



आईएफटीएम विश्वविद्यालय, मुरादाबाद, उत्तर प्रदेश
IFTM University, Moradabad, Uttar Pradesh
NAAC ACCREDITED

E-Content

IFTM University, Moradabad

Database Management System (DBMS)

UNIT-I

Data: Data is a collection of information. In other word we can say that the facts that can be recorded and which have implicit meaning known as 'data'.

Ex: Customer -

1. customer_name
2. customer_no.
3. customer_city.

Database: Collection of that interrelated data. These data can be stored in tables form.

Ex: Customer database consists the fields as c_name, c_no, and c_city

customer_name	customer_no	customer_city
Ram	45662	Moradabad
Shyam	78598	Delhi

DBMS DBMS stands for **Database Management System**. We can break it like this

DBMS = Database + Management System.

As we discuss that by data we mean useful information. In other ways Data as a general concept refers to the fact that have some accessible information or knowledge is represented or coded in some form suitable for better usage or processing.

Database is a collection of that data and Management System is a set of programs to store and retrieve those data. In view of this we can characterize DBMS like this:

Database management system (DBMS) as a collection of interconnected data and set of programs to update,store & access that data in a simple and effective way.

Need of DBMS

Database systems are fundamentally developed for large amount of data. When managing enormous amount of information, there are two things that require improvement:

The way we store the data and the way we access data

Storage of Data: According to the philosophy of database systems, the data must be stored in such a way that it acquires lot less space as the redundant data (duplicate data) has been removed earlier than storage.

Let's take an example to clear this: In a financial framework, assume a client is having two accounts, one is current account and another is pay account. Suppose bank stores current record information at one spot and pay account information at somewhere else, all things considered if the client data, for example, client name, address and so forth are put away at the two places then this is only a wastage of capacity (excess/duplication of information), to compose the information in a superior way the data ought to be put away at one spot and both the records ought to be connected to that data some way or another. Something very similar we accomplish in DBMS.

Fast Retrieval of data: Along with storing the data in an optimized and systematic manner, it is also important that we retrieve the data quickly when needed. Database systems ensure that the data is retrieved as quickly as possible.

Purpose of Database Systems

The main purpose of database systems is to **deal with the data**. Let's take an example of a college which has data of students, faculty, dept, courses, books etc. To manage this data we need to store this data somewhere where we can add new data, delete unused data, update outdated data, retrieve data, to perform these tasks on data we need a system called Database management system that allows us to store the data in such a way so that all these tasks can be performed on the data efficiently.

Applications areas of DBMS

Applications areas where Database Management Systems are used:

- **In the field of telecommunication:** There is an information database to monitor the data with respect to calls made, network utilization, client subtleties and so on. Without the information database frameworks it is difficult to keep up that enormous measure of information that continues refreshing each millisecond.
- **In the field of organization:** Where it is a manufacturing organization, stockroom or circulation focus, everyone needs an information base to keep the records of intricate

details. For instance conveyance focus should monitor the item units that provided into the middle just as the items that got conveyed out from the appropriation community on every day; this is the place DBMS comes into picture.

- **In the field of Banking System:** For putting client information, following everyday credit and charge exchanges, producing bank proclamations and so forth. This work has been finished with the assistance of Database frameworks.
- **In the field of Sales:** To store client data, creation data and receipt subtleties and sales data we need database system.
- **In the field of Airlines:** To travel through airlines, we reserve early, this booking data alongside flight plan is put away in information database.
- **In the field of Education:** Information database frameworks are oftentimes utilized in schools and universities to store and recover the information with respect to understudy subtleties, staff subtleties, course subtleties, test subtleties, finance information, student subtleties, expenses subtleties and so forth. There is a enormous related information that should be put away and recovered in an effective way.
- **In the field of E-Commerce:** You should know about the web based shopping sites, for example, Snapdeal, Amazon, Flipkart and many more. These websites store the item data, our addresses and inclinations, credit subtleties and give us the significant rundown of items dependent on our question. This includes a Database the board framework.

Database System Vs File System

In traditional approach, before to computer, all information and data were stored in papers. At that time when we need data, we used to look through the papers. After the invention of computer all data and information were stored in files.

File System

File processing system was an early endeavor to modernize the manual documenting system. A file system is a strategy for putting and arranging PC files and the information they contain to make it simple to discover and get to them. File system may utilize a capacity gadget, for example, a hard disk.

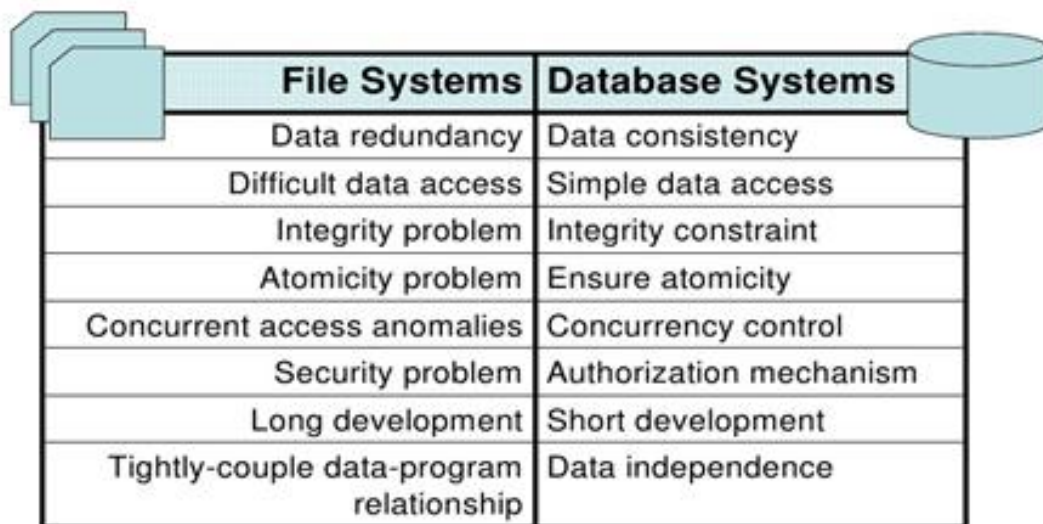
So a file system is a process that oversees how and where information on a storage device, normally a hard disk drive (HDD). It is an intelligent disk that deals with a disk's inward activities as it identifies with a PC and is conceptual to a human client.

A file system commonly oversees activities, for example, storage management, naming of file, directories and folders, metadata, retrieval rules and privileges.

There are various types of file system. Every type has distinctive structure and foundation, properties of speed, adaptability, security, size etc. Some file system has been intended to be utilized for explicit applications. For instance, the ISO-9660 file system is planned explicitly for optical discs.

Advantage of DBMS over file system

There are several advantages of Database management system over file system. Few of them are as follows:



File Systems	Database Systems
Data redundancy	Data consistency
Difficult data access	Simple data access
Integrity problem	Integrity constraint
Atomicity problem	Ensure atomicity
Concurrent access anomalies	Concurrency control
Security problem	Authorization mechanism
Long development	Short development
Tightly-couple data-program relationship	Data independence

Now we will describe in detail how DBMS is different from traditional file system and what its limitations were.

S.No	Difference factor	File System	DBMS
1	Definition	A file management system is an abstraction to store, retrieve, management and update a set of files. A File Management System keep track on the files and also manage them.	DBMS is a collection of interrelated data and a set of programs to access those data. Some well known DBMS are MS Access, SQL server, Oracle, SAP, etc.
2	Data Redundancy	In file system approach, each user defines and implements the needed files for a specific application to run. For example in sales department of an enterprise, One user will be maintaining the details of how many sales personnel are there in the sales department and their grades. Another user will be maintaining the sales person salary details.	Although the database approach does not remove redundancy completely, it controls the amount of redundancy in the database because in database approach, a single repository of data is maintained that is defined once and then accessed by many users. The fundamental characteristic of database approach is that the database system not only contains data's but it contains complete definition or description of the database structure and constraints.
3	Sharing of data	File system doesn't allow sharing of data or data sharing is very complex.	In DBMS data can be shared very easily due to centralized system
4	Data Consistency	When data is redundant, it is difficult to update. For e.g. if we want to change or update employee's address, then we have to make changes at all the places where data of that employee is stored. If by mistake, we forgot to change or update the address at one or more place then data inconsistency will occur i.e. the appearance of same data will differ from each other.	In DBMS, as there is no or less data redundancy, data remains consistent.

5	Difficult to search/access data	In conventional file system, if we want to search/retrieve/access some data item, it becomes very difficult because in file system for every operation we have to we have to make different programs.	In DBMS searching/retrieval/accessing of data item is very easy and user-friendly because searching and querying operations are already available in the system.
6	Data isolation	In file system there is no standard format of data or we can say data is scattered in various formats or files which also make data retrieval difficult.	In DBMS, due to centralized system the format of similar type of data remains same.
7	Data Integrity	The value of data in database must follow or satisfy some rules or consistency constraints. For e.g. A company have a policy that the age of an employee must be ≥ 18 . The value which is not satisfying this constraint must not be stored in the respective column. In file system, there is no procedure to check these constraints automatically.	DBMS maintains the data integrity by enforcing the constraints by adding appropriate code.
8	Security problems	In file system there is no or very less security. General security provided by file system are locks, guards etc.	DBMS have high level security like encryption, passwords, biometric security (fingerprint matching, face and voice detection etc) etc.

9	Atomicity	Atomicity means a transaction must be all-or-nothing i.e. the transaction must either fully happen, or not happen at all. It must not complete partially. E.g. if A want to transfer 5000rs to B's a/c. In this case A's a/c should be debited and B's a/c should be credited with the same amount. Let suppose A's a/c is debited with 5000rs and then transaction fails. Now the transaction is incomplete because B's a/c is not credited. These type of problems occur in file system because there is no procedure to stop such type of anomalies.	Transaction atomicity is a special feature of DBMS. In DBMS either a transaction completed fully or none of the action is performed. For this, DBMS maintains the transaction log in which intermediate values are stored.
10	Concurrent Access Anomalies	Any multi-user database application has to have some method for dealing with concurrent access to data -- when more than one user is accessing the same data at the same time. A problem occurs when user X reads a row for editing, user Y reads the same row for editing, user Y saves changes, then user X saves changes. The changes made by user Y are lost unless something prevents user X from blindly overwriting the row. File system does not provide any procedure to stop such type of anomalies.	DBMS along with an appropriate application provides safety towards concurrent access. For this locks are available in DBMS. If 2 or more transactions want to change/update or write a data item, an exclusive lock is issued to one of these transactions. Until and unless the transaction release that lock no other transaction can acquire the lock and hence cannot update/write the data item.

DBMS Architecture

Database management systems architecture will assist us with understanding the segments of database and the connection among them.

The architecture of DBMS relies upon the PC framework on which it runs. For instance, in a client-server DBMS architecture, the database system frameworks at server machine can run a few requests made by client machine.

Types of DBMS Architecture

Types of DBMS architecture are as follows:

1. Single tier architecture
2. Two tier architecture
3. Three tier architecture

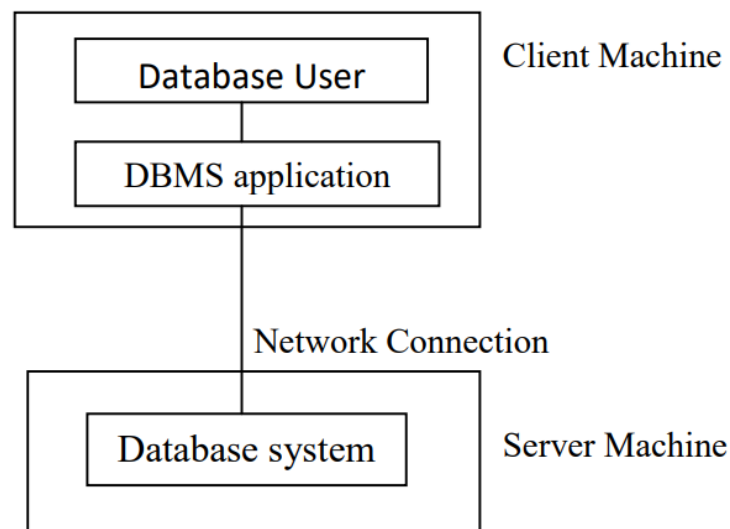
1. Single tier architecture

In this kind of architecture, the database is rapidly accessible on the user-machine, any request made by user doesn't need a connection to put out the activity on the database.

For instance, let's state you need to get the records of worker from the database and the database is accessible on your PC framework, so the request to get representative subtleties will be finished by your PC and the records will be brought from the database by your PC also. This type of framework is known as local database system.

2. Two tier architecture

In two-tier architecture, the Database system is there at the server machine and the DBMS application is there at the client machine, these two machines are linked with each other by a reliable network as shown in the two-tier diagram. When user machine makes a request to get to the database present at server utilizing an inquiry language like sql, the server play out the request on the data set and returns the back to the user.

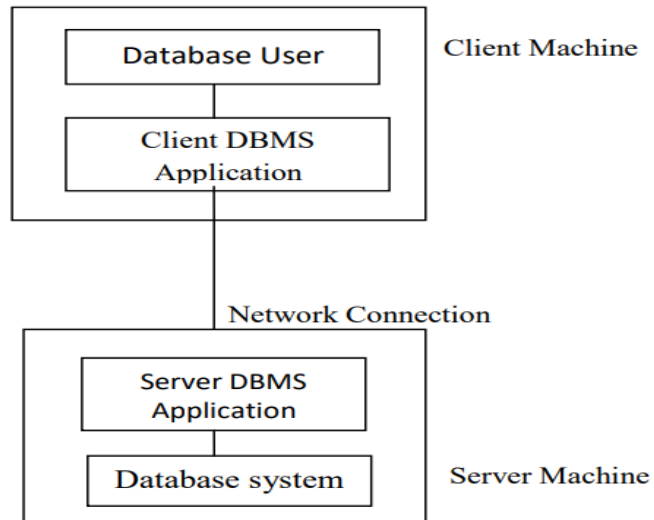


Two-Tier Architecture

For the connection between server and client application connection interface such as JDBC, ODBC are used.

3. Three tier architecture

In three-level architecture, another layer is available between the user machine and server machine.



Three-Tier Architecture

In this design, the user application doesn't discuss straight forwardly with the database system present at the server machine, rather the user application connect with server application and the server application inside connect with the database system present at the server machine.

DBMS – Three Level Architecture

This architecture consists of three levels as follows:

- 1.External level
- 2.Conceptual level
3. Internal level

1. External level

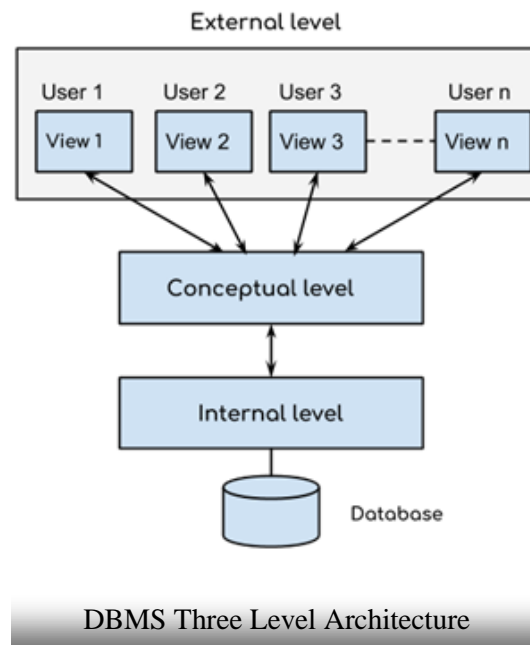
It is likewise called see level. The explanation this level is classified "see" is on the grounds that few clients can see their ideal information from this level which is inside brought from database with the assistance of conceptual and internal level connectivity.

External level is also called "high level" of the Three Level DBMS Architecture.

2. Conceptual level

It is also called logical level. The entire plan of the database, for example, relationship among data, types of data, description of data and many more things are described in this level.

Database requirements/constraints and security are additionally actualized in this level of architecture.



3. Internal level

This level is also known as physical level. This level defines how the data is stored actually in the storage devices .allocating space to the data is also key responsibility of this level. This is the lowest level of the architecture.

Database system concepts

DATA Model

Logical structure of Database is Data model. Data models depict how data is related to one another and how they are handled and put inside the system.

Types of Data Models

There are different types of data models in DBMS.

Object based logical Models-Describe data at the conceptual and view levels.

1. E-R Model
2. Object oriented Model

Record based logical Models

1. Relational Model
2. Hierarchical Model
3. Network Model

Physical Data Models

Data Abstraction - Abstraction is one of the main principle of database system. Hiding unimportant details from client and giving dynamic perspective on data to clients, helps in simple and effective client-database association. This cycle of hiding unessential details from client is called data abstraction.

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

Schemas: The description of a database is known as the database schema, which is specified when we design the database and is not expected to change generally. Most data models have sure conventions for displaying schemas as diagrams. A displayed schema is called a schema diagram. Some examples of schema are as follows:

STUDENT

Name	Student_number	Class	Major
------	----------------	-------	-------

COURSE

Course_name	Course_number	Credit_hours	Department
-------------	---------------	--------------	------------

PREREQUISITE

Course_number	Prerequisite_number
---------------	---------------------

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
--------------------	---------------	----------	------	------------

fig-schema

DBMS Instance

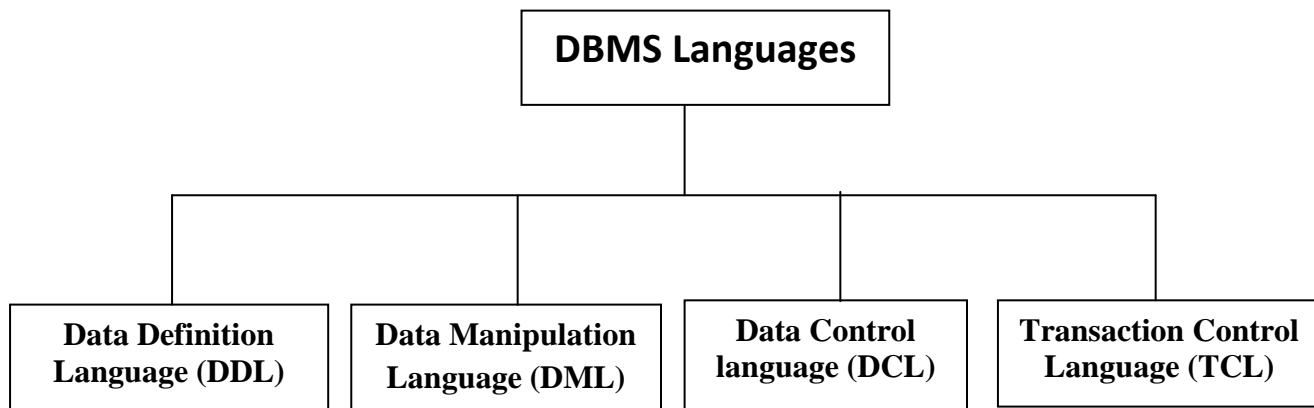
The data stored in database at a particular moment of time is known as instance of database. The real data in an data base may change every now and again.

For example, lets say we have a table employee in the database, today the table has 100 employee, so today the instance of the database has 100 records. Lets say we are going to add another 100 employees in this table by tomorrow so the instance of database tomorrow will be 200 records in table. So we can say that at a particular moment the data stored in database is called the instance, that changes over time when we add /delete data from the database.

DBMS languages

In DBMS we need to update/access/read/store data. We can perform these operations with the help of DBMS languages. Database languages are utilized to read, update and store information in a data base. There are several different DBMS languages that can be used for this motive; one well-known language of them is SQL (Structured Query Language).

Types of DBMS languages:



Data Definition Language (DDL)

We use DDL (Data Definition Language) to specifying the database schema. It can perform very functions like table creation, defining schema, defining indexes, applying constraints and many more in database. Let's see the operations that we can perform on database using DDL:

- CREATE- is used to create the database instance.
- ALTER-is used to alter the structure of database .
- DROP- is used to drop database instances.
- TRUNCATE- is used to delete tables in a database instance.
- RENAME- is used to rename database instances.
- DROP- is used to drop objects from database such as tables.
- Comment- is used to Comment.

```
CREATE TABLE table_name (  
    column1 datatype,  
    column2 datatype,  
    column3 datatype, ....  
);
```

```
CREATE TABLE Student (  
    Roll_No int,  
    Name varchar(255),  
    F_Name varchar(255),  
    Address varchar(255),  
    City varchar(255)  
);
```

For updating and defining design of database we use these commands so that they are come under Data Definition Language.

Data Manipulation Language (DML)

For accessing data and manipulating data in a database we use DML (Data Manipulation Language).Under DML following operations are performed:

- SELECT- is used to read records from table(s).
- INSERT- is used to insert record(s) into the table(s).
- UPDATE-is used to update the data in table(s).
- DELETE – is used to delete all the records from the table.

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

```
INSERT INTO student (stu_name, stu_rollno, stu_course ...)
VALUES (ram, 19005221, BCA, ...);
```

```
SELECT column1, column2, ...
FROM table_name;
```

```
SELECT stu_name, stu_rollno FROM student;
```

Data Control language (DCL)

For granting and revoking user access on a database we use DCL (Data Control Language)

- GRANT – is used to grant access to user.
- REVOKE – is used to revoke access from user.

Transaction Control Language (TCL)

Using TCL (Transaction Control Language) we can rollback or performed the changes in the database that we made using DML commands.

- COMMIT- is used to persist the changes made by DML commands in database.
- ROLLBACK- is used to rollback the changes made to the database.

Entity Relationship Diagram – ER Diagram in DBMS

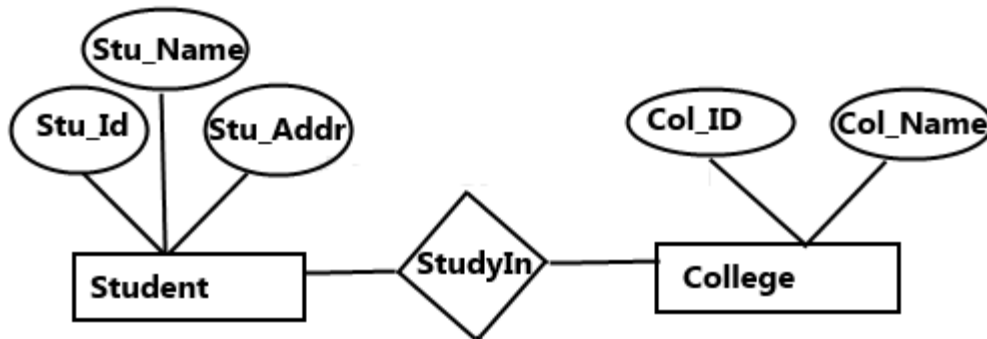
An **Entity–relationship model (ER model)** defines the structure of a database that is schema with the help of a diagram, which is called **Entity Relationship Diagram (ER Diagram)**. An ER model is a plan or outline of a database that can later be actualized as a database.

There are two main components of E-R model that are: entity set and relationship set.

ER Diagram example:

Here we are showing a simple ER-Diagram. In the diagram we have two entities named Student and College and both entity are linked with each other by a relationship. Attributes of

student entity are Stu_Id, Stu_Name & Stu_Addr and attributes of college entity are Col_Id, Col_Name.



Geometric shapes and their meaning in an E-R Diagram are as follows.

Rectangle: By rectangle shape we represent an entity sets.

Ellipses: By ellipses or oval shape we represents an Attributes.

Diamonds: By diamond shape we represent a relationship Set

Lines: By lines we link entity sets to relationship set and attributes to entity set.

Double Ellipses: By double ellipses we represent multivalued Attributes.

Dashed Ellipses: By double ellipses we represent derived Attributes.

Double Rectangles: By double rectangles we represent weak Entity Sets.

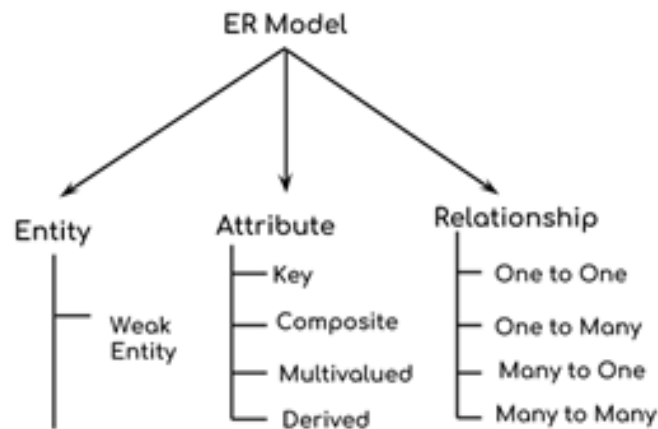
Double Lines: By double lines we represent total participation of an entity in a relationship set.

ER Diagram components

1. Entity

An entity is a real life object or concept. In an ER diagram we represent entity as rectangle. For example: In the above ER diagram two entities are there one is Student and another is College which is represented by a rectangle.





Components of ER Diagram

Weak Entity:

Weak entity is an entity can't be uniquely identified with the help of its own attributes and depends on the relationship with other entity for this purpose. We represent weak entity by a double rectangle box. For example – if we want to uniquely identify a bank by only bank account. This cannot be possible without having the bank in which the account exists, so in this case bank account is a weak-entity.



2. Attribute

An attribute of an entity describes the property of that's entity. We represent an attribute by Oval in an ER diagram. Four types of attributes are there in ER-Model as

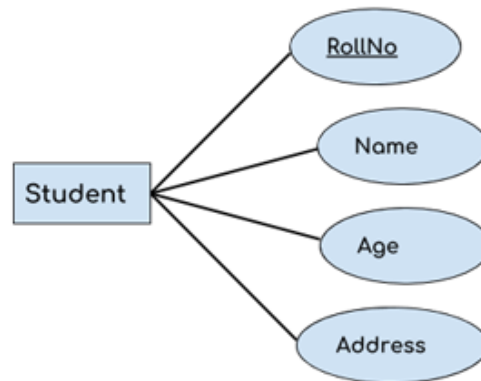
1. Key attribute
2. Composite attribute

3. Multivalued attribute

4. Derived attribute

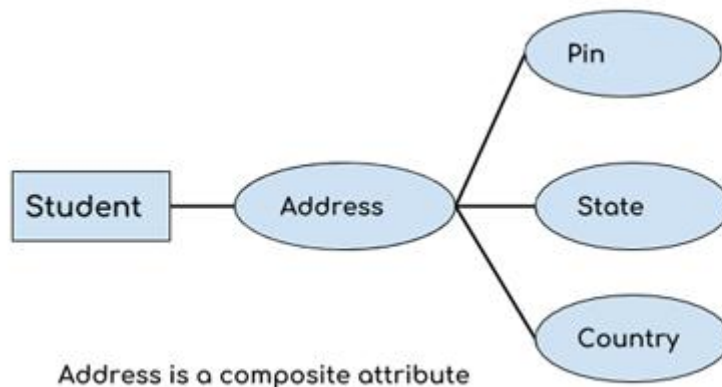
1. Key attribute:

A key attribute of an entity has the whole sole ability to uniquely identify that entity from an entity set. Lets take an example, we can uniquely identify a student by its attribute roll no from a set of students. We represent Key attribute by an oval same as other attributes however we underlined the text of key attribute.



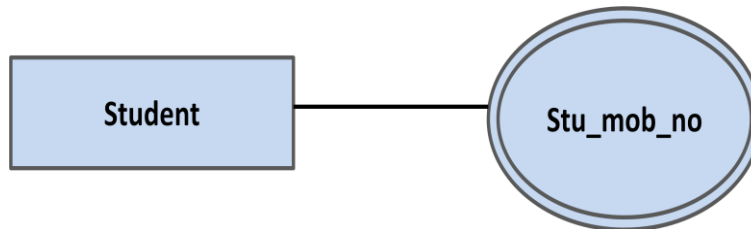
2. Composite attribute:

Composite attribute is a combination of other attributes. For example as we can see in below diagram that in student entity address is a composite attribute because address is collection of other attributes such as pin code, state, country in student entity.



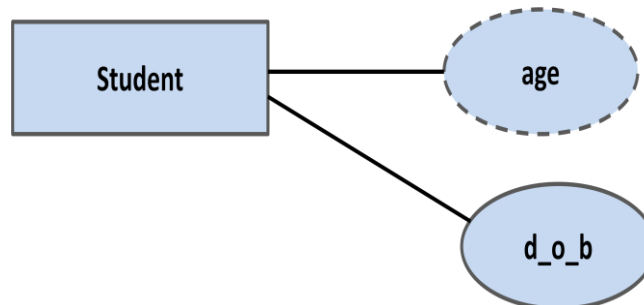
3. Multivalued attribute:

Multivalued attribute is the attribute that can keep multiple. We represented a multivalued attribute with **double ovals** in an ER Diagram. For example as we can see in the diagram below – A student can have more than one phone numbers so the phone number attribute is multivalued.

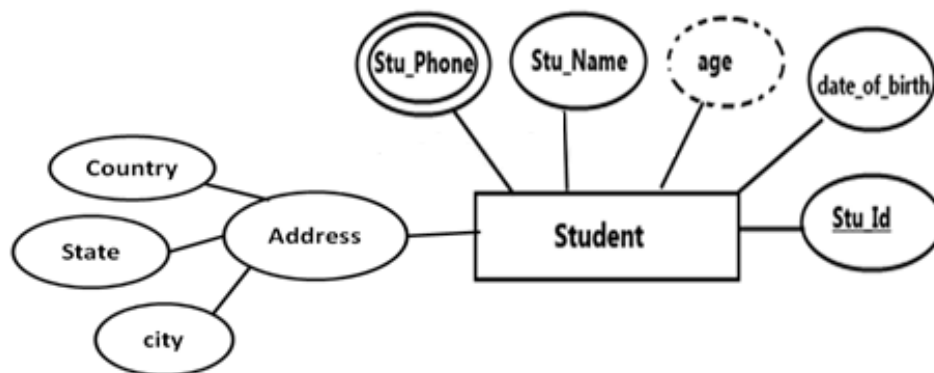


4. Derived attribute:

A derived attribute is the attribute whose value is not stable over time i.e. dynamic and which can be derived from another attribute. We represent it by **dashed oval** in an ER Diagram. For example see in below diagram – student age is a derived attribute because it changes over time and can be derived from another attribute i.e (Date of birth) of that student.



E-R diagram with key, composite, multivalued and derived attributes:



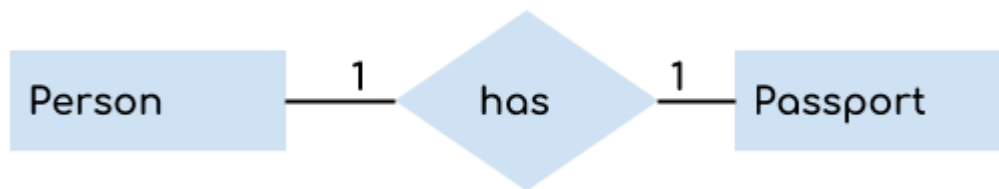
3. Relationship

A relationship set describes the relationship between entities that they hold. We represent it by diamond shape in ER diagram; it shows the relationship among entities. There are four types of relationships in ER-Model:

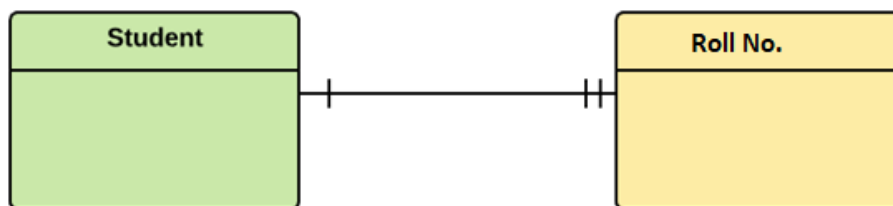
1. One to One
2. One to Many
3. Many to One
4. Many to Many

1. One to One Relationship

When a single instance of an entity is connected to a single instance of another entity then this type of relationship is called one to one relationship. Let's take an example, a person can have only one passport and a passport is issued to one person.



Example: One student can have one roll no. And one roll no can be assigned to one student.



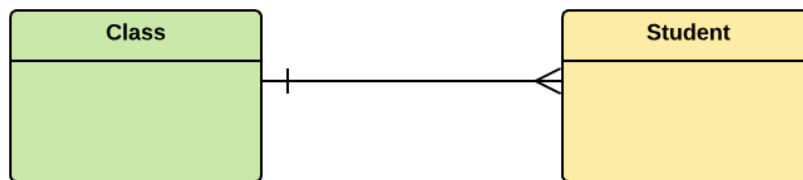
2. One to Many Relationship

When a single instance of an entity is connected to more than one instances of another entity then this type of relationship is called one to many relationship. For example – a customer can

purchase many orders but a order cannot be placed by many customers.



For example, one class is consisting of multiple students and multiple students may have in a class.

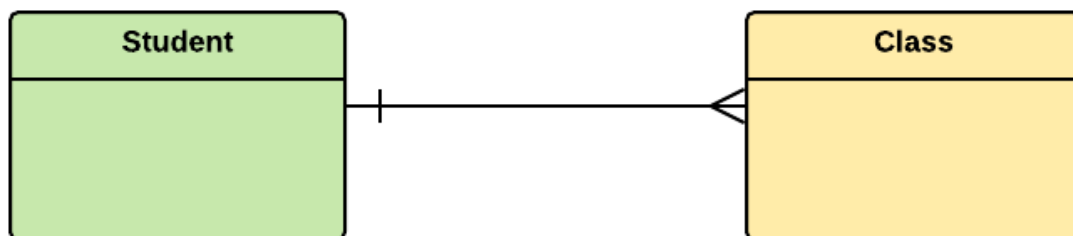


3. Many to One Relationship

When more than one instances of an entity is connected to a single instance of another entity then this type of relationship is called many to one relationship. For example – many students studies in a single college but a student cannot study in many colleges at the same time.



For example, many students belong to the same class.

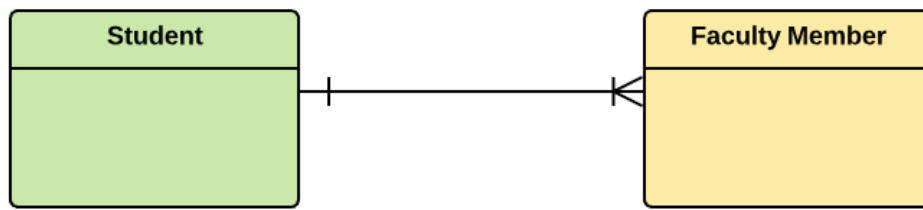


4. Many to Many Relationship

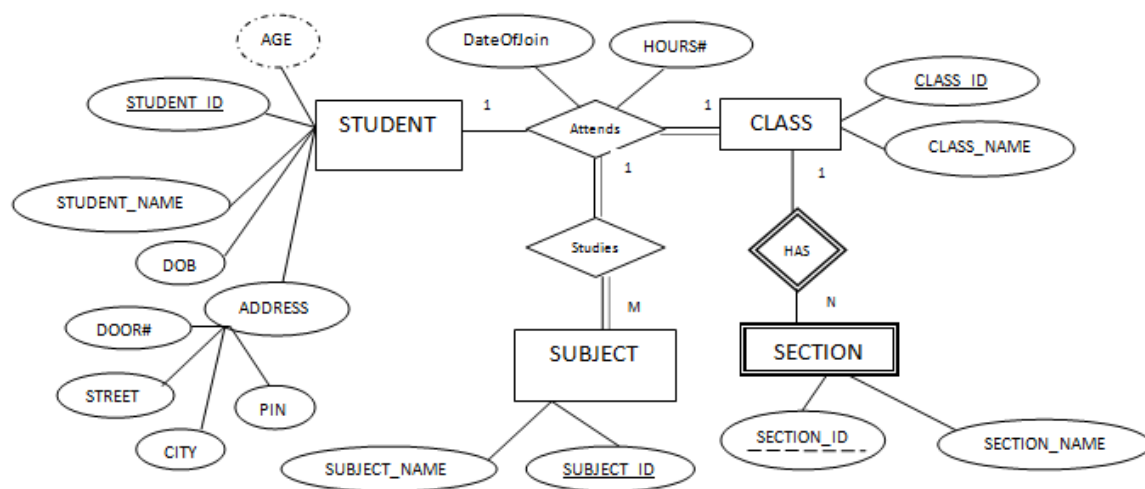
When more than one instances of an entity is connected to more than one instances of another entity then this type of relationship is called many to many relationship. For example, a student can be allotted to many task or projects and a task or project can be issued to many students.



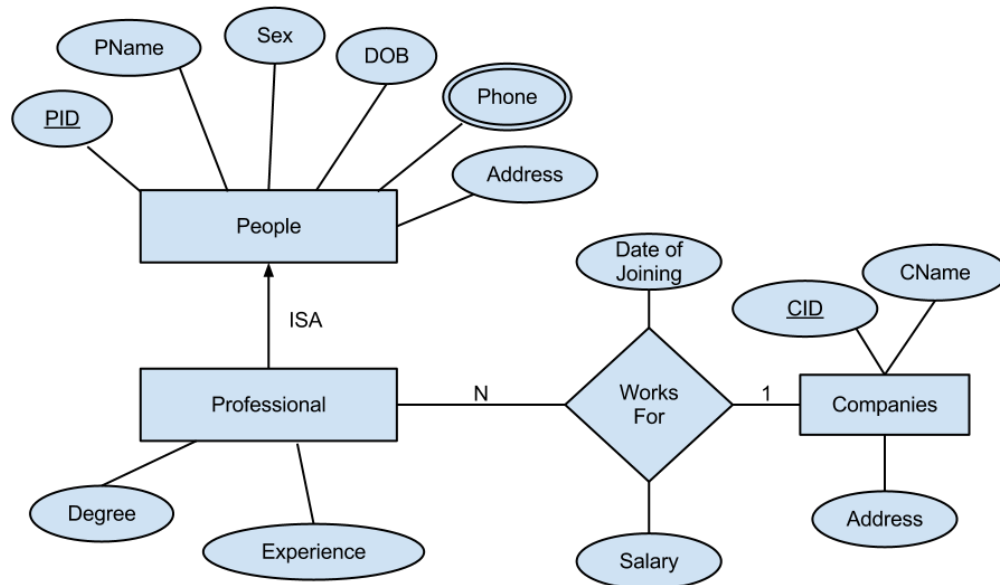
For example, group of students are connected to multiple faculty members, and a faculty members can be associated with multiple students.



Example of an ER diagram for College Database

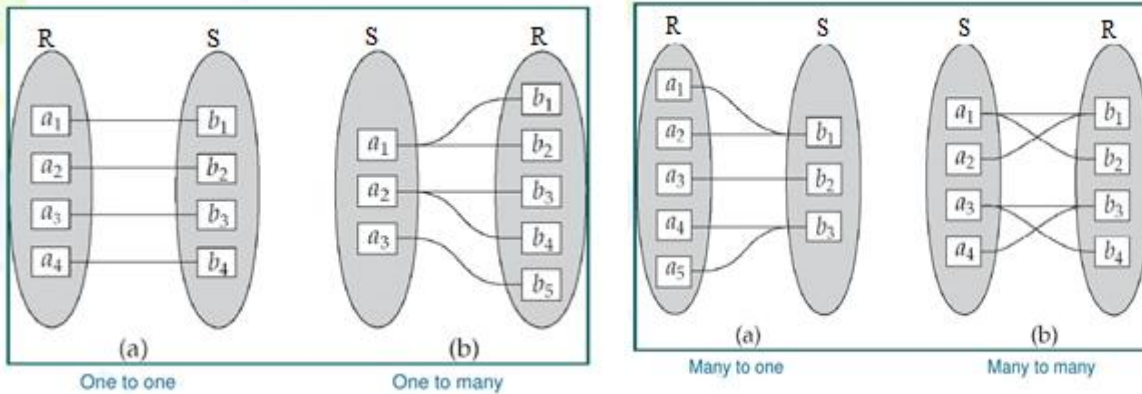


Example of an ER diagram for Company Database



Mapping Constraints

- A mapping constraint is a data constraint that describes the number of entities to which another entity can be connected by a relationship set.
- It is very important in describing the relationship sets that engage more than two entity sets.
- For relationship set R on an entity set R and S, there are four probable mapping cardinalities which are as follows:
 - One to one (1:1)
 - One to many (1:M)
 - Many to one (M:1)
 - Many to many (M:M)



- One to One: An entity of entity-set R can be connected with at most one entity of entity-set S and an entity in entity-set S can be connected with at most one entity of entity-set R.
- One to Many: An entity of entity-set R can be connected with any number of entities of entity-set S and an entity in entity-set S can be connected with at most one entity of entity-set R.
- Many to One: An entity of entity-set R can be connected with at most one entity of entity-set S and an entity in entity-set S can be connected with any number of entities of entity-set R.
- Many to Many: An entity of entity-set R can be connected with any number of entities of entity-set S and an entity in entity-set S can be connected with any number of entities of entity-set R.

DBMS

Unit-2

Relational data Model

Relational data model is the primary data model, which is used widely around the world for data storage and processing. This model is simple and it has all the properties and capabilities required to process data with storage efficiency.

A relational model can be represented as a table of rows and columns. It represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship. Some popular Relational Database management systems are:

- DB2 and Informix Dynamic Server - IBM
- Oracle and RDB – Oracle
- SQL Server and Access – Microsoft

The table name and column names are helpful to interpret the meaning of values in each row.

The data are represented as a set of relations. In the relational model, data are stored as tables.

However, the physical storage of the data is independent of the way the data are logically organized.

Relational Model Concepts

1. Table

A table is a collection of data represented in rows and columns. Each table has a name in database. In the Relational model the, relations are saved in the table format. It is stored along with its entities. Rows represent records and columns represent attributes.

For example, the following table “STUDENT” stores the information of students in database.

Table: STUDENT

Student_Id	Student_Name	Student_Addr	Student_Age
101	Chaitanya	Dayal Bagh, Agra	27
102	Ajeet	Delhi	26
103	Rahul	Gurgaon	24
104	Shubham	Chennai	25

2. Record or Tuple

Each row of a table is known as record. It is also known as tuple. For example, the following row is a record that we have taken from the above table.

102	Ajeet	Delhi	26
-----	-------	-------	----

3. Attribute

Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME, etc.

The above table “STUDENT” has four fields (or attributes): Student_Id, Student_Name, Student_Addr & Student_Age.

4. Domain

A domain is a set of permitted values for an attribute in table. For example, a domain of month-of-year can accept January, February, ... December as values, a domain of dates can accept all possible valid dates etc. We specify domain of attribute while creating a table.

An attribute cannot accept values that are outside of their domains. For example, In the above table “STUDENT”, the Student_Id field has integer domain so that field cannot accept values that are not integers for example, Student_Id cannot have values like, “First”, 10.11 etc.

- 5. Degree:** The total number of attribute in the relation is called the degree of the relation.
- 6. Column:** The column represents the set of values for a specific attribute.
- 7. Cardinality:** Total number of rows present in the Table.
- 8. Instance:** The data stored in database at a particular moment of time is called instance of database.
- 9. Schema:** Design of a database is called the schema. Schema is of three types: Physical schema, logical schema and view schema.

Table also called Relation

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Primary Key

Domain
Ex: NOT NULL

© guru99.com

Tuple OR Row

Total # of rows is Cardinality

Column OR Attributes

Total # of column is Degree

Dr Edgar F Codd-----Codd's 12 Rule

E.F Codd was a Computer Scientist who invented the **Relational model** for Database management. Based on relational model, the **Relational database** was created. Codd proposed 13 rules popularly known as **Codd's 12 rules** to test DBMS's concept against his relational model. Codd's rule actually define what quality a DBMS requires in order to become a Relational Database Management System (RDBMS). Till now, there is hardly any commercial product that follows all the 13 Codd's rules. Even **Oracle** follows only eight and half (8.5) out of 13. The Codd's 12 rules are as follows.

Rule zero: Foundation Rule

This rule states that for a system to qualify as an **RDBMS**, it must be able to manage database entirely through the relational capabilities.

Rule 1: Information rule

All information (including metadata) is to be represented as stored data in **cells of tables**. The rows and columns have to be strictly unordered.

Rule 2: Guaranteed Access

Each unique piece of data (atomic value) should be accessible by : **Table Name + Primary Key(Row) + Attribute(column)**.

Rule 3: Systematic treatment of NULL

Null has several meanings; it can mean missing data, not applicable or no value. It should be handled consistently. Also, Primary key must not be null, ever. Expression on NULL must give null.

Rule 4: Active Online Catalog

Database dictionary (catalog) is the structure description of the complete **Database** and it must be stored online. The Catalog must be governed by same rules as rest of the database. The same query language should be used on catalog as used to query database.

Rule 5: Powerful and Well-Structured Language

One well structured language must be there to provide all manners of access to the data stored in the database. Example: **SQL**, etc. If the database allows access to the data without the use of this language, then that is a violation.

Rule 6: View Updation Rule

All the view that are theoretically updatable should be updatable by the system as well.

Rule 7: Relational Level Operation

There must be Insert, Delete, Update operations at each level of relations. Set operation like Union, Intersection and minus should also be supported.

Rule 8: Physical Data Independence

The physical storage of data should not matter to the system. If say, some file supporting table is renamed or moved from one disk to another, it should not effect the application.

Rule 9: Logical Data Independence

If there is change in the logical structure (table structures) of the database the user view of data should not change. Say, if a table is split into two tables, a new view should give result as the join of the two tables. This rule is most difficult to satisfy.

Rule 10: Integrity Independence

The database should be able to enforce its own integrity rather than using other programs. Key and Check constraints, trigger etc, should be stored in Data Dictionary. This also makes **RDBMS** independent of front-end.

Rule 11: Distribution Independence

A database should work properly regardless of its distribution across a network. Even if a database is geographically distributed, with data stored in pieces, the end user should get an impression that it is stored at the same place. This lays the foundation of **distributed database**.

Rule 12: Nonsubversion Rule

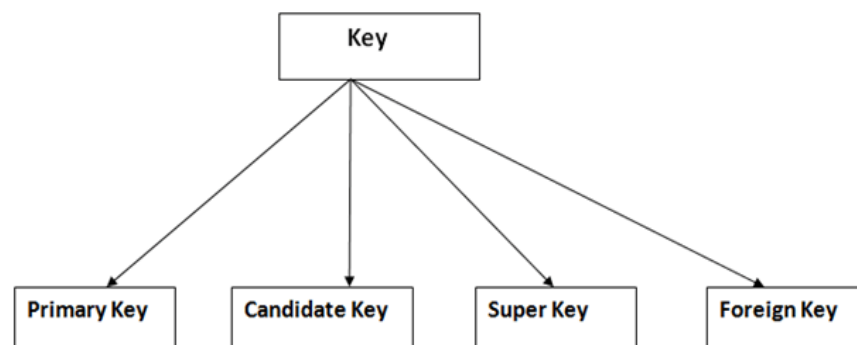
If low level access is allowed to a system it should not be able to subvert or bypass integrity rules to change the data. This can be achieved by some sort of locking or encryption.

Keys

Key plays an important role in database; it is used for identifying unique rows (tuple) from table. It also establishes relationship among tables. Keys ensure integrity of data is maintained.

A DBMS Key is an attribute or collection of attributes that uniquely identifies a record or a row of data in a relation(table).

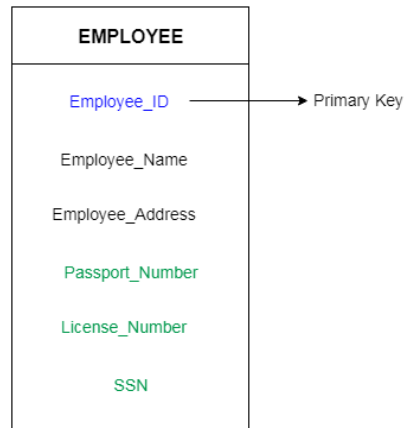
For example, the roll_number of a student makes him/her identifiable among students.



Primary key

- It is the first key which is used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys as we saw in PERSON table. The key which is most suitable from those lists become a primary key.
- We usually denote it by underlining the attribute name (column name).

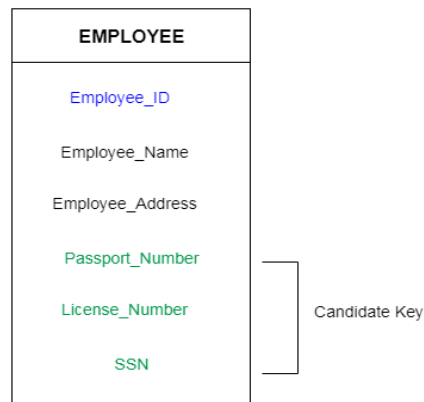
- The value of primary key should be unique and not null.
- In the EMPLOYEE table, ID can be primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License_Number and Passport_Number as primary key since they are also unique.



Candidate key

- A candidate key is an attribute or set of an attribute which can uniquely identify a tuple.
- The remaining attributes except for primary key are considered as a candidate key. The candidate keys are as strong as the primary key. It should not have any redundant attribute.

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



Super Key

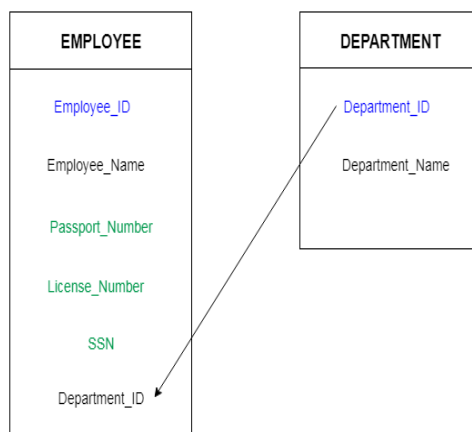
Super key is a set of an attribute which can uniquely identify a tuple. Super key is a superset of a candidate key.

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

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

Foreign key

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



Student

St_Roll_No	St_Name	St_DOB	Course_Id
156	Ram	15.10.2000	BCA
160	Shyam	02.11.2001	MCA
170	Mohan	05.05.2000	PGDCA
190	Sohan	01.02.2002	B.Tech
108	Geeta	07.08.2001	B.Sc Nursuing

← Not Allowed

Course

Course_Id	Course_dept	Course_duration	Course_room
BCA	Computer Application	3	15
MCA	Computer Application	3	30
PGDCA	Computer Application	1	17
B.Tech	Engineering	4	20

Integrity Constraints

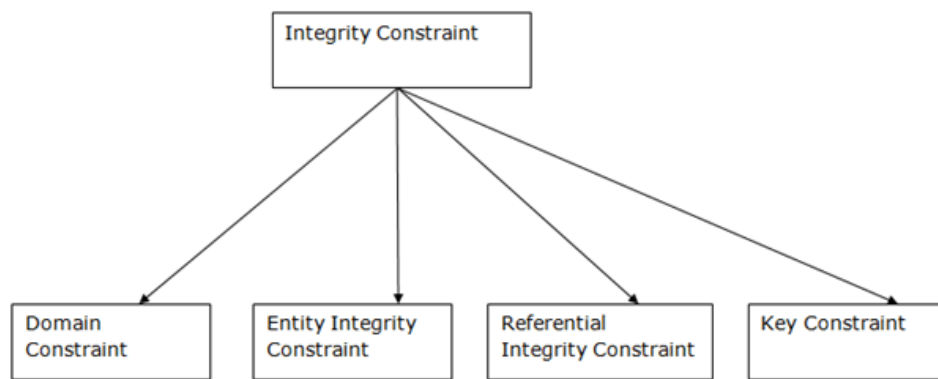
Data integrity is the overall accuracy, completeness, and consistency of data. Data integrity also refers to the safety of data. When creating databases, attention needs to be given to data integrity and how to maintain it. A good database will enforce data integrity whenever possible.

Integrity constraints ensure that when the authorized users **modify** the database they do not disturb the **data consistency**. Integrity constraints are introduced at the time of designing the **database schema**. The constraints are specified within the SQL DDL command like 'create table' and 'alter table' command.

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.

```
CREATE TABLE table_name (  
    column1 datatype constraint,  
    column2 datatype constraint,  
    column3 datatype constraint,  
    ....  
);
```

Types of Integrity Constraint



1. Domain constraints

Domain constraint ensures that the value associated with an attribute is justifying its domain. Whenever we declare any relation to the database while declaring its attribute we specify a particular domain for each attribute.

The domain of an attribute specifies all the possible values that attribute can hold. Declaring the domain of an attribute it acts as a constraint on that attribute which specifies the possible values that it can take.

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

Example:

ID	NAME	SEMENSTER	AGE
1000	Tom	1 st	17
1001	Johnson	2 nd	24
1002	Leonardo	5 th	21
1003	Kate	3 rd	19
1004	Morgan	8 th	A

Not allowed. Because AGE is an integer attribute

```
CREATE TABLE Student (  
    Roll_no int,  
    Name varchar(255),  
    F_Name varchar(255),  
);
```

STUDENT

Roll_No	Name	F_Name
156	Ram	Prakash
160	158	Deepak
170	Mohan	152
Geeta	Sohan	Ashish

Not Allowed

2. Entity integrity constraints

Entity integrity constraint ensures that the **primary key attribute** in a relation, should **not** accept a **null value**. This is because the primary key attribute value **uniquely** defines an entity in a relation. So, it being null would not work.

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

Example:

EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

```
CREATE TABLE Studnet (  
    Roll_no int,  
    Name varchar(255),  
    F_Name varchar(255),  
    PRIMARY KEY (Roll_no)  
);
```

Student

Roll_No	Name	F_Name
156	Ram	Prakash
160	Ankit	Deepak
170	Mohan	Jeevan
	Sohan	Ashish

Not Allowed

3. Referential Integrity Constraints

Referential integrity is concerned with relationships. When two or more tables have a relationship, we have to ensure that the foreign key value matches the primary key value at all times. We don't want to have a situation where a foreign key value has no matching primary key value in the primary table. This would result in an orphaned record.

- A referential integrity constraint is specified between two tables.

- 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.

To understand this clearly let us take another example of student relation: *Student*(*St_Roll_No*, *St_Name*, *St_DOB*, *Course_Id*)

Student

St_Roll_No	St_Name	St_DOB	Course_Id
156	Ram	15.10.2000	BCA
160	Shyam	02.11.2001	MCA
170	Mohan	05.05.2000	PGDCA
190	Sohan	01.02.2002	B.Tech
108	Geeta	07.08.2001	B.Sc Nursuing

Not
Allowed

Course

Course_Id	Course_dept	Course_duration	Course_room
BCA	Computer Application	3	15
MCA	Computer Application	3	30
PGDCA	Computer Application	1	17
B.Tech	Engineering	4	20

Here, we can see that student relation has the attribute *Course_Id*. Now suppose, we have a student tuple with the *Course_Id* value as **B.Sc Nursing** . Then the **Course** relation must also have a tuple that would have *Course_Id* attribute value as **B.Sc Nursing**. If the Department relation does not have any tuple with the *Course_Id* **B.Sc.Nursing** how had the student opt the **B.Sc Nursing** course?

4. Keys constraints

- Keys are the entity set that is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

Example:

ID	NAME	SEMENSTER	AGE
1000	Tom	1 st	17
1001	Johnson	2 nd	24
1002	Leonardo	5 th	21
1003	Kate	3 rd	19
1002	Morgan	8 th	22

Not allowed. Because all row must be unique

```
CREATE TABLE Students (  
    Roll_no int NOT NULL,  
    Name varchar(255) NOT NULL,  
    F_Name varchar(255) NOT NULL,  
    Aadhar_no int UNIQUE,  
    City varchar(255) DEFAULT 'Unknown'  
    Age int,  
    CHECK (Age>=21)  
    PRIMARY KEY (Roll_no)  
);
```

STUDNET

Roll_No	Name	F_Name	Aadhar_no	City	Age
156	Ram	Prakash	985845672252	Moradabad	22
160	Radha	Deepak	524589752158	Delhi	23
170		Ankit	524586545664	Moradabad	19
170	Pinky	Bablu	985845672252	Unknown	21

Not Allowed

SQL(Structure Query Language)

SQL stands for Structured Query Language. Structure Query Language (SQL) is a database query language used for storing and managing data in Relational DBMS. SQL was the first commercial language introduced for E.F Codd's **Relational** model of database. Today almost all RDBMS (MySQL, SQL server, Oracle, Informix, Sybase, MS Access, PostgreSQL) use **SQL** as the standard database query language. It enables a user to **create**, **read**, **update** and **delete** relational databases and tables.

Rules:

SQL follows the following rules:

- Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.
- Using the SQL statements, we can perform most of the actions in a database.
- SQL depends on tuple relational calculus and relational algebra.

Characteristics of SQL

- SQL is easy to learn.
- SQL is used to access data from relational database management systems.
- SQL can execute queries against the database.
- SQL is used to describe the data.
- SQL is used to define the data in the database and manipulate it when needed.
- SQL is used to create and drop the database and table.
- SQL is used to create a view, stored procedure, function in a database.
- SQL allows users to set permissions on tables, procedures, and views.

Advantages of SQL

There are the following advantages of SQL:

- **High speed**

Using the SQL queries, the user can quickly and efficiently retrieve a large amount of records from a database.

- **No coding needed**

In the standard SQL, it is very easy to manage the database system. It doesn't require a substantial amount of code to manage the database system.

- **Well defined standards**

Long established are used by the SQL databases that are being used by ISO and ANSI. There are no standards adhered by the non-SQL databases.

- **Portability**

SQL can be used in the program in PCs, servers, laptops, and even some of the mobile phones.

- **Interactive language**

SQL is a domain language used to communicate with the database. It is also used to receive answers to the complex questions in seconds.

- **Multiple data view**

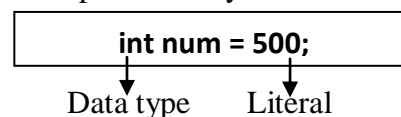
Using the SQL language, the users can make different views of the database structure and databases for the different users.

SQL Data type and literals

SQL Data Type is an attribute that specifies the type of data of any object. Each column, variable and expression has a related data type in SQL. We can use these data types while creating our tables. We can choose a data type for a table column based on our requirement.

- SQL Data type is used to define the values that a column can contain.
- Every column is required to have a name and data type in the database table.

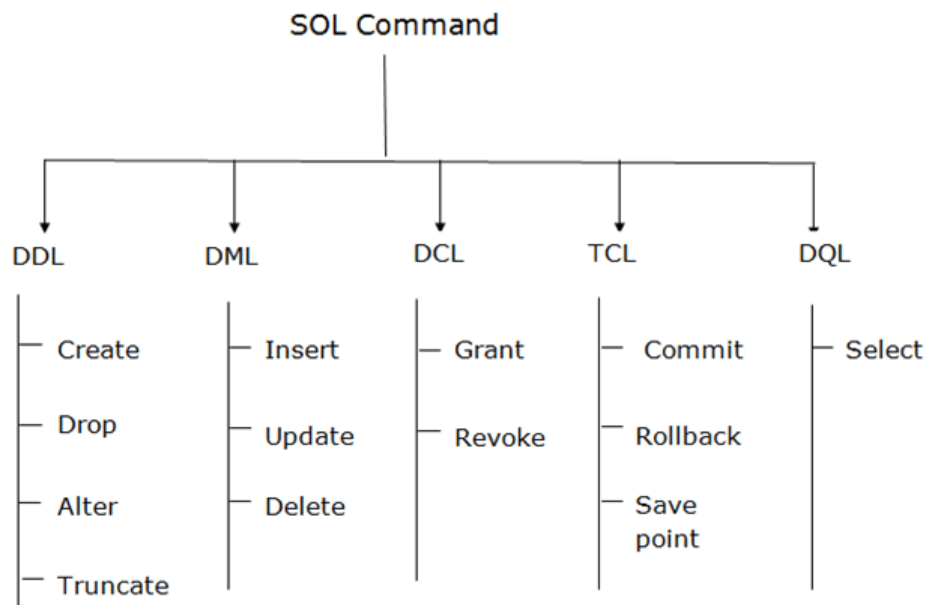
Data types are those which specify the type of data represented by the variable and literal is the value that is stored in to the variable.



Datatype	Use	Literal
INT	used for columns which will store integer values.	050 ,78, -14, 0 , +32767
FLOAT	used for columns which will store float values.	6.2, 2.9, 55.89
VARCHAR	used for columns which will be used to store characters and integers, basically a string.	'Hello world!'
DATE	used for columns which will store date values.	'1978-12-25';
TIME	used for columns which will store time values.	'12:01:01';
TEXT	used for columns which will store text which is generally long in length. For example, if you create a table for storing profile information of a social networking website, then for about me section you can have a column of type TEXT.	'Hello Myself John. I am a database designer. '

Types of SQL Commands

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.



1. Data Definition Language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

1.	<p>CREATE It is used to create a new table in the database.</p> <p>Syntax:</p> <pre>CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES [,...]);</pre> <p>Example:</p> <pre>CREATE TABLE Student(St_Name VARCHAR2(20), Stu_id int (20), Email VARCHAR2(100), DOB DATE);</pre>
2.(a)	<p>ALTER: It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.</p> <p>Syntax:</p> <p>To add a new column in the table</p> <pre>ALTER TABLE table_name ADD column_name datatype;</pre> <p>EXAMPLE</p> <pre>ALTER TABLE Students ADD Address varchar(255);</pre>
(b)	<p>To delete a column in a table</p> <pre>ALTER TABLE table_name DROP COLUMN column_name;</pre> <p>Example</p> <pre>ALTER TABLE Students DROP COLUMN address;</pre>

(c)	<p>To change the data type of a column in a table</p> <p>ALTER TABLE <i>table_name</i></p> <p>ALTER COLUMN <i>column_name datatype</i>;</p> <p>Example</p> <p>ALTER TABLE Persons</p> <p>ALTER COLUMN DateOfBirth year;</p>
3.	<p>DROP: It is used to drop an existing table in a database. It is used to delete both the schema/structure and record stored in the table. DROP is permanently lost and it cannot be rolled back.</p> <p>Syntax</p> <p>DROP TABLE <i>table_name</i>;</p> <p>Example</p> <p>DROP TABLE Employees;</p>
4.	<p>TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.</p> <p>Syntax:</p> <p>TRUNCATE TABLE <i>table_name</i>;</p>

2. Data Manipulation Language

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

1.(a)	<p>INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.</p> <p>Syntax:</p> <p>INSERT INTO <i>table_name</i> (<i>column1</i>, <i>column2</i>, <i>column3</i>, ...)</p> <p>VALUES (<i>value1</i>, <i>value2</i>, <i>value3</i>, ...);</p> <p>Example</p> <p>INSERT INTO Customers (CustomerName, City, Country)</p> <p>VALUES ('Ramesh', Moradabad', India');</p>
(b)	<p>For adding values for all the columns of the table, we do not need to specify the column names in the SQL query.</p> <p>INSERT INTO <i>table_name</i></p> <p>VALUES (<i>value1</i>, <i>value2</i>, <i>value3</i>, ...);</p>
2.	<p>UPDATE: This command is used to update or modify the value of a column in the table.</p> <p>Syntax:</p> <p>UPDATE <i>table_name</i></p> <p>SET <i>column1</i> = <i>value1</i>, <i>column2</i> = <i>value2</i>, ...</p> <p>WHERE <i>condition</i>;</p> <p>Example</p> <p>UPDATE Students</p> <p>SET Stu_Add = 'Rampur', Phone= '4568233545'</p> <p>WHERE St_rollno = 184524552;</p>
3.	<p>DELETE: It is used to remove one or more row from a table.</p> <p>Syntax:</p> <p>DELETE FROM <i>table_name</i> WHERE <i>condition</i>;</p> <p>Example</p> <p>DELETE FROM Students WHERE St_Id='1820530255';</p>

3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

- Grant
- Revoke

1.	<p>Grant: It is used to give user access privileges to a database.</p> <p>Syntax:</p> <p>GRANT <privilege list> ON <table name or view name> TO <user/role list>;</p> <p>Example</p> <p>GRANT SELECT, UPDATE ON Student TO Faculty1, Faculty2;</p> <p>Or</p> <p>GRANT ALL ON Student TO DIRECTOR;</p>
2.	<p>Revoke: It is used to take back permissions from the user.</p> <p>Syntax:</p> <p>REVOKE <privilege list> ON <relation name or view name> FROM <user name>;</p> <p>Example</p> <p>REVOKE SELECT, UPDATE ON Student FROM Faculty1, Faculty2;</p>

4. Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

1.	<p>COMMIT: Commit command is used to save all the transactions to the database.</p> <p>Syntax:</p> <p>COMMIT;</p> <p>Example</p> <p>DELETE FROM STUDENT WHERE AGE = 25; COMMIT;</p>
----	--

2.	<p>Rollback: This command restores database to original since the last COMMIT. It is used to restores the database to last committed state.</p> <p>Syntax: ROLLBACK;</p> <p>Example DELETE FROM CUSTOMERS WHERE AGE = 25; ROLLBACK;</p>
3.	<p>Savepoint: It is used for identifying a point in the transaction to which a programmer can later roll back. It is used to temporarily save a transaction.</p> <p>Syntax: SAVEPOINT savepointname;</p> <p>Example UPDATE... DELETE... SAVEPOINT s1; INSERT.... UPDATE.... SAVEPOINT s2; INSERT.... UPDATE.... ROLLBACK TO SAVEPOINT s2; This results in the rollback of all statements after savepoint s2.</p>

5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

1.	<p>SELECT: The SELECT statement is used to select data from a database.</p> <p>Syntax: SELECT column1, column2, ... FROM table_name;</p>
----	--

	Example SELECT <i>stu_rollno ,stu_name, stu_add</i> FROM <i>student</i> ; Or SELECT * FROM <i>student</i> ; Or SELECT student_name FROM student WHERE age > 20; Select stu_roll, stu_name FROM student WHERE course=BCA;
--	--

SQL Operators

While handling data in databases, we often need to perform different kinds of operations to manipulate and retrieve data. SQL being the base of database management systems, offers various operators to perform such operations. SQL operators are reserved keywords used in the WHERE clause of a SQL statement to perform arithmetic, logical and comparison operations.

Generally there are three types of operators in SQL:

1. SQL Arithmetic Operators
2. SQL Comparison Operators
3. SQL Logical Operators

SQL Arithmetic Operators:

These operators are used to perform operations such as addition, multiplication, subtraction etc.

Operator	Operation	Description
+	Addition	Add values on either side of the operator
-	Subtraction	Used to subtract the right hand side value from the left hand side value
*	Multiplication	Multiples the values present on each side of the operator
/	Division	Divides the left hand side value by the right hand side value
%	Modulus	Divides the left hand side value by the right hand side value; and returns the remainder

Example

```

SELECT 40 + 20;
SELECT 40 - 20;
SELECT 40 * 20;
SELECT 40 / 20;

```

SELECT 40 % 20;

Output:

60
20
800
2
0

SQL Comparison Operators

These operators are used to perform operations such as equal to, greater than, less than etc.

Operator	Operation	Description
=	Equal to	Used to check if the values of both operands are equal or not. If they are equal, then it returns TRUE.
>	Greater than	Returns TRUE if the value of left operand is greater than the right operand.
<	Less than	Checks whether the value of left operand is less than the right operand, if yes returns TRUE.
>=	Greater than or equal to	Used to check if the left operand is greater than or equal to the right operand, and returns TRUE, if the condition is true.
<=	Less than or equal to	Returns TRUE if the left operand is less than or equal to the right operand.
<> or !=	Not equal to	Used to check if values of operands are equal or not. If they are not equal then, it returns TRUE.
!>	Not greater than	Checks whether the left operand is not greater than the right operand, if yes then returns TRUE.
!<	Not less than	Returns TRUE, if the left operand is not less than the right operand.

Example- We have a table of student:

StudentID	FirstName	LastName	Age
1	Atul	Mishra	23
2	Priya	Kapoor	21
3	Rohan	Singhania	21
4	Akanksha	Jain	20
5	Vaibhav	Gupta	25

Example [Equal to]:

SELECT * FROM Students
WHERE Age = 20;

Output:

StudentID	FirstName	LastName	Age
4	Akanksha	Jain	20

Example [Greater than]:

```
SELECT * FROM students
WHERE Age > 23;
```

Output:

StudentID	FirstName	LastName	Age
5	Vaibhav	Gupta	25

Example [Less than or equal to]:

```
SELECT * FROM students
WHERE Age <= 21;
```

Output:

StudentID	FirstName	LastName	Age
2	Priya	Kapoor	21
3	Rohan	Singhania	21
4	Akanksha	Jain	20

Example [Not equal to]:

```
SELECT * FROM students
WHERE Age <> 25;
```

Output:

StudentID	FirstName	LastName	Age
1	Atul	Mishra	23
2	Priya	Kapoor	21
3	Rohan	Singhania	21
4	Akanksha	Jain	20

SQL Logical Operators

The logical operators are used to perform operations such as AND,OR,NOT, BETWEEN etc.

Operator	Description
AND	Allows the user to mention multiple conditions in a WHERE clause.
OR	Combines multiple conditions in a WHERE clause.
NOT	A negate operators, used to reverse the output of the logical operator.
IN	Used to compare a specific value to the literal values mentioned.
BETWEEN	Searches for values within the range mentioned.

For Ex. - **SELECT** ID, NAME, SALARY

FROM CUSTOMERS

WHERE SALARY > 10000 **AND** AGE < 21;

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 10000 OR AGE < 21;
```

Example [BETWEEN AND]

```
SELECT * FROM Students
WHERE Age BETWEEN 22 AND 25;
```

Output:

StudentID	FirstName	LastName	Age
1	Atul	Mishra	23

Example [IN]

```
SELECT * FROM Students
WHERE Age IN ('23', '20');
```

Output:

StudentID	FirstName	LastName	Age
1	Atul	Mishra	23
4	Akanksha	Jain	20

Tables

Tables are database objects that contain all the data in a database. In tables, data is logically organized in a row-and-column format similar to a spreadsheet. Each row represents a unique record, and each column represents a field in the record.

For example, a table that contains student data for a university might contain a row for each student and columns representing student information such as student name, roll no, address, course, and telephone number.

- The number of tables in a database is limited only by the number of objects allowed in a database (2,147,483,647). Objects include tables, views, stored procedures, user-defined functions, triggers, rules, defaults, and constraints. The sum of the number of all objects in a database cannot exceed 2,147,483,647. A standard user-defined table can have up to 1,024 columns. The number of rows in the table is limited only by the storage capacity of the server.

- We can assign properties to the table and to each column in the table to control the data that is allowed and other properties. For example, we can create constraints on a column to disallow null values or provide a default value if a value is not specified, or we can assign a key constraint on the table that enforces uniqueness or defines a relationship between tables.
- Syntax to create a table:
`CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES [...]);`

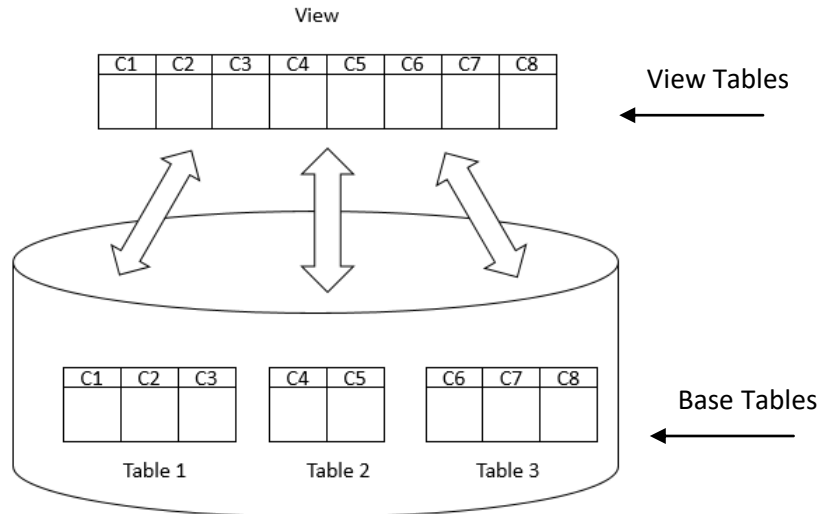
Views in SQL

View can be described as virtual table which derived its data from one or more than one table columns. Views are used for security purposes because they provide encapsulation of the name of the table.

Data in the virtual table is not stored permanently. Views display only selected data. A view does not contain any data or exist in physical storage.

A **view** is simply any SELECT query that has been given a name and saved in the database. For this reason, a view is sometimes called a **named query** or a **stored query**.

- The view does not take up any disk space for data storage, and it does not create any redundant copies of data that is already stored in the tables that it references (which are sometimes called the **base tables** of the view).
- Syntax to create a view:
`CREATE VIEW view_name AS`
`SELECT column1, column2, ...`
`FROM table_name`
`WHERE condition;`



Using views

A view name may be used in exactly the same way as a table name in any SELECT query. Once stored, the view can be used again and again, rather than re-writing the same query many times.

- One of the most important uses of views is in large multi-user systems, where they make it easy to control access to data for different types of users. As a very simple example, suppose that we have a table of employee information as:

Employees = {employeeID, empFName, empLName, empPhone, jobTitle, payRate, managerID}. Obviously, we can't let everyone in the company look at all of this information.

- Our database administrator (DBA) can define roles to represent different groups of users, and then grant membership in one or more roles to any specific user account (schema). In turn, we can grant table-level or view-level permissions to a role as well as to a specific user. Suppose that the DBA has created the roles managers and payroll for people who occupy those positions. In Oracle®, there is also a pre-defined role named public, which means every user of the database.

We could create separate views even on just the Employees table, and control access to them like this:

```
CREATE VIEW phone_view AS
SELECT empFName, empLName, empPhone FROM Employees;
GRANT SELECT ON phone_view TO public;
```

```
CREATE VIEW job_view AS
SELECT employeeID, empFName, empLName, jobTitle, managerID FROM Employees;
GRANT SELECT, UPDATE ON job_view TO managers;
```

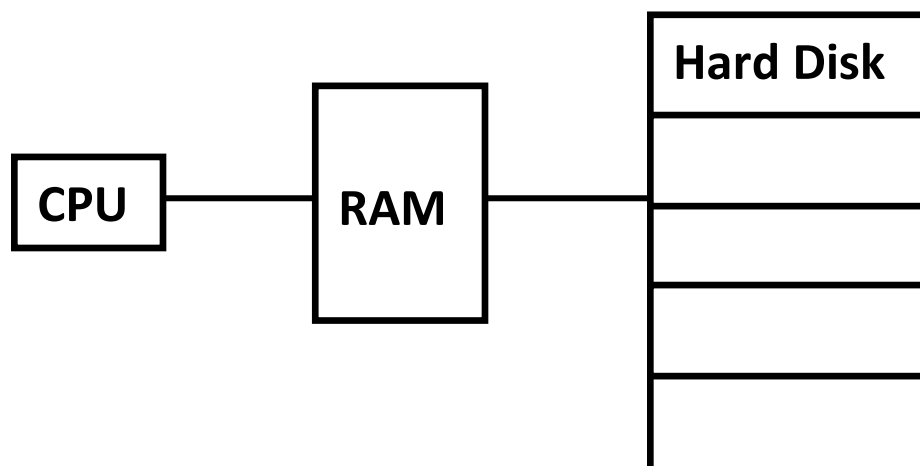
```
CREATE VIEW pay_view AS
SELECT employeeID, empFName, empLName, payRate FROM Employees;
GRANT SELECT, UPDATE ON pay_view TO payroll;
```

- Only a very few trusted people would have SELECT, UPDATE, INSERT, and DELETE privileges on the entire Employees base table; everyone else would now have exactly the access that they need, but no more.
- When a view is the target of an UPDATE statement, the base table value is changed. We can't change a computed value in a view, or any value in a view that is based on a UNION query. We may also use a view as the target of an INSERT or DELETE statement, subject to any integrity constraints that have been placed on the base tables.

Indexes

An index is a table that indicates where data is stored in another table. It contains the locations of specified columns in the base table that are queried frequently. The index can speed up the retrieval of information.

We can examine an index's structure but cannot perform any other operations on it in the Tables Utility. We create an index on one or more columns of a table, using a query language such as SQL. Whenever a user enters a query based on the indexed column in the base table, the index helps locate the information quickly. Use of indexes is recommended to improve performance of the queries in our applications. Index table only have (Key value and Pointer).



Syntax:

CREATE INDEX *index_name*

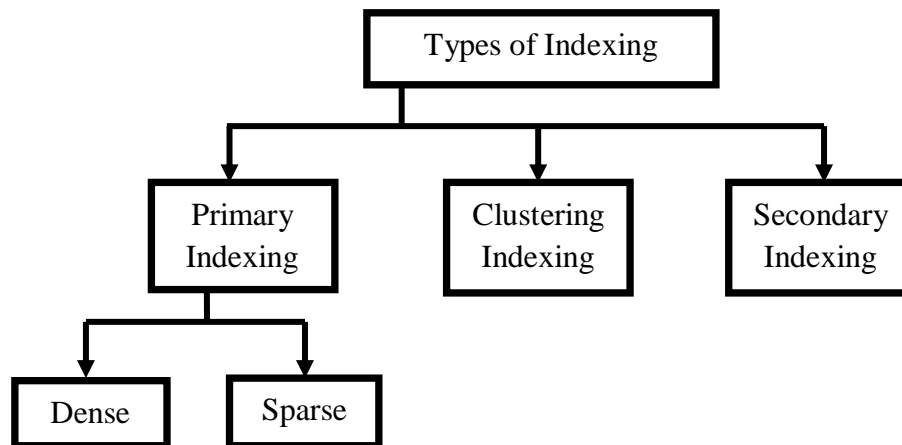
ON *table_name* (*column1*, *column2*, ...);

Example

CREATE INDEX idx_pname

ON Persons (LastName, FirstName);

Types of Indexing



Indexing is defined based on its indexing attributes. Indexing can be of the following types –

- **Primary Index** – Primary index is defined on an ordered data file. The data file is ordered on a **key field**. The key field is generally the primary key of the relation.
- **Clustering Index** – Clustering index is defined on an ordered data file. The data file is ordered on a non-key field. In this type of indexing value may repeat.
- **Secondary Index** – Secondary index is defined on an unordered data file. It may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.

Primary (Ordered) Indexing is of two types –

- Dense Index
- Sparse Index

Dense Index

In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.



Sparse Index

In sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search a record, we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.



QUERY

- A query is an operation that retrieves data from one or more tables or views.
- SELECT statement can be used for retrieving the data from various tables in a database.

Example:

<Employee> Table

Eid	Ename	Age	City	Salary
E001	ABC	29	Pune	20000
E002	PQR	30	Pune	30000
E003	LMN	25	Mumbai	5000

E004	XYZ	24	Mumbai	4000
E005	STU	32	Bangalore	25000

- Selecting all columns (SELECT *)
`SELECT * FROM Employee;`
- Displaying particular record with condition (WHERE)
`SELECT Ename FROM Employee`
`WHERE City = 'Pune';`

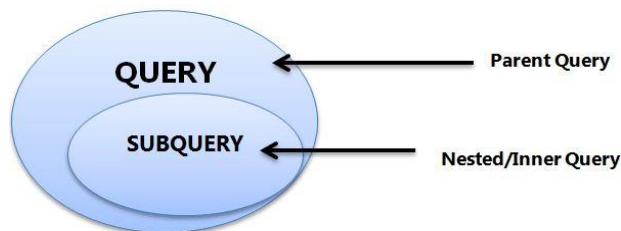
Output:

Ename
ABC
PQR

SUB-QUERY

In SQL a Subquery can be simply defined as a query within another query. In other words we can say that a Subquery is a query that is embedded in WHERE clause of another SQL query.

- Sub-queries are mostly appear within the WHERE clause of another SQL statement.
- It produces a single column of data as its result.
- Sub-query is always enclosed in parentheses.
- It cannot be a UNION, only a single SELECT statement is allowed.



Example:

`SELECT Ename, Salary FROM Employee`
`WHERE Salary IN (SELECT MAX (Salary) FROM Employee);`

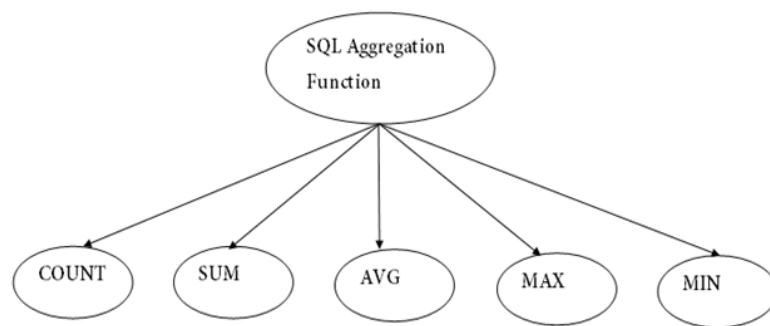
Output:

Ename	Salary
-------	--------

PQR	30000
-----	-------

Aggregate functions

- SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
- It is also used to summarize the data.
- Aggregate functions are often used with the GROUP BY clause of the SELECT statement.



1. COUNT FUNCTION

- COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.
- COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Syntax

COUNT(*)

or

SELECT COUNT(*column_name*)

FROM *table_name*

WHERE *condition*;

Example

SELECT COUNT(*)

FROM Students;

Or

```
SELECT COUNT(stu_name)
FROM Students
WHERE city=Moradabad;
```

2. SUM Function

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax:

```
SELECT SUM (column_name)
FROM table_name
WHERE condition;
```

Example

```
SELECT SUM (Marks)
FROM Students;
```

or

```
SELECT SUM (Marks)
FROM Students
WHERE course=DBMS;
```

3. AVG function

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax

AVG()

Or

```
SELECT AVG (Coloumn_name)
FROM Table_name;
```

Example

```
SELECT AVG(Price)
FROM Products;
```

4. MAX Function

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax

MAX()

or

SELECT MAX (Coloumn_name)

FROM Table_name;

Example:

SELECT MAX (Marks)

FROM Students

WHERE course =DBMS;

5. MIN Function

MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

Syntax

MIN()

or

SELECT MIN (Coloumn_name)

FROM Table_name;

Example:

SELECT MIN (Marks)

FROM Students

WHERE course =DBMS;

SQL JOIN

A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. A JOIN locates related column values in the two tables.

Different types of Joins are:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

Consider the two tables below:

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

The simplest Join is INNER JOIN.

1. **INNER JOIN:** The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be same.

Syntax:

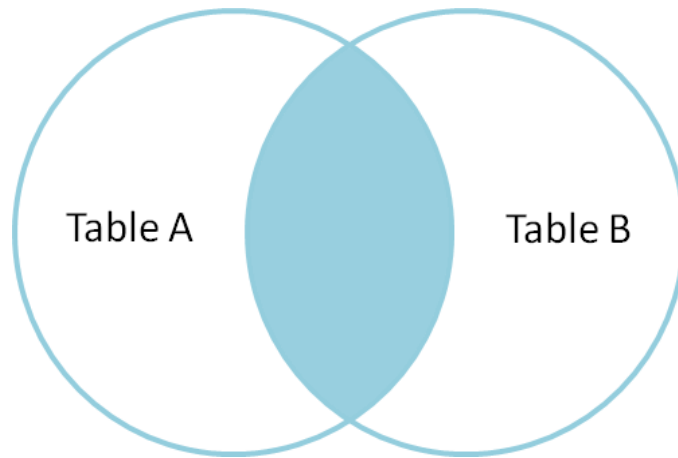
```
SELECT table1.column1, table1.column2, table2.column1, ....  
FROM table1 INNER JOIN table2  
ON table1.matching_column = table2.matching_column;
```

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

Note: We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.



Example Queries(INNER JOIN)

This query will show the names and age of students enrolled in different courses.

```
SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE  
FROM Student INNER JOIN StudentCourse  
ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```

Output:

COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

2. **LEFT JOIN:** This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on right side, the result-set will contain null. LEFT JOIN is also known as LEFT OUTER JOIN.Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
```

```
FROM table1 LEFT JOIN table2
```

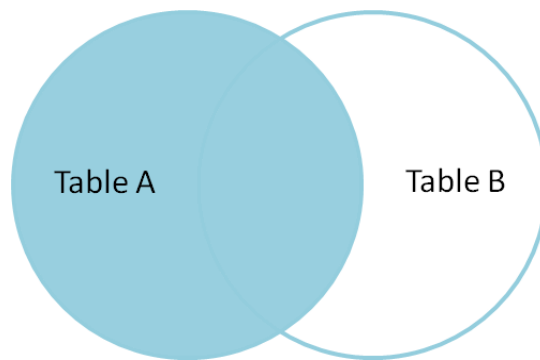
```
ON table1.matching_column = table2.matching_column;
```

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

Note: We can also use LEFT OUTER JOIN instead of LEFT JOIN, both are same.



Example Queries(LEFT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
LEFT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	NULL
ROHIT	NULL
NIRAJ	NULL

3. **RIGHT JOIN:** RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.Syntax:

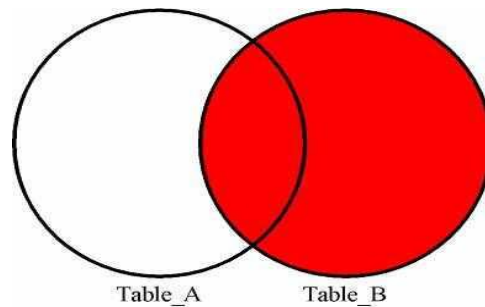
```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1 RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

Note: We can also use RIGHT OUTER JOIN instead of RIGHT JOIN, both are same.



Example Queries(RIGHT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student RIGHT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

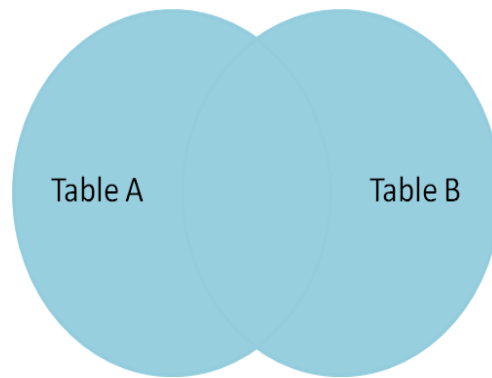
4. **FULL JOIN:** FULL JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain NULL values.Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,...  
FROM table1 FULL JOIN table2  
ON table1.matching_column = table2.matching_column;
```

table1: First table.

table2: Second table

matching_column: Column common to both the tables.



Example Queries(FULL JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID  
FROM Student FULL JOIN StudentCourse  
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	NULL
ROHIT	NULL
NIRAJ	NULL
NULL	9
NULL	10
NULL	11

SQL | SEQUENCES

Sequence is a set of integers 1, 2, 3, ... that are generated and supported by some database systems to produce unique values on demand.

- A sequence is a user defined schema bound object that generates a sequence of numeric values.
- Sequences are frequently used in many databases because many applications require each row in a table to contain a unique value and sequences provides an easy way to generate them.
- The sequence of numeric values is generated in an **ascending or descending order** at defined intervals and can be configured to restart when max_value exceeds.

Syntax:

```
CREATE SEQUENCE sequence_name  
START WITH initial_value  
INCREMENT BY increment_value  
MINVALUE minimum_value  
MAXVALUE maximum_value  
CYCLE|NOCYCLE ;
```

sequence_name: Name of the sequence.

initial_value: starting value from where the sequence starts.

Initial_value should be greater than or equal
to minimum value and less than equal to maximum value.

increment_value: Value by which sequence will increment itself.

Increment_value can be positive or negative.

minimum_value: Minimum value of the sequence.

maximum_value: Maximum value of the sequence.

cycle: When sequence reaches its set_limit
it starts from beginning.

nocycle: An exception will be thrown
if sequence exceeds its max_value.

Example

Following is the sequence query creating sequence in ascending order.

Example 1:

```
CREATE SEQUENCE sequence_1  
start with 1  
increment by 1  
minvalue 0  
maxvalue 100  
cycle;
```

Above query will create a sequence named *sequence_1*. Sequence will start from 1 and will be incremented by 1 having maximum value 100. Sequence will repeat itself from start value after exceeding 100.

Example2:

Following is the sequence query creating sequence in descending order

```
CREATE SEQUENCE sequence_2  
start with 100  
increment by -1  
minvalue 1  
maxvalue 100  
cycle;
```

Above query will create a sequence named *sequence_2*. Sequence will start from 100 and should be less than or equal to maximum value and will be incremented by -1 having minimum value 1.

Example to use sequence: create a table named students with columns as id and name.

```
CREATE TABLE students  
(  
ID number(10),  
NAME char(20)  
);
```

Now insert values into table

```
INSERT into students VALUES(sequence_1.nextval,'Ramesh');  
INSERT into students VALUES(sequence_1.nextval,'Suresh');
```

where *sequence_1.nextval* will insert id's in id column in a sequence as defined in sequence_1.

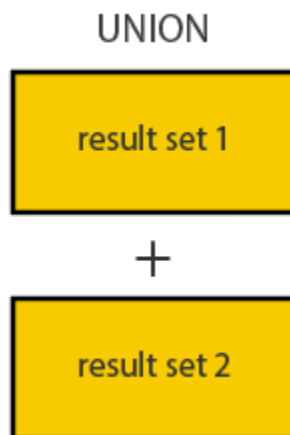
Output:

ID	NAME
1	Ramesh
2	Suresh

The SQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

- Each SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in each SELECT statement must also be in the same order



UNION Syntax

SELECT *column_name(s)* **FROM** *table1*

UNION

SELECT *column_name(s)* **FROM** *table2*;

UNION ALL Syntax

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

SELECT *column_name(s)* **FROM** *table1*

UNION ALL

SELECT *column_name(s)* **FROM** *table2*;

Difference between UNION and UNION ALL

- **UNION** removes duplicate rows.
- **UNION ALL** does **not** remove duplicate rows.

Example: If we have the *suppliers* table populated with the following records:

supplier_id	supplier_name
1000	Microsoft
2000	Oracle
3000	Apple
4000	Samsung

And the *orders* table populated with the following records:

order_id	order_date	supplier_id
1	2015-08-01	2000
2	2015-08-01	6000
3	2015-08-02	7000
4	2015-08-03	8000

And we executed the following UNION statement:

```
SELECT supplier_id
FROM suppliers
UNION
SELECT supplier_id
FROM orders
ORDER BY supplier_id;
```

We would get the following results:

supplier_id
1000
2000
3000
4000
6000
7000
8000

Union All example:

```
SELECT supplier_id
FROM suppliers
UNION ALL
SELECT supplier_id
FROM orders
ORDER BY supplier_id;
```

We would get the following results:

supplier_id
1000
2000
2000
3000
4000
6000
7000
8000

Cursors in SQL

A cursor is a database object which is used to retrieve data from result set one row at a time. The cursor can be used when the data needs to be updated row by row. It is a Temporary Memory or Temporary Work Station. It is allocated by Database Server at the Time of Performing DML operations on Table by User. In other words, a cursor can hold more than one row but can process only one row at a time. The set of rows the cursor holds is called the **active set**.

SQL Cursor Life Cycle

The following steps are involved in a SQL cursor life cycle.

1. ***Declaring Cursor***

A cursor is declared by defining the SQL statement.

2. ***Opening Cursor***

A cursor is opened for storing data retrieved from the result set.

3. ***Fetching Cursor***

When a cursor is opened, rows can be fetched from the cursor one by one or in a block to do data manipulation.

4. ***Closing Cursor***

The cursor should be closed explicitly after data manipulation.

5. ***Deallocating Cursor***

Cursors should be deallocated to delete cursor definition and release all the system resources associated with the cursor.

Need of Cursor

In relational databases, operations are made on a set of rows. For example, a SELECT statement returns a set of rows which is called a result set. Sometimes the application logic needs to work with one row at a time rather than the entire result set at once. This can be done using cursors.

In programming, we use a loop like FOR or WHILE to iterate through one item at a time, the cursor follows the same approach and might be preferred because it follows the same logic.

There are 2 types of Cursors: Implicit Cursors, and Explicit Cursors. These are explained as following below.

1. Implicit Cursors:

Implicit Cursors are also known as Default Cursors of SQL SERVER. These types of cursors are generated and used by the system during the manipulation of a DML query (INSERT, UPDATE, and DELETE). An implicit cursor is also generated by the system when a single row is selected by a SELECT command. These Cursors are allocated by SQL SERVER when the user performs DML operations.

2. Explicit Cursors:

Explicit Cursors are Created by Users whenever the user requires them. Explicit Cursors are used for Fetching data from Table in Row-By-Row Manner. This type of cursor is generated by the user using a SELECT command. An explicit cursor contains more than one row, but only one row can be processed at a time. An explicit cursor moves one by one over the records. An explicit cursor uses a pointer that holds the record of a row. After fetching a row, the cursor pointer moves to the next row.

How to create Explicit Cursor:

1. Declare Cursor Object.

Syntax : DECLARE cursor_name CURSOR FOR SELECT * FROM table_name

```
DECLARE s1 CURSOR FOR SELECT * FROM studDetails
```

Trigger on SQL

A trigger is a special type of stored procedure that automatically runs when an event occurs in the database server. DML triggers run when a user tries to modify data through a data manipulation language (DML) event. DML events are INSERT, UPDATE, or DELETE statements on a table or view. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

```
create trigger [trigger_name]
```

```
[before | after]
```

```
{insert | update | delete}
```

```
on [table_name]
```

```
[for each row]
```

```
[trigger_body]
```

Explanation of syntax:

1. `create trigger [trigger_name]`: Creates or replaces an existing trigger with the `trigger_name`.
2. `[before | after]`: This specifies when the trigger will be executed.
3. `{insert | update | delete}`: This specifies the DML operation.
4. `on [table_name]`: This specifies the name of the table associated with the trigger.
5. `[for each row]`: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
6. `[trigger_body]`: This provides the operation to be performed as trigger is fired

BEFORE and AFTER of Trigger:

BEFORE triggers run the trigger action before the triggering statement is run.

AFTER triggers run the trigger action after the triggering statement is run.

Example:

Given Student Report Database, in which student marks assessment is recorded. In such schema, create a trigger so that the total and average of specified marks is automatically inserted whenever a record is insert.

Here, as trigger will invoke before record is inserted so, BEFORE Tag can be used.

Suppose the database Schema –

```
mysql> desc Student;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| tid   | int(4)    | NO   | PRI | NULL    | auto_increment |
| name  | varchar(30) | YES  |     | NULL    |                |
| subj1 | int(2)    | YES  |     | NULL    |                |
| subj2 | int(2)    | YES  |     | NULL    |                |
| subj3 | int(2)    | YES  |     | NULL    |                |
| total | int(3)    | YES  |     | NULL    |                |
| per   | int(3)    | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```


SQL Trigger to problem statement.

```
create trigger stud_marks
before INSERT
on
Student
for each row
set Student.total = Student.subj1 + Student.subj2 + Student.subj3,
Student.per = Student.total * 60 / 100;
```

Above SQL statement will create a trigger in the student database in which whenever subjects marks are entered, before inserting this data into the database, trigger will compute those two values and insert with the entered values. i.e.,

```
mysql> insert into Student values(0, "ABCDE", 20, 20, 20, 0, 0);
Query OK, 1 row affected (0.09 sec)
```

```
mysql> select * from Student;
```

```
+-----+-----+-----+-----+-----+-----+-----+
| tid | name | subj1 | subj2 | subj3 | total | per |
+-----+-----+-----+-----+-----+-----+-----+
| 100 | ABCDE | 20 | 20 | 20 | 60 | 36 |
+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

In this way trigger can be creates and executed in the databases.

Stored Procedure

A stored procedure is a prepared SQL code that we can save, so the code can be reused over and over again.

So if we have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

A stored procedure in PL/SQL is nothing but a series of declarative SQL statements which can be stored in the database catalogue.

Stored Procedure Syntax

```
CREATE PROCEDURE procedure_name
```

```
AS
```

sql_statement

GO;

Execute a Stored Procedure

EXEC *procedure_name*;

Packages

A **package** is a container for other database objects. A package can hold other database objects such as variables, constants, cursors, exceptions, procedures, functions and sub-programs.

A package has usually two components, a **specification** and a **body**.

A package's specification declares the types (variables of the record type), memory variables, constants, exceptions, cursors, and subprograms that are available for use.

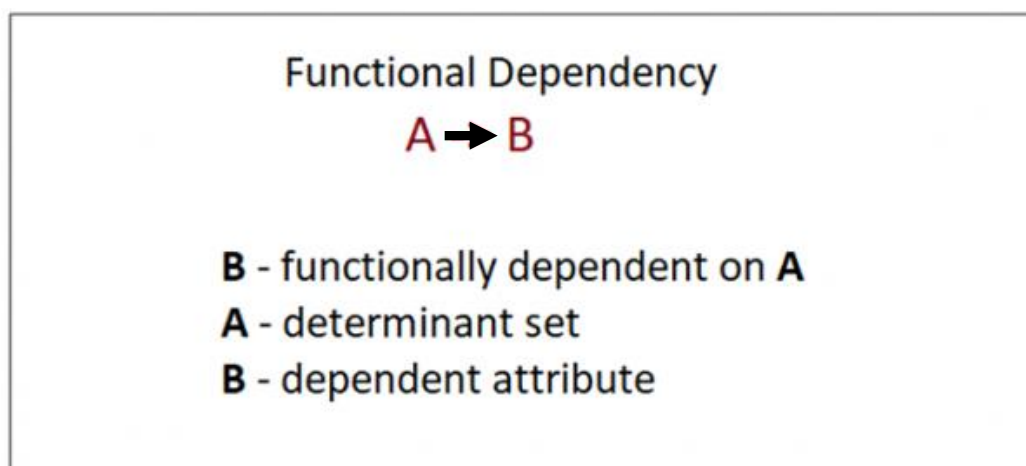
A package's body fully defines cursors, functions, and procedures and thus implements the specification.

The advantage of a **package** over a stand-alone **procedure** is that all the **procedures** and functions are loaded into memory so that when one **procedure** within the **package** calls another within the same **package** it is already loaded so this should give performance benefits if designed properly.

Unit-3

Functional Dependency

Functional Dependency (FD) determines the [relation of one attribute to another attribute](#) in a database management system (DBMS) system. Functional dependency helps us to maintain the [quality of data](#) in the database. A functional dependency is denoted by an arrow \rightarrow . The functional dependency of B on A is represented by $A \rightarrow B$. Functional Dependency plays a very important role to find the difference between [good and bad database design](#).



Example:

Employee number	Employee Name	Salary	City
101	Mohan	50000	Moradabad
201	Geeta	38000	Rampur
301	Shyam	25000	Amroha

In this example, if we know the value of Employee number, we can obtain Employee Name, city, salary, etc. By this, we can say that the city, Employee Name, and salary are functionally depended on Employee number.

{Employee number \rightarrow Employee Name, Salary, City}

Key terms

Here, are some key terms for functional dependency:

Key Terms	Description
Axiom	Axioms are a set of inference rules used to infer all the functional dependencies on a relational database.
Decomposition	It is a rule that suggests if we have a table that appears to contain two entities which are determined by the same primary key then we should consider breaking them up into two different tables.
Dependent	It is displayed on the right side of the functional dependency diagram.
Determinant	It is displayed on the left side of the functional dependency Diagram.
Union	It suggests that if two tables are separate, and the PK is the same, we should consider putting them together.

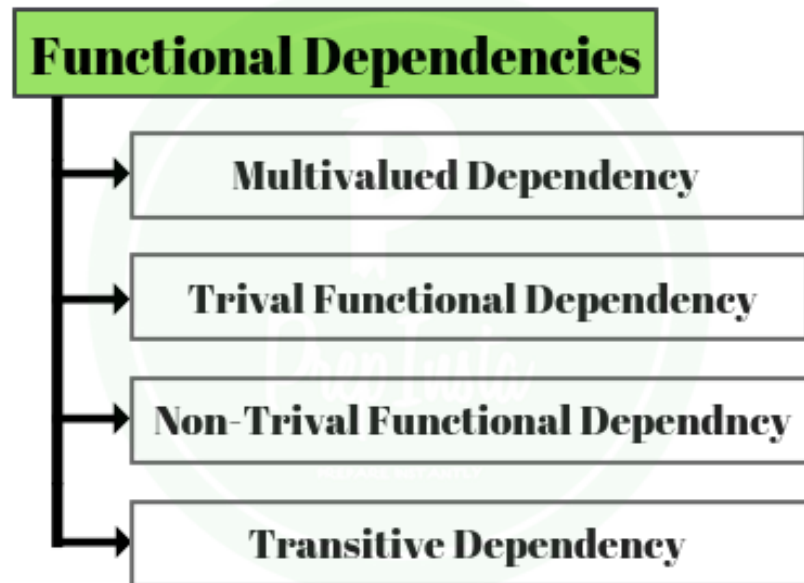
Rules of Functional Dependencies

Below given are the three most important rules for Functional Dependency:

- **Reflexive rule:** –If X is a set of attributes and Y is subset of X, then X holds a value of Y.
Ex: X= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Y= {1, 2, 3, 4, 5}
- **Augmentation rule:** When $a \rightarrow b$ holds, and c is attribute set, then $ac \rightarrow bc$ also holds.
That is adding attributes which do not change the basic dependencies.

- **Transitivity rule:** This rule is very much similar to the transitive rule in algebra if $x \rightarrow y$ holds and $y \rightarrow z$ holds, then $x \rightarrow z$ also holds.

Types of Functional Dependencies



1. Multivalued dependency in DBMS

Multivalued dependency occurs in the situation where there are multiple independent multivalued attributes in a single table.

- Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
- A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.
- It is represented by double arrow $\rightarrow\rightarrow$

For example:

$P \rightarrow\rightarrow Q$

$P \rightarrow\rightarrow R$

Example

Let us see an example

StudentName	CourseDiscipline	Activities
Amit	BCA	Singing
Amit	BCA	Dancing
Yuvraj	B.Tech	Cricket
Akash	MCA	Dancing
Akash	MCA	Cricket
Akash	MCA	Singing

In the above table, we can see Students **Amit** and **Akash** have interest in more than one activity.

This is multivalued dependency because **CourseDiscipline** of a student are independent of Activities, but are dependent on the student.

Therefore, multivalued dependency –

StudentName \twoheadrightarrow **CourseDiscipline**

StudentName \twoheadrightarrow **Activities**

2. Trivial Functional dependency:

The Trivial dependency is a set of attributes which are called a trivial if the set of attributes are included in that attribute. If a functional dependency (FD) $X \rightarrow Y$ holds, where Y is a subset of X, then it is called a trivial FD.

So, $X \rightarrow Y$ is a trivial functional dependency if Y is a subset of X.

For example:

Emp_id	Emp_name
AS555	Ram
AS811	Krishna
AS999	Aarav

Consider this table with two columns Emp_id and Emp_name.

Emp_id → {**Emp_id**, **Emp_name**} is a trivial functional dependency as Emp_id is a subset of {Emp_id, Emp_name}.

3. Non trivial functional dependency in DBMS

Functional dependency is known as a non-trivial dependency when $A \rightarrow B$ holds true where B is not a subset of A. In a relationship, if attribute B is not a subset of attribute A, then it is considered as a non-trivial dependency.

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Apple	Tim Cook	57

Example:

{**Company**} → {**CEO**} (if we know the Company, we know the CEO name)

But CEO is not a subset of Company, and hence it's non-trivial functional dependency.

4. Transitive dependency:

A transitive is a type of functional dependency which happens when it is indirectly formed by two functional dependencies.

Example:

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Alibaba	Jack Ma	54

{**Company**} → {**CEO**} (if we know the company, we know its CEO's name)

{**CEO**} → {**Age**} If we know the CEO, we know the Age

Therefore according to the rule of transitive dependency:

$\{\text{Company}\} \rightarrow \{\text{Age}\}$ should hold, that makes sense because if we know the company name, we can know his age.

Note: We need to remember that transitive dependency can only occur in a relation of three or more attributes.

Advantages of Functional Dependency

- Functional Dependency avoids data redundancy. Therefore same data do not repeat at multiple locations in that database.
- It helps us to maintain the quality of data in the database.
- It helps us to defined meanings and constraints of databases.
- It helps us to identify bad designs.
- It helps us to find the facts regarding the database design.

Normalization

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

- Eliminating redundant (useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

The inventor of the relational model Edgar Codd proposed the theory of normalization with the introduction of the First Normal Form, and he continued to extend theory with Second and Third Normal Form. Later he joined Raymond F. Boyce to develop the theory of Boyce-Codd Normal Form (BCNF).

Anomalies in DBMS

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let's take an example to understand this.

Example: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee's id, emp_name for storing employee's name, emp_address for storing employee's address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:

emp_id	emp_name	emp_address	emp_dept
101	Ashish	Delhi	D001
101	Ashish	Delhi	D002
123	Seema	Agra	D890
166	Ankit	Chennai	D900
166	Ankit	Chennai	D004

The above table is not normalized. We will see the problems that we face when a table is not normalized.

Update anomaly: In the above table we have two rows for employee Ashish as he belongs to two departments of the company. If we want to update the address of Ashish then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Ashish would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert anomaly: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

Delete anomaly: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Seema since she is assigned only to this department.

To overcome these anomalies we need to normalize the data.

Types of Normal Forms

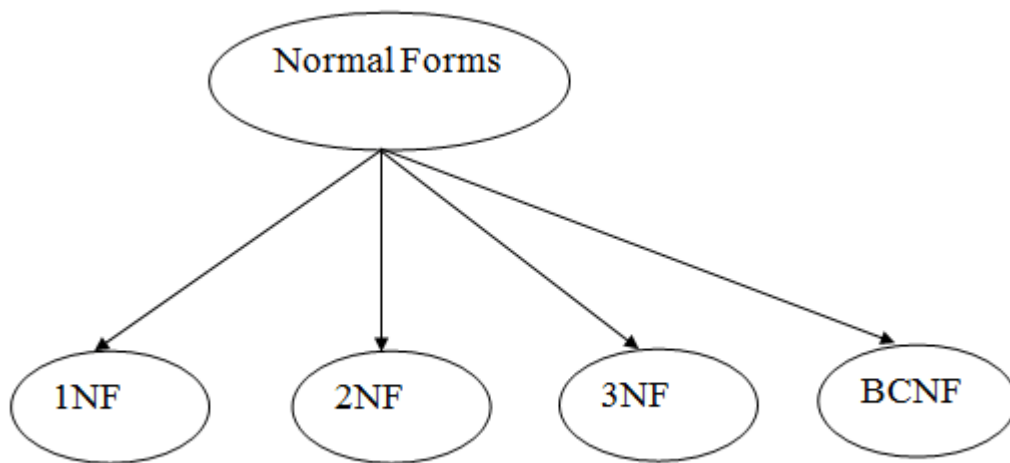
There are the four types of normal forms:

First Normal Form (1NF)

Second Normal Form (2NF)

Third Normal Form (3NF)

Bayce-Codd's Normal Form (BCNF)



Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic (single) value.
2NF	<ul style="list-style-type: none"> The table should be in the 1NF. There should be no Partial Dependency.
3NF	<ul style="list-style-type: none"> The table should be in the 2NF. There should be no transitive Dependency.
BCNF (3.5NF)	<ul style="list-style-type: none"> It should be in the 3NF. And, for any dependency $A \rightarrow B$, A should be a super key.

First Normal Form

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

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

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	Mohan	7272826385, 9064738238	UP
20	Harish	8574783832	Bihar
12	Shyam	7390372389, 8589830302	Punjab

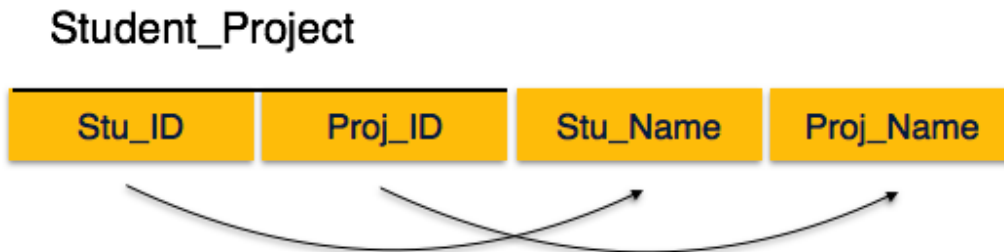
The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	Mohan	7272826385	UP
14	Mohan	9064738238	UP
20	Harish	8574783832	Bihar
12	Shyam	7390372389	Punjab
12	Shyam	8589830302	Punjab

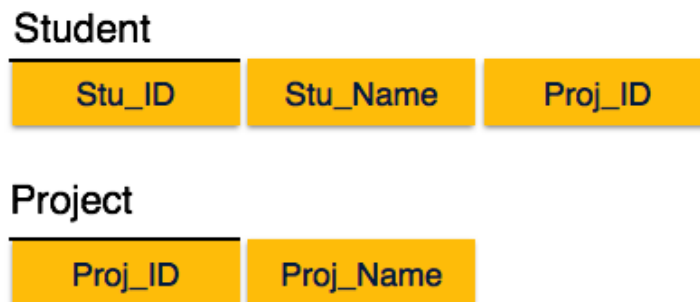
Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, there should be no Partial Dependency. All non-key attributes should be fully functional dependent on the primary key

If we follow second normal form, then every non-prime attribute should be fully functionally dependent on prime key attribute. That is, if $X \rightarrow A$ holds, then there should not be any proper subset Y of X , for which $Y \rightarrow A$ also holds true.



We see here in Student_Project relation that the prime key attributes are Stu_ID and Proj_ID. According to the rule, non-key attributes, i.e. Stu_Name and Proj_Name must be dependent upon both and not on any of the prime key attribute individually. But we find that Stu_Name can be identified by Stu_ID and Proj_Name can be identified by Proj_ID independently. This is called partial dependency, which is not allowed in Second Normal Form.



We broke the relation in two as depicted in the above picture. So there exists no partial dependency.

Third Normal Form (3NF)

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

Student_Detail



We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that Zip can be identified by Stu_ID and as well as city can be identified by Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally, $\text{Stu_ID} \rightarrow \text{Zip} \rightarrow \text{City}$, so there exists transitive dependency.

To bring this relation into third normal form, we break the relation into two relations as follows-

Student_Detail



ZipCodes



Boyce Codd normal form (BCNF) (3.5NF)

BCNF stands for Boyce-Codd normal form and was made by R.F Boyce and E.F Codd in 1947. BCNF is the advance version of 3NF. It is stricter than 3NF. A functional dependency is said to be in BCNF if these properties hold:

- It should already be in 3NF.
- For a functional dependency say $P \rightarrow Q$, P should be a super key.

Example:

Below we have a college enrolment table with columns student_id, subject and professor.

As we can see, we have also added some sample data to the table.

student_id	student_name	Subject	Professor
101	Ram	Java	P.Java1
101	Ram	C++	P.Cpp
102	Mohan	Java	P.Java2
103	Suresh	C#	P.Chash
104	Hitesh	Java	P.Java

In the table above:

- One student can enrol for multiple subjects. For example, student with **student_id** 101 **student_name** Ram, has opted for subjects - Java & C++
- For each subject, a professor is assigned to the student.
- And, there can be multiple professors teaching one subject like we have for Java.

One more important point to note here is, one professor teaches only one subject, but one subject may have two different professors.

Hence, there is a dependency between **subject** and **professor** here, where **subject** depends on the professor name.

This table satisfies the **1st Normal form** because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain.

This table also satisfies the **2nd Normal Form** as there is no **Partial Dependency**.

And, there is no **Transitive Dependency**, hence the table also satisfies the **3rd Normal Form**.

But this table is not in **Boyce-Codd Normal Form**.

Why this table is not in BCNF?

In the table above, **student_id**, **subject** are keys, or we can say that **student_id** and **subject** are **prime attribute** in this table.

But, there is one more dependency, **professor** → **subject**.

And while **subject** is a **prime attribute**, **professor** is a **non-prime attribute**, which is not allowed by BCNF.

How to satisfy BCNF?

To make this relation(table) satisfy BCNF, we will decompose this table into two tables, **student** table and **professor** table.

Below we have the structure for both the tables.

Student Table

student_id	p_id
101	1
101	2
and so on...	

And, **Professor Table**

p_id	Professor	Subject
1	P.Java	Java
2	P.Cpp	C++
and so on...		

And now, this relation satisfy Boyce-Codd Normal Form.

$P_Id \rightarrow \{professor, subject\}$

Decomposition

Decomposition in DBMS removes redundancy, anomalies and inconsistencies from a database by dividing the table into multiple tables. Decomposition is the process of breaking down in parts or elements. It replaces a relation with a collection of smaller relations. It breaks the table into multiple tables in a database. It should always be lossless, because it confirms that the information in the original relation can be accurately reconstructed based on the decomposed relations.

- When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.

- In a database, it breaks the table into multiple tables.
- If the relation has no proper decomposition, then it may lead to problems like loss of information.
- Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

The following are the types of Decomposition –

1. Lossless Decomposition
2. Lossy Decomposition

1. Lossless Decomposition

- If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.
- The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.
- The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

Let us see an example –

<EmpInfo>

Emp_ID	Emp_Name	Emp_Age	Emp_Location	Dept_ID	Dept_Name
E001	Jacob	29	Alabama	Dpt1	Operations
E002	Henry	32	Alabama	Dpt2	HR
E003	Tom	22	Texas	Dpt3	Finance

Decompose the above table into two tables:

<EmpDetails>

Emp_ID	Emp_Name	Emp_Age	Emp_Location
E001	Jacob	29	Alabama
E002	Henry	32	Alabama
E003	Tom	22	Texas

<DeptDetails>

Dept_ID	Emp_ID	Dept_Name
Dpt1	E001	Operations
Dpt2	E002	HR
Dpt3	E003	Finance

Now, Natural Join is applied on the above two tables –

The result will be –

Emp_ID	Emp_Name	Emp_Age	Emp_Location	Dept_ID	Dept_Name
E001	Jacob	29	Alabama	Dpt1	Operations
E002	Henry	32	Alabama	Dpt2	HR
E003	Tom	22	Texas	Dpt3	Finance

Therefore, the above relation had lossless decomposition i.e. no loss of information.

2. Lossy Decomposition

- If the information is lost from the relation that is decomposed, then the decomposition will be lossy.
- The lossy decomposition does not guarantee that the join of relations will result in the same relation as it was decomposed.
- The relation is said to be lossy decomposition if natural joins of all the decomposition did not give the original relation.

"The decomposition of relation R into R1 and R2 is **lossy** when the join of R1 and R2 does not yield the same relation as in R."

One of the disadvantages of decomposition into two or more relational schemes (or tables) is that some information is lost during retrieval of original relation or table.

Let us see an example –

<EmpInfo>

Emp_ID	Emp_Name	Emp_Age	Emp_Location	Dept_ID	Dept_Name
E001	Jacob	29	Alabama	Dpt1	Operations
E002	Henry	32	Alabama	Dpt2	HR
E003	Tom	22	Texas	Dpt3	Finance

Decompose the above table into two tables –

<EmpDetails>

Emp_ID	Emp_Name	Emp_Age	Emp_Location
E001	Jacob	29	Alabama
E002	Henry	32	Alabama
E003	Tom	22	Texas

<DeptDetails>

Dept_ID	Dept_Name
Dpt1	Operations
Dpt2	HR
Dpt3	Finance

Now, you won't be able to join the above tables, since **Emp_ID** isn't part of the **DeptDetails** relation.

Therefore, the above relation has lossy decomposition.

Inclusion Dependency in DBMS

Inclusion dependencies (IND) support an essential semantics of the standard relational data model. It is a statement in which a few of the columns of a relation are in the other columns. Typically, inclusion dependency has a very insignificant influence on the design of a database. For example, a foreign key is inclusion dependency. Inclusion Dependency (IND). IND is the rule among different schemas.

Multivalued dependency and join dependency can be used to guide database design although they both are less common than functional dependencies.

Inclusion dependencies are quite common. They typically show little influence on designing of the database.

Transaction Processing In DBMS

Transaction

A transaction can be defined as a group of operations to perform a logical unit of work. A single task is the minimum processing unit which cannot be divided further.

- The transaction is a set of logically related operation. It contains a group of tasks.
- A transaction is an action or series of actions. It is performed by a single user to perform operations for accessing the contents of the database.

For example, we are transferring money from our bank account to our friend's account; the set of operations would be like this:

Simple Transaction Example

1. Read our account balance
2. Deduct the amount from our balance
3. Write the remaining balance to our account
4. Read our friend's account balance
5. Add the amount to his account balance
6. Write the new updated balance to his account

This whole set of operations can be called a transaction.

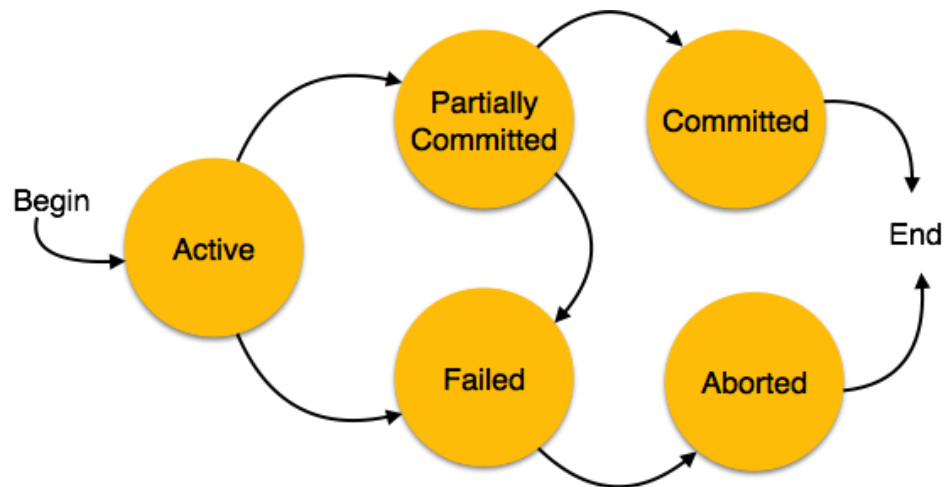
In DBMS, we write the above 6 steps transaction like this:

Lets say your account is A and your friend's account is B, you are transferring 10000 from A to B, the steps of the transaction are:

1. R(A);
2. A = A - 10000;
3. W(A);
4. R(B);
5. B = B + 10000;
6. W(B);

States of Transactions

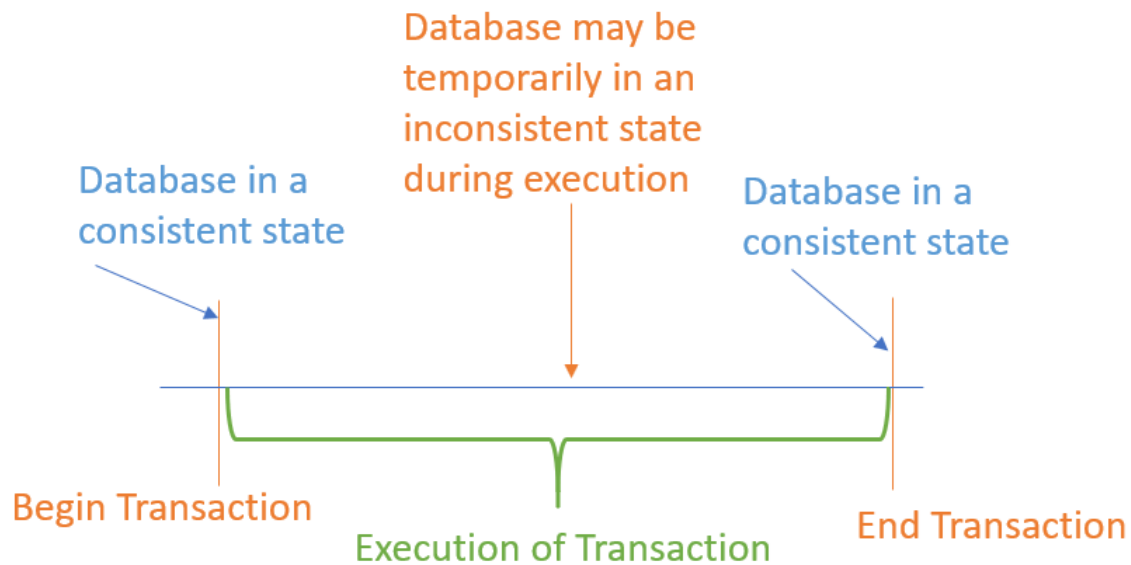
A transaction in a database can be in one of the following states –



- **Active** – In this state, the transaction is being executed. This is the initial state of every transaction.
- **Partially Committed** – When a transaction executes its final operation, it is said to be in a partially committed state.
- **Failed** – A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.
- **Aborted** – If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts –
 - Re-start the transaction
 - Kill the transaction
- **Committed** – If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.

Transaction failure in between the operations

The main problem that can happen during a transaction is that the transaction can fail before finishing all the operations in the set. This can happen due to power failure, system crash etc. This is a serious problem that can leave database in an inconsistent state.



Assume that transaction fail after third operation (see the example above) then the amount would be deducted from our account but our friend will not receive it.

To solve this problem, we have the following two operations

Commit: If all the operations in a transaction are completed successfully then commit those changes to the database permanently.

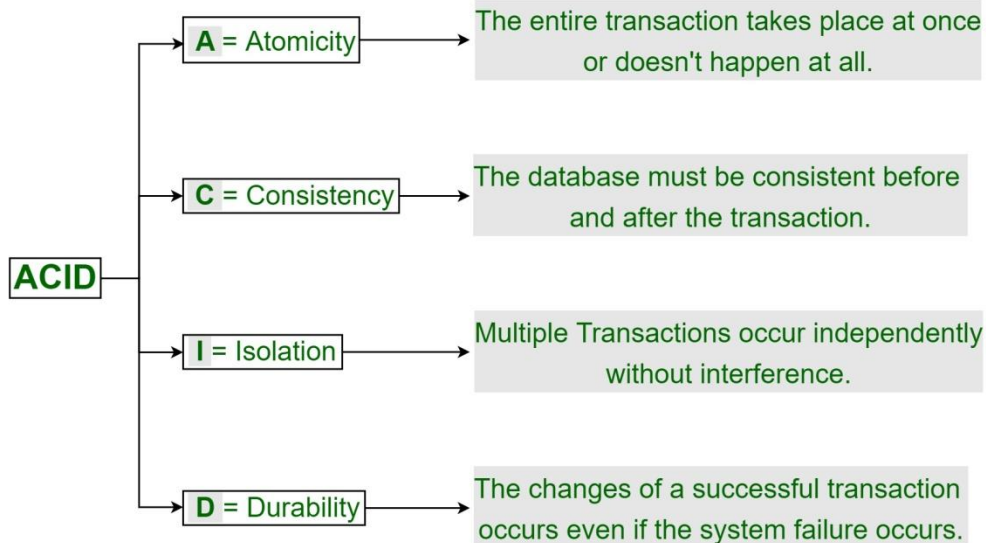
Rollback: If any of the operation fails then rollback all the changes done by previous operations. Even though these operations can help us avoiding several issues that may arise during transaction but they are not sufficient when two transactions are running concurrently. To handle those problems we need to understand database **ACID properties**.

ACID Properties.

A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain Atomicity, Consistency, Isolation, and Durability – commonly known as ACID properties – in order to ensure accuracy, completeness, and data integrity.

ACID (atomicity, consistency, isolation, durability) is a set of properties of database transactions intended to guarantee data validity despite errors, power failures, and other mishaps.

ACID Properties in DBMS

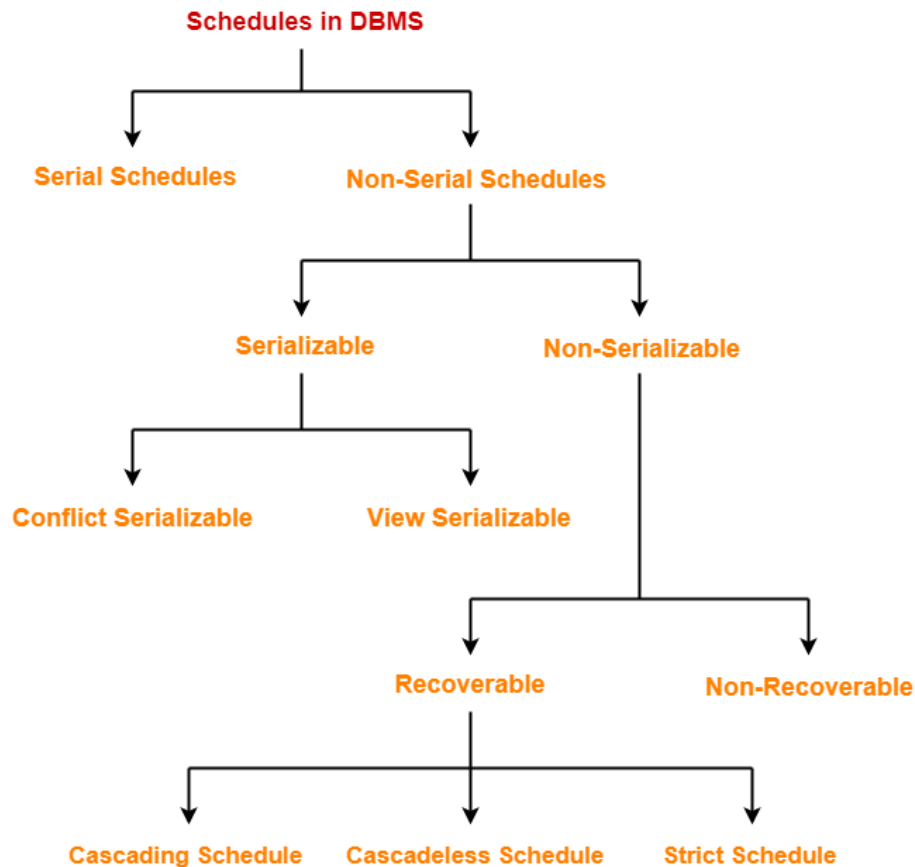


- **Atomicity** –This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. We either execute it entirely or do not execute it at all. There cannot be partial execution.
 - This property ensures that either the transaction occurs completely or it does not occur at all.
 - In other words, it ensures that no transaction occurs partially.
 - That is why, it is also referred to as “**All or nothing rule**“.
 - It is the responsibility of Transaction Control Manager to ensure atomicity of the transactions.
- **Consistency** –The database must remain in a consistent state after any transaction. No transaction should have any undesirable effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
 - This property ensures that integrity constraints are maintained.
 - In other words, it ensures that the database remains consistent before and after the transaction.

- It is the responsibility of DBMS and application programmer to ensure consistency of the database.
- **Isolation** – In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.
 - This property ensures that multiple transactions can occur simultaneously without causing any inconsistency.
 - During execution, each transaction feels as if it is getting executed alone in the system.
 - A transaction does not realize that there are other transactions as well getting executed parallelly.
 - Changes made by a transaction becomes visible to other transactions only after they are written in the memory.
 - It is the responsibility of concurrency control manager to ensure isolation for all the transactions.
- **Durability** – The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.
 - This property ensures that all the changes made by a transaction after its successful execution has written successfully to the disk.
 - It also ensures that these changes exist permanently and are never lost even if there occurs a failure of any kind.
 - It is the responsibility of recovery manager to ensure durability in the database.

Schedules in DBMS

Schedule – A schedule is the order in which the operations of multiple transactions appear for execution. A schedule can have many transactions in it, each comprising of a number of instructions/tasks. There are two types of schedule in DBMS as:-



- 1. Serial Schedule** –In serial schedules, all the transactions execute serially one after the other. When one transaction executes, no other transaction is allowed to execute. Transactions are ordered one after the other. This type of schedule is called a serial schedule, as transactions are executed in a serial manner. Serial schedules are always consistent.

Ex1:

Transaction T1	Transaction T2
R (A)	
W (A)	
R (B)	
W (B)	
Commit	
	R (A)
	W (B)
	Commit

In this schedule,

- There are two transactions T1 and T2 executing serially one after the other.
- Transaction T1 executes first.
- After T1 completes its execution, transaction T2 executes.
- So, this schedule is an example of a **Serial Schedule**.

Ex2:

Transaction T1	Transaction T2
	R (A)
	W (B)
	Commit
R (A)	
W (A)	
R (B)	
W (B)	
Commit	

In this schedule,

- There are two transactions T1 and T2 executing serially one after the other.
- Transaction T2 executes first.
- After T2 completes its execution, transaction T1 executes.
- So, this schedule is an example of a **Serial Schedule**.

2. Non-serial schedules - In non-serial schedules, multiple transactions execute concurrently.

Operations of all the transactions are interleaved or mixed with each other. Non-serial schedules are not always consistent.

Ex1:

Transaction T1	Transaction T2
R (A)	
W (B)	
	R (A)
R (B)	
W (B)	
Commit	
	R (B)
	Commit

In this schedule,

- There are two transactions T1 and T2 executing concurrently.
- The operations of T1 and T2 are interleaved.
- So, this schedule is an example of a **Non-Serial Schedule**.

Serializability in DBMS-

When multiple transactions are running concurrently (non-serial Schedule) then there is a possibility that either the database may be left in an inconsistent state or instructions of one transaction are interleaved with some other transaction. Some non-serial schedules may lead to inconsistency of the database.

Serializability is a concept that helps us to check which schedules are serializable. A serializable schedule is the one that always leaves the database in consistent state.

It helps to identify which non-serial schedules are correct and will maintain the consistency of the database.

Serializability of schedules

A serial schedule is always a serializable schedule because any transaction only starts its execution when another transaction has already completed its execution. However, a non-serial schedule of transactions needs to be checked for Serializability.

If a schedule of concurrent 'n' transactions can be converted into an equivalent serial schedule. Then we can say that the schedule is serializable. And this property is known as serializability.

Testing for Serializability

To test the serializability of a schedule, we can use the serialization graph. Consider a schedule S. We construct a direct graph, called a precedence graph, from S. This graph consists of a pair $G = (V, E)$, where V is a set of vertices and E is a set of edges. The set of vertices consists of all the transactions participating in the schedule. The set of edges consists of all edges $T_i \rightarrow T_j$ for which one of three conditions holds:

1. T_i executes write (Q) before T_j executes read (Q). (**Write(Q) - Read(Q)**)
2. T_i executes read (Q) before T_j executes write (Q). (**Read(Q) - Write(Q)**)
3. T_i executes write (Q) before T_j executes write (Q). (**Write(Q) - Write(Q)**)

Precedence graph for Schedule S

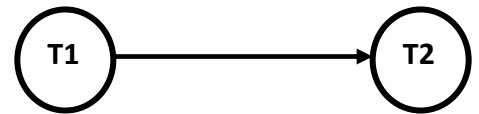


Schedule S Precedence Graph

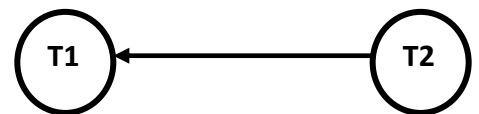
- If a precedence graph contains a single edge $T_i \rightarrow T_j$, then all the instructions of T_i are executed before the first instruction of T_j is executed.
- If a precedence graph for schedule S contains a cycle, then S is non-serializable. If the precedence graph has no cycle, then S is known as serializable.

Example1: Serialization graphs (precedence graph) for serial schedules S having two transactions T1 & T2-

Transaction T1	Transaction T2
R (A)	
W (A)	
R (B)	
W (B)	
Commit	
	R (A)
	W (B)
	Commit

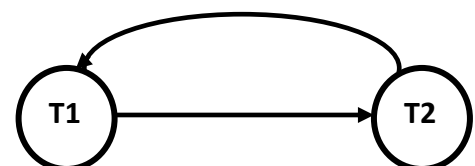


Transaction T1	Transaction T2
	R (A)
	W (B)
	Commit
R (A)	
W (A)	
R (B)	
W (B)	
Commit	

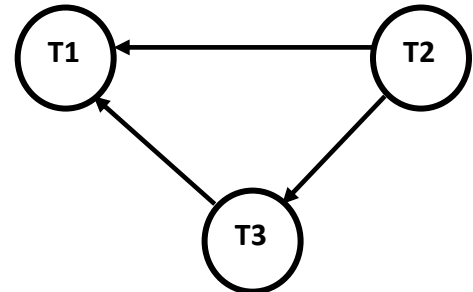


Example2: Serialization graphs (precedence graphs) for non-serial schedule S having two transactions T1 & T2-

Transaction T1	Transaction T2
R(A)	
	R(A)
	W(A)
W(A)	

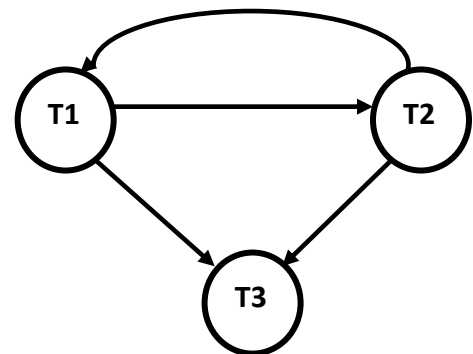


Transaction T1	Transaction T2	Transaction T3
R(A)		R(B)
		R(A)
	R(B)	
	R(C)	
	W(C)	W(B)
R(C)		
W(A)		
W(C)		



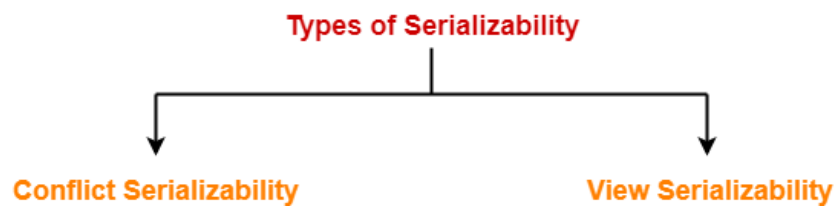
Example2: Serialization graphs (precedence graphs) for a non-serial schedule S having three transactions T1, T2 & T3-

Transaction T1	Transaction T2	Transaction T3
R(A)		
	W(A)	
W(A)		W(A)



Types of Serializability-

Serializability is mainly of two types-



1. Conflict Serializability
2. View Serializability

Conflict Serializability-

If a given non-serial schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called as a **conflict serializable schedule**.

Conflicting Operations-

Two operations are called as **conflicting operations** if all the following conditions hold true for them-

- Both the operations belong to different transactions
- Both the operations are on the same data item
- At least one of the two operations is a write operation

Example-

Consider the following schedule-

Transaction T1	Transaction T2
R1 (A)	
W1 (A)	
	R2 (A)
R1 (B)	

In this schedule,

- W1 (A) and R2 (A) are called as conflicting operations.
- This is because all the above conditions hold true for them.

Example

Non-serial schedule

T1	T2
Read(A) Write(A)	
	Read(A) Write(A)
Read(B) Write(B)	
	Read(B) Write(B)

Schedule S1

Serial Schedule

T1	T2
Read(A) Write(A) Read(B) Write(B)	
	Read(A) Write(A) Read(B) Write(B)

Schedule S2

Schedule S2 is a serial schedule because, in this, all operations of T1 are performed before starting any operation of T2. Schedule S1 can be transformed into a serial schedule by swapping non-conflicting operations of S1.

After swapping of non-conflict operations, the schedule S1 becomes:

T1	T2
Read(A) Write(A) Read(B) Write(B)	
	Read(A) Write(A) Read(B) Write(B)

Since, S1 is conflict serializable.

Example

Let us consider the following schedule and see if it is serializable.

Transaction T1	Transaction T2
	R(X)
R(X)	
	R(Y)
	W(Y)
R(Y)	
W(X)	

Now, let us figure out if the above schedule is serializable.

1. Swapping R(X) of T1 and R(Y) of T2.
2. Swapping R(X) of T1 and W(Y) of T2.

Transaction T1	Transaction T2
	R(X)
	R(Y)
	W(Y)
R(X)	
R(Y)	
W(X)	

Thus, after changing the conflicting operations in the initial schedule we get a serial schedule.
Hence, this schedule is serializable.

Another Example for Conflict Serializability –

Let us consider the following transaction schedule and test it for Conflict Serializability.

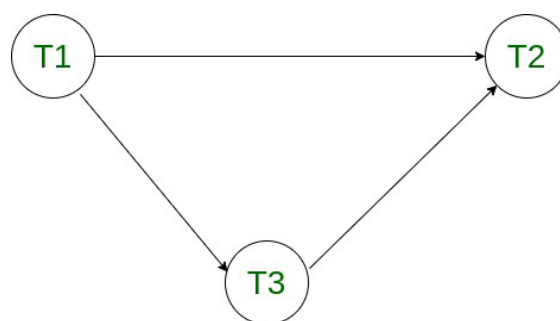
Transaction T1	Transaction T2	Transaction T3
	R(X)	
		R(X)
W(Y)		
	W(X)	
		R(Y)
	W(Y)	

Now, we will list all the conflicting operations. Further, we will determine whether the schedule is conflict serializable using Precedence Graph.

Two operations are said to be conflicting if they belong to different transactions, operate on same data and at least one of them is a read operation.

1. R3(X) and W2(X) [T3 → T2]
2. W1(Y) and R3(Y) [T1 → T3]
3. W1(Y) and W2(Y) [T1 → T2]
4. R3(Y) and W2(Y) [T3 → T2]

Constructing the precedence graph, we see there are no cycles in the graph. Therefore, the schedule is Conflict Serializable.



View Serializability

View Serializability is a process to find out that a given schedule is view serializable or not.

To check whether a given schedule is view serializable, we need to check whether the given schedule is **View Equivalent** to its serial schedule. If a schedule is conflict serializable, then it is surely view serializable. If a given schedule S is not conflict serializable, it may or may not be view serializable.

View Equivalent

Lets learn how to check whether the two schedules are view equivalent.

Two schedules S1 and S2 are said to be view equivalent, if they satisfy all the following conditions:

1. Initial Read: Initial read of each data item in transactions must match in both schedules. For example, if transaction T1 reads a data item X before transaction T2 in schedule S1 then in schedule S2, T1 should read X before T2.

Read vs Initial Read: Here initial read means the first read operation on a data item, for example, a data item X can be read multiple times in a schedule but the first read operation on X is called the initial read. This will be more clear once we will get to the example in the next section of this same article.

Thumb Rule

“Initial readers must be same for all the data items”.

2. Final Write: Final write operations on each data item must match in both the schedules. For example, a data item X is last written by Transaction T1 in schedule S1 then in S2, the last write operation on X should be performed by the transaction T1.

Thumb Rule

“Final writers must be same for all the data items”.

3. Update Read: If in schedule S1, the transaction T1 is reading a data item updated by T2 then in schedule S2, T1 should read the value after the write operation of T2 on same data item. For example, In schedule S1, T1 performs a read operation on X after the write operation on X by T2 then in S2, T1 should read the X after T2 performs write on X.

Thumb Rule

“Write-read sequence must be same.”

View Serializable

If a schedule is view equivalent to its serial schedule then the given schedule is said to be View Serializable. Lets take an example.

View Serializable Example

Non-Serial		Serial	
S1		S2	
T1	T2	T1	T2
R(X)		R(X)	
W(X)		W(X)	
	R(X)	R(Y)	
	W(X)	W(Y)	
R(Y)			R(X)
W(Y)			W(X)
	R(Y)		R(Y)
	W(Y)		W(Y)

Lets check the three conditions of view serializability:

Initial Read

In schedule S1, transaction T1 first reads the data item X. In S2 also transaction T1 first reads the data item X.

Lets check for Y. In schedule S1, transaction T1 first reads the data item Y. In S2 also the first read operation on Y is performed by T1.

We checked for both data items X & Y and the **initial read** condition is satisfied in S1 & S2.

Final Write

In schedule S1, the final write operation on X is done by transaction T2. In S2 also transaction T2 performs the final write on X.

Lets check for Y. In schedule S1, the final write operation on Y is done by transaction T2. In schedule S2, final write on Y is done by T2.

We checked for both data items X & Y and the **final write** condition is satisfied in S1 & S2.

Update Read

In S1, transaction T2 reads the value of X, written by T1. In S2, the same transaction T2 reads the X after it is written by T1.

In S1, transaction T2 reads the value of Y, written by T1. In S2, the same transaction T2 reads the value of Y after it is updated by T1.

The update read condition is also satisfied for both the schedules.

Result: Since all the three conditions that checks whether the two schedules are view equivalent are satisfied in this example, which means S1 and S2 are view equivalent. Also, as we know that the schedule S2 is the serial schedule of S1, thus we can say that the schedule S1 is view serializable schedule.

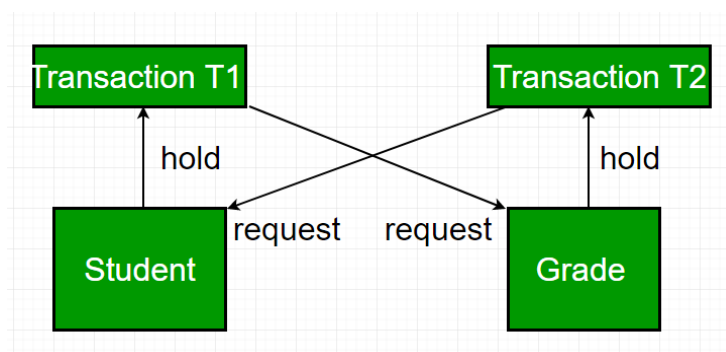
Transaction failures

The transaction failure occurs when it fails to execute some of its operations or when it reaches a point from where it can't go any further. Sometimes a transaction may not execute completely due to a [software issue](#), [system crash](#) or [hardware failure](#). In that case, the failed transaction has to be rollback.

If a few transaction or process is hurt, then this is called as transaction failure.

Reasons for a transaction failure could be -

1. **Logical errors:** If a transaction cannot complete due to some code error or an internal error condition, then the logical error occurs. For example any value divided by zero ($n/0$), or any integer value having range from 1 to 100, and if we put value out of range like -50,105 then due to such type of logical error transaction fails.
2. **Concurrency Control Enforcement:** It occurs where the DBMS itself terminates an active transaction because the database system is not able to execute it. **For example,** The system aborts an active transaction, in case of deadlock or resource unavailability.



3. **System Failure/Crash** -System failure can occur due to many numbers of reasons as power failure, network failure or other hardware or software failure.
4. **Local Errors and Exception-** Local errors occurs when a transaction begins and it between systems realize an exception. For example- we are going to withdraw Rs.5000 from our account and we are having just Rs 4000 in our account, then transaction will fail due to local error or exception error.
5. **Hard-Disk Failure-** Hard drives are generally used to store large files, back up data, and to save important data to keep them safe and secure. A hard disk drive failure occurs when a hard disk drive malfunctions and the stored information cannot be accessed with a properly configured computer. There are a number of causes for hard drives to fail including: manufacturing error, heat, water damage, power issues and mishaps. Other reasons of disk failure occur due to the formation of bad sectors, disk head crash, and unreachability to the disk.
6. **Physical problem or catastrophe-** If whole system destroys due to some external issues like natural disaster as earth-quake or flood then transaction will fail.

Recovery from transaction failures

Here we will discuss two method of transaction recovery

1. Log-Based Recovery

- The log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.
- If any operation is performed on the database, then it will be recorded in the log.
- But the process of storing the logs should be done before the actual transaction is applied in the database.
- And logs may or may not contain the record of reading of data item. It is so because reading the data item does not affect the consistency of the database and is nowhere helpful in recovery mechanism.

An update log record represented as: $\langle Ti, Xj, V1, V2 \rangle$ has these fields:

1. **Transaction identifier:** Unique Identifier of the transaction that performed the write operation.
2. **Data item:** Unique identifier of the data item written.

3. **Old value:** Value of data item prior to write.
4. **New value:** Value of data item after write operation.

Other type of log records is:

- | | | |
|--------------------------|---|--|
| <Ti start> | : | It contains information about when a transaction Ti starts. |
| <Ti commit> | : | It contains information about when a transaction Ti commits. |
| <Ti abort> | : | It contains information about when a transaction Ti aborts |

Undo and Redo Operations –

Because all database modifications must be preceded by creation of log record, the system has available both the old value prior to modification of data item and new value that is to be written for data item. This allows system to perform redo and undo operations as appropriate:

1. **Undo:** using a log record sets the data item specified in log record to old value.
2. **Redo:** using a log record sets the data item specified in log record to new value.

After a system crash has occurred, the system consults the log to determine which transactions need to be redone and which need to be undone.

1. Transaction Ti needs to be undone if the log contains the record <Ti start> but does not contain either the record <Ti commit> or the record <Ti abort>.
2. Transaction Ti needs to be redone if log contains record <Ti start> and either the record <Ti commit> or the record <Ti abort>.

Example

```
<To start>
<To A, 1000, 950>
<To B, 2000, 2050>
<To commit>
<T1 start>
<T1 C, 700, 600>
```

We consider an example of banking system taken earlier for transaction To and T1 such that To is followed by T1. If the system crash occurs just after the log record and during recovery we do redo (To) and undo (T1) as we have both < To start > and <To commit> in the log record. But we do not have <T1 commit> with <T1 start> in log record. Undo (T1) should be done first then redo (To) should be done.

2. Checkpoints-Based Recovery

When a system crash occurs, user must consult the log. In principle, that needs to search the entire log to determine this information. There are two major difficulties with this approach:

1. The search process is time-consuming.
2. Most of the transactions that need to be redone have already written their updates into the database. Although redoing them will cause no harm, it will cause recovery to take longer.

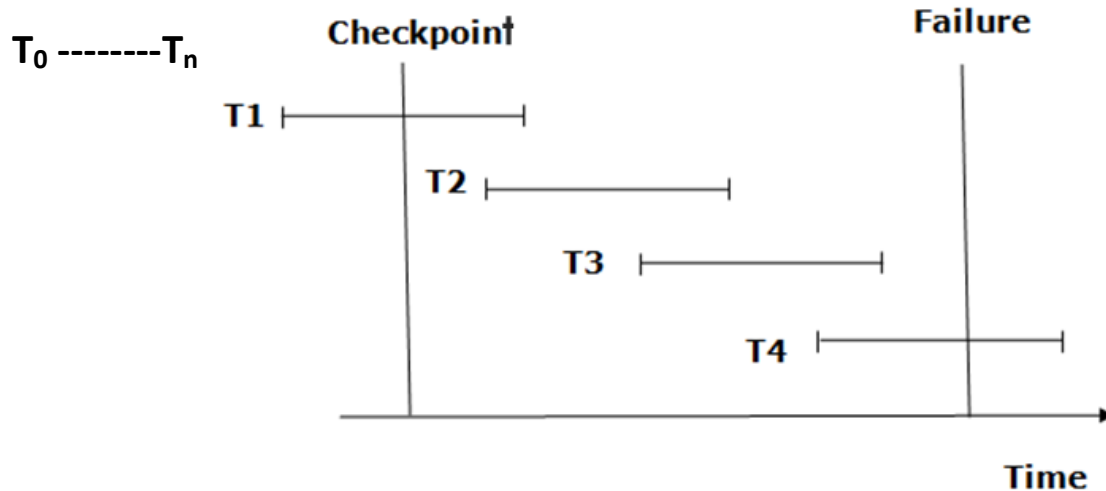
To reduce these types of overhead, user introduce checkpoints.

- The checkpoint is a type of mechanism where all the previous logs are removed from the system and permanently stored in the storage disk.
- The checkpoint is like a bookmark. While the execution of the transaction, such checkpoints are marked.



- When it reaches to the checkpoint, then the transaction will be updated into the database, and till that point, the entire log file will be removed from the file. Then the log file is updated with the new step of transaction till next checkpoint and so on.
- The checkpoint is used to declare a point before which the DBMS was in the consistent state, and all transactions were committed.

In the following manner, a recovery system recovers the database from this failure:



- The recovery system reads log files from the end to start. It reads log files from T4 to T1.
- Recovery system maintains two lists, a redo-list, and an undo-list.
- The transaction is put into redo state if the recovery system sees a log with $\langle T_n, \text{Start} \rangle$ and $\langle T_n, \text{Commit} \rangle$ or just $\langle T_n, \text{Commit} \rangle$. In the redo-list and their previous list, all the transactions are removed and then redone before saving their logs.
- **For example:** In the log file, transaction T2 and T3 will have $\langle T_n, \text{Start} \rangle$ and $\langle T_n, \text{Commit} \rangle$. The T1 transaction will have only $\langle T_n, \text{commit} \rangle$ in the log file. That's why the transaction is committed after the checkpoint is crossed. Hence it puts T1, T2 and T3 transaction into redo list.
- The transaction is put into undo state if the recovery system sees a log with $\langle T_n, \text{Start} \rangle$ but no commit or abort log found. In the undo-list, all the transactions are undone, and their logs are removed.
- **For example:** Transaction T4 will have $\langle T_n, \text{Start} \rangle$. So T4 will be put into undo list since this transaction is not yet complete and failed in the middle of it.

Unit-4 DBMS

Concurrency control

When multiple transactions are running simultaneously then there are chances of a conflict to occur which can leave database to an inconsistent state. To handle these conflicts we need concurrency control in DBMS, which allows transactions to run simultaneously but handles them in such a way so that the integrity of data remains intact.

Concurrency Control in Database Management System is a procedure of managing simultaneous operations without conflicting with each other.

Concurrent access is quite easy if all users are just reading data. But database have a mix of READ and WRITE operations and hence the concurrency is a challenge.

DBMS Concurrency Control is used to address such conflicts, which mostly occur with a multi-user system. Therefore, Concurrency Control is the most important element for proper functioning of a Database Management System where two or more database transactions are executed simultaneously, which require access to the same data.

Concurrency Control Techniques

Following are the Concurrency Control techniques in DBMS:

- Lock Techniques for concurrency control
- Time stamping protocols for concurrency control
- Validation-Based Protocols

1. Lock Techniques for concurrency control

Lock Based Protocols in DBMS is a mechanism in which a transaction cannot Read or Write the data until it acquires an appropriate lock.

Locks are of two kinds –

Shared/Exclusive Locking Protocol: This type of locking mechanism separates the locks in DBMS based on their uses. If a lock is acquired on a data item to perform a read-only operation, it is called a shared lock and if a lock is acquired on a data item to perform a write operation, it is called an exclusive lock.

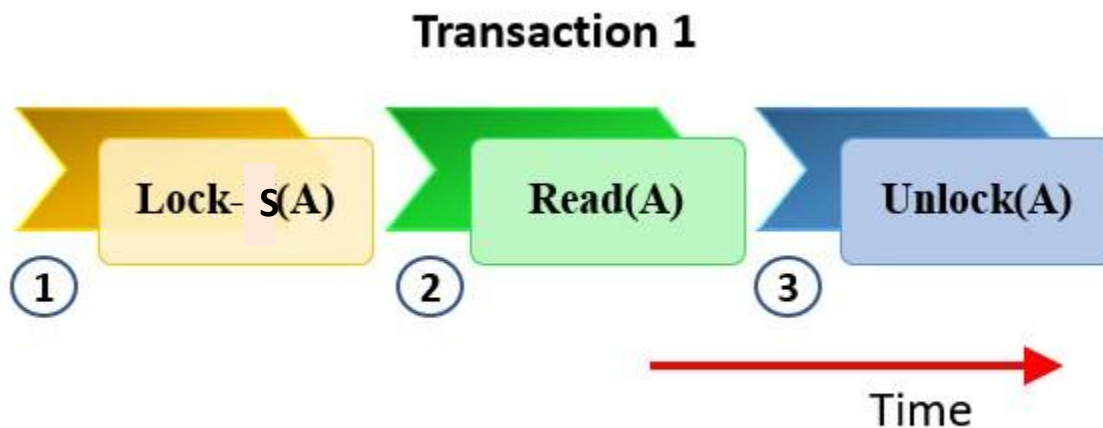
1. Shared Lock (S):

A shared lock is also called a Read-only lock. With the shared lock, the data item can be shared between transactions. This is because we will never have permission to update data on the data item.

For example, consider a case where two transactions are reading the account balance of a person. The database will let them read by placing a shared lock. However, if another transaction wants to update that account's balance, shared lock prevent it until the reading process is over.

Consider a transaction (T1) which requires only reading the data item value A. The following steps take place when lock protocol is applied to this transaction

1. T1 will acquire an shared lock on the data item A
2. Read the current value of data item A
3. Once the transaction is completed, the data item will be unlocked.



2. Exclusive Lock (X):

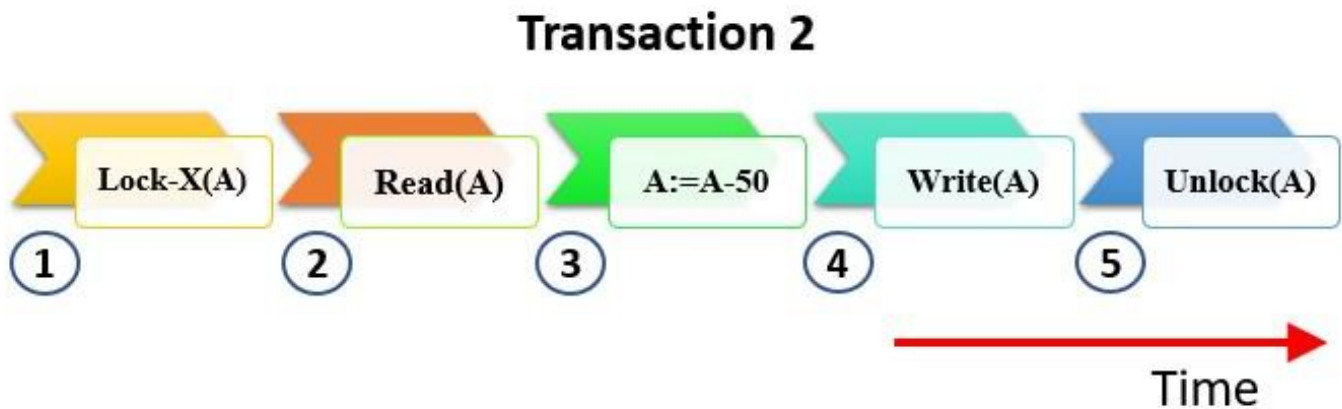
With the Exclusive Lock, a data item can be read as well as written. This is exclusive and can't be held concurrently on the same data item. X-lock is requested using lock-x instruction. Transactions may unlock the data item after finishing the 'write' operation.

For example, when a transaction needs to update the account balance of a person. We can allow this transaction by placing X lock on it. Therefore, when the second transaction wants to read or write, exclusive lock prevent this operation.

Consider a transaction (T2) which requires updating the data item value A. The following steps take place when lock protocol is applied to this transaction

1. T2 will acquire an exclusive lock on the data item A
2. Read the current value of data item A
3. Modify the data item as required. In the example illustrated, a value of 50 is subtracted from the data item A

4. Write the updated value of the data item
5. Once the transaction is completed, the data item will be unlocked.



Lock Compatibility Matrix –

→ Request

		S	X
↓ Grant	S	✓	X
	X	X	X

The figure illustrates that when two transactions are involved, and both attempt to only read a given data item, it is permitted and no conflict arises, but when one transaction attempts to write the data item and another one tries to read or write at the same time, conflict occurs resulting in a denied interaction.

Problems associated with simple locking:

Deadlock

Deadlock refers to a specific situation where two or more processes are waiting for each other to release a resource or more than two processes are waiting for the resource in a circular chain.

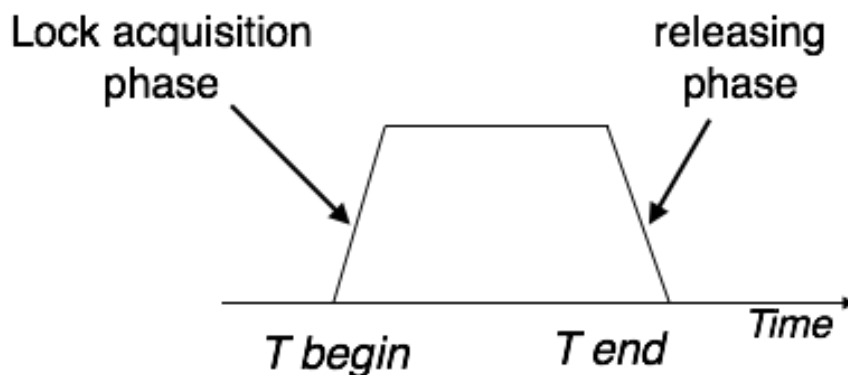
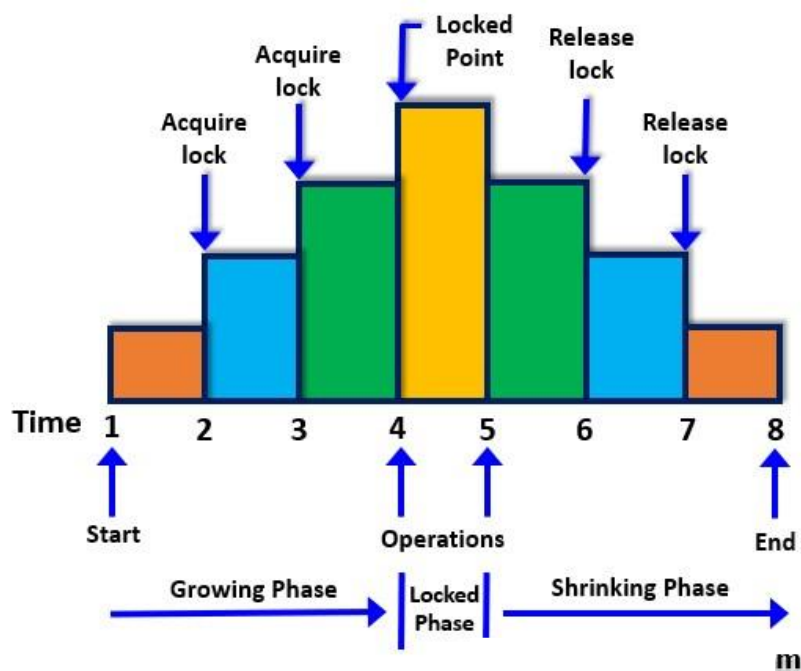
Starvation

Starvation is the situation when a transaction needs to wait for an indefinite period to acquire a lock.

Two Phase Locking Protocol

Two Phase Locking Protocol also known as 2PL protocol is a method of concurrency control. This locking protocol divides the execution phase of a transaction into three parts.

- In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.
- In the second part, the transaction acquires all the locks. The third phase is started as soon as the transaction releases its first lock.
- In the third part, the transaction cannot demand any new locks. It only releases the acquired locks.

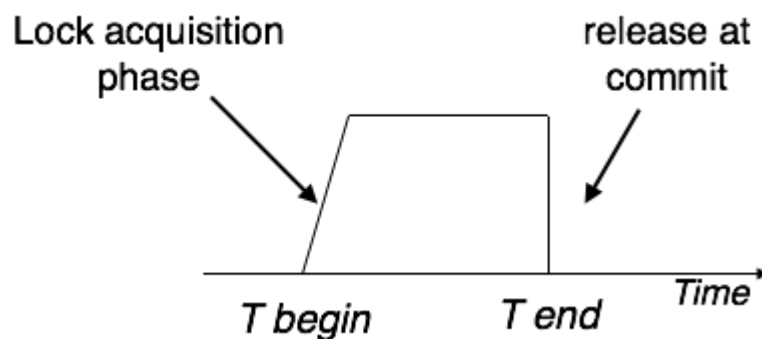


- **Growing Phase:** In this phase transaction may obtain locks but may not release any locks.
- **Shrinking Phase:** In this phase, a transaction may release locks but not obtain any new lock

Strict Two-Phase Locking Method

Strict-Two phase locking system is almost similar to 2PL. The only difference is that Strict-2PL never releases a lock after using it. It holds all the locks until the commit point and releases all the locks at one go when the process is over.

- The first phase of Strict-2PL is similar to 2PL. In the first phase, after acquiring all the locks, the transaction continues to execute normally.
- The only difference between 2PL and strict 2PL is that Strict-2PL does not release a lock after using it.
- Strict-2PL waits until the whole transaction to commit, and then it releases all the locks at a time.



2. Time stamping protocols for concurrency control

The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp. A timestamp is a tag (unique identifier) assign to every transaction.

The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.

- The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.

- The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.
- Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first.
- The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

Example:

Suppose there are three transactions T1, T2, and T3.

T1 has entered the system at time 0010

T2 has entered the system at 0020

T3 has entered the system at 0030

Priority will be given to transaction T1, then transaction T2 and lastly Transaction T3.

Advantages:

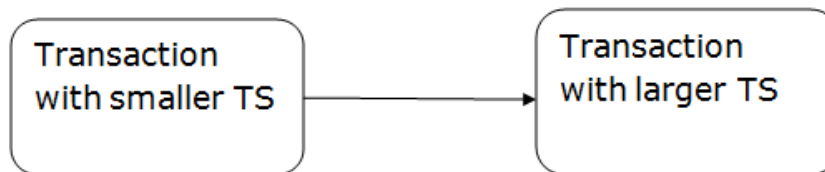


Image: Precedence Graph for TS ordering

- Schedules are serializable just like 2PL protocols
- No waiting for the transaction, which eliminates the possibility of deadlocks.

We need following timestamp for this technique

- The timestamp of transaction T_i is denoted as $TS(T_i)$.-It is the time when transaction entered in the system.
- Read time-stamp of data-item X is denoted by $R-TS(X)$ - It is the largest timestamp of any transaction that executed **read(X)** successfully.
- Write time-stamp of data-item X is denoted by $W-TS(X)$ - It is the largest timestamp of any transaction that executed **write(X)** successfully.

Example

Time	11.00	11.10	11.20
Timestamp	1100	1110	1120
Transaction	T1	T2	T3
	R(A)		
	W(A)		
		R(A)	
			R(A)
			W(A)
		W(A)	

TS(T1) = 1100

TS(T2) = 1110

TS(T3) = 1120

R-TS(A) = 1120

W-TS(A) = 1110

Basic Timestamp ordering protocol works as follows:

1. Check the following condition whenever a transaction T_i issues a **Read (X)** operation:
 - If $W_TS(X) > TS(T_i)$ then the operation is rejected.
 - If $W_TS(X) \leq TS(T_i)$ then the operation is executed.
 - Timestamps of all the data items are updated.
2. Check the following condition whenever a transaction T_i issues a **Write(X)** operation:
 - If $TS(T_i) < R_TS(X)$ or if $TS(T_i) < W_TS(X)$ then the operation is rejected and T_i is rolled back otherwise the operation is executed.

Example-Allowed cases

Timestamp	100	200
Transaction	T1	T2
	R(A)	
		W(A)

Timestamp	100	200
Transaction	T1	T2
	W(A)	
		R(A)

Timestamp	100	200
Transaction	T1	T2
	W(A)	
		W(A)

Example- Not Allowed cases

Timestamp	100	200
Transaction	T1	T2
		R(A)
	W(A)	

Timestamp	100	200
Transaction	T1	T2
	R(A)	
		W(A)

Timestamp	100	200
Transaction	T1	T2
	W(A)	
		W(A)

Validation Based Protocol

Validation Based Protocol is also called **Optimistic Concurrency Control Technique**. This protocol is used in DBMS (Database Management System) for avoiding concurrency in transactions. It is called optimistic because of the assumption it makes, i.e. very less interference occurs. It is a type of concurrency control techniques that works on the [validation rules](#) and [time-stamps](#).

In this technique, no checking is done while the transaction is been executed. Until the transaction end is reached updates in the transaction are not applied directly to the database. All updates are applied to local copies of data items kept for the transaction. At the end of transaction execution, while execution of the transaction, a **validation phase** checks whether any of transaction updates violate serializability. If there is no violation of serializability the transaction is committed and the database is updated; or else, the transaction is updated and then restarted.

The Validation based Protocol is performed in the following three phases:

1. Read Phase
2. Validation Phase
3. Write Phase

Read Phase

In the Read Phase, the data values from the database can be read by a transaction but the write operation or updates are only applied to the local data copies, not the actual database.

Validation Phase

In Validation Phase, the data is checked to ensure that there is no violation of serializability while applying the transaction updates to the database.

Write Phase

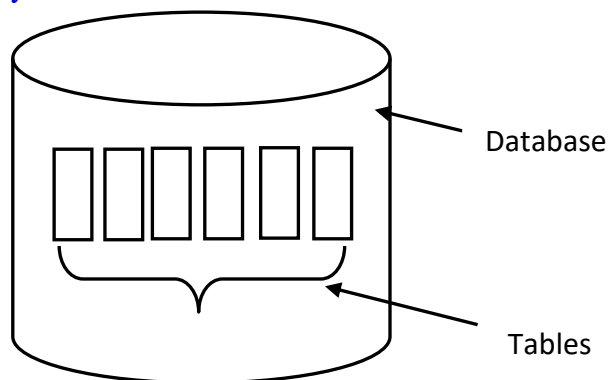
In the Write Phase, the updates are applied to the database if the validation is successful, else; the updates are not applied, and the transaction is rolled back.

There are three timestamps that control the serializability of the validation based protocol in the database management system, such as.

- **Start(Timestamp):** The start timestamp is the initial timestamp when the data item being read and executed in the read phase of the validation protocol.
- **Validation(Timestamp):** The validation timestamp is the timestamp when T1 completed the read phase and started with the validation phase.
- **Finish(Timestamp):** The finish timestamp is the timestamp when T1 completes the writing process in the writing phase.

Granularity

Concurrency Control Techniques used different methods for serializability. A certain drawback of this technique is if a transaction T_i needs to access the entire database and a locking protocol is used, then T_i must lock each item in the database. Suppose another transaction just needs to access a few data items from a database, so locking the entire database seems to be unnecessary also it may cost us loss of Concurrency, which was our primary goal. To solve this problem between Efficiency and Concurrency we use [Granularity](#).



Granularity – It is the size of data item allowed to lock.

In general term we can say granularity is directly concern with [hierarchy of level](#).

Multiple Granularity

- It can be defined as [hierarchically breaking up the database into blocks](#) which can be locked.
- The Multiple Granularity protocol enhances concurrency and reduces lock overhead.
- It maintains the track of [what to lock](#) and [how to lock](#).
- It makes easy to decide either to lock a data item or to unlock a data item. This type of hierarchy can be graphically [represented as a tree](#).

For example: Consider a tree which has four levels of nodes.

- The first level or higher level shows the entire database.
- The second level represents a node of files or tables.
- The file/table consists of children nodes which are known as Records.
- Finally, each record contains child nodes known as Fields.
- Hence, the levels of the tree starting from the top level are as follows:
 1. Database
 2. Files/Tables
 3. Records
 4. Fields

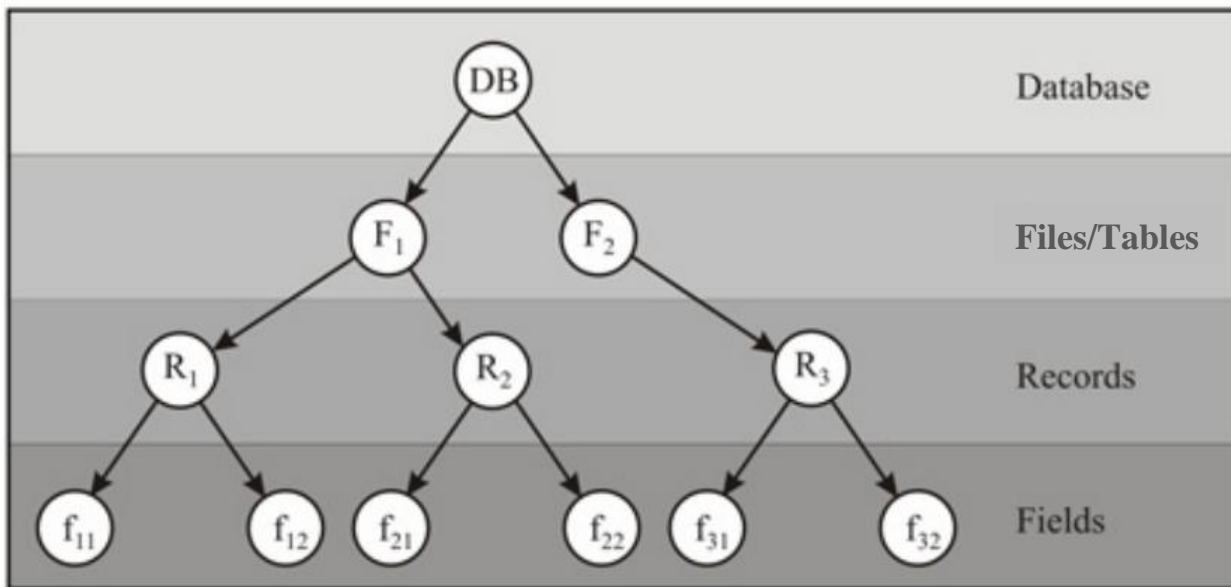


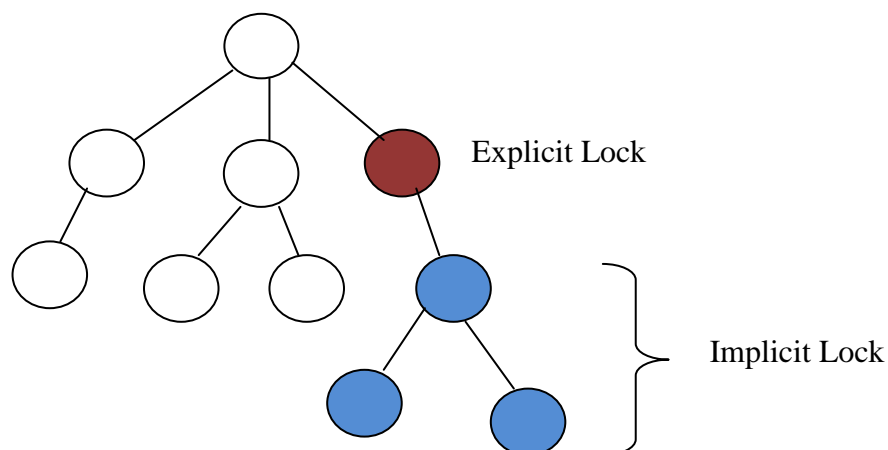
Figure – Multi Granularity tree Hierarchy

Consider the above diagram for the example, **each node in the tree can be locked individually**. As in the 2PL, it shall use **shared** and **exclusive** lock modes. When a transaction locks a node, in either shared or exclusive mode, the transaction also **implicitly locks all the descendants** of that node in the same lock mode. For example, if transaction T_1 gets an explicit lock on file F_2 in exclusive mode, then it has an implicit lock in exclusive mode on all the records belonging to that file. It does not need to lock the individual records of F_2 explicitly. This is the main difference between 2PL and Hierarchical locking for multiple granularity.

Lock Concept

- If we lock the database then everything is lock.
- So when the parent node is lock then children node is automatic lock.
- This is origin point to understand the concept of **Explicit Lock** and **Implicit Lock Mode**.

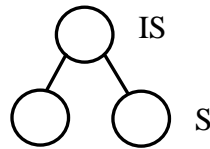
Explicit Lock If any transaction apply this lock at higher level and due to its effect if lower level node is automatic lock then it is called **implicit lock**.



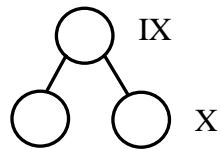
There are three additional lock modes with multiple granularity:

Intention Mode Lock

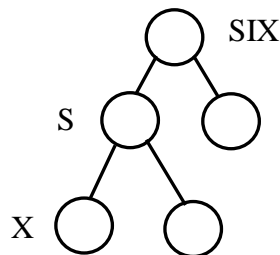
1. **Intention-shared (IS)** - It contains explicit locking at a lower level of the tree but only with shared locks.



2. **Intention-exclusive (IX)** - It contains explicit locking at a lower level with exclusive or shared locks.



3. **Shared-intention-exclusive (SIX)** - the sub-tree rooted by that node is locked explicitly in shared mode and explicit locking is being done at a lower level with exclusive mode locks.



Recovery with concurrent transaction

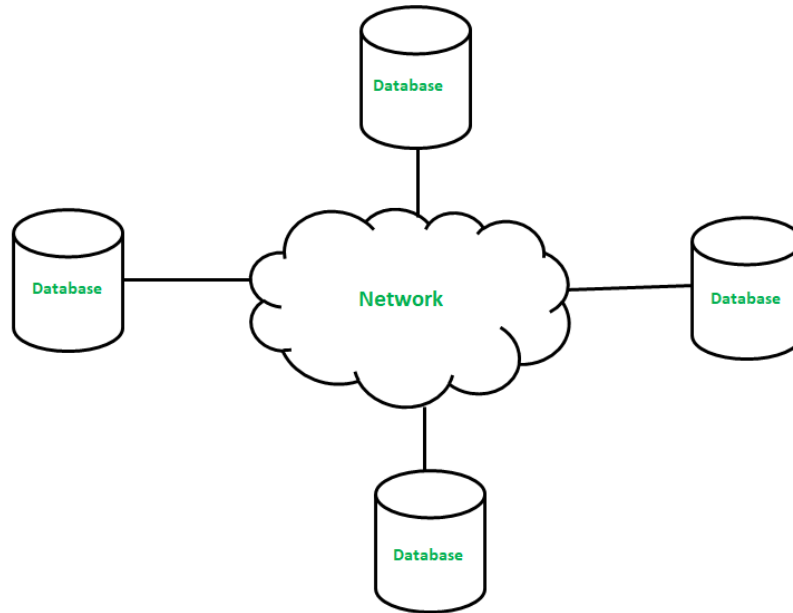
Whenever more than one transaction is being executed, then the interleaved of logs occur. During recovery, it would become difficult for the recovery system to backtrack all logs and then start recovering. To ease this situation, 'checkpoint' concept is used by most DBMS.

Distributed Database Management System (DDBMS)

A distributed database is basically a database that is not limited to one system; it is spread over different sites, i.e, on multiple computers. In a distributed database, there are a number of databases that may be geographically distributed all over the world.

A **distributed database** is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.

This may be required when a particular database needs to be accessed by various users globally. It needs to be managed such that for the users it looks like one single database.



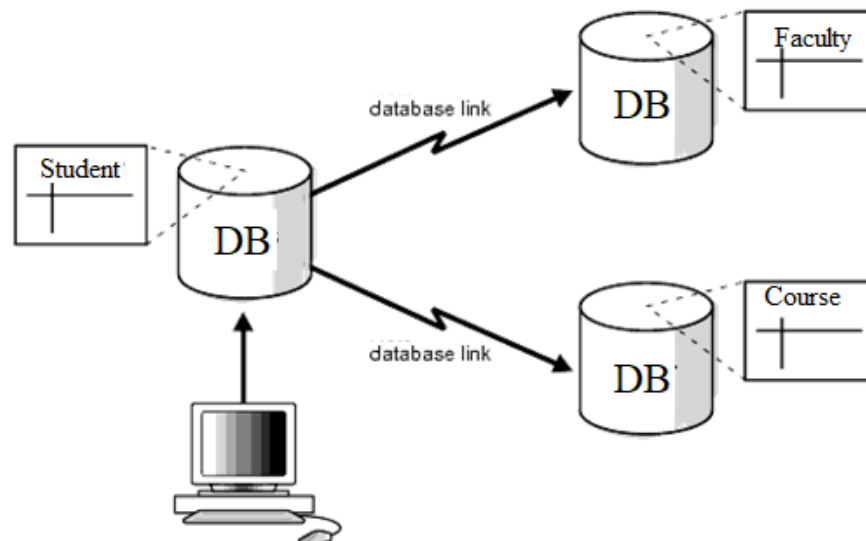
A distributed database management system (DDBMS) is a centralized software system that manages a distributed database in a manner as if it were all stored in a single location.

Transaction Processing in Distributed system

A **distributed transaction** is a set of operations on data that is performed across two or more data databases. It is typically coordinated across separate nodes connected by a network.

There are two possible outcomes:

- 1) All operations successfully complete, or
- 2) None of the operations are performed at all due to a failure somewhere in the system.



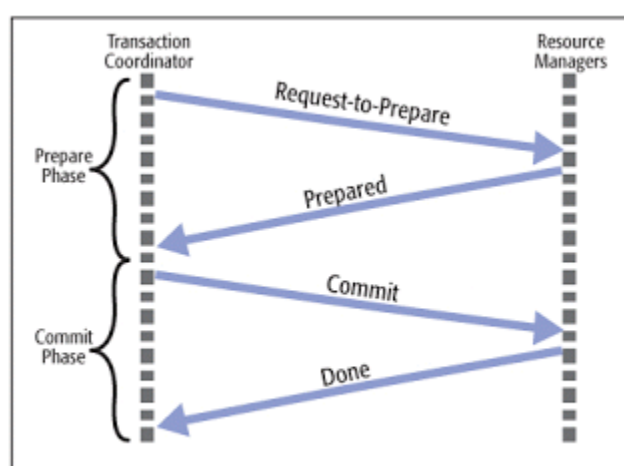
In the second case, if some work was completed prior to the failure, that work will be undone. This type of operation is in fulfillment with the “ACID” properties of databases that ensure data integrity. ACID is most commonly associated with transactions on a single database server, but distributed transactions uses “[two-phase commit](#)” (2PC) that guarantee across multiple databases.

Working of Distributed Transactions

Distributed transactions have the same processing completion requirements as regular database transactions, but they are more challenging. The multiple resources add [more points of failure](#), such as the [separate software systems](#) that run the resources (e.g., the database software), the [extra hardware servers](#), and [network](#) failures. This makes distributed transactions at risk to failures.

For a distributed transaction to occur, transaction managers coordinate the resources (either multiple databases or multiple nodes of a single database). The transaction manager decides whether to [commit a successful transaction or rollback an unsuccessful transaction](#), the latter of which leaves the database unchanged.

First, an application requests the distributed transaction to the transaction manager. The transaction manager then branches to each resource, which will have its own “resource manager” to help it participate in distributed transactions.



Distributed transactions are often done in [two phases](#).

Phase 1: Prepare Phase

- After each resource manager has locally completed its transaction, it sends a “[DONE](#)” message to the transaction manager. When the transaction manager has received “[DONE](#)” message from all resource manager, it sends a “[Prepare](#)” message to the all resource managers.

- The resource managers vote on whether they still want to commit or not. If a resource manager wants to commit, it sends a “Ready” message.
- A resource manager that does not want to commit sends a “Not Ready” message. This may happen when the resource manager has conflicting concurrent transactions or there is a timeout.

Phase 2: Commit/Abort Phase

- After the transaction manager has received “Ready” message from all the resource managers.
- The transaction manager sends a “Global Commit” message to the resource manager.
- The resource managers apply the transaction and send a “Commit ACK” message to the transaction manager.
- When the transaction manager receives “Commit ACK” message from all the resource managers, it considers the transaction as committed.

Distributed Data Storage

There are 2 ways in which data can be stored on different sites. These are:

1. Replication
2. Fragmentation

1. Data Replication

Data Replication is the process of storing data in more than one site or node. It is useful in **improving the availability of data**. It is simply copying data from a database from one server to another server so that all the users can share the same data without any inconsistency. The result is a **distributed database** in which users can access data relevant to their tasks without interfering with the work of others.

Advantages of replication

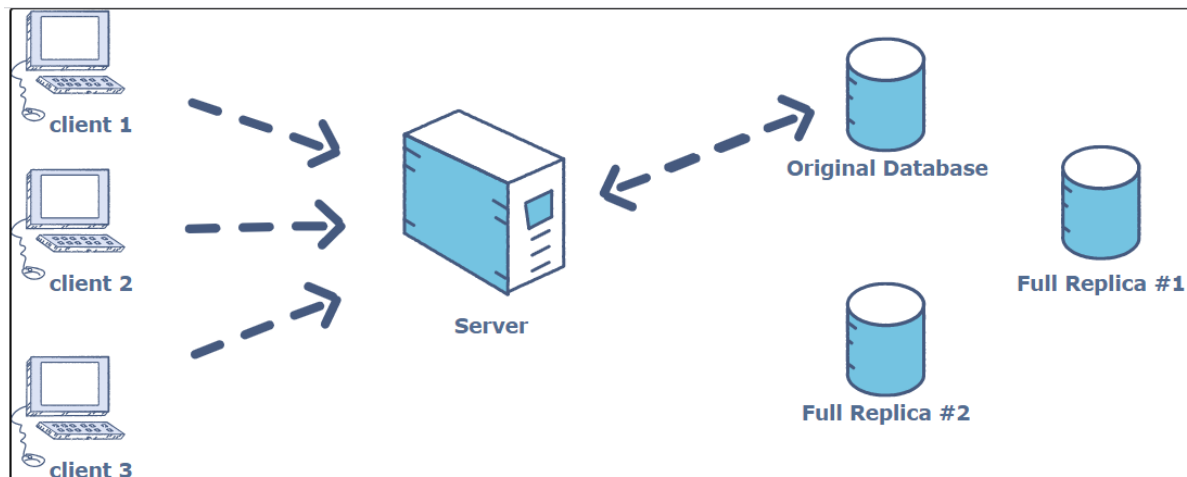
Replication in a database offers database users various benefits that are termed below

- Availability
- Has Reliability
- Gives high Performance
- Facilitates load reduction
- Provide disconnected computing
- Supports many users

There are two kinds of replication techniques:

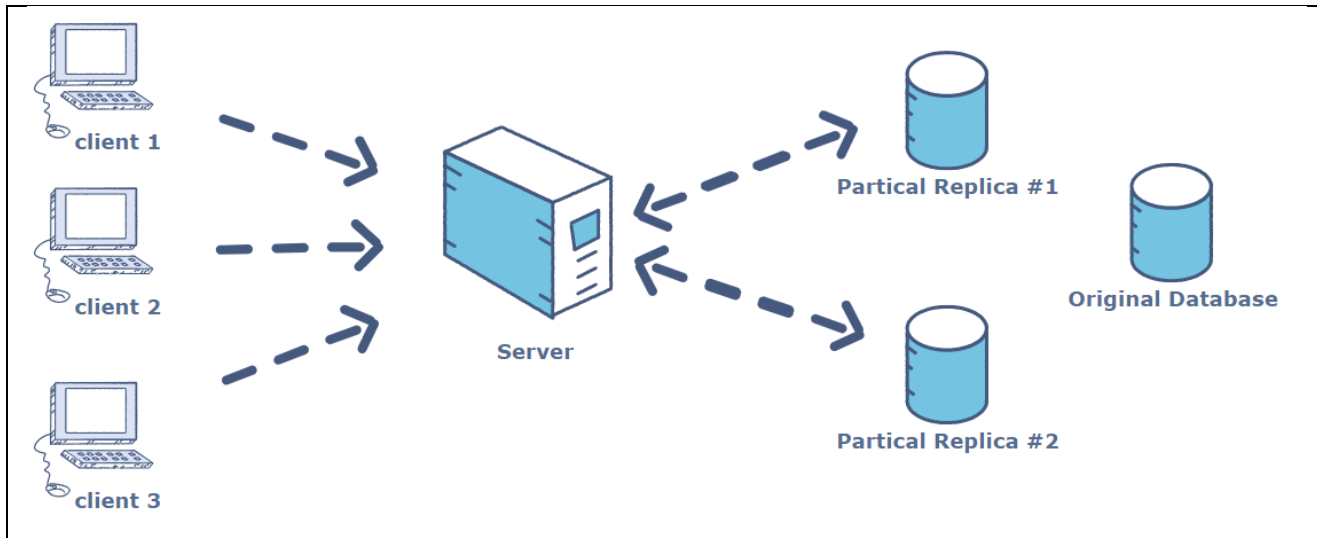
a) Full Replication

Full replication involves replicating the entire database at all sites of the distributed system. This improves data availability, but all the sites need to be constantly updated so that only *updated* data is available to every user.



b) Partial Replication

In this replication technique, only a frequently portion of the database is replicated while the other portion is left out. This does reduce the amount of data to be replicated and updated, but it also means that there are fewer sites (than full replication) that hold some particular data (data availability is compromised).



2. Data Fragmentation

Fragmentation is the task of dividing a table into a set of smaller tables. The subsets of the table are called **fragments**.

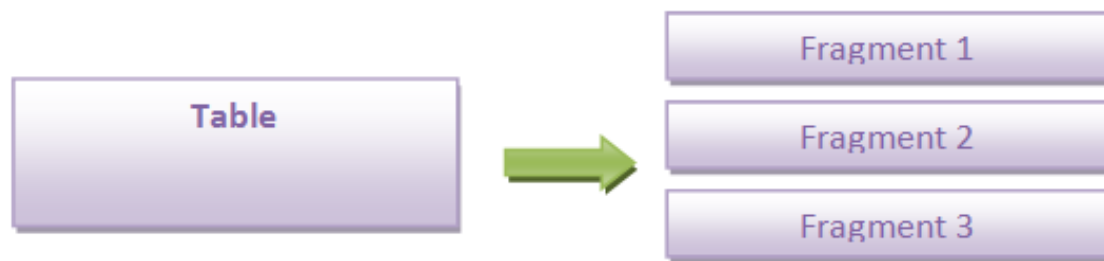
In this approach, the tables are fragmented (i.e., they're divided into smaller parts) and each of the fragments is stored in different sites where they're required. It must be made sure that the fragments are such that they can be used to reconstruct the original relation (i.e, there isn't any loss of data). Fragmentation is useful as it doesn't create copies of data, consistency is not a problem.

There are 3 types of data fragmentations in DDBMS.

- a) Horizontal Data Fragmentation
- b) Vertical Data Fragmentation
- c) Hybrid Data Fragmentation

a) Horizontal Data Fragmentation: (*Splitting by rows*) – The relation (table) is fragmented into groups of tuples so that each tuple is assigned to at least one fragment.

As the name suggests, here the data / records are fragmented horizontally. i.e.; horizontal subset of table data is created and are stored in different database in DDB.



Students

Roll No	Name	DOB	Course
156	Ram	15.10.2000	BCA
157	Pankaj	12.12.2000	BCA
158	Suresh	01.10.2001	BCA
159	Anjali	05.10.2000	BCA
160	Shyam	02.11.1999	MCA
161	Mohd Ali	05.12.1998	MCA
162	Shane Alam	10.10.1998	MCA
163	Meeta	19.12.1999	MCA
170	Mohan	05.05.1994	PGDCA
171	Abhijeet	10.12.1995	PGDCA
172	Darshika	10.15.1996	PGDCA
173	Harpreet	02.02.1996	PGDCA

Roll No	Name	DOB	Course
156	Ram	15.10.2000	BCA
157	Pankaj	12.12.2000	BCA
158	Suresh	01.10.2001	BCA
159	Anjali	05.10.2000	BCA

Roll No	Name	DOB	Course
160	Shyam	02.11.1999	MCA
161	Mohd Ali	05.12.1998	MCA
162	Shane Alam	10.10.1998	MCA
163	Meeta	19.12.1999	MCA

Roll No	Name	DOB	Course
170	Mohan	05.05.1994	PGDCA
171	Abhijeet	10.12.1995	PGDCA
172	Darshika	10.15.1996	PGDCA
173	Harpreet	02.02.1996	PGDCA

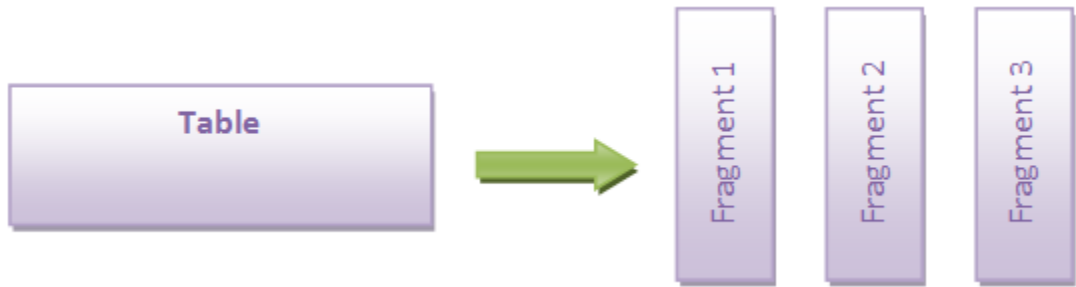
For example, consider the students studying in different courses in any university like BCA, MCA, PGDCA etc. number of students from of these courses are not a small number. They are huge in number. When any details of any one student are required, whole table needs to be accessed to get the information. Again the student table may present in any location in the university. But the concept of DDB is to place the data in the nearest DB so that it will be accessed quickly. Hence what we do is divide the entire student table data horizontally based on the course. i.e.;

```

SELECT * FROM Students WHERE Course = 'BCA';
SELECT * FROM Students WHERE Course = 'MCA';
SELECT * FROM Students WHERE Course = 'PGDCA';
  
```

Now these queries will give the subset of records from Students table depending on the course of the students. Any insert, update and delete on the student records will be done on the DBs at their location and it will be synched with the main table at regular intervals.

b) Vertical Data Fragmentation: (*Splitting by columns*) – The schema of the relation is divided into smaller schemas. Each fragment must contain a common candidate key so as to ensure lossless join.



Students

Roll No	Name	DOB	Course		Roll No	Name	Roll No	Course
156	Ram	15.10.2000	BCA		156	Ram	156	BCA
157	Pankaj	12.12.2000	BCA		157	Pankaj	157	BCA
158	Suresh	01.10.2001	BCA		158	Suresh	158	BCA
159	Anjali	05.10.2000	BCA		159	Anjali	159	BCA
160	Shyam	02.11.1999	MCA		160	Shyam	160	MCA
161	Mohd Ali	05.12.1998	MCA		161	Mohd Ali	161	MCA
162	Shane Alam	10.10.1998	MCA		162	Shane Alam	162	MCA
163	Meeta	19.12.1999	MCA		163	Meeta	163	MCA
170	Mohan	05.05.1994	PGDCA		170	Mohan	170	PGDCA
171	Abhijeet	10.12.1995	PGDCA		171	Abhijeet	171	PGDCA
172	Darshika	10.15.1996	PGDCA		172	Darshika	172	PGDCA
173	Harpreet	02.02.1996	PGDCA		173	Harpreet	173	PGDCA

For example consider the Students table with Roll No, Name, DOB, Course. The vertical fragmentation of this table may be dividing the table into different tables with one or more columns from Students.

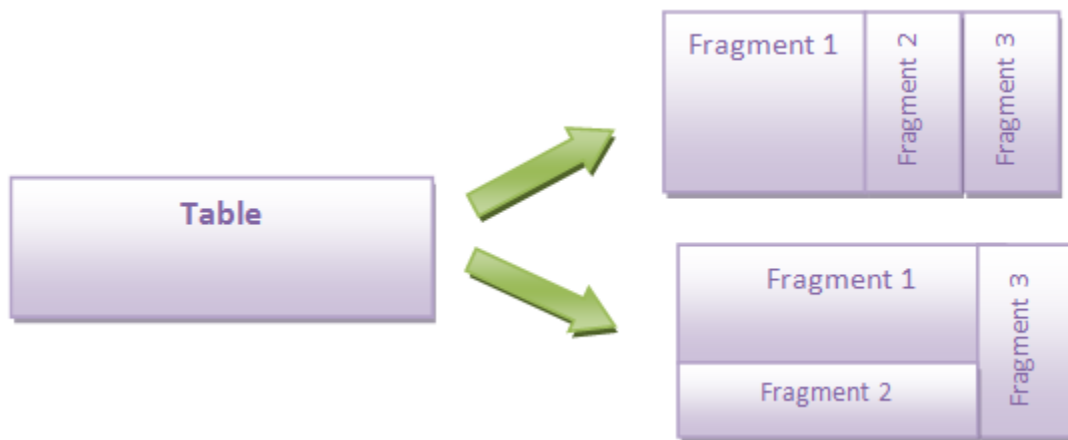
```
SELECT Roll No, Name FROM Students;
SELECT Roll No, Course FROM Students;
```

This type of fragment will have fragmented details about whole students. This will be useful when the user needs to query only few details about the students. For example consider a query to find the course of the students. This can be done by querying the second fragment of the table.

c) Hybrid (Mix) Data Fragmentation:

This is the combination of horizontal as well as vertical fragmentation. Hybrid fragmentation can be done in two alternative ways –

- At first, generate a set of horizontal fragments; then generate vertical fragments from one or more of the horizontal fragments.
- At first, generate a set of vertical fragments; then generate horizontal fragments from one or more of the vertical fragments.



Student

Roll No	Name	DOB	Course
156	Ram	15.10.2000	BCA
157	Pankaj	12.12.2000	BCA
158	Suresh	01.10.2001	BCA
159	Anjali	05.10.2000	BCA
160	Shyam	02.11.1999	MCA
161	Mohd Ali	05.12.1998	MCA
162	Shane Alam	10.10.1998	MCA
163	Meeta	19.12.1999	MCA
170	Mohan	05.05.1994	PGDCA
171	Abhijeet	10.12.1995	PGDCA
172	Darshika	10.15.1996	PGDCA
173	Harpreet	02.02.1996	PGDCA



Roll No	Name	Course
156	Ram	BCA
157	Pankaj	BCA
158	Suresh	BCA
159	Anjali	BCA

Roll No	Name	Course
160	Shyam	MCA
161	Mohd Ali	MCA
162	Shane Alam	MCA
163	Meeta	MCA

Consider the students table with below fragmentations.

```
SELECT Roll No, Name FROM Students WHERE Course = 'BCA';  
SELECT Roll No, Name FROM Students WHERE Course = 'MCA';
```

This is a hybrid or mixed fragmentation of Students table.

Allocation in distributed system

Allocation: Each copy of a fragment must be assigned to a particular site in the distributed system. This process is called **data distribution** or **allocation**.

The allocation technique is used for the allocation of fragments or replicas of fragments for storage at various sites.

All the information concerning data fragmentation, allocation, and replication is stored in a global directory. This directory can be accessed by the DDBS applications as and when required. The choice of sites and the degree of replication depend on the performance and availability goals of the system and on the types and frequencies of transactions submitted at each site.

For example, if high availability is required, transactions can be submitted at any site, and most transactions are retrieval only, a fully replicated database is a good choice.

However, if certain transactions that access particular parts of the database are mostly submitted at a particular site, the corresponding set of fragments can be allocated at that site only. Data that is accessed at multiple sites can be replicated at those sites. If many updates are performed, it may be useful to limit replication.

Database migration

Database migration, in simple words, means moving data from one platform to another. There are many reasons we might want to move to a different platform. For example, a company might decide to save money by moving to a cloud-based database. Or a company might find that some particular database software has features that are critical for their business needs. Or the legacy systems are simply outdated. The process of database migration can involve multiple phases and iterations, including assessing the current databases and future needs of the company, migrating the schema, and normalizing and moving the data.

Benefits of Database Migration

Costs. One of the primary reasons that companies migrate databases is to save money. Often, companies will move from an on-premise database to a cloud database. This saves on infrastructure as well as the manpower and expertise needed to support it.

Modernized software. Another common reason for migration is to move from an outdated system or legacy systems to a system that is designed for modern data needs. In the age of big data, new storage techniques are a necessity. For example, a company might choose to move from a legacy SQL database to a data lake or another flexible system.

One source of truth. Another common reason to migrate data is to move all the data into one place that is accessible by all divisions of the company. Sometimes, this occurs after an acquisition, when systems need to be combined. Or, it can happen when different systems are siloed throughout a company. For example, the IT department might use one database while the Marketing group uses another database and these systems cannot "talk" to each other. When we have different databases that are incompatible, it's hard to get insights from our data.

Database Migration Challenges

Database migration can be very complex, but with proper planning, these common challenges can be mitigated:

Challenge #1: Finding our Siloed Databases

If our company has been around a while, we likely have many disparate databases that exist within various parts of our company. They may be in different departments and different geographies. They may have been brought in through acquisitions. Part of our task in migrating databases is to locate the disparate databases in our company and plan how we will normalize data and convert schemas.

Challenge #2: Data Loss or Corruption

When migrating databases, it's critical to ensure our data is safely moved without loss or corruption. We'll need to plan how to test for data loss or corruption that can happen when we move data from one system to another.

Challenge # 3: Security

When we move data from one platform to another, it's critical that the data is secure. Unfortunately, there are many nefarious actors who would love to get their hands on the personal data we have stored away. We might choose to encrypt the data or remove personally identifiable information (PII) as a part of the migration process.

Database Migration Process Phases

Database migration is a multiphase process that involves some or all the following steps:

Assessment. At this stage, we'll need to gather business requirements, assess the costs and benefits, and perform data profiling. Data profiling is a process by which we get to know our existing data and database schema. We'll also need to plan how we will move the data — will we use an ETL (Extraction, Transformation, and Loading) tool, scripting, or some other tool to move the data?

Database schema conversion. The schema is a blueprint of how the database is structured, and it varies based on the rules of a given database. When we move data from one system to another, we'll need to convert the schemas so that the structure of the data works with the new database.

Data migration. After we have completed all the preliminary requirements, we'll need to actually move the data. This may involve scripting or using an ETL tool or some other tool to move the data. During the migration, we will likely transform the data, normalize data types, and check for errors.

Testing and tuning. Once we've moved the data, we need to verify that the data was moved correctly, is complete, isn't missing values, doesn't contain null values, and is valid.

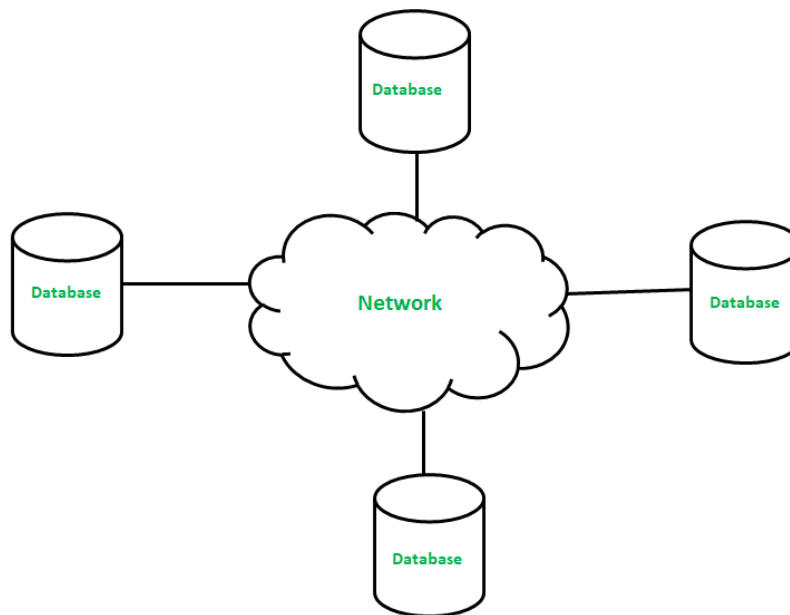
Unit-5 DBMS

Distributed Database Management System (DDBMS)

A distributed database is basically a database that is not limited to one system; it is spread over different sites, i.e., on multiple computers. In a distributed database, there are a number of databases that may be geographically distributed all over the world.

A **distributed database** is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.

This may be required when a particular database needs to be accessed by various users globally. It needs to be managed such that for the users it looks like one single database.



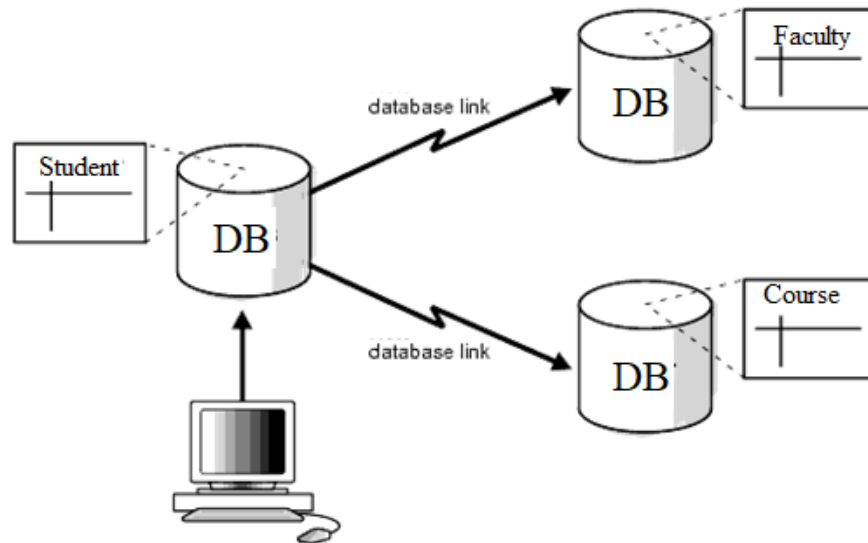
A distributed database management system (DDBMS) is a centralized software system that manages a distributed database in a manner as if it were all stored in a single location.

Transaction Processing in Distributed system

A **distributed transaction** is a set of operations on data that is performed across two or more data databases. It is typically coordinated across separate nodes connected by a network.

There are two possible outcomes:

- 1) All operations successfully complete, or
- 2) None of the operations are performed at all due to a failure somewhere in the system.



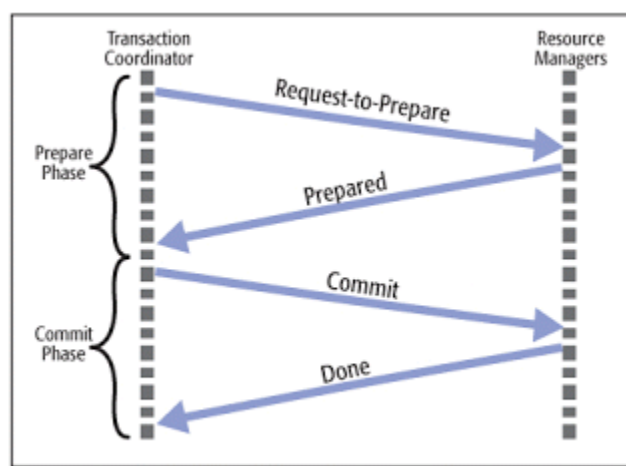
In the second case, if some work was completed prior to the failure, that work will be undone. This type of operation is in fulfillment with the “ACID” properties of databases that ensure data integrity. ACID is most commonly associated with transactions on a single database server, but distributed transactions uses “[two-phase commit](#)” (2PC) that guarantee across multiple databases.

Working of Distributed Transactions

Distributed transactions have the same processing completion requirements as regular database transactions, but they are more challenging. The multiple resources add [more points of failure](#), such as the [separate software systems](#) that run the resources (e.g., the database software), the [extra hardware servers](#), and [network](#) failures. This makes distributed transactions at risk to failures.

For a distributed transaction to occur, transaction managers coordinate the resources (either multiple databases or multiple nodes of a single database). The transaction manager decides whether to [commit a successful transaction or rollback an unsuccessful transaction](#), the latter of which leaves the database unchanged.

First, an application requests the distributed transaction to the transaction manager. The transaction manager then branches to each resource, which will have its own “resource manager” to help it participate in distributed transactions.



Distributed transactions are often done in **two phases**.

Phase 1: Prepare Phase

- After each resource manager has locally completed its transaction, it sends a “**DONE**” message to the transaction manager. When the transaction manager has received “**DONE**” message from all resource manager, it sends a “**Prepare**” message to the all resource managers.
- The resource managers vote on whether they still want to commit or not. If a resource manager wants to commit, it sends a “**Ready**” message.
- A resource manager that does not want to commit sends a “**Not Ready**” message. This may happen when the resource manager has conflicting concurrent transactions or there is a timeout.

Phase 2: Commit/Abort Phase

- After the transaction manager has received “**Ready**” message from all the resource managers.
- The transaction manager sends a “**Global Commit**” message to the resource manager.
- The resource managers apply the transaction and send a “**Commit ACK**” message to the transaction manager.
- When the transaction manager receives “**Commit ACK**” message from all the resource managers, it considers the transaction as committed.

Distributed Data Storage

There are 2 ways in which data can be stored on different sites. These are:

1. Replication
2. Fragmentation

1. Data Replication

Data Replication is the process of storing data in more than one site or node. It is useful in **improving the availability of data**. It is simply copying data from a database from one server to another server so that all the users can share the same data without any inconsistency. The result is a **distributed database** in which users can access data relevant to their tasks without interfering with the work of others.

Advantages of replication

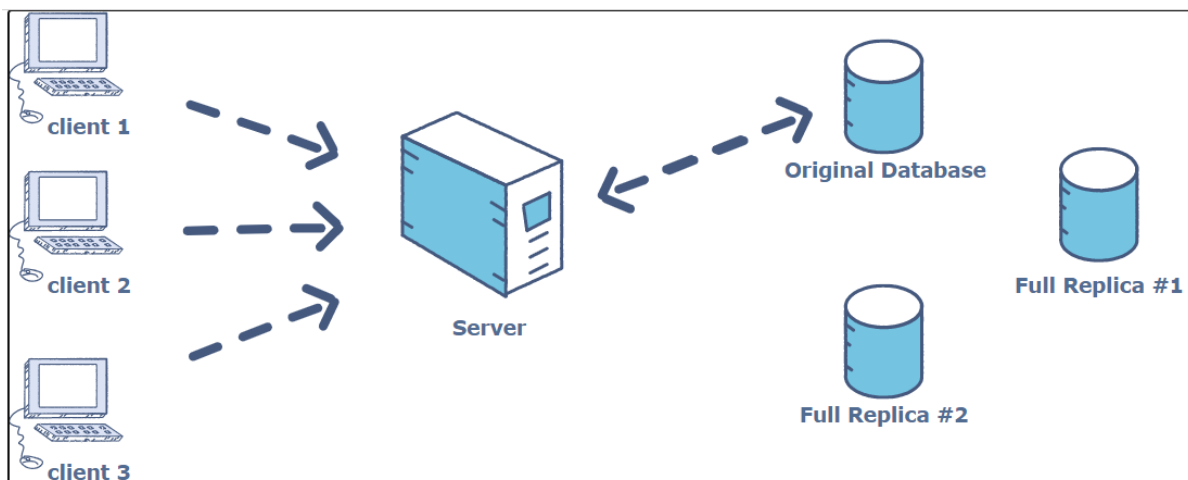
Replication in a database offers database users various benefits that are termed below

- Availability
- Has Reliability
- Gives high Performance
- Facilitates load reduction
- Provide disconnected computing
- Supports many users

There are two kinds of replication techniques:

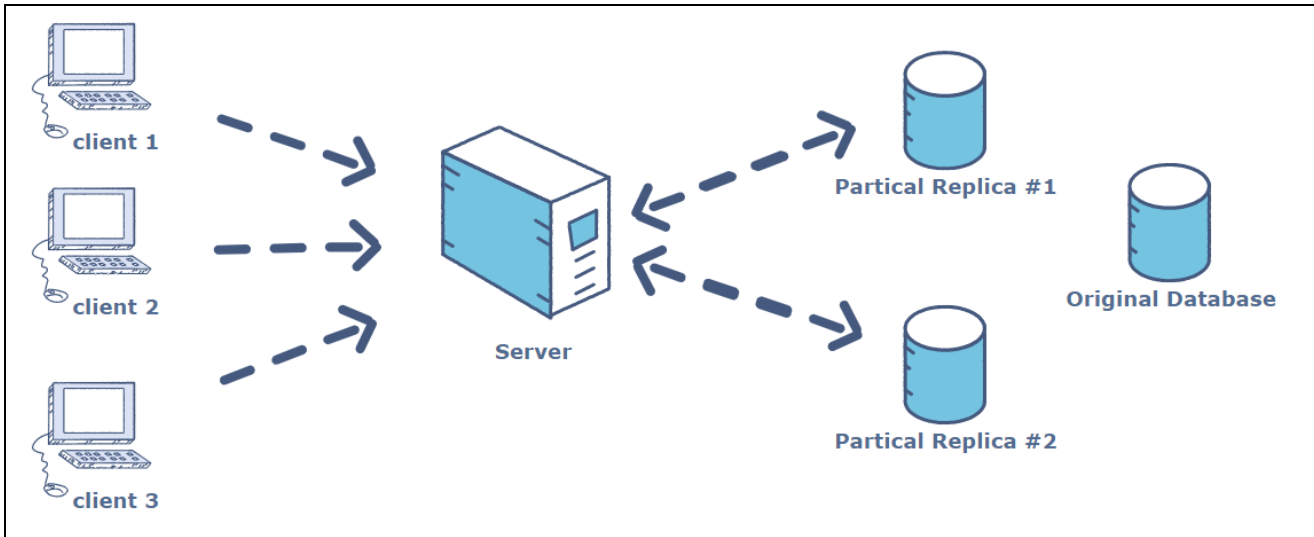
a) Full Replication

Full replication involves replicating the entire database at all sites of the distributed system. This improves data availability, but all the sites need to be constantly updated so that only *updated* data is available to every user.



b) Partial Replication

In this replication technique, only a frequently portion of the database is replicated while the other portion is left out. This does reduce the amount of data to be replicated and updated, but it also means that there are fewer sites (than full replication) that hold some particular data (data availability is compromised).



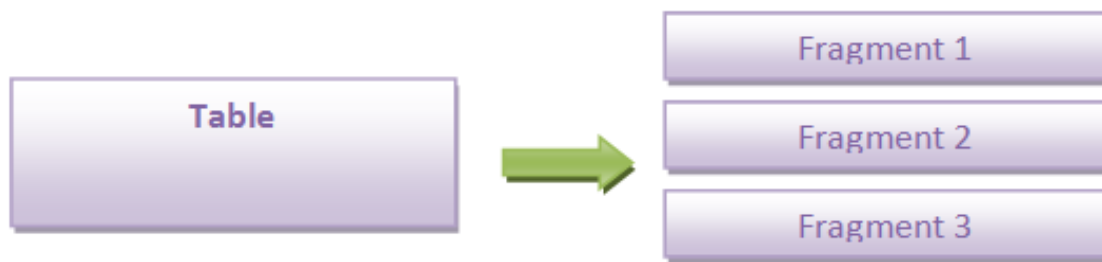
2. Data Fragmentation

Fragmentation is the task of dividing a table into a set of smaller tables. The subsets of the table are called **fragments**.

In this approach, the tables are fragmented (i.e., they're divided into smaller parts) and each of the fragments is stored in different sites where they're required. It must be made sure that the fragments are such that they can be used to reconstruct the original relation (i.e., there isn't any loss of data). Fragmentation is useful as it doesn't create copies of data, consistency is not a problem.

There are 3 types of data fragmentations in DDBMS.

- a) Horizontal Data Fragmentation
 - b) Vertical Data Fragmentation
 - c) Hybrid Data Fragmentation
- a) **Horizontal Data Fragmentation:** (*Splitting by rows*) – The relation (table) is fragmented into groups of tuples so that each tuple is assigned to at least one fragment.
- As the name suggests, here the data / records are fragmented horizontally. i.e.; horizontal subset of table data is created and are stored in different database in DDB.



Students

Roll No	Name	DOB	Course
156	Ram	15.10.2000	BCA
157	Pankaj	12.12.2000	BCA
158	Suresh	01.10.2001	BCA
159	Anjali	05.10.2000	BCA
160	Shyam	02.11.1999	MCA
161	Mohd Ali	05.12.1998	MCA
162	Shane Alam	10.10.1998	MCA
163	Meeta	19.12.1999	MCA
170	Mohan	05.05.1994	PGDCA
171	Abhijeet	10.12.1995	PGDCA
172	Darshika	10.15.1996	PGDCA
173	Harpreet	02.02.1996	PGDCA

Roll No	Name	DOB	Course
156	Ram	15.10.2000	BCA
157	Pankaj	12.12.2000	BCA
158	Suresh	01.10.2001	BCA
159	Anjali	05.10.2000	BCA

Roll No	Name	DOB	Course
160	Shyam	02.11.1999	MCA
161	Mohd Ali	05.12.1998	MCA
162	Shane Alam	10.10.1998	MCA
163	Meeta	19.12.1999	MCA

Roll No	Name	DOB	Course
170	Mohan	05.05.1994	PGDCA
171	Abhijeet	10.12.1995	PGDCA
172	Darshika	10.15.1996	PGDCA
173	Harpreet	02.02.1996	PGDCA

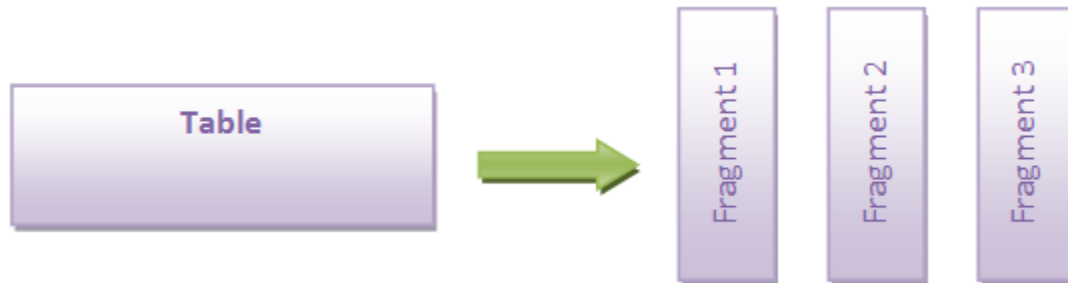
For example, consider the students studying in different courses in any university like BCA, MCA, PGDCA etc. number of students from of these courses are not a small number. They are huge in number. When any details of any one student are required, whole table needs to be accessed to get the information. Again the student table may present in any location in the university. But the concept of DDB is to place the data in the nearest DB so that it will be accessed quickly. Hence what we do is divide the entire student table data horizontally based on the course. i.e.;

```

SELECT * FROM Students WHERE Course = 'BCA';
SELECT * FROM Students WHERE Course = 'MCA';
SELECT * FROM Students WHERE Course = 'PGDCA';
  
```

Now these queries will give the subset of records from Students table depending on the course of the students. Any insert, update and delete on the student records will be done on the DBs at their location and it will be synched with the main table at regular intervals.

b) Vertical Data Fragmentation: (*Splitting by columns*) – The schema of the relation is divided into smaller schemas. Each fragment must contain a common candidate key so as to ensure lossless join.



Students

Roll No	Name	DOB	Course
156	Ram	15.10.2000	BCA
157	Pankaj	12.12.2000	BCA
158	Suresh	01.10.2001	BCA
159	Anjali	05.10.2000	BCA
160	Shyam	02.11.1999	MCA
161	Mohd Ali	05.12.1998	MCA
162	Shane Alam	10.10.1998	MCA
163	Meeta	19.12.1999	MCA
170	Mohan	05.05.1994	PGDCA
171	Abhijeet	10.12.1995	PGDCA
172	Darshika	10.15.1996	PGDCA
173	Harpreet	02.02.1996	PGDCA



Roll No	Name
156	Ram
157	Pankaj
158	Suresh
159	Anjali
160	Shyam
161	Mohd Ali
162	Shane Alam
163	Meeta
170	Mohan
171	Abhijeet
172	Darshika
173	Harpreet

Roll No	Course
156	BCA
157	BCA
158	BCA
159	BCA
160	MCA
161	MCA
162	MCA
163	MCA
170	PGDCA
171	PGDCA
172	PGDCA
173	PGDCA

For example consider the Students table with Roll No, Name, DOB, Course. The vertical fragmentation of this table may be dividing the table into different tables with one or more columns from Students.

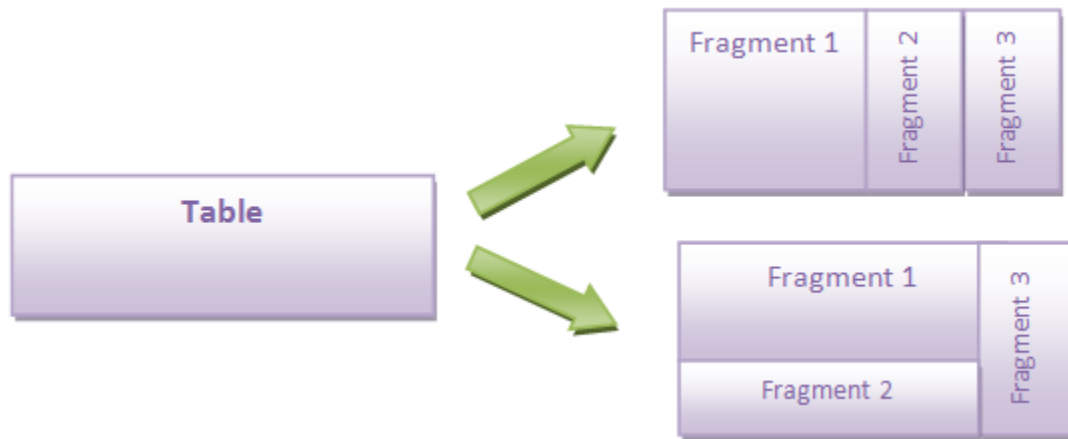
```
SELECT Roll No, Name FROM Students;
SELECT Roll No, Course FROM Students;
```

This type of fragment will have fragmented details about whole students. This will be useful when the user needs to query only few details about the students. For example consider a query to find the course of the students. This can be done by querying the second fragment of the table.

c) Hybrid (Mix) Data Fragmentation:

This is the combination of horizontal as well as vertical fragmentation. Hybrid fragmentation can be done in two alternative ways –

- At first, generate a set of horizontal fragments; then generate vertical fragments from one or more of the horizontal fragments.
- At first, generate a set of vertical fragments; then generate horizontal fragments from one or more of the vertical fragments.



Student

Roll No	Name	DOB	Course
156	Ram	15.10.2000	BCA
157	Pankaj	12.12.2000	BCA
158	Suresh	01.10.2001	BCA
159	Anjali	05.10.2000	BCA
160	Shyam	02.11.1999	MCA
161	Mohd Ali	05.12.1998	MCA
162	Shane Alam	10.10.1998	MCA
163	Meeta	19.12.1999	MCA
170	Mohan	05.05.1994	PGDCA
171	Abhijeet	10.12.1995	PGDCA
172	Darshika	10.15.1996	PGDCA
173	Harpreet	02.02.1996	PGDCA



Roll No	Name	Course
156	Ram	BCA
157	Pankaj	BCA
158	Suresh	BCA
159	Anjali	BCA

Roll No	Name	Course
160	Shyam	MCA
161	Mohd Ali	MCA
162	Shane Alam	MCA
163	Meeta	MCA

Consider the students table with below fragmentations.

```
SELECT Roll No, Name , Course FROM Students WHERE Course = 'BCA';
SELECT Roll No, Name, Course FROM Students WHERE Course = 'MCA';
```

This is a hybrid or mixed fragmentation of Students table.

Allocation in distributed system

Allocation: Each copy of a fragment must be assigned to a particular site in the distributed system. This process is called **data distribution** or **allocation**.

The allocation technique is used for the allocation of fragments or replicas of fragments for storage at various sites.

All the information concerning data fragmentation, allocation, and replication is stored in a global directory. This directory can be accessed by the DDBS applications as and when required.

The choice of sites and the degree of replication depend on the performance and availability goals of the system and on the types and frequencies of transactions submitted at each site.

For example, if high availability is required, transactions can be submitted at any site, and most transactions are retrieval only, a fully replicated database is a good choice.

However, if certain transactions that access particular parts of the database are mostly submitted at a particular site, the corresponding set of fragments can be allocated at that site only. Data that is accessed at multiple sites can be replicated at those sites. If many updates are performed, it may be useful to limit replication.

Recovery in distributed databases

Recovery is the most complicated process in distributed databases. Recovery of a failed system in the communication network is very difficult.

For example:

Consider that, location A sends message to location B and expects response from B but B is unable to receive it. There are several problems for this situation which are as follows.

- Message was failed due to failure in the network.
- Location B sent message but not delivered to location A.
- Location B crashed down.
- So it is actually very difficult to find the cause of failure in a large communication network.
- Distributed commit in the network is also a serious problem which can affect the recovery in a distributed databases.

Two-phase commit protocol in Distributed databases

- Two-phase protocol is a type of atomic commitment protocol. This is a distributed algorithm which can coordinate all the processes that participate in the database and decide to commit or terminate the transactions. The protocol is based on commit and terminates action.
- The two-phase protocol ensures that all participant which are accessing the database server can receive and implement the same action (Commit or terminate), in case of local network failure.
- Two-phase commit protocol provides automatic recovery mechanism in case of a system failure.
- The location at which original transaction takes place is called as coordinator and where the sub process takes place is called as **Cohort**.

Commit request:

In commit phase the coordinator attempts to prepare all cohorts and take necessary steps to commit or terminate the transactions.

Commit phase:

The commit phase is based on voting of cohorts and the coordinator decides to commit or terminate the transaction.

Concurrency problems in distributed databases.

Some problems which occur while accessing the database are as follows:

1. Failure at local locations

When system recovers from failure the database is out dated compared to other locations. So it is necessary to update the database.

2. Failure at communication location

System should have a ability to manage temporary failure in a communicating network in distributed databases. In this case, partition occurs which can limit the communication between two locations.

3. Dealing with multiple copies of data

It is very important to maintain multiple copies of distributed data at different locations.

4. Distributed commit

While committing a transaction which is accessing databases stored on multiple locations, if failure occurs on some location during the commit process then this problem is called as distributed commit.

5. Distributed deadlock

Deadlock can occur at several locations due to recovery problem and concurrency problem (multiple locations are accessing same system in the communication network).

Concurrency Controls in distributed databases

Since transactions are processed at multiple sites, two or more sites may get involved in deadlock. This must be resolved in a distributed manner. Concurrency control protocols can be broadly divided into two categories—

- Distributed Lock based protocols
- Distributed Time stamp based protocols
- Distributed Validation based protocols

Distributed Two-phase Locking Algorithm

The basic principle of distributed two-phase locking is same as the basic two-phase locking protocol. However, in a distributed system there are sites designated as lock managers. A lock manager controls lock acquisition requests from transaction monitors. In order to enforce co-ordination between the lock managers in various sites, at least one site is given the authority to see all transactions and detect lock conflicts.

Depending upon the number of sites who can detect lock conflicts, distributed two-phase locking approaches can be of three types –

- **Centralized two-phase locking** – In this approach, one site is designated as the central lock manager. All the sites in the environment know the location of the central lock manager and obtain lock from it during transactions.
- **Primary copy two-phase locking** – In this approach, a number of sites are designated as lock control centers. Each of these sites has the responsibility of managing a defined set of locks. All the sites know which lock control center is responsible for managing lock of which data table/fragment item.
- **Distributed two-phase locking** – In this approach, there are a number of lock managers, where each lock manager controls locks of data items stored at its local site. The location of the lock manager is based upon data distribution and replication.

Distributed Timestamp Concurrency Control

In a centralized system, timestamp of any transaction is determined by the physical clock reading. But, in a distributed system, any site's local physical/logical clock readings cannot be used as global timestamps, since they are not globally unique. So, a timestamp comprises of a combination of site ID and that site's clock reading.

For implementing timestamp ordering algorithms, each site has a **scheduler** that maintains a separate queue for each transaction manager. During transaction, a transaction manager sends a lock request to the site's

scheduler. The scheduler puts the request to the corresponding queue in increasing timestamp order. Requests are processed from the front of the queues in the order of their timestamps, i.e. the oldest first.

Distributed Validation Concurrency Control Algorithm

Distributed optimistic concurrency control algorithm extends validation concurrency control algorithm. For this extension, two rules are applied –

Rule 1 – According to this rule, a transaction must be validated locally at all sites when it executes. If a transaction is found to be invalid at any site, it is aborted. Local validation guarantees that the transaction maintains serializability at the sites where it has been executed. After a transaction passes local validation test, it is globally validated.

Rule 2 – According to this rule, after a transaction passes local validation test, it should be globally validated. Global validation ensures that if two conflicting transactions run together at more than one site, they should commit in the same relative order at all the sites they run together. This may require a transaction to wait for the other conflicting transaction, after validation before commit. This requirement makes the algorithm less optimistic since a transaction may not be able to commit as soon as it is validated at a site.

Recovery Manager (RMAN)

Recovery Manager (RMAN) is an Oracle utility that can back up, restore, and recover database files. The product is a feature of the Oracle database server and does not require separate installation.

Recovery Manager is a client/server application that uses database server sessions to perform backup and recovery. It stores metadata about its operations in the control file of the target database and, optionally, in a recovery catalog schema in an Oracle database.

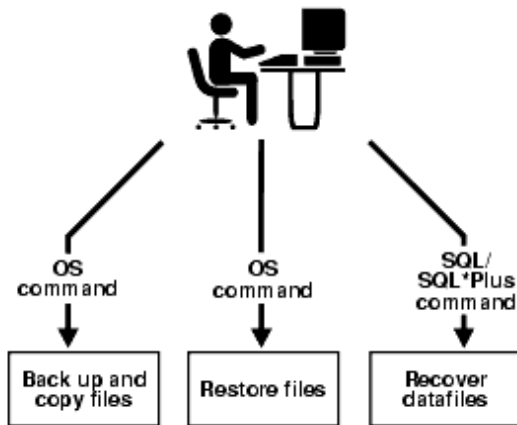
You can invoke RMAN as a command-line executable from the operating system prompt or use some RMAN features through the Enterprise Manager GUI.

Need of RMAN

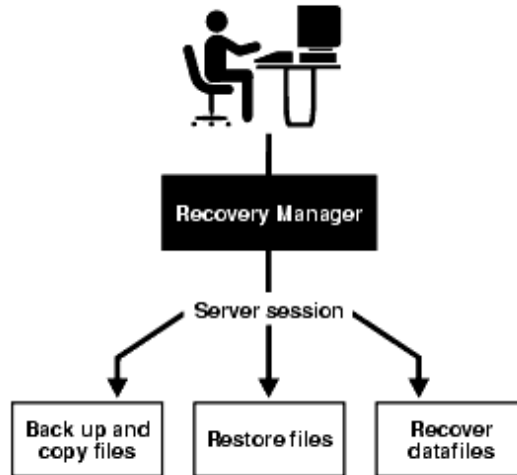
Most production database systems impose stringent requirements on backup and recovery. As a DBA in charge of backup and recovery, you must:

- Manage the complexity of backup and recovery operations
- Minimize the possibility of human error
- Make backups scalable and reliable
- Utilize all available media hardware
- Make backups proportional to the size of transactional changes, not to the size of database
- Make recovery time proportional to the amount of data recovered

User-Managed backup and recovery



Backup and recovery with Recovery Manager



RMAN (Recovery Manager) is a backup and recovery manager supplied for Oracle databases (from version 8) created by the Oracle Corporation. It provides database backup, restore, and recovery capabilities addressing high availability and disaster recovery concerns.

Database migration

Database migration, in simple words, means moving data from one platform to another. There are many reasons we might want to move to a different platform. For example, a company might decide to save money by moving to a cloud-based database. Or a company might find that some particular database software has features that are critical for their business needs. Or the legacy systems are simply outdated. The process of database migration can involve multiple phases and iterations, including assessing the current databases and future needs of the company, migrating the schema, and normalizing and moving the data.

Benefits of Database Migration

Costs. One of the primary reasons that companies migrate databases is to save money. Often, companies will move from an on-premise database to a cloud database. This saves on infrastructure as well as the manpower and expertise needed to support it.

Modernized software. Another common reason for migration is to move from an outdated system or legacy systems to a system that is designed for modern data needs. In the age of big data, new storage techniques are a necessity. For example, a company might choose to move from a legacy SQL database to a data lake or another flexible system.

One source of truth. Another common reason to migrate data is to move all the data into one place that is accessible by all divisions of the company. Sometimes, this occurs after an acquisition, when systems need to be combined. Or, it can happen when different systems are siloed throughout a company. For example, the IT department might use one database while the Marketing group uses another database and these systems cannot "talk" to each other. When we have different databases that are incompatible, it's hard to get insights from our data.

Database Migration Challenges

Database migration can be very complex, but with proper planning, these common challenges can be mitigated:

Challenge #1: Finding our Siloed Databases

If our company has been around a while, we likely have many disparate databases that exist within various parts of our company. They may be in different departments and different geographies. They may have been brought in through acquisitions. Part of our task in migrating databases is to locate the disparate databases in our company and plan how we will normalize data and convert schemas.

Challenge #2: Data Loss or Corruption

When migrating databases, it's critical to ensure our data is safely moved without loss or corruption. We'll need to plan how to test for data loss or corruption that can happen when we move data from one system to another.

Challenge # 3: Security

When we move data from one platform to another, it's critical that the data is secure. Unfortunately, there are many nefarious actors who would love to get their hands on the personal data we have stored away. We might choose to encrypt the data or remove personally identifiable information (PII) as a part of the migration process.

Database Migration Process Phases

Database migration is a multiphase process that involves some or all the following steps:

Assessment. At this stage, we'll need to gather business requirements, assess the costs and benefits, and perform data profiling. Data profiling is a process by which we get to know our existing data and database schema. We'll also need to plan how we will move the data — will we use an ETL (Extraction, Transformation, and Loading) tool, scripting, or some other tool to move the data?

Database schema conversion. The schema is a blueprint of how the database is structured, and it varies based on the rules of a given database. When we move data from one system to another, we'll need to convert the schemas so that the structure of the data works with the new database.

Data migration. After we have completed all the preliminary requirements, we'll need to actually move the data. This may involve scripting or using an ETL tool or some other tool to move the data. During the migration, we will likely transform the data, normalize data types, and check for errors.

Testing and tuning. Once we've moved the data, we need to verify that the data was moved correctly, is complete, isn't missing values, doesn't contain null values, and is valid.

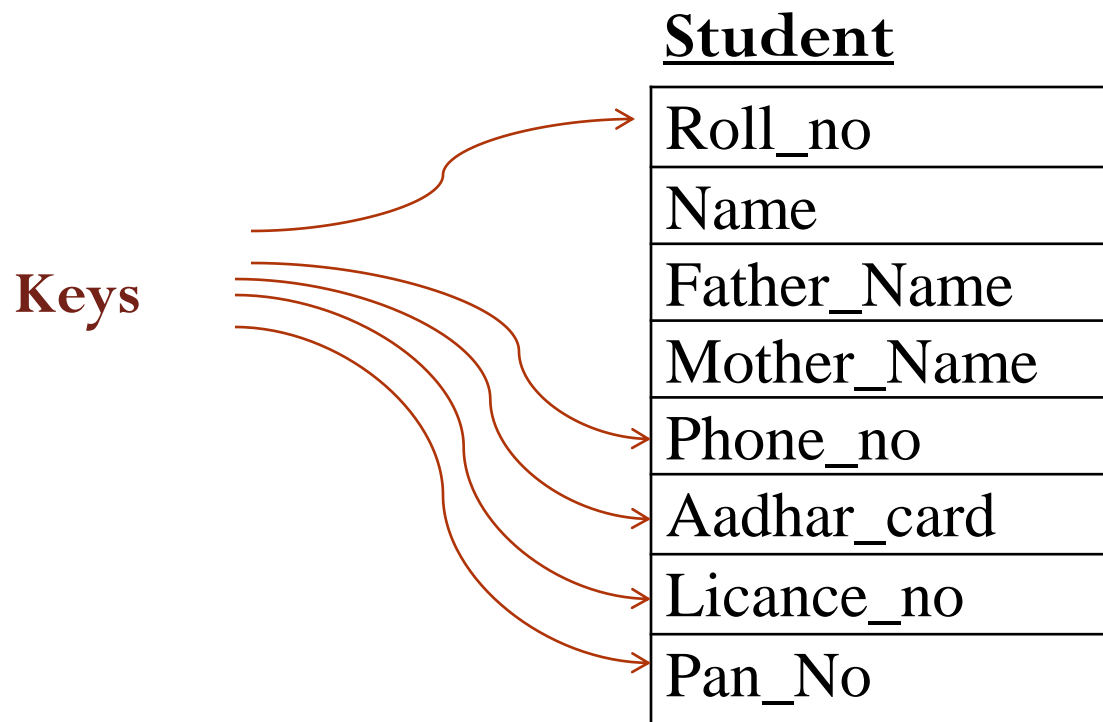
Database Management System



DBMS Keys

DBMS Keys

A DBMS Key is an attribute or collection of attributes that uniquely identifies a record or a row of data in a relation(table).



Types of Keys

Types of Keys

```
graph TD; A[Types of Keys] --> B[Super Key]; A --> C[Candidate Key]; A --> D[Primary Key]; A --> E[Foreign Key];
```

Super Key

**Candidate
Key**

**Primary
Key**

**Foreign
Key**

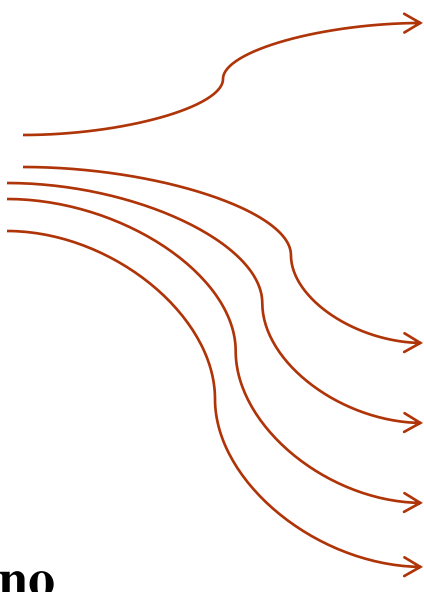
Super Keys

- **Super Key** is an attribute (or set of attributes) that is used to uniquely identifies all attributes in a relation.
- Super Set of keys in which all possible keys are included.
- Every Key in a table is a super key.
- It may have redundant attribute which isn't necessary for uniquely identifying the rows in the table.

Super Key

Roll_no
Aadhar_card
Phone_no
Licance_no
Pan_No
Aadhar card + Phone_no
Roll_no + Pan_No
Aadhar card + Roll_No + Phone_no
Aadhar card + Licance_no + Pan_no
Roll_no + Name

Super Key



The diagram illustrates the concept of a super key by mapping attributes from a list to a table. On the left, a list of attributes and combinations is provided. On the right, a table named 'Student' contains several attributes. Red arrows originate from the list and point to specific attributes in the table: 'Roll_no' points to the 'Roll_no' attribute, 'Aadhar_card' points to the 'Aadhar_card' attribute, 'Phone_no' points to the 'Phone_no' attribute, 'Licance_no' points to the 'Licance_no' attribute, and 'Pan_No' points to the 'Pan_No' attribute. Additionally, a bracket groups the first five attributes in the list ('Roll_no' through 'Pan_No') and points to the 'Roll_no' attribute in the table, indicating that this set of attributes is a super key.

Roll_no
Name
Father_Name
Mother_Name
Phone_no
Aadhar_card
Licance_no
Pan_No

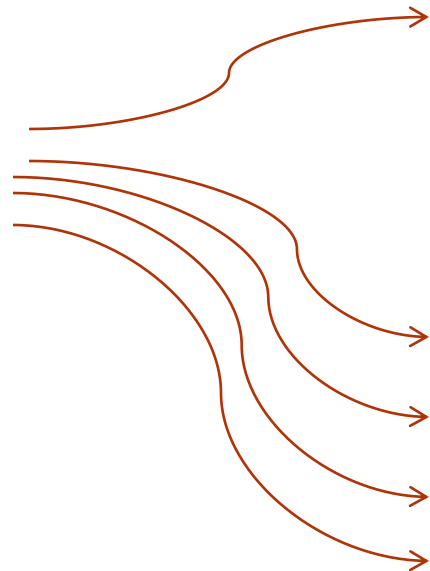
Candidate Key

- Minimal subset of super key that is used to uniquely identifies all attributes in a relation.
- It should not have any redundant attribute.

Candidate Key

Roll_no
Aadhar_card
Phone_no
Licance_no
Pan_No

Candidate Key

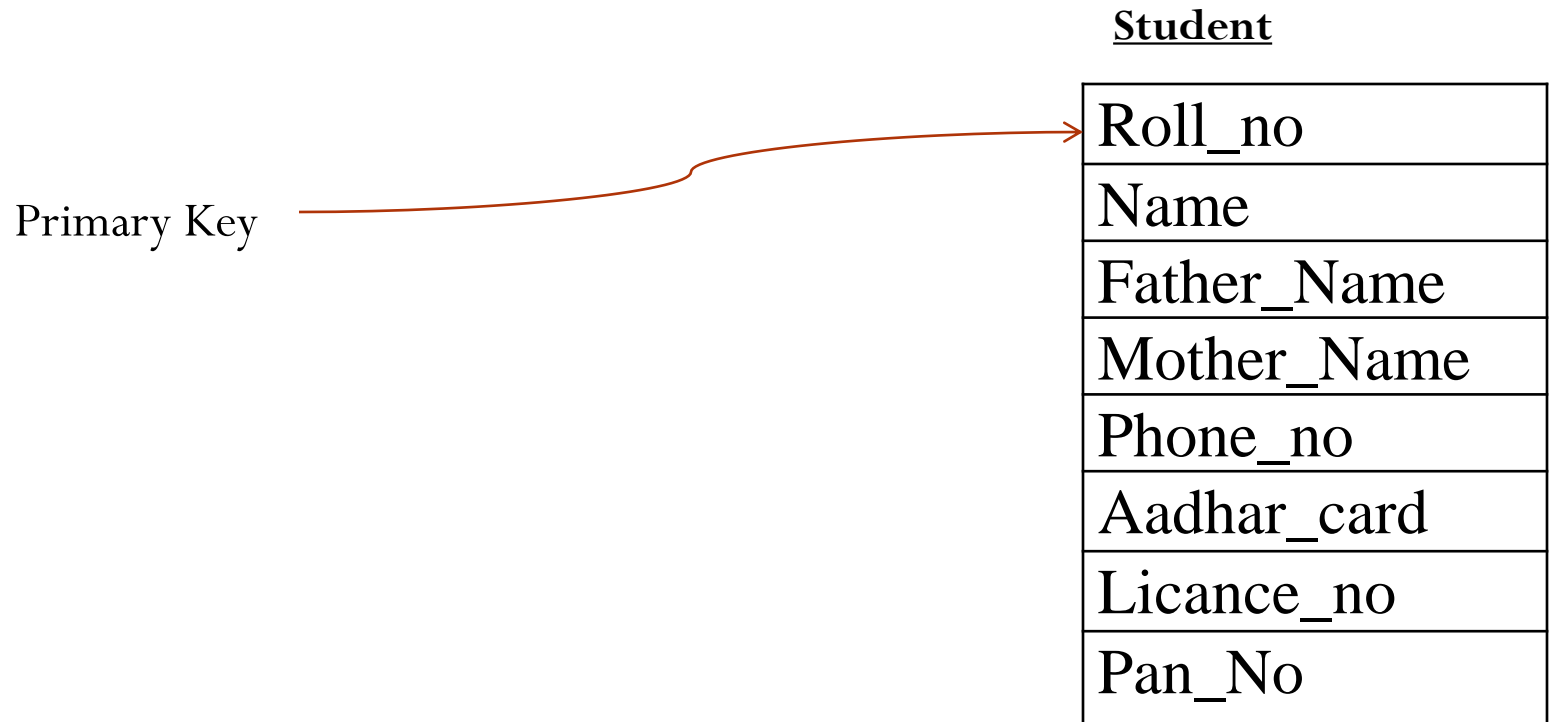


<u>Student</u>	
Roll_no	
Name	
Father_Name	
Mother_Name	
Phone_no	
Aadhar_card	
Licance_no	
Pan_No	

Primary Key

- The candidate chosen to uniquely identify each row of data in a table.
- The value of primary key should be unique and not null.
- Each table has a primary key.
- A table can not have two or more primary key.
- It is generated by the database, not input by the user.

Primary Key



Foreign Key

- It is an attribute in a table which is used to define its relationship with another table.
- Using foreign key helps in maintaining data integrity for tables in a relationship.
- A foreign key is a column or a set of columns in a table whose values correspond to the values of the primary key in another table.

Foreign Key

Student

St_Roll_No	St_Name	St_DOB	Course_Id
156	Ram	15.10.2000	BCA
160	Shyam	02.11.2001	MCA
170	Mohan	05.05.2000	PGDCA
190	Sohan	01.02.2002	B.Tech
108	Geeta	07.08.2001	B.Sc Nursuing



Not Allowed

Course



Course_Id	Course_dept	Course_duration	Course_room
BCA	Computer Application	3	15
MCA	Computer Application	3	30
PGDCA	Computer Application	1	17
B.Tech	Engineering	4	20

Thank You