACTION -**

This TCL commands uncommit all the changes or restore the current state to any previously saved state.

General form-1:

ROLLBACK; /* This rollback the previously committed command to it's initial state*/

General form-1:

ROLLBACK TO savedpoint; /*This is undo the current state and restore it to savepoint*/

- **SAVEPOINT TRANSACTION -**

This TCL command saves the current state into save point. That save point can later be accessed.

General form:

SAVEPOINT Save_point_name;

Now let us understand all these four commands using an example

Table - **Ninja**:

ID	Ninja_Name	Course
1	Suchit	DBMS
2	Kuldeep	OS
3	Lokesh	CP

```
BEGIN TRANSACTION t1;
DELETE FROM Student WHERE ID = 3;
COMMIT;
```

Output:

Table - **Ninja:**

ID	Ninja_Name	Course
1	Suchit	DBMS
2	Kuldeep	OS

```
INSERT INTO Ninja
VALUES (1, 'Lokesh', 'CP');
ROLLBACK;
```

ID	Ninja_Name	Course
1	Suchit	DBMS
2	Kuldeep	OS

/* Now as we can see even though we inserted new values it is not visible in changes. As we used the ROLLBACK command to undo it. */

SAVEPOINT S1;

/* this saves the current progress to the save point S1*/ DELETE FROM Ninja WHERE ID = 2;

SAVEPOINT S2; /* save point S2 created*/

Output:

ID	Ninja_Name	Course
1	Suchit	DBMS

/* after savepoint 2 above is the current state*/

ROLLBACK TO S1;

Output:

ID	Ninja_Name	Course
1	Suchit	DBMS
2	Kuldeep	OS

Locks:

1. READ LOCK: This lock allows a user to only read the data from a table.
2. WRITE LOCK: This lock allows a user to do both reading and writing into a table.

Query:-

LOCK TABLES T_name [READ | WRITE];

We can lock multiple tables together too.

Query:-

LOCK TABLES T1_name [READ | WRITE],

T2_name [READ | WRITE],. ,

Tn_name [READ | WRITE];

Ex- Now let us suppose we want to lock our table Ninja to read only we can write -

LOCK TABLES Ninja READ;

Now we can only retrieve data from the Ninja table i.e we can not make changes to it.Similarly, if we use the WRITE then we restrict users from both reading and writing.

Import Export

- **Importing :**

When importing from a local computer, the client program reads the file on the client and sends it to the MySQL server.

The file will be uploaded into the database server operating system.

Query: -

```
LOAD DATA LOCAL INFILE 'c:/tmp/xyz.csv'
INTO TABLE T_name
                FIELDS TERMINATED BY
                ',' ENCLOSED BY '"'
                LINES TERMINATED BY
                '\n' IGNORE 1 ROWS;
```

- **Exporting :**

To export our data into a CSV file.

Query:-

```
SELECT column_name(s)
FROM T_name
WHERE id = 1 INTO
OUTFILE 'C:/tmp/xyz_exported.csv'
                FIELDS ENCLOSED BY
                '"' TERMINATED BY ';'
                ESCAPED BY '"'
                LINES TERMINATED BY '\r\n';
```

What is a functional dependency?

To define this in simpler terms, it's a relationship between the primary key attribute (usually) of the Relation to that of the other attributes in that relation.

Suppose, M is the Primary key attribute of the Relation XYZ. Now there's another attribute named N. Now let's suppose that attribute 'N' has certain data entries with the same values and to distinguish among them or you say to accurately determine the values in the attribute 'N' we take help of the Primary key attribute 'M' and we determine values for attribute 'N' through it.

Now this is called a functional dependency of 'N' on 'M'.

We usually represent Functional dependency with an arrow symbol. As shown below:

$\longrightarrow$

To represent the above stated situation with the symbol, it will be like,

$M \longrightarrow N$

Here, we call 'M' the determinant and 'N' the dependent.

Note: Usually the determinant is the Primary key of the relation, but there could exist the case where a certain set of attributes can be determined out of some attribute but that attribute isn't the primary key attribute.

Let's take a small example to visualise functional dependencies better,

C1	C2
7	21
3	100
7	5
5	21

Here, if we have to consider the permutation and combinations of all FD's relations (whether possible or not) we can have 2 relations that is

C1$\longrightarrow$ C2

C2⟶ C1

Now on this instance,

- If we observe the values carefully in the Relation above it's quite safe to say that **C1⟶ C2 does NOT hold on the above relation** because '7' appears in C1 twice and due to that we won't be able to determine correct corresponding values in C2. Hence we can't determine C2 from C1.

- Whereas, **C2⟶ C1 does hold on the relation,** as C1 uniquely identifies each value in C2.

We can also define FD's as the legal relations of the given instance of table/relation. Like in the above example, C2⟶ C1 is a legal relation while C1⟶ C2 is an illegal one.

Why do we need Functional Dependencies ?

It helps us to maintain the quality of the data in the database. As we have already familiarised with the notion that Functional Dependencies is a relationship between columns/attributes of a Relation/Table dependent on each other.
How does it do that?

- Functional dependencies check the legality of the relation tuple under a given set of FDs.
- It defines the constraints on the Relation.

Rules of Functional Dependencies:

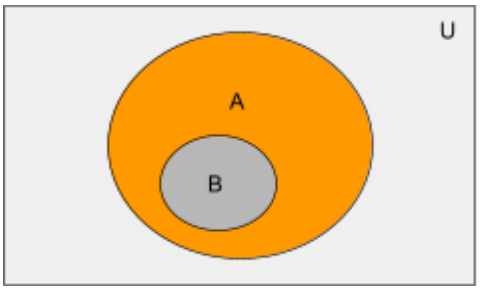
1. Reflexive Rule: If 'A' is a set of attributes and 'B' is a subset of 'A'. Then, A⟶ B holds.
2. Augmentation Rule: If B can be determined from A, then adding an attribute to this functional dependency won't change anything.
   i.e. If A⟶ B holds, then AM⟶ BM holds too. 'M' being a set of attributes.
3. Transitivity Rule: if A determines B and B determines C, we can say that A determines C.
   i.e. if A⟶ B and B⟶ C then A⟶ C

   Note: These rules are also known as Armstrong's axioms.

Types of Functional Dependencies:

1. **Trivial Functional Dependency:** Let there be set of attributes A which determines set of attributes B (i.e. A⟶ B).
   Now if the dependent (i.e. B) is a subset of the determinant (i.e. A).
   i.e. If A⟶ B and B⊆A
   Hence, we call this a trivial functional dependency.

   Venn Diagram for better understanding:



   Note: A set is also a subset of its own, i.e. A⊆A. Hence, A⟶ A is also a trivial functional dependency.

For Example:

owner_id	owner_name
2352352	Ted Mosby
3523523	Ross Geller
7485685	Joey Tribbiani

Consider the above given table with attributes named owner_id and owner_name.
Here, {owner_id,owner_name} ⟶ owner_name is a trivial functional dependency since owner_name is a subset of {owner_id,owner_name}.
owner_id⟶ owner_id is also a trivial functional dependency.

2. **Non-trivial Functional Dependency:** Let there be set of attributes A which determines set of attributes B (i.e. A⟶ B).
   Now if the dependent (i.e. B) is **NOT** a subset of the determinant (i.e. A).
   Hence, we call this a Non-trivial functional dependency.

For Example:

owner_id	owner_name	cars_owned
2352352	Ted Mosby	Pontiac Fiero
3523523	Ross Geller	MG model B
7485685	Joey Tribbiani	Porsche

Consider the above given table with attributes named owner_id, owner_name and cars_owned.

Here, owner_id⟶ owner_name , {owner_id,owner_name} ⟶ cars_owned are examples of non-trivial functional dependencies as their dependent isn't a subset of its respective determinant.

What are Anomalies?

According to the dictionary meaning anomalies means an abnormality. In regard to DBMS, here we will discuss the abnormalities that might be present in the database due to data redundancy.

Basically there are three types of Anomalies:

- Insertion anomaly
- Updation anomaly
- Deletion anomaly

These anomalies are the problems that we will solve using Normalisation. The occurrence of these data anomalies in the database is natural.

Let's understand the three operation which cause anomaly using an example:

Roll_no	Name	Department	HOD	Dept_contact
22	Sonam	Prosthodontics	Dr. Neeta	98674534
39	Rohan	Prosthodontics	Dr. Neeta	98674534
07	Ajay	Prosthodontics	Dr. Neeta	98674534
21	Priya	Prosthodontics	Dr. Neeta	98674534

In the table above we have a database of 4 Students of final year pursuing BDS. They have been given an opportunity to choose a department for their monthly posting and they chose "Prosthodontics".

Now we can observe that the attributes like department, HOD are repeated for all the students that opted for it in the University.
This is called Data Redundancy.

1. **Insertion Anomaly:** Suppose that a new student signs up for the posting but hasn't chosen his/her department of practice yet. Now, until or unless that student makes up his/her mind about which department they need to work for, we can't insert the data or if we do we need to set the Department Attribute to NULL.

Secondly, now if we had to insert data of like 50 students signing up this department (i.e. Prosthodontics), then the department information will be repeated for all those entries.
This is known as Insertion Anomalies.

2. **Updation Anomaly:** Suppose Dr. Neeta got an offer from abroad and she decided to leave the university. That means she won't be the HOD of the Prosthodontics department anymore for the university.
   Now if that is to happen, the DBA (Database Administrator) of the university has to update the student database and in that too he needs to update each record present in the database, which contains Dr. Neeta as HOD of Prosthodontics.
   Now if due to human error or by any means any of the records is missed, it will lead to Data Inconsistency.
   This is Known as Updation Anomaly.

3. **Deletion Anomaly:** In our table, two different pieces of information are kept together, Student information and Department information. Therefore, at the end of the posting of students when they shift to another department, if student records are deleted, we will also lose the Department information.
   This is Deletion anomaly.

Hence, to deal with these anomalies or abnormalities in the database, we use normalisation.

Normalization:

We need to perform Normalization on our database to reduce Data Redundancy.
It minimizes redundancy using certain rules or sets of rules.

There are many types of normal forms, although we are going to focus on 1Nf, 2NF, 3NF and BCNF (also known as 3.5 NF).

Types of Normal Forms:

1. **First Normal Form (1NF):** This is the Step 1 of the Normalisation Process.
   For a Relation/table to justify 1NF it needs to satisfy 4 basic conditions:
   - Each attribute should contain atomic values. (i.e. No multivalued attributes)
   - Each Value stored in an attribute should be of the same type.
   - All the attributes in a table should have unique names.
   - The order of the data stored in the table doesn't matter.

   Let's take a look at an Example:

Chrac_id	Chrac_name	Actor
54342	Iron Man	Robert Downey Jr
33243	Spiderman	Tobey Margiue, Andrew Garfield, Tom Holland
24352	Loki	Tom Hiddleston
74564	Thor	Chris Hemsworth

Above is the table of Characters with the Actors name who played that certain character over the years for Marvel Cinematic Universe.

Now on observing the data above, if we apply the rules of normalisation known to us and check whether this data is in first normal form or not.

We observe that 3 out of 4 rules are followed, that is,

- Each attribute has a unique name,
- Each attribute contains data of the same type and

- the order of the data is the form we want it to be.

But the value of data in each attribute isn't atomic, for example, the 2nd instance of the above relation has multiple values in the Actor attribute as Spiderman was played by three different actors over the years.
This is a violation of 1NF.

Now, how to solve this problem, to bring the table to satisfy all conditions/rules of the first normal form we need to break the multiple values into atomic values.
Therefore, now the above table will look like:

Chrac_id	Chrac_name	Actor
54342	Iron Man	Robert Downey Jr
33243	Spiderman	Tobey Margiue
33243	Spiderman	Andrew Garfield
33243	Spiderman	Tom Holland
24352	Loki	Tom Hiddleston
74564	Thor	Chris Hemsworth

Now the above table is said to be in 1NF.

2. **Second Normal Form:** For a Relation/table to justify 2NF it needs to satisfy 2 rules:
    - It should be in First Normal Form.
    - It should not have any partial dependencies i.e. when a nonprime attribute is derivable from only a part of a candidate key.

Let's take a look at an Example:

charac_id	movie_name	Charac_rating	Director
54342	Iron Man 2	9	Jon Favreau
33243	Spiderman	10	Sam Raimi
33243	The Amazing Spider Man	7	Marc Webb

53463	Iron Man 2	9	Jon Favreau
74564	Thor: Ragnarok	9	Taika Waititi
54342	Iron Man 3	9	Shane Black

Above table lists famous movies with their main characters and the actors it was played by along with the movie director.

Now for the above Relation to be in 2NF, it should be in 1NF, which it is already in. Secondly, there should be no partial dependencies.

Although, here we can observe that, (charac_id + movie_name) acts as the primary key (which is a chosen candidate key).

Although, in the above relation from movie_name attribute Director attribute can be derived and has nothing to do with charc_id attribute.
Now an attribute is only dependent on part of the primary key but not on the whole set, hence it's called Partial Dependencies.

How to fix this?
One of the simplest solutions is to remove the Director Attribute from the table.

The Table becomes:

charac_id movie_nameCharac_rating

54342	Iron Man 2	9
33243	Spiderman	10
33243	The Amazing Spider Man	7
53463	Iron Man 2	9
74564	Thor: Ragnarok	9
54342	Iron Man 3	9

And now it is in Second normal form.

And the attribute Director gets adjusted to other relation in the database as below:

movie_name	Director
Iron Man 2	Jon Favreau
Spiderman	Sam Raimi
The Amazing Spider Man	Marc Webb
Thor: Ragnarok	Taika Waititi
Iron Man 3	Shane Black

Let's continue with the types of Normalisation:

3. <u>Third Normal Form (3 NF):</u> For a table to be in the third normal form:

- It should already be in the 2NF.
- It should not have Transitive Dependency.

Let's take a look at an Example:

std_id	sub_id	marks	exam_name	total_marks
3160	9	85	Maths - II	100
7067	10	98	Ortho	120
3160	7	70	CS271	90
5276	9	78	Maths - II	100
3523	4	66	CE201	100

Above is the Score table, with attributes stating Student marks in the examination.
The primary key for Score table is a composite key, i.e. { std_id + sub_id }
Let's analyse the dependencies of attributes on the primary key,

Attribute marks are dependent on both student and subject as it tells us, whose (student) marks are these and in which subject these marks are scored.

Marks can be determined using a combination of std_id and sub_id, because if we are asked to get the marks of a student with id 3160, we won't be able to fetch anything as we don't know for which subject. Similarly, when asked about marks for subjects with ID 9, we won't get anything as we don't know for which student.

So we can say that marks are dependent on both student_id and subject_id.

Attribute exam_name depends on both student and subject.
For illustration, a Chemical engineering student (std_id=3523), will have a Lab exam but all the engineering mathematics students won't have it. Also, some subjects might have practical exams and some might not.
So we notice that exam_name is dependent on both student_id and subject_id.

Now last but not the least attribute total_marks, this one depends on exam_name because see different exam types different weightage.
For example, different markings for practical and theoretical exams.
But, exam_name is not a primary key or a part of the primary key and total_marks depends on it.
Hence, exam_name⟶ Total_marks (i.e. nonprime attribute ⟶ nonprime attribute), and { std_id + sub_id } ⟶ exam_name.
Therefore, by working out the transitive rule of FD's, { std_id + sub_id } ⟶ Total_marks.
This is Known as the Transitive dependency, Which is a violation of Third Normal form.

How to fix this ?
Create another table named Exams with attributes exam_name and total_marks along with exam_id use it when required for reference.

Table Score:

std_id	sub_id	marks	exam_id
3160	9	85	3
7067	10	98	1
3160	7	70	2
5276	9	78	3
3523	4	66	3

It's in Third Normal Form.

Table Exams:

exam_id	exam_name	total
1	Practicals	120
2	Labs	90

3	Theoreticals	100

4. <u>Boyce-Codd Normal Form (BCNF):</u> It is an extension of 3NF and is also known as 3.5 Normal Form.

For a table to be in the Boyce-Codd Normal Form, it should satisfy 2 rules:

- It should be in the Third Normal Form.
- A prime attribute shouldn't be dependent on a non-prime attribute. (i.e. if M→ N, then M is a super key)

Let's take a look at an Example:

ninja_id	Actor	Movie
1	Robert Downey Jr	Iron Man 2
5	Chris Evans	Avengers: Endgame
5	Elizabeth Oleson	Avengers: Infinity War
7	Chris Evans	Captain America: Civil War
4	Chris Hemsworth	Thor: Ragnarok

Now the above table is formulated by surveying every ninja's favorite Marvel actor along with the movie in which they liked that actor the most.

We can observe that a single ninja might have multiple favorite Marvel Actors and there is also a case present that certain ninjas like the same Actor but in different movies like ninja 5 likes Chris Evans in movie Avengers: Endgame, whereas ninja 7 likes Chris Evans in the movie Captain America: Civil War.

In This table ninja_id and Actor combined can be used as Primary key to determine all the other table attributes.

Now if we only talk about this table, we can also observe that the ‘Actor’ attribute is being determined from the ‘Movie’ attribute.
So the FD’s will be like,
{ninja_id, Actor} ⟶ Movie
Movie ⟶ Actor

But Actor is a prime attribute, as it is part of the candidate key, whereas movie is a non-prime attribute.
Hence, in this table above we have a prime attribute being dependent on a non-prime attribute. Hence the table isn’t satisfying BCNF.

How to fix this?

To make the table satisfy the BCNF conditions we need to split the above table into two.
One table containing the ninja_id and the movie_id.
Another table containing the movie_id, the movie and Actor.

Note: The above table satisfied the Third normal form, then only we evaluated for the second rule of BCNF.

What Is Transaction ?

It is a unit of program execution by the user or application that can access and update different data points.
It is a logical unit of work that contains one or more SQL statements.
The result of all these statements in a transaction either gets completed successfully (all the changes made to the database are permanent) or if at any point any failure happens it gets reversed (all the changes being done are undone.)
Eg: Transfer 70 rupees from your account(X) to your friends account(Y):

Transaction:

1. READ(X)
2. X=X-70
3. WRITE(X)
4. READ(Y)
5. Y=Y+70
6. WRITE(Y)

A transaction can be a whole program, or even a single clause command.

Before we discuss further about the properties of transaction, let's have a quick look at the issues that we might face within a transaction:

- Hardware failures or system crashes.
- Concurrent execution of multiple transactions.

These issues can result in loss of data consistency. In order for that to not happen, We have certain properties of Transaction that we need to follow.

They are called **ACID** properties.

Transaction Properties:
ACID stands for:

- Atomicity
- Consistency
- Isolation
- Durability

Let's understand them briefly with the example of the transaction that we also performed above.

➔ Atomicity:

Now, suppose there are system failures(software or hardware) due to which transaction fails after step 3 and before step 6. The money that was being

transferred will be lost(i.e. It has been deducted from your account but hasn't been added to your friends account), this leads to the inconsistency in the database state.

Therefore, In layman terms, 'all or nothing' property. That is, **if a transaction is to happen either it will be performed entirely or will not be executed at all.**

We know a transaction can have a set of operations in it, but we consider one whole transaction as one unit.

Also it involves two operations:

- ➢ Abort: If we abort the transaction, the changes that might have happened due to the transaction to the database aren't visible.
- ➢ Commit: If we commit the transaction, the changes that might have happened due to the transaction to the database are visible.

➔ Consistency:

Now, the sum of money in the accounts before the transaction should be equal to the sum of money in the accounts after the transaction.

Therefore, we define consistency as the effect of integrity constraints on the database due to which data remains consistent before and after the transaction when it transfers the database from one state to another.

During the transaction, the database can be inconsistent.

➔ Isolation:

Now suppose while this transaction (mentioned above) is taking place and another transaction has started in between and it is accessing the partially updated database.

Now that transaction will encounter an inconsistent database.

Therefore, isolation ensures that the transactions are executing independently, i.e. when one transaction is being done, it won't be interrupted by the other one. Although multiple transactions can happen simultaneously, given that each transaction is unaware of the other concurrently executing transactions.

➔ Durability:

Now suppose, you received a message on your mobile phone stating that Rs. 70 have been debited from your account to transfer to this account.
But your friend states that he hasn't received the money yet.
Now this could happen due to the software or hardware crash, but the updates to the database by the transaction has been made and they won't be lost even after the system broke down.

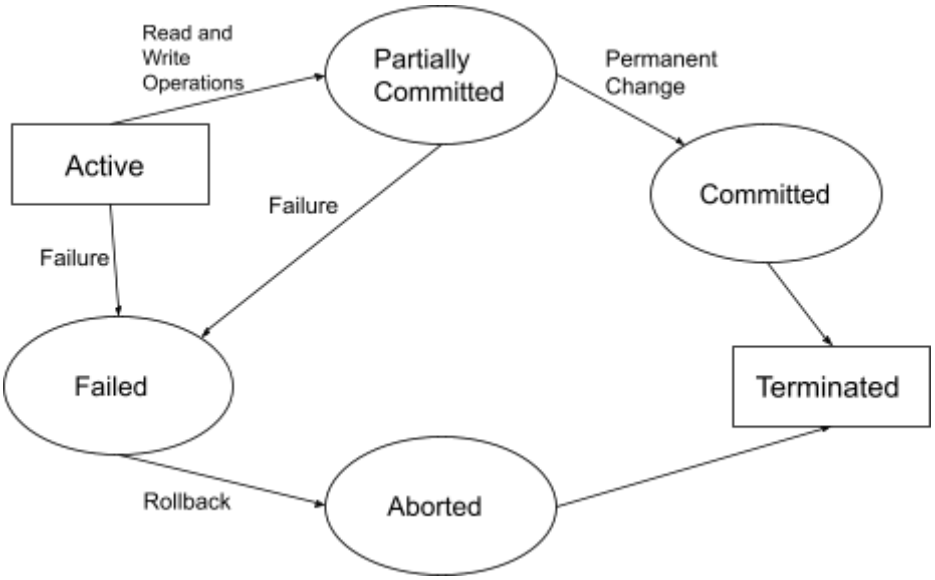
Therefore, This property ensures all the changes or updates to the database have been recorded and have been stored and will be never lost even if the system crashes.

What are transaction states?

A transaction goes through numerous states throughout its life cycle.
These states are known as transaction states.

The states are:

1. Active state
2. Partially committed state
3. Committed state
4. Failed state
5. Aborted state
6. Terminated state



- **Active State:-** The very first state of the life cycle of transaction, all the read and write operations are being performed and if they execute without any error the transaction comes to ‘partially committed’ state, although if any error then it leads to ‘failed’ state.
  Note: All the changes made by the transaction now are stored in the buffer in main memory.

- **Partially Committed State:-** After the last command of the transaction is executed the changes are saved in the buffer in Main Memory. If the changes made are permanent on the Database then the state will transfer to the ‘committed’ state and if there's any kind of failure it will go to the ‘failed’ state.

- **Committed Sate:-** When the updates are made permanent on the database. Then the transaction is said to be in Committed state.
  Note: Rollback can't be done from here. At this state, a new consistent state is achieved by the database.

- **Failed State:-** When a transaction is being executed and some failure occurs due to which it becomes impossible to continue the execution. Transaction enters into a failed state. We can come to this state from an Active or Partially committed state.

- **Aborted State:-** From Failed State, when all the changes made in the buffer are reversed, and now that transaction needs to Rollback completely it enters Aborted state.

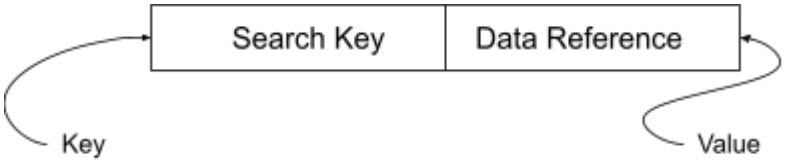
- **Terminated State:-** This is the final state of the life cycle of the transaction. It is where it ends. A transaction can come to this state only from a 'committed' or 'aborted' state.

After completing the cycle, now the database is consistent and is ready for new transactions.

What is Indexing?

Indexing is a method to help us locate a record/data present in the memory faster and quicker. It gets this done by minimizing the number of disk accesses required when we process a query.
It is a Data Structure that we use to perform the above stated operation.
Indexes contain few database columns:

- The first column is the Search Key, which contains the copy of the Primary key of the table so that the data access time could be reduced which means data can be accessed quickly. Order of the key may or may not be sorted.
- Second column is the Data Reference. It contains pointers holding the address of the disk block where the value corresponding to the key is stored.



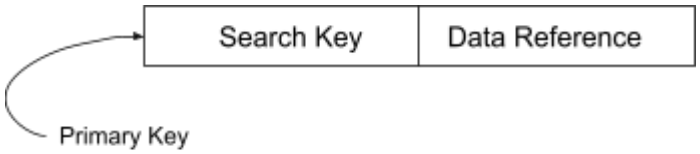
Why do we need Indexing?

Usually, many queries reference only a small proportion of the records in a file.
Suppose, you need to find all instructors in the “Anatomy department”.

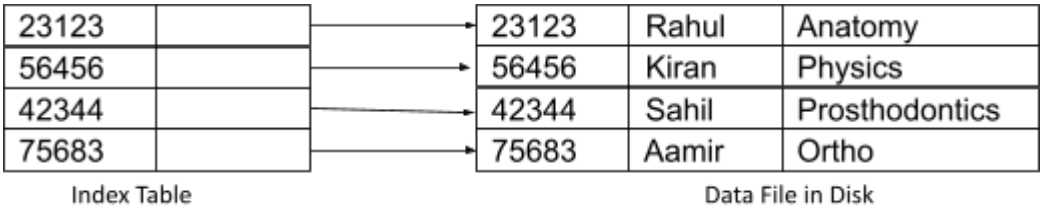
- Now, It is quite inefficient for the system to read every tuple in the instructor relation to check if the dept_name value is “Anatomy”
- To quickly access, our DBMS stores additional structures associated with the files called indices.
- To retrieve Professors records given a department name, the database system would look up an index to find on which disk block the corresponding record resides and then will fetch the disk block, to get the appropriate Professor record.

Types of Indexing:

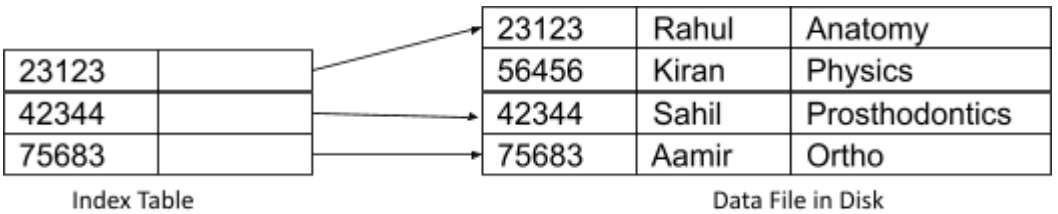
➢ **Primary Index:-** It is the index that is created and ordered on the basis of the primary key of the table. It is further of two types Dense and Sparse Index.



- **Dense Index:-** An index record appears for every search-key value in the file, i.e. number of records in the dense index table is the same as the main table. In this the structure contains the search-key value and a pointer to the data record with that search-key value.
  This type of indexing is not space friendly, as it takes extra space to store the index table.



- **Sparse Index:-** An index record appears for some of the search-key values in the file**.** To locate a record when this type of indexing is performed on the datafile, we lookout for the largest value key in the table which is less than or equal to the our desired search key and once it is found, we then search that key sequentially starting from the row where the pointer of the index table pointed us till we find desired key and the record corresponding to it. The space efficiency for this is better than that of Dense but it is slower than dense index.



.

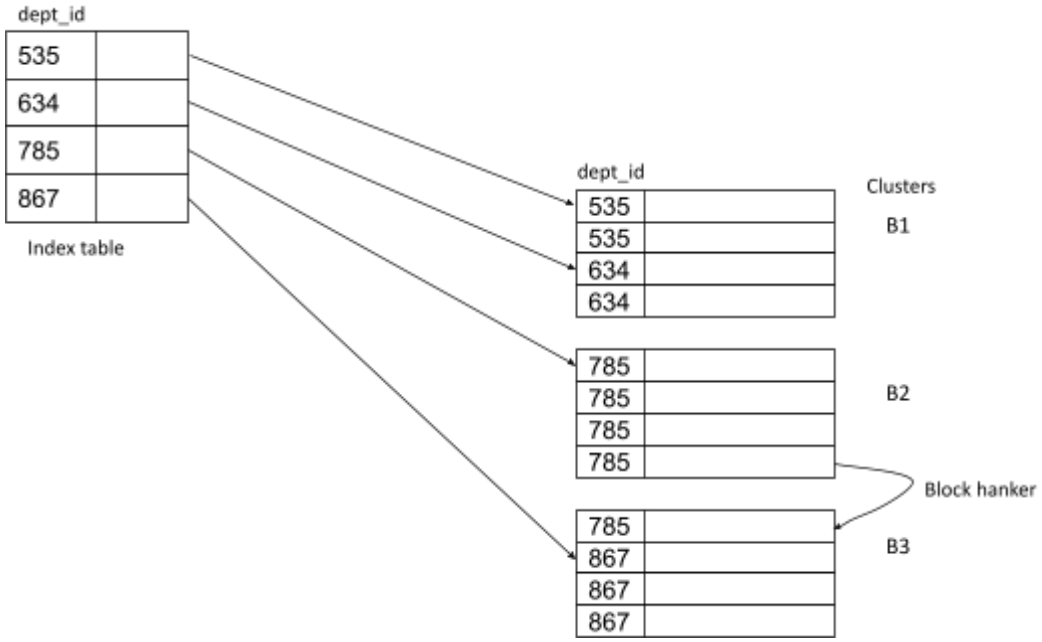
Difference Between Dense and Sparse Indexing:

Dense Index	Sparse Index
Space taken for the index table is large.	Space taken for the index table is smaller.
Time taken to locate the record is less in comparison	Time taken to locate the record is more.
The records in the data file are in specific order and need not be in any kind of cluster or chunk.	The records in the data file are in specific order but the data records are in a cluster or chunk. (i.e. pointers from the index table point to certain data records, and all records between those pointers are considered in one cluster or chunk.)

Types of Indexing:

- **Clustering Index:-** It is the index that is created and ordered on the basis of the non-primary key attributes of the table which are not known to be unique for each record. In Clustering index, to fetch a record we group two or more attributes together to get the unique values and create an index out of them**.**

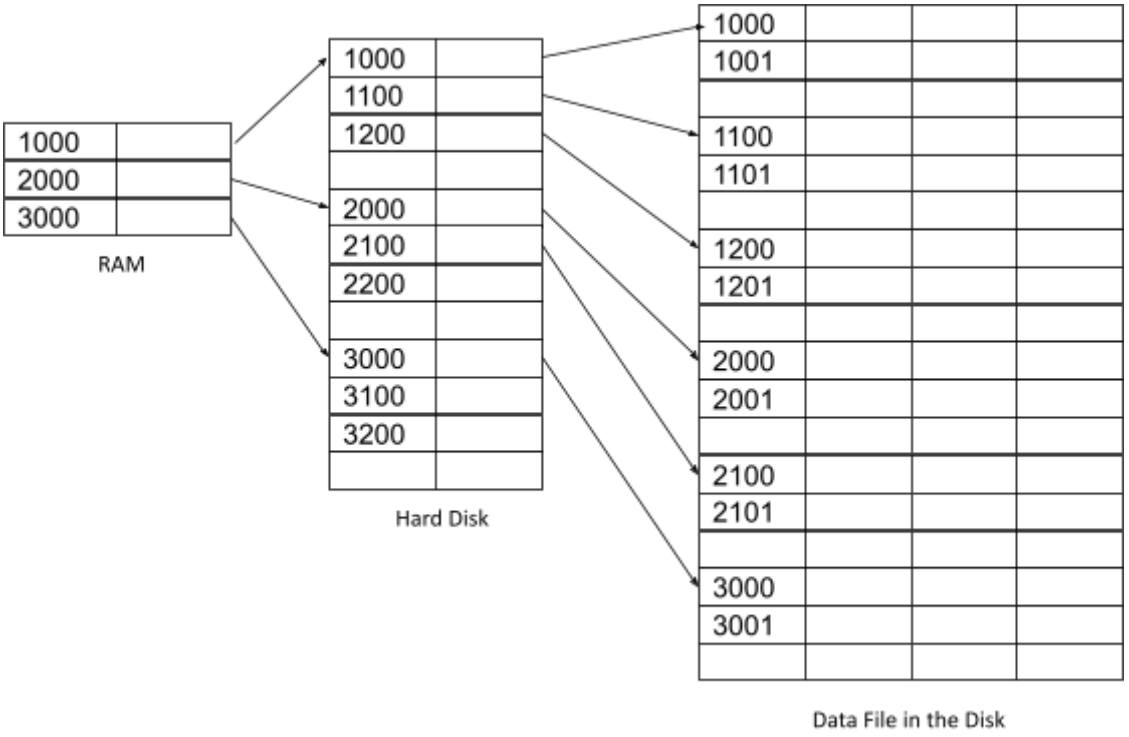
  Example: Suppose a Dental college contains several students posted in a particular department.
  Suppose we use a clustering index, where all Students which are posted in the same department (i.e. same dept_id) are considered within a single cluster, and index pointers point to the cluster as a whole.
  Also, here dept_id is a non-unique key.



  Here, we have 3 memory blocks as clusters. We can see two id's present in a single block and also a case when all the records corresponding to one key cannot be contained in one block it gets stored in another blck and is connected to the previous block with another pointer named Block Hanker.
  We can see, The pointer from the index table points to the first record of the main table corresponding to that particular key id.
  These cluster blocks are present in the main memory.

➢ **Secondary Index:-** It is introduced to tackle the problem when the size of datafile increases i.e. size of table increases, sparse indexing starts to slow down. Now to overcome this we introduce another level of indexing.
In this, we select a huge chunk of columns initially and put it up at the first level of indexing that is Primary Level index, it is stored in primary memory.
Now each chunk is divided into smaller ranges in the second level of indexing, this is stored in the secondary memory along with the actual datafile.

Example:



In this, say if we are looking for a record with search key value S. We have to find an index record with the largest search key value less than or equal to S.
Remember, this searching of the record in the file happens sequentially.
For example we need to find the record corresponding to the key value 2101.
We will start looking for the key, in the first index table present in the RAM, whose value will be either less than or equal to 2101.
So, upon searching we will fetch key 2000 from the first index table.

Now we will perform the same step of searching for the second index table stored at Disk as we did above for the first index table.
Similarly, upon searching we will fetch key 2100 from the second index table.
We will move to the third table in the data file and again repeat the very first step for searching.
Now, upon searching we will find out the desired key, i.e. 2101.
Remember the above procedure will keep continuing until the key is found. (if it exists in the table.)

Note: This procedure is also the same for DML operations like Insert, Update and Delete.

Hence, we need Indexing because,

- We can access and retrieve data faster.
- Indexing helps us with sorting the data, as index is a sorted data structure.
- Indexing reduces the number of I/O operations needed to be performed for retrieving data.

Drawbacks of Indexing:

- They take up additional space to store the index table.
- Indexing slows down INSERT,UPDATE and DELETE operations.

What is NoSQL?

NoSQL, popularly misunderstood as No-SQL but it actually means **Not only SQL database**. It's a mechanism for storing and retrieving data and is also claimed as the next generation database.They are used in real-time web applications and big data and their usage is peaking over time.

Earlier databases weren't compatible with the scale and agility of modern applications, nor were they able to optimally use commodity storage and processing power which is available today. They, with time, weren't able to catch up with the flow of information demanded by people.

Hence, to cope up with all the parameters, in mid 1990's a non-relational database named NoSql was introduced.

- These are schema free
- Data structure being used for it is different from what we learned in Relational databases, data structures used here are more flexible than the relation used in relational databases.
- It can handle huge amounts of data (big data) and it's performance can be tuned by adding more machines to our clusters.
- Most of the NoSQL are open source and have a capability of horizontal scalability which means that commodity kinds of machines could be added.

Why NoSQL?

The Relational Databases we use are not designed to manage all kinds of data efficiently, like structured, unstructured and semi-structured data whereas NoSQL databases are designed to manage these types of data efficiently.

NoSQL databases go against the conventional attitude of storing information at a single location, instead it distributes and stores information over a set of multiple servers. The data are stored in flexible and fluid data models. This distribution of data helps the NoSQL database server to distribute the load at the database tier therefore it also signifies that the system can scale out rather than just scale up.

What is Scaling?

It is the ability to expand or concise the capacity of the system by either adding or removing the resources from it, in order to alter the usage of our application.

There is mainly two types of scaling:

- ➔ Horizontal Scaling.
- ➔ Vertical Scaling.

Horizontal Scaling:

Horizontal Scaling, also known as **Scale out,** refers to addition of extra servers/nodes to distribute the load among them.

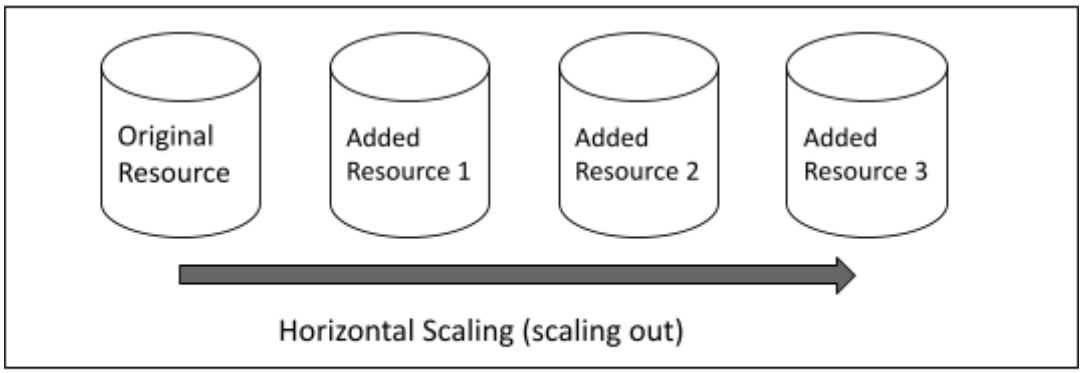
Now, this type of scaling puts up a little difficulties when dealing with relational databases because maintaining those relations while spreading the data out is tough to work out.

Although if we take into account the non-relational databases, then scaling out doesn't seem to be a bad option.

Therefore, to be little precise Horizontal Scaling refers to the addition of resources to the existing system to optimise the performance of the system accordingly.

A small example:

Original System



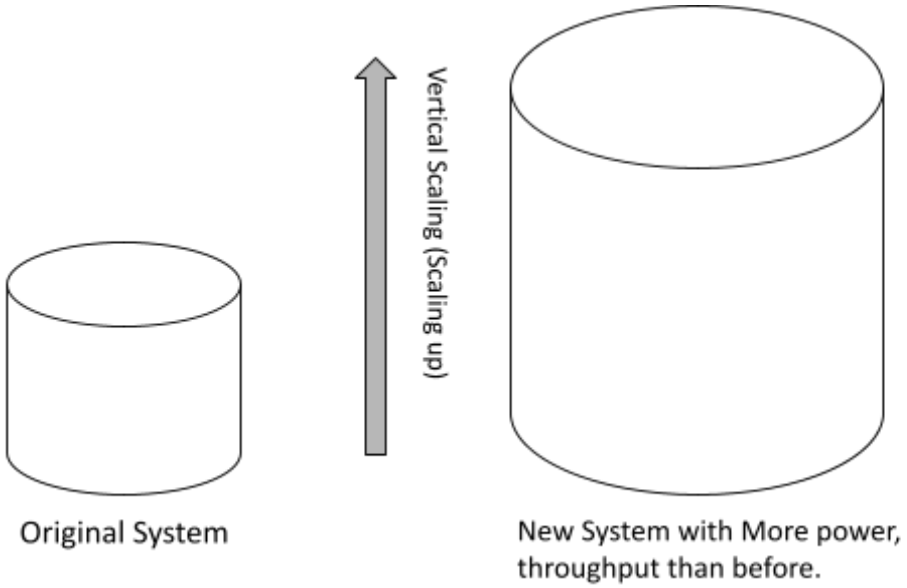
A new System with added resources

Three resources were added in the resources pool to build up a new system.

Vertical Scaling:

Vertical Scaling also known as **Scale up**, refers to increasing the power of the existing system/machine.
This type of scaling can be adapted by both relational as well as non-relational databases. But as access to everything is bad, scaling up also has a certain limit after which doing it further makes no sense.
Instead of combining multiple machines/servers we build one huge machine/server having much more power and throughput than the previous system.

Below is how scaling up is done.



Difference between Horizontal Scaling and Vertical Scaling:

Horizontal scaling	Vertical Scaling
Horizontal Scaling refers to the addition of resources to the existing system.	Vertical Scaling refers to increasing the computational power of the existing system/machine.
Difficult to implement.	Relatively, easy to implement.
Cost is expensive.	Not that expensive.

Advantages of NoSQL:

- **Horizontal Scaling (scaling out):** Databases have been scaling out for years and DBAs had been dependent on scaling up relational databases so that they could increase the efficiency. Scaling up means stacking up larger servers like the load raises or increasing the head count of hardware assets for the existing machine to overcome its limitations.
  But a point will arrive when more scaling up will not be possible as a machine can handle a particular amount of hardware assets and scaling out is not possible with relational databases because of it's technical restrictions. Scaling out means distributing the database throughout multiple computers when the load increases. Therefore, it can handle more traffic simply by adding more servers to the database.
  They have the ability to become larger and much more powerful, also it is quite cost effective. Hence, making them the preferred choice for large or constantly evolving data sets.

- **High Availability:** NoSQL databases are highly available due to its auto replication feature i.e. whenever any kind of failure happens data replicates itself to the preceding consistent state.
  Not only this, but whenever a server fails, we can access that data from another server as well, as in nosql database data is stored at multiple servers.

- **Flexible Schema:** Relational databases are used to have a defined schema, which is quite an issue because in case you need to make any modification or addition to the database we need to change the schema as well every time. Whereas in NoSQL databases, we could start working with the data without schema.

- **Shrading:** It is a database partitioning technique used by b lockchain companies with the purpose of scalability. In case of NoSQL it is an automated process that is the database is partitioned among a random number of the servers on it's own, with no requirement for application to be aware of it.\

- **Integrated Caching**: It's a technique of keeping frequently used data in the system memory for its instant use which removes the need for a separate

caching layer. NoSQL is equipped with such intelligent integrated caching capability.

- **Less Management:** NoSQL databases require minimum to zero management, they are equipped to auto-repair data distribution and due to it having flexible data models that results in reducing administration and performance difficulties.

Note: NoSQL also has certain other benefits like more compatible for cloud, helps raise funding for business as it is one reliable source for data storage and performing analytics as well.

Disadvantage of NoSQL:

- When compared to SQL or relational databases, NoSQL database is designed for storing data of many types but it lacks functionality like it doesn't support Transaction properties like ACID.
- The process of managing big data on NoSQL is quite complex as compared to RDBMS.
- It is also not compatible with SQL, although few NoSQL databases do use SQL but many don't. Hence, will require manual query language which is quite slow and highly complex.
- It doesn't support data entry with constraints like RDBMS.
  For eg: There's an on-campus placement going on for a company in multiple colleges. Now, they maintain a database of all students applied for the Job role. Now we need to enter only those whose CGPA is above 7.5.
  Here, NoSQL wouldn't be able to get the job done, whereas RDBMS will get the job done.
- Other concerns for NoSQL are standardization, Interfaces and Interoperability and also less community support(due to it being new in the industry).

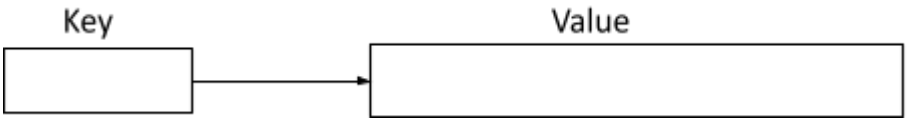
Key-Value NoSQL:

NoSQL has become quite popular lately, with some of the most popular platforms and services relying on it to deliver content to use with lightning speed. It has a variety of database types, but the most popular of them is Key-Value storage or we call it Key-Value NoSQL Database.

Its data model is extremely simple, which allows it to perform better and faster than a relational database. It also allows flexibility in storing and easy retrieval of data.

- **How does this work ?**
  It is quite simple to understand though. A value that is basically any piece of data or information is saved with a key to identify its location at time of operation. It's design concept is similar to that of an array or map in any programming language.
  Although it's stored in a continuous manner in the database system.

  It's representation is as shown below:



  Note: Now here value can be any piece of data in any format.

  They are considered so good because here information is stored as a sort of opaque blob (sizable piece of data), rather than discrete data.
  No index is required for the database to boost its performance. It is already well equipped within its structure to perform optimally.

  It depends on certains command to operate it like,
  Like,
  1. get(key): fetch us the value of the given key.
  2. put(key,value): Creates or updates a value of the given key.
  3. delete(key): Deletes the value for the given key.

4. execute(key): Invoke an operation to a value of the given key.

Although this may lack control of data, it gets overshadowed with the fact that this db is so fast and reliable.

- **Some Benefits of Key-Value NoSQL database:**
  It holds certain benefits with it like,
  - Highly Scalable Horizontally
  - Simpler queries
  - Mobility i.e. they are easy to move from one system to another without the need for new architecture or changing the code.
  - Handling concurrency issues better as only need to resolve one key.
  - Consistency.
  - Easy and fast retrieval of data.
  - Integrated caching

➢ Key-Value NoSQL Database - *Redis, Amazon DynamoDB, Oracle NoSQL, Aerospike*

- **When to use these kinds of databases?**
  Our Relational databases are not really compatible to handle a heavy number of read/write operations, which is where key-value db's shine.
  Since it's easily scalable, it can handle a high number of users at any given point.
  Also, with it's built-in redundancy feature, it can take care of lost storage or data without any difficulties.
  It is also preferable to be used when dealing with:
  1. Fulfills need to provide users with session management on quite a large scale.
  2. Can help in the formation of recommendation engines for products or anything.
  3. Can be used to advertise Advertisements to users depending on their current data usage.

  It is also preferred to be used for big data research or even for the seasonal surges in purchasing on platforms.

- **Limitations:**

- No relationships among Multiple-Data.
- Can’t Query Data by 'value'.
- Query is performed on one key at a time.

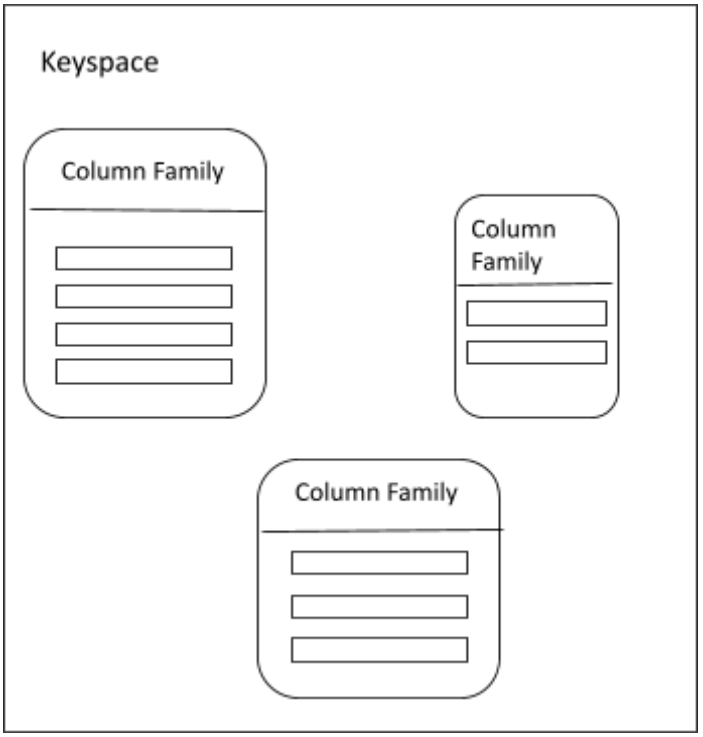
Therefore, the real deal with the Key-Value NoSQL database is that it’s simple. Although it might create issues when dealing with complex issues like financial transactions.
Its purpose is to bridge the gaps of relational databases.However,, by mixing both relational and non-relational together we can come up with a more efficient pipeline, being it for data analysis or managing users.

Columnar NoSQL Database:

This kind of database stores data in columns instead of rows. It speeds up the read and write process from the memory to return a query faster. It stores data in a way that greatly improves disk I/O performance.
They are majorly helpful for data analytics and data warehousing.

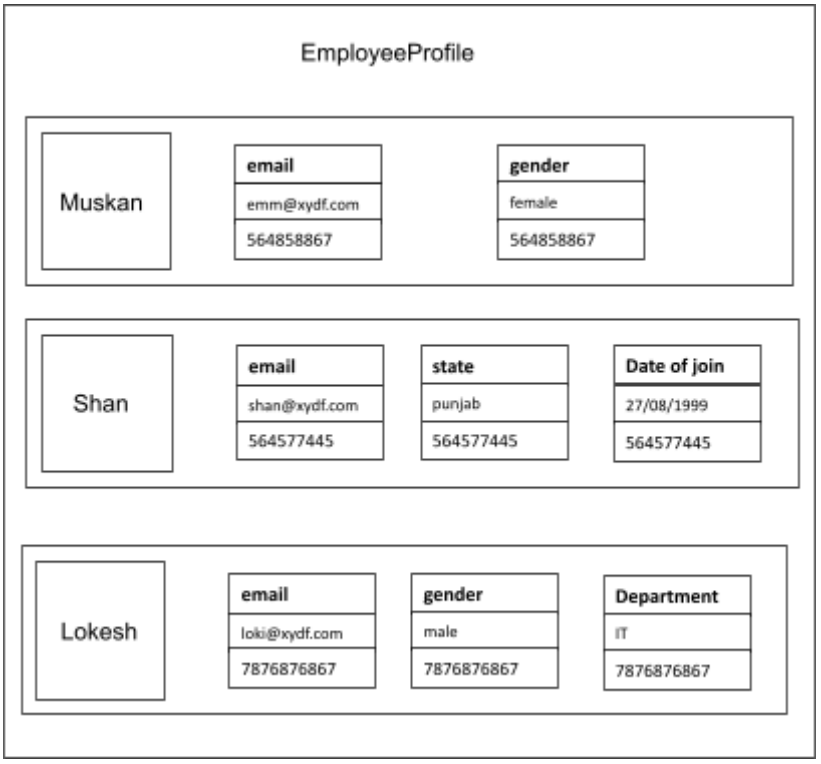
- **How does this work ?**
  Data is organized into columns instead of rows. Hence they function same as those tables in relational databases but being a NoSQL it's model allows it to be much more flexible.
  It uses the concept of keyspace, which is similar to schema in relational models. It contains all the column families, which further contain rows, which then contain columns.

  Below is the representation of column families in the key space,



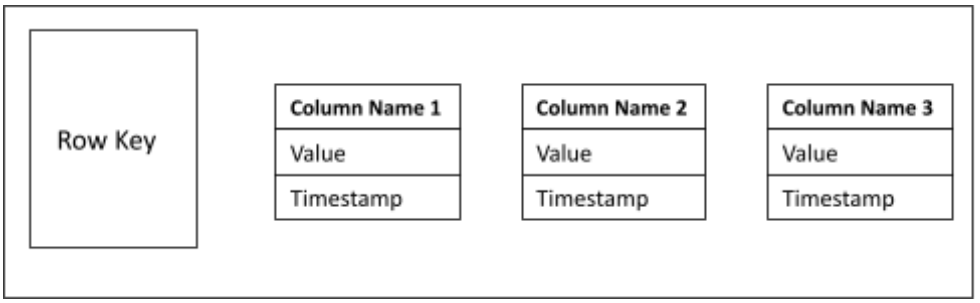
  Each column family contains rows, these rows further contain columns with different names, links, sizes i.e. they don't need to be of the same size, they can vary.
  These columns only exist within their own row and contain a value with the timestamp it was entered on.

Let’s view a one column family. Let it be of an employee of a company:



Here, we see the identifier be the employees name, and each employee has different kinds of data present along with their names.

Now, Let’s view one row for an example.



Row key here is an unique identifier of the row.

➢ Some Columnar NoSQL Database - *Casandra, CosmoDB, HBase*

- **Benefits of Column based NoSQL Database:**
  There are few advantages of using columnar database:
  - Efficient in terms of storage as column stores are good at compression. Hence, we reduce disk resources when handling huge data chunks in a single column.
  - Aggregation queries operate fast.

- Highly scalable, can be scaled out infinitely.
- Load and query time is quick.
- Self-indexing
- Extreme flexibility as columns can be added without disrupting the database.
- Great for analytics and reporting.

- **Limitations:**
  - It’s hard and time consuming to design an index schema for a columnar database.
  - Incremental data loading is not suitable enough.
  - Security issues and vulnerabilities in web apps.
  - Online Transaction Processing (OLTP) applications don't consider column based databases due the fashion in which data is stored here.
  - Can’t write complex queries, it’s hard to connect different columns and extract the desired information.

On observing it could be said that column based databases are nearly identical to SQL methods. The keyspaces here act as schema, thus some kind of schema management is being done. Also, the metadata can be similar to that of a typical relational DBMS.

Such databases are quite impactful but just like everything isn’t perfect they also do offer some limitations as well.
Like, in comparison to relational databases which are written sequentially to the disk on the other hand columnar NoSQL databases lack consistency due to multiple writes to the memory.
Irrespective of all that, they are still one of the most used data models.

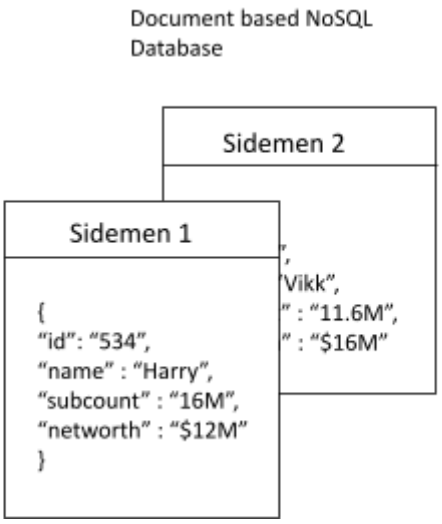
Document Based NoSQL:

These databases are very similar to key-value databases. Here information is stored in a document along with a key pair. It uses the internal structure of the document for identification and storage.
The data is saved as an instance in the database in comparison to how we do it in relational databases i.e. in tabular form. This method of storing data makes it easier for users to map the data in the database.

Below is a visualised difference between a Traditional Relational Database and a document-store.

id	name	subcount	networth
534	Harry	16M	$12M
645	Vikk	11.6M	$16M
557	KSI	23.4M	$20M
675	Simon	9.7M	$10M

Relational Database



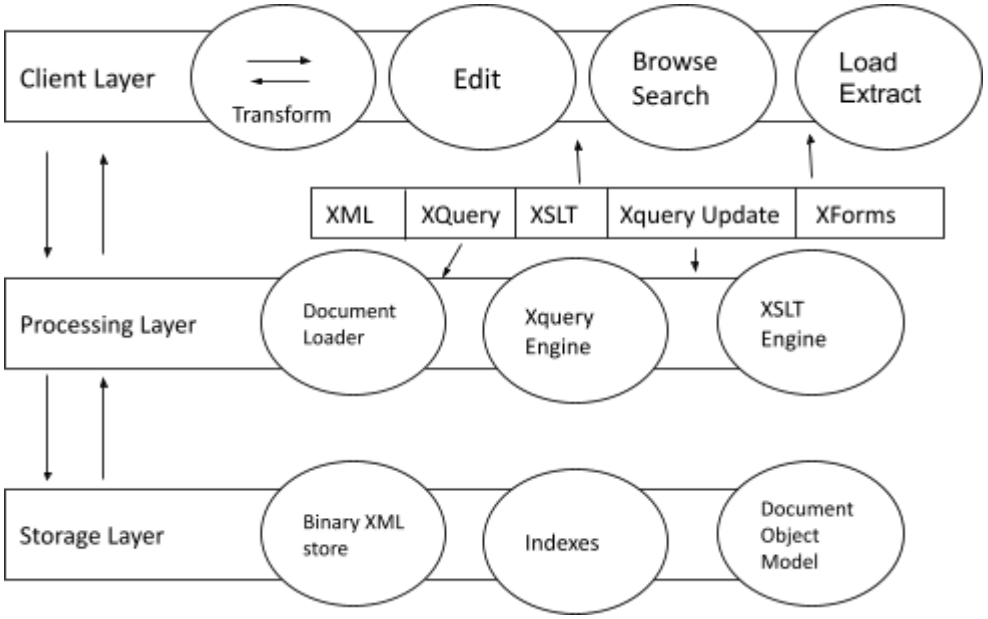
- **How does this work ?**

  These types of databases can store any sort of information in a document. Although ideally we use some form of schema with a certain file format(for example : JSON) to store data, we can intermix and store any kind of data we want to, irrespective of whatever kind of data has been stored our database will be able to parse it.
  It is much simpler to work with data and queries are pretty much simple to comprehend as compared to what we faced in SQL.
  Same operations performed in a Sql database like delete, add and further querying can be done on it.

As earlier it was stated that data is stored with a key, this key is unique for every information and can be used to retrieve or access the data in the document instead of being taken out on a column by column basis.

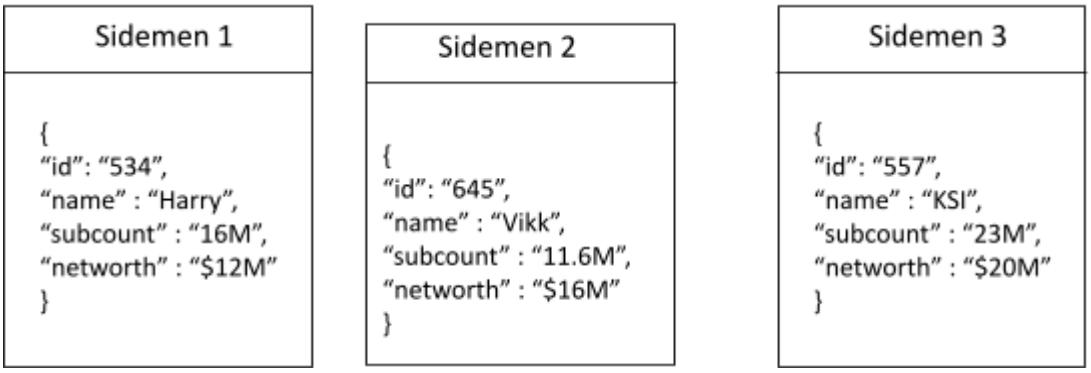
There are different types of Document databases like XML or JSON.

XML Databases:



Here documents store data in XML format. It is best designed for business purposes or where data is being processed in XML format.
It was also the first Document Database.
We use XML databases in eXist, MarkLogic.

JSON Databases:



Now if we consider the first example we gave above, these documents are stored in JSON format files.

Above is the collection of files. Although generally, it's not important for documents in the collection to be of the same type.

Few JSON Databases are MongoDB, OrientDB, CouchDB.

We can do certain operations in document- stores like,

- ➔ Can perform searches using different fields, regular expressions.
- ➔ Can perform queries by implementing javascript functions.
- ➔ We can index the database using any field we want to.

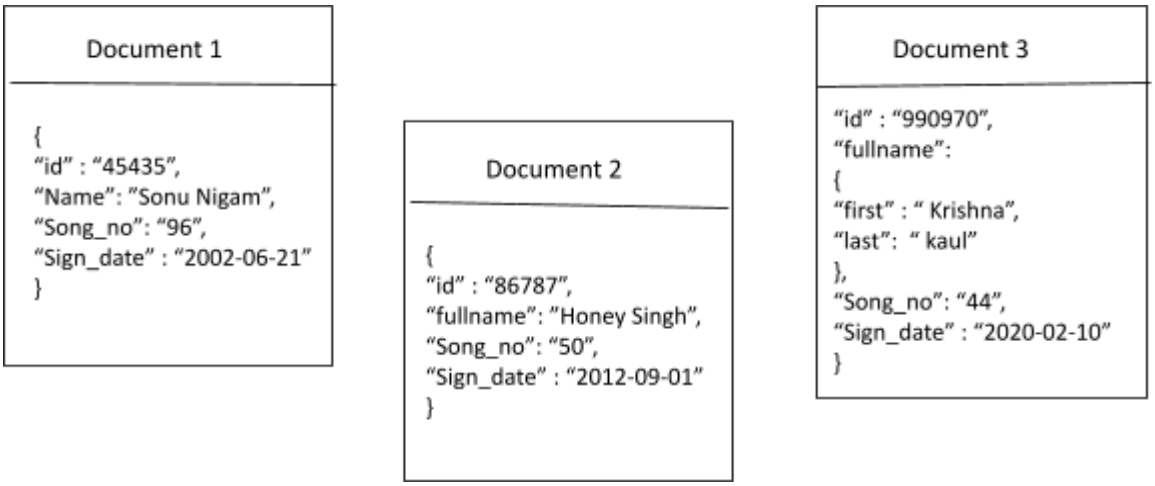
➢ Some Document based NoSQL Database - *MongoDB, CouchDB, OrientDB*

- **Benefits of Document Based NoSQL:**

  There are certain benefits of using document based database like,

  - High Scalability
  - Data Model is collection documents in JSON or XML format
  - Information is available in a single database, rather than having it spread across several linked databases.
  - More flexibility, no need for consistency among documents.
  - Capable of storing huge chunks of data with relatively no issues.
  - Addition of new data is easy.
  - Supports ACID transactions.
  - Flexible schema.

Below is an example of document based database, it is a database for an Artist management company.



If you observe that over the years the style of storing the information the document has changed, but it won't affect the previously stored data.

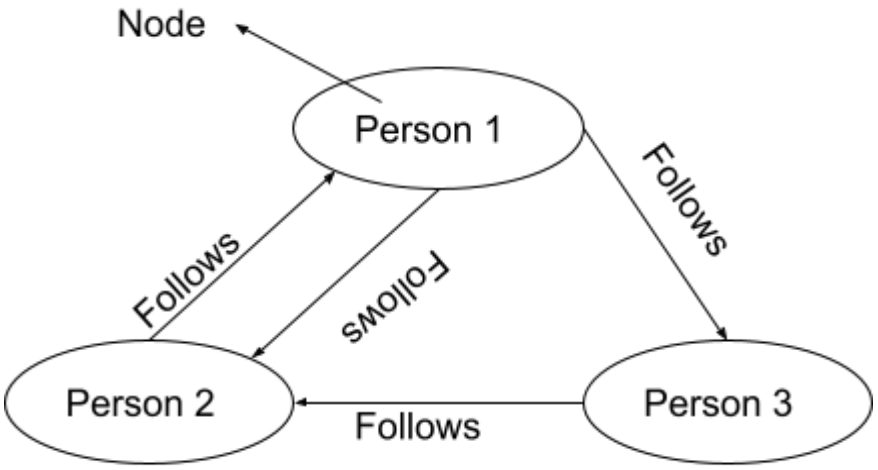
- **Limitations:**
    - Loss of data can occur due to wrong configuration or by using a single node.
    - Can't run complex queries.
    - Security issues.
    - Doesn't enforce data constraints
    - Interlinking documents can be very complex to operate.
    - References don't work easily in document based NoSQL.

Document based Databases are widely in use due to the flexibility they offer. With database applications becoming more and more complex, it allows us to scale up and add huge data easily, which makes the process of handling and working with huge data pretty simple for us.
It, like many other NoSQL databases, also helps with analytics as a business-venture might need to store data of various kinds.

- **Some real life applications of Document Based Database:**
    - Blogging sites like twitter
    - Analytical platforms
    - E-commerce platforms like Amazon
    - Content management systems

Graph Based NoSQL:

It's a collection of related objects. Each node in the graph represents an entity, where all of them are interconnected, the edge through which nodes are connected defines relationships among them.
Let's try to visualise it:



Here, Person 1 and Person 2 and Person 3 are entities and the edge that connects them is a relationship.
Above is a social network of their instagram handles.
Here, Relationships can hold more details like, when did person 1 followed person 3 and when did he/she followed back person Person 2.
Hence, relationships also can hold multiple details to define it.

- **How do they work ?**
  Here, nodes and edges i.e. entity and relationship hold on to certain properties like nodes depict properties similar to that of tables or JSON documents and relationship is defined with properties like it's history, it's type and how much strength it withholds.

  We can perform operations like addition or removal or nodes or relationships although the only thing that we need to take care of is when we remove a node, all relationships associated with it also get deleted as a relationship always intends to have a start and end node. We also know this as a core rule called 'No Broken Lines'.

➢ Graph based NoSQL Database - *Neo4j*

- **Benefits of Graph based Database:**
    - Highly scalable
    - Multiple query languages, like Neo4j uses Cypher query language.
    - Flexible schema to work with
    - Query speed usually depends on the number of relationships and is independent of the amount of data to be processed.

- **Limitations:**
    - No standard query language available, each database has it's own
    - Multiple reads but only single write transaction provision, leads to poor performance and concurrency.

- **Real Life applications where Graph based Database is used:**
    - ➢ It can work really well when working on social networks.
    - ➢ It can be used to support transportation systems as well.
    - ➢ It can be used to detect fraud in transactions.
    - ➢ It can be used to store criminal network data.

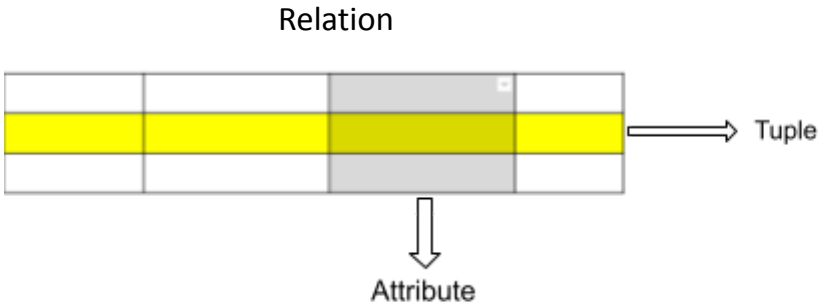
Types of Databases:

There are four type of databases based on the data models-

- Relational Database
- Object-Oriented Database (OODBMS)
- Hierarchical Database
- Network Database

Relational Database:

A RDBMS is a type of database that stores and provides access to data points that are related to one another. RDBMS stores data in **tables or relations**. It is based on a relational model i.e. each row is a unique record that is called **tuple**. Column of the table stores attributes. Oracle, MySQL etc are some examples of RDBMS software.



- **Relation:** In relational Model relations are saved in the format of tables. This table stores relations among entities. Rows of tables represent records and columns of table represent attributes.
- **Tuple:** A single row or record of the table is known as tuple.
- **Relation Instance:** A row or tuple of a table is called a relationship instance.
- **Relation schema:** Relational schema defines design and structure of the relation and also the variable declarations in tables. It consists of the relation name, set of attributes, column name.

Important properties of table

- The values have to be atomic (they can't be broken down any further).
- The values in each column have to be of the same data types.
- Each row has to be unique in a table to avoid redundancy, otherwise it will go against the concepts of RDBMS.
- The sequence of column and row is insignificant.
- Each attribute must be unique.

Relational Database Management Systems maintains data integrity by applying some constraints that are Entity Constraints, Referential Constraints, Domain Constraints.

Advantages:

- Simple to handle, every relation can be manipulated without affecting another one.
- Highly Secure.
- It supports client-side architecture storing multiple users together.
- Data Handling is quite simple
- Replication of databases helps to recover the system incase of any failure.

Disadvantages:

- Scalability is not handy.
- Requires expensive hardware and software support.
- As the data gets huge, complexity increases.
- Data loss can happen.

Some examples of Relational databases are Microsoft SQL Server, Oracle Database, MySQL and IBM DB2.

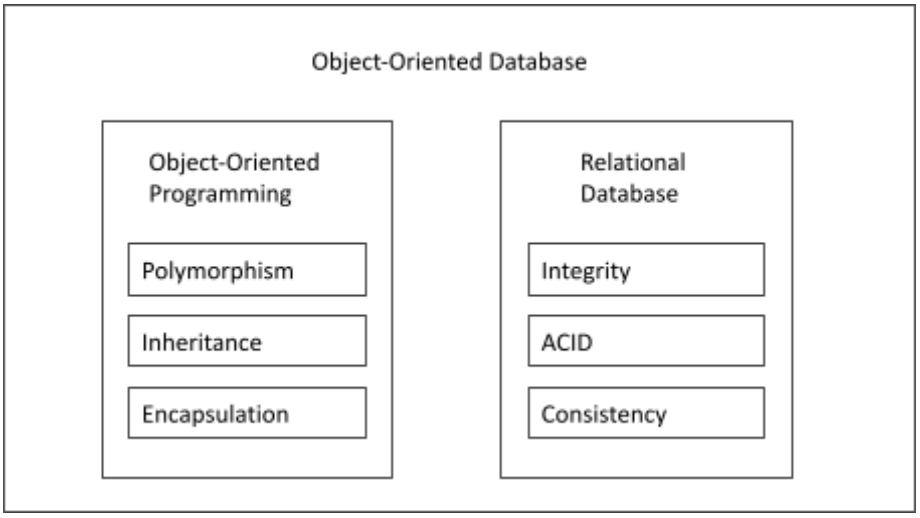
Object-Oriented Database:

It is a type of database in which data is stored in form of objects, which are nothing but the instances of classes (like we have objects for classes in C++)
This makes up what we call an Object-Oriented Database.
This database adds database functionality to the object programming languages. (combination of relational database and object-oriented language)

It is formed of various components:

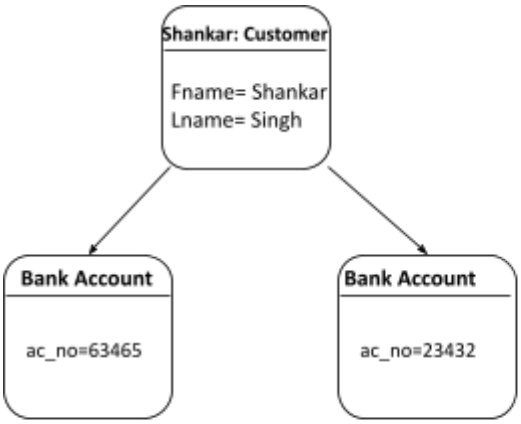
- **Objects:** They are either built-in(default) or user-defined.
- **Classes:** Schema or blueprint for objects.
- **Methods:** determine the behavior of a class.
- **Pointers:** Retrieve elements from the object database and define relations between them.

Below is the image that depicts an Object-oriented database formed by combination of Object programming language and Relational Database.

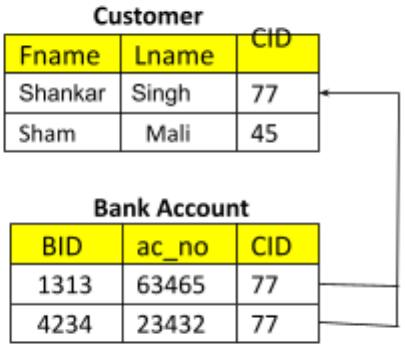


The main motive of objects in OODBMS is the possibility of user-constructed types. Sometimes the database can be very complex, having multiple relations. So, maintaining a relationship between them can be tedious at times.

- In Object-oriented databases data is treated as an object.
- All bits of information come in one instantly available object package instead of multiple tables.



Object-Oriented



Relational

Above is the difference between the representation of databases presenting the same data.

An object-oriented management system provides the functionality of an Object oriented language when dealing with complex data.
This feature brings attributes and behaviors of data together into one entity.

Advantages:

- Data storage and retrieval is easy and quick.
- Can handle complex data relations and more variety of data types than standard relational databases.
- Relatively friendly to model the advance real world problems
- Works with functionality of OOPs and Object Oriented languages.
- Object Id is auto assigned.

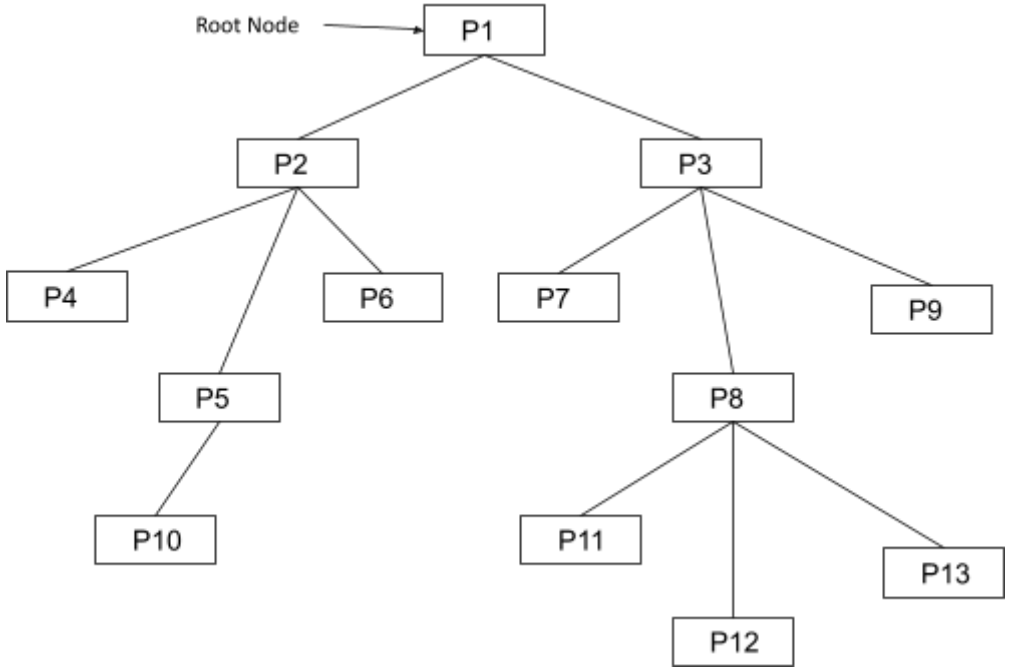
Disadvantages:

- High complexity causes performance issues like read, write, update and delete operations are slowed down.
- Not much of a community support as isn't widely adopted as relational databases.
- Does not support views like relational databases do.

Some examples of Object-oriented databases are ObjectDB, Versant, GemStone.

Hierarchical database:

In this kind of data model the data is stored in the form of records and organized into a tree-like structure, in which a node can have as many sub-nodes as it wants, connected through edges.



In the above diagram P1 is the root directory, and its children are P2 and P3. P2 is a parent to P4, P5, and P6, which in turn has children of its own.

The topmost node is known as root node and the nodes which don't have any children further are called leaf nodes.

All the records/files in this database are represented in parent-child manner, where each parent node has multiple child nodes (one or more), although every child node has only one parent node.

Few features of Hierarchical Databases:

- They have useful real world applications as some real-world occurrences of events are hierarchical in nature like biological structures, political, or social structures.
- Since the disk storage system is also inherently a hierarchical structure, these models can also be used as physical models.
- Deletion of parent node, will lead to deletion of child node.
- It supports one to many relationships but couldn't bear many to many relationships.

- As parent and child nodes are placed closer in the memory, the load of input and output on the hard disk are minimized.

Advantages:

- Promotes data sharing.
- Data Security is ensured.
- Data can be accessed quickly due to the links present between the relations.
- Performance is high
- Referential integrity is always maintained.
- It’s design is simple to comprehend.

Disadvantages:

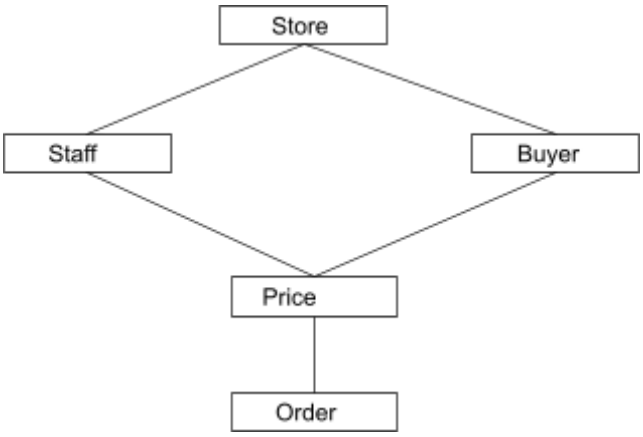
- No data manipulation or data definition language**.**
- Can’t handle Complex Relations
- Redundancy which results in inaccurate information.
- Offers Poor flexibility i.e. a child entry needs to have parent entry, and also tasks like deletion, updation are quite hideous.

Some examples of Hierarchical databases are IBM Information Management System (IMS) and the RDM Mobile.

Network Database:

A network database is a modified form of Hierarchical Database in which a member entity (similar to child entity in a hierarchical database) can have multiple relations with different owner entities (similar to parent entity in hierarchical database).

Let's try to visualise this with simple example:



Here, we can view that the entity Store has relationship with two entities i.e. Staff and Buyer whereas Price entity has relationship with two entities i.e Staff and Buyer. Now In this specific example Store is an owner entity of entities staff and Buyer and similarly, entity staff and entity Buyer are owner entities for Price entity.

Therefore, In the network database model one owner can be related to multiple member nodes and every member node can have multiple owner nodes.

Advantages:

- It's design is simple to comprehend.
- Can handle multiple types of relationships i.e. one to one, one to many, many to many which makes it more suitable to tackle the real world problems.
- Data can be accessed quickly due to the links present between the relations.
- Data Integrity is maintained as no member entity can exist without an owner entity.

Disadvantages:

- Complex Schema due to all multiple relationships.
- Will have deletion, update anomalies due to so many relationships.
- Maintaining a Network Database is a tedious job.
- Not much Community Support.

Some examples of Network databases are Integrated Data Store (IDS), IDMS (Integrated Database Management System), Raima Database Manager, TurboIMAGE, Univac DMS-1100.

Difference between Hierarchical, Network and Relational Data Model:

Relational Data Model	**Hierarchical Data Model**	**Network Data Model**	**Object-Oriented Data Model**
Records are stored in tables.	Records are stored in a tree structure.	Records are stored in the form of graphs.	Records are stored in the form of objects.
It provides data independence.	It lacks Data Independence.	Provides Partial Data Independence.	It supports Data Encapsulation.
Can implement multiple relationships like one to one, one to many, many to many.	Can implement few relationships like one to one, one to many. It's not capable of working with many to many relationships.	Can implement multiple relationships like one to one, one to many, many to many.	Can implement multiple relationships like one to one, one to many, many to one, many to many

Concurrency Control

Concurrency Control is the management procedure that controls concurrent execution of the multiple operations done by different users at the same time on the same database.

As stated above it controls “concurrent execution”. We need to keep a check on it because of the conflicts that might arise due to it.
Like,

- Write-Write Conflict (Lost Update Problem): When an update/write operation is lost as it is overwritten by the update/write operation done by another transaction.
  Eg: T1 transaction performs write on the database, only then T2 transaction starts with it’s write operation on that database only.
  The value updated by T1 is lost. Hence called the Lost Update Problem.
- Write-Read Conflict : It occurs when a transaction reads the data which is written by the other transaction before committing.

Therefore, we need concurrency control to manage these concurrent executions of the operations and help maintain **consistency in the database**.

Other Advantages of Concurrency Control:

- ➢ Increases Throughput
- ➢ Reduces Wait time for Transactions
- ➢ Optimal Resource Utilization

To help us do all that, we have multiple Concurrency Control Protocols:

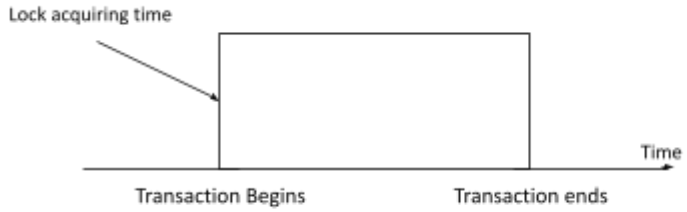
- Lock Based Concurrency Control Protocol
- Timestamp Concurrency Control Protocol

Note: Although there are more protocols (Multi version concurrency control and Validation concurrency control) we will focus on these only for now.

Lock Based Concurrency Control:-

In Lock Based Concurrency Control, any transaction in execution has Lock embedded on it due to which another transaction cannot read or write data until the previous transaction is completed and terminated.

Then after the completion of the first transaction, the lock is acquired by the next transaction and thus now it can read or write data.
By using this mechanism, we isolate a transaction in execution.



Disadvantage:

- The issue with this type of concurrency control is it might lead to Deadlock at some point.

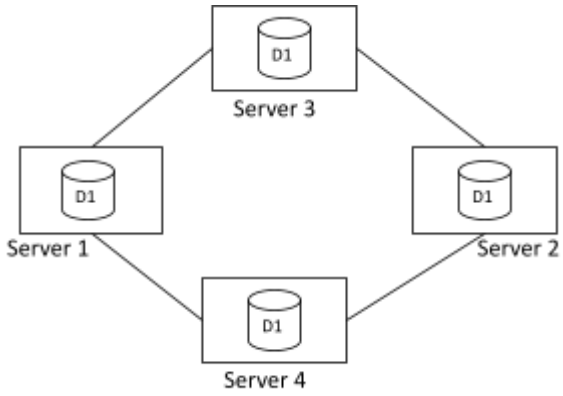
Timestamp Based Concurrency Control:-
The Timestamp Based Concurrency Control is used to order the transactions based on their Timestamps, it ensures that every conflicting operation is executed in that order only and it is ascending order based on transaction's creation time.
Although the priority is high , for the transaction that arrived first for being executed and for determining that priority order, system time or logical counter is used.

For example, Suppose concurrent transactions T1, T2, T3 that arrive at 10:00 AM, 12:00 PM and 9:00 AM respectively.
Now, as T3 arrives the earliest it will be given the priority and will be executed first, then T1 and then last but not the least T2.

The main advantage of using this over Lock Based concurrency control is it is free from deadlocks.
Disadvantage:

- It can cause starvation at a point where a transaction is being restarted and aborted again and again.

Clustering:
It is the process of combining more than one server or instances holding the same database. Or we can say it is a set of replicated servers.



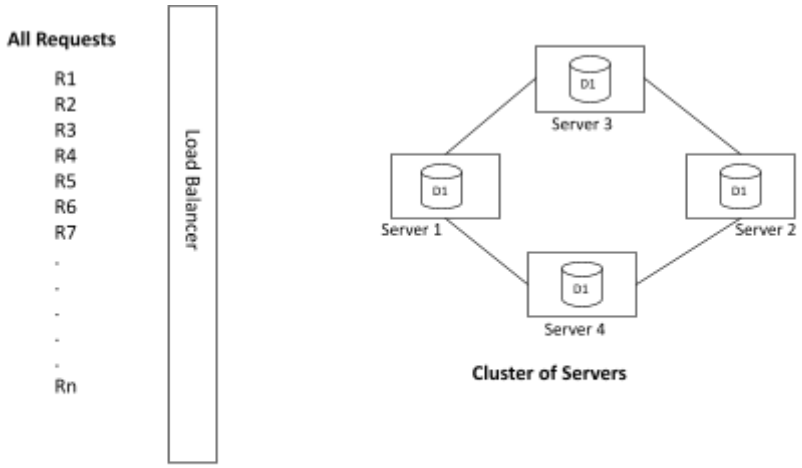
Here, we have cluster of 4 servers, which have been loaded with the same database i.e. D1.

Why do we need Clustering ?
Clustering databases helps servers with a lot of features like,

- **Data Redundancy:** Clustering of databases helps with data redundancy, as we store the same data at multiple servers. Don't confuse this data redundancy as repetition of the same data that might lead to some anomalies or ambiguities. The redundancy that clustering offers is required and is quite certain due to the synchronization.
  In case any of the servers had to face a failure due to any possible reason, the data is available at other servers to access.
  Like in the image above, if the server 3 holding database D1 had any kind of malfunction or issues. Other servers 1,2 and 4 won't be affected and data can be accessed from them.

- **Load Balancing:** Clustering of databases also helps the servers with the load balancing i.e. suppose if there's only one server and there are a lot of requests coming in to access the database then it may become difficult for a server to handle all the requests and the response time for the request will also become huge.

Although, if we introduce more servers holding the same database now the load of the request can be divided among them.
Hence, the response time for the requests decreases too.
So, load balancing refers to the process where the load balancer smartly assign the requests to the servers with the same database.



In the above image, load balancer randomly assigns the ‘N’ requests among the cluster of servers keeping in mind the load they already have, so that the load of requests is equally divided among them and the response time for the request can be improved.

- **Availability:** An Availability of of a database is defined as the time when we can access the database. Now if it wasn’t for clustering, or replication of a database to exist we would have faced this problem very often whenever a database would have been going through a transaction.
  Now as we have clusters of servers available, even if one of the databases is going through a transaction, now the other servers can be used to access the database with the help of load balancer.
  Also, even if a server fails, the database will be available.
  Hence, due to clustering the databases have high availability.

Types of Database Clusters:

On the basis of the requirement of the system there are multiple type of cluster architecture present in the market:

- High-Performance Cluster
- Load Balancing Clusters
- Failover/High Availability Clusters

Hence, Clustering makes the database more available, and improves the performance of the database.

Therefore, improving the scalability of the database.

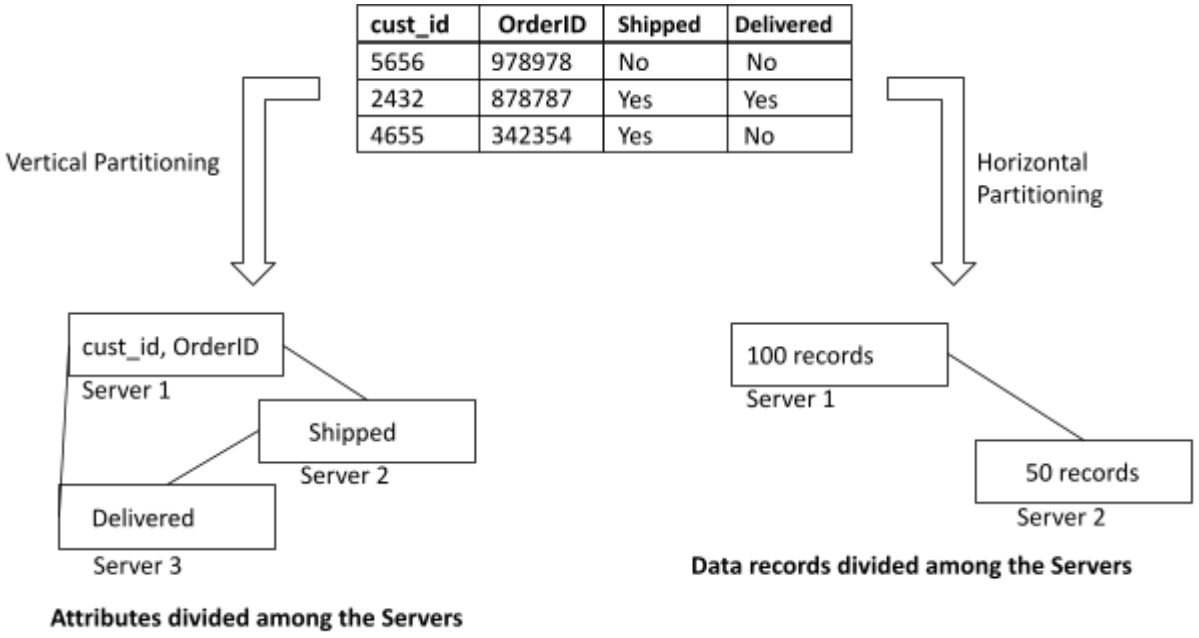
Partitioning:
It is a technique used to divide stored database objects into separate servers. Due to this, there is an increase in performance, controllability of the data. We can manage huge chunks of data optimally.

When we horizontally scale our machines/servers, we know that it gives us a challenging time dealing with relational databases as it's quite tough to maintain the relations.
But if we apply partitioning to the database that is already scaled out i.e. equipped with multiple servers, we can partition our database among those servers and handle the big data easily.

Now to partition the data they are two techniques:

- **Vertical Partitioning**: In this, we partition the given data vertically i.e column wise. So, if we are provided with table students with attributes student id, name, courseid, address, we can store this data by distributing it among servers where we store studentid, name is one server, courseid in another and address in the third one.
- **Horizontal Partitioning**: In this, we partition the given data horizontally i.e. row wise. So, have chunks of a certain size of data stored at different servers.
  Like, to refer to the same example as given in the above point. Now we will be storing different numbers of entries or rows in different servers as we store the first 100 entries in server 1 and other 100 entries (i.e. from 101 to 200) in another server (attributes remain the same.).

Let’s visualise it with an example:



Above is an image representing how data partitions happen vertically and horizontally.

Some Advantage of Partitioning:

- it plays a key role in building systems with extremely high availability requirements.
- Improves performance and manageability.
- It helps in reducing the cost of storing a large amount of data.
- We can upload rarely used data into cheap data storage devices.

Sharding:

It is a method of partitioning and storing a single logical set of data in multiple databases which is stored on multiple machines/servers.
It is an extension to Horizontal Partitioning.
The smaller chunks of the data that are created after Shrading are called Shards.

cust_id	OrderID	Shipped	Delivered
5656	978978	No	No
2432	878787	Yes	Yes
4655	342354	Yes	No
1443	656677	Yes	No

Shrad 1

cust_id	OrderID	Shipped	Delivered
5656	978978	No	No
2432	878787	Yes	Yes

Shrad 2

cust_id	OrderID	Shipped	Delivered
4655	342354	Yes	No
1443	656677	Yes	No

On Shrading, the table is divided into “Shards”, each new table has the same schema but unique distinct tuples.

Why Shrading ?

Let’s take a very basic example: Consider a giant database of college/university. In this database, all the details of students, being a present student or an alumni, exists.

So, it might contain a huge number of entries, let's say if a college has been there for 100 years and has had 1000 thousands in each batch. The total number of students will be 1,00,000.

Now when we need to fetch some details of a student from this Database, each time around 1 lakh transactions have to be done to find that student, which seems awful.

Now, if we divide the data into ‘Shrads” batch wise, where each shrad contains nearly 1000 entries.

Now the transaction cost is reduced by the factor of 100. As now we have to fetch the information from only 1000 entries and that’s possible because of Shrading.

Advantages:

- High availability of the data.
- Increase in Storage capacity
- Read/Write operations speed increased.

Disadvantages:

- The cost of infrastructure required increases
- Complexity for administration to maintain the database increase

Hence, Sharding can be a good solution if one needs to scale their database horizontally. But with few benefits it does bring complexity in application. It might be necessary at some places but the money and time to maintain such a model could be overwhelming as compared to benefits at certain situations.

Important Functions and commands

1. MOD() in SQL :-
   The **MOD()** function returns the remainder of a number divided by another number.

   Definition: MOD(x,y) is x mod y.

   Example - To return remainder of 5/2 :

   ```
   SELECT MOD(5,2);
   ```
   Output: 2

2. MONTH() and YEAR() :-

   The **MONTH()** function returns the month part for a specified date (a number from 1 to 12).

   Example- SELECT MONTH('2021/11/10 12:00') AS Month; Output:

Month
11

   The **YEAR()** function returns the year part for a specified date.

   Example - SELECT YEAR('2021/11/10 12:00') AS Year;
   Output:

Year
2021

3. CONCAT() :-
   The **CONCAT()** function adds two or more strings together

   Syntax: CONCAT(*string1*, *string2*, ..., *string_n*)

   Example - CONCAT('Be', ' ','a',' ', 'Coding',' ','Ninja');
   Output: Be a Coding Ninja

   Example 2: CONCAT('Be','a','Coding','Ninja');
   Output: BeaCodingNinja

4. Cloning a table:-

   CREATE TABLE new_table LIKE original_table; /* This Creates empty new table without data*/
   INSERT INTO new_table SELECT * FROM original_table; /*This inserts data of original table into new table*/

5. LIMIT():-

   Syntax-
   SELECT column_name
   FROM table_name
   LIMIT offset, count;

   Column_name: It is the name of the column that you want to return.
   Table_name: It is the name of the table that contains your column name.
   Offset: It specifies the number of a row from which you want to return. The offset of the row starts from 0, not 1.
   Count: It specifies the maximum number of rows you want to return.

Example-

Consider the following : ninjas_tab

Id	Name	City
1	Raju Ninja	Delhi
2	Abhi Ninja	Mumbai
3	Shyam Ninja	Chennai
4	Golu Ninja	Kolkata
5	Ramu Ninja	Bangalore

Example1:

SELECT* FROM ninjas_tab LIMIT 1,4;

Output:

Id	Name	City
2	Abhi Ninja	Mumbai
3	Shyam Ninja	Chennai
4	Golu Ninja	Kolkata

Example 2:-

SELECT* FROM ninjas_tab LIMIT 2; /* Assumes offset to be zero*/
Output:

Id	Name	City
1	Raju Ninja	Delhi
2	Abhi Ninja	Mumbai

6. _ operator:-

_ Represents a single character.

Example-
Consider the following : ninjas_tab

Id	Name	City
1	Raju Ninja	Delhi
2	Abhi Ninja	Mumbai
3	Shyam Ninja	Chennai
4	Golu Ninja	Kolkata
5	Ramu Ninja	Bangalore

SELECT * FROM ninjas_tab WHERE Name LIKE 'R___'; /* Note R is followed by three _ */

Output:

Id	Name	City
1	Raju Ninja	Delhi
5	Ramu Ninja	Bangalore