Database Systems - Lecture notes 1

Database management systems (Jomo Kenyatta University of Agriculture and Technology)

DATABASE SYSTEMS

What is a Database?

A Database is a collection of inter-related data. For Example, a university database organizes the data about students, faculty, and admin staff etc. which helps in efficient retrieval, insertion and deletion of data from it.

Another example is the library. The library contains a huge collection of books of different genres, here the library is database and books are the data. Modern databases are managed using a database management system (DBMS).

Database Management Systems

DBMS stands for Database Management System. We can break it like this DBMS = Database + Management System. Database is a collection of data and Management System is a set of programs to store and retrieve those data. Based on this we can define DBMS like this: DBMS is a collection of inter-related data and set of programs to store & access those data in an easy and effective manner.

What is the need of DBMS?

Database systems are basically developed for large amount of data. When dealing with huge amount of data, there are two things that require optimization: Storage of data and retrieval of data.

Storage: According to the principles of database systems, the data is stored in such a way that it acquires lot less space as the redundant data (duplicate data) has been removed before storage. Let's take a layman example to understand this:

In a banking system, suppose a customer is having two accounts, one is saving account and another is salary account. Let's say bank stores saving account data at one place (these places are called tables we will learn them later) and salary account data at another place, in that case if the customer information such as customer name, address etc. are stored at both places then this is just a wastage of storage (redundancy/ duplication of data), to organize the data in a better way the information should be stored at one place and both the accounts should be linked to that information somehow. The same thing we achieve 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 manage the data. Consider a university that keeps the data of students, teachers, 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 operations on data we need a Database management system that allows us to store the data in such a way so that all these operations can be performed on the data efficiently.

Examples of Database Applications

Applications where we use Database Management Systems are:

1. **Telecom:** There is a database to keeps track of the information regarding calls made, network usage, customer details etc. Without the database systems it is hard to maintain that huge amount of data that keeps updating every millisecond.
2. **Industry:** Where it is a manufacturing unit, warehouse or distribution centre, each one needs a database to keep the records of ins and outs. For example distribution centre should keep a track of the product units that supplied into the centre as well as the products that got delivered out from the distribution centre on each day; this is where DBMS comes into picture.
3. **Banking System:** For storing customer info, tracking day to day credit and debit transactions, generating bank statements etc. All this work has been done with the help of Database management systems.
4. **Sales:** To store customer information, production information and invoice details.
5. **Airlines:** To travel though airlines, we make early reservations, this reservation information along with flight schedule is stored in database.
6. **Education sector:** Database systems are frequently used in schools and colleges to store and retrieve the data regarding student details, staff details, course details, exam details, payroll data, attendance details, fees details etc. There is a hell lot amount of inter-related data that needs to be stored and retrieved in an efficient manner.
7. **Online shopping:** You must be aware of the online shopping websites such as Amazon, Flipkart etc. These sites store the product information, your addresses and preferences, credit details and provide you the relevant list of products based on your query. All this involves a Database management system.

Advantages of DBMS over file system

What is a file processing system?

A **file processing system** is a collection of programs that store and manage files in computer hard-disk. On the other hand, A database management system is collection of programs that enables users to create and maintain a database.

Drawbacks of File system

1. **Data redundancy:** Data redundancy refers to the duplication of data, lets say we are managing the data of a college where a student is enrolled for two courses, the same student details in such case will be stored twice, which will take more storage than needed. Data redundancy often leads to higher storage costs and poor access time.
2. **Data inconsistency:** Data redundancy leads to data inconsistency, lets take the same example that we have taken above, a student is enrolled for two courses and we have student address stored twice, now lets say student requests to change his address, if the address is changed at one place and not on all the records then this can lead to data inconsistency.

3. **Data Isolation:** Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.
4. **Dependency on application programs**: Changing files would lead to change in application programs.
5. **Atomicity issues:** Atomicity of a transaction refers to "All or nothing", which means either all the operations in a transaction executes or none.

For example: Lets say Steve transfers 100$ to Negan's account. This transaction consists multiple operations such as debit 100$ from Steve's account, credit 100$ to Negan's account. Like any other device, a computer system can fail lets say it fails after first operation then in that case Steve's account would have been debited by 100$ but the amount was not credited to Negan's account, in such case the rollback of operation should occur to maintain the atomicity of transaction. It is difficult to achieve atomicity in file processing systems.

6. **Data Security:** Data should be secured from unauthorised access, for example a student in a college should not be able to see the payroll details of the teachers, such kind of security constraints are difficult to apply in file processing systems.

Advantage of DBMS over file system

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

No redundant data: Redundancy removed by data normalization. No data duplication saves storage and improves access time.

Data Consistency and Integrity: As we discussed earlier the root cause of data inconsistency is data redundancy, since data normalization takes care of the data redundancy, data inconsistency also been taken care of as part of it

Data Security: It is easier to apply access constraints in database systems so that only authorized user is able to access the data. Each user has a different set of access thus data is secured from the issues such as identity theft, data leaks and misuse of data.

Privacy: Limited access means privacy of data.

Easy access to data – Database systems manages data in such a way so that the data is easily accessible with fast response times.

Easy recovery: Since database systems keeps the backup of data, it is easier to do a full recovery of data in case of a failure.

Flexible: Database systems are more flexible than file processing systems.

Disadvantages of DBMS:

- DBMS implementation cost is high compared to the file system
- Complexity: Database systems are complex to understand
- Performance: Database systems are generic, making them suitable for various applications. However this feature affect their performance for some applications

DBMS Architecture

The architecture of DBMS depends on the computer system on which it runs. For example, in a client-server DBMS architecture, the database systems at server machine can run several requests made by client machine.

Types of DBMS Architecture

There are three types of DBMS architecture:

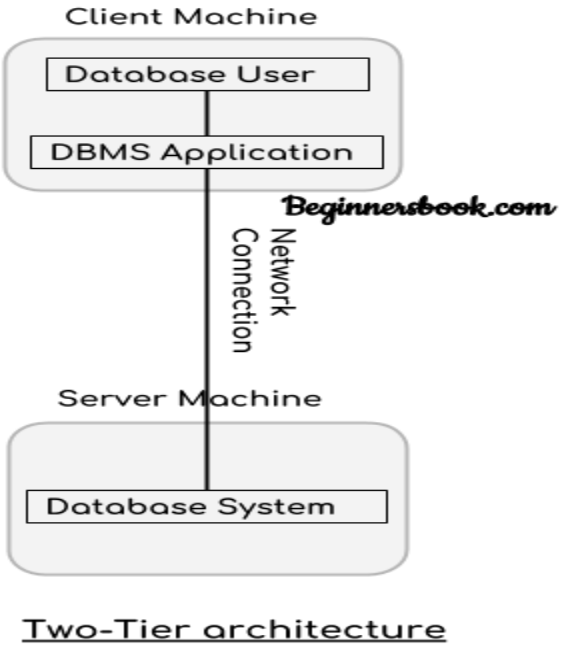
1. Single tier architecture
2. Two tier architecture
3. Three tier architecture

1. Single tier architecture

In this type of architecture, the database is readily available on the client machine, any request made by client doesn't require a network connection to perform the action on the database.

For example, lets say you want to fetch the records of employee from the database and the database is available on your computer system, so the request to fetch employee details will be done by your computer and the records will be fetched from the database by your computer as well. This type of system is generally referred as local database system.

2. Two tier architecture

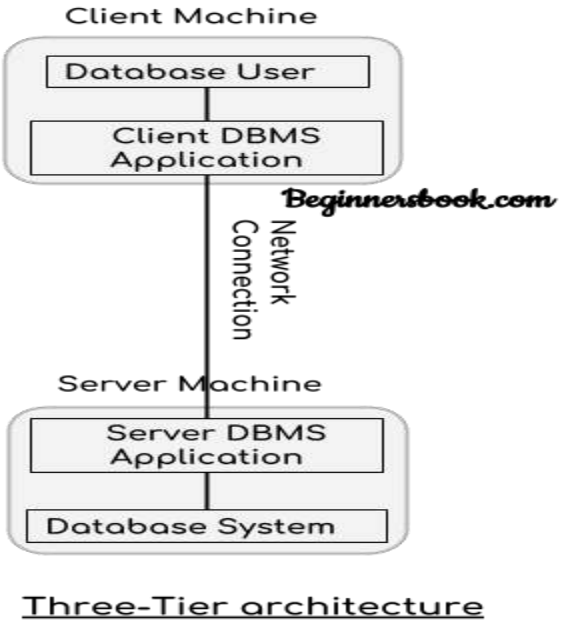


Two-Tier architecture

In two-tier architecture, the Database system is present at the server machine and the DBMS application is present at the client machine, these two machines are connected with each other through a reliable network as shown in the above diagram.

Whenever client machine makes a request to access the database present at server using a query language like sql, the server perform the request on the database and returns the result back to the client. The application connection interface such as JDBC, ODBC are used for the interaction between server and client.

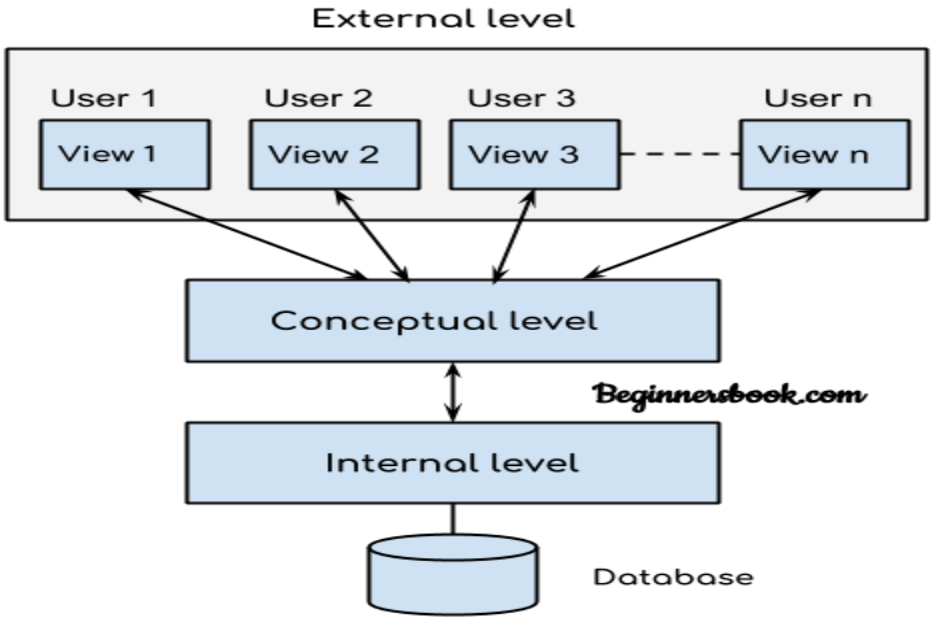
3. Three tier architecture



Three-Tier architecture

In three-tier architecture, another layer is present between the client machine and server machine. In this architecture, the client application doesn't communicate directly with the database systems present at the server machine, rather the client application communicates with server application and the server application internally communicates with the database system present at the server.

DBMS: Three Level Architecture Diagram



This architecture has three levels:
1. External level
2. Conceptual level
3. Internal level

1. External level

It is also called **view level**. The reason this level is called “view” is because several users can view their desired data from this level which is internally fetched from database with the help of conceptual and internal level mapping.

The user doesn’t need to know the database schema details such as data structure, table definition etc. user is only concerned about data which is what returned back to the view level after it has been fetched from database (present at the internal level).

External level is the **“top level”** of the Three Level DBMS Architecture.

2. Conceptual level

It is also called **logical level**. The whole design of the database such as relationship among data, schema of data etc. are described in this level.

Database constraints and security are also implemented in this level of architecture. This level is maintained by DBA (database administrator).

3. Internal level

This level is also known as physical level. This level describes how the data is actually stored in the storage devices. This level is also responsible for allocating space to the data. This is the lowest level of the architecture.

View of Data in DBMS

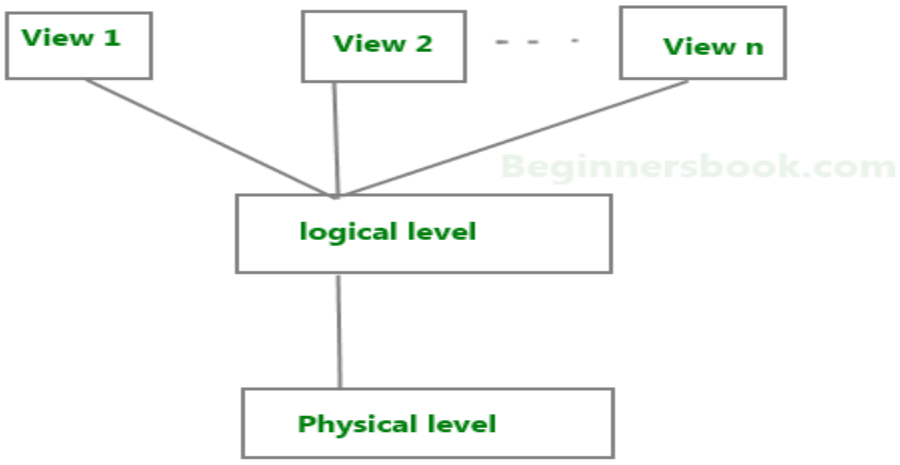
Abstraction is one of the main features of database systems. Hiding irrelevant details from user and providing abstract view of data to users, helps in easy and efficient user-database interaction.

The top level of DBMS architecture also known as the “view level” provides the **“view of data”** to the users and hides the irrelevant details such as data relationship, database schema, constraints, security etc from the user.

To fully understand the view of data, we must have a basic knowledge of data abstraction and instance & schema.

Data Abstraction in DBMS

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



Three Levels of data abstraction

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

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

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

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

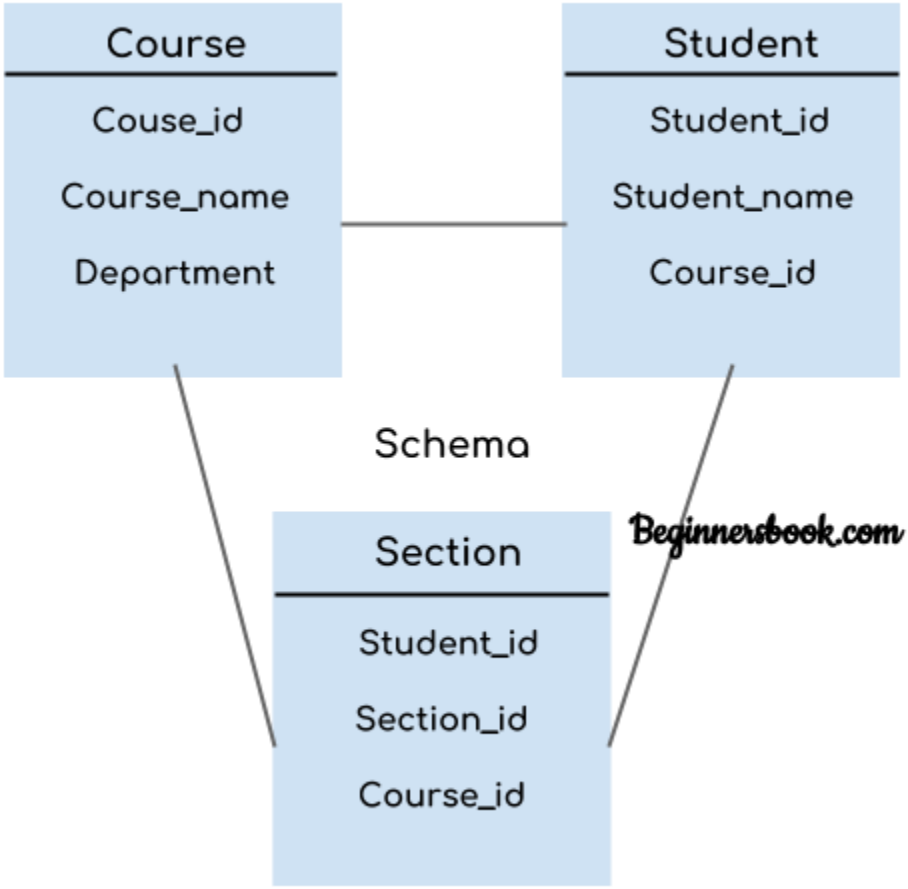
At the **logical level** these records can be described as fields and attributes along with their data types, their relationship among each other can be logically implemented. The programmers generally work at this level because they are aware of such things about database systems.

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

DBMS Schema

Definition of schema: Design of a database is called the schema. Schema is of three types: Physical schema, logical schema and view schema.

For example: In the following diagram, we have a schema that shows the relationship between three tables: Course, Student and Section. The diagram only shows the design of the database, it doesn't show the data present in those tables. Schema is only a structural view(design) of a database as shown in the diagram below.



The design of a database at physical level is called **physical schema,** how the data stored in blocks of storage is described at this level.

Design of database at logical level is called **logical schema,** programmers and database administrators work at this level, at this level data can be described as certain types of data records gets stored in data structures, however the internal details such as implementation of data structure is hidden at this level (available at physical level).

Design of database at view level is called **view schema**. This generally describes end user interaction with database systems.

DBMS Instance

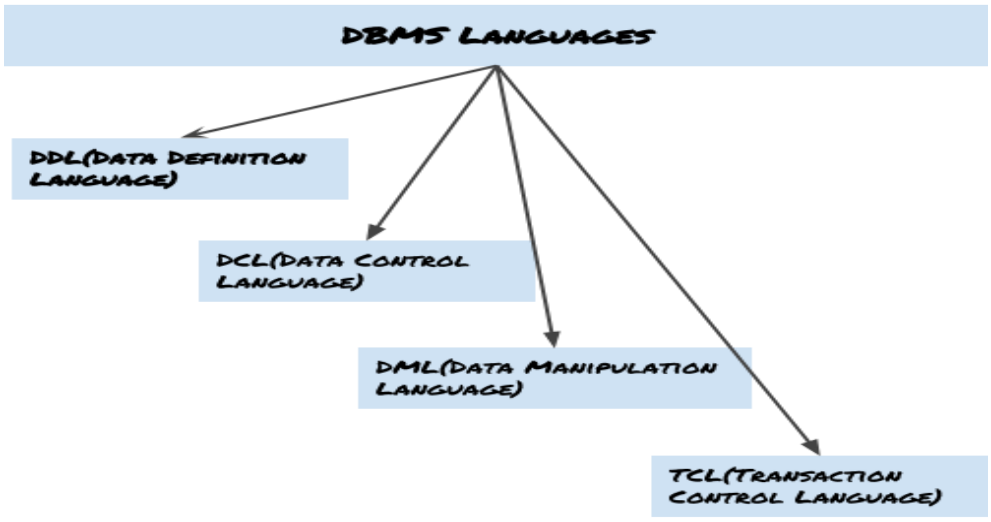
Definition of instance: The data stored in database at a particular moment of time is called instance of database. Database schema defines the variable declarations in tables that belong to a particular database; the value of these variables at a moment of time is called the instance of that database.

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

DBMS languages

Database languages are used to read, update and store data in a database. There are several such languages that can be used for this purpose; one of them is SQL (Structured Query Language).

Types of DBMS languages:



Data Definition Language (DDL)

DDL is used for specifying the database schema. It is used for creating tables, schema, indexes, constraints etc. in database. Lets see the operations that we can perform on database using DDL:

- To create the database instance – **CREATE**
- To alter the structure of database – **ALTER**
- To drop database instances – **DROP**
- To delete tables in a database instance – **TRUNCATE**
- To rename database instances – **RENAME**
- To drop objects from database such as tables – **DROP**
- To Comment – **Comment**

All of these commands either defines or update the database schema that's why they come under Data Definition language.

Data Manipulation Language (DML)

DML is used for accessing and manipulating data in a database. The following operations on database comes under DML:

- To read records from table(s) – **SELECT**
- To insert record(s) into the table(s) – **INSERT**
- Update the data in table(s) – **UPDATE**
- Delete all the records from the table – **DELETE**

Data Control language (DCL)

DCL is used for granting and revoking user access on a database –

- To grant access to user – **GRANT**
- To revoke access from user – **REVOKE**

In practical data definition language, data manipulation language and data control languages are not separate language, rather they are the parts of a single database language such as SQL.

Transaction Control Language(TCL)

The changes in the database that we made using DML commands are either performed or rollbacked using TCL.

- To persist the changes made by DML commands in database – **COMMIT**
- To rollback the changes made to the database – **ROLLBACK**

Data models in DBMS

Data models in DBMS

Data Model is a logical structure of Database. It describes the design of database to reflect entities, attributes, relationship among data, constrains etc.

Types of Data Models

There are several types of data models in DBMS. We will cover them in detail in separate articles(Links to those separate tutorials are already provided below). In this guide, we will just see a basic overview of types of models.

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

1. E-R Model
2. Object oriented Model

Record based logical Models – Like Object based model, they also describe data at the conceptual and view levels. These models specify logical structure of database with records, fields and attributes.

1. Relational Model
2. Hierarchical Model
3. Network Model – Network Model is same as hierarchical model except that it has graph-like structure rather than a tree-based structure. Unlike hierarchical model, this model allows each record to have more than one parent record.

Physical Data Models – These models describe data at the lowest level of abstraction.

Entity Relationship Diagrams – ER Diagram in DBMS

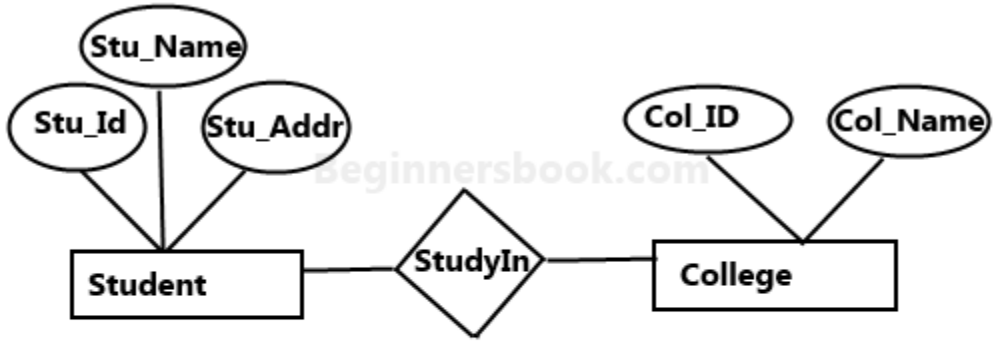
An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: **entity set and relationship set.**

What is an Entity Relationship Diagram (ER Diagram)?

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. Lets have a look at a simple ER diagram to understand this concept.

A simple ER Diagram:

In the following diagram we have two entities Student and College and their relationship. The relationship between Student and College is many to one as a college can have many students however a student cannot study in multiple colleges at the same time. Student entity has attributes such as Stu_Id, Stu_Name & Stu_Addr and College entity has attributes such as Col_ID & Col_Name.

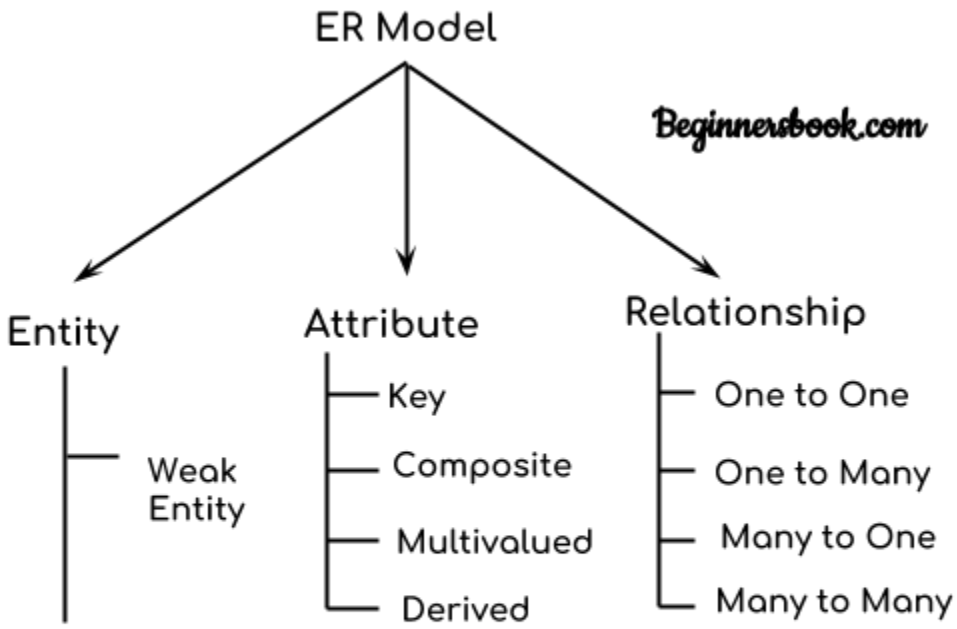


Sample E-R Diagram

Here are the geometric shapes and their meaning in an E-R Diagram. We will discuss these terms in detail in the next section(Components of a ER Diagram) of this guide so don't worry too much about these terms now, just go through them once.

Rectangle**: Represents Entity sets.**
Ellipses**: Attributes**
Diamonds**: Relationship Set**
Lines**: They link attributes to Entity Sets and Entity sets to Relationship Set**
Double Ellipses: **Multivalued Attributes**
Dashed Ellipses**: Derived Attributes**
Double Rectangles**: Weak Entity Sets**
Double Lines**: Total participation of an entity in a relationship set**

Components of a ER Diagram



Components of ER Diagram

As shown in the above diagram, an ER diagram has three main components:
1. Entity
2. Attribute
3. Relationship

1. Entity

An entity is an object or component of data. An entity is represented as rectangle in an ER diagram.
For example: In the following ER diagram we have two entities Student and College and these two entities have many to one relationship as many students study in a single college. We will read more about relationships later, for now focus on entities.

Beginnersbook.com

Weak Entity:
An entity that cannot be uniquely identified by its own attributes and relies on the relationship with other entity is called weak entity. The weak entity is represented by a double rectangle. For example – a bank account cannot be uniquely identified without knowing the bank to which the account belongs, so bank account is a weak entity.

Beginnersbook.com

2. Attribute

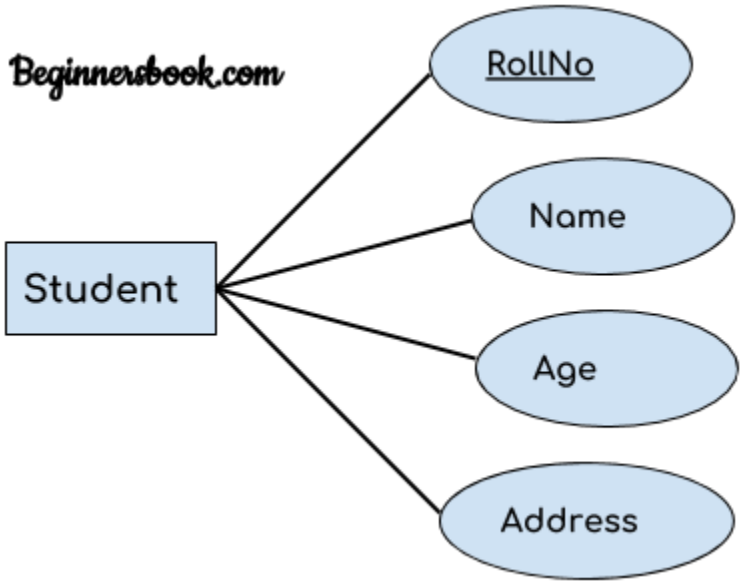
An attribute describes the property of an entity. An attribute is represented as Oval in an ER diagram. There are four types of attributes:

1. Key attribute
2. Composite attribute
3. Multivalued attribute
4. Derived attribute

1. Key attribute:

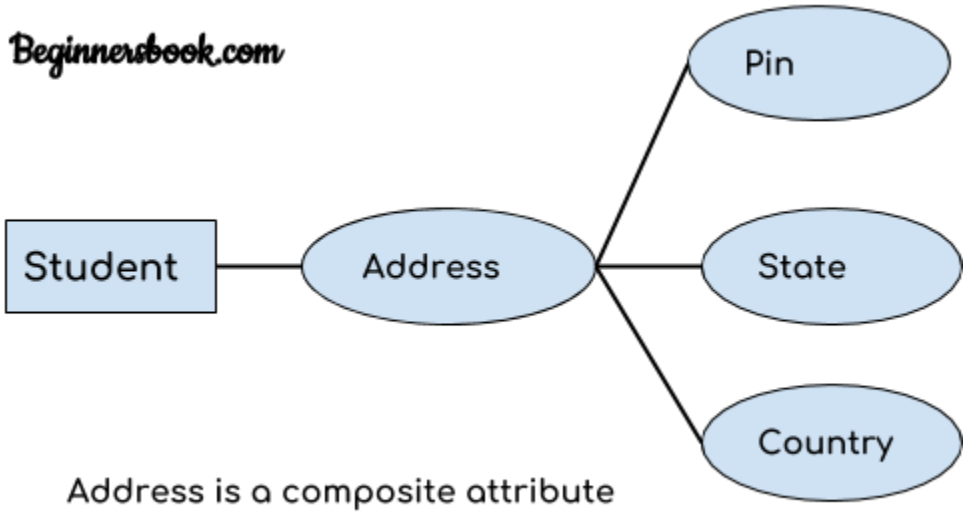
A key attribute can uniquely identify an entity from an entity set. For example, student roll number can uniquely identify a student from a set of students. Key attribute is represented by

oval same as other attributes however the text of key attribute is underlined.



2. Composite attribute:

An attribute that is a combination of other attributes is known as composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other attributes such as pin code, state, country.



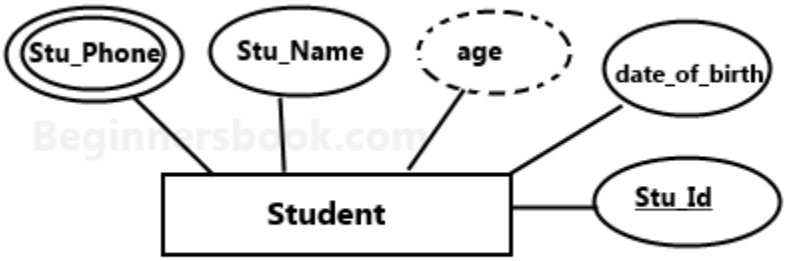
3. Multivalued attribute:

An attribute that can hold multiple values is known as multivalued attribute. It is represented with double ovals in an ER Diagram. For example – A person can have more than one phone numbers so the phone number attribute is multivalued.

4. Derived attribute:

A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by dashed oval in an ER Diagram. For example – Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).

E-R diagram with multivalued and derived attributes:

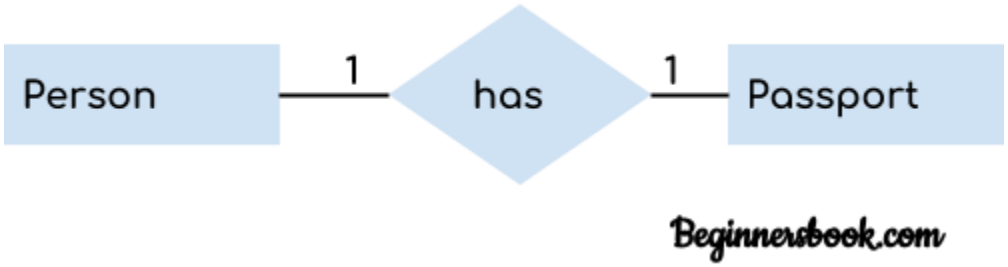


3. Relationship

A relationship is represented by diamond shape in ER diagram, it shows the relationship among entities. There are four types of relationships:
1. One to One
2. One to Many
3. Many to One
4. Many to Many

1. One to One Relationship

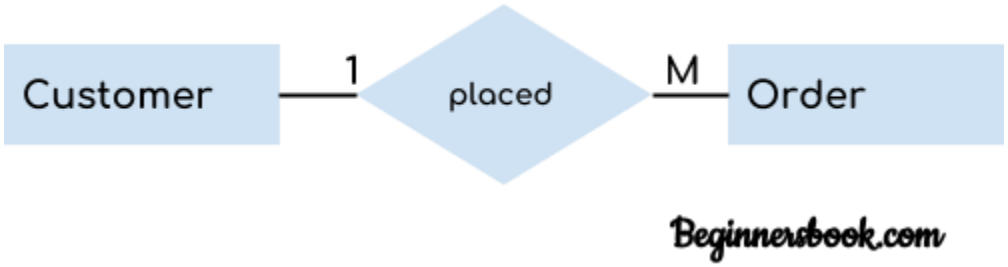
When a single instance of an entity is associated with a single instance of another entity then it is called one to one relationship. For example, a person has only one passport and a passport is given to one person.



2. One to Many Relationship

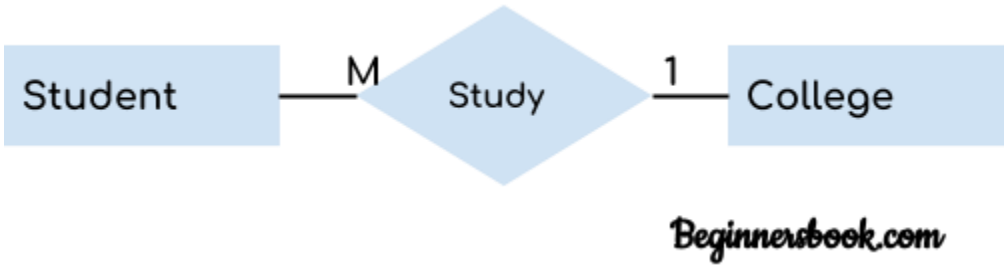
When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationship. For example – a customer can place many orders but a

order cannot be placed by many customers.



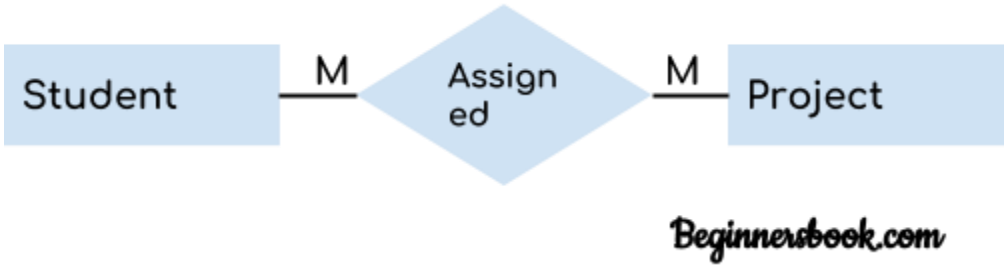
3. Many to One Relationship

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



4. Many to Many Relationship

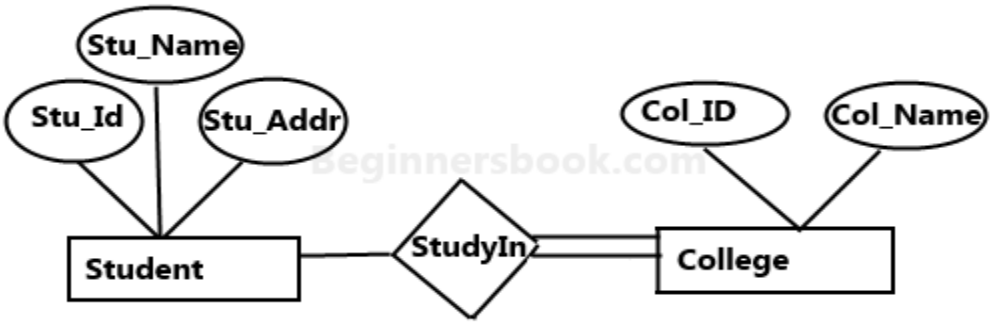
When more than one instances of an entity is associated with more than one instances of another entity then it is called many to many relationship. For example, a can be assigned to many projects and a project can be assigned to many students.



Total Participation of an Entity set

A Total participation of an entity set represents that each entity in entity set must have at least one relationship in a relationship set. For example: In the below diagram each college must have

at-least one associated Student.



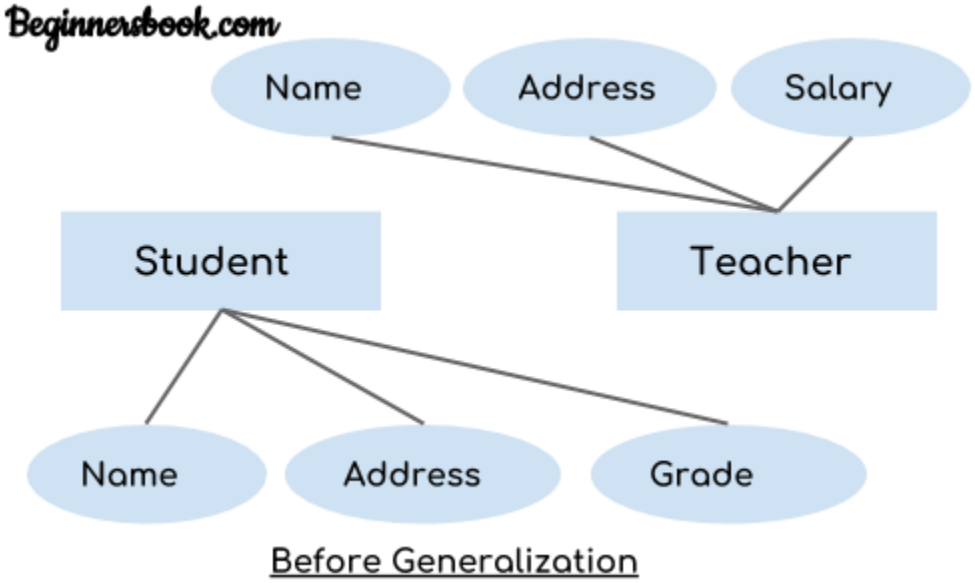
E-R Digram with total participation of College entity set in StudyIn relationship Set - This indicates that each college must have atleast one associated Student.

DBMS Generalization

Generalization is a process in which the common attributes of more than one entities form a new entity. This newly formed entity is called generalized entity.

Generalization Example
Lets say we have two entities Student and Teacher.
Attributes of Entity Student are: Name, Address & Grade
Attributes of Entity Teacher are: Name, Address & Salary

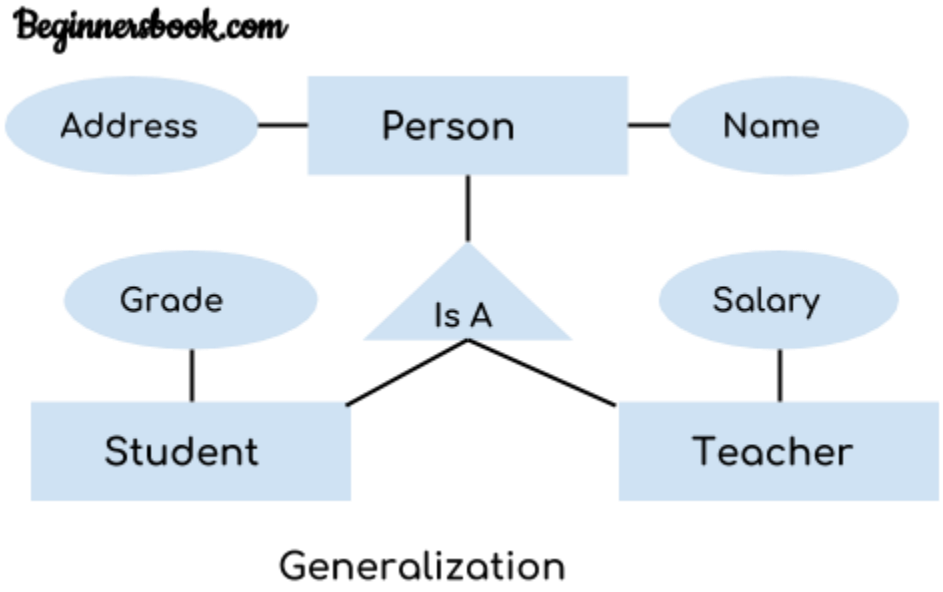
The ER diagram before generalization looks like this:



Before Generalization

These two entities have two common attributes: Name and Address, we can make a generalized entity with these common attributes. Lets have a look at the ER model after generalization.

The ER diagram after generalization:
We have created a new generalized entity Person and this entity has the common attributes of both the entities. As you can see in the following ER diagram that after the generalization process the entities Student and Teacher only has the specialized attributes Grade and Salary respectively and their common attributes (Name & Address) are now associated with a new entity Person which is in the relationship with both the entities (Student & Teacher).



Generalization

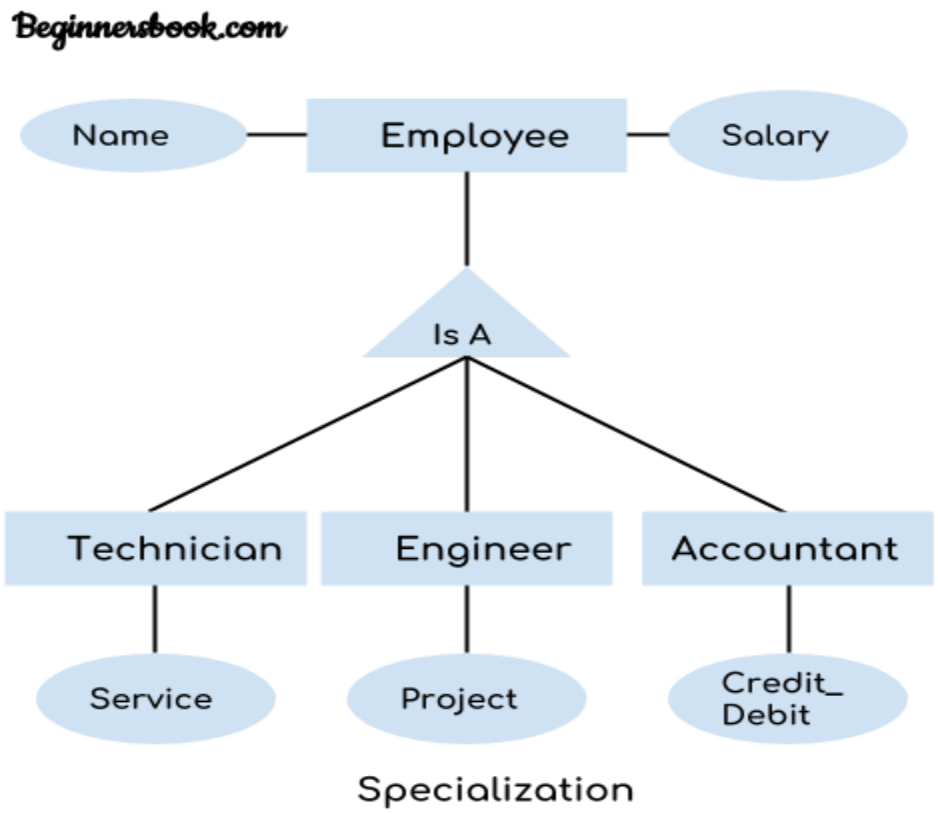
Note:
1. Generalization uses bottom-up approach where two or more lower level entities combine together to form a higher level new entity.
2. The new generalized entity can further combine together with lower level entity to create a further higher level generalized entity.

DBMS Specialization

Specialization is a process in which an entity is divided into sub-entities. You can think of it as a reverse process of generalization, in generalization two entities combine together to form a new higher level entity. Specialization is a top-down process.

The idea behind Specialization is to find the subsets of entities that have few distinguish attributes. For example – Consider an entity employee which can be further classified as sub-entities Technician, Engineer & Accountant because these sub entities have some distinguish **attributes.**

Specialization Example

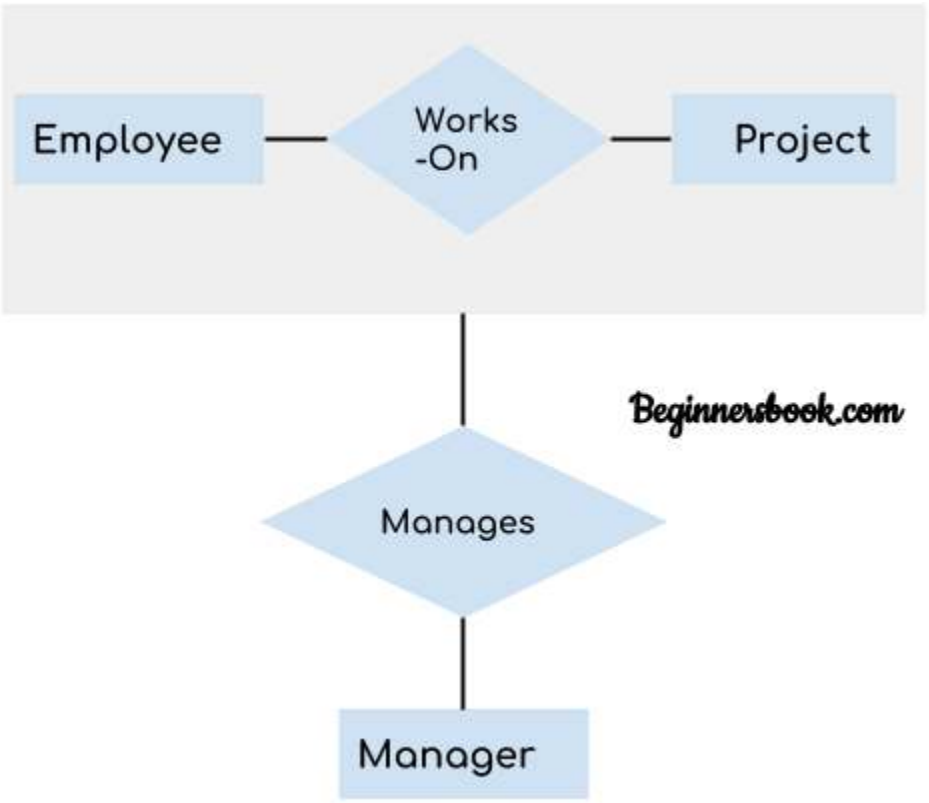


In the above diagram, we can see that we have a higher level entity “Employee” which we have divided in sub entities “Technician”, “Engineer” & “Accountant”. All of these are just an employee of a company, however their role is completely different and they have few different attributes. Just for the example, I have shown that Technician handles service requests, Engineer works on a project and Accountant handles the credit & debit details. All of these three employee types have few attributes common such as name & salary which we had left associated with the parent entity “Employee” as shown in the above diagram.

DBMS Aggregation

Aggregation is a process in which a single entity alone is not able to make sense in a relationship so the relationship of two entities acts as one entity. I know it sounds confusing but don’t worry the example we will take, will clear all the doubts.

Aggregation Example



In real world, we know that a manager not only manages the employee working under them but he has to manage the project as well. In such scenario if entity “Manager” makes a “manages” relationship with either “Employee” or “Project” entity alone then it will not make any sense because he has to manage both. In these cases the relationship of two entities acts as one entity. In our example, the relationship “Works-On” between “Employee” & “Project” acts as one entity that has a relationship “Manages” with the entity “Manager”.

Relational model in DBMS

In relational model, the data and relationships are represented by collection of inter-related tables. Each table is a group of column and rows, where column represents attribute of an entity and rows represents records.

Sample relationship Model: Student table with 3 columns and four records.

Table: Student

Stu_Id	Stu_Name	Stu_Age
111	Ashish	23
123	Saurav	22
169	Lester	24
234	Lou	26

Table: Course

Stu_Id	Course_Id	Course_Name
111	C01	Science
111	C02	DBMS
169	C22	Java
169	C39	Computer Networks

Here Stu_Id, Stu_Name & Stu_Age are attributes of table Student and Stu_Id, Course_Id & Course_Name are attributes of table Course. The rows with values are the records (commonly known as tuples).

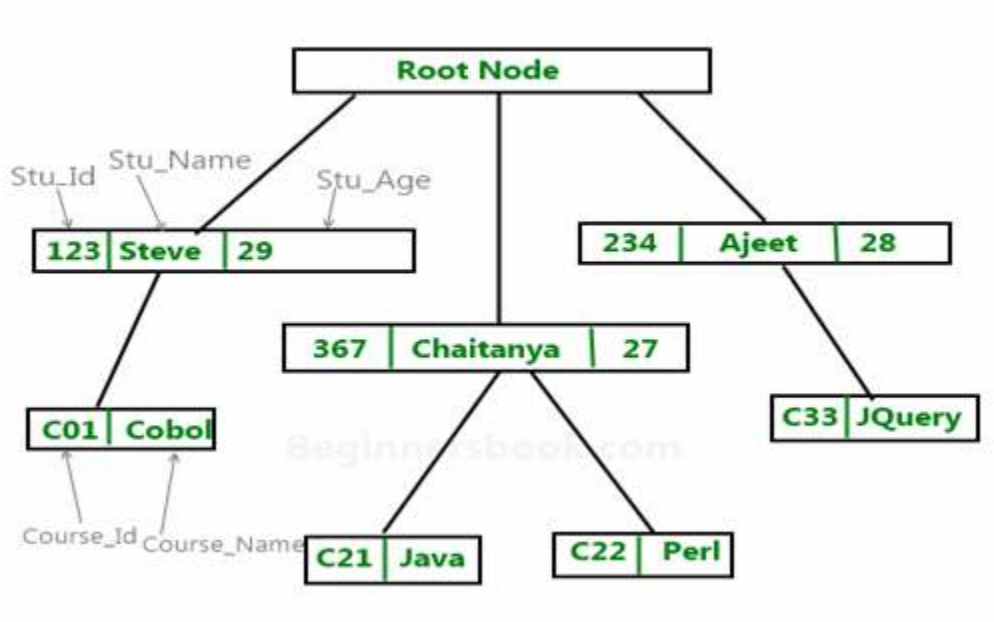
Hierarchical model in DBMS

In **hierarchical model,** data is organized into a tree like structure with each record is having one parent record and many children. The main drawback of this model is that, it can have only one to many relationships between nodes.

Note: Hierarchical models are rarely used now.

Sample Hierarchical Model Diagram:
Lets say we have few students and few courses and a course can be assigned to a single student

only, however a student take any number of courses so this relationship becomes one to many.



Example of hierarchical data represented as relational tables: The above hierarchical model can be represented as relational tables like this:

Stu_Id	Stu_Name	Stu_Age
123	Steve	29
367	Chaitanya	27
234	Ajeet	28

Course Table:

Course_Id	Course_Name	Stu_Id
C01	Cobol	123
C21	Java	367
C22	Perl	367
C33	JQuery	234

Constraints in DBMS

Constraints enforce limits to the data or type of data that can be inserted/updated/deleted from a table. The whole purpose of constraints is to maintain the data integrity during an update/delete/insert into a table. In this tutorial we will learn several types of constraints that can be created in RDBMS.

Types of constraints

- NOT NULL
- UNIQUE
- DEFAULT
- CHECK
- Key Constraints – PRIMARY KEY, FOREIGN KEY
- Domain constraints
- Mapping constraints

NOT NULL:
NOT NULL constraint makes sure that a column does not hold NULL value. When we don't provide value for a particular column while inserting a record into a table, it takes NULL value by default. By specifying NULL constraint, we can be sure that a particular column(s) cannot have NULL values.

Example:

```
CREATE TABLE STUDENT(
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (235),
PRIMARY KEY (ROLL_NO)
);
```

UNIQUE:
UNIQUE Constraint enforces a column or set of columns to have unique values. If a column has a unique constraint, it means that particular column cannot have duplicate values in a table.

```
CREATE TABLE STUDENT(
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL UNIQUE,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (35) UNIQUE,
PRIMARY KEY (ROLL_NO)
);
```

DEFAULT:

The DEFAULT constraint provides a default value to a column when there is no value provided while inserting a record into a table.

```
CREATE TABLE STUDENT(
ROLL_NO   INT  NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
EXAM_FEE INT  DEFAULT 10000,
STU_ADDRESS VARCHAR (35) ,
PRIMARY KEY (ROLL_NO)
);
```

CHECK:

This constraint is used for specifying range of values for a particular column of a table. When this constraint is being set on a column, it ensures that the specified column must have the value falling in the specified range.

```
CREATE TABLE STUDENT(
ROLL_NO   INT  NOT NULL CHECK(ROLL_NO >1000) ,
STU_NAME VARCHAR (35)  NOT NULL,
STU_AGE INT  NOT NULL,
EXAM_FEE INT DEFAULT 10000,
STU_ADDRESS VARCHAR (35) ,
PRIMARY KEY (ROLL_NO)
);
```

In the above example we have set the check constraint on ROLL_NO column of STUDENT table. Now, the ROLL_NO field must have the value greater than 1000.

Key constraints:

PRIMARY KEY:

Primary key uniquely identifies each record in a table. It must have unique values and cannot contain nulls. In the below example the ROLL_NO field is marked as primary key, that means the ROLL_NO field cannot have duplicate and null values.

```
CREATE TABLE STUDENT(
ROLL_NO   INT  NOT NULL,
STU_NAME VARCHAR (35)  NOT NULL UNIQUE,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (35) UNIQUE,
PRIMARY KEY (ROLL_NO)
);
```

FOREIGN KEY:

Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

Domain constraints:

Each table has certain set of columns and each column allows a same type of data, based on its data type. The column does not accept values of any other data type.

Domain constraints are user defined data type and we can define them like this:

Domain Constraint = data type + Constraints (NOT NULL / UNIQUE / PRIMARY KEY / FOREIGN KEY / CHECK / DEFAULT)

Cardinality in DBMS

In DBMS you may hear cardinality term at two different places and it has two different meanings as well.

In Context of Data Models:

In terms of data models, cardinality refers to the relationship between two tables. Relationship can be of four types as we have already seen in Entity relationship guide:

One to One – A single row of first table associates with single row of second table. For example, a relationship between person and passport table is one to one because a person can have only one passport and a passport can be assigned to only one person.

One to Many – A single row of first table associates with more than one rows of second table. For example, relationship between customer and order table is one to many because a customer can place many orders but a order can be placed by a single customer alone.

Many to One – Many rows of first table associate with a single row of second table. For example, relationship between student and university is many to one because a university can have many students but a student can only study only in single university at a time.

Many to Many – Many rows of first table associate with many rows of second table. For example, relationship between student and course table is many to many because a student can take many courses at a time and a course can be assigned to many students.

In Context of Query Optimization:

In terms of query, the cardinality refers to the uniqueness of a column in a table. The column with all unique values would be having the high cardinality and the column with all duplicate values would be having the low cardinality. These cardinality scores helps in query optimization.

RELATIONAL DATABASES

RDBMS Concepts

RDBMS stands for relational database management system. A relational model can be represented as a table of rows and columns. A relational database has following major components:

1. Table
2. Record or Tuple
3. Field or Column name or Attribute
4. Domain
5. Instance
6. Schema
7. Keys

1. Table

A table is a collection of data represented in rows and columns. Each table has a name in database. 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. Field or Column name or Attribute

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 has values like, “First”, 10.11 etc.

5. Instance and Schema

I have already covered instance and schema.

6. Keys

We shall cover this in our next topic.

Introduction to Relational algebra & Relational calculus

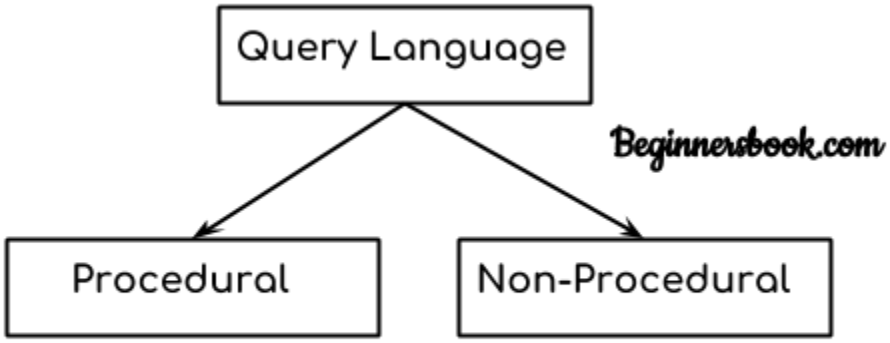
In this topic, we will discuss what is **Relational algebra and relational calculus** and why we use these concepts. In the previous topics, we discussed the designing of database using Relational model, E-R diagram and normalization. Now that we have designed the database, we need to store and retrieve data from the database, for this purpose we need to understand the concept of Relational algebra and relational calculus.

Let’s start with the basics.

Query Language

In simple words, a Language which is used to store and retrieve data from database is known as query language. For example – SQL

There are two types of query language:
1.Procedural Query language
2.Non-procedural query language



1. Procedural Query language:

In procedural query language, user instructs the system to perform a series of operations to produce the desired results. Here users tells what data to be retrieved from database and how to retrieve it.

For example – Let's take a real world example to understand the procedural language, you are asking your younger brother to make a cup of tea, if you are just telling him to make a tea and not telling the process then it is a non-procedural language, however if you are telling the step by step process like switch on the stove, boil the water, add milk etc. then it is a procedural language.

2. Non-procedural query language:

In Non-procedural query language, user instructs the system to produce the desired result without telling the step by step process. Here users tells what data to be retrieved from database but doesn't tell how to retrieve it.

Now let's back to our main topic of relational algebra and relational calculus.

Relational Algebra:

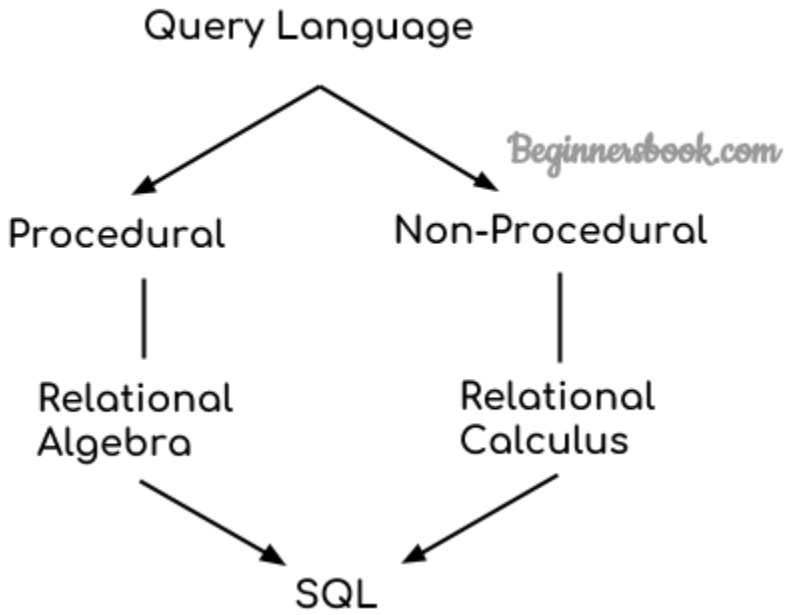
Relational algebra is a conceptual procedural query language used on relational model.

Relational Calculus:

Relational calculus is a conceptual non-procedural query language used on relational model.

Note:
I have used word conceptual while describing relational algebra and relational calculus, because

they are theoretical mathematical system or query language, they are not the practical implementation, SQL is a practical implementation of relational algebra and relational calculus.



Relational Algebra, Calculus, RDBMS & SQL:

Relational algebra and calculus are the theoretical concepts used on relational model.

RDBMS is a practical implementation of relational model.

SQL is a practical implementation of relational algebra and calculus.

Next we will cover the relational algebra and calculus in detail.

DBMS Relational Algebra

What is Relational Algebra in DBMS?

Relational algebra is a procedural query language that works on relational model. The purpose of a query language is to retrieve data from database or perform various operations such as insert, update, delete on the data. When I say that relational algebra is a procedural query language, it means that it tells what data to be retrieved and how to be retrieved.

On the other hand relational calculus is a non-procedural query language, which means it tells what data to be retrieved but doesn't tell how to retrieve it. We will discuss relational calculus in a separate tutorial.

Types of operations in relational algebra

We have divided these operations in two categories:
1. Basic Operations
2. Derived Operations

Basic/Fundamental Operations:

1. Select (σ)
2. Project (∏)
3. Union (∪)
4. Set Difference (-)
5. Cartesian product (X)
6. Rename (ρ)

Derived Operations:

1. Natural Join (⋈)
2. Left, Right, Full outer join (⟕, ⟖, ⟗)
3. Intersection (∩)
4. Division (÷)

Lets discuss these operations one by one with the help of examples.

1. Select Operator (σ)

Select Operator is denoted by sigma (σ) and it is used to find the tuples (or rows) in a relation (or table) which satisfy the given condition.

If you understand little bit of SQL then you can think of it as a where clause in SQL, which is used for the same purpose.

Syntax of Select Operator (σ)

```
σ Condition/Predicate(Relation/Table name)
```

Select Operator (σ) Example

```
Table: CUSTOMER
---------------

Customer_Id     Customer_Name      Customer_City
-----------     -------------      -------------
C10100           Steve              Agra
C10111           Raghu              Agra
C10115           Chaitanya          Noida
C10117           Ajeet              Delhi
C10118           Carl               Delhi
```

Query:

```
σ Customer_City="Agra" (CUSTOMER)
```

Output:

```
Customer_Id   Customer_Name    Customer_City
-----------   -------------    -------------
C10100        Steve            Agra
C10111        Raghu            Agra
```

2. Project Operator (∏)

Project operator is denoted by ∏ symbol and it is used to select desired columns (or attributes) from a table (or relation).

Project operator in relational algebra is similar to the Select statement in SQL.

Syntax of Project Operator (∏)

```
∏ column_name1, column_name2, ...., column_nameN(table_name)
```

Project Operator (∏) Example

In this example, we have a table CUSTOMER with three columns, we want to fetch only two columns of the table, which we can do with the help of Project Operator ∏.

```
Table: CUSTOMER

Customer_Id     Customer_Name      Customer_City
-----------     -------------      -------------
```

```
C10100          Steve             Agra
C10111          Raghu             Agra
C10115          Chaitanya         Noida
C10117          Ajeet             Delhi
C10118          Carl              Delhi
```

Query:

```
∏ Customer_Name, Customer_City (CUSTOMER)
```

Output:

```
Customer_Name      Customer_City
-------------      -------------
Steve              Agra
Raghu              Agra
Chaitanya          Noida
Ajeet              Delhi
Carl               Delhi
```

3. Union Operator (∪)

Union operator is denoted by ∪symbol and it is used to select all the rows (tuples) from two tables (relations).

Lets discuss union operator a bit more. Lets say we have two relations R1 and R2 both have same columns and we want to select all the tuples(rows) from these relations then we can apply the union operator on these relations.

Note: The rows (tuples) that are present in both the tables will only appear once in the union set. In short you can say that there are no duplicates present after the union operation.

Syntax of Union Operator (∪)

```
table_name1 ∪ table_name2
```

Union Operator (∪) Example

Table 1: COURSE

```
Course_Id  Student_Name   Student_Id
---------  ------------   ----------
C101        Aditya         S901
C104        Aditya         S901
C106        Steve          S911
C109        Paul           S921
```

```
C115        Lucy            S931
```

Table 2: STUDENT

```
Student_Id     Student_Name   Student_Age
------------   ----------     -----------
S901           Aditya         19
S911           Steve          18
S921           Paul           19
S931           Lucy           17
S941           Carl           16
S951           Rick           18
```

Query:

```
∏ Student_Name (COURSE) ∪ ∏ Student_Name (STUDENT)
```

Output:

```
Student_Name
------------
Aditya
Carl
Paul
Lucy
Rick
Steve
```

Note: As you can see there are no duplicate names present in the output even though we had few common names in both the tables, also in the COURSE table we had the duplicate name itself.

4. Intersection Operator (∩)

Intersection operator is denoted by ∩ symbol and it is used to select common rows (tuples) from two tables (relations).

Lets say we have two relations R1 and R2 both have same columns and we want to select all those tuples(rows) that are present in both the relations, then in that case we can apply intersection operation on these two relations R1 ∩ R2.

Note: Only those rows that are present in both the tables will appear in the result set.

Syntax of Intersection Operator (∩)

```
table_name1 ∩ table_name2
```

Intersection Operator (∩) Example

Lets take the same example that we have taken above.
Table 1: COURSE

```
Course_Id  Student_Name   Student_Id
---------  ------------   ----------
C101        Aditya         S901
C104        Aditya         S901
C106        Steve          S911
C109        Paul           S921
C115        Lucy           S931
```

Table 2: STUDENT

```
Student_Id     Student_Name   Student_Age
------------   ----------     -----------
S901           Aditya         19
S911           Steve          18
S921           Paul           19
S931           Lucy           17
S941           Carl           16
S951           Rick           18
```

Query:

```
∏ Student_Name (COURSE) ∩ ∏ Student_Name (STUDENT)
```

Output:

```
Student_Name
------------
Aditya
Steve
Paul
Lucy
```

5. Set Difference (-)

Set Difference is denoted by – symbol. Lets say we have two relations R1 and R2 and we want to select all those tuples(rows) that are present in Relation R1 but not present in Relation R2, this can be done using Set difference R1 – R2.

Syntax of Set Difference (-)

```
table_name1 - table_name2
```

Set Difference (-) Example

Lets take the same tables COURSE and STUDENT that we have seen above.

Query:
Lets write a query to select those student names that are present in STUDENT table but not present in COURSE table.

```
∏ Student_Name (STUDENT) - ∏ Student_Name (COURSE)
```

Output:

```
Student_Name
------------
Carl
Rick
```

6. Cartesian product (X)

Cartesian Product is denoted by X symbol. Lets say we have two relations R1 and R2 then the cartesian product of these two relations (R1 X R2) would combine each tuple of first relation R1 with the each tuple of second relation R2. I know it sounds confusing but once we take an example of this, you will be able to understand this.

Syntax of Cartesian product (X)

```
R1 X R2
```

Cartesian product (X) Example

Table 1: R

```
Col_A    Col_B
-----    ------
AA        100
BB        200
CC        300
```

Table 2: S

Col_X	Col_Y
XX	99
YY	11
ZZ	101

Query:
Lets find the Cartesian product of table R and S.

R X S
Output:

Col_A	Col_B	Col_X	Col_Y
AA	100	XX	99
AA	100	YY	11
AA	100	ZZ	101
BB	200	XX	99
BB	200	YY	11
BB	200	ZZ	101
CC	300	XX	99
CC	300	YY	11
CC	300	ZZ	101

Note: The number of rows in the output will always be the cross product of number of rows in each table. In our example table 1 has 3 rows and table 2 has 3 rows so the output has 3×3 = 9 rows.

7. Rename (ρ)

Rename (ρ) operation can be used to rename a relation or an attribute of a relation.
Rename (ρ) Syntax:
ρ(new_relation_name, old_relation_name)

Rename (ρ) Example

Lets say we have a table customer, we are fetching customer names and we are renaming the resulted relation to CUST_NAMES.

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City

```
C10100          Steve           Agra
C10111          Raghu           Agra
C10115          Chaitanya       Noida
C10117          Ajeet           Delhi
C10118          Carl            Delhi
```

Query:

```
ρ(CUST_NAMES, ∏(Customer_Name)(CUSTOMER))
```

Output:

```
CUST_NAMES

----------
Steve
Raghu
Chaitanya
Ajeet
Carl
```

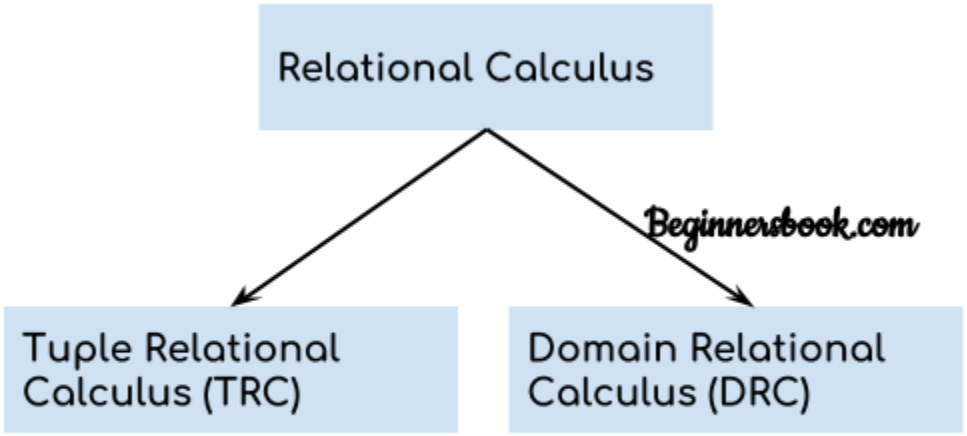
DBMS Relational Calculus

In the previous tutorial, we discussed Relational Algebra which is a procedural query language. In this tutorial, we will discuss Relational Calculus, which is a non-procedural query language.

What is Relational Calculus?

Relational calculus is a non-procedural query language that tells the system what data to be retrieved but doesn't tell how to retrieve it.

Types of Relational Calculus



1. Tuple Relational Calculus (TRC)

Tuple relational calculus is used for selecting those tuples that satisfy the given condition.
Table: Student

```
First_Name     Last_Name     Age
----------     ---------     ----
Ajeet          Singh         30
Chaitanya      Singh         31
Rajeev         Bhatia        27
Carl           Pratap        28
```

Lets write relational calculus queries.

Query to display the last name of those students where age is greater than 30

```
{ t.Last_Name | Student(t) AND t.age > 30 }
```

In the above query you can see two parts separated by | symbol. The second part is where we define the condition and in the first part we specify the fields which we want to display for the selected tuples.

The result of the above query would be:

```
Last_Name
---------
Singh
```

Query to display all the details of students where Last name is 'Singh'

```
{ t | Student(t) AND t.Last_Name = 'Singh' }
```

Output:

```
First_Name     Last_Name     Age
----------     ---------     ----
Ajeet          Singh         30
Chaitanya      Singh         31
```

2. Domain Relational Calculus (DRC)

In domain relational calculus the records are filtered based on the domains.
Again we take the same table to understand how DRC works.

Table: Student

```
First_Name     Last_Name     Age
----------     ---------     ----
Ajeet          Singh         30
Chaitanya      Singh         31
Rajeev         Bhatia        27
Carl           Pratap        28
```

Query to find the first name and age of students where student age is greater than 27

```
{< First_Name, Age > | ∈ Student ∧ Age > 27}
```

Note:
The symbols used for logical operators are: ∧ for AND, ∨ for OR and ¬ for NOT.

Output:

```
First_Name     Age
----------     ----
Ajeet          30
Chaitanya      31
Carl           28
```

KEYS in DBMS

Key plays an important role in relational database; it is used for identifying unique rows from table. It also establishes relationship among tables.

1. Primary key in DBMS

Definition**: A primary** key is a minimal set of attributes (columns) in a table that uniquely identifies tuples (rows) in that table.

Primary Key Example in DBMS

Lets take an example to understand the concept of primary key. In the following table, there are three attributes: Stu_ID, Stu_Name & Stu_Age. Out of these three attributes, one attribute or a set of more than one attributes can be a primary key.

Attribute Stu_Name alone cannot be a primary key as more than one students can have same name.

Attribute Stu_Age alone cannot be a primary key as more than one students can have same age.

Attribute Stu_Id alone is a primary key as each student has a unique id that can identify the student record in the table.

Note: In some cases an attribute alone cannot uniquely identify a record in a table, in that case we try to find a set of attributes that can uniquely identify a row in table. We will see the example of it after this example.

Table Name: STUDENT

Stu_Id	Stu_Name	Stu_Age
101	Steve	23
102	John	24
103	Robert	28
104	Steve	29
105	Carl	29

Points to Note regarding Primary Key

- We usually denote it by underlining the attribute name (column name).
- The value of primary key should be unique for each row of the table. The column(s) that makes the key cannot contain duplicate values.
- The attribute(s) that is marked as primary key is not allowed to have null values.
- Primary keys are not necessarily to be a single attribute (column). It can be a set of more than one attributes (columns). For example {Stu_Id, Stu_Name} collectively can identify the tuple in the above table, but we do not choose it as primary key because Stu_Id alone is enough to uniquely identifies rows in a table and we always go for minimal set. Having that said, we should choose more than one columns as primary key only when there is no single column that can uniquely identify the tuple in table.

Another example of primary key – More than one attributes

Consider this table ORDER, this table keeps the daily record of the purchases made by the customer. This table has three attributes: Customer_ID, Product_ID & Order_Quantity.

Customer_ID alone cannot be a primary key as a single customer can place more than one order thus more than one rows of same Customer_ID value. As we see in the following example that customer id 1011 has placed two orders with product if 9023 and 9111.

Product_ID alone cannot be a primary key as more than one customers can place a order for the same product thus more than one rows with same product id. In the following table, customer id 1011 & 1122 placed an order for the same product (product id 9023).

Order_Quantity alone cannot be a primary key as more more than one customers can place the order for the same quantity.

Since none of the attributes alone were able to become a primary key, lets try to make a set of attributes that plays the role of it.

{Customer_ID, Product_ID} together can identify the rows uniquely in the table so this set is the primary key for this table.

Table Name: ORDER

Customer_ID	Product_ID	Order_Quantity
1011	9023	10
1122	9023	15
1099	9031	20
1177	9031	18
1011	9111	50

Note: While choosing a set of attributes for a primary key, we always choose the minimal set that has minimum number of attributes. For example, if there are two sets that can identify row in table, the set that has minimum number of attributes should be chosen as primary key.

How to define primary key in RDBMS?

In the above example, we already had a table with data and we were trying to understand the purpose and meaning of primary key, however you should know that generally we define the primary key during table creation. We can define the primary key later as well but that rarely happens in the real world scenario.

Lets say we want to create the table that we have discussed above with the customer id and product id set working as primary key. We can do that in SQL like this:

```
Create table ORDER
(
    Customer_ID int not null,
    Product_ID int not null,
    Order_Quantity int not null,
    Primary key (Customer_ID, Product_ID)
```

```
)
```
Suppose we didn't define the primary key while creating table then we can define it later like this:

```
ALTER TABLE ORDER
ADD CONSTRAINT PK_Order PRIMARY KEY (Customer_ID, Product_ID);
```

Another way:
When we have only one attribute as primary key, like we see in the first example of STUDENT table. we can define the key like this as well:

```
Create table STUDENT
(
    Stu_Id int primary key,
    Stu_Name varchar(255) not null,
    Stu_Age int not null
)
```

2. Super key in DBMS

Definition: A super key is a set of one or more attributes (columns), which can uniquely identify a row in a table. Often DBMS beginners get confused between super key and candidate key, so we will also discuss candidate key and its relation with super key in this article.

How candidate key is different from super key?

Answer is simple – Candidate keys are selected from the set of super keys, the only thing we take care while selecting candidate key is: It should not have any redundant attribute. That's the reason they are also termed as minimal super key.

Let's take an example to understand this:
Table: Employee

```
Emp_SSN      Emp_Number    Emp_Name
---------    ----------    --------
123456789         226       Steve
999999321         227       Ajeet
888997212         228       Chaitanya
777778888         229       Robert
```
Super keys: The above table has following super keys. All of the following sets of super key are able to uniquely identify a row of the employee table.

- {Emp_SSN}
- {Emp_Number}
- {Emp_SSN, Emp_Number}

- {Emp_SSN, Emp_Name}
- {Emp_SSN, Emp_Number, Emp_Name}
- {Emp_Number, Emp_Name}

Candidate Keys: As I mentioned in the beginning, a candidate key is a minimal super key with no redundant attributes. The following two set of super keys are chosen from the above sets as there are no redundant attributes in these sets.

- {Emp_SSN}
- {Emp_Number}

Only these two sets are candidate keys as all other sets are having redundant attributes that are not necessary for unique identification.

Super key vs Candidate Key

I have been getting lot of comments regarding the confusion between super key and candidate key. Let me give you a clear explanation.
1. First you have to understand that all the candidate keys are super keys. This is because the candidate keys are chosen out of the super keys.
2. How we choose candidate keys from the set of super keys? We look for those keys from which we cannot remove any fields. In the above example, we have not chosen {Emp_SSN, Emp_Name} as candidate key because {Emp_SSN} alone can identify a unique row in the table and Emp_Name is redundant.

Primary key:
A Primary key is selected from a set of candidate keys. This is done by database admin or database designer. We can say that either {Emp_SSN} or {Emp_Number} can be chosen as a primary key for the table Employee.

3. Candidate Key in DBMS

Definition: A super key with no redundant attribute is known as candidate key. Candidate keys are selected from the set of super keys, the only thing we take care while selecting candidate key is that the candidate key should not have any redundant attributes. That's the reason they are also termed as minimal super key.

Candidate Key Example

Lets take an example of table "Employee". This table has three attributes: Emp_Id, Emp_Number & Emp_Name. Here Emp_Id & Emp_Number will be having unique values and Emp_Name can have duplicate values as more than one employees can have same name.

```
Emp_Id    Emp_Number        Emp_Name
------  ----------     --------
```

E01	2264	Steve
E22	2278	Ajeet
E23	2288	Chaitanya
E45	2290	Robert

How many super keys the above table can have?
1. {Emp_Id}
2. {Emp_Number}
3. {Emp_Id, Emp_Number}
4. {Emp_Id, Emp_Name}
5. {Emp_Id, Emp_Number, Emp_Name}
6. {Emp_Number, Emp_Name}

Lets select the candidate keys from the above set of super keys.

1. {Emp_Id} – No redundant attributes
2. {Emp_Number} – No redundant attributes
3. {Emp_Id, Emp_Number} – Redundant attribute. Either of those attributes can be a minimal super key as both of these columns have unique values.
4. {Emp_Id, Emp_Name} – Redundant attribute Emp_Name.
5. {Emp_Id, Emp_Number, Emp_Name} – Redundant attributes. Emp_Id or Emp_Number alone are sufficient enough to uniquely identify a row of Employee table.
6. {Emp_Number, Emp_Name} – Redundant attribute Emp_Name.

The **candidate keys** we have selected are:
{Emp_Id}
{Emp_Number}

Note: A primary key is selected from the set of candidate keys. That means we can either have Emp_Id or Emp_Number as primary key. The decision is made by DBA (Database administrator)

4. Foreign key in DBMS

Definition: Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

For example:
In the below example the Stu_Id column in Course_enrollment table is a foreign key as it points to the primary key of the Student table.

Course_enrollment table:

Course_Id	Stu_Id
C01	101
C02	102
C03	101
C05	102
C06	103
C07	102

Student table:

Stu_Id	Stu_Name	Stu_Age
101	Chaitanya	22
102	Arya	26
103	Bran	25
104	Jon	21

Note: Practically, the foreign key has nothing to do with the primary key tag of another table, if it points to a unique column (not necessarily a primary key) of another table then too, it would be a foreign key. So, a correct definition of foreign key would be: Foreign keys are the columns of a table that points to the candidate key of another table.

5. Composite key in DBMS

Definition: A key that has more than one attributes is known as composite key. It is also known as compound key.

Note: Any key such as super key, primary key, candidate key etc. can be called composite key if it has more than one attributes.

Composite key Example

Lets consider a table Sales. This table has four columns (attributes) – cust_Id, order_Id, product_code & product_count.

Table – Sales

```
cust_Id    order_Id    product_code    product_count
--------   --------    ------------    -------------
C01         O001         P007           23
C02         O123          P007           19
C02         O123          P230           82
C01         O001          P890           42
```

None of these columns alone can play a role of key in this table.

Column **cust_Id** alone cannot become a key as a same customer can place multiple orders, thus the same customer can have multiple entries.

Column **order_Id** alone cannot be a primary key as a same order can contain the order of multiple products, thus same order_Id can be present multiple times**.**

Column **product_code** cannot be a primary key as more than one customers can place order for the same product.

Column **product_count** alone cannot be a primary key because two orders can be placed for the same product count.

Based on this, it is safe to assume that the key should be having more than one attributes:
Key in above table: {cust_id, product_code}

This is a composite key as it is made up of more than one attributes.

6. Alternate key in DBMS

As we have seen in the candidate key guide that a table can have multiple candidate keys. Among these candidate keys, only one key gets selected as primary key, the remaining keys are known as **alternative or secondary keys.**

Alternate Key Example

Lets take an example to understand the alternate key concept. Here we have a table Employee, this table has three attributes: Emp_Id, Emp_Number & Emp_Name.

Table: Employee/strong>

```
Emp_Id   Emp_Number    Emp_Name
------   ----------    --------
E01       2264          Steve
E22       2278          Ajeet
```

```
E23      2288          Chaitanya
E45      2290          Robert
```

There are two candidate keys in the above table:
{Emp_Id}
{Emp_Number}

DBA (Database administrator) can choose any of the above key as primary key. Lets say Emp_Id is chosen as primary key.

Since we have selected Emp_Id as primary key, the remaining key Emp_Number would be called alternative or secondary key.

Normalization in DBMS: 1NF, 2NF, 3NF and BCNF in Database

Normalization is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly. Let's discuss about anomalies first then we will discuss normal forms with examples.

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	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	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 Rick as he belongs to two departments of the company. If we want to update the address of Rick 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, Rick 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 Maggie since she is assigned only to this department.

To overcome these anomalies we need to normalize the data. In the next section we will discuss about normalization.

Normalization

Here are the most commonly used normal forms:

1. First normal form(1NF)
2. Second normal form(2NF)
3. Third normal form(3NF)
4. Boyce & Codd normal form (BCNF)

1. First normal form (1NF)

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Example: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123

Two employees (Jon & Lester) are having two mobile numbers so the company stored them in the same field as you can see in the table above.

This table is not in 1NF as the rule says "each attribute of a table must have atomic (single) values", the emp_mobile values for employees Jon & Lester violates that rule.

To make the table complies with 1NF we should have the data like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
102	Jon	Kanpur	9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123
104	Lester	Bangalore	8123450987

2. Second normal form (2NF)

A table is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)
- No non-prime attribute is dependent on the proper subset of any candidate key of table.

An attribute that is not part of any candidate key is known as non-prime attribute.

Example: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

teacher_id	subject	teacher_age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Candidate Keys: {teacher_id, subject}
Non prime attribute: teacher_age

The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says “no non-prime attribute is dependent on the proper subset of any candidate key of the table”.

To make the table complies with 2NF we can break it in two tables like this:

teacher_details table:

teacher_id	teacher_age
111	38
222	38
333	40

teacher_subject table:

teacher_id	subject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

Now the tables comply with Second normal form (2NF).

3. Third Normal form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- Transitive functional dependency of non-prime attribute on any super key should be removed.

An attribute that is not part of any candidate key is known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

- X is a super key of table
- Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

Example: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on
Candidate Keys: {emp_id}
Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.

Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

employee table:

emp_id	emp_name	emp_zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

employee_zip table:

emp_zip	emp_state	emp_city	emp_district

282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

4. Boyce Codd normal form (BCNF)

It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every functional dependency X->Y, X should be the super key of the table.

Example: Suppose there is a company wherein employees work in more than one department. They store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

Functional dependencies in the table above:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate key: {emp_id, emp_dept}

The table is not in BCNF as neither emp_id nor emp_dept alone are keys.

To make the table comply with BCNF we can break the table in three tables like this:

emp_nationality table:

emp_id	emp_nationality
1001	Austrian
1002	American

emp_dept table:

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_dept_mapping table:

emp_id	emp_dept
1001	Production and planning
1001	stores
1002	design and technical support
1002	Purchasing department

Functional dependencies:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate keys:
For first table: emp_id
For second table: emp_dept
For third table: {emp_id, emp_dept}

This is now in BCNF as in both the functional dependencies left side part is a key.

Functional dependency in DBMS

The attributes of a table is said to be dependent on each other when an attribute of a table uniquely identifies another attribute of the same table.

For example: Suppose we have a student table with attributes: Stu_Id, Stu_Name, Stu_Age. Here Stu_Id attribute uniquely identifies the Stu_Name attribute of student table because if we know the student id we can tell the student name associated with it. This is known as functional dependency and can be written as Stu_Id->Stu_Name or in words we can say Stu_Name is functionally dependent on Stu_Id.

Formally:
If column A of a table uniquely identifies the column B of same table then it can represented as A->B (Attribute B is functionally dependent on attribute A)

Types of Functional Dependencies

1. Trivial functional dependency
2. non-trivial functional dependency
3. Multivalued dependency
4. Transitive dependency

1. Trivial functional dependency in DBMS with example

The dependency of an attribute on a set of attributes is known as trivial functional dependency if the set of attributes includes that attribute.

Symbolically: A ->B is trivial functional dependency if B is a subset of A.

The following dependencies are also trivial: A->A & B->B

For example: Consider a table with two columns Student_id and Student_Name.

{Student_Id, Student_Name} -> Student_Id is a trivial functional dependency as Student_Id is a subset of {Student_Id, Student_Name}. That makes sense because if we know the values of Student_Id and Student_Name then the value of Student_Id can be uniquely determined.

Also, Student_Id -> Student_Id & Student_Name -> Student_Name are trivial dependencies too.

2. Non trivial functional dependency in DBMS

If a functional dependency X->Y holds true where Y is not a subset of X then this dependency is called non trivial Functional dependency.

For example:
An employee table with three attributes: emp_id, emp_name, emp_address.
The following functional dependencies are non-trivial:
emp_id -> emp_name (emp_name is not a subset of emp_id)
emp_id -> emp_address (emp_address is not a subset of emp_id)

On the other hand, the following dependencies are trivial:
{emp_id, emp_name} -> emp_name [emp_name is a subset of {emp_id, emp_name}]
Refer: trivial functional dependency.

Completely non trivial FD:
If a FD X->Y holds true where X intersection Y is null then this dependency is said to be completely non trivial function dependency.

3. Multivalued dependency in DBMS

Multivalued dependency occurs when there are more than one independent multivalued attributes in a table.

For example: Consider a bike manufacture company, which produces two colors (Black and white) in each model every year.

bike_model	manuf_year	color
M1001	2007	Black
M1001	2007	Red
M2012	2008	Black
M2012	2008	Red
M2222	2009	Black
M2222	2009	Red

Here columns manuf_year and color are independent of each other and dependent on bike_model. In this case these two columns are said to be multivalued dependent on bike_model. These dependencies can be represented like this:

bike_model ->> manuf_year

bike_model ->> color

4. Transitive dependency in DBMS

A functional dependency is said to be transitive if it is indirectly formed by two functional dependencies. For e.g.

X -> Z is a transitive dependency if the following three functional dependencies hold true:

- X->Y

- Y does not ->X
- Y->Z

Note: A transitive dependency can only occur in a relation of three of more attributes. This dependency helps us normalizing the database in 3NF (3rd Normal Form).

Example: Let's take an example to understand it better:

Book	**Author**	**Author_age**
Game of Thrones	George R. R. Martin	66
Harry Potter	J. K. Rowling	49
Dying of the Light	George R. R. Martin	66

{Book} ->{Author} (if we know the book, we knows the author name)

{Author} does not ->{Book}

{Author} -> {Author_age}

Therefore as per the rule of **transitive dependency:** {Book} -> {Author_age} should hold, that makes sense because if we know the book name we can know the author's age.

Transaction Management in DBMS

A **transaction** is a set of logically related operations. For example, you are transferring money from your bank account to your friend's account, the set of operations would be like this:

Simple Transaction Example

1. Read your account balance
2. Deduct the amount from your balance
3. Write the remaining balance to your account
4. Read your 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. Although I have shown you read, write and update operations in the above example but the transaction can have operations like read, write, insert, update, delete.

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);
```

In the above transaction **R** refers to the **Read operation** and **W** refers to the **write operation.**

Transaction failure in between the operations

Now that we understand what is transaction, we should understand what are the problems associated with it.

The main problem that can happen during a transaction is that the transaction can fail before finishing the 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 your account but your 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 in DBMS

To ensure the integrity of data during a transaction, the database system maintains the following properties. These properties are widely known as ACID properties:

1. **Atomicity:** This property ensures that either all the operations of a transaction reflect in database or none. Let's take an example of banking system to understand this: Suppose Account A has a balance of 400$ & B has 700$. Account A is transferring 100$ to Account B. This is a transaction that has two operations

a) Debiting 100$ from A's balance

b) Creating 100$ to B's balance.

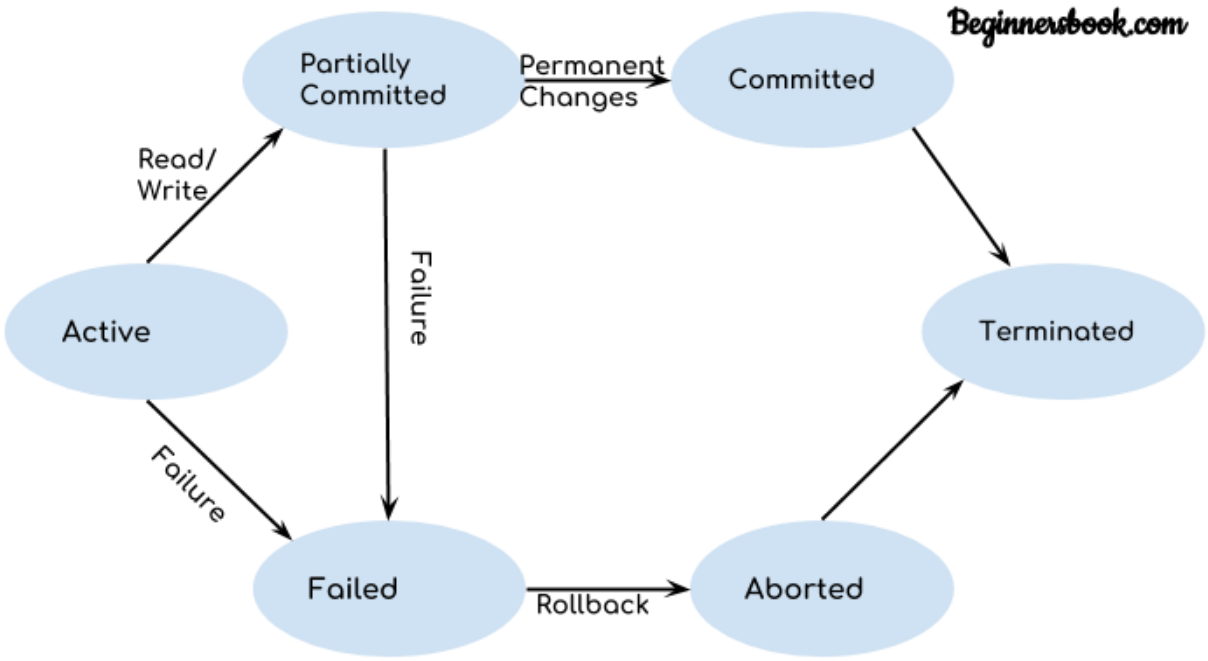
Let's say first operation passed successfully while second failed, in this case A's balance would be 300$ while B would be having 700$ instead of 800$. This is unacceptable in a banking system. Either the transaction should fail without executing any of the operation or it should process both the operations. The Atomicity property ensures that.

2. **Consistency:** To preserve the consistency of database, the execution of transaction should take place in isolation (that means no other transaction should run concurrently when there is a transaction already running). For example account A is having a balance of 400$ and it is transferring 100$ to account B & C both. So we have two transactions here. Let's say these transactions run concurrently and both the transactions read 400$ balance, in that case the final balance of A would be 300$ instead of 200$. This is wrong. If the transaction were to run in isolation then the second transaction would have read the correct balance 300$ (before debiting 100$) once the first transaction went successful.
3. **Isolation:** For every pair of transactions, one transaction should start execution only when the other finished execution. I have already discussed the example of Isolation in the Consistency property above.
4. **Durability:** Once a transaction completes successfully, the changes it has made into the database should be permanent even if there is a system failure. The recovery-management component of database systems ensures the durability of transaction.

DBMS Transaction States

A transaction in DBMS can be in one of the following states**.**

DBMS Transaction States Diagram



Lets discuss these states one by one.

Active State

As we have discussed in the DBMS transaction introduction that a transaction is a sequence of operations. If a transaction is in execution then it is said to be in active state. It doesn't matter which step is in execution, until unless the transaction is executing, it remains in active state.

Failed State

If a transaction is executing and a failure occurs, either a hardware failure or a software failure then the transaction goes into failed state from the active state.

Partially Committed State

As we can see in the above diagram that a transaction goes into "partially committed" state from the active state when there are read and write operations present in the transaction.

A transaction contains number of read and write operations. Once the whole transaction is successfully executed, the transaction goes into partially committed state where we have all the read and write operations performed on the main memory (local memory) instead of the actual database.

The reason why we have this state is because a transaction can fail during execution so if we are making the changes in the actual database instead of local memory, database may be left in an inconsistent state in case of any failure**.** This state helps us to rollback the changes made to the database in case of a failure during execution.

Committed State

If a transaction completes the execution successfully then all the changes made in the local memory during **partially committed** state are permanently stored in the database. You can also see in the above diagram that a transaction goes from partially committed state to committed state when everything is successful.

Aborted State

As we have seen above, if a transaction fails during execution then the transaction goes into a failed state. The changes made into the local memory (or buffer) are rolled back to the previous consistent state and the transaction goes into aborted state from the failed state. Refer the diagram to see the interaction between failed and aborted state.

DBMS Schedules and the Types of Schedules

When multiple transactions are running concurrently then there needs to be a sequence in which the operations are performed because at a time only one operation can be performed on the database. This sequence of operations is known as **Schedule.**

Lets take an example to understand what is a schedule in DBMS.

DBMS Schedule example

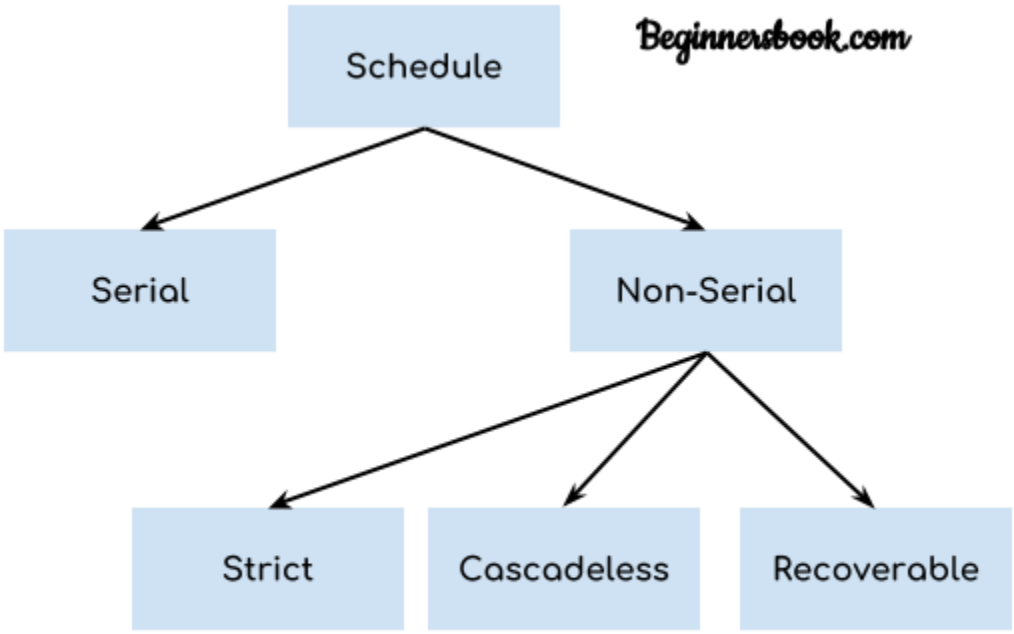
The following sequence of operations is a schedule. Here we have two transactions T1 & T2 which are running concurrently.

This schedule determines the exact order of operations that are going to be performed on database. In this example, all the instructions of transaction T1 are executed before the instructions of transaction T2, however this is not always necessary and we can have various types of schedules which we will discuss in this article.

```
T1          T2
----   ----
R(X)
W(X)
R(Y)
     R(Y)
     R(X)
     W(Y)
```

Types of Schedules in DBMS

We have various types of schedules in DBMS. Lets discuss them one by one.



Serial Schedule

In **Serial schedule**, a transaction is executed completely before starting the execution of another transaction. In other words, you can say that in serial schedule, a transaction does not start execution until the currently running transaction finished execution. This type of execution of transaction is also known as non-interleaved execution. The example we have seen above is the serial schedule.

Lets take another example.

Serial Schedule example
Here R refers to the read operation and W refers to the write operation. In this example, the transaction T2 does not start execution until the transaction T1 is finished.

```
T1          T2
----    ----
R(A)
R(B)
W(A)
commit
```

```
      R(B)
      R(A)
      W(B)
      commit
```

Strict Schedule

In Strict schedule, if the write operation of a transaction precedes a conflicting operation (Read or Write operation) of another transaction then the commit or abort operation of such transaction should also precede the conflicting operation of other transaction.

Strict Schedule example
Lets say we have two transactions Ta and Tb. The write operation of transaction Ta precedes the read or write operation of transaction Tb, so the commit or abort operation of transaction Ta should also precede the read or write of Tb.

```
Ta            Tb
-----   -----
R(X)
      R(X)
W(X)
commit
      W(X)
      R(X)
      commit
```

Here the write operation W(X) of Ta precedes the conflicting operation (Read or Write operation) of Tb so the conflicting operation of Tb had to wait the commit operation of Ta.

Cascadeless Schedule

In Cascadeless Schedule, if a transaction is going to perform read operation on a value, it has to wait until the transaction who is performing write on that value commits.

Cascadeless Schedule example
For example, lets say we have two transactions Ta and Tb. Tb is going to read the value X after the W(X) of Ta then Tb has to wait for the commit operation of transaction Ta before it reads the X.

```
Ta             Tb
-----    -----
R(X)
W(X)
       W(X)
commit
       R(X)
       W(X)
       commit
```

Recoverable Schedule

In Recoverable schedule, if a transaction is reading a value which has been updated by some other transaction then this transaction can commit only after the commit of other transaction which is updating value.

Recoverable Schedule example
Here Tb is performing read operation on X after the Ta has made changes in X using W(X) so Tb can only commit after the commit operation of Ta.

```
Ta            Tb
-----    -----
R(X)
W(X)
       R(X)
       W(X)
       R(X)
commit
       commit
```

DBMS Serializability

When multiple transactions are running concurrently then there is a possibility that the database may be left in an inconsistent state. 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.

What is a serializable schedule?

A serializable schedule always leaves the database in consistent state. A serial schedule is always a serializable schedule because in serial schedule, a transaction only starts when the other transaction finished execution. However a non-serial schedule needs to be checked for Serializability.

A non-serial schedule of n number of transactions is said to be serializable schedule, if it is equivalent to the serial schedule of those n transactions. A serial schedule doesn't allow concurrency, only one transaction executes at a time and the other starts when the already running transaction finished.

Types of Serializability

There are two types of Serializability.

1. Conflict Serializability
2. View Serializability

DBMS Conflict Serializability

In the DBMS Schedules guide, we learned that there are two types of schedules – Serial & Non-Serial. A Serial schedule doesn't support concurrent execution of transactions while a non-serial schedule supports concurrency. We also learned in Serializability tutorial that a non-serial schedule may leave the database in inconsistent state so we need to check these non-serial schedules for the Serializability.

Conflict Serializability is one of the type of Serializability, which can be used to check whether a non-serial schedule is conflict serializable or not.

What is Conflict Serializability?

A schedule is called conflict serializable if we can convert it into a serial schedule after swapping its non-conflicting operations.

Conflicting operations

Two operations are said to be in conflict, if they satisfy all the following three conditions:

1. Both the operations should belong to different transactions.
2. Both the operations are working on same data item.
3. At least one of the operation is a write operation.

Lets see some examples to understand this:
Example 1: Operation W(X) of transaction T1 and operation R(X) of transaction T2 are conflicting operations, because they satisfy all the three conditions mentioned above. They belong to different transactions, they are working on same data item X, one of the operation in write operation.

Example 2: Similarly Operations W(X) of T1 and W(X) of T2 are conflicting operations.

Example 3: **O**perations W(X) of T1 and W(Y) of T2 are non-conflicting operations because both the write operations are not working on same data item so these operations don't satisfy the second condition.

Example 4: Similarly R(X) of T1 and R(X) of T2 are non-conflicting operations because none of them is write operation.

Example 5: Similarly W(X) of T1 and R(X) of T1 are non-conflicting operations because both the operations belong to same transaction T1.

Conflict Equivalent Schedules

Two schedules are said to be conflict Equivalent if one schedule can be converted into other schedule after swapping non-conflicting operations.

Conflict Serializable check

Lets check whether a schedule is conflict serializable or not. If a schedule is conflict Equivalent to its serial schedule then it is called Conflict Serializable schedule. Lets take few examples of schedules.

Example of Conflict Serializability

Lets consider this schedule:

```
T1       T2
-----    ------
R(A)
R(B)
       R(A)
       R(B)
       W(B)
W(A)
```

To convert this schedule into a serial schedule we must have to swap the R(A) operation of transaction T2 with the W(A) operation of transaction T1. However we cannot swap these two operations because they are conflicting operations, thus we can say that this given schedule is **not Conflict Serializable.**

Lets take another example:

```
T1       T2
-----    ------
R(A)
       R(A)
       R(B)
       W(B)
R(B)
W(A)
```

Lets **swap non-conflicting operations:**

After swapping R(A) of T1 and R(A) of T2 we get:

```
T1        T2
-----     ------
        R(A)
R(A)
        R(B)
        W(B)
R(B)
W(A)
```
After swapping R(A) of T1 and R(B) of T2 we get:

```
T1        T2
-----     ------
        R(A)
        R(B)
R(A)
        W(B)
R(B)
W(A)
```
After swapping R(A) of T1 and W(B) of T2 we get:

```
T1        T2
-----     ------
        R(A)
        R(B)
        W(B)
R(A)
R(B)
W(A)
```
We finally got a serial schedule after swapping all the non-conflicting operations so we can say that the given schedule is **Conflict Serializable.**

DBMS View Serializability

In the last tutorial, we learned Conflict Serializability. In this article, we will discuss another type of serializability which is known as **View Serializability.**

What is 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. Lets take an example to understand what I mean by that.

Given Schedule:

```
T1      T2
-----   ------
R(X)
W(X)
        R(X)
        W(X)
R(Y)
W(Y)
        R(Y)
        W(Y)
```

Serial Schedule of the above given schedule:
As we know that in Serial schedule a transaction only starts when the current running transaction is finished. So the serial schedule of the above given schedule would look like this:

```
T1      T2
-----   ------
R(X)
W(X)
R(Y)
W(Y)
        R(X)
        W(X)
        R(Y)
        W(Y)
```

If we can prove that the given schedule is View Equivalent to its serial schedule then the given schedule is called **view Serializable.**

Why we need View Serializability?

We know that a serial schedule never leaves the database in inconsistent state because there are no concurrent transactions execution. However a non-serial schedule can leave the database in inconsistent state because there are multiple transactions running concurrently. By checking that a given non-serial schedule is view serializable, we make sure that it is a consistent schedule.

You may be wondering instead of checking that a non-serial schedule is serializable or not, can't we have serial schedule all the time? The answer is no, because concurrent execution of transactions fully utilize the system resources and are considerably faster compared to serial schedules.

View Equivalent

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

Two schedules T1 and T2 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: You may be confused by the term 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.

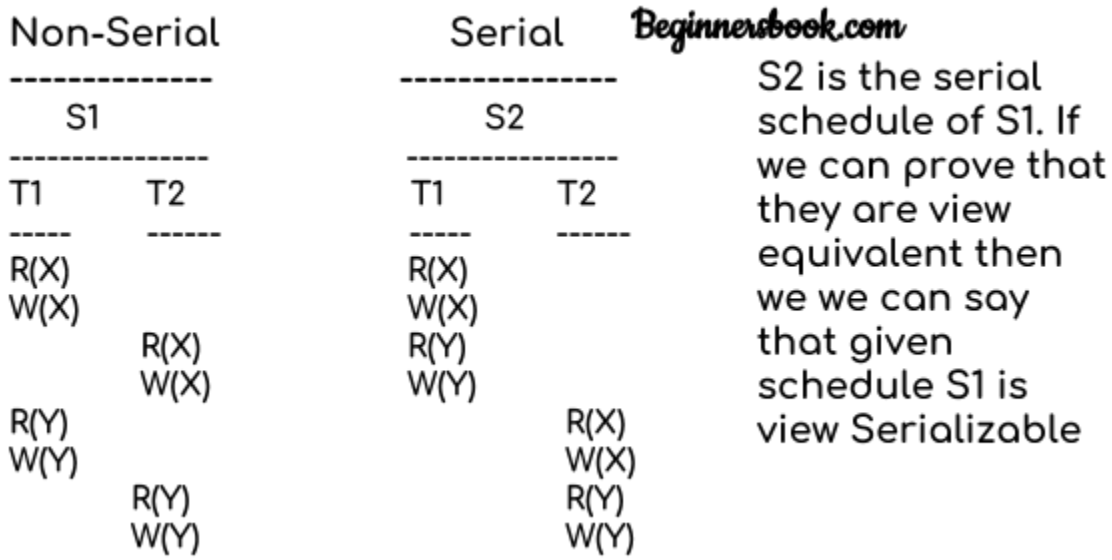
2. Final Write: Final write operations on each data item must match in both the schedules. For example, a data item X is last written by Transaction T1 in schedule S1 then in S2, the last write operation on X should be performed by the transaction T1.

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.

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



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.

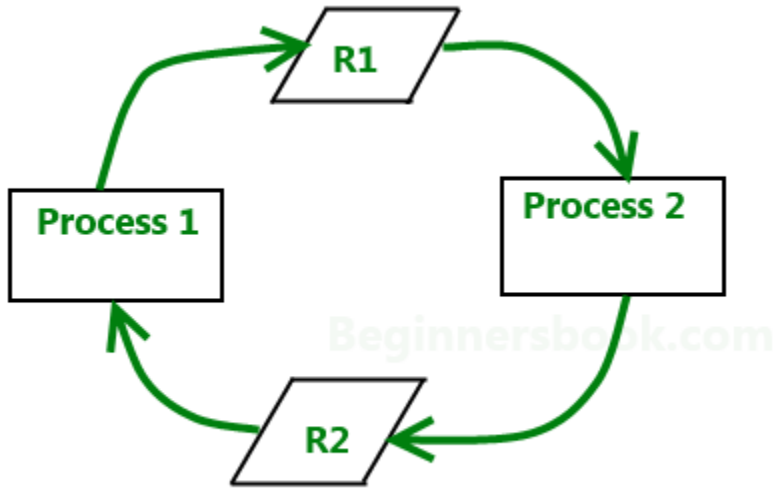
The update read condition is also satisfied for both the schedules.

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.

Deadlock in DBMS

A **deadlock** is a condition wherein two or more tasks are waiting for each other in order to be finished but none of the task is willing to give up the resources that other task needs. In this

situation no task ever gets finished and is in waiting state forever.



Coffman conditions

Coffman stated four conditions for a deadlock occurrence. A deadlock may occur if all the following conditions holds true.

- **Mutual exclusion condition:** There must be at least one resource that cannot be used by more than one process at a time.
- **Hold and wait condition:** A process that is holding a resource can request for additional resources that are being held by other processes in the system.
- **No preemption condition:-**A resource cannot be forcibly taken from a process. Only the process can release a resource that is being held by it.
- **Circular wait condition:** A condition where one process is waiting for a resource that is being held by second process and second process is waiting for third process ….so on and the last process is waiting for the first process. Thus making a circular chain of waiting.

Deadlock Handling

Ignore the deadlock (Ostrich algorithm)

Did that made you laugh? You may be wondering how ignoring a deadlock can come under deadlock handling. But to let you know that the windows you are using on your PC, uses this approach of deadlock handling and that is reason sometimes it hangs up and you have to reboot it to get it working. Not only Windows but UNIX also uses this approach.

The question is why? Why instead of dealing with a deadlock they ignore it and why this is being called as Ostrich algorithm?

Well! Let me answer the second question first, This is known as Ostrich algorithm because in this approach we ignore the deadlock and pretends that it would never occur, just like Ostrich behavior “to stick one’s head in the sand and pretend there is no problem.”

Let’s discuss why we ignore it: When it is believed that deadlocks are very rare and cost of deadlock handling is higher, in that case ignoring is better solution than handling it. For example: Let’s take the operating system example – If the time requires handling the deadlock is higher than the time requires rebooting the windows then rebooting would be a preferred choice considering that deadlocks are very rare in windows.

Deadlock detection

Resource scheduler is one that keeps the track of resources allocated to and requested by processes. Thus, if there is a deadlock it is known to the resource scheduler. This is how a deadlock is detected.

Once a deadlock is detected it is being corrected by following methods:

- **Terminating processes involved in deadlock:** Terminating all the processes involved in deadlock or terminating process one by one until deadlock is resolved can be the solutions but both of these approaches are not good. Terminating all processes cost high and partial work done by processes gets lost. Terminating one by one takes lot of time because each time a process is terminated, it needs to check whether the deadlock is resolved or not. Thus, the best approach is considering process age and priority while terminating them during a deadlock condition.
- **Resource Preemption:** Another approach can be the preemption of resources and allocation of them to the other processes until the deadlock is resolved.

Deadlock prevention

We have learnt that if all the four Coffman conditions hold true then a deadlock occurs so preventing one or more of them could prevent the deadlock.

- **Removing mutual exclusion:** All resources must be sharable that means at a time more than one processes can get a hold of the resources. That approach is practically impossible.
- **Removing hold and wait condition:** This can be removed if the process acquires all the resources that are needed before starting out. Another way to remove this to enforce a rule of requesting resource when there are none in held by the process.
- **Preemption of resources:** Preemption of resources from a process can result in rollback and thus this needs to be avoided in order to maintain the consistency and stability of the system.
- **Avoid circular wait condition:** This can be avoided if the resources are maintained in a hierarchy and process can hold the resources in increasing order of precedence. This avoid circular wait. Another way of doing this to force one resource per process rule – A process

can request for a resource once it releases the resource currently being held by it. This avoids the circular wait.

Deadlock Avoidance

Deadlock can be avoided if resources are allocated in such a way that it avoids the deadlock occurrence. There are two algorithms for deadlock avoidance.

- Wait/Die
- Wound/Wait

Here is the table representation of resource allocation for each algorithm. Both of these algorithms take process age into consideration while determining the best possible way of resource allocation for deadlock avoidance.

	Wait/Die	**Wound/Wait**
Older process needs a resource held by younger	Older	Younger
Younger process needs a resource held by older	Younger	Younger

One of the famous deadlock avoidance algorithm is **Banker's algorithm**

Concurrency Control in DBMS

When more than one transactions are running simultaneously 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.

Let's take an example to understand what I'm talking here.

Conflict Example

You and your brother have a joint bank account, from which you both can withdraw money. Now let's say you both go to different branches of the same bank at the same time and try to withdraw 5000 INR, your joint account has only 6000 balance. Now if we don't have concurrency control in place you both can get 5000 INR at the same time but once both the transactions finish the account balance would be -4000 which is not possible and leaves the database in inconsistent state.
We need something that controls the transactions in such a way that allows the transaction to run concurrently but maintaining the consistency of data to avoid such issues.

Solution of Conflicts: Locks

A lock is kind of a mechanism that ensures that the integrity of data is maintained. There are two types of a lock that can be placed while accessing the data so that the concurrent transaction can not alter the data while we are processing it.

1. Shared Lock(S)
2. Exclusive Lock(X)

1. Shared Lock(S): Shared lock is placed when we are reading the data, multiple shared locks can be placed on the data but when a shared lock is placed no exclusive lock can be placed.

For example, when two transactions are reading Steve's account balance, let them read by placing shared lock but at the same time if another transaction wants to update the Steve's account balance by placing Exclusive lock, do not allow it until reading is finished.

2. Exclusive Lock(X): Exclusive lock is placed when we want to read and write the data. This lock allows both the read and write operation, Once this lock is placed on the data no other lock (shared or Exclusive) can be placed on the data until Exclusive lock is released.

For example, when a transaction wants to update the Steve's account balance, let it do by placing X lock on it but if a second transaction wants to read the data(S lock) don't allow it, if another transaction wants to write the data(X lock) don't allow that either.

So based on this we can create a table like this:

Lock Compatibility Matrix

```
_____________________________
|    |  S   |   X   |
|-------------------------
| S  | True   |   False |
|-------------------------
| X  | False  |   False |
-------------------------
```

How to read this matrix?:
There are two rows, first row says that when S lock is placed, another S lock can be acquired so it is marked true but no Exclusive locks can be acquired so marked False.
In second row, When X lock is acquired neither S nor X lock can be acquired so both marked false.