UNIT-1

Introduction to DBMS

Database is a collection of data. It contains information about one particular enterprise. A Database Management System (DBMS) consists of a collection of interrelated data and a set of programs to access that data. DBMS is a set of prewritten programs that are used to store, update and retrieve a database. A DBMS controls the organization, storage, management and retrieval of database. The primary goal of DBMS 1s to provide an environment that is both convenient and efficient to use in retrievin8 and storing a database information.

Database systems are designed to manage large bodies of information. The database system must provide the safety of the information stored, despite system crashes or attempts of unauthorized access. If the data is to be shared among several users, the system must maintain consistency.

The DBMS accepts requests for data from the application program and instructs DBMS is used, the operating system to transfer the appropriate data. When information system can be changed much more easily as the organization's information requirements change. New categories of data can be added to the database without disruption to the existing system.

Purpose of DBMS

A DBMS provides a secure and survivable medium for the storage and retrieval of data. In the real world, the data is shared among several users and is persistent. Also the real world data have a structure, related to one another and have constraints. These features are well represented and can be efficiently managed using a DBMS. Also the different users of the data need to create, access and manipulate the data. The DBMS provides mechanism to achieve these objectives without compromising security and integrity of data.

Therefore, it the data is shared, if it is persistent, it the users want it to be secure and easy to access and manipulate, then use or d database management system is the best available alternative.

Functions of DBMMS

There are many functions a DBMS serves that are key components to the operation of database management. When deciding to implement a DBMS In your business, first you must decide what type of DBMS you want. Common types of DBMS are the relational, network, hierarchy and object oriented models. Each kind of data structure has its own pros and cons. while each is unique in its own way, there are some standard functions of a DBMS, and those are

a) The ability to update and retrieve data

This is a fundamental component of a DBMS and essential to database management. Without the ability to view or manipulate data, there would be no point to using a database system.

Updating data in a database includes adding new records, deleting existing records and changing information within a record. The user does not need to be aware of how DBMS structures this data, all the user needs to be aware of is the availability of updating and/or pulling up information, the DBMS handles the processes and the structure of the data on a disk.

b) Support concurrent updates

Concurrent updates occur when multiple users make updates to the database simultaneously. Supporting concurrent updates is also crucial to database management as this component ensures that updates are made correctly and the end result is accurate. Without DBMS intervention, important data could be lost and/or inaccurate data stored.

DBMS uses features to support concurrent updates Such as batch processing, locking, two-phase locking and time stamping to help make certain that updates are done accurately. Again, the user is not aware all this is happening as it is the database management system's responsibility to make sure all updates are stored properly.

c) Recovery of data

If system failure occurs, DBMS must provide ways to recover a database so that data is not permanently lost. There are times computers may crash, a fire or other natural disaster may occur or a user may enter incorrect information invalidating or making records inconsistent.

If the database is destroyed or damaged in any way, the DBMS must be able to recover the correct state of the database and this process is called Recovery. The easiest way to do this is to make regular backups of information. This can be done at a bet structured time so in the event a disaster occurs, the database can be restored to the state that it was last at prior to backup.

A disadvantage to this is any data or changes entered after the backup would be lost. A way to counteract this is to set the DBMS to provide a feature called Journaling. This involves keeping a log of all updates made to the database, it 1s maintained in a file separate from the database and can be obtained to re-update the database after it is recovered from the backup.

d) Security

Security is the prevention of unauthorized users accessing the database. DBMS uses features such as encryption, authentication, authorization and views to provide security to the database. In encryption DBMS converts the data in a database to an indecipherable format. No unauthorized person trying to access this information will be able to read it. Authorized users will be able to see it in normal form.

Authentication is a technique in which the database administrator can identify the person accessing the database. Authorized users are given passwords and successful entry of a valid password will allow the user entry into the database, if a password is not successfully entered, the user will be denied access. Authorization is a set of rules that the database administrator (DBA) sets up to specify levels of usage that individuals or groups are allowed to have. Some users may only be allowed viewing options, while others may be allowed to both view/make changes.

In some Circumstances, users may only be allowed to access certain pieces of the database, and be denied access to areas that does not relate to their specific needs. In these cases, the DBA will assign workgroups, and these workgroups will be assigned levels of access and permissions. In views DBA allows certain users the ability to view the tables or fields that pertain to them, any other view does not exist for them in the database, DBMS does this behind the scenes and to the user it appears that the information they see are the only existing data.

e) Data integrity

Data integrity is an important function in database management. This is a set of rules that DBMS provides to see that data integrity is enforced, thus avoiding incorrect or inconsistent data. Types of integrity that DBMS provides are data type, legal values and format. Key integrity also falls into this function of DBMS. This enforces that the primary key of a record remains unique.

Without any of the above listed functions, a database would not be able to work effectively. Each of these functions plays an important role in database management**.**

Database System Applications

Database is widely used all around the world in different sectors. Various applications of DBMS in these sectors are given below:

1. Banking: For customer information, accounts, loans and banking transactions**.**

2. Airlines: For reservations and schedule information: Airlines were first to use database in a geographically disturbed manner-terminals the situated around the world accessed the central database system through phone lines and other data networks.

3. Universities: For student information, course registrations and grades.

4. Credit card transactions: For purchases on create cards and generation of monthly statements.

5. Telecommunications: For keeping records of calls made, generating bills, maintaining balances on prepaid callhn8 cards and storing information about the communication networks.

6. Finance: For storing information about holdings, sales and purchase of financial instruments such as stocks and bonds.

7. Sales: For customer, product and purchase information.

8. Manufacturing: For management of supply chain and for tracking production of items in factories, inventories of items in warehouses/ stores and orders for items.

9. Human resources: For information about employees, Salaries, payroll taxes and benefits and for generation of paychecks.

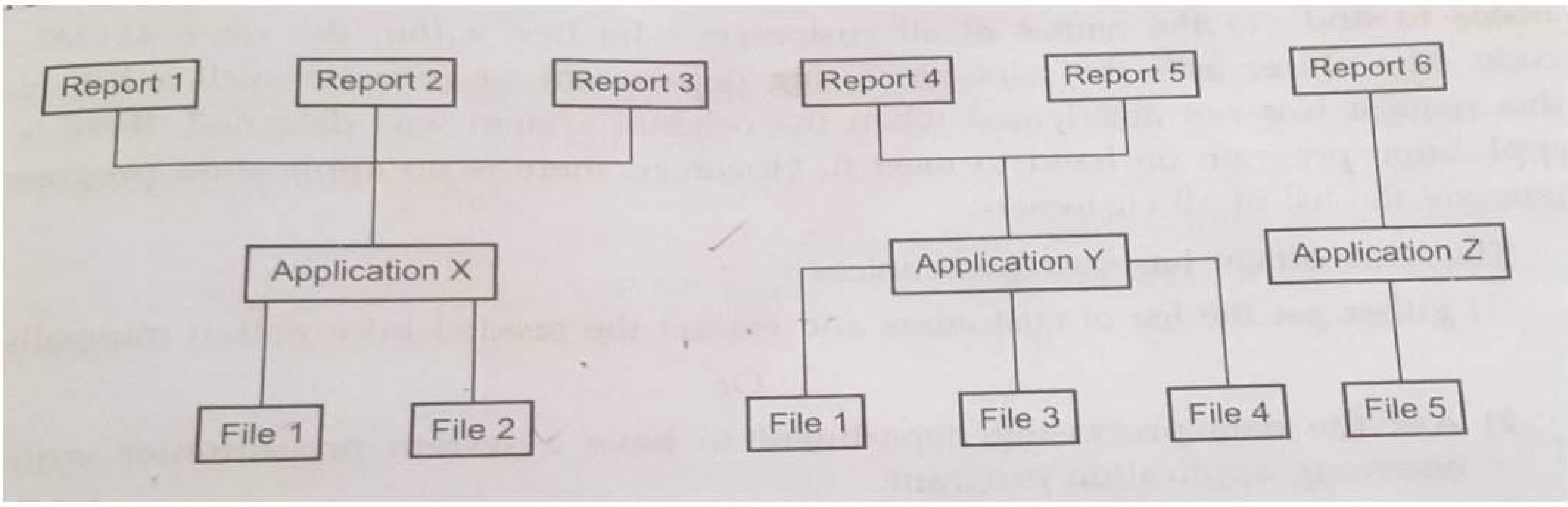
10. Web based services: For taking web users feedback, responses, and resource sharing etc.

File Processing System

Introduction to File Processing System

The information can be either a conventional file processing system or a database system. In the conventional file processing system, each and every subsystem of the information system will have its own set of files. As a result, there will be duplication of data between various subsystems of the information system. But, in database systems, there is a single centralized database which minimizes the redundancy of data to a greater extent.

The concept of the conventional file processing system is shown in Fig. which consists of three application/subsystems, namely Application X, Application Y and Application 2.



Conventional file processing system

The different inputs and outputs of these applications are summarized in below table

From Table, it is clear that some of the files are duplicated in different subsystems of the conventional file processing system.

Name of application	Input	Output
Application X	File 1 and File 2	Report 1, Report 2, Report 3
Application Y	File 1, File 3, File 4	Report 4, Report 5
Application Z	File 5	Report 6

Input and output of applications

Drawbacks of File Processing System

i) Data redundancy and inconsistency

Since the files and application programs are created by different programmers over long period of time, the files have different formats and the programs may be written in several programming languages. The same piece of information may be duplicated in several files. For e.g. the address and phone number of particular customer may appear n a file that consists of personal information and in saving account records file also. This redundancy leads to higher storage and access cost. It may lead to data inconsistency that is, the various copies of the same data may no longer agree.

For example, a changed customer address may be reflected in personal information file, but not in saving account records file.

ii) Difficulty in accessing data

Conventional file processing environments do not allow needed data to be retrieved in a convenient and efficient manner. for e-g: suppose that bank officer needs to find out the names of all customers Who live in the city’s 411027 zip code. The officer asks the data processing department to generate such a list. Since this request was not anticipated when the original 5ystem was designed, there is no application program on hand to meet it. However, there is no application program to generate the list of all customers.

The bank officer has now two choices

1) Either get the list of customers and extract the needed information manually,

Or

2) Ask the data processing department to have a system programmer write the necessary application program.

Both alternatives are unsatisfactory

ii) Data isolation

Since, data is scattered in various files, and files may be in different formats, it is difficult to write new application programs to retrieve appropriate data.

iv) Concurrent access anomalies

ln order to improve the overall performance of the system and obtain a faster response time many systems allow multiple users to update the data simultaneously. In such environment, interaction of concurrent updates may result in 1nconsistent data.

Consider bank account A, with $ 500 balance. If two customers withdraw funds (say $ 50 and $ 100 resp) from account A at the same time, the result of the concurrent executions may leave the account in an inconsistent state. The account may contain either $ 450 or $ 400, rather than $ 350. In order to guard against this possibility, some form of supervision must be maintained in the system.

v) Security problems

Not every user of the database system should be able to access all the data. For e.g in a banking system, pay roll personnel need only see that part of the database that has information about the various bank employees. They do not need access to information about customer accounts. Since application programs added to the system in an ad-hoc manner, it is difficult to enforce such security constraints.

vi) Integrity problems

The data values stored in the database must satisfy certain types of consistency constraints. For e.g, the balance of a bank account may never fall below a prescribed amount(say $ 100) these constraints are enforced in the system by adding appropriate code n the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem compounded when constraints involve several data items from different files.

Advantages and Disadvantages of DBMS

Advantages of DBMS

Database is a way to consolidate and control the operational data centrally, It is a better way to control the operational data. The advantages of having a centralized control of data are:

i) Redundancy can be reduced

In non-database systems, each application or department has its own private en resulting in considerable amount of redundancy of the stored data, 1hus storage pace is wasted. By having a centralized database most of this can be avoided.

ii) Inconsistency can be avoided

When he same data is duplicated and changes are made at one side, which in not propagated to the other site, it gives rise to inconsistency. Then the two entries regarding the same data will not agree. So, if the redundancy is removed, chances of having inconsistent data are also removed.

iii) The data can be shared

The data stored from one application, can be used for another application.

iv) Standards can be enforced

With central control of the database, the DBA can ensure that all applicable standards are observed in the representation of the data.

v) Security can be enforced

DBA can define the access paths for accessing the data stored in database and he can define authorization checks whenever access to sensitive data is attempted

vi) Integrity can be maintained

Integrity means that the data in the database is accurate. Centralized control of the data helps in permitting the administrator to define integrity constraints to the data in the database.

Disadvantages of DBMS

A database system generally provides on-line access to the database for many often designed to meet a specific need and therefore generally provides access to only a small number of users. Because and of the larger number of users accessing the data when a database is used, the enterprise may involve additional risks as compared to a conventional data processing system in the following areas:

i) Confidentiality, Privacy and Security

When information is centralized and is made available to users from remote locations, the possibilities of misuse are often more than in a conventional data processing system. To reduce the chances of unauthorized users accessing sensitive information, it is necessary to take technical, administrative and, possibly, legal measures.

Most databases store valuable information that must be protected must be protected against unauthorized access.

ii) Enterprise Vulnerability

Centralizing all data of an enterprise in one database may mean that the database becomes an indispensable resource. The survival of the enterprise may depend on reliable information being available from its database. The enterprise therefore becomes vulnerable to the destruction of the database or to unauthorized modification of the database.

iii) The Cost of using a DBMS

Database systems are costly as compare to conventional data processing systems.

The other disadvantages of DBMS are:

1) Database systems have few graphical or statistical capabilities.

2) Proprietary formats may limit archival quality of data.

3) Database systems require expertise and resources to administer.

4) Database damage

In most organizations, all data is integrated into single database. If database is damaged due to electric failure or database corruption, valuable data is lost forever.

Database System Vs File System

- The DBMS allows access to tables at a time. A File Management System allows access to single file at a time. File Management System's accommodate flat files that have no relation to other files.
- A DBMS co-ordinates the physical and logical access to the data; a file processing system only co-ordinates physical access to the data.
- A DBMS reduces the amount of data duplication; files often have redundant or duplicate data items.
- A DBMS is designed to allow flexibility in what queries give access to the data; a file processing system only allows pre-determined access to data (by specific compiled programs).
- A DBMS is designed to co-ordinate and permit multiple users to access data at the same time; a file processing system is much more restrictive in simultaneous data access.
- Database has a unique key or index (e.g. social security number) in order to access data directly or randomly. This allows the data to be segregated into multiple databases, but the key allows the user to choose data from multiple databases to produce the desired output. Files do not have keys or indices in order to find data rapidly.
- DBMS is collection of related tables; File system is a collection of related records.

Abstraction Data

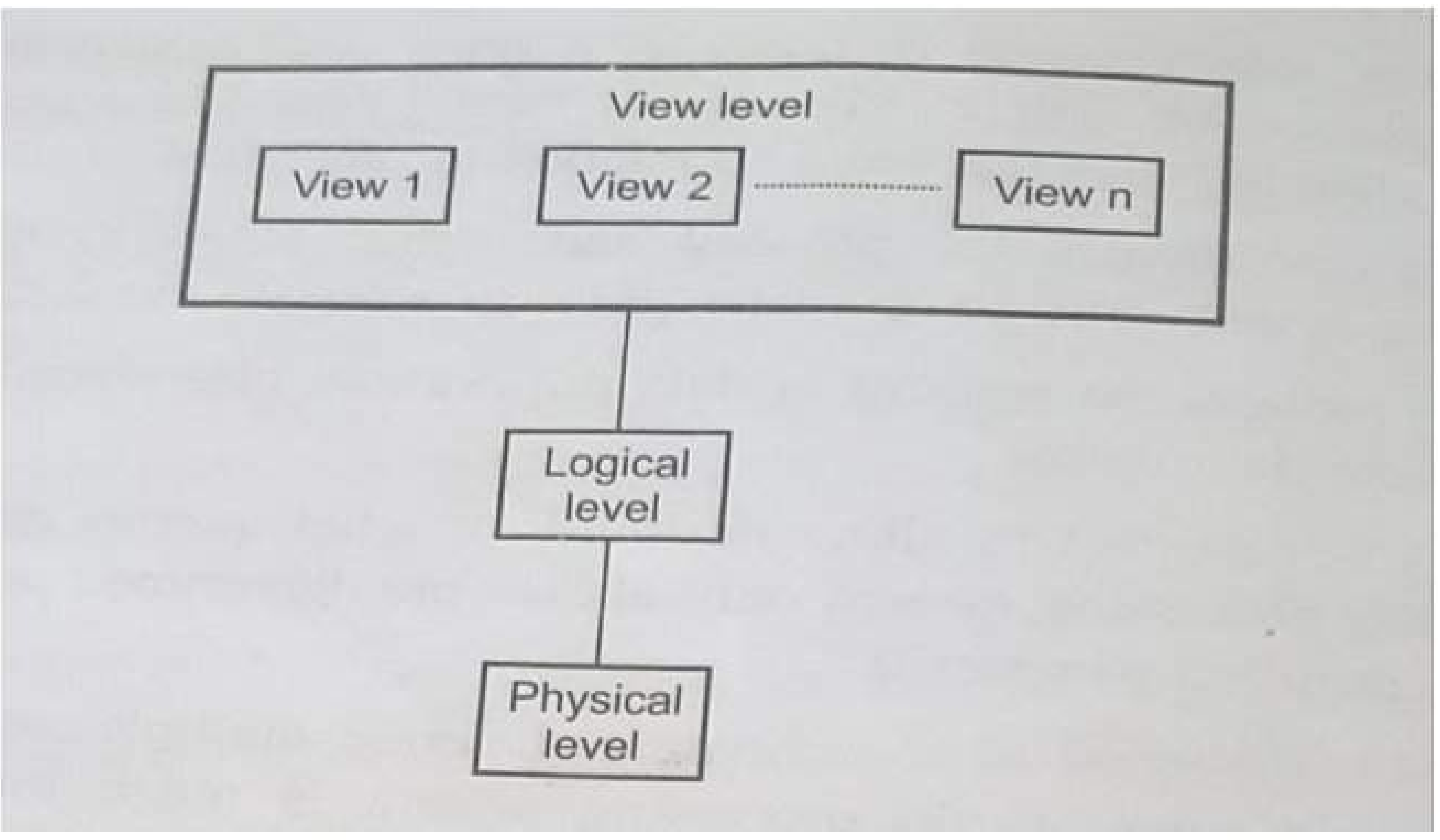
A major purpose of a database system is to provide users with an abstract view of the data. That 1s, the system hides certain details of how the data are stored and maintained.

There are three levels of data abstraction

i) Physical level: It is the lowest level of abstraction that describes how the data are actually stored. The physical level describes complex low-level data structures in detail.

ii) Logical level: It is the next higher level of abstraction that describes what data are stored in the database and what relationships exist among those data.

ii) View level: It is the highest level of abstraction that describes only part of the entire database.



The three levels of data abstraction

For example, consider a banking example with records:

- Account, with fields Acc_no, Balance
- Employee, with fields Employee_name, and Salary
- Customer, with fields Customer_name, and Customer_id, Address.

At the physical level, a Customer, Account, or Employee record can be as a block of consecutive storage locations (for example, words or bytes). The language compiler hides this level of detail from programmers.

At the logical level, each record is specified by type definition. For example, we declare a record as

stuct Customer

{

```
int Customer_id
char Customer_name[20];
char Customer_address[30];
};
```

Programmers using a programming language work at this level of abstraction Similarly database administrators usually work at this level of abstraction.

Finally, at the view level, computer users see a set of application programs that hide details of the data types.

Instances and Schemas-

Databases change over imes as information is inserted and deleted. The collection of information stored n the database at a particular moment is called an instance of the database.

The overall design of the database is called the database schema

A database schema corresponds to the variable declarations in a program. Each variable has a particular value at a given instant. The values of the variables in a program at a point in time correspond to an instance of a database schema.

Types of database schemas

i) Physical schema: It describes the database design at the physical level.

ii) Logical schema: It describes the database design at the logical level.

iii) Subschema: A database may also have several subschema's at the view level, called as subschema's, that describe different views of the database.

Data Models

Underlying structure of the database is called as data model. It is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

Different types of data models are

- Entity relationship model
- Relational model
- Hierarchical model
- Network model
- Object oriented model
- Object relational model.

ER Model

The entity relationship (E-R) model consists of a collection of basic objects, called entities and of relationships among these entities.

- **Entity**: An entity is a thing' or object in the real world that is distinguishable from other objects. For example, each person is an entity, and bank accounts is an entity.

Entities are described in a database by a set of attributes. For example, the attributes of account entity are account no and balance.

- **Relationship**: A relationships an association among several entities. For example, a depositor relationship associates a customer with each account that she has. The set of all entities of same type and the set of all relationships of the same type are termed as an entity set and respectively.

The overall logical structure of a database can be expressed graphically by an E-R diagram set, which consists of following components.

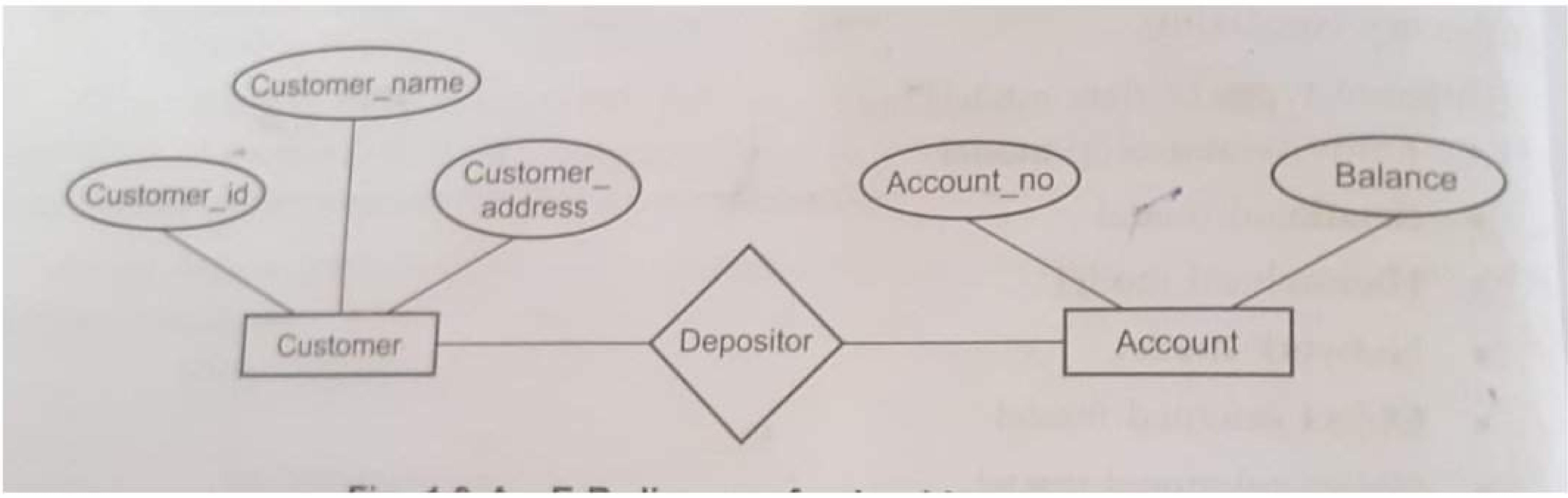
Rectangles, which represent entity sets.

Ellipses, which represent attributes

Diamonds, which represents relationships among entity sets

Lines, which link attributes to entity sets and entity sets to relationships.

Following Fig. shows an E-R diagram for banking system consisting of customers and accounts. The E-R diagram indicates that there are two entity sets, Customer and Account with attributes Customer_id, Customer _name, Customer_address of entity set Customer, and attributes Account_ no, Balance of entity set Account. The diagram also shows a relationship Depositor between Customer and Account.



An E-R diagram for banking system

In addition to entities and relationships, the E-R model also represents certain constraints to which the contents of a database must conform. One important constraint is mapping cardinalities, which express the number of entities to which another entity can be associated via a relationship set. For example, it each account must belong to only one customer, the E-R model can express that constraint.

The entity relationship model is widely used in database design.

Advantages:

- It is easy to develop relational model using E-R model.
- E-R model specifies mapping cardinalities.
- It specifies keys like primary key.
- We can specify generalization and specialization using E-R diagram.

Disadvantage;

- It is just used for database design not for implementation.

Relational Model

The relational model represents data and relationships among data by a collection of tables, each of which has a number of columns with unique names.

Following Fig shows a sample relational database showing customers and accounts they have. For example, the customer Smith lives on Sidehill in Brooklyn, and has two accounts 10l and 249. The account no. 101 has balance $2000 and other numbered with 249 has balance S3000.

Table name : Customer

Name	**Street**	**City**	**Account_No**
John	North	Queens	900
Smith	Sidehill	Brooklyn	101
Smith	Sidehill	Brooklyn	249
Kim	Lake view	Perryridge	303

Table name: Account_Info

Account_No	**Balance**
900	500
101	2000
249	3000
303	900

A sample relational database

Advantages:

The major advantages of the relational model are

- **Structural independence**: When it is possible to make change to the database structure without affecting the DBMS's capability to access data, we can say that structural independence have been achieved.

In relational database, changes in the database access. So relational database has structural independence.

- **Conceptual simplicity**: The relational database model is simpler at the conceptual level. Since the relational data model frees the designer from the physical data storage details, the designers can concentrate on the logical View of the database.
- **Design, implementation, maintenance and usage:** The relational database model achieves both data independence and structural independence making the database design, maintenance administration and usage much easier than the other models.
- Good for ad hoc requests
- It is simpler to navigate
- Greater flexibility.

Disadvantages

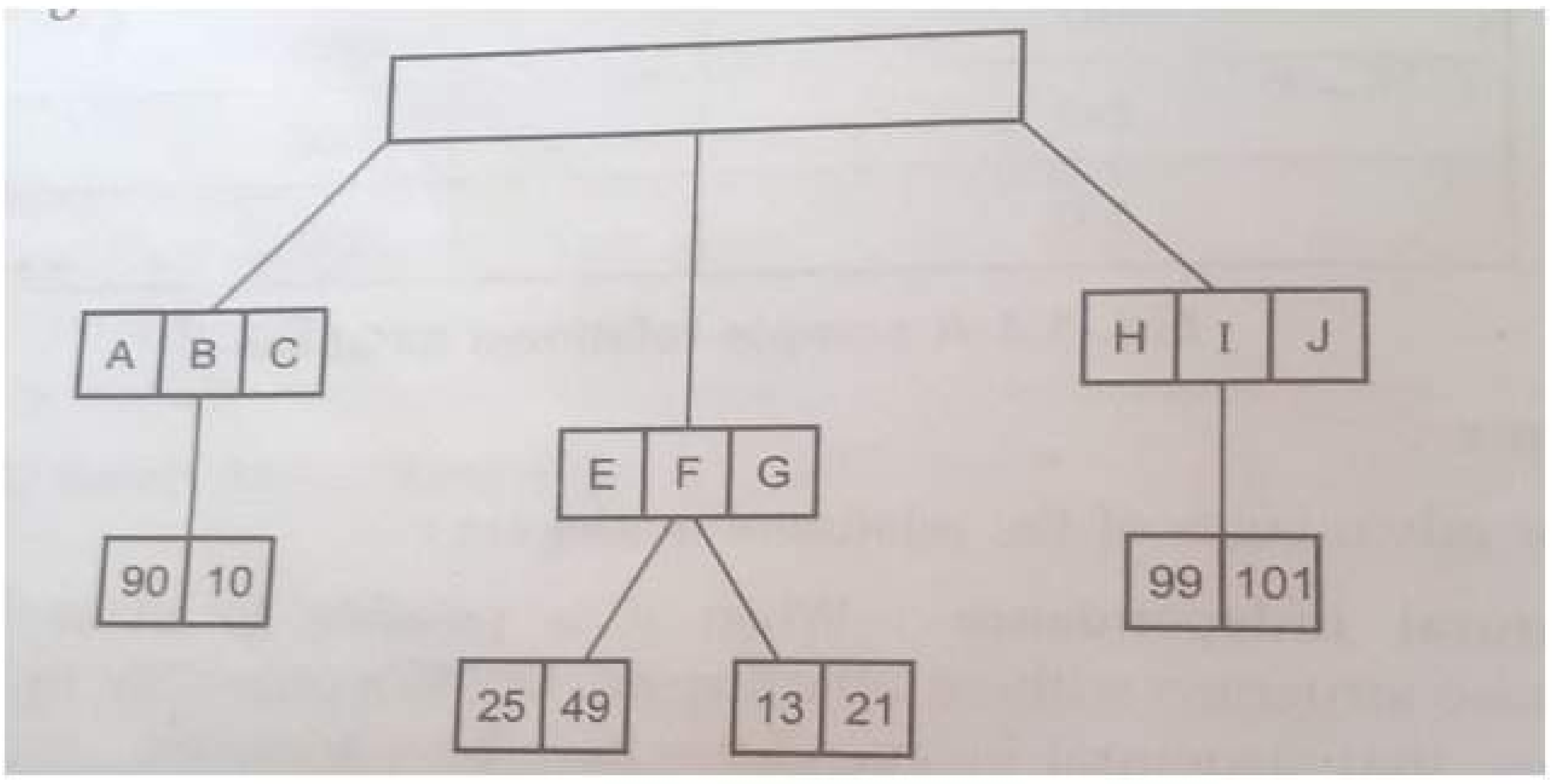
- Significant hardware and software overheads.
- Not as good for transaction process modeling as hierarchical and models.
- May have slower processing times than hierarchical and network model.

Hierarchical Model

A hierarchical database is a kind of database management system the records together in a tree data structure such that each record type has only one owner, e.g. an order is owned by only one customer.

Hierarchical structures were widely used in the first main frame database management systems.

Following Fig. shows a sample hierarchical database model.



A sample hierarchical database

Advantages

- High speed ot access to large datasets.
- Ease of updates.
- Simplicity: The design of a hierarchical database is simple.
- Data security :Hierarchical model was the first database model that offered the data security that is provided and enforced by the DBMS.
- Efficiency: The hierarchical database model is a very efficient one when the database contains a large number of 1: n relationships and when the users require large number of transactions, using data whose relationships are fixed.

Disadvantages

- **Implementation complexity**: Although the hierarchical database model is conceptually simple and easy to design, it is quite complex to implement.
- **Database management problems**: If you make any changes in the database structure of a hierarchical database, then you need to make the necessary changes in all the application programs that access the database. Thus maintaining the database and the applications can become very difficult.
- **Lack of structural independence**: Hierarchical database system use physical storage paths to navigate to the different data segments. So if the physical structure is changed

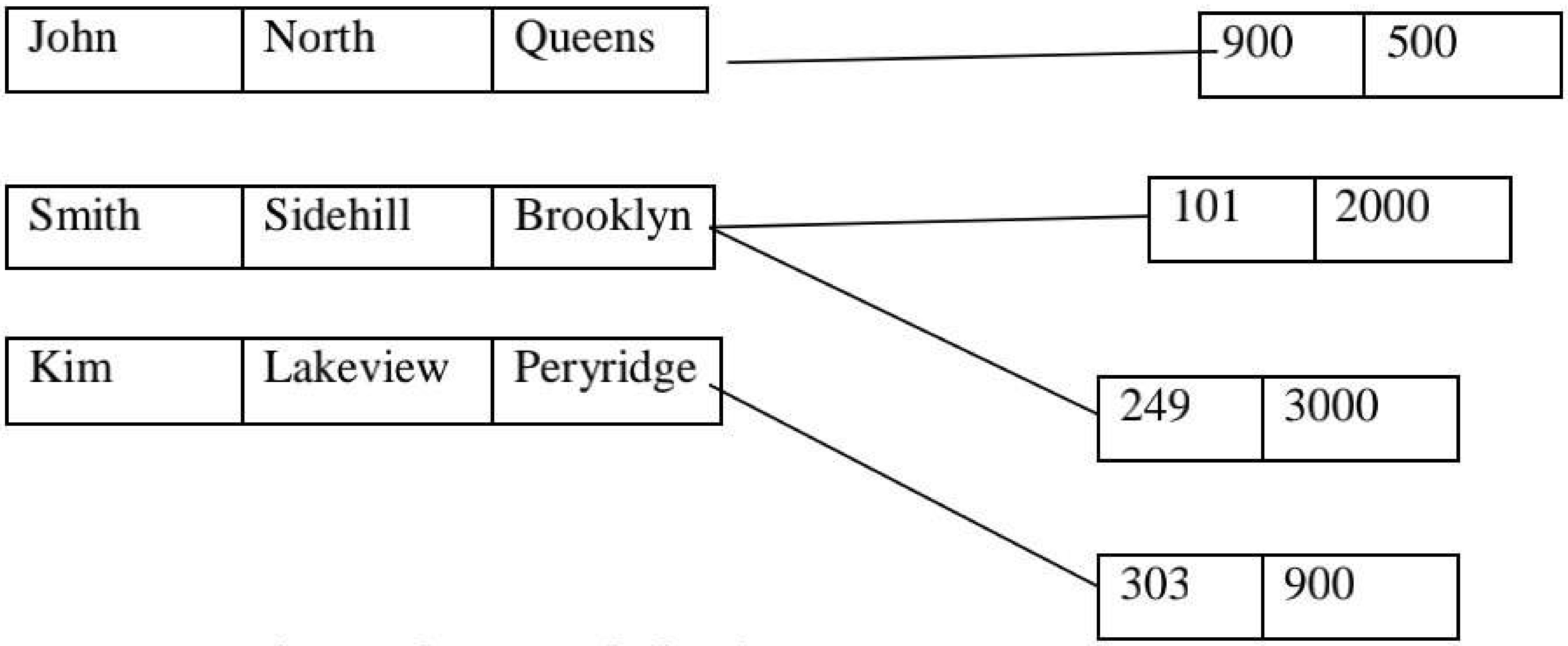
the applications will also have to be modified. Thus, in a hierarchical database the benefits of data independence are limited by structural dependence.

Linkages are only possible vertically but not horizontally or diagonally, i.e. there is no relation between different trees at the same level unless they share the same parent.

Network Model

The network model is based on directed graph theory. The network model replaces the hierarchical tree with a graph thus allowing more general connections among the nodes. The main difference of the network model from the hierarchical model is its ability to handle many-to-many (n: n) relationships or in other words, it allows a record to have more than one parent. Example is, an employee working for two departments.

shows a sample network model.



A sample network database

Advantages

- **Conceptual simplicity**: Just like the hierarchical model, the network model is also conceptually simple and easy to design.
- **Capability to handle more relationship types**: The network model can handle the one-to-many (1 : n) and many-to-many (n: n) relationships.
- **Data independence**: The changes in data characteristics do not require changes to the application programs.

Disadvantages

- Detailed structural knowledge is required.
- Lack of structural independence.

Object-Oriented Model

The object-oriented model is based on a collection of objects. An object Contain values stored in instance variables within the object. An object also contains bodies of code that operate on the object. These bodies of code are called methods. Objects that contain the same types of values and the same methods are grouped together into classes.

Advantages

- Applications require less code.
- Applications use more natural data model.
- Code is easier to maintain.
- It provides higher performance management of objects and complex interrelationships between objects.
- Object-oriented features improve productivity.
- Data access is easy.

Object Relational Model

System that includes both object infrastructure and a set of relational extenders is called an object relational model. Object-relational systems combine the advantages of modern object-oriented programming languages with relational database features such as multiple views of data and a high level, nonprocedural query language. Some of the object-relational systems available in the market are IBM's DE2 universal server, oracle corporations oracle 8, Microsoft Corporations SQL server 7 and so on.

Comparison between various Database Models

Model	Data element organization	Relationship representation	Identity	Access language
Hierarchical	FIles, records	Tree	Record based	Procedural
Network	FIles, records	Graph	Record based	Procedural
Relational	Tables	Foreign key concept	Value based	Non-procedural
Object-oriented	Objects	Logical containment	Record based	Procedural
Object-relational	Objects	Relational extenders	Value based	Non-procedural

Various database models

Database Languages

A database system provides a data definition language to specify the database schema and a data manipulation language to express database queries and updates.

Data Definition Language (DDL)

We specify a database schema by a set of definitions expressed by a special language, called a Data Definition Language (DDL).

DDL is a set ot SQL commands used to create, modify and delete database structures but not data. These commands are not normally used by a general user, who should be accessing the database via an application. They are normally used by the DBA to a limited extent, a database designer, or application developer.

Examples of DDL commands are

- **Create**: To create objects in the database

Example:

```
Create table account

(account_no char(10),

balance integer);
```

- **Alter**: Alter the structure of the database.
- **Drop**: Deletes objects from the database.
- **Truncate**: Removes all records from table, including all spaces allocated for the records are removed.
- **Comment**: Add comments to the data dictionary.

Data Manipulation Language

Data manipulation is

- The retrieval of information stored in the database.
- The insertion of new information into the database.
- The deletion of information from the database.
- The modification of information stored in the database.

A Data Manipulation Language (DML) 1s a language that enables users to access or manipulate data as organized by the appropriate data model.

Types of Data Manipulation Languages

There are two types of DMLs

i) Procedural DMLs: Require a user to specify what data are needed and how get those data.

ii) Declarative DMLs (Nonprocedural DMLS): Require a user to specify what data are needed without specifying how to get those data.

Declarative DMLs are easier to learn and use than procedural DMLs. The DML component of the SQL language is nonprocedural.

Example of DML commands are

Insert: Insert data into a table.

For example

```
insert into account values
  ('A101,1000);
```

Above SQL DML statement inserts a record with values account_no = 'A101’ and balance = 1000 into account table.

Update: Updates existing data within a table.

Delete: Deletes all records from a table.

Query: A query is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a query language. Following query in the SQL language displays account information of account _no = ‘A101’.

```
 select account no, balance
from account
where account no = "A101;
```

Database Users and Administrator

People who work with a database can be categorized as:

- Database users
- Database administrators.

Database Users

There are four different types of database system users, differentiated by the way they interact with the system:

a) Naive users

They are unsophisticated users who interact with the system by invoking one or the application programs that have been written previously. For example, a bank teller who needs to transfer $50 from account A to account B, invokes a program called transfer. This program asks the teller for the amount of money to be transferred, and the account to which the money is to be transferred.

The typical user interface for naive users is a forms interface, where the user can fill in appropriate fields of the form. Naive users may also simply read reports generated from the database.

b) Application programmers

There are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. Rapid Application Development (AD) tools are tools that enable an application programmer to construct forms and reports without writing a program. Application programmers also use fourth generation languages to facilitate the generation of forms and the display of data on the screen.

c) Sophisticated users

They interact with the system without writing programs. Instead, they form their requests in a database query language. They submit each such query to a query processor, whose function is to break down DML statements into instructions that the storage manager understands. Analysts who submit queries to explore data in the database fall in this category.

Online analytical processing (OLAP) tools simplify analysts tasks by letting them view summaries of data in different ways. For instance, an analyst can see the total sales by region (for example, North, South, East and west) or by product or by combination of region and product (that is, total sales of each product in each region).

Another class of tools for analyst is data mining tools, which help then find certain kinds of patterns in data.

d) Specialized users

These are sophisted users who write special1zed database applications that that do not fit into the traditional data processing frame work. Among these applications are computer aided design systems, knowledge base and expert systems, systems that store data with complex data types (for example, graphics data and audio data) and environment modeling systems.

Database Administrator

A person who has central control over the system is called a database administrator (DBA).

The functions of a DBA include:

Schema definition: The DBA creates the Original database schema by executing a set of data definition statements in the DDL.

Schema and physical organization modification: The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical Organization to improve performance.

Granting of authorization for data access: By granting different types of authorization, the DBA can regulate different users accessing different parts of database.

Routine maintenance: Examples of database administrator's routine maintenance activities are:

- Periodic backup, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.
- Ensuring that enough free disk space is available for normal operations and upgrading disk space as required.
- Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

Transaction Management

Database transaction is a collection of several operations that forms single task. Transaction is a reliable units of work that allow correct recovery from failures and keep a database consistent even cases of system failure.

A database transaction must be atomic, consistent, isolated and durable. Transactions provide an all-or-nothing proposition, stating that each work-unit Performed in a database must either complete in its entirety or have no effect whatsoever. Further, the system must isolate each transaction from other transactions, results must conform to existing constraints in the database and transactions that complete successfully must get written to durable storage.

A single transaction might require several queries, each reading and/or writing information in the database. When this happens it is usually important to be sure that database 1s not left with only some of the queries carried out. For example, when the doing a money transfer, if the money was debited from one account, it is important that it also De credited to the depositing account. Also, transactions should not interfere with each other.

A simple transactions is usually issued to the database system in a language like

SQL in this form:

1. Begin the transaction

2. Execute several queries

3. Commit the transaction

If the transaction fails at any point (possibly before getting to the Commit phase), the database system will rollback any changes. All other transactions will behave the same way as if the transaction had never existed

Concurrent Execution of Transactions

An important task of a DBMS is to schedule concurrent accesses to data so that each user can work in isolation with other user. For example, if a request that deposits cash into an account is submitted to the DBMS and at the same time another request that debits money from the same account is submitted to the database. These steps are executed in such a manner that they do not interfere with each other.

Advantages of Concurrent Executions

One of major objectives in developing a database is to create an information resource that can be shared by many users. If transactions execute one at a time, i.e. serially, with each transaction doing a commit before the next one begins there is no problem of interference with one another's transactions. However, users often need to access data simultaneously. If all users are reading data, there is no way they can interfere with one another and there is no need tor concurrency control. If users are accessing different parts of the database, the transactions can concurrently without a problem. However, when two users try to make updates simultaneously on the same data, or one updates while another reads the same data, there may be of concurrency control mechanism.

Thus the concurrency control is a:

- Process of managing simultaneous Operations on the database having them interferes with one another.
- Prevents interference when two or more users are accessing database simultaneously and at least one is updating data.
- Although two transactions may De stored In themselves, interleaving operations may produce an incorrect result.

Incomplete Transactions and System Crashes

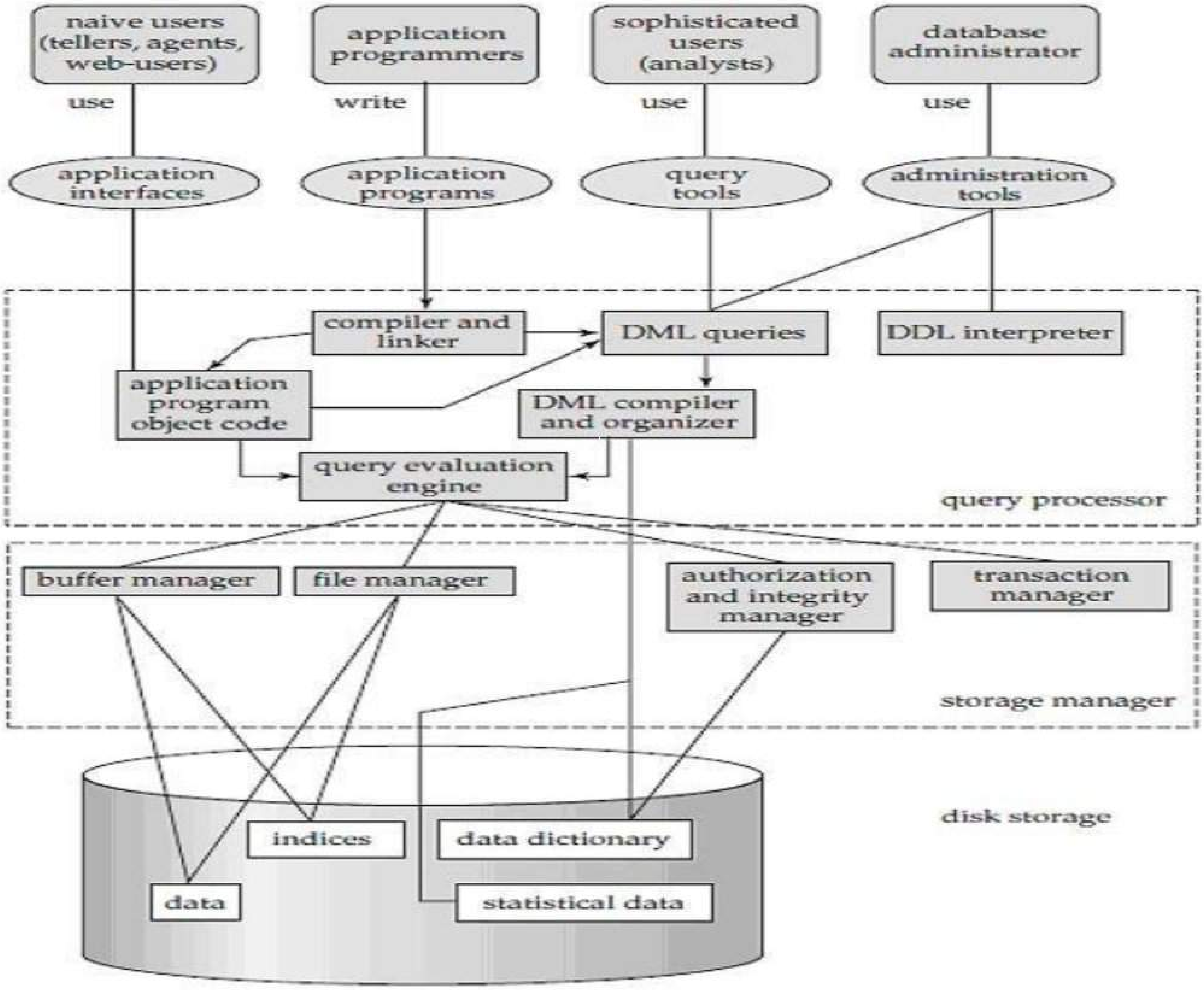
Transactions can be interrupted before running to completion for variety of reasons eg, a system crash, power failure etc. A DBMS must ensure-that changes the made by such incomplete transactions are removed from the database. For example, if the DBMS is in the middle of transferring money from account A to account B, i.e account A is debited but account B is not yet credited when the money debited from account A must be restored when the system comes back up after crash. To do so, the DBMS maintains a log of all writes to the data log of all writes to the database. Each write action must be recorded in the log (on disk) Before the corresponding Change is reflected in the database.

Database System Structure

A database system is partitioned into modules that deal with the responsibilities of the overall system. The functional components of a database can be broadly divided into the storage manager and the query processor components.

Storage Manager

A storage manager is a program module that provides the interface between the low level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file to the system. The storage manager translates the various DML Statements into low-level file system commands. Thus, the storage manager is responsible for storing, retrieving and updating data in the database.



Database System Structure

The various components of the storage manager are

- **Authorization and integrity manager:** It tests for satisfaction of various integrity constraints and checks the authority of users accessing the data.
- **Transaction manager** : It ensures that the database remains in a consistent state despite system failures, and concurrent executions proceed without conflicting
- **File manager**: It manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
- **Buffer manager**: It is responsible for fetching data from disk storage into main memory and deciding what data to cache in main memory.
- **Storage manager**: It implements several data structures as part of physical system implementation.
- **Data files**, which store the database itself.
- **Data dictionary**: It contains metadata that is data about data. The schema of a table is an example of metadata. A database system consults the consults the data dictionary before reading and modifying actual data.
- **Indices,** which provide fast access to data items that hold particular values.

Query Processor

The query processor is an important part of the database system. It helps the database system to simplify and facilitate access to data.

The query processor components include:

- **DDL interpreter**, which interprets DDL statements and records the definitions in the data dictionary.
- **DML compiler**, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.

A query can be translated into any number of evaluation plans that all gives the same result. The DML compiler also performs query optimization, that is, it picks up the lowest cost evaluation plan from among the alternatives.

- **Query evaluation engine**, which executes low-level instructions generated by the DML compiler.

Database Design and ER Diagrams

The entity-relationship (ER) data model allows us to describe the data in a real-world enterprise in terms of entities and their relationships and is widely used to develop an initial database design. The ER model is important primarily for its role in database design. It provides useful concepts that allow us to move from an informal description of what users want from their database to a more detailed and precise, description that can be implemented in a DBMS.

Overview of Database Design

The six steps of database design are given below, The ER model is most relevant to first three steps

1. Requirements Analysis: The first step in designing a database application to understand what data is to be stored in the database, what applications must be built on top of it, and what operations are performed on it. The purpose of this step is to produce a description of the users requirements.

2. Conceptual Database Design: The information gathered in the requirements analysis step is used to develop a high-level description of the data to stored in the database, along with the constraints over the data. This step creates a conceptual schema for the database using a high level conceptual data model. The conceptual schema is a concise description of the data requirements of the users and includes detailed descriptions of entity types, relationship types, and constraints. This step is often carried out often carried out using the ER model, or a similar high-level data model.

3. Logical Database Design: Purpose of this step is to transform the conceptual schema (which is at a high/abstract level) into a (lower-level) representational/implementational model supported by whatever DBMS is to be used. This step is also called data model mapping.

Beyond ER Design

ER modeling is sometimes considered as a complete approach for designing a logical database schema. This is incorrect as the ER diagram is just an approximate description of the data, Constructed using the information collected requirements analysis. A more careful analysis can often refine the logical schema obtained at the end of Step 3. Once we have a good logical schema, we must consider performance criteria and design the physical schema. Finally, we must address security issues and ensure that users are able to access the data they need, but not data that we wish to hide from them. The remaining three steps of database design are described below:

1. Schema Refinement: The purpose of this step is to analyze the collection of relations in our relational database schema to identity potential problems, and to refine it.

2. Physical Design: Purpose of this step is to decide upon the internal storage structures, access paths (indexes), etc., that will be used in realizing the representational model produced in previous phase.

3. Security Design: In this step, different user groups and different roles played by various users (e.g, the development team for a product, the customer support representatives, and the product manager) are identified. For each role and user group, the parts of the database that they must be able to access and the parts of the database that they should not be allowed to access are identified; Steps are taken to ensure that they can access only the necessary parts.

ER diagram

- ER diagram or Entity Relationship diagram is a conceptual model that gives the graphical representation of the logical structure of the database.
- It shows all the constraints and relationships that exist among the different components.

Components of ER diagram-

An ER diagram is mainly composed of following three components-

1. Entity Sets
2. Attributes
3. Relationship Set

Example-

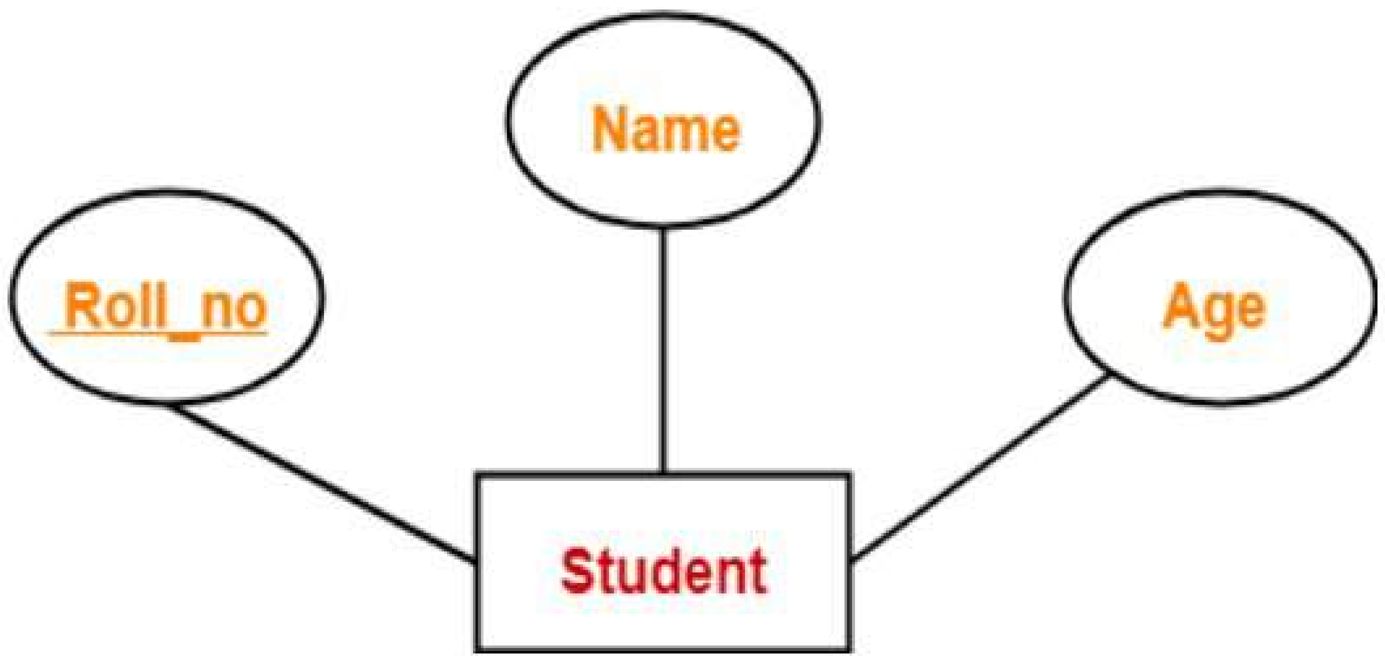
Consider the following Student table-

Roll_no	Name	Age
1	Akshay	20
2	Rahul	19
3	Pooja	20
4	Aarti	19

This complete table is referred to as "Student Entity Set" and each row represents an "entity".

Representation as ER Diagram-

The above table may be represented as ER diagram as-



Here,

- Roll_no is a primary key that can identify each entity uniquely.
- Thus, by using student’s roll number, a student can be identified uniquely.

ER Diagram Symbols-

An ER diagram is composed of several components and each component in ER diagram is represented using a specific symbol.

ER diagram symbols are discussed below-

Entities and Entity Sets

- An entity is thing or 'object in the real world that is distinguishable from all other objects For example, each person in an enterprise is an entity. An entity has a set of properties, and the values for some set of properties may uniquely 1dentify an entity. For instance, a Customer with Customer id property with value CI01 uniquely identifies that person.
- An entity may De concrete, such as person or a book, or it may be abstract such as a loan, or a holiday.
- An entity set 1s a set of entities of the same type that share the same properties, or attributes. The set of all customers at a given bank can De defined as the entity set Customer.

Following Fig shows two entity sets

i) Customer, with properties Customer_id, Customer_name, City.

ii) Acco1unt with properties Account_no and Balance.

Entity sets Customer and Account

Customer_id	Customer_name	City
C101	Hari	Bombay
C102	Ram	Pune
C104	John	Nasthik
C103	Jim	Solapur

Customer

Account_no	Balance
A121	1000
A305	2000
A417	500
A519	3000

Account

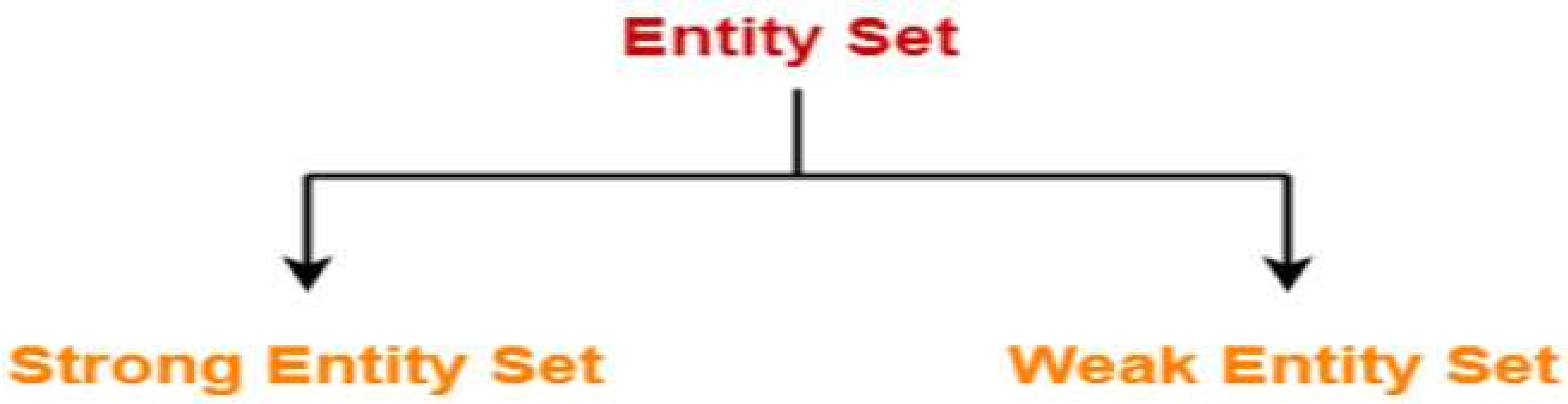
1. For Entity Sets-

An entity set is a set of same type of entities.

An entity refers to any object having-

- Either a physical existence such as a particular person, office, house or car.
- Or a conceptual existence such as a school or a company.

An entity set may be of the following two types-



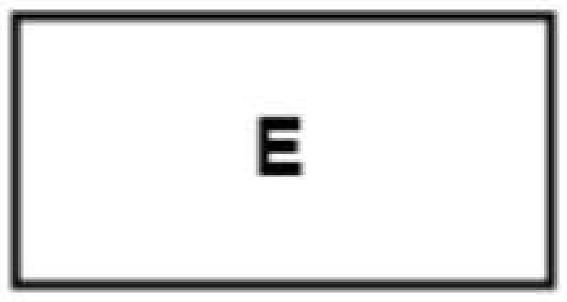
1. Strong entity set
2. Weak entity set

1. Strong Entity Set-

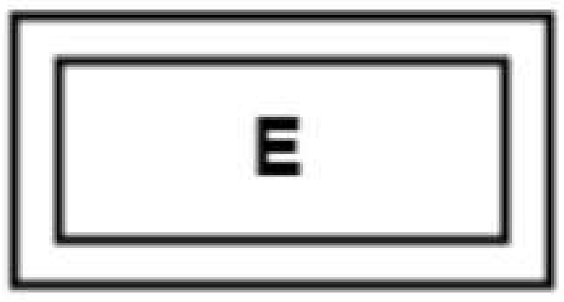
- A strong entity set possess its own primary key.
- It is represented using a single rectangle.

2. Weak Entity Set-

- A weak entity set do not possess its own primary key.
- It is represented using a double rectangle.



Strong Entity Set

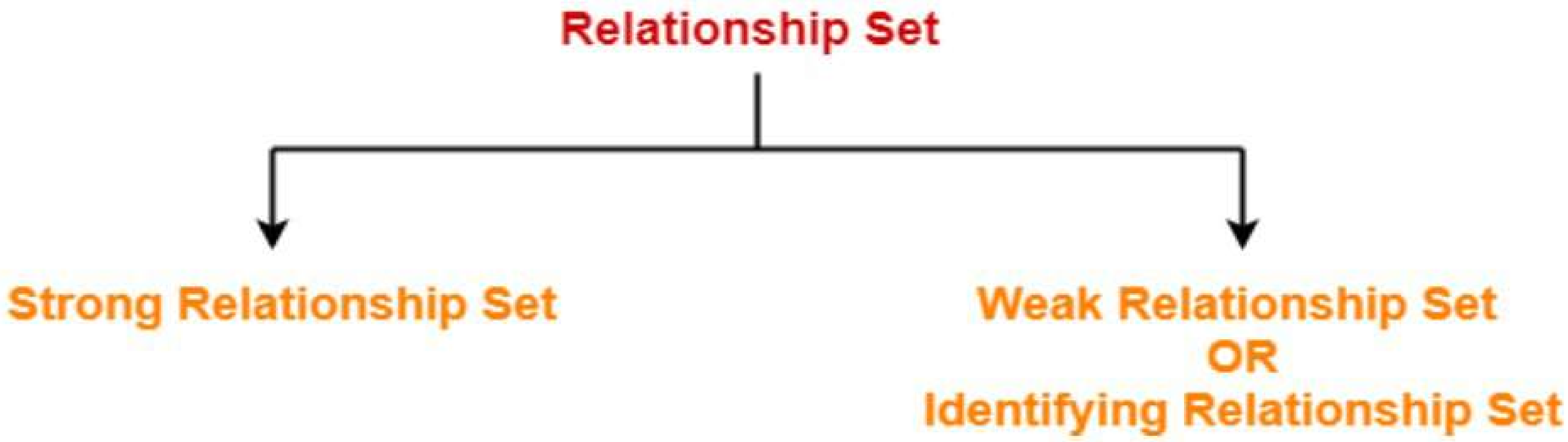


Weak Entity Set

2. For Relationship Sets-

- Relationship defines an association among several entities.
- A relationship set is a set of same type of relationships.

A relationship set may be of the following two types-



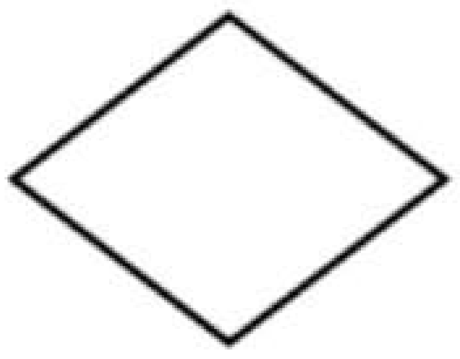
1. Strong relationship set
2. Weak relationship set

1. Strong Relationship Set-

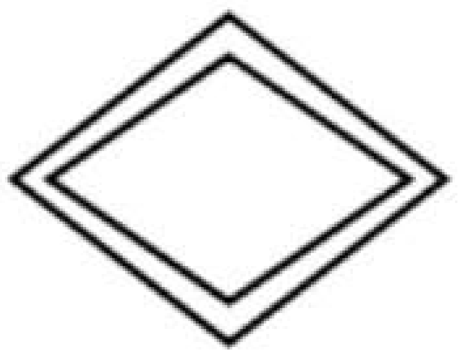
- A strong relationship exists between two strong entity sets.
- It is represented using a diamond symbol.

2. Weak Relationship Set-

- A weak or identifying relationship exists between the strong and weak entity set.
- It is represented using a double diamond symbol.



Strong Relationship Set



Weak or Identifying Relationship Set

For Attributes-

3. Attributes

- An entity is represented by a set of attributes. Attributes are descriptive properties possessed by each member of an entity set.
- The attributes of Customer entity set are Customer_id, Customer-name, and City possible attributes of the Account entity set are Account_no, and Balance.
- Each entity has a value for each of its attributes. For instance a particular Customer entity may have the value 'Cl0l' for attribute Customor_id, 'Hari' for Customer_name, and 'Bombay' for City.
- For each attribute, there is set of permitted values, called the domain, or value set of that attribute. The domain of attribute Customer_name might be set of all text strings of a certain length.

Types of attributes

Attributes are classified as:

- **Simple**
- **Composite**
- **Single-valued**
- **Multi-valued**
- **Derived**

i) Simple attribute: A simple attribute is an attribute composed of a single component with an independent existence. Simple attributes cannot be further subdivided. Examples of simple attributes include Roll_ No, Salary etc.

ii) Composite attribute: An attribute composed of multiple components each with an independent existence is called a composite attribute.

Examples of composite attributes are:

1) Name, which is composed of attributes like first name, middle name and last name.

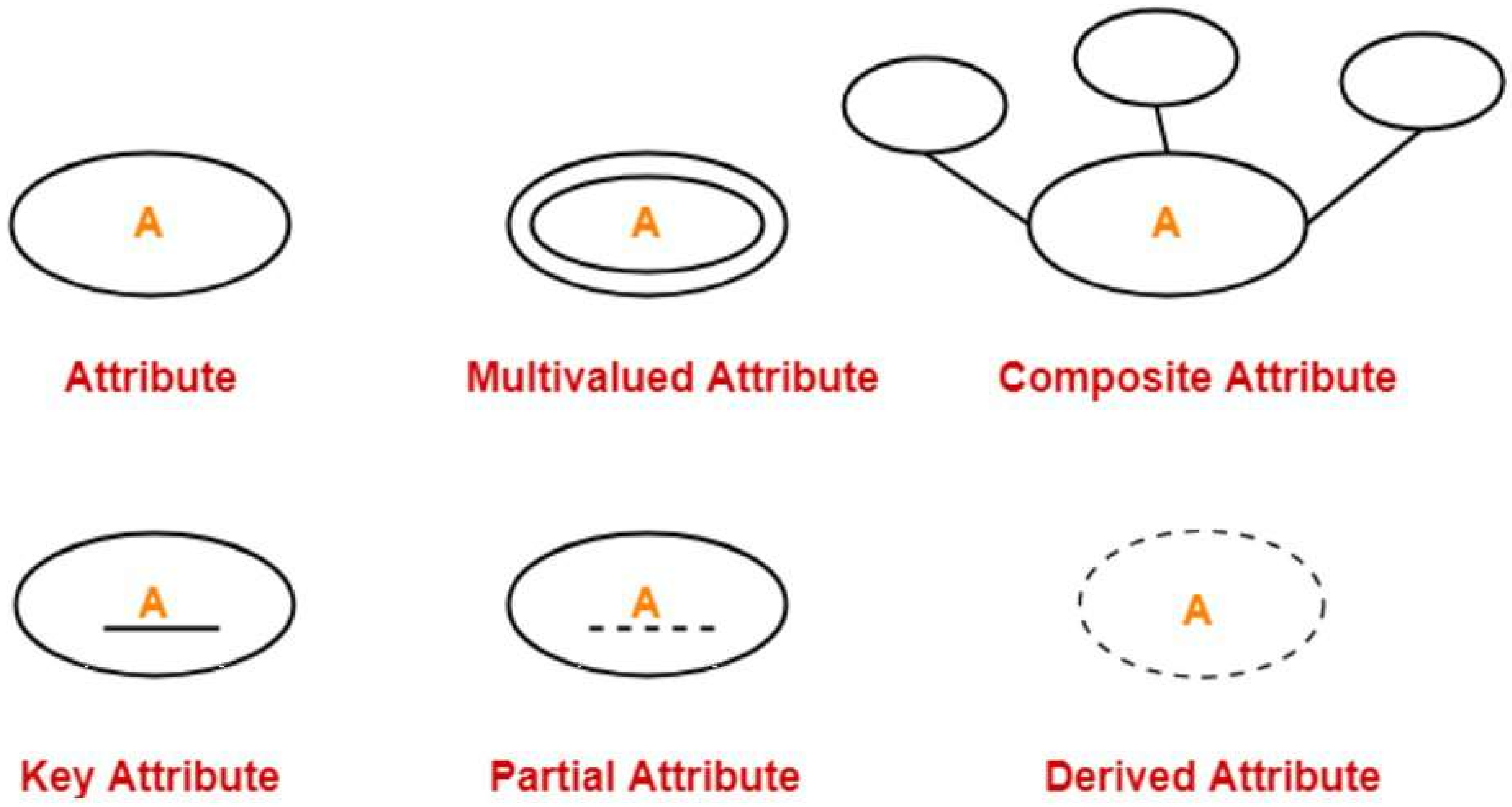
2) Address, which is composed of other components like Street, City, Pin code.

ii) Single-valued attributes: A single-valued attribute is one that holds a single value for a single entity. Examples are Room_no, Customer_id. Single-valued attributes are also called as atomic attributes.

iv) Multi-valued attributes: A multi-valued attribute is one that holds multiple values for a single entity. For example, a student entity can have multiple values for the Hobby attribute such as reading, music, and painting.

v) Derived attribute: A derived attribute is one that represents a value that is derivable from the value of a related attribute or set of attributes. For example, the age attribute can be derived from the date of birth attribute**.**

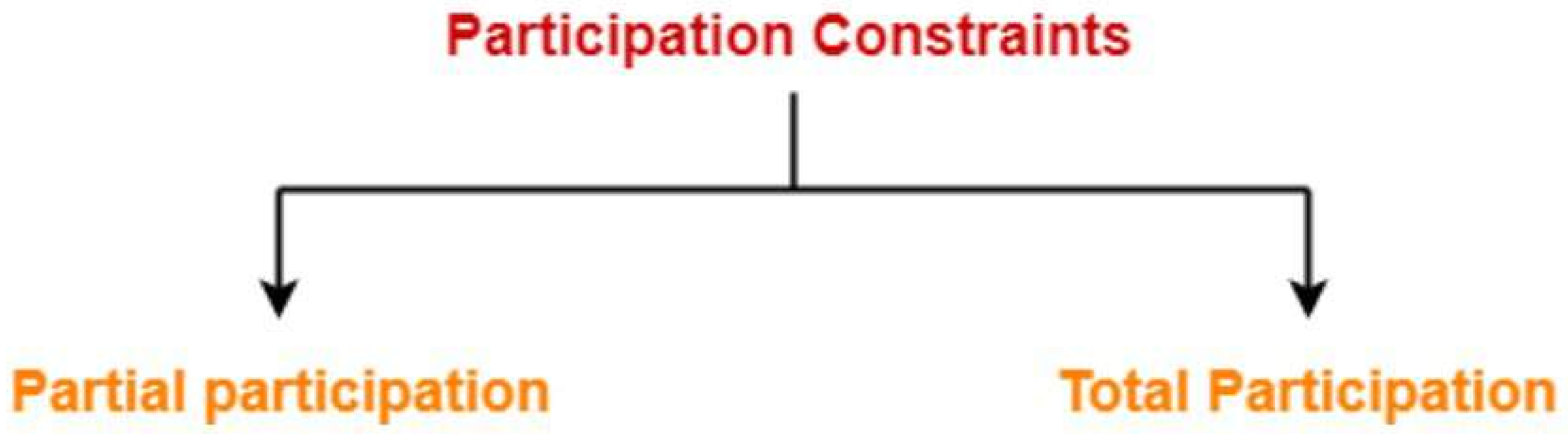
- Attributes are the properties which describe the entities of an entity set.
- There are several types of attributes.



4. For Participation Constraints-

Participation constraint defines the least number of relationship instances in which an entity has to necessarily participate.

There are two types of participation constraints-



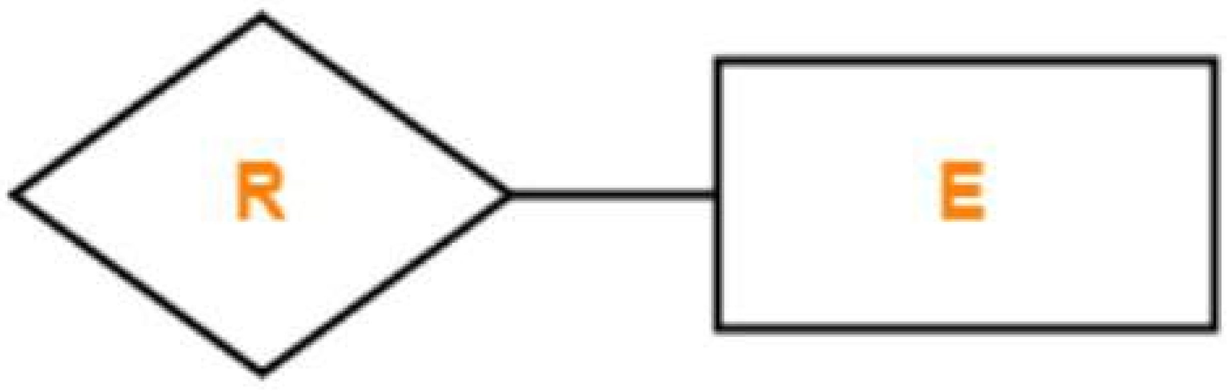
1. Partial participation
2. Total participation

1. Partial Participation-

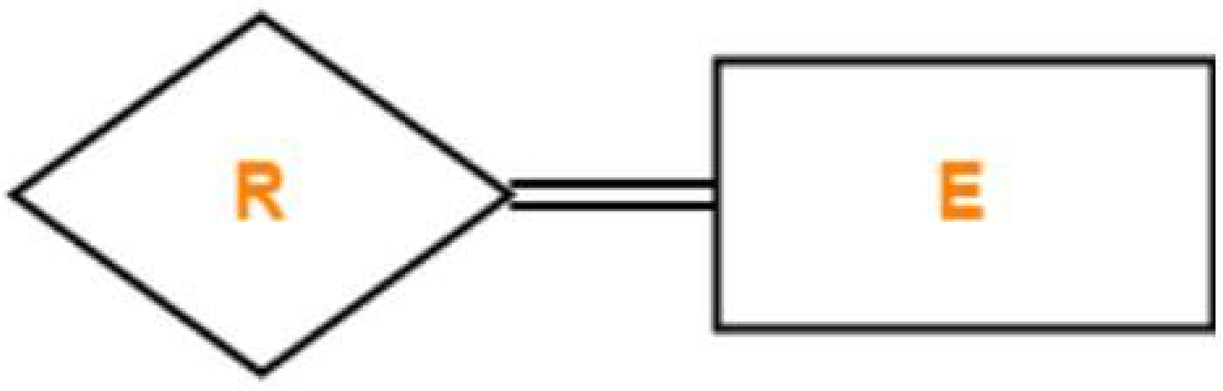
Partial participation is represented using a single line between the entity set and relationship set.

2. Total Participation-

Total participation is represented using a double line between the entity set and relationship set.



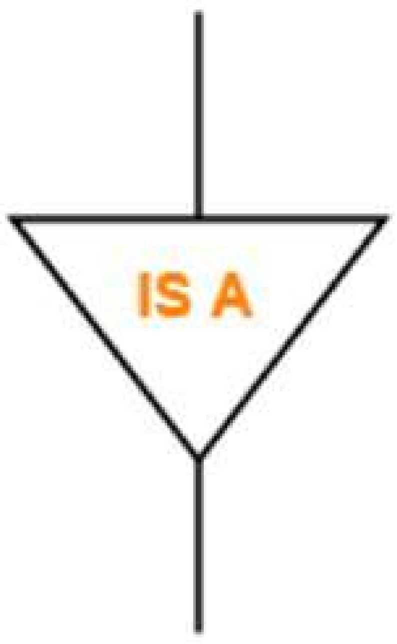
Partial Participation



Total Participation

5. For Specialization and Generalization-

- Generalization is a process of forming a generalized super class by extracting the common characteristics from two or more classes.
- Specialization is a reverse process of generalization where a super class is divided into sub classes by assigning the specific characteristics of sub classes to them.



IS A specialization or generalization

6. For Cardinality Constraints / Ratios-

- Either a physical existence such as a particular person, office, house or car.
- Or a conceptual existence such as a school or a company.

Entity Set in DBMS-

An entity refers to any object having-

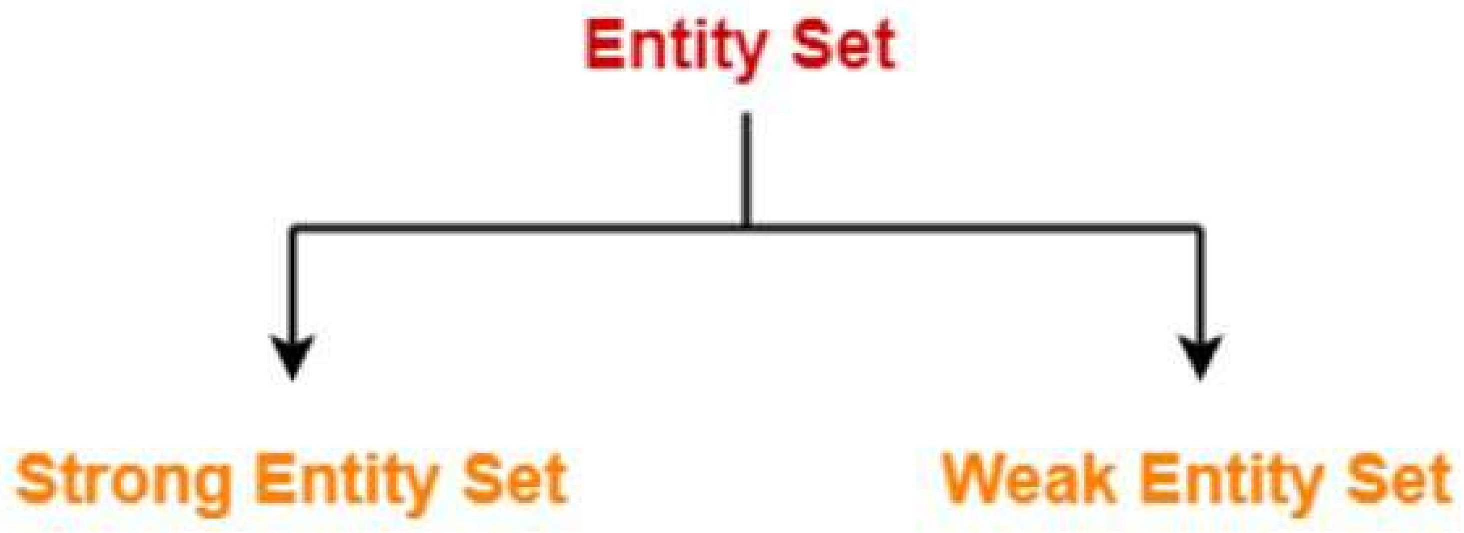
- Either a physical existence such as a particular person, office, house or car.
- Or a conceptual existence such as a school, a university, a company or a job.

In ER diagram,

- Attributes are associated with an entity set.
- Attributes describe the properties of entities in the entity set.
- Based on the values of certain attributes, an entity can be identified uniquely.

Types of Entity Sets-

An entity set may be of the following two types-



1. Strong entity set
2. Weak entity set

1. Strong Entity Set-

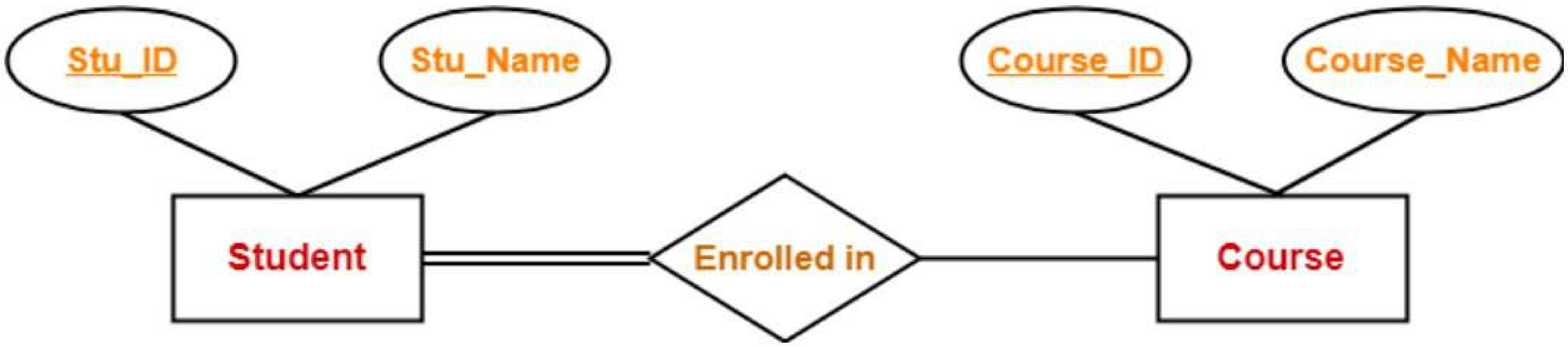
- A strong entity set is an entity set that contains sufficient attributes to uniquely identify all its entities.
- In other words, a primary key exists for a strong entity set.
- Primary key of a strong entity set is represented by underlining it.
-

Symbols Used-

- A single rectangle is used for representing a strong entity set.
- A diamond symbol is used for representing the relationship that exists between two strong entity sets.
- A single line is used for representing the connection of the strong entity set with the relationship set.
- A double line is used for representing the total participation of an entity set with the relationship set.
- Total participation may or may not exist in the relationship.

Example-

Consider the following ER diagram-



In this ER diagram,

- Two strong entity sets "**Student**" and "**Course**" are related to each other.
- Student ID and Student name are the attributes of entity set "Student".
- Student ID is the primary key using which any student can be identified uniquely.
- Course ID and Course name are the attributes of entity set "Course".
- Course ID is the primary key using which any course can be identified uniquely.
- Double line between Student and relationship set signifies total participation.
- It suggests that each student must be enrolled in at least one course.
- Single line between Course and relationship set signifies partial participation.
- It suggests that there might exist some courses for which no enrollments are made.

2. Weak Entity Set-

- A weak entity set is an entity set that does not contain sufficient attributes to uniquely identify its entities.
- In other words, a primary key does not exist for a weak entity set.
- However, it contains a partial key called as a **discriminator.**
- Discriminator can identify a group of entities from the entity set.
- Discriminator is represented by underlining with a dashed line.

NOTE-

- The combination of discriminator and primary key of the strong entity set makes it possible to uniquely identify all entities of the weak entity set.
- Thus, this combination serves as a primary key for the weak entity set.
- Clearly, this primary key is not formed by the weak entity set completely.

Primary key of weak entity set

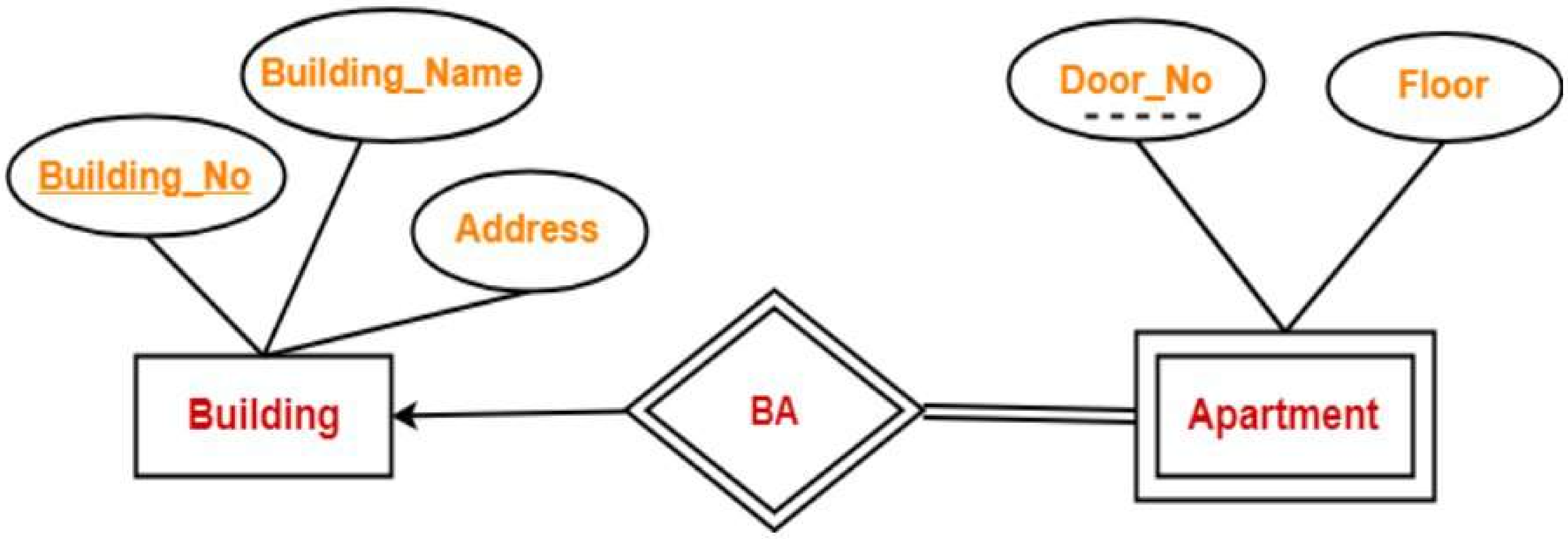
= Its own discriminator + Primary key of strong entity set

Symbols Used-

- A double rectangle is used for representing a weak entity set.
- A double diamond symbol is used for representing the relationship that exists between the strong and weak entity sets and this relationship is known as **identifying relationship**.
- A double line is used for representing the connection of the weak entity set with the relationship set.
- Total participation always exists in the identifying relationship.

Example-

Consider the following ER diagram-



In this ER diagram,

- One strong entity set "**Building**" and one weak entity set "**Apartment**" are related to each other.
- Strong entity set "Building" has building number as its primary key.
- Door number is the discriminator of the weak entity set "Apartment".
- This is because door number alone can not identify an apartment uniquely as there may be several other buildings having the same door number.
- Double line between Apartment and relationship set signifies total participation.
- It suggests that each apartment must be present in at least one building.
- Single line between Building and relationship set signifies partial participation.
- It suggests that there might exist some buildings which has no apartment.

To uniquely identify any apartment,

- First, building number is required to identify the particular building.
- Secondly, door number of the apartment is required to uniquely identify the apartment.

Thus,

Primary key of Apartment

= Primary key of Building + Its own discriminator.

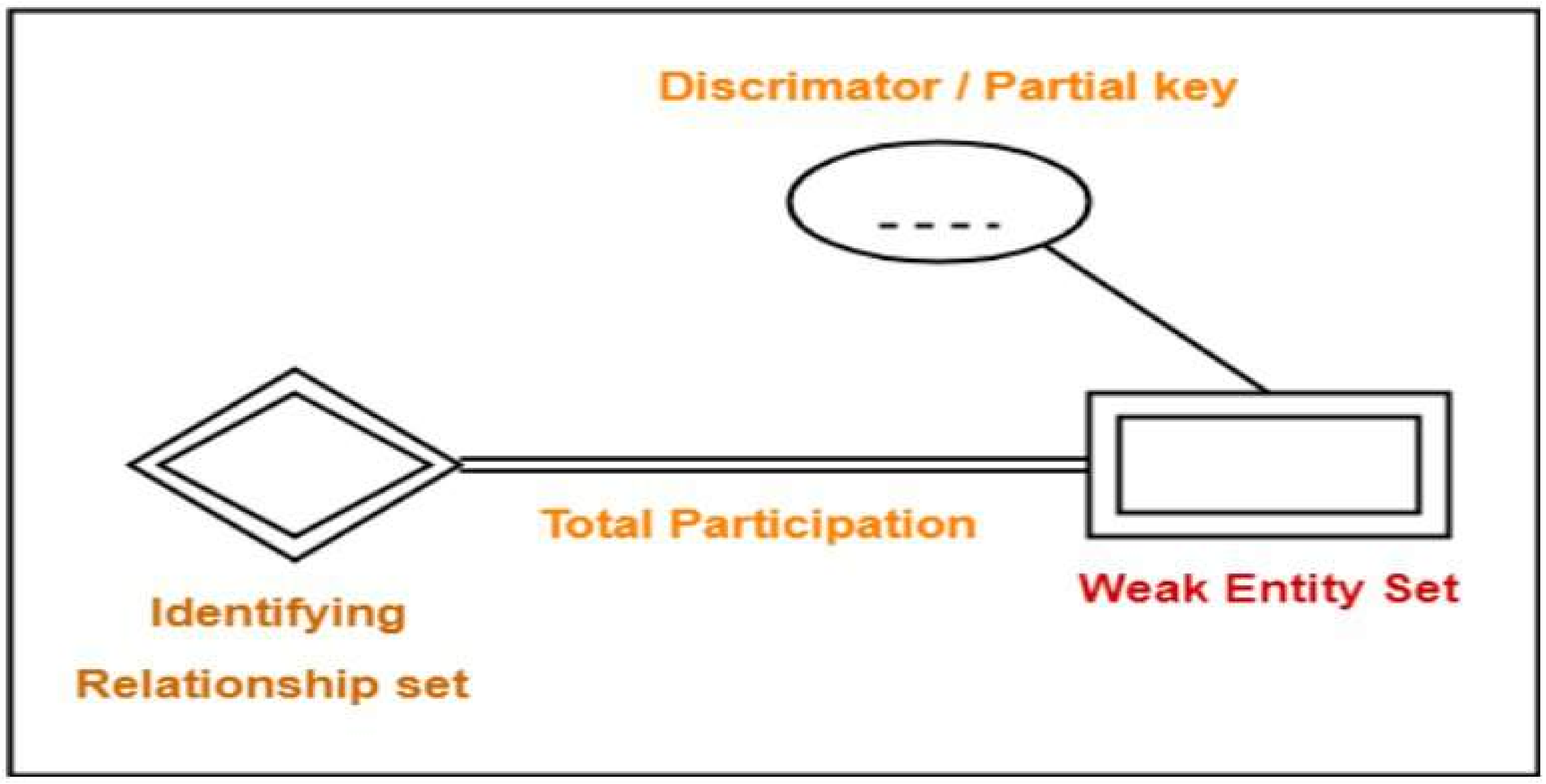
Differences between Strong entity set and Weak entity set-

Strong entity set	Weak entity set
A single rectangle is used for the representation of a strong entity set.	A double rectangle is used for the representation of a weak entity set.
It contains sufficient attributes to form its primary key.	It does not contain sufficient attributes to form its primary key.
A diamond symbol is used for the representation of the relationship that exists between the two strong entity sets.	A double diamond symbol is used for the representation of the identifying relationship that exists between the strong and weak entity set.
A single line is used for the representation of the connection between the strong entity set and the relationship.	A double line is used for the representation of the connection between the weak entity set and the relationship set.
Total participation may or may not exist in the relationship.	Total participation always exists in the identifying relationship.

Important Note-

In ER diagram, weak entity set is always present in total participation with the identifying relationship set.

So, we always have the picture like shown here-



Relationships and Relationship Sets

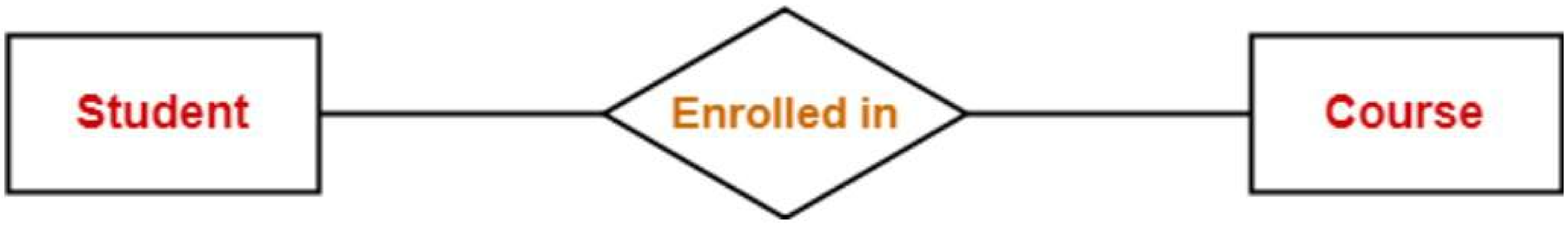
A relationship expresses an association among several entities. A relationship set is a set of relationships of the same type.

For example, consider two entities Person and Company as shown in below. The relationship works-for represents association between Person and Company. This is a binary relationship set.

Relationship in DBMS-

Example-

'Enrolled in' is a relationship that exists between entities **Student** and **Course**.

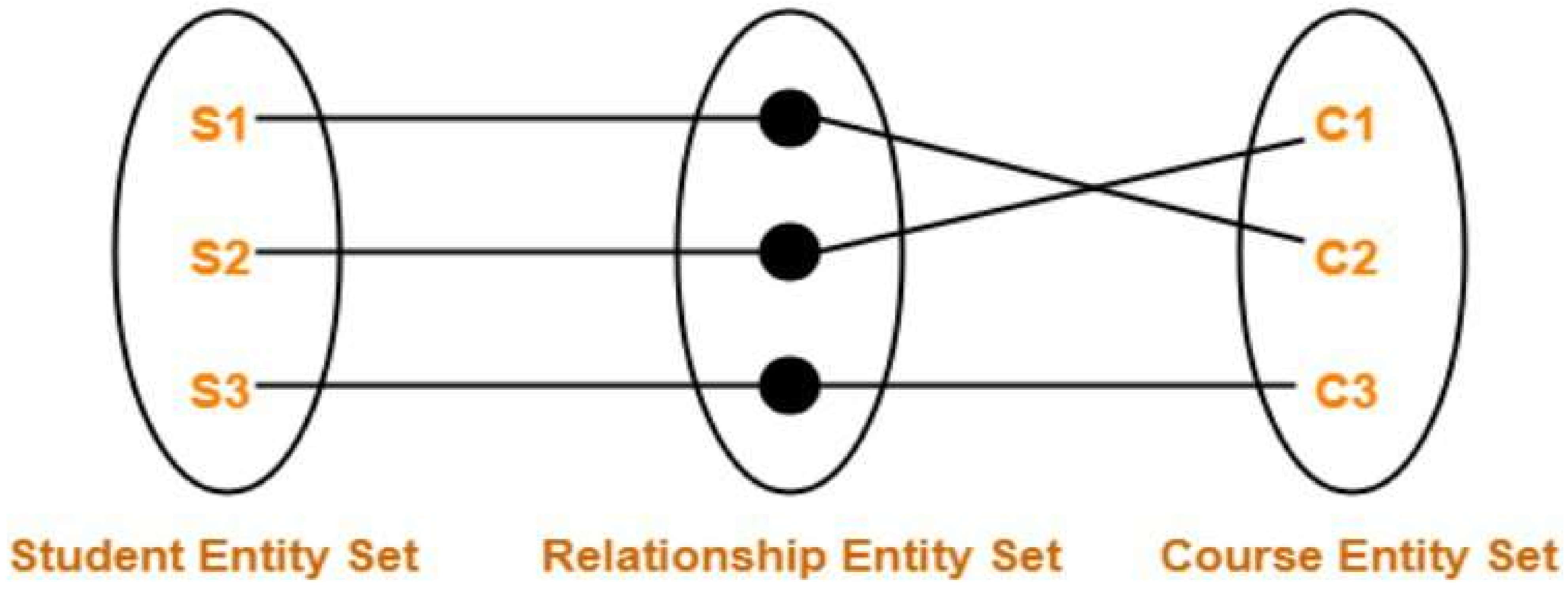


Relationship Set-

A relationship set is a set of relationships of same type.

Example-

Set representation of above ER diagram is-



Set Representation of ER Diagram

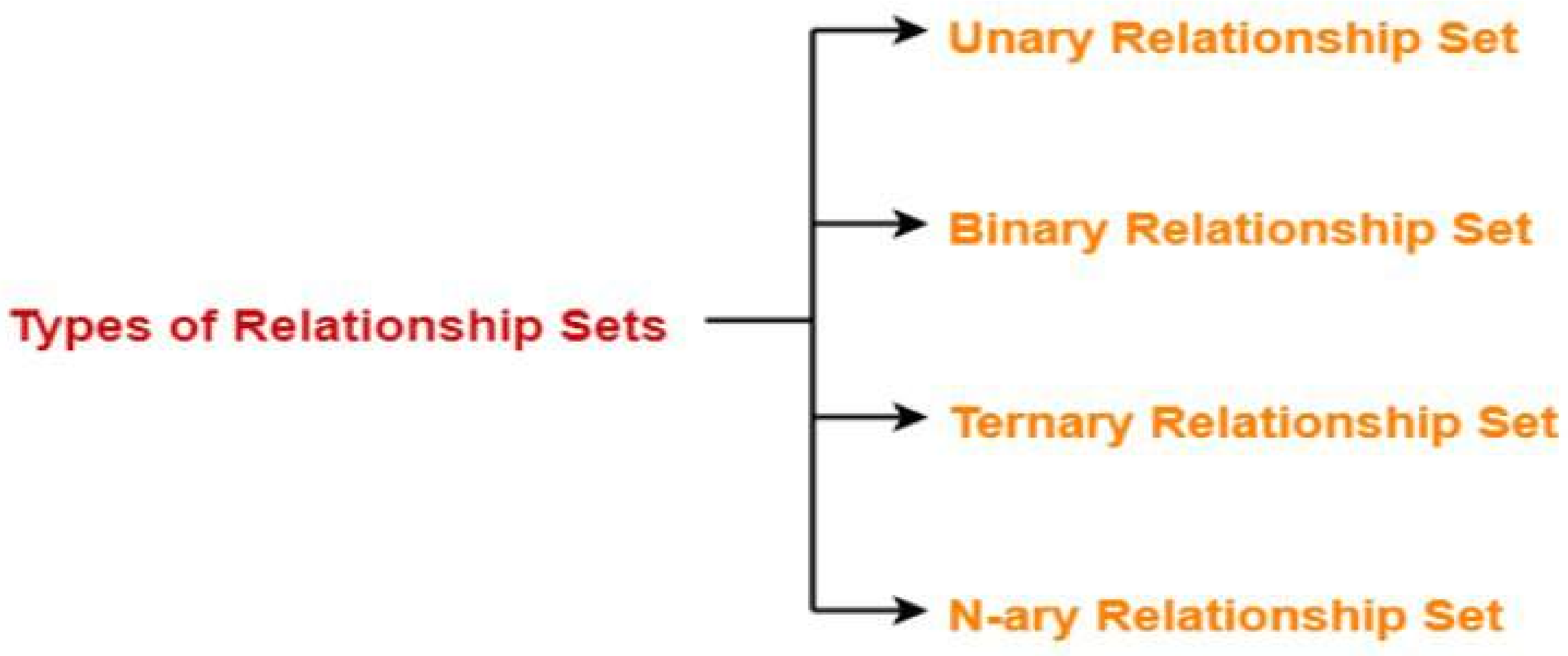
Degree of a Relationship Set-

The number of entity sets that participate in a relationship set is termed as the degree of that relationship set. Thus,

Degree of a relationship set = Number of entity sets participating in a relationship set

Types of Relationship Sets-

On the basis of degree of a relationship set, a relationship set can be classified into the following types-



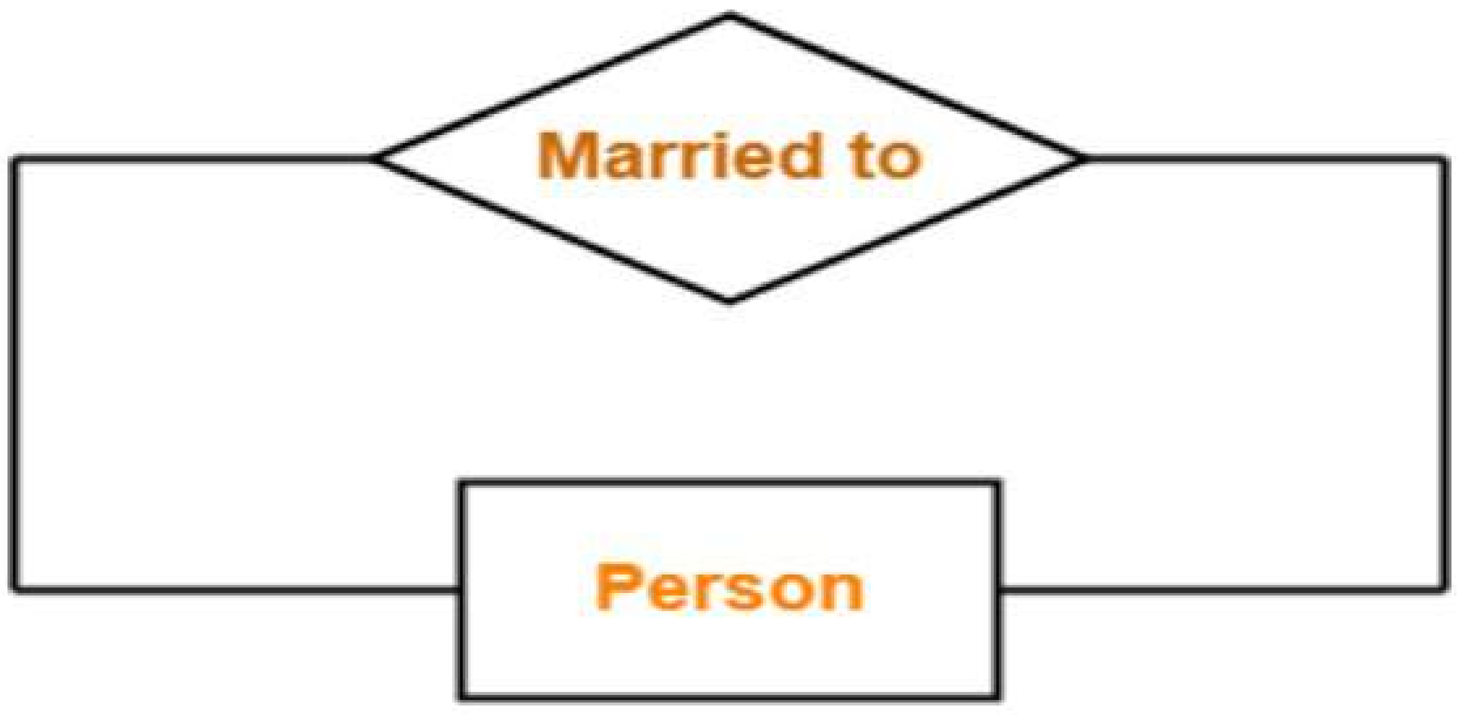
1. Unary relationship set
2. Binary relationship set
3. Ternary relationship set
4. N-ary relationship set

1. Unary Relationship Set-

Unary relationship set is a relationship set where only one entity set participates in a relationship set.

Example-

One person is married to only one person

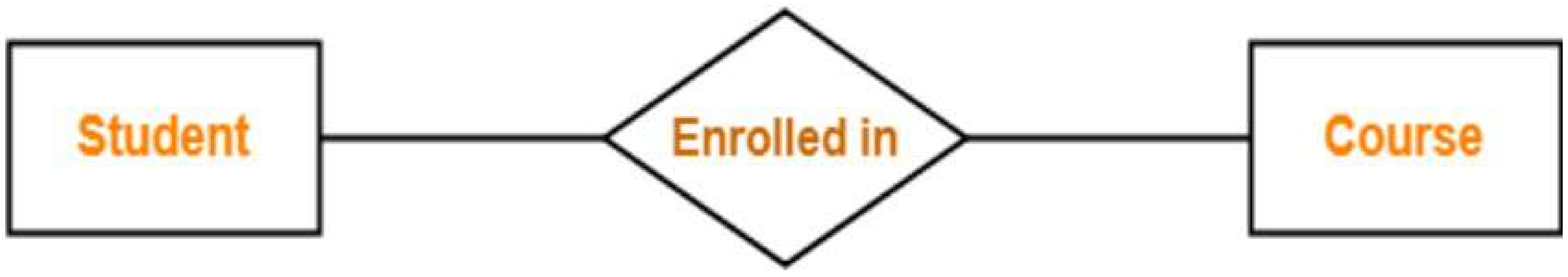


Unary Relationship Set

2. Binary Relationship Set-

Binary relationship set is a relationship set where two entity sets participate in a relationship set.

Example-

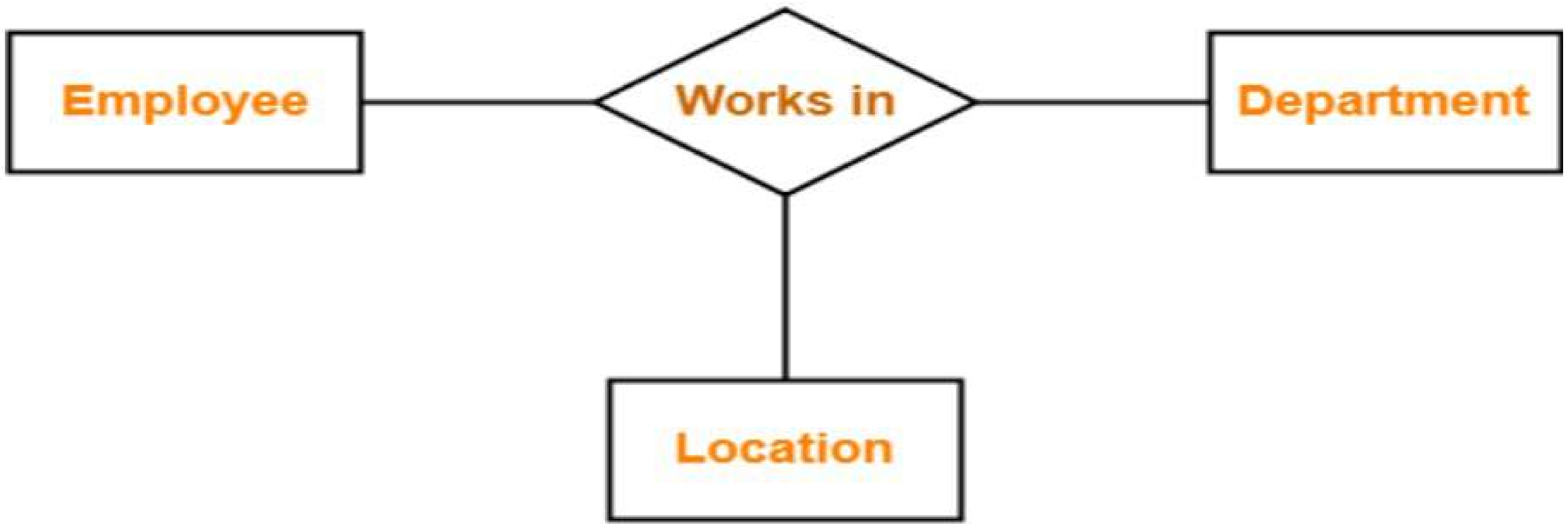


Binary Relationship Set

3. Ternary Relationship Set-

Ternary relationship set is a relationship set where three entity sets participate in a relationship set.

Example-



Ternary Relationship Set

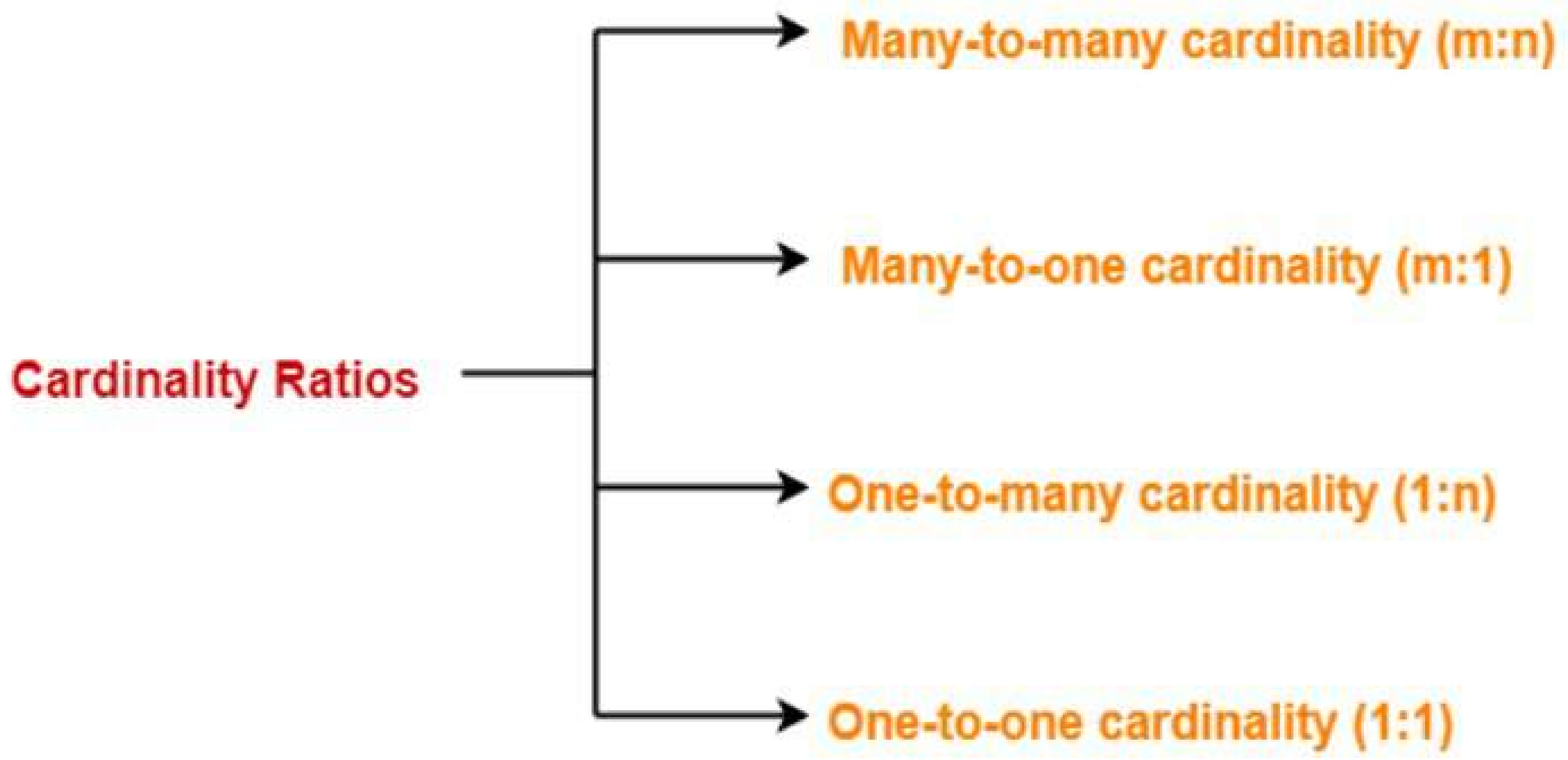
4. N-ary Relationship Set-

N-ary relationship set is a relationship set where 'n' entity sets participate in a relationship set.

Cardinality Constraint-

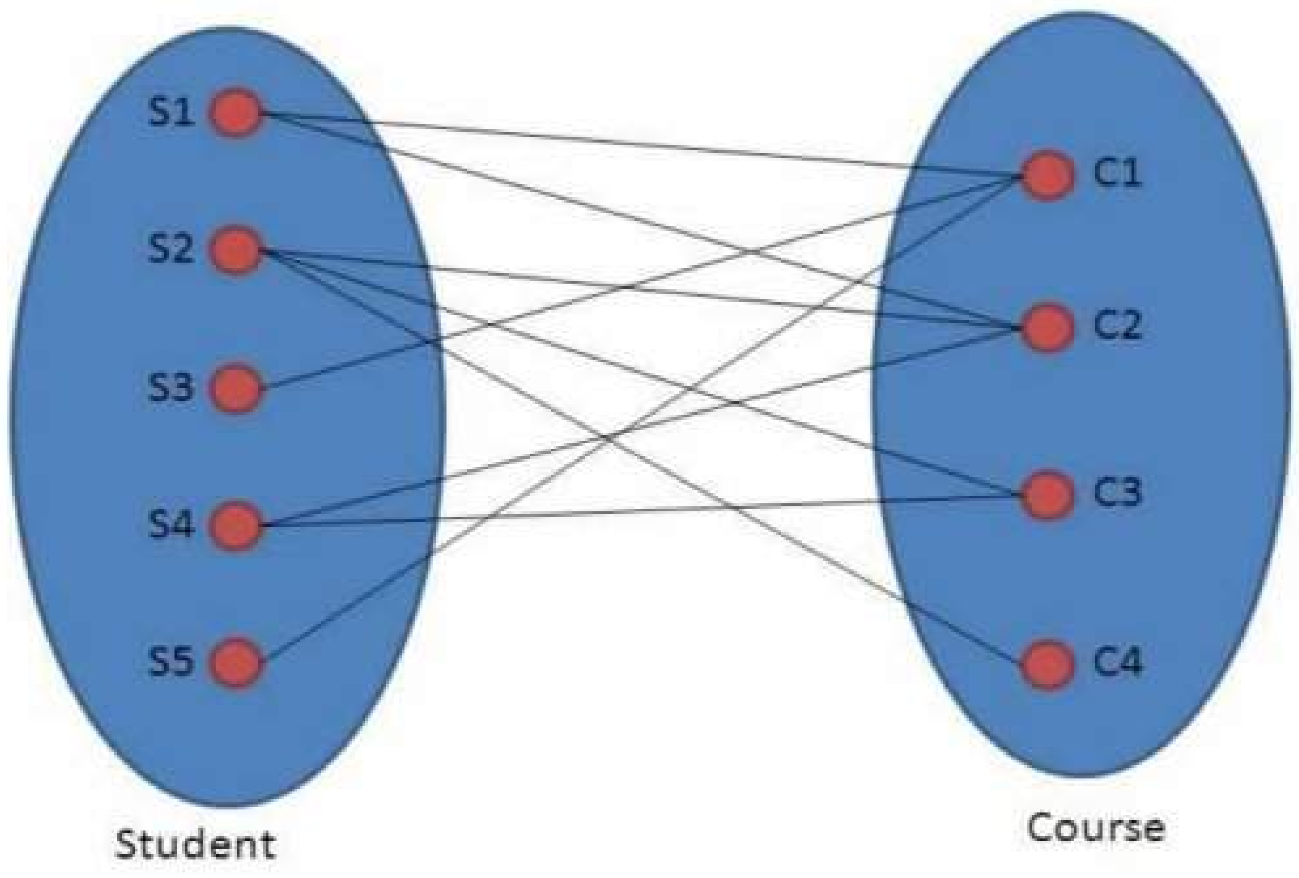
Types of Cardinality Ratios-

There are 4 types of cardinality ratios-


Cardinality Ratios
Many-to-many cardinality (m:n)
Many-to-one cardinality (m:1)
One-to-many cardinality (1:n)
One-to-one cardinality (1:1)

1. Many-to-Many cardinality (m:n)
2. Many-to-One cardinality (m:1)
3. One-to-Many cardinality (1:n)
4. One-to-One cardinality (1:1)

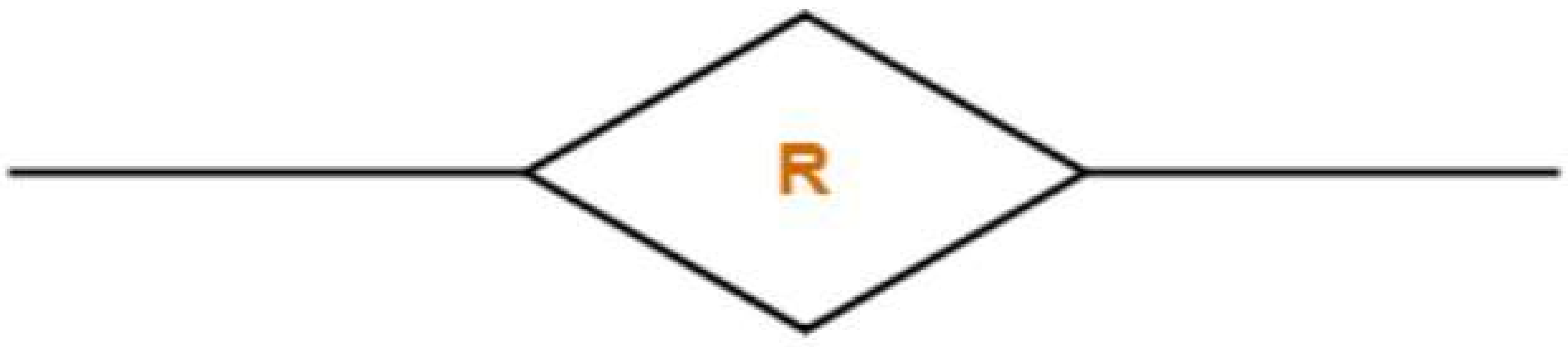
1. <u>Many-to-Many Cardinality-</u>



By this cardinality constraint,

- An entity in set A can be associated with any number (zero or more) of entities in set B.
- An entity in set B can be associated with any number (zero or more) of entities in set A.

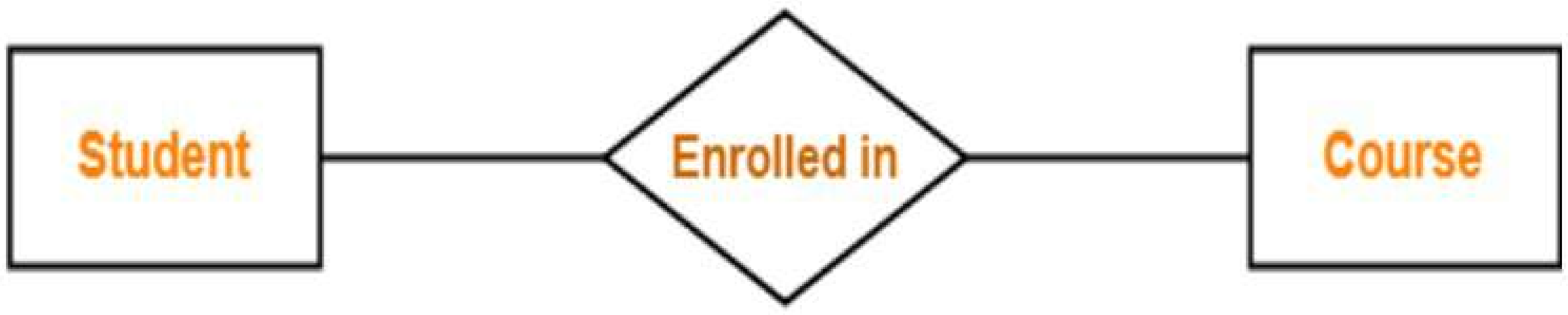
Symbol Used-



Cardinality Ratio = m : n

Example-

Consider the following ER diagram-

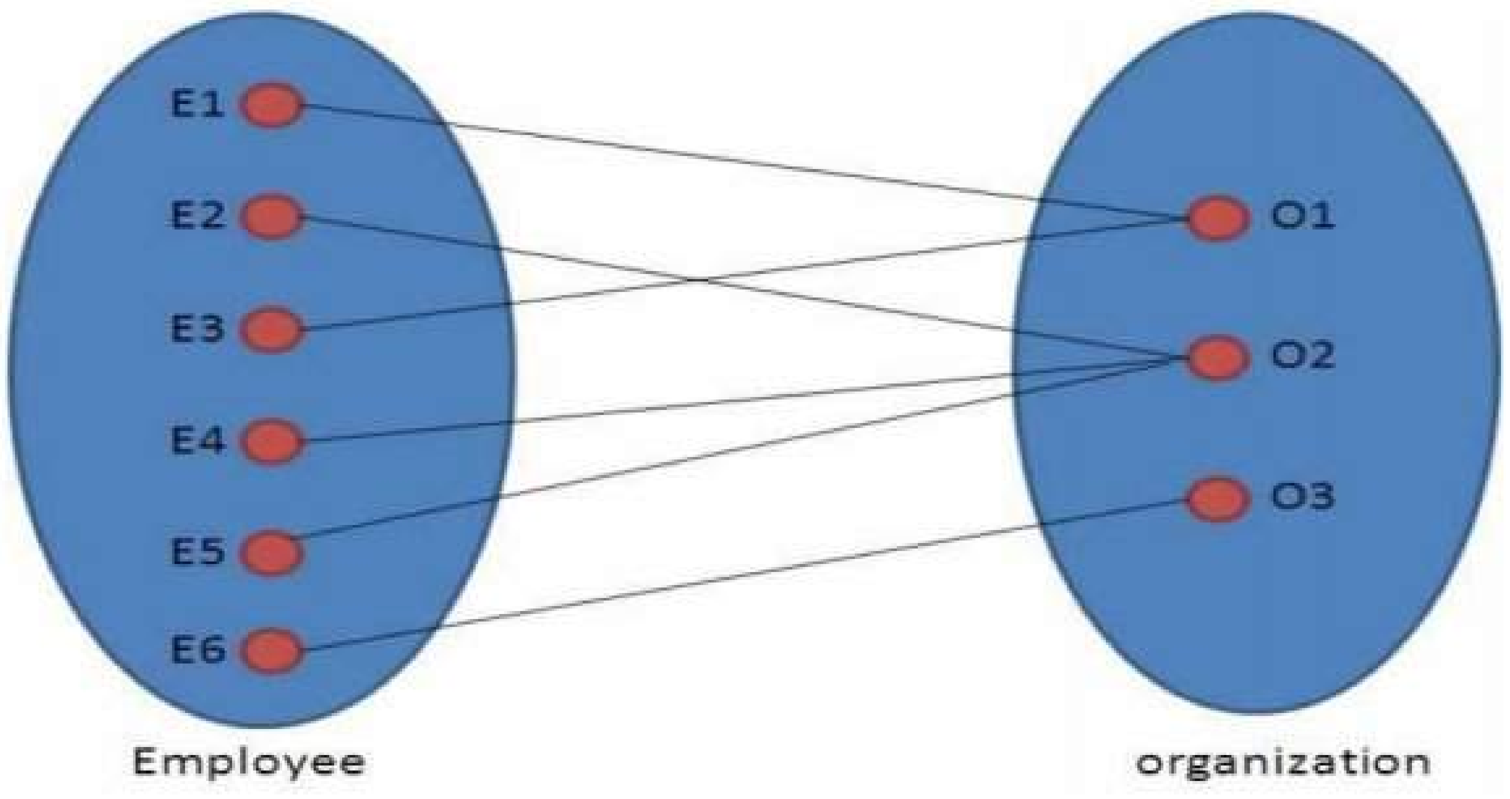


Many to Many Relationship

Here,

- One student can enroll in any number (zero or more) of courses.
- One course can be enrolled by any number (zero or more) of students.

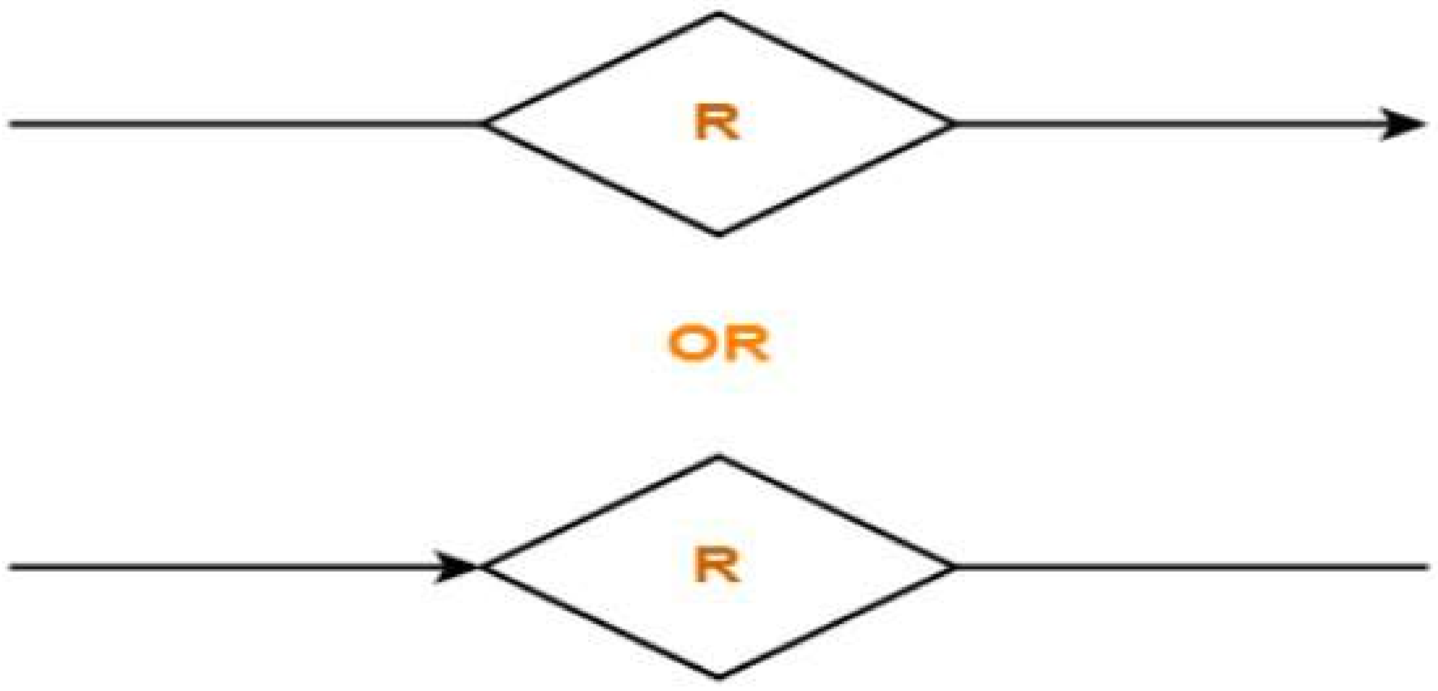
2. Many-to-One Cardinality-



By this cardinality constraint,

- An entity in set A can be associated with at most one entity in set B.
- An entity in set B can be associated with any number (zero or more) of entities in set A.
-

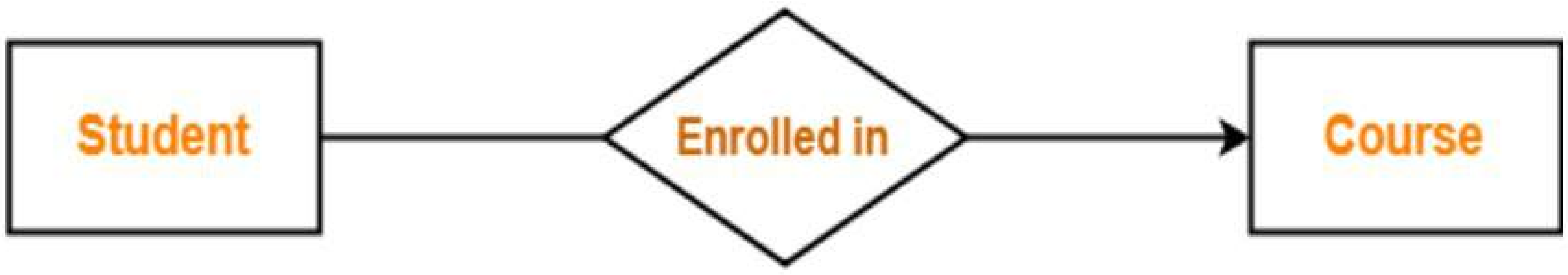
Symbol Used-



Cardinality Ratio = m : 1

Example-

Consider the following ER diagram-

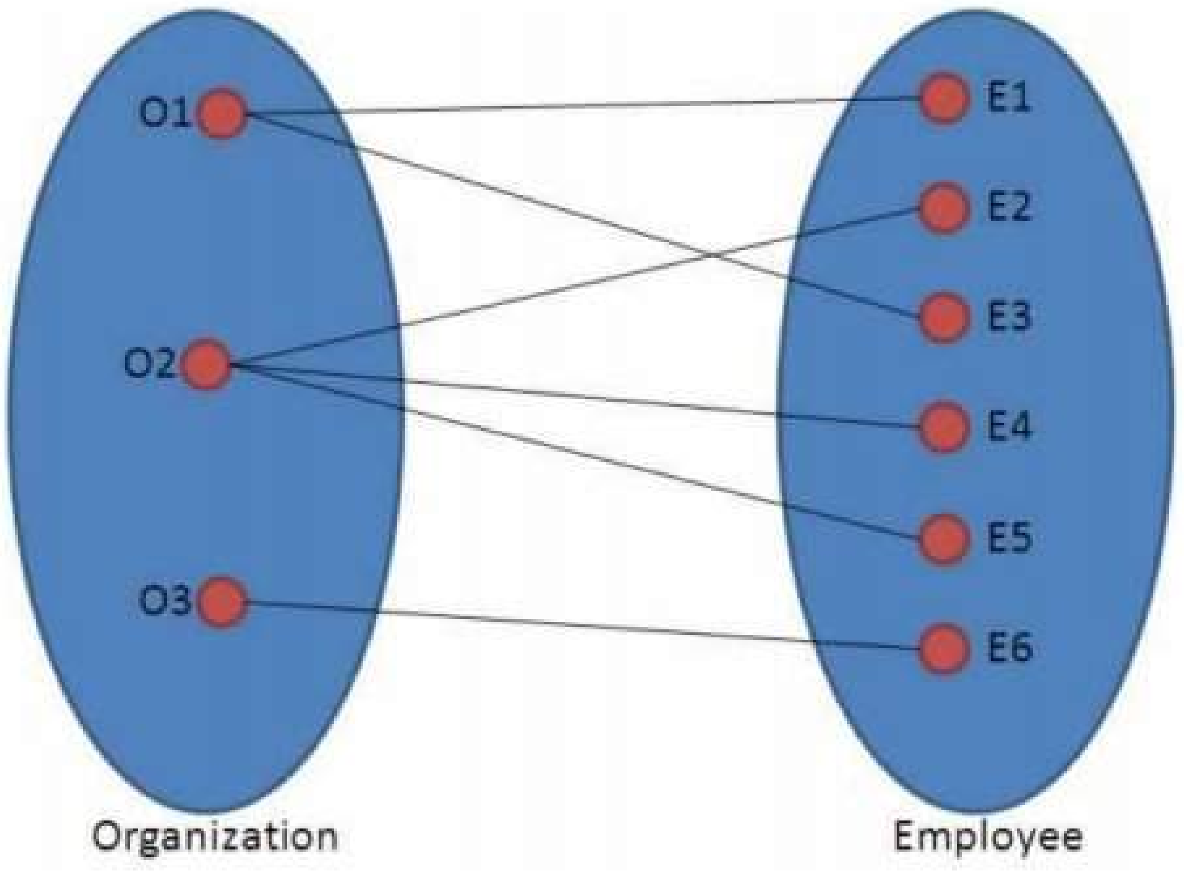


Many to One Relationship

Here,

- One student can enroll in at most one course.
- One course can be enrolled by any number (zero or more) of students.

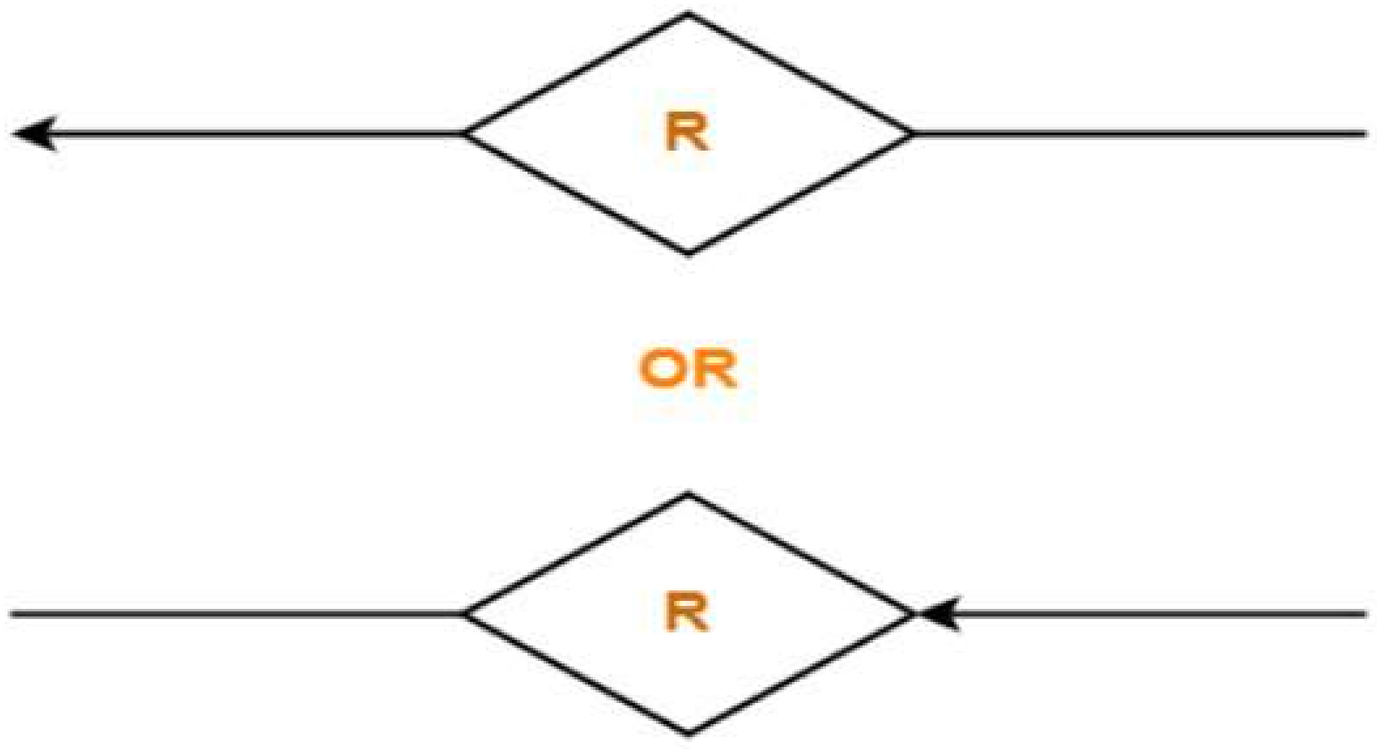
3. <u>One-to-Many Cardinality-</u>



By this cardinality constraint,

- An entity in set A can be associated with any number (zero or more) of entities in set B.
- An entity in set B can be associated with at most one entity in set A.

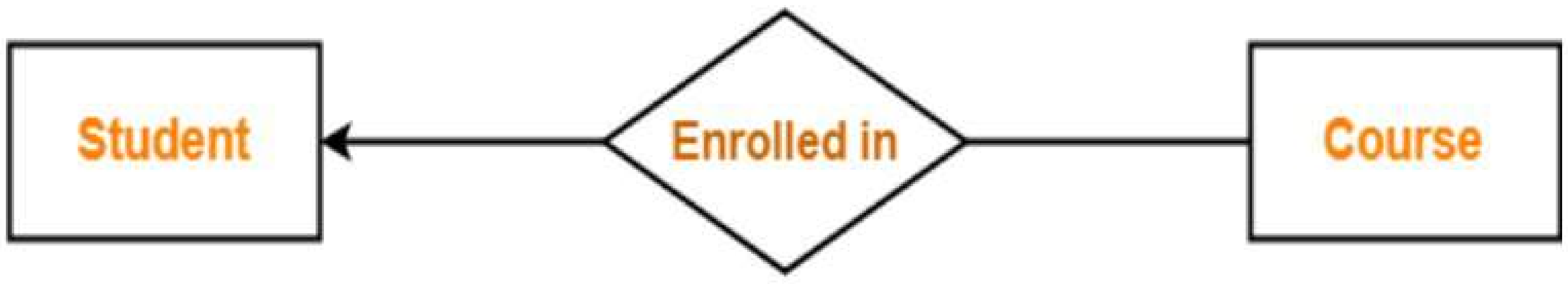
Symbol Used-



Cardinality Ratio = 1 : n

Example-

Consider the following ER diagram



One to Many Relationship

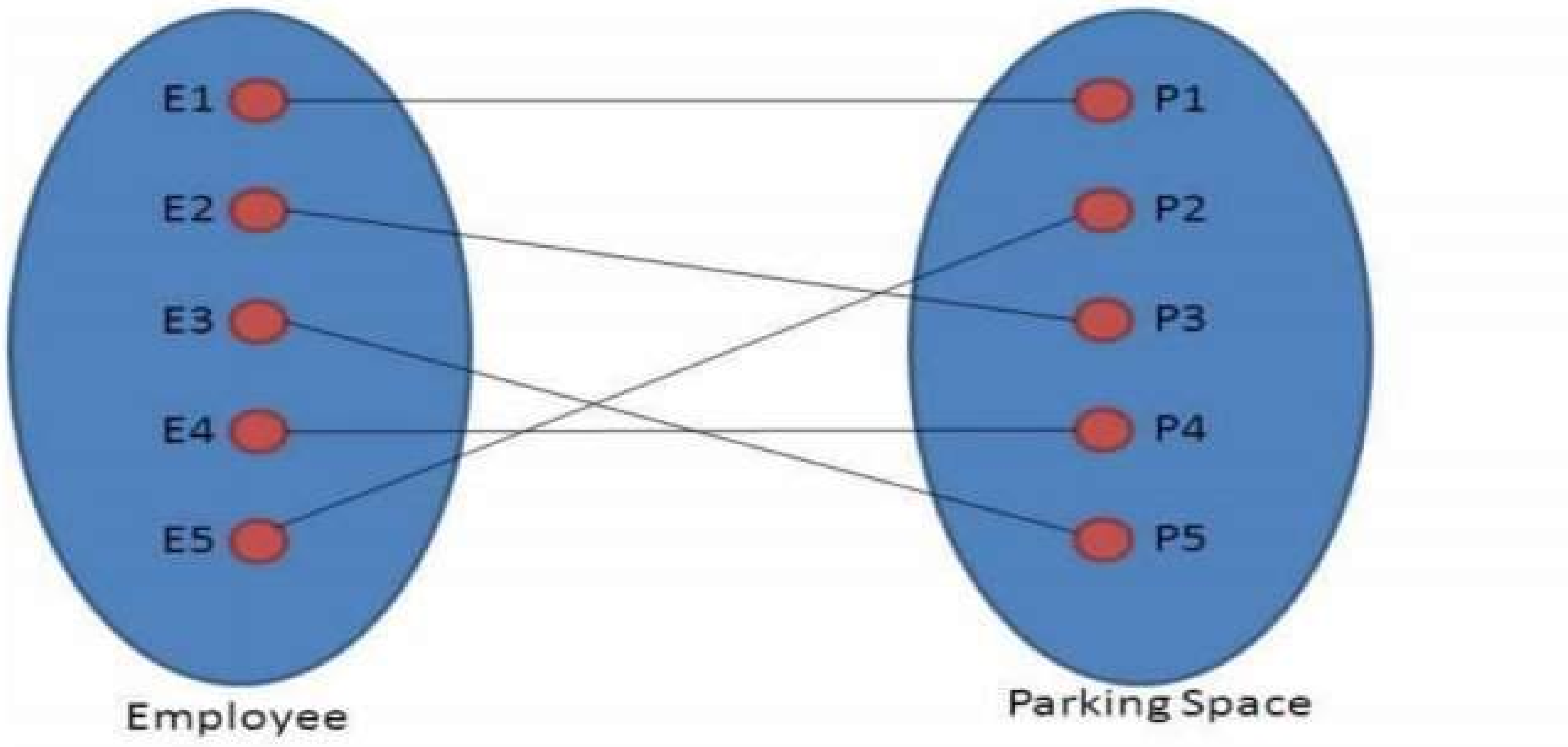
Here,

- One student can enroll in any number (zero or more) of courses.
- One course can be enrolled by at most one student.
-

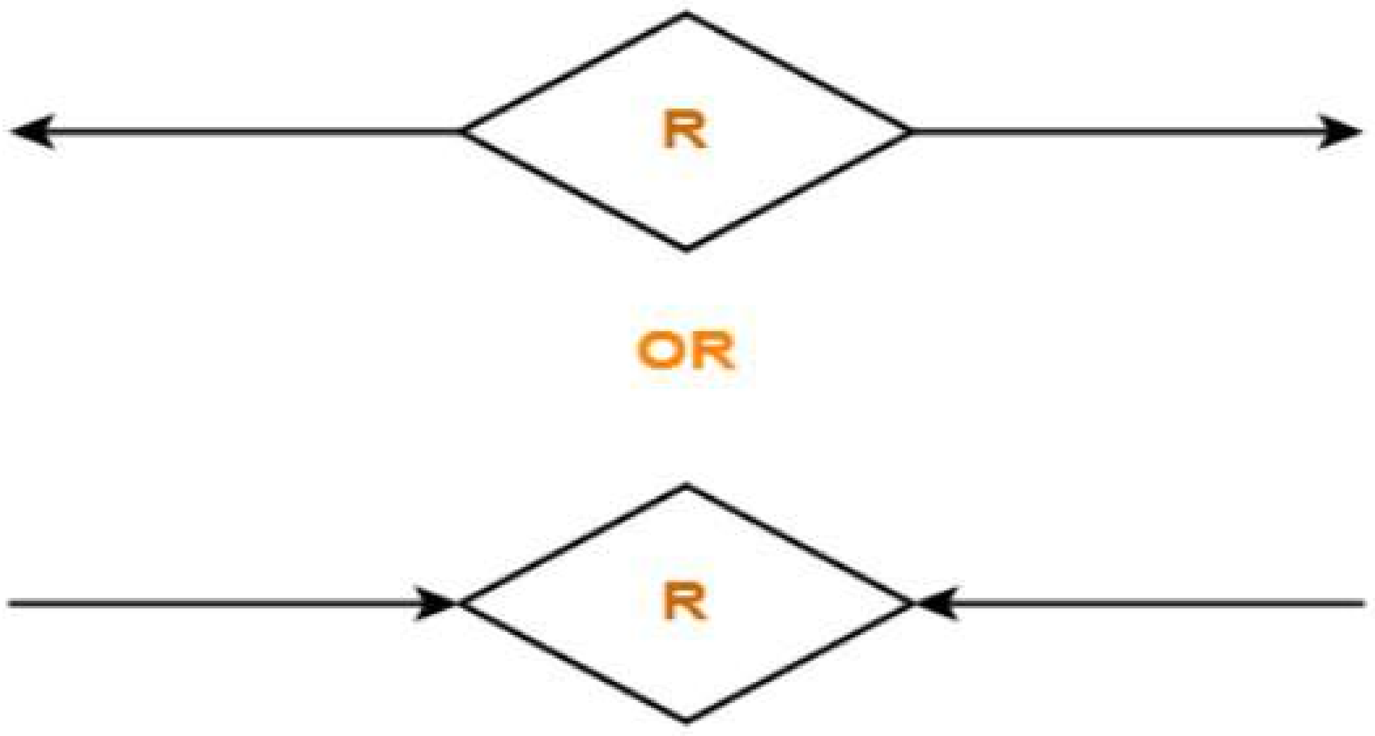
4. One-to-One Cardinality-

By this cardinality constraint,

- An entity in set A can be associated with at most one entity in set B.
- An entity in set B can be associated with at most one entity in set A.



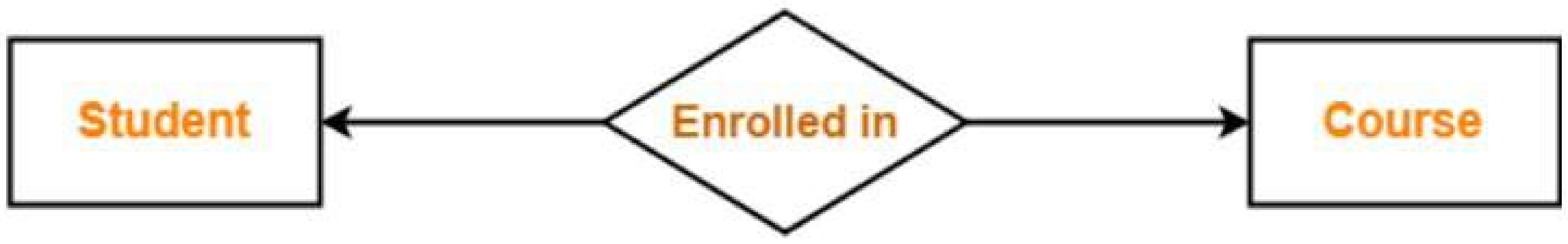
Symbol Used-



Cardinality Ratio = 1 : 1

Example-

Consider the following ER diagram-



One to One Relationship

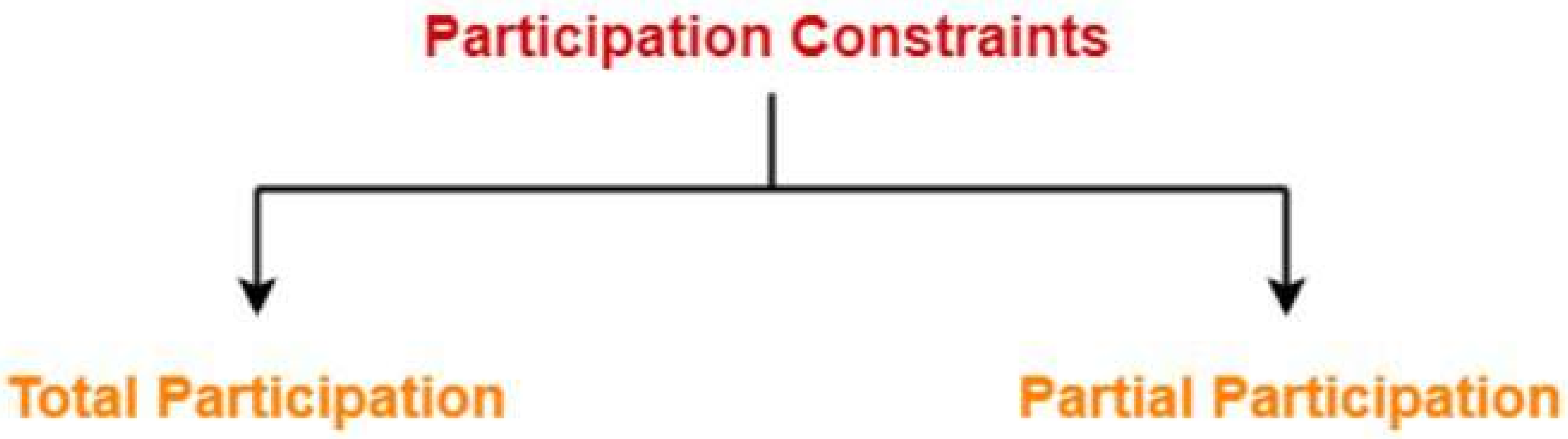
Here,

- One student can enroll in at most one course.
- One course can be enrolled by at most one student.

Participation Constraints-

Types of Participation Constraints-

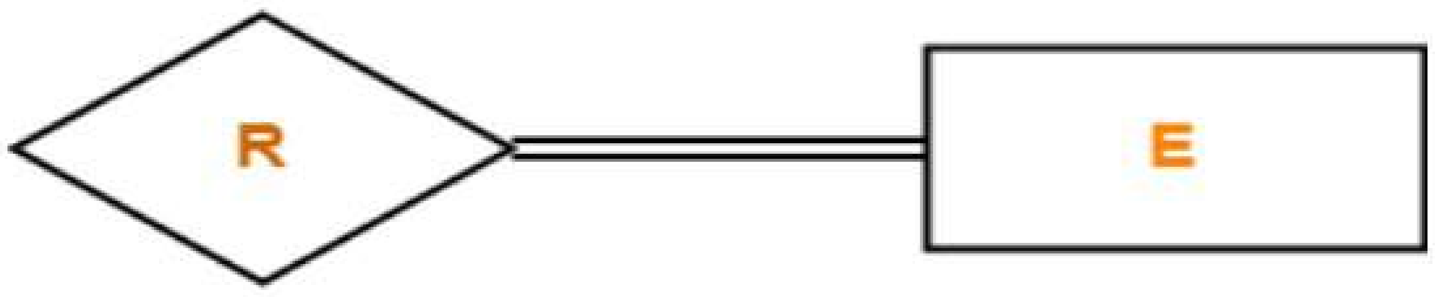
There are two types of participation constraints-



1. Total participation
2. Partial participation

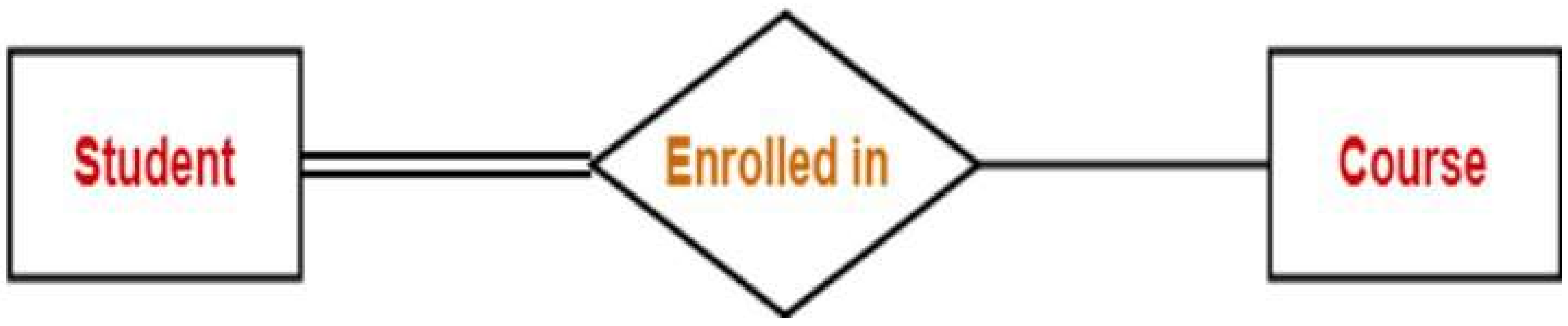
1. Total Participation-

- It specifies that each entity in the entity set must compulsorily participate in at least one relationship instance in that relationship set.
- That is why, it is also called as **mandatory participation.**
- Total participation is represented using a double line between the entity set and relationship set.



Total Participation

Example-

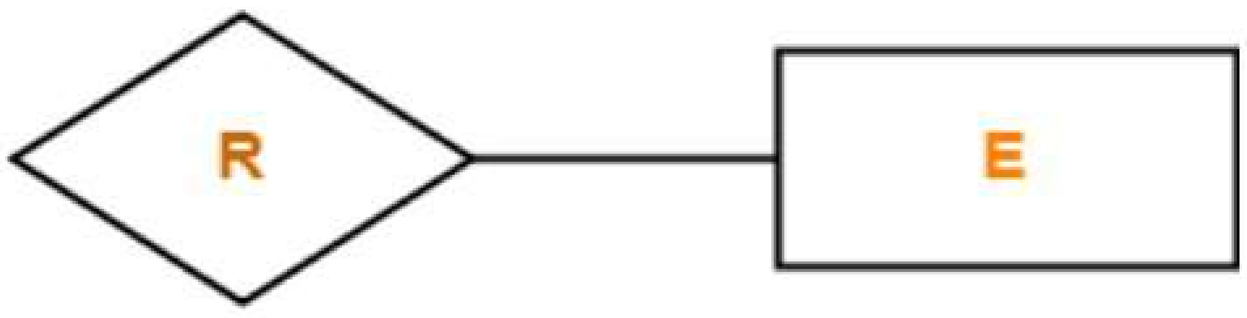


Here,

- Double line between the entity set "Student" and relationship set "Enrolled in" signifies total participation.
- It specifies that each student must be enrolled in at least one course.

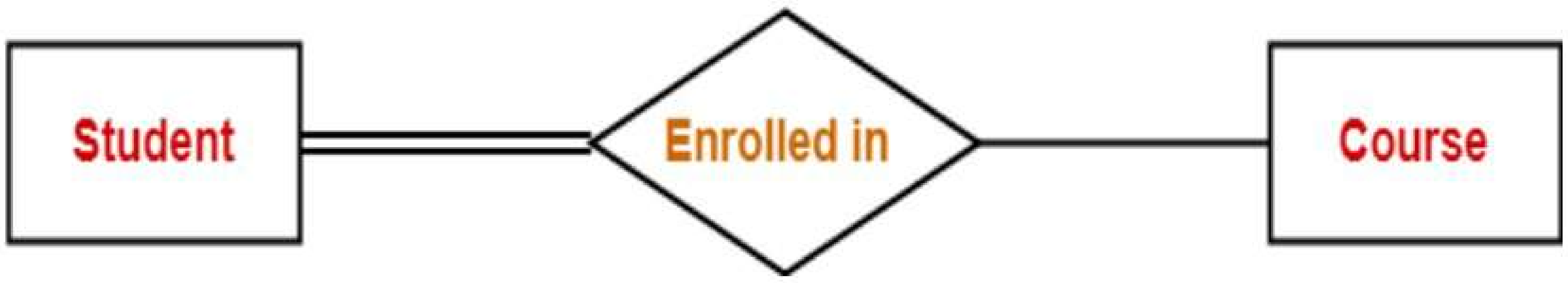
2. Partial Participation-

- It specifies that each entity in the entity set may or may not participate in the relationship instance in that relationship set.
- That is why, it is also called as **optional participation.**
- Partial participation is represented using a single line between the entity set and relationship set.



Partial Participation

Example-



Here,

- Single line between the entity set "Course" and relationship set "Enrolled in" signifies partial participation.
- It specifies that there might exist some courses for which no enrollments are made.

Relationship between Cardinality and Participation Constraints-

Minimum cardinality tells whether the participation is partial or total.

- If minimum cardinality = 0, then it signifies partial participation.
- If minimum cardinality = 1, then it signifies total participation.

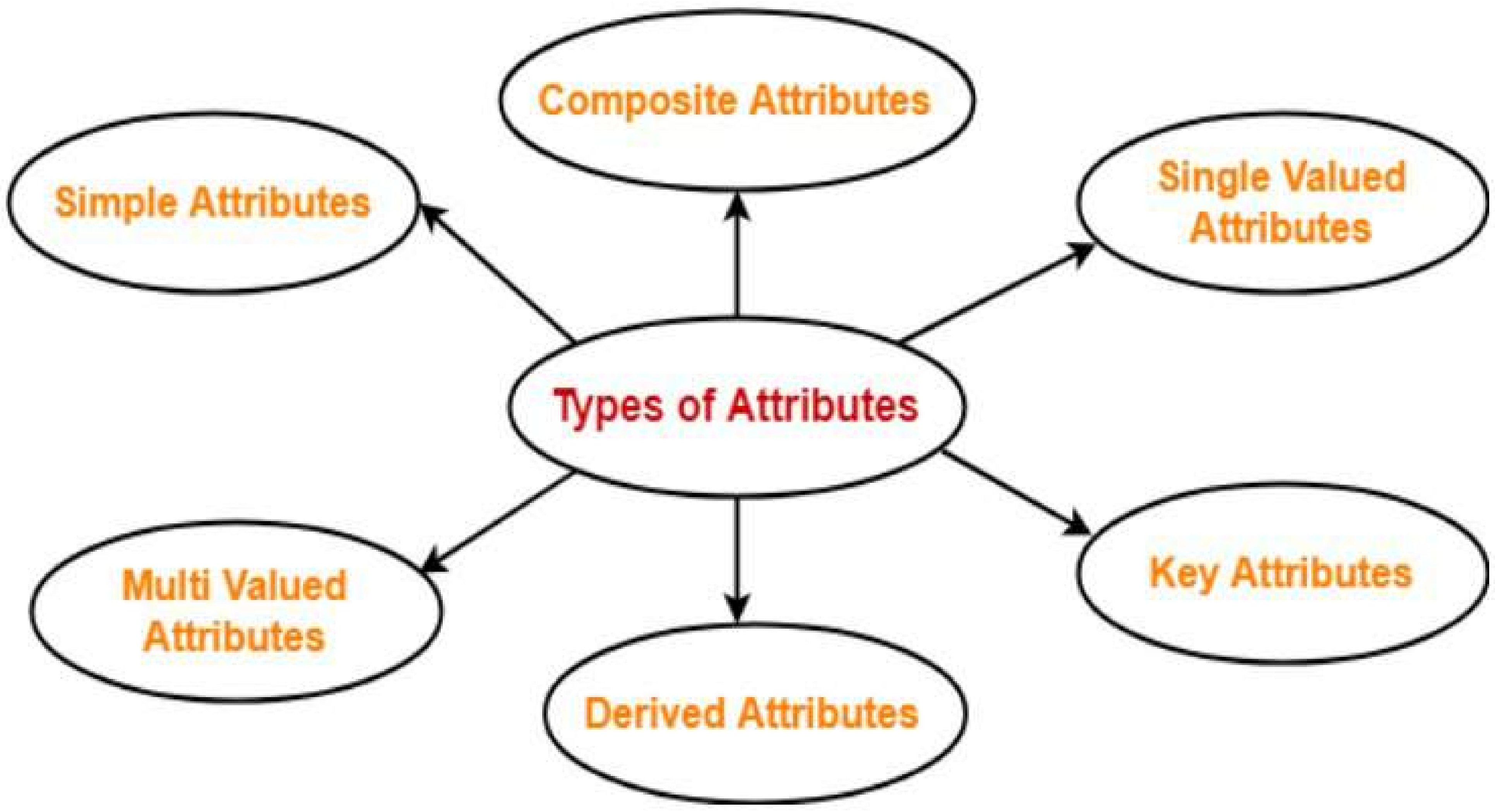
Maximum cardinality tells the maximum number of entities that participates in a relationship set.

Attributes in ER Diagram-

- Attributes are the descriptive properties which are owned by each entity of an **Entity Set**.
- There exist a specific domain or set of values for each attribute from where the attribute can take its values.

Types of Attributes-

In ER diagram, attributes associated with an entity set may be of the following types-

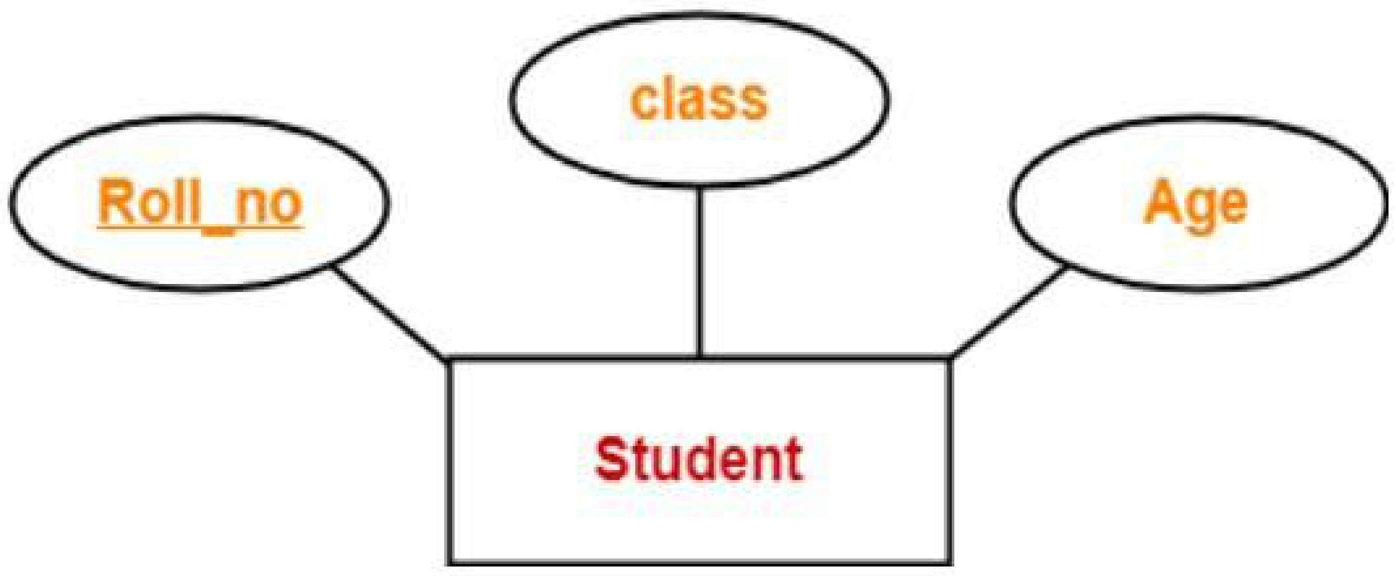


1. Simple attributes
2. Composite attributes
3. Single valued attributes
4. Multi valued attributes
5. Derived attributes
6. Key attributes

1. Simple Attributes-

Simple attributes are those attributes which can not be divided further.

Example-

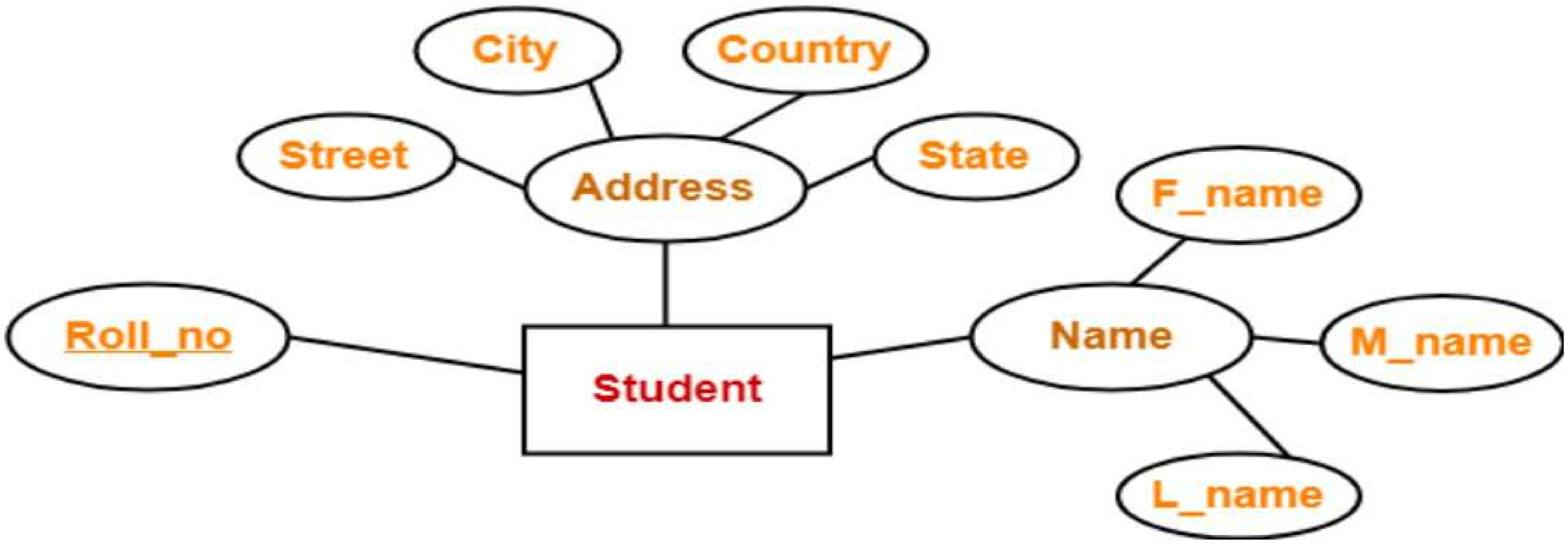


Here, all the attributes are simple attributes as they can not be divided further.

2. Composite Attributes-

Composite attributes are those attributes which are composed of many other simple attributes.

Example-

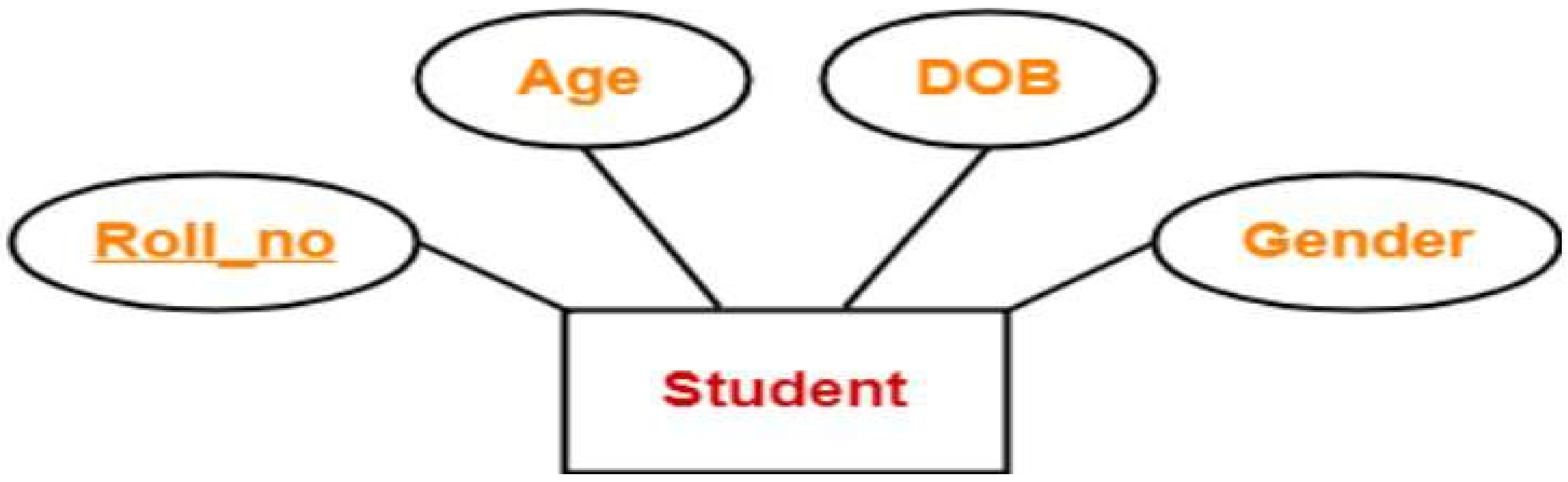


Here, the attributes “Name” and “Address” are composite attributes as they are composed of many other simple attributes.

3. Single Valued Attributes-

Single valued attributes are those attributes which can take only one value for a given entity from an entity set.

Example-

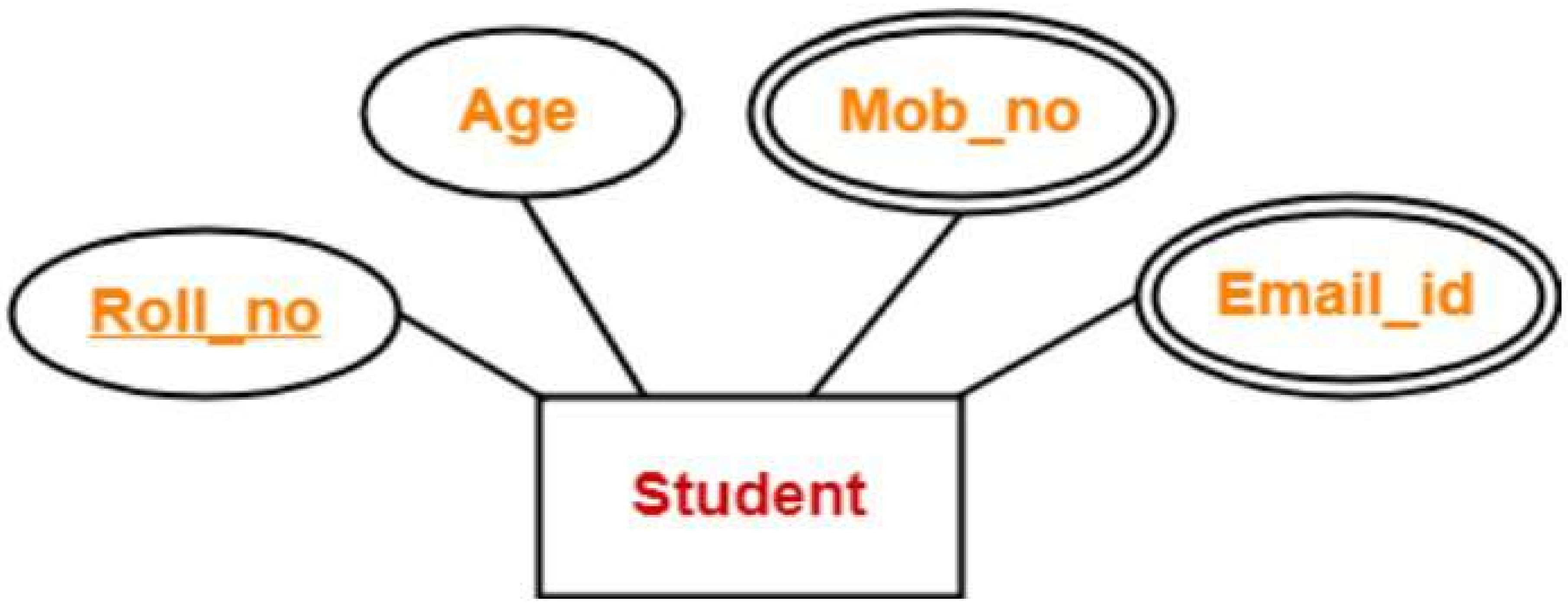


Here, all the attributes are single valued attributes as they can take only one specific value for each entity.

4. Multi Valued Attributes-

Multi valued attributes are those attributes which can take more than one value for a given entity from an entity set.

Example-

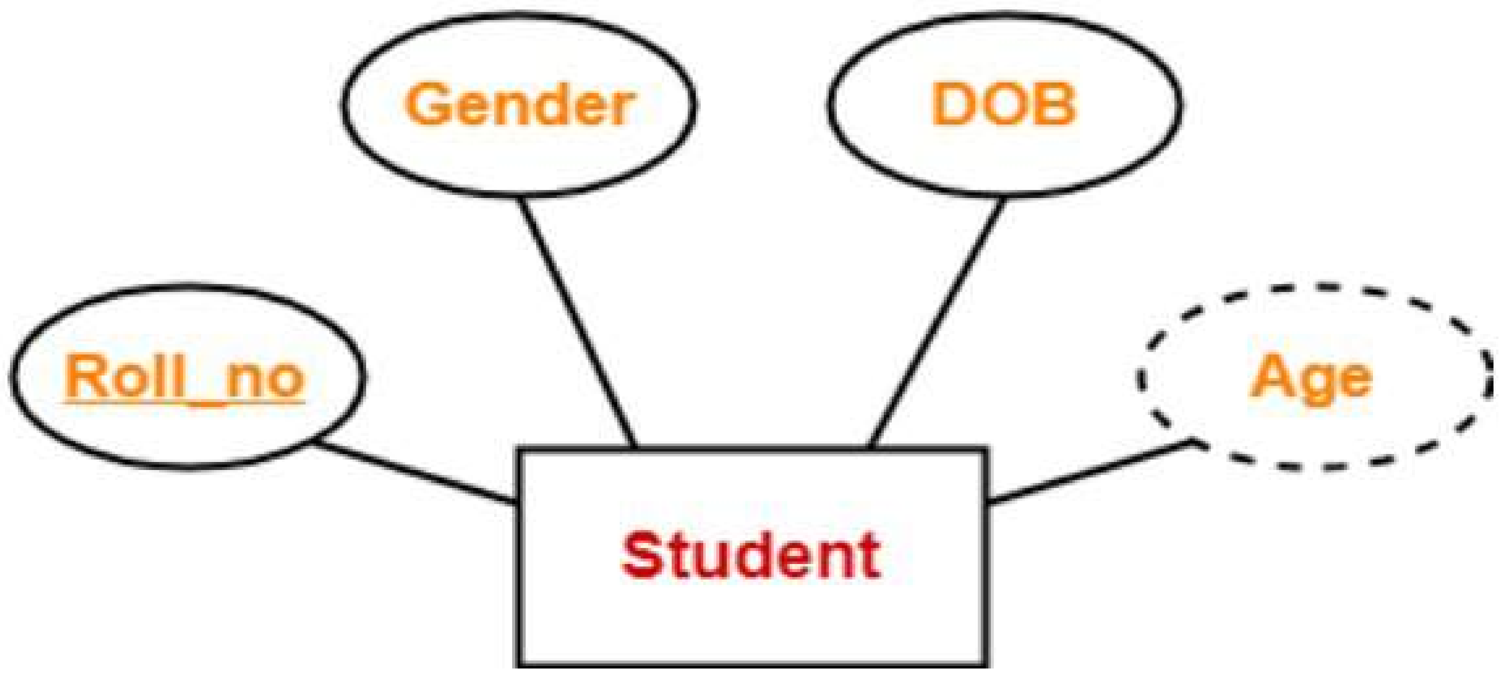


Here, the attributes "Mob_no" and "Email_id" are multi valued attributes as they can take more than one values for a given entity.

5. Derived Attributes-

Derived attributes are those attributes which can be derived from other attribute(s).

Example-

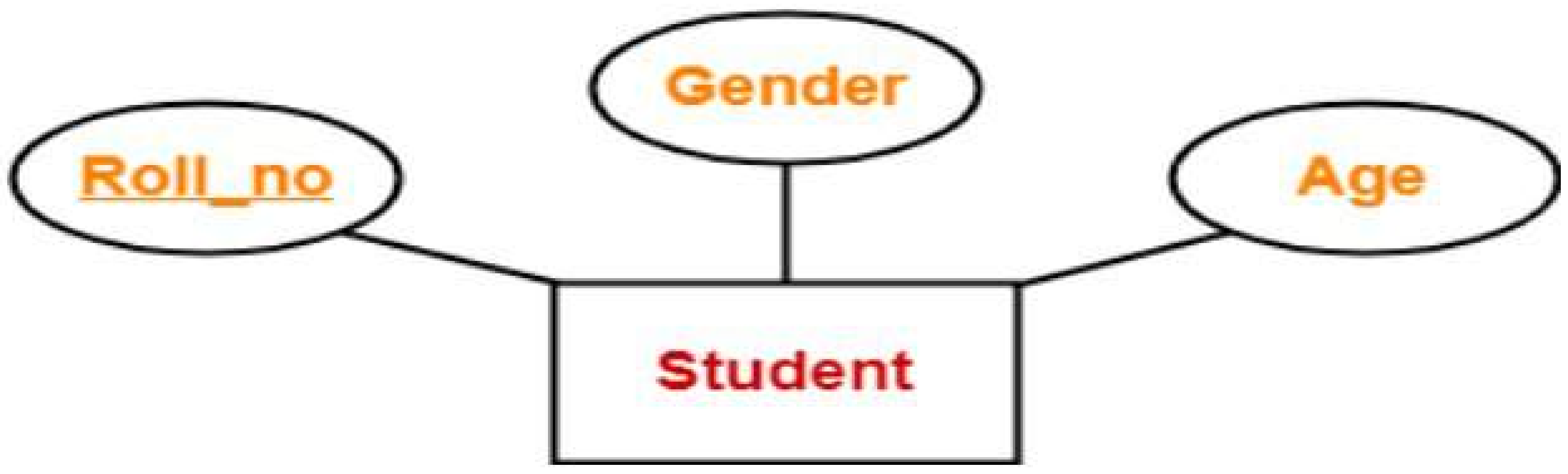


Here, the attribute "Age" is a derived attribute as it can be derived from the attribute "DOB".

6. Key Attributes-

Key attributes are those attributes which can identify an entity uniquely in an entity set.

Example-

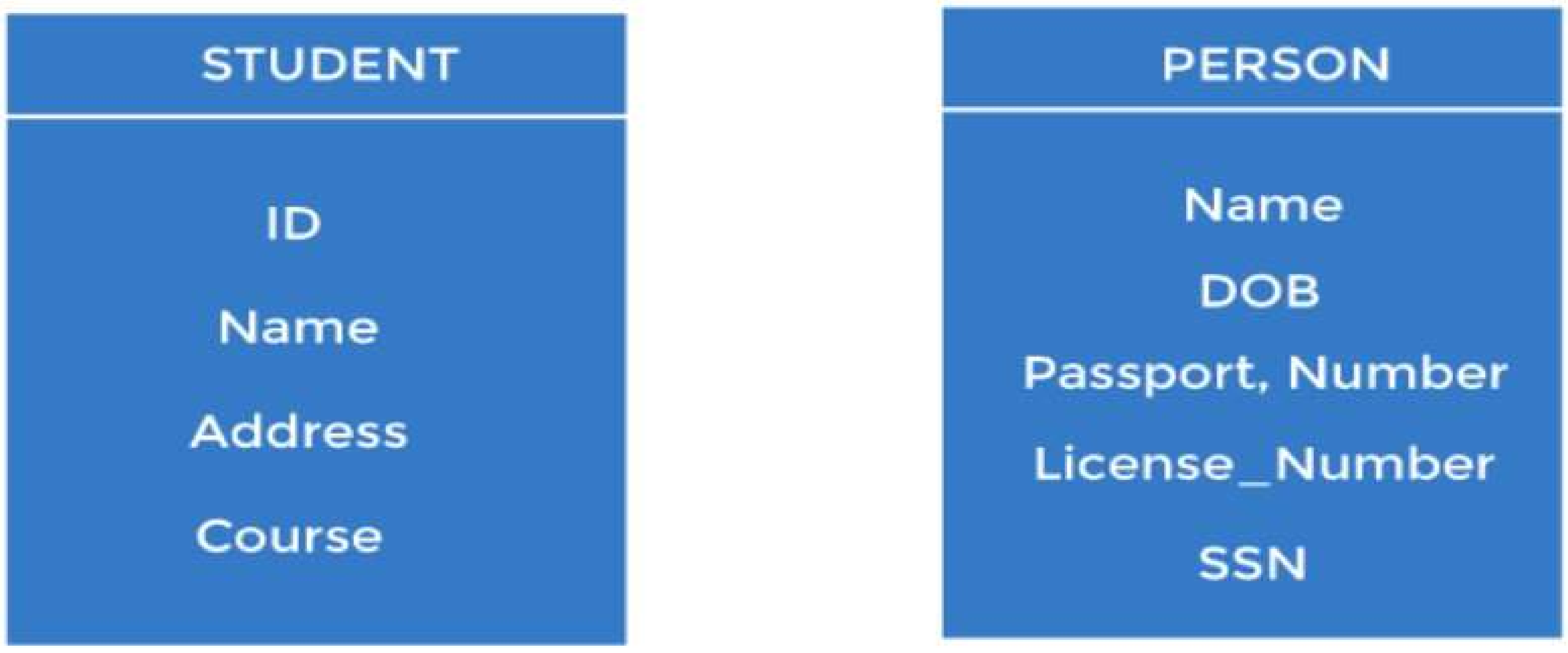


Here, the attribute "Roll_no" is a key attribute as it can identify any student uniquely.

Keys in DBMS

- Keys play an important role in the relational database.
- It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

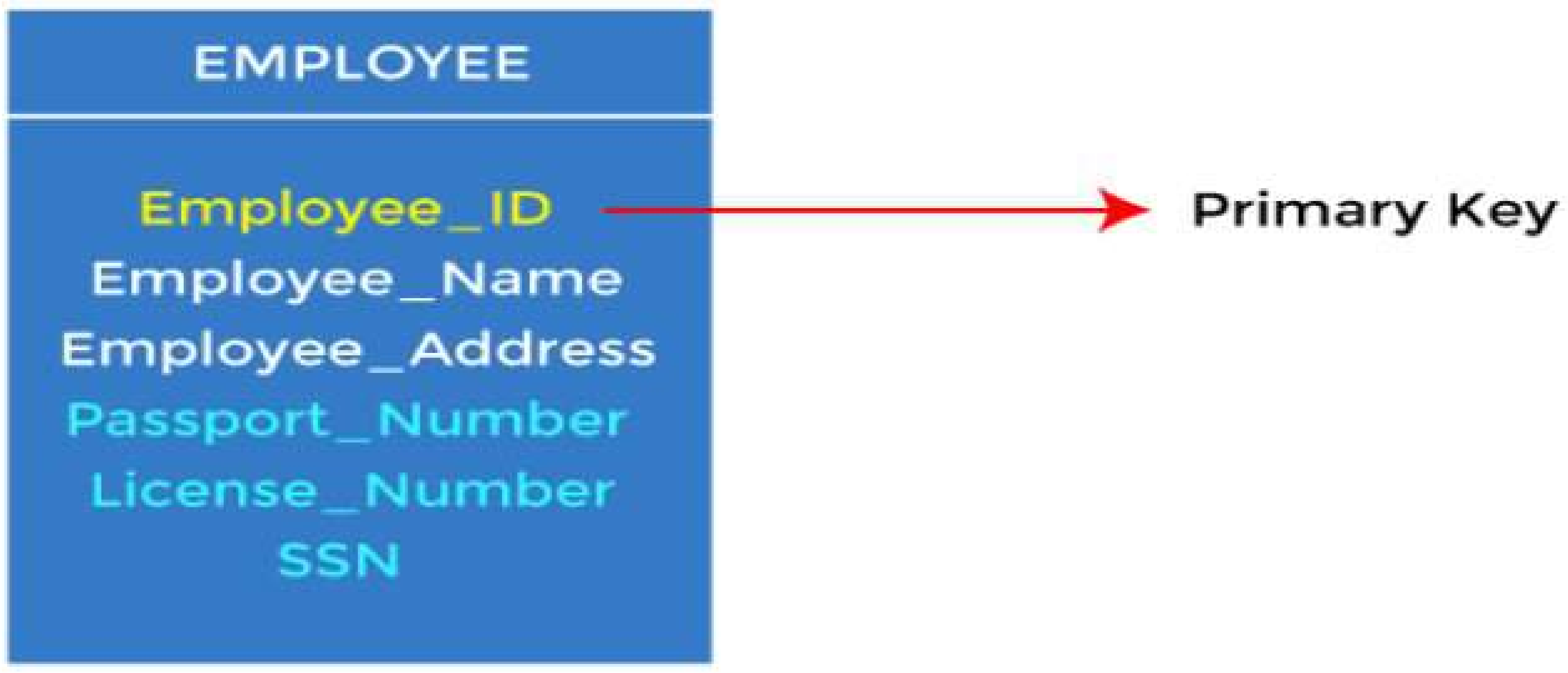
For example, ID is used as a key in the Student table because it is unique for each student. In the PERSON table, passport_number, license_number, SSN are keys since they are unique for each person.



Types of keys:

1. Primary key

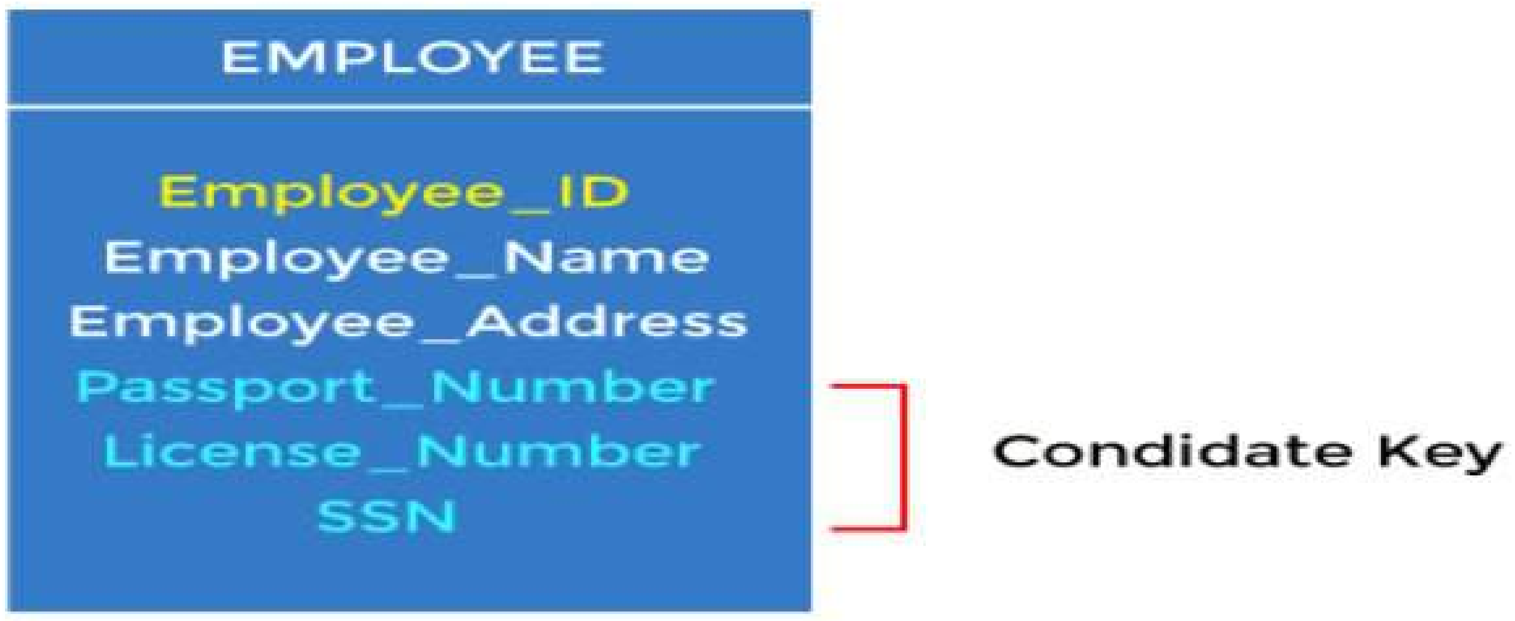
- It is the first key used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys, as we saw in the PERSON table. The key which is most suitable from those lists becomes a primary key.
- In the EMPLOYEE table, ID can be the primary key since it is unique for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



2. Candidate key

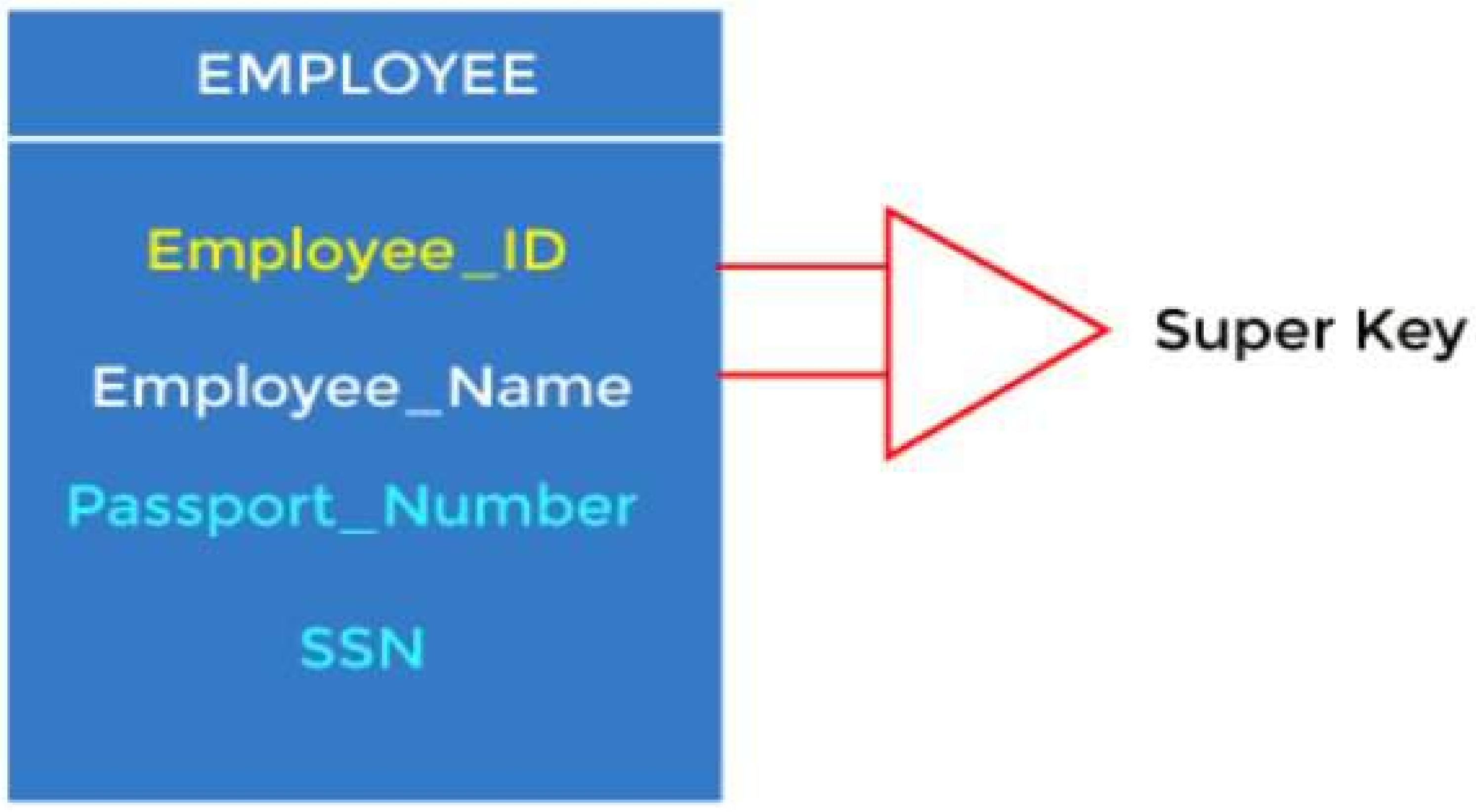
- A candidate key is an attribute or set of attributes that can uniquely identify a tuple.
- Except for the primary key, the remaining attributes are considered a candidate key. The candidate keys are as strong as the primary key.

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



3. Super Key

Super key is an attribute set that can uniquely identify a tuple. A super key is a superset of a candidate key.

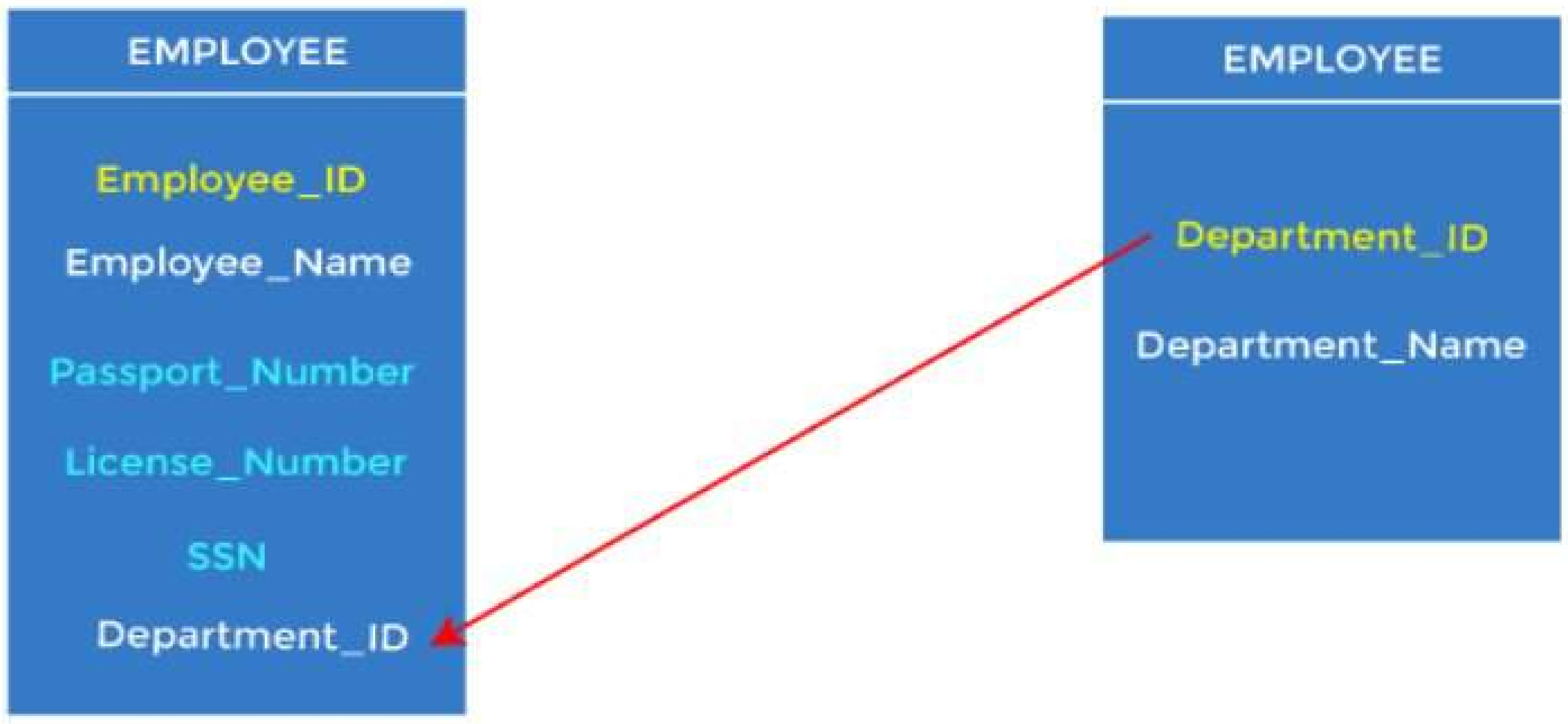


For example: In the above EMPLOYEE table, for(EMPLOEE_ID, EMPLOYEE_NAME), the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key.

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

4. Foreign key

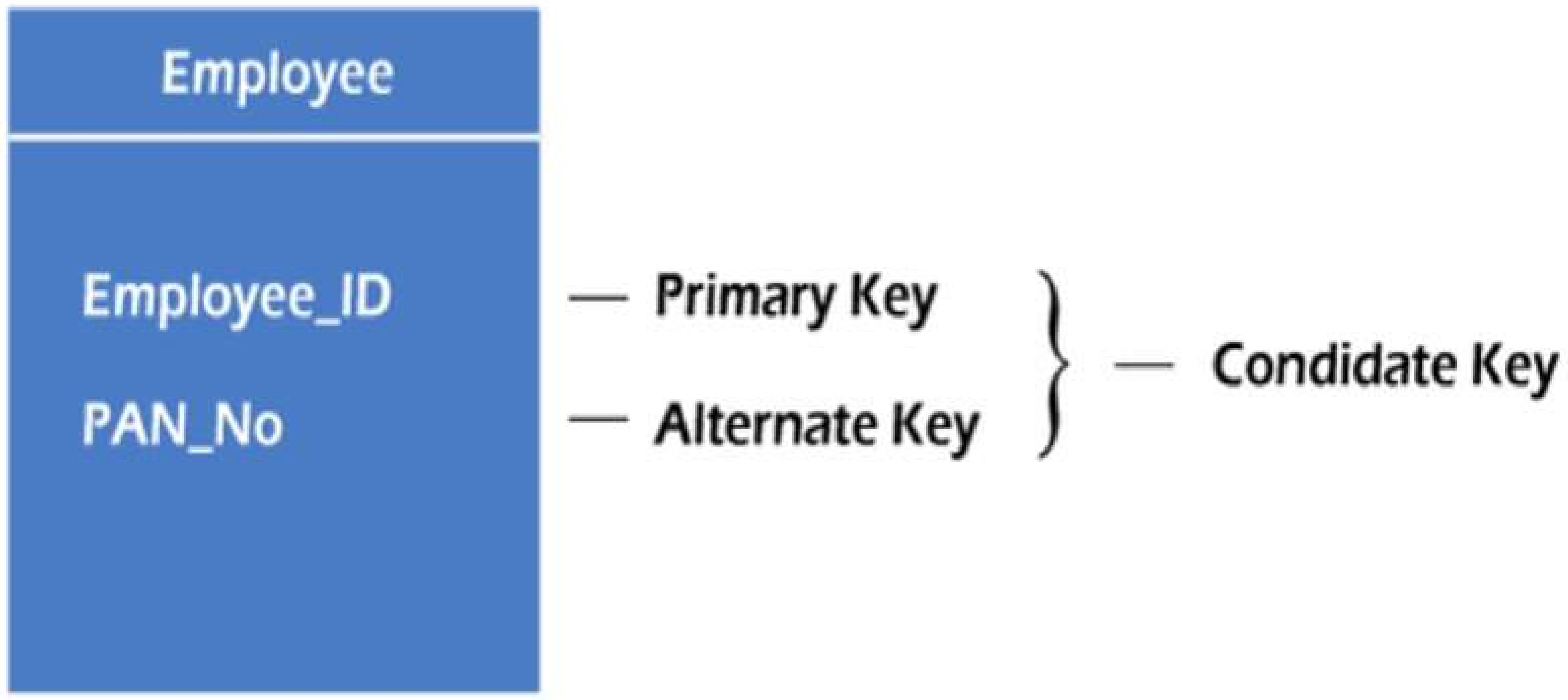
- Foreign keys are the column of the table used to point to the primary key of another table.
- Every employee works in a specific department in a company, and employee and department are two different entities. So we can't store the department's information in the employee table. That's why we link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



5. Alternate key

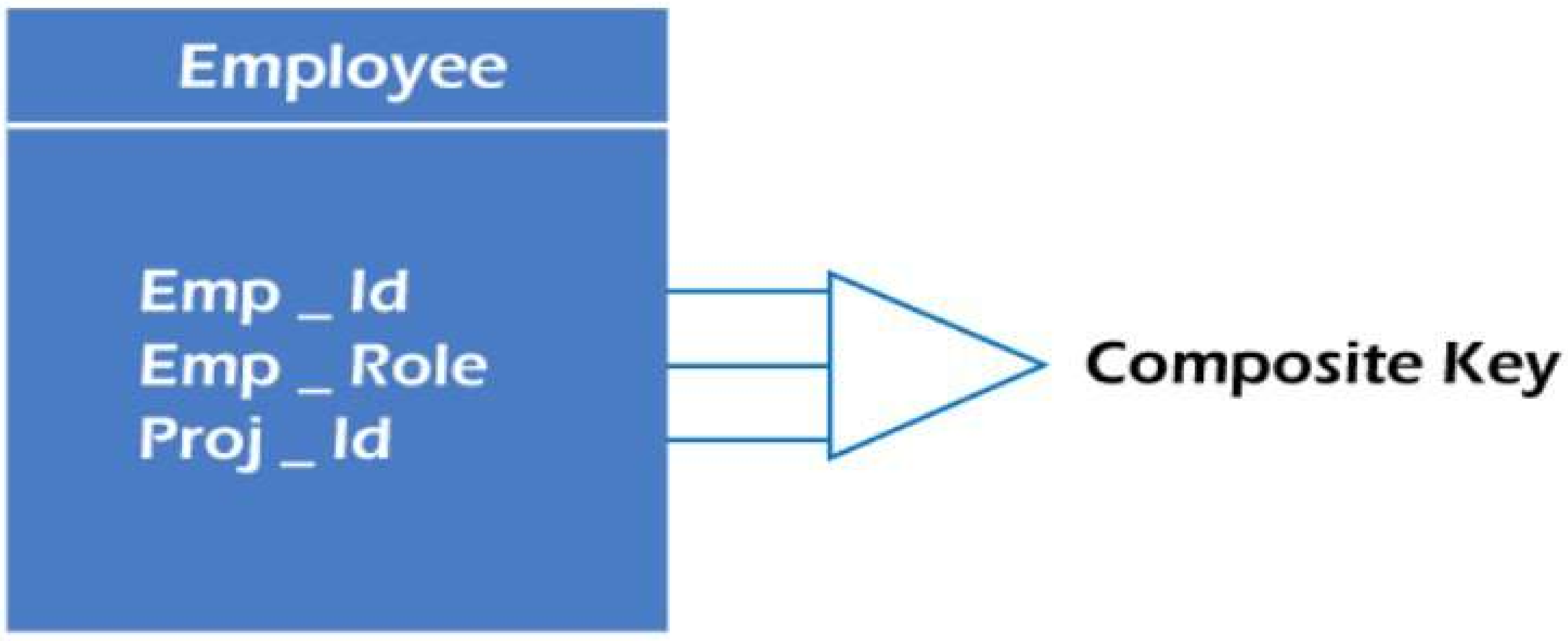
There may be one or more attributes or a combination of attributes that uniquely identify each tuple in a relation. These attributes or combinations of the attributes are called the candidate keys. One key is chosen as the primary key from these candidate keys, and the remaining candidate key, if it exists, is termed the alternate key. **In other words,** the total number of the alternate keys is the total number of candidate keys minus the primary key. The alternate key may or may not exist. If there is only one candidate key in a relation, it does not have an alternate key.

For example, employee relation has two attributes, Employee_Id and PAN_No, that act as candidate keys. In this relation, Employee_Id is chosen as the primary key, so the other candidate key, PAN_No, acts as the Alternate key.

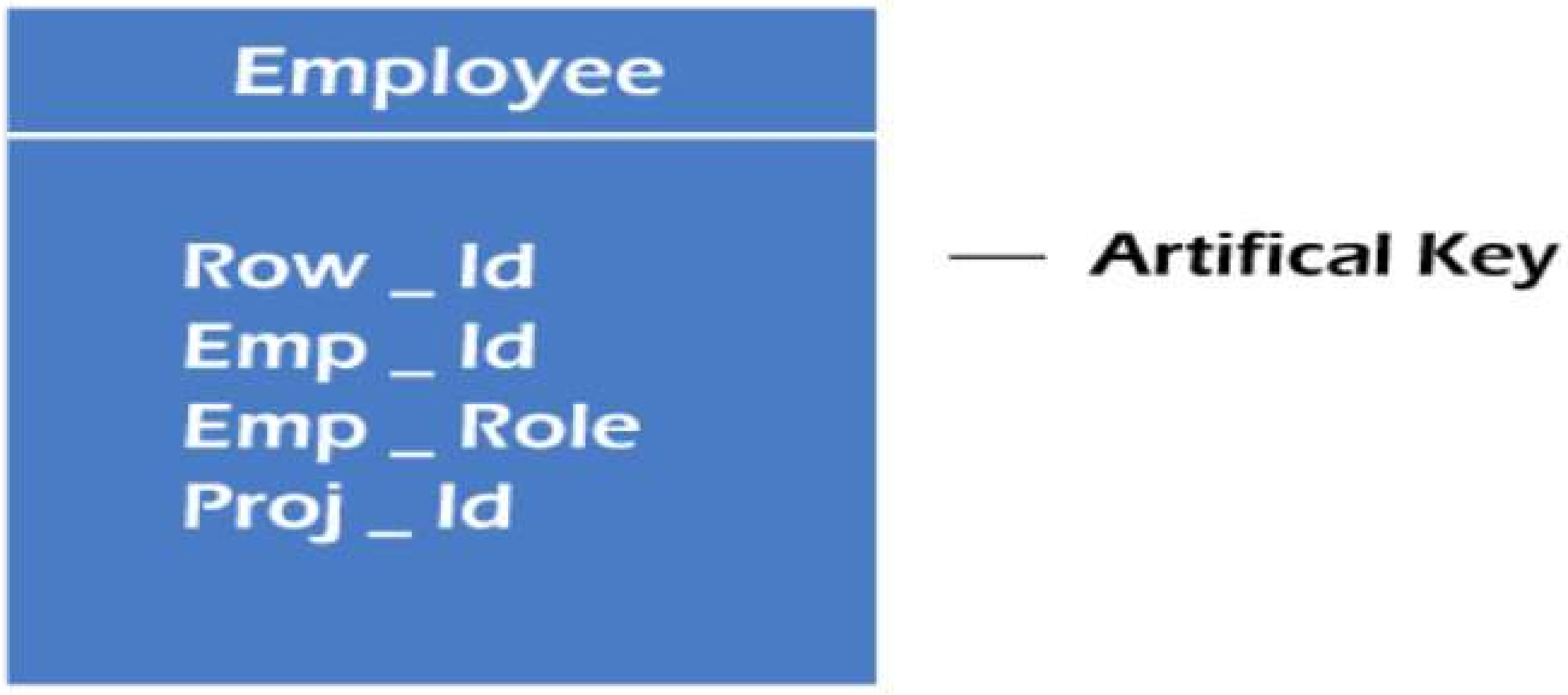


6. Composite key

Whenever a primary key consists of more than one attribute, it is known as a composite key. This key is also known as Concatenated Key.



For example, in employee relations, we assume that an employee may be assigned multiple roles, and an employee may work on multiple projects simultaneously. So the primary key will be composed of all three attributes, namely Emp_ID, Emp_role, and Proj_ID in combination. So these attributes act as a composite key since the primary key comprises more than one attribute.



7. Artificial key

The key created using arbitrarily assigned data are known as artificial keys. These keys are created when a primary key is large and complex and has no relationship with many other relations. The data values of the artificial keys are usually numbered in a serial order.

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

Class Hierarchies

In certain situations, some objects in a class have properties that are not shared by all objects n the class.

- In such a case, we might consider that, groups of objects with shared Properties form subclasses of the whole class.
- The subclass-super class relationship is an inheritance, and are often called is-a relationships because a member of the subclass is-a member of the super class.
- The relationship between super class and subclass is represented using Class Hierarchies.

The basic ER model fails to describe completely the reality of the data to be stored. With the increase in the types of the database applications, the basic concepts of the ER modeling are not sufficient to represent the requirements of the complex applications, such as Specialization and generalization (class hierarchies). The ER model that supports these additional semantic concepts is called Extended Entity Relationship (EER) Model.

The EER model includes all the concepts of the original E-R model together with the following additional concepts:

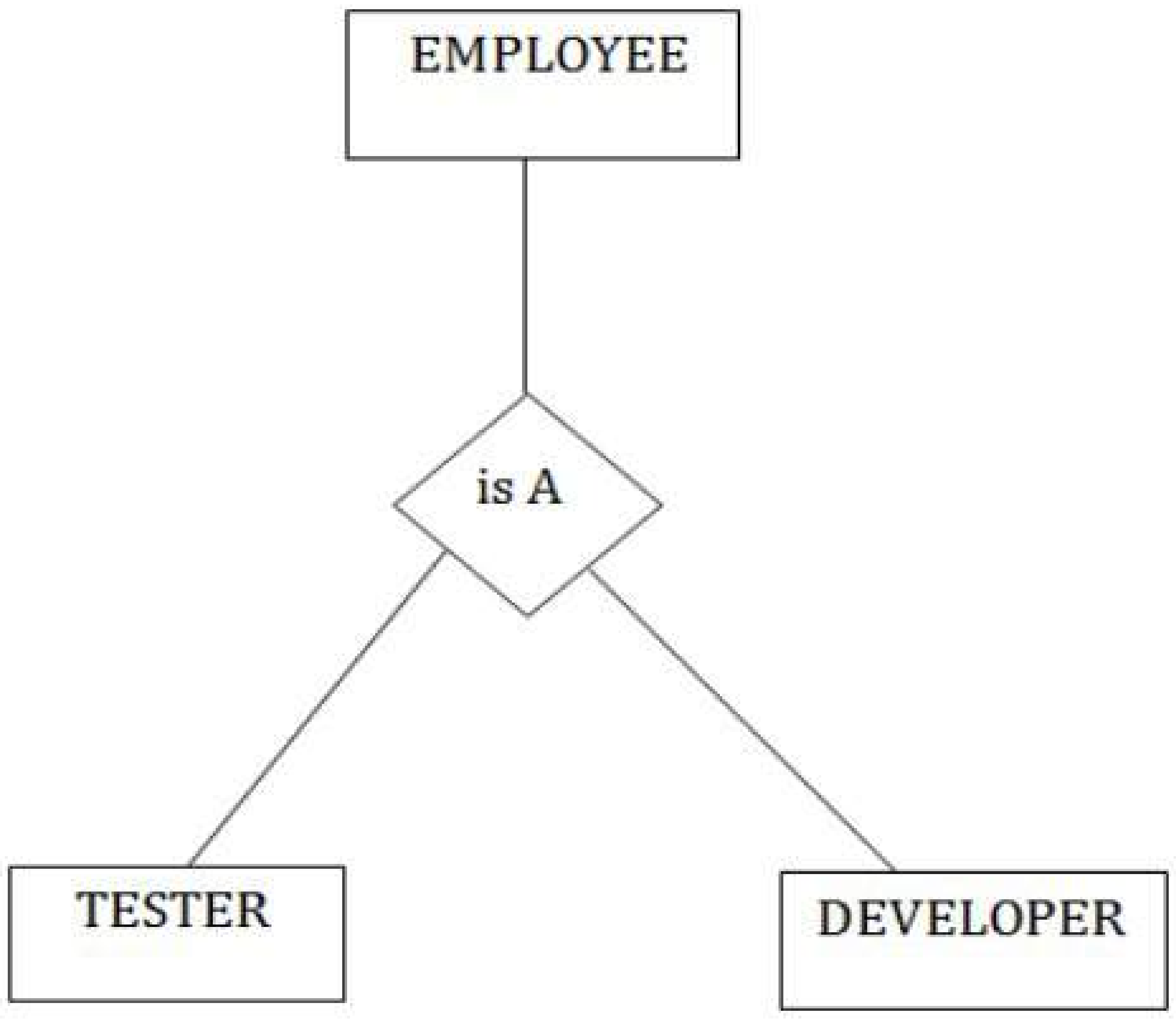
- Specializations
- Generalization
- Aggregation

Specialization

- Specialization is a top-down approach, and it is opposite to Generalization. In specialization, one higher level entity can be broken down into two lower level entities.

- Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics.
- Normally, the super class is defined first, the subclass and its related attributes are defined next, and relationship set are then added.

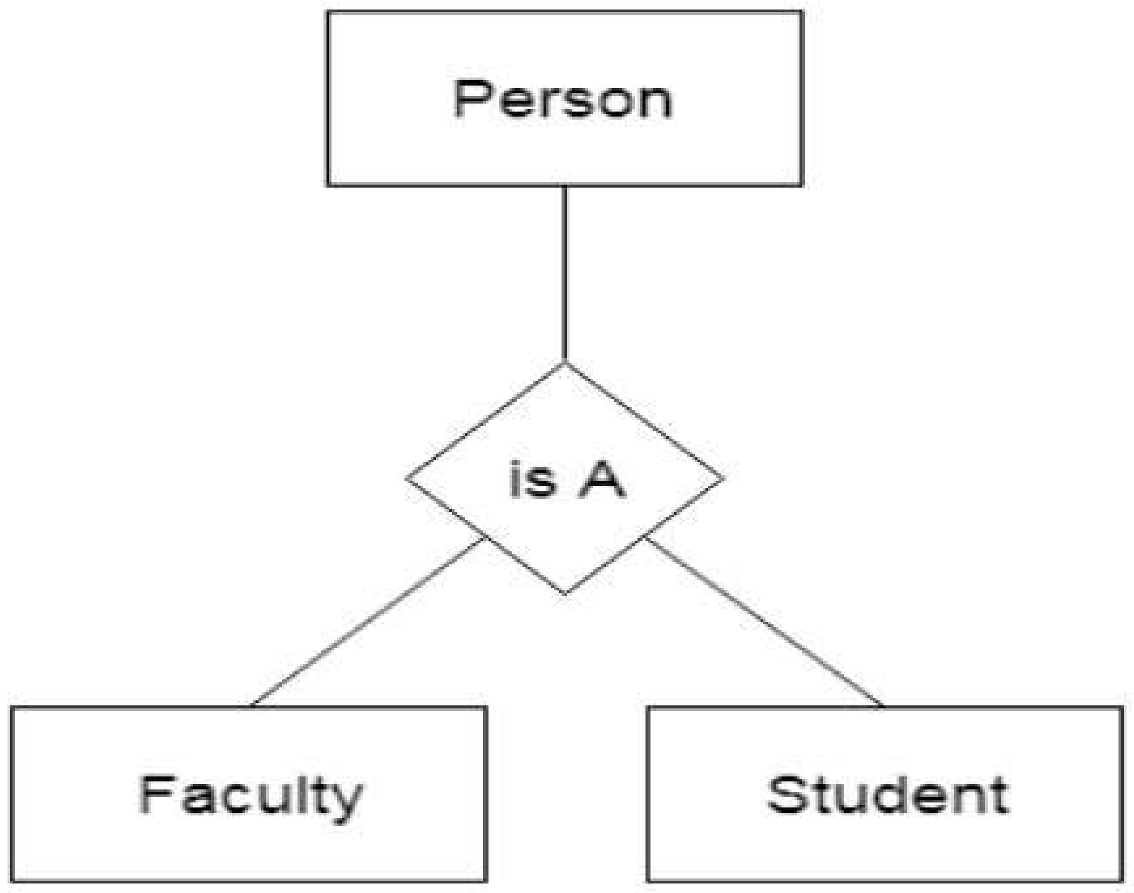
For example: In an Employee management system, EMPLOYEE entity can be specialized as TESTER or DEVELOPER based on what role they play in the company.



Generalization

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

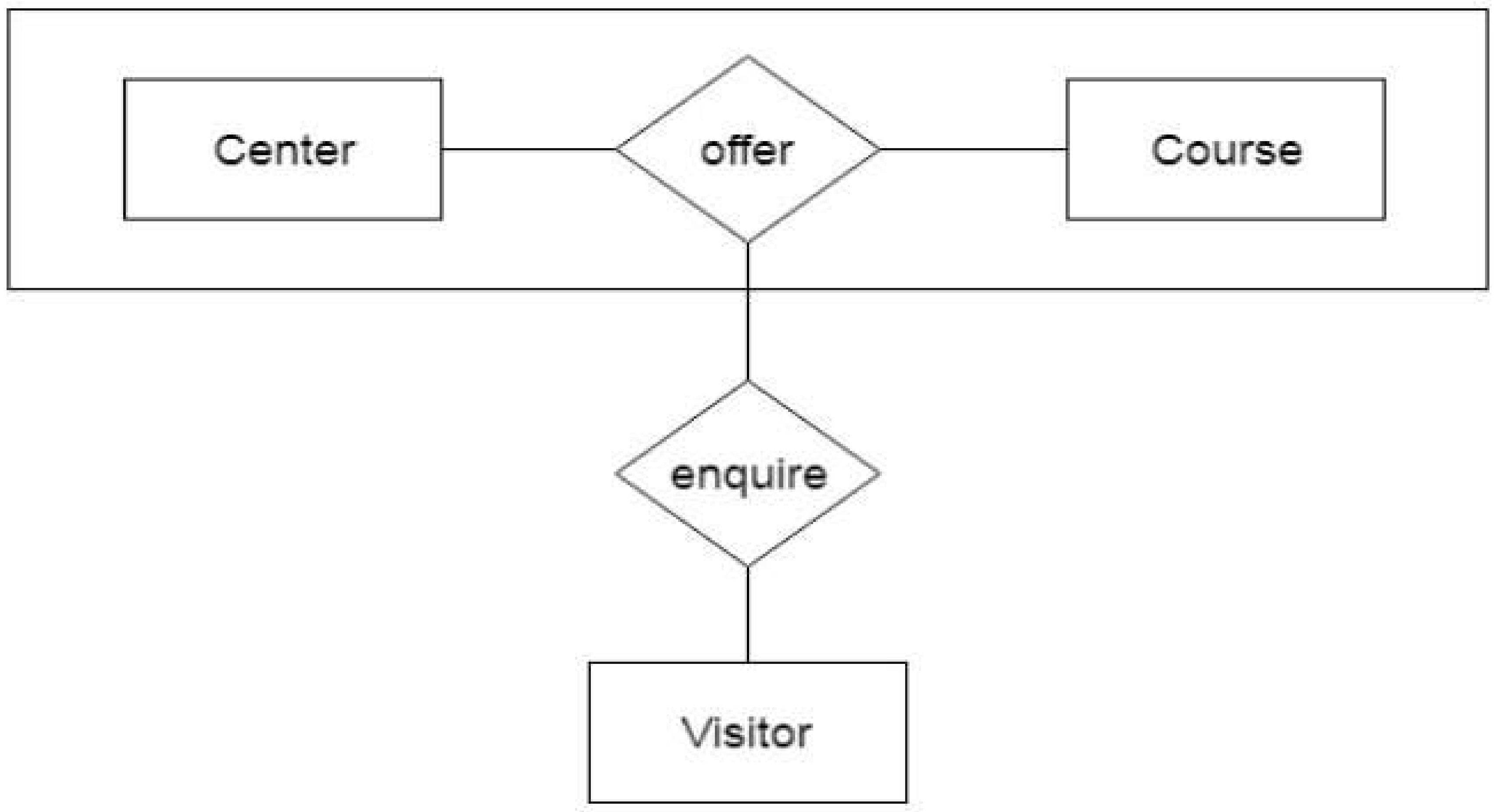
For example, Faculty and Student entities can be generalized and create a higher level entity Person.



Aggregation

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

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



ER Diagram for an Online Banking System

Description:

We want to build a system for online banking system. In the system the accounts can be opened in a branch. The branch gives loan to the customer. The customer borrows loan and the loan is paid through a weak entity “payment”. The customer can deposit in his account. The employee serves the customer. An account can be a saving account or a current account. Identify the possible entities and their attribute, the relationships among the attributes and draw the E-R diagram for the above mentioned activities and associations.

Solution:

Step 1: Identify the entities and their attributes

Entity_________________ Attributes____________________________

1. Branch branch-name, branch-city, assets

2. Customer customer-id, customer-name,

Customer-street, customer-city

3. Loan loan-number, amount

4. Payment payment-number, payment-amount, payment-date

5. Account account-number, balance

6. Saving-account interest-rate, Account no., Name

7. Current-account interest-rate, Account no. Name

8. Employee employee-ID, employee-name, start-date, telephone-number

Step 2: Identify the relationships among the entities

1. Branch have Account

2. Branch gives Loan

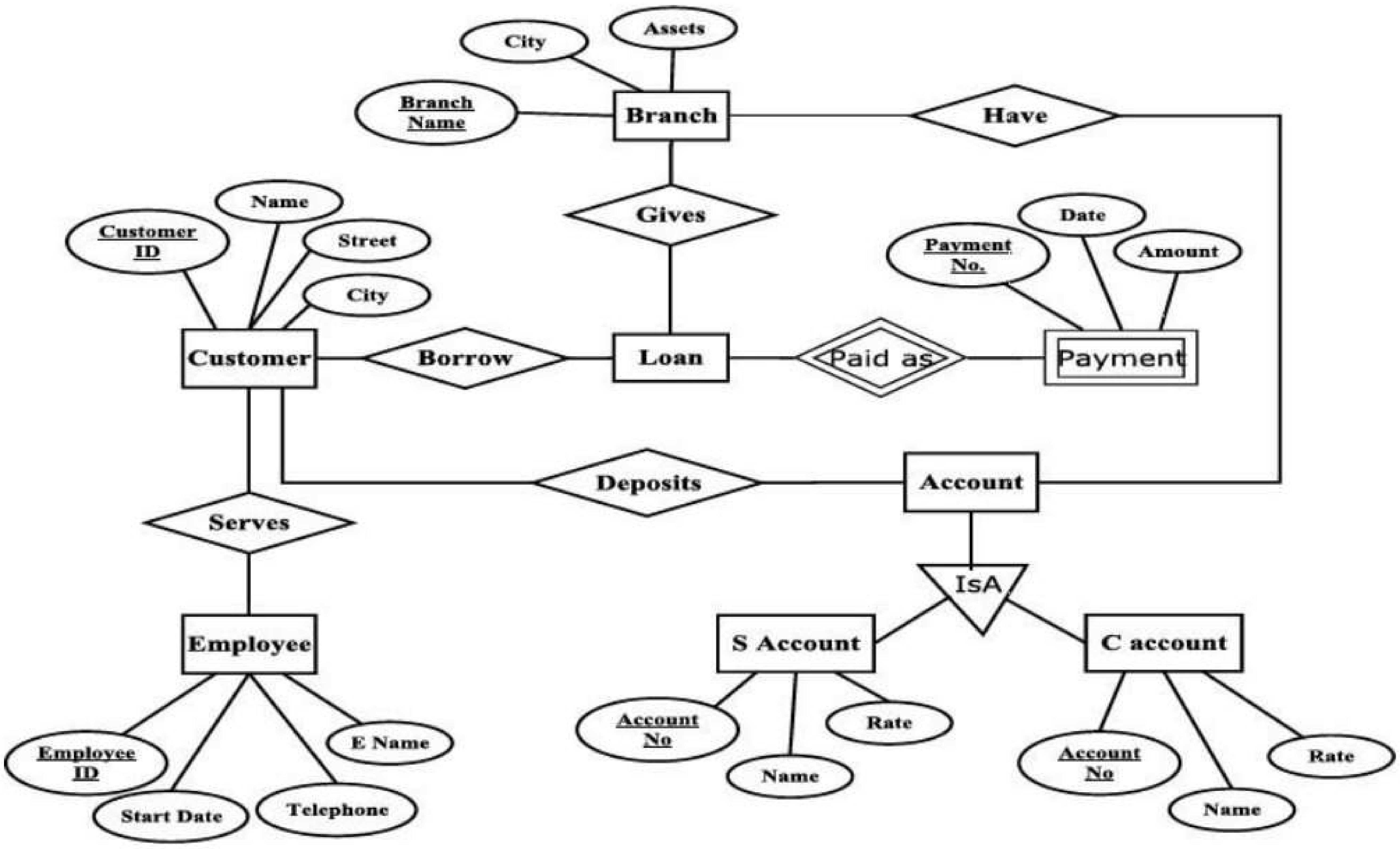
3. Payment loan-payment loan

4. Customer borrows loan

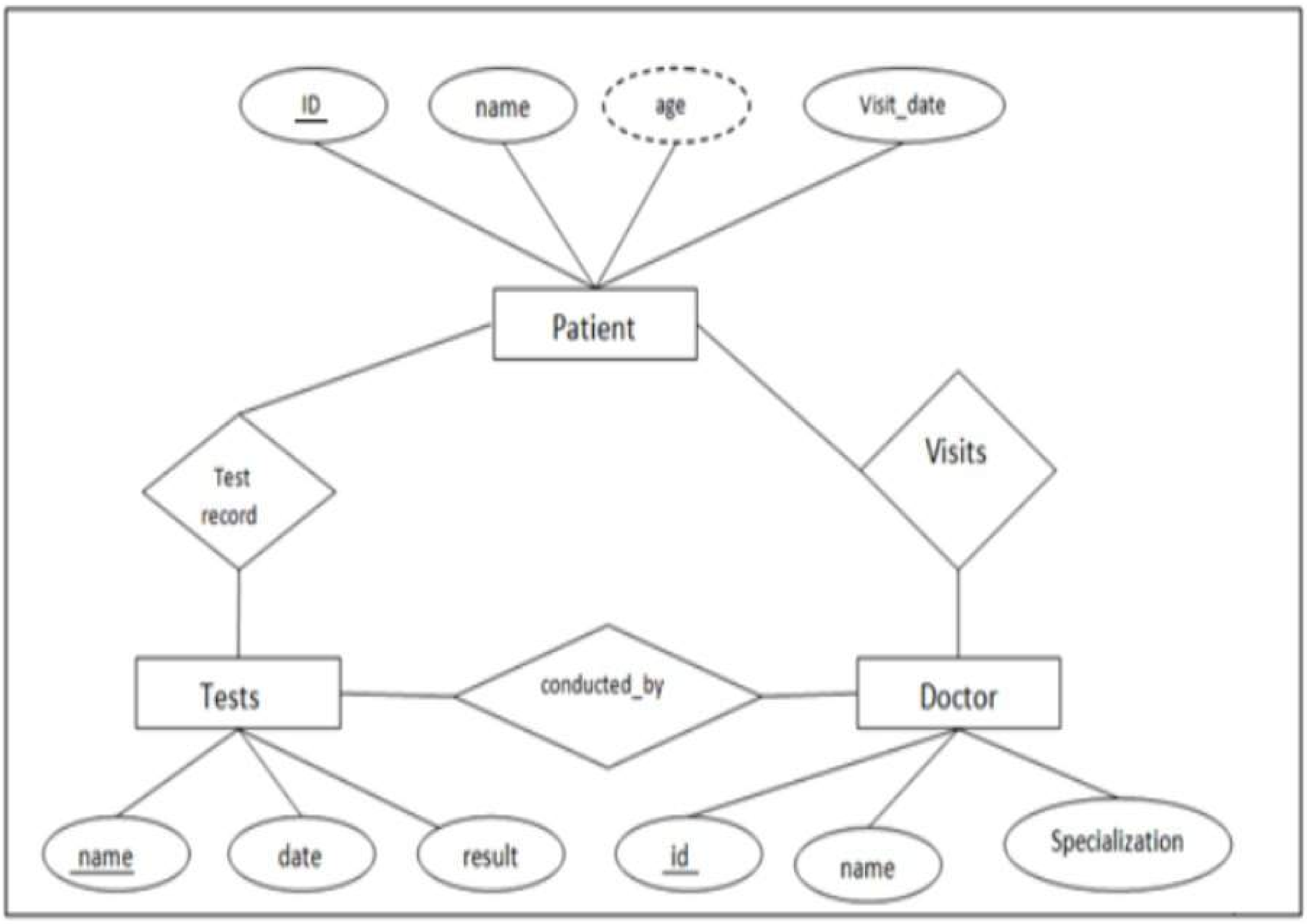
5. Customer deposits account

6. Employee serves Customer

Step 3: Draw the ER Diagram



Hospital ER Model



This is an ER model of a Hospital. The entities are represented in rectangular boxes and are Patient, Tests and Doctor.

Each of these entities have their respective attributes which are −

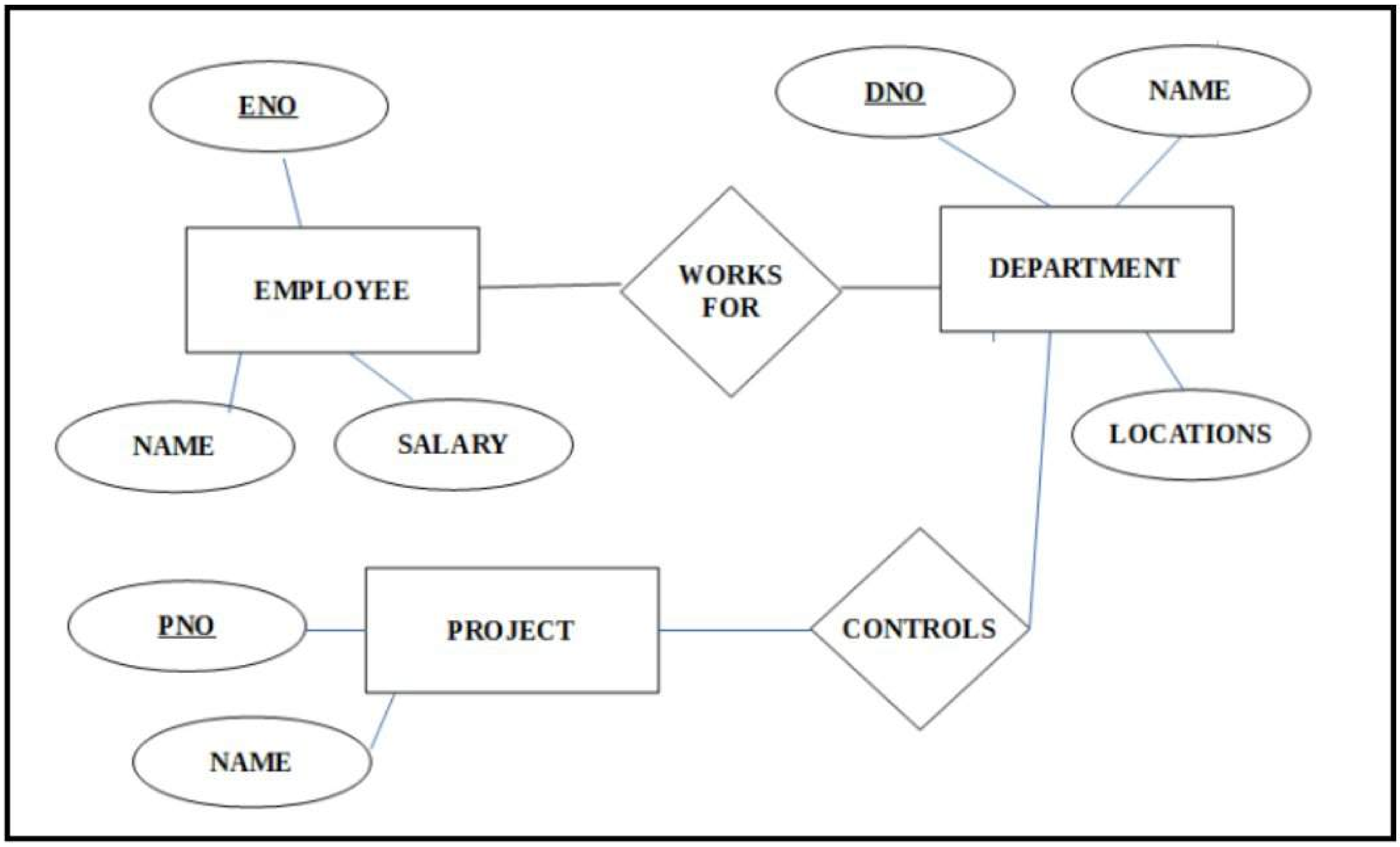
Patients - ID(primary key), name, age,visit_date

Tests- Name(primary key), date, result

Doctor- ID(primary key), name, specialization

The relationships between different entities are represented by a diamond shaped box.

Company ER Model



The entities in this ER model are Employee, Department and Project. These entities have the following attributes −

Employee - ENO(Primary Key) , Name, Salary

Department - DNO(Primary key), Name, Locations

Project - PNO(Primary key), Name

The relationships in this ER model are represented as Works for and Controls.

ER diagram for the Library management system

- In a library multiple students can enroll.
- Students can become a member by paying an appropriate fee.
- The books in the library are identified by a unique ID.

- Students can borrow multiple books from subscribed libraries.

Solution

Follow the steps given below to draw an ER model for the library management application −

Step 1 − Identify the entity sets

The entity set has multiple instances in a given business scenario.

As per the given constraints the entity sets are as follows −

- Book
- Publisher
- Member
- Section
- Granter

Step 2 − Identify the attributes for the given entities

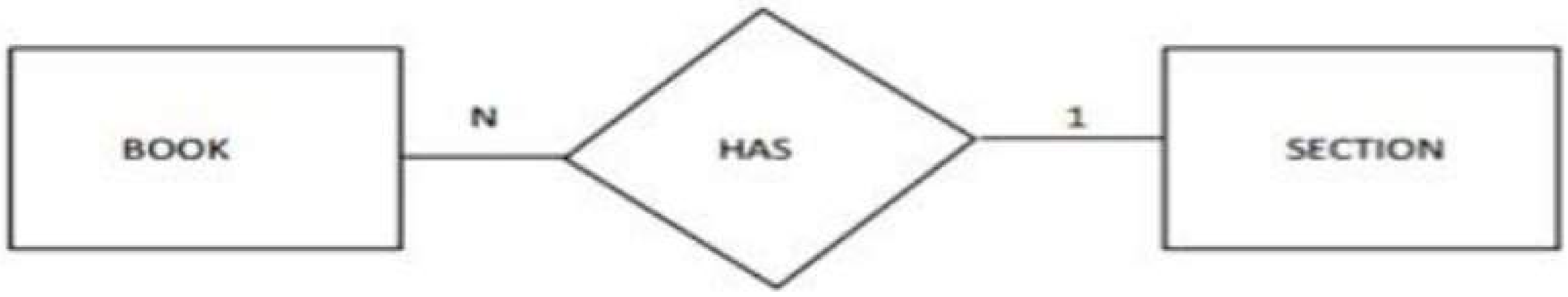
- Book − The relevant attributes are title, author, price, Isbn.
- Member − The relevant attributes are Name, Bday, MID, address, phone, age.
- Section − The relevant attributes are Sid, name, phone.
- Publisher − The relevant attributes are name, phone, Pid, address.
- Granter − The relevant attributes are phone, name, Nic, post, address.

Step 3 − Identify the Key attributes

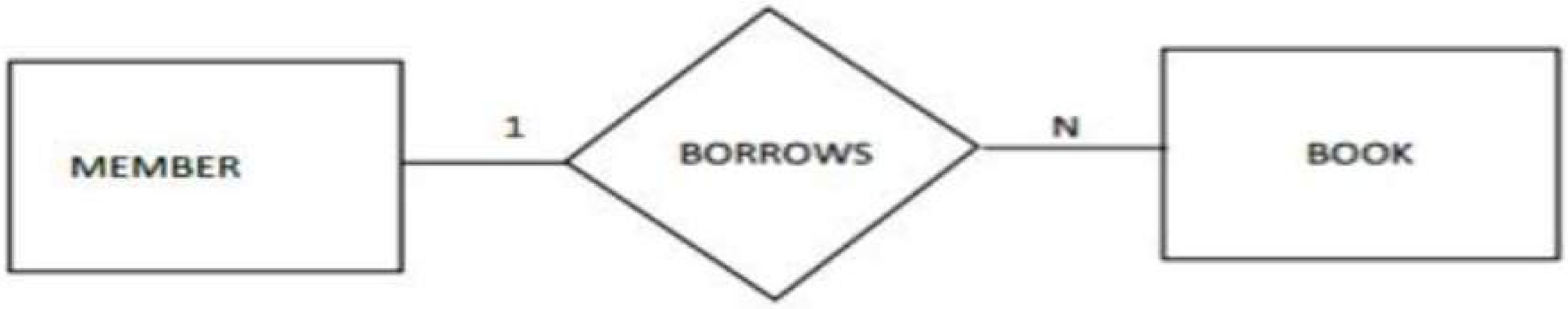
- Sid is the key attribute for the section.
- Mid is the key attribute for member entities.
- Isbn is the key attribute for a book entity.
- Pid is the key attribute for a publisher entity.
- Nic is the key attribute for a granter entity.

Step 4 − Identify the relationship between entity sets

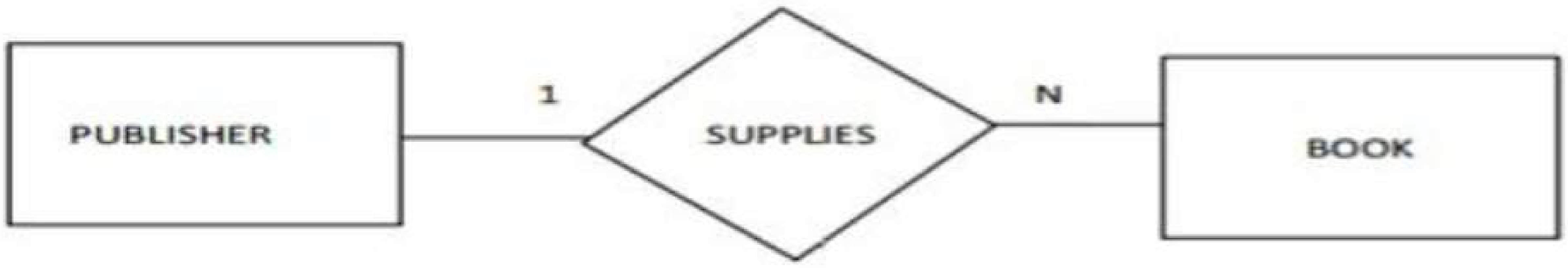
- Multiple books are arranged in a single section and one section has multiple books. Hence, the relationship between book and section is many to one.



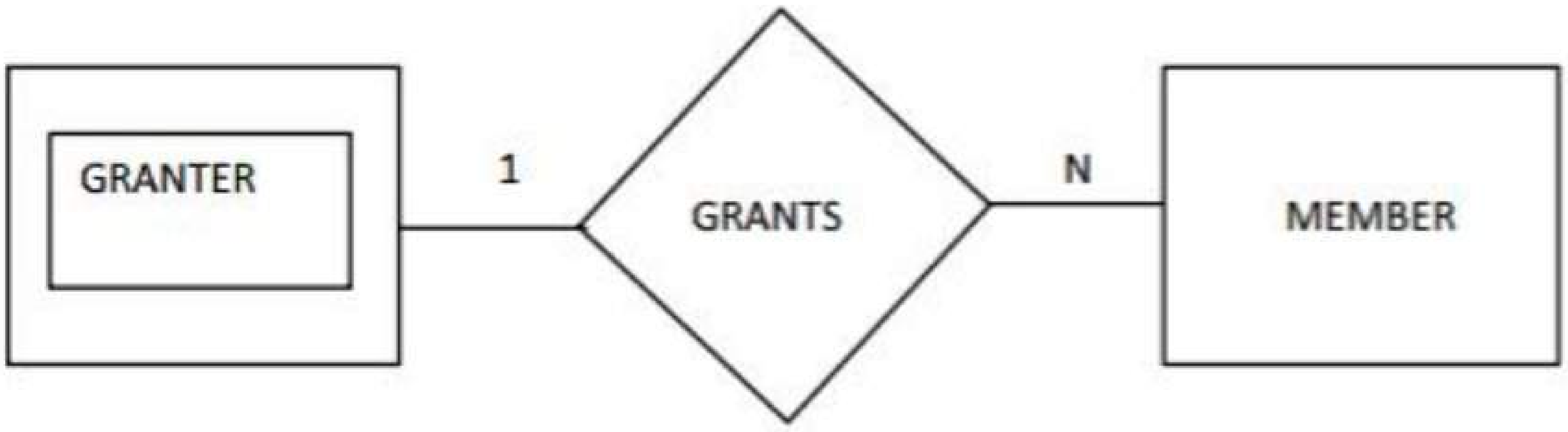
- One member borrows multiple books and multiple books can borrow a single person. Hence, the relationship between member and book is one-to-many.



- One publisher can supply multiple books and multiple books can be supplied by a single publisher. Hence, the relationship between publisher and book is one-to-many.

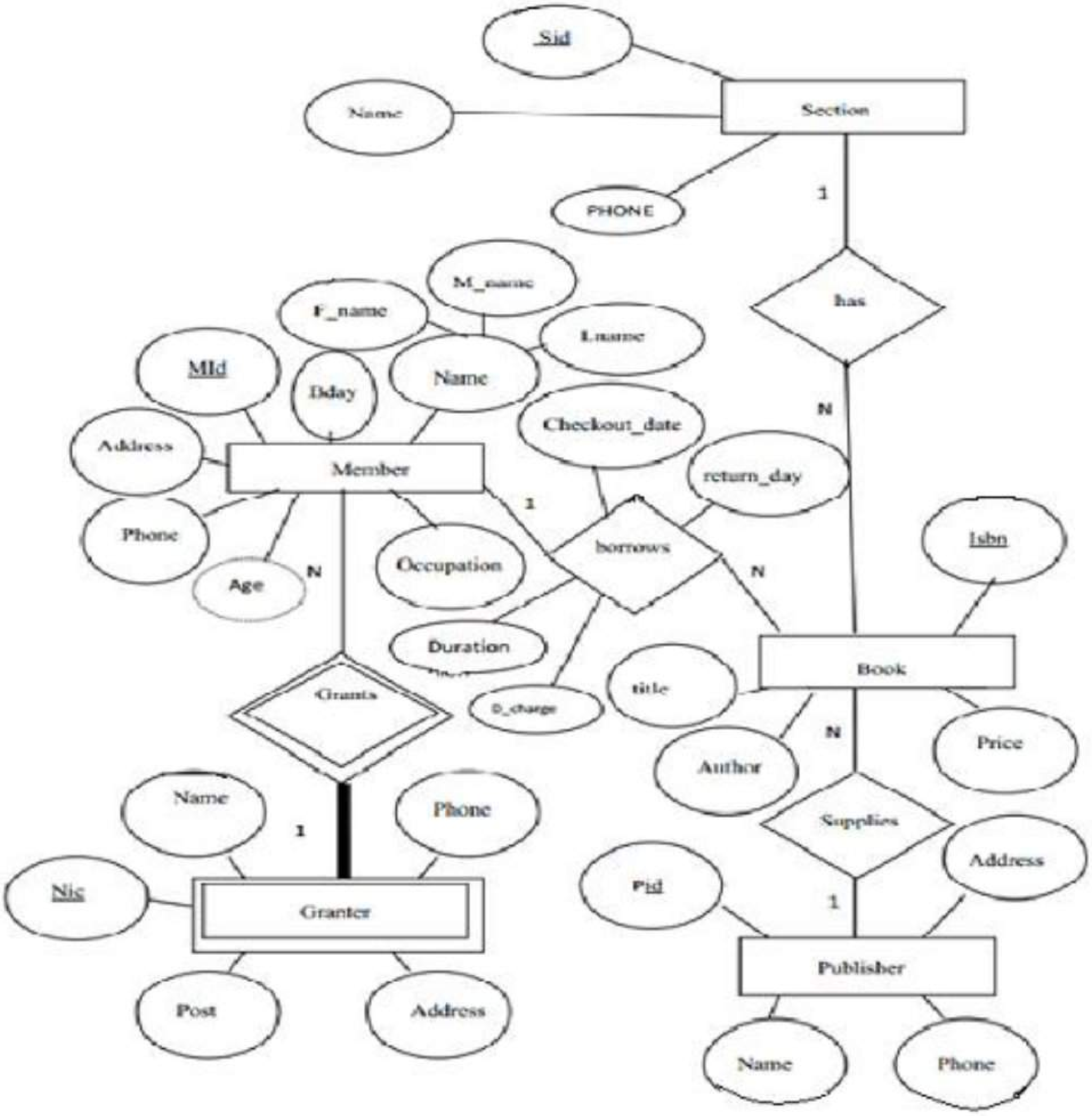


- One granter can grant multiple members and multiple members can grant a single granter. Hence, the relationship between grantor and member is one-to-many.



Step 5 − Complete ER diagram

The complete ER diagram is as follows −



Conceptual database Design with the E-R Model

Various design issues related to E-R model are given below

In the previous sections of the data modeling, we learned to design an ER diagram. We also discussed different ways of defining entity sets and relationships among them. We also understood the various designing shapes that represent a relationship, an entity, and its attributes. However, users often mislead the concept of the elements and the design process of the ER diagram. Thus, it leads to a complex structure of the ER diagram and certain issues that does not meet the characteristics of the real-world enterprise model.

Here, we will discuss the basic design issues of an ER database schema in the following points:

1) Use of Entity Set vs Attributes

The use of an entity set or attribute depends on the structure of the real-world enterprise that is being modeled and the semantics associated with its attributes. It leads to a mistake when the user use the primary key of an entity set as an attribute of another entity set. Instead, he should use the relationship to do so. Also, the primary key attributes are implicit in the relationship set, but we designate it in the relationship sets.

2) Use of Entity Set vs. Relationship Sets

It is difficult to examine if an object can be best expressed by an entity set or relationship set. To understand and determine the right use, the user need to designate a relationship set for describing an action that occurs in-between the entities. If there is a requirement of representing the object as a relationship set, then its better not to mix it with the entity set.

3) Use of Binary vs n-ary Relationship Sets

Generally, the relationships described in the databases are binary relationships. However, non-binary relationships can be represented by several binary relationships. For example, we can create and represent a ternary relationship 'parent' that may relate to a child, his father, as well as his mother. Such relationship can also be represented by two binary relationships i.e, mother and father, that may relate to their child. Thus, it is possible to represent a non-binary relationship by a set of distinct binary relationships.

4) Placing Relationship Attributes

The cardinality ratios can become an affective measure in the placement of the relationship attributes. So, it is better to associate the attributes of one-to-one or one-to-many relationship sets with any participating entity sets, instead of any relationship set. The decision of placing the specified attribute as a relationship or entity attribute should possess the charactestics of the real world enterprise that is being modeled.

For example, if there is an entity which can be determined by the combination of participating entity sets, instead of determining it as a separate entity. Such type of attribute must be associated with the many-to-many relationship sets.

Thus, it requires the overall knowledge of each part that is involved into designing and modeling an ER diagram. The basic requirement is to analyze the real-world enterprise and the connectivity of one entity or attribute with other.

UNIT-2

Relational Model concept

Relational model can represent as a table with columns and rows. Each row is known as a tuple. Each table of the column has a name or attribute.

Domain: It contains a set of atomic values that an attribute can take.

Attribute: It contains the name of a column in a particular table. Each attribute Ai must have a domain, dom(Ai)

Relational instance: In the relational database system, the relational instance is represented by a finite set of tuples. Relation instances do not have duplicate tuples.

Relational schema: A relational schema contains the name of the relation and name of all columns or attributes.

Relational key: In the relational key, each row has one or more attributes. It can identify the row in the relation uniquely.

Example: STUDENT Relation

NAME	ROLL_NO	PHONE_NO	ADDRESS	AGE
Ram	14795	7305758992	Noida	24
Shyam	12839	9026288936	Delhi	35
Laxman	33289	8583287182	Gurugram	20
Mahesh	27857	7086819134	Ghaziabad	27

Ganesh	17282	9028 9i3988	Delhi	40

- In the given table, NAME, ROLL_NO, PHONE_NO, ADDRESS, and AGE are the attributes.
- The instance of schema STUDENT has 5 tuples.
- t3 = <Laxman, 33289, 8583287182, Gurugram, 20>

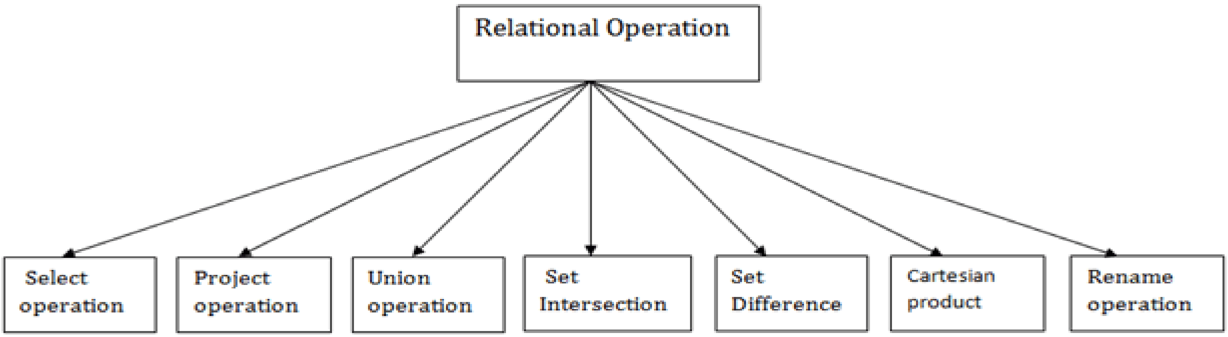
Properties of Relations

- Name of the relation is distinct from all other relations.
- Each relation cell contains exactly one atomic (single) value
- Each attribute contains a distinct name
- Attribute domain has no significance
- tuple has no duplicate value
- Order of tuple can have a different sequence

Relational Algebra

Relational algebra is a procedural query language. It gives a step by step process to obtain the result of the query. It uses operators to perform queries.

Types of Relational operation



1. Select Operation:

- The select operation selects tuples that satisfy a given predicate.
- It is denoted by sigma (σ).

Notation: σ p(r)

Where:

σ is used for selection prediction
r is used for relation
p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like =, ≠, ≥, <, >, ≤.

For example: LOAN Relation

BRANCH_NAME	LOAN_NO	AMOUNT
Downtown	L-17	1000
Redwood	L-23	2000
Perryride	L-15	1500
Downtown	L-14	1500
Mianus	L-13	500
Roundhill	L-11	900
Perryride	L-16	1300

Input:

σ BRANCH_NAME="perryride" (LOAN)

Output:

BRANCH_NAME	**LOAN_NO**	**AMOUNT**
Perryride	L-15	1500
Perryride	L-16	1300

2. Project Operation:

- This operation shows the list of those attributes that we wish to appear in the result. Rest of the attributes are eliminated from the table.
- It is denoted by ∏.

Notation: ∏ A1, A2, An (r)

Where

A1, **A2**, **A3** is used as an attribute name of relation **r**.

Example: CUSTOMER RELATION

NAME	**STREET**	**CITY**
Jones	Main	Harrison
Smith	North	Rye

Hays	Main	Harrison
Curry	North	Rye
Johnson	Alma	Brooklyn
Brooks	Senator	Brooklyn

Input:

∏ NAME, CITY (CUSTOMER)

Output:

NAME	**CITY**
Jones	Harrison
Smith	Rye
Hays	Harrison
Curry	Rye
Johnson	Brooklyn
Brooks	Brooklyn

3. Union Operation:

- Suppose there are two tuples R and S. The union operation contains all the tuples that are either in R or S or both in R & S.
- It eliminates the duplicate tuples. It is denoted by ∪.

Notation: R ∪ S

A union operation must hold the following condition:

- R and S must have the attribute of the same number.
- Duplicate tuples are eliminated automatically.

Example:

DEPOSITOR RELATION

CUSTOMER_NAME	**ACCOUNT_NO**
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472

Lindsay	A-284

BORROW RELATION

CUSTOMER_NAME	**LOAN_NO**
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17

Input:

∏ CUSTOMER_NAME (BORROW) ∪ ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Johnson

Smith
Hayes
Turner
Jones
Lindsay
Jackson
Curry
Williams
Mayes

4. Set Intersection:

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S.
- It is denoted by intersection ∩.

Notation: R ∩ S

Example: Using the above DEPOSITOR table and BORROW table

Input:

∏ CUSTOMER_NAME (BORROW) ∩ ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Smith
Jones

5. Set Difference:

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in R but not in S.
- It is denoted by intersection minus (-).

Notation: R - S

Example: Using the above DEPOSITOR table and BORROW table

Input:

∏ CUSTOMER_NAME (BORROW) - ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Jackson
Hayes
Willians

Curry

6. Cartesian product

- The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.
- It is denoted by X.

Notation: E X D

Example:

EMPLOYEE

EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

DEPARTMENT

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal

Input:

1. EMPLOYEE X DEPARTMENT

Output:

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

7. Rename Operation:

The rename operation is used to rename the output relation. It is denoted by **rho** (ρ).

Example: We can use the rename operator to rename STUDENT relation to STUDENT1.

Join Operations:

A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by ⋈.

Example:

EMPLOYEE

EMP_CODE	**EMP_NAME**
101	Stephan
102	Jack
103	Harry

SALARY

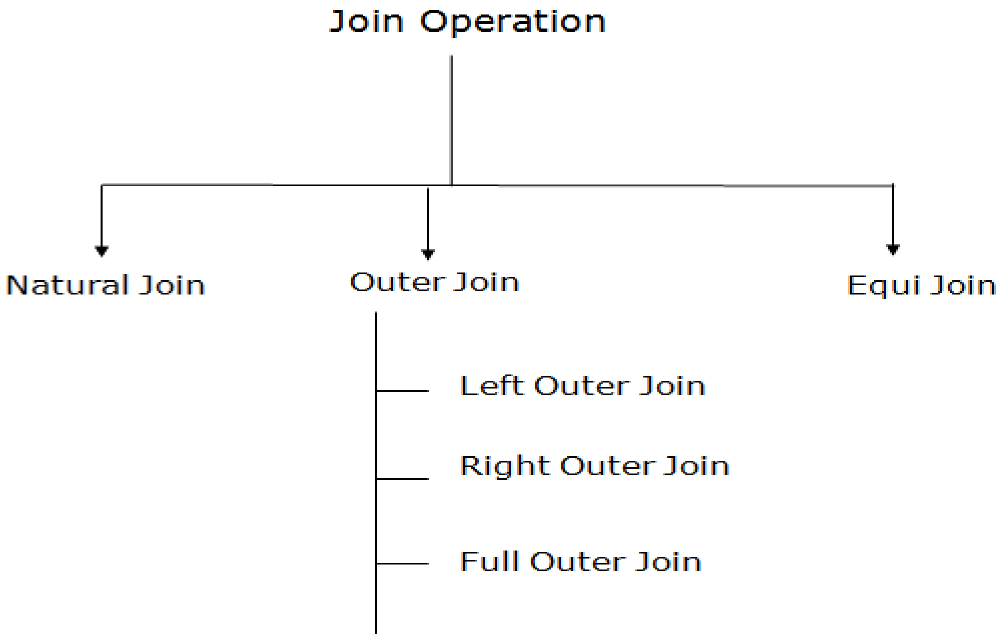
EMP_CODE	**SALARY**
101	50000
102	30000
103	25000

1. Operation: (EMPLOYEE ⋈ SALARY)

Result:

EMP_CODE	**EMP_NAME**	**SALARY**
101	Stephan	50000
102	Jack	30000
103	Harry	25000

Types of Join operations:



1. Natural Join:

- A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.
- It is denoted by ⋈.

Example: Let's use the above EMPLOYEE table and SALARY table:

Input:

1. ∏EMP_NAME, SALARY (EMPLOYEE ⋈ SALARY)

Output:

EMP_NAME	SALARY
Stephan	50000
Jack	30000
Harry	25000

2. Outer Join:

The outer join operation is an extension of the join operation. It is used to deal with missing information.

Example:

EMPLOYEE

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai
Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000
Hari	TCS	50000

Input:

1. (EMPLOYEE ⋈ FACT_WORKERS)

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru nagar	Hyderabad	TCS	50000

An outer join is basically of three types:

a. Left outer join

b. Right outer join

c. Full outer join

a. Left outer join:

- Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In the left outer join, tuples in R have no matching tuples in S.
- It is denoted by ⋈.

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

1. EMPLOYEE ⋈ FACT_WORKERS

EMP_NAME	STREET	CITY	BRANCH	SALARY

Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL

b. Right outer join:

- Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In right outer join, tuples in S have no matching tuples in R.
- It is denoted by ⟖.

Example: Using the above EMPLOYEE table and FACT_WORKERS Relation

Input:

1. EMPLOYEE ⟖ FACT_WORKERS

Output:

EMP_NAME	BRANCH	SALARY	STREET	CITY
Ram	Infosys	10000	Civil line	Mumbai
Shyam	Wipro	20000	Park street	Kolkata
Hari	TCS	50000	Nehru street	Hyderabad

Kuber	HCL	30000	NULL	NULL

c. Full outer join:

- Full outer join is like a left or right join except that it contains all rows from both tables.
- In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.
- It is denoted by ⟗.

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

1. EMPLOYEE ⟗ FACT_WORKERS

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	HCL	30000

3. Equi join:

It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator(=).

Example:

CUSTOMER RELATION

CLASS_ID	NAME
1	John
2	Harry
3	Jackson

PRODUCT

PRODUCT_ID	CITY
1	Delhi
2	Mumbai
3	Noida

Input:

1. CUSTOMER ⋈ PRODUCT

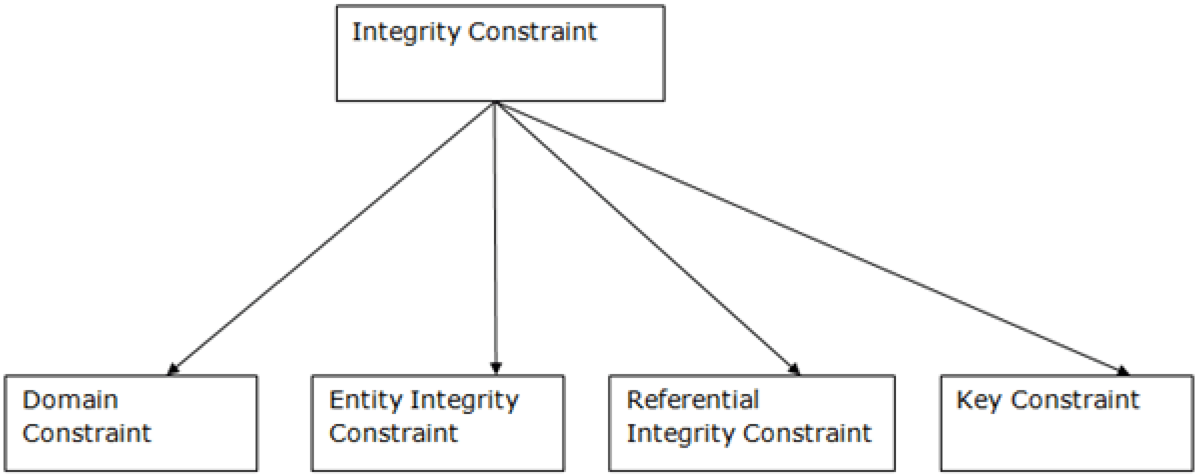
Output:

CLASS_ID	NAME	PRODUCT_ID	CITY
1	John	1	Delhi
2	Harry	2	Mumbai
3	Harry	3	Noida

Integrity Constraints

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

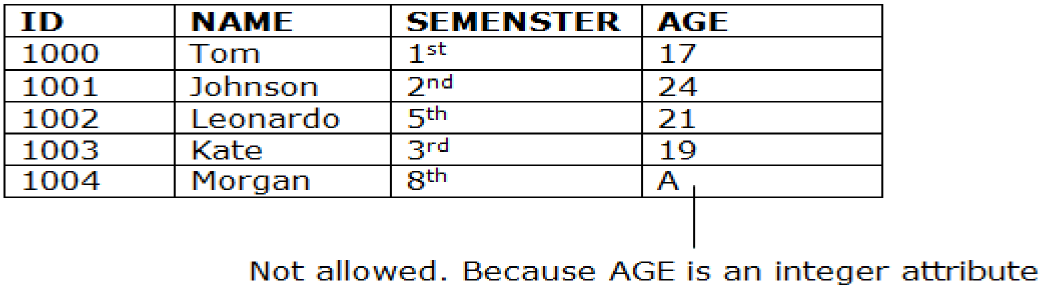
Types of Integrity Constraint



1. Domain constraints

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

Example:



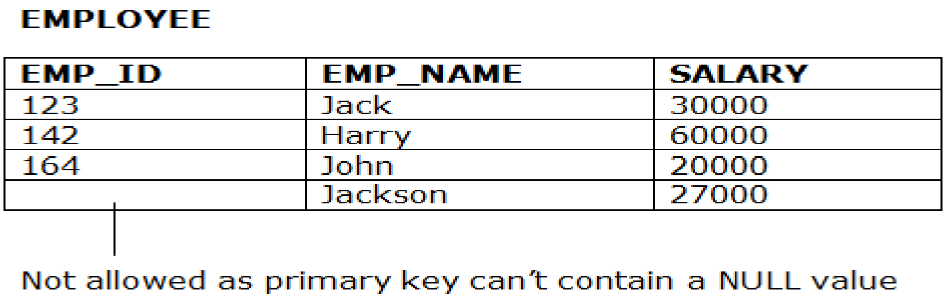
ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

2. Entity integrity constraints

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

Example:



EMPLOYEE

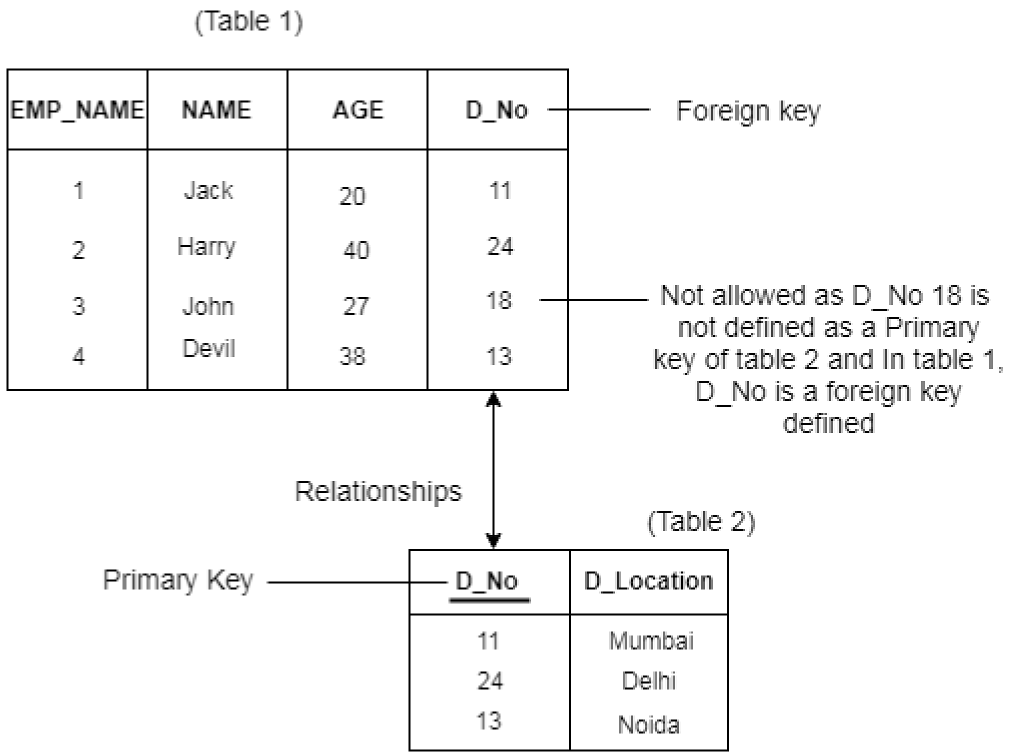
EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

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

3. Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Example:



(Table 1)
Foreign key
Not allowed as D_No 18 is not defined as a Primary key of table 2 and In table 1, D_No is a foreign key defined
Relationships
(Table 2)
Primary Key

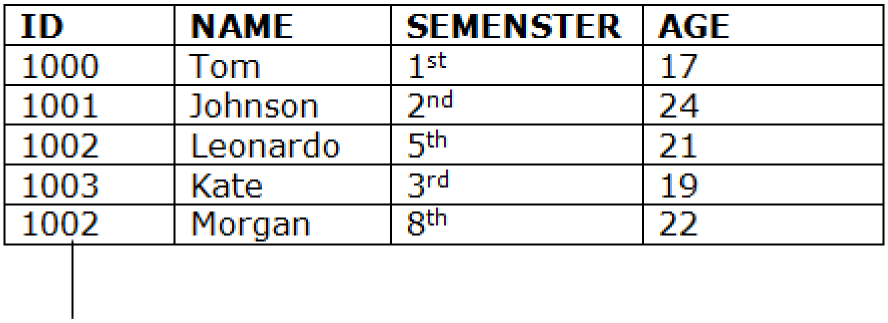
EMP_NAME	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

4. Key constraints

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

Example:



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

Not allowed. Because all row must be unique

<u>Relational Calculus</u>

There is an alternate way of formulating queries known as Relational Calculus. Relational calculus is a non-procedural query language. In the non-procedural query language, the user is concerned with the details of how to obtain the end results. The relational calculus tells what to do but never explains how to do. Most commercial relational languages are based on aspects of relational calculus including SQL-QBE and QUEL.

Why it is called Relational Calculus?

It is based on Predicate calculus, a name derived from branch of symbolic language. A predicate is a truth-valued function with arguments. On substituting values for the arguments, the function result in an expression called a proposition. It can be either true or false. It is a tailored version of a subset of the Predicate Calculus to communicate with the relational database.

Many of the calculus expressions involves the use of Quantifiers. There are two types of quantifiers:

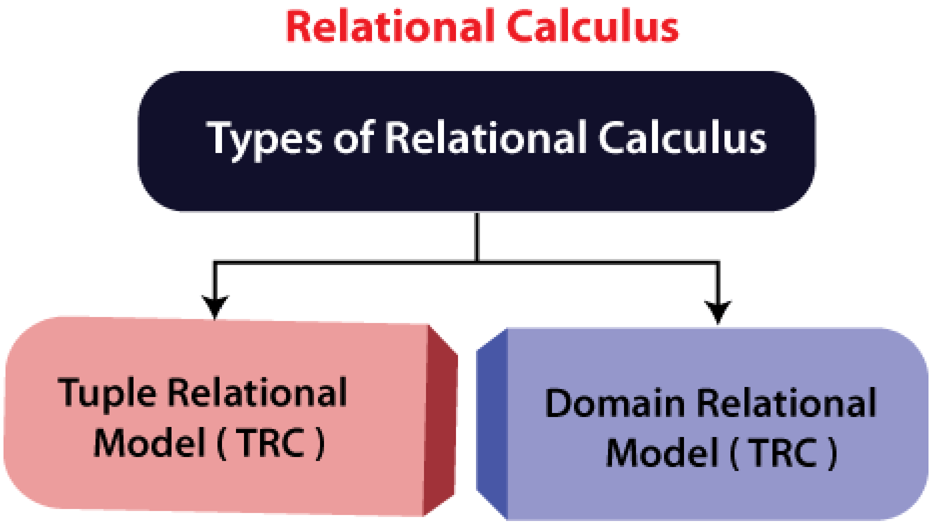
- **Universal Quantifiers:** The universal quantifier denoted by ∀ is read as for all which means that in a given set of tuples exactly all tuples satisfy a given condition.
- **Existential Quantifiers:** The existential quantifier denoted by ∃ is read as for all which means that in a given set of tuples there is at least one occurrences whose value satisfy a given condition.

Before using the concept of quantifiers in formulas, we need to know the concept of Free and Bound Variables.

A tuple variable t is bound if it is quantified which means that if it appears in any occurrences a variable that is not bound is said to be free.

Free and bound variables may be compared with global and local variable of programming languages.

Types of Relational calculus:


Relational Calculus
Types of Relational Calculus
Tuple Relational Model (TRC)
Domain Relational Model (TRC)

1. Tuple Relational Calculus (TRC)

It is a non-procedural query language which is based on finding a number of tuple variables also known as range variable for which predicate holds true. It describes the desired information without giving a specific procedure for obtaining that information. The tuple relational calculus is specified to select the tuples in a relation. In TRC, filtering variable uses the tuples of a relation. The result of the relation can have one or more tuples.

Notation:

A Query in the tuple relational calculus is expressed as following notation

{T | P (T)}   or {T | Condition (T)}

Where

T is the resulting tuples

P(T) is the condition used to fetch T.

For example:

{ T.name | Author(T) AND T.article = 'database' }

Output: This query selects the tuples from the AUTHOR relation. It returns a tuple with 'name' from Author who has written an article on 'database'.

TRC (tuple relation calculus) can be quantified. In TRC, we can use Existential (∃) and Universal Quantifiers (∀).

For example:

{ R| ∃T ∈ Authors(T.article='database' AND R.name=T.name)}

Output: This query will yield the same result as the previous one.

2. Domain Relational Calculus (DRC)

The second form of relation is known as Domain relational calculus. In domain relational calculus, filtering variable uses the domain of attributes. Domain relational calculus uses the same operators as tuple calculus. It uses logical connectives ∧ (and), ∨ (or) and ┓ (not). It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable. The QBE or Query by example is a query language related to domain relational calculus.

Notation:

{ a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}

Where

a1,a2 areattributes
P stands for formula built by inner attributes

For example:

$\{< \text{article, page, subject} > \mid \ \in \text{javatpoint} \wedge \text{subject} = \text{'database'}\}$

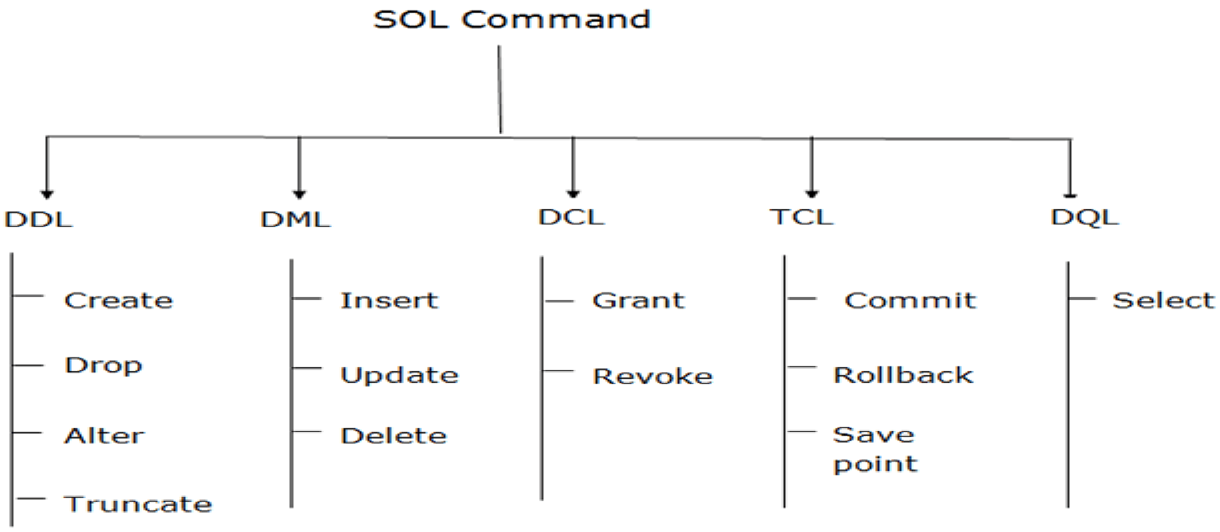
Output: This query will yield the article, page, and subject from the relational javatpoint, where the subject is a database.

SQL Commands

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

Types of SQL Commands

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



1. **Data Definition Language (DDL)**

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

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

a. CREATE It is used to create a new table in the database.

Syntax:

```
CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);
```

Example:

```
CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);
```

b. DROP: It is used to delete both the structure and record stored in the table.

Syntax

```
DROP TABLE table_name;
```

Example

```
DROP TABLE EMPLOYEE;
```

c. ALTER: It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

Syntax:

To add a new column in the table

```
ALTER TABLE table_name ADD column_name COLUMN-definition;
```

To modify existing column in the table:

```
ALTER TABLE table_name MODIFY(column_definitions....);
```

EXAMPLE

```
ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));
ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));
```

d. TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.

Syntax:

```
TRUNCATE TABLE table_name;
```

Example:

```
TRUNCATE TABLE EMPLOYEE;
```

2. Data Manipulation Language(DML)

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

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

a. INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

Syntax:

```
INSERT INTO TABLE_NAME (col1, col2, col3,.... col N)  VALUES (value1, value2, value3, .... valueN);
```

Or

```
INSERT INTO TABLE_NAME  VALUES (value1, value2, value3, .... valueN);
```

For example:

```
INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");
```

b. UPDATE: This command is used to update or modify the value of a column in the table.

Syntax:

```
UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CONDITION]
```

For example:

```
UPDATE students
SET User_Name = 'Sonoo'
WHERE Student_Id = '3';
```

c. DELETE: It is used to remove one or more row from a table.

Syntax:

```
DELETE FROM table_name [WHERE condition];
```

For example:

```
DELETE FROM javatpoint

WHERE Author="Sonoo";
```

3. Data Control Language(DCL)

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

Here are some commands that come under DCL:

- Grant
- Revoke

a. Grant: It is used to give user access privileges to a database.

Example

```
GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;
```

b. Revoke: It is used to take back permissions from the user.

Example

```
REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;
```

4. Transaction Control Language(TCL)

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

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

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

a. Commit: Commit command is used to save all the transactions to the database.

Syntax:

```
COMMIT;
```

Example:

```
DELETE FROM CUSTOMERS

WHERE AGE = 25;
COMMIT;
```

b. Rollback: Rollback command is used to undo transactions that have not already been saved to the database.

Syntax:

```
ROLLBACK;
```

Example:

DELETE FROM CUSTOMERS

WHERE AGE = 25;
ROLLBACK;

c. SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:

SAVEPOINT SAVEPOINT_NAME;

5. Data Query Language(DQL)

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

a. SELECT: This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Syntax:

SELECT expressions

FROM TABLES

WHERE conditions;

For example:

SELECT emp_name FROM employee WHERE age > 20;

Difference between DDL and DML

A database is a list of related records, and the Database Management System is the most common way to manage these databases (DBMS). The SQL (Structured Query Language)

Commands are needed to interact with database systems. These SQL commands can be used to build tables, insert data into tables, remove or drop tables, change tables, and set permissions for users. We can categorize the SQL commands as DDL, DQL, DCL, and DML.

This article explains the complete overview of DDL and DML languages. The difference between DDL and DML commands is the most common part of an interview question. **The key distinction is that the DDL command is used to create a database schema, while the DML command is used to modify the table's existing data**. Before making the comparison, we will first know these SQL commands

What is a DDL command?

DDL stands for Data Definition Language. As the name suggests, the DDL commands help to define the structure of the databases or schema. When we execute DDL statements, it takes effect immediately. The changes made in the database using this command are saved permanently because its commands are auto-committed. The following commands come under DDL language:

- **CREATE**: It is used to create a new database and its objects such as table, views, function, stored procedure, triggers, etc.
- **DROP**: It is used to delete the database and its objects, including structures, from the server permanently.
- **ALTER**: It's used to update the database structure by modifying the characteristics of an existing attribute or adding new attributes.
- **TRUNCATE**: It is used to completely remove all data from a table, including their structure and space allocates on the server.
- **RENAME**: This command renames the content in the database.

Why we use DDL commands?

The following are the reasons to use DDL commands:

- It allows us to store shared data in a database.
- It improved integrity due to the data independence feature.
- It will enable multiple users to work on the same databases.
- It improved security efficient data access.

What is a DML command?

It stands for Data Manipulation Language. The DML commands deal with the manipulation of existing records of a database. It is responsible for all changes that occur in the database. The changes made in the database using this command can't save permanently because its commands are not auto-committed. Therefore, changes can be rollback. The following commands come under DML language:

- **SELECT**: This command is used to extract information from a table.
- **INSERT**: It is a SQL query that allows us to add data into a table's row.
- **UPDATE**: This command is used to alter or modify the contents of a table.
- **DELETE**: This command is used to delete records from a database table, either individually or in groups.

Why we use DML commands?

The following are the reasons to use the DML commands:

- It helps users to change the data in a database table.
- It helps users to specify what data is needed.
- It facilitates human interaction with the system.

Key Differences between DDL and DML Commands

The following points explain the main differences between DDL and DML commands:

- Data Definition Language (DDL) statements describe the structure of a database or schema. Data Manipulation Language (DML) statements, on the other hand, allow altering data that already exists in the database.
- We use the DDL commands for creating the database or schema, while DML commands are used to populate and manipulate the database.
- DDL commands can affect the whole database or table, whereas DML statements only affect single or multiple rows based on the condition specified in a query.
- Since DDL commands are auto-committed, modifications are permanent and cannot be reversed. DML statements, on the other hand, are not auto-committed, which means that modifications are not permanent and can be reversed.
- DML is an imperative and procedural method, whereas DDL is a declarative method.
- The data in DML statements can be filtered with a WHERE clause, while the records in DDL statements cannot be filtered with a WHERE clause.

DDL vs. DML Comparison

The following comparison chart explains their main differences in a quick manner:

Comparison Basis	DDL	DML
Basic	It helps us define a database's structure or schema and deals with how data is stored in the database.	It allows us to manipulate, i.e., retrieve, update, and delete the data stored in the database.

Full-Form	The full form of DDL is Data Definition Language.	The full form of DML is Data Manipulation Language.
Categorization	The DDL commands have no further classification.	The DML commands are classified as procedural and non-procedural (declarative) DMLs.
Command uses	The commonly used commands under DDL language are: o CREATE o DROP o ALTER o TRUNCATE o RENAME	The commonly used commands under DML language are: o INSERT o UPDATE o DELETE o SELECT
Auto-commit	DDL commands are auto-committed, so changes that happen in the database will be permanent.	DML commands are not auto-committed, so database changes are not permanent.
Rollback	DDL commands made changes permanent; therefore, we cannot roll back these statements.	DML commands do not make changes permanent; therefore, rollback is possible for these statements.
WHERE clause	DDL commands have no use of a WHERE clause because here, filtration of records is not possible.	The DML statements can use a WHERE clause while manipulating data in a database.
Effect	The DDL command affects the entire database or table.	The DML commands will affect the single or multiple records based on the specified condition.

UNIT-3

SQL

- SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a number of ways, using English-like statements.

Rules:

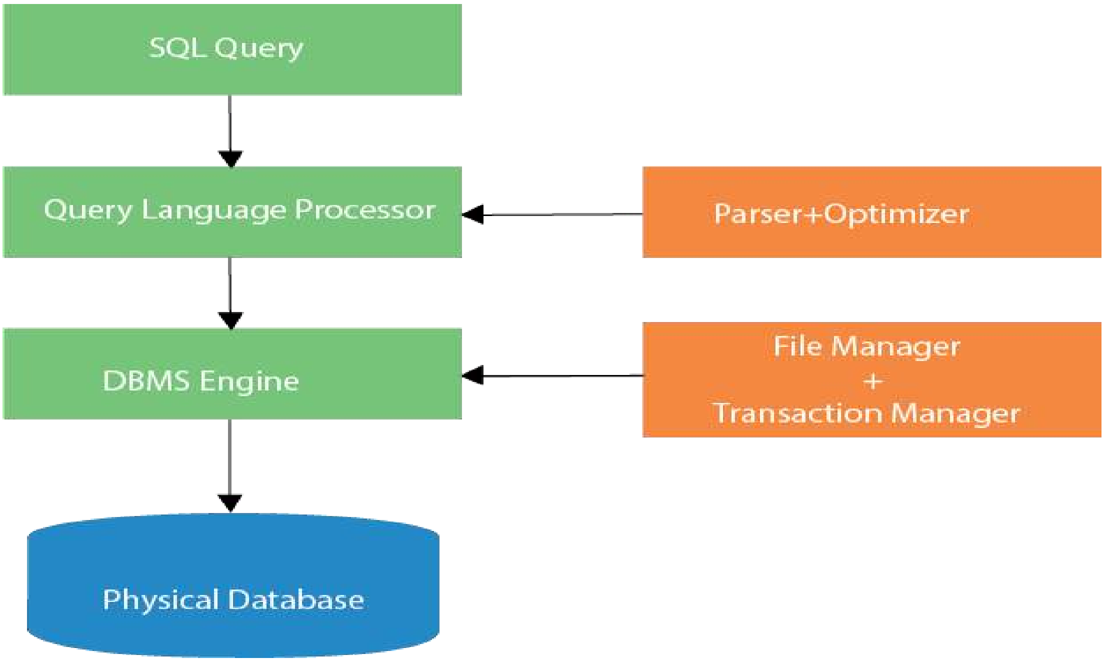
SQL follows the following rules:

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

SQL process:

- When an SQL command is executing for any RDBMS, then the system figure out the best way to carry out the request and the SQL engine determines that how to interpret the task.
- In the process, various components are included. These components can be optimization Engine, Query engine, Query dispatcher, classic, etc.

- All the non-SQL queries are handled by the classic query engine, but SQL query engine won't handle logical files.



Characteristics of SQL

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

Advantages of SQL

- There are the following advantages of SQL:
- High speed
- Using the SQL queries, the user can quickly and efficiently retrieve a large amount of records from a database.
- No coding needed
- In the standard SQL, it is very easy to manage the database system. It doesn't require a substantial amount of code to manage the database system.
- Well defined standards
- Long established are used by the SQL databases that are being used by ISO and ANSI.

Portability

SQL can be used in laptop, PCs, server and even some mobile phones.

Interactive language

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

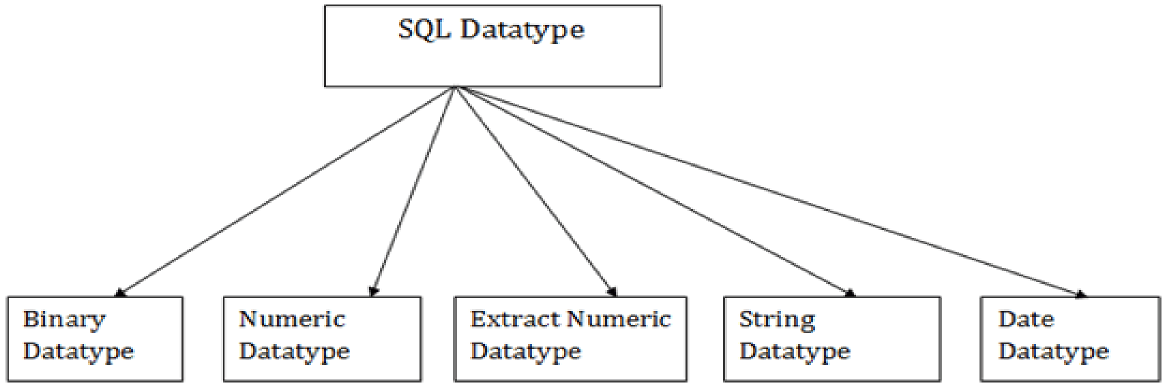
Multiple data view

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

SQL Datatype

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

Datatype of SQL:



1. Binary Data types

There are Three types of binary Datatypes which are given below:

Data Type	Description
binary	It has a maximum length of 8000 bytes. It contains fixed-length binary data.
varbinary	It has a maximum length of 8000 bytes. It contains variable-length binary data.
image	It has a maximum length of 2,147,483,647 bytes. It contains variable-length binary data.

2. Approximate Numeric Data type:

The subtypes are given below:

Data type	From	To	Description
float	-1.79E + 308	1.79E + 308	It is used to specify a floating-point value e.g. 6.2, 2.9 etc.
real	-3.40e + 38	3.40E + 38	It specifies a single precision floating point number

3. Exact Numeric Data type

The subtypes are given below:

Data type	Description
int	It is used to specify an integer value.
smallint	It is used to specify small integer value.
bit	It has the number of bits to store.
decimal	It specifies a numeric value that can have a decimal number.
numeric	It is used to specify a numeric value.

4. Character String Data type

The subtypes are given below:

Data type	Description
char	It has a maximum length of 8000 characters. It contains Fixed-length non-unicode characters.
varchar	It has a maximum length of 8000 characters. It contains variable-length non-unicode characters.
text	It has a maximum length of 2,147,483,647 characters. It contains variable-length non-unicode characters.

5. Date and time Data types

The subtypes are given below:

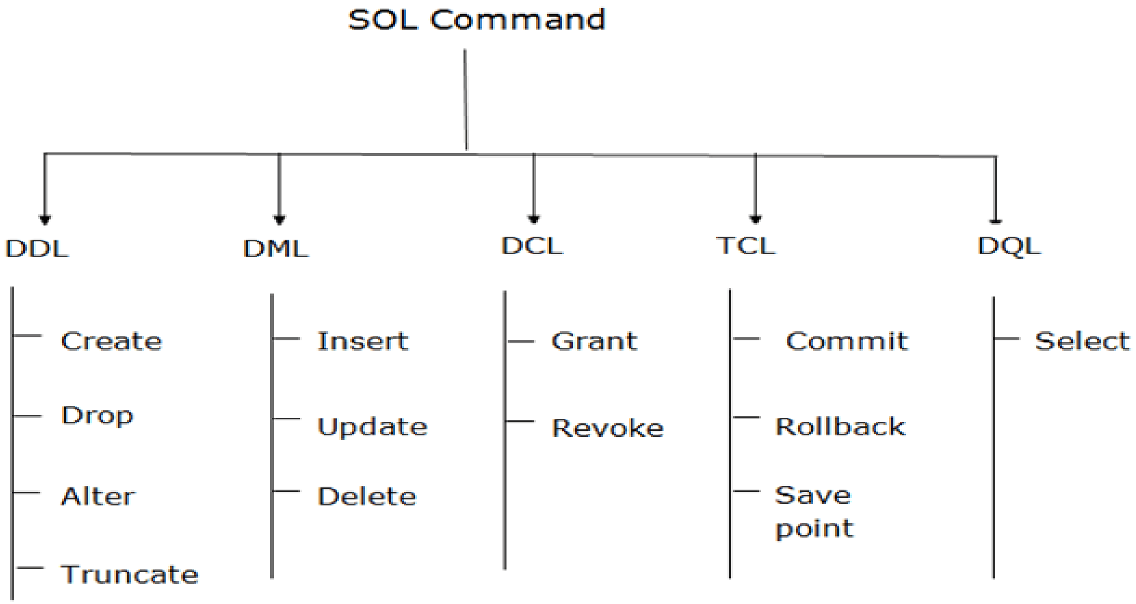
Datatype	Description
date	It is used to store the year, month, and days value.
time	It is used to store the hour, minute, and second values.
timestamp	It stores the year, month, day, hour, minute, and the second value.

SQL Commands

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

Types of SQL Commands

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



1. Data Definition Language (DDL)

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

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

a. CREATE It is used to create a new table in the database.

Syntax:

CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);

Example:

CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DAT E);

b. DROP: It is used to delete both the structure and record stored in the table.

Syntax

DROP TABLE table_name;

Example

DROP TABLE EMPLOYEE;

c. ALTER: It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

Syntax:

To add a new column in the table

ALTER TABLE table_name ADD column_name COLUMN-definition;

To modify existing column in the table:

ALTER TABLE table_name MODIFY(column_definitions....);

EXAMPLE

```
ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));
ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));
```

d. TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.

Syntax:

```
TRUNCATE TABLE table_name;
```

Example:

```
TRUNCATE TABLE EMPLOYEE;
```

2. Data Manipulation Language

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

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

a. INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

Syntax:

```
INSERT INTO TABLE_NAME
(col1, col2, col3,.... col N)
VALUES (value1, value2, value3, .... valueN);
```

Or

```
INSERT INTO TABLE_NAME
VALUES (value1, value2, value3, .... valueN);
```

For example:

```
INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");
```

b. UPDATE: This command is used to update or modify the value of a column in the table.

Syntax:

```
UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CO
NDITION]
```

For example:

```
UPDATE students
SET User_Name = 'Sonoo'
WHERE Student_Id = '3'
```

c. DELETE: It is used to remove one or more row from a table.

Syntax:

```
DELETE FROM table_name [WHERE condition];
```

For example:

```
DELETE FROM javatpoint
WHERE Author="Sonoo";
```

3. Data Control Language

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

Here are some commands that come under DCL:

- Grant
- Revoke

a. Grant: It is used to give user access privileges to a database.

Example

GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

b. Revoke: It is used to take back permissions from the user.

Example

REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

4. Transaction Control Language

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

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

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

a. Commit: Commit command is used to save all the transactions to the database.

Syntax:
COMMIT;

Example:

```
DELETE FROM CUSTOMERS
WHERE AGE = 25;
COMMIT;
```

b. Rollback: Rollback command is used to undo transactions that have not already been saved to the database.

Syntax:

```
ROLLBACK;
```

Example:

```
DELETE FROM CUSTOMERS
WHERE AGE = 25;
ROLLBACK;
```

c. SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:

```
SAVEPOINT SAVEPOINT_NAME;
```

5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

a. SELECT: This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Syntax:

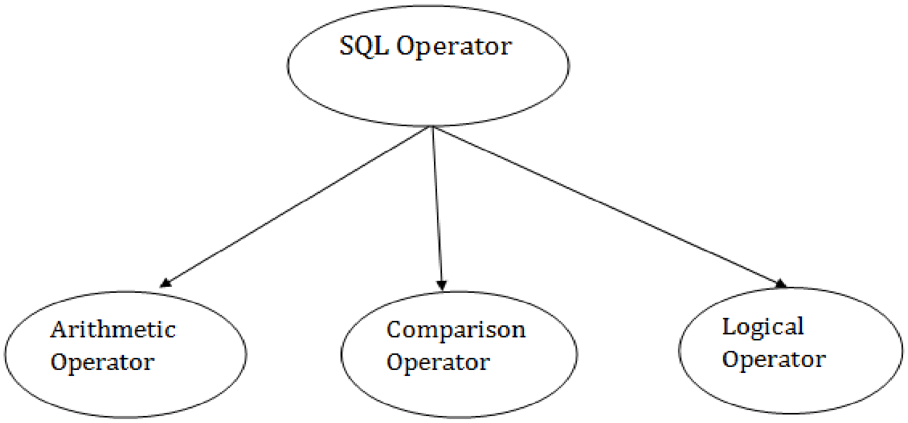
```
SELECT expressions
FROM TABLES
WHERE conditions;
```

For example:

```
SELECT emp_name
FROM employee
WHERE age > 20;
```

SQL Operator

There are various types of SQL operator:



SQL Arithmetic Operators

Let's assume 'variable a' and 'variable b'. Here, 'a' contains 20 and 'b' contains 10.

Operator	Description	Example
+	It adds the value of both operands.	a+b will give 30

-	It is used to subtract the right-hand operand from the left-hand operand.	a-b will give 10
*	It is used to multiply the value of both operands.	a*b will give 200
/	It is used to divide the left-hand operand by the right-hand operand.	a/b will give 2
%	It is used to divide the left-hand operand by the right-hand operand and returns reminder.	a%b will give 0

SQL Comparison Operators:

Let's assume 'variable a' and 'variable b'. Here, 'a' contains 20 and 'b' contains 10.

Operator	Description	Example
=	It checks if two operands values are equal or not, if the values are queal then condition becomes true.	(a=b) is not true
!=	It checks if two operands values are equal or not, if values are not equal, then condition becomes true.	(a!=b) is true
<>	It checks if two operands values are equal or not, if values are not equal then condition becomes true.	(a<>b) is true
>	It checks if the left operand value is greater than right operand value, if yes then condition becomes true.	(a>b) is not true
<	It checks if the left operand value is less than right operand value, if yes then condition becomes true.	(a<b) is true
>=	It checks if the left operand value is greater than or equal to the right operand value, if yes then condition becomes true.	(a>=b) is not true

<=	It checks if the left operand value is less than or equal to the right operand value, if yes then condition becomes true.	(a<=b) is true
!<	It checks if the left operand value is not less than the right operand value, if yes then condition becomes true.	(a!=b) is not true
!>	It checks if the left operand value is not greater than the right operand value, if yes then condition becomes true.	(a!>b) is true

SQL Logical Operators

There is the list of logical operator used in SQL:

Operator	Description
ALL	It compares a value to all values in another value set.
AND	It allows the existence of multiple conditions in an SQL statement.
ANY	It compares the values in the list according to the condition.
BETWEEN	It is used to search for values that are within a set of values.
IN	It compares a value to that specified list value.
NOT	It reverses the meaning of any logical operator.
OR	It combines multiple conditions in SQL statements.
EXISTS	It is used to search for the presence of a row in a specified table.
LIKE	It compares a value to similar values using wildcard operator.

SQL Table

- SQL Table is a collection of data which is organized in terms of rows and columns. In DBMS, the table is known as relation and row as a tuple.
- Table is a simple form of data storage. A table is also considered as a convenient representation of relations.

Let's see an example of the **EMPLOYEE** table:

EMP_ID	**EMP_NAME**	**CITY**	**PHONE_NO**
1	Kristen	Washington	7289201223
2	Anna	Franklin	9378282882
3	Jackson	Bristol	9264783838
4	Kellan	California	7254728346
5	Ashley	Hawaii	9638482678

In the above table, "EMPLOYEE" is the table name, "EMP_ID", "EMP_NAME", "CITY", "PHONE_NO" are the column names. The combination of data of multiple columns forms a row, e.g., 1, "Kristen", "Washington" and 7289201223 are the data of one row.

Operation on Table

1. Create table
2. Drop table
3. Delete table
4. Rename table

SQL Create Table

SQL create table is used to create a table in the database. To define the table, you should define the name of the table and also define its columns and column's data type.

Syntax

```
create table "table_name"
("column1" "data type",
"column2" "data type",
"column3" "data type",
...
"columnN" "data type");
```

Example

```
SQL> CREATE TABLE EMPLOYEE (
EMP_ID INT NOT NULL,
EMP_NAME VARCHAR (25) NOT NULL,
PHONE_NO INT NOT NULL,
ADDRESS CHAR (30),
PRIMARY KEY (ID)
);
```

If you create the table successfully, you can verify the table by looking at the message by the SQL server. Else you can use DESC command as follows:

```
SQL> DESC EMPLOYEE;
```

Field	Type	Null	Key	Default	Extra
EMP_ID	int(11)	NO	PRI	NULL	

EMP_NAME	varchar(25)	NO		NULL	
PHONE_NO	NO	int(11)		NULL	
ADDRESS	YES			NULL	char(30)

- 4 rows in set (0.35 sec)

Now you have an EMPLOYEE table in the database, and you can use the stored information related to the employees.

Drop table

A SQL drop table is used to delete a table definition and all the data from a table. When this command is executed, all the information available in the table is lost forever, so you have to very careful while using this command.

Syntax

1. DROP TABLE "table_name";

Firstly, you need to verify the **EMPLOYEE** table using the following command:

1. SQL> DESC EMPLOYEE;

Field	**Type**	**Null**	**Key**	**Default**	**Extra**
EMP_ID	int(11)	NO	PRI	NULL	
EMP_NAME	varchar(25)	NO		NULL	
PHONE_NO	NO	int(11)		NULL	
ADDRESS	YES			NULL	char(30)

- 4 rows in set (0.35 sec)

This table shows that EMPLOYEE table is available in the database, so we can drop it as follows:

```
SQL>DROP TABLE EMPLOYEE;
```

Now, we can check whether the table exists or not using the following command:

```
Query OK, 0 rows affected (0.01 sec)
```

As this shows that the table is dropped, so it doesn't display it.

SQL DELETE table

In SQL, DELETE statement is used to delete rows from a table. We can use WHERE condition to delete a specific row from a table. If you want to delete all the records from the table, then you don't need to use the WHERE clause.

Syntax

```
DELETE FROM table_name WHERE condition;
```

Example

Suppose, the EMPLOYEE table having the following records:

EMP_ID	EMP_NAME	CITY	PHONE_NO	SALARY
1	Kristen	Chicago	9737287378	150000
2	Russell	Austin	9262738271	200000
3	Denzel	Boston	7353662627	100000
4	Angelina	Denver	9232673822	600000

5	Robert	Washington	9367238263	350000
6	Christian	Los angels	7253847382	260000

The following query will DELETE an employee whose ID is 2.

```
SQL> DELETE FROM EMPLOYEE
WHERE EMP_ID = 3;
```

Now, the EMPLOYEE table would have the following records.

EMP_ID	EMP_NAME	CITY	PHONE_NO	SALARY
1	Kristen	Chicago	9737287378	150000
2	Russell	Austin	9262738271	200000
4	Angelina	Denver	9232673822	600000
5	Robert	Washington	9367238263	350000
6	Christian	Los angels	7253847382	260000

If you don't specify the WHERE condition, it will remove all the rows from the table.

```
DELETE FROM EMPLOYEE;
```

Now, the EMPLOYEE table would not have any records.

SQL SELECT Statement

In SQL, the SELECT statement is used to query or retrieve data from a table in the database. The returns data is stored in a table, and the result table is known as result-set.

Syntax

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

Here, the expression is the field name of the table that you want to select data from.

Use the following syntax to select all the fields available in the table:

```
SELECT  *  FROM table_name;
```

Example:

EMPLOYEE

EMP_ID	EMP_NAME	CITY	PHONE_NO	SALARY
1	Kristen	Chicago	9737287378	150000
2	Russell	Austin	9262738271	200000
3	Angelina	Denver	9232673822	600000
4	Robert	Washington	9367238263	350000
5	Christian	Los angels	7253847382	260000

To fetch the EMP_ID of all the employees, use the following query:

```
SELECT EMP_ID FROM EMPLOYEE;
```

Output

EMP_ID
1
2
3
4
5

To fetch the EMP_NAME and SALARY, use the following query:

```
SELECT EMP_NAME, SALARY FROM EMPLOYEE;
```

EMP_NAME	SALARY
Kristen	150000
Russell	200000
Angelina	600000
Robert	350000
Christian	260000

To fetch all the fields from the EMPLOYEE table, use the following query:

```
SELECT * FROM EMPLOYEE
```

Output

EMP_ID	EMP_NAME	CITY	PHONE_NO	SALARY
1	Kristen	Chicago	9737287378	150000
2	Russell	Austin	9262738271	200000
3	Angelina	Denver	9232673822	600000
4	Robert	Washington	9367238263	350000
5	Christian	Los angels	7253847382	260000

SQL INSERT Statement

The SQL INSERT statement is used to insert a single or multiple data in a table. In SQL, You can insert the data in two ways:

1. Without specifying column name
2. By specifying column name

Sample Table

EMPLOYEE

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30

2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36

1. Without specifying column name

If you want to specify all column values, you can specify or ignore the column values.

Syntax

```
INSERT INTO TABLE_NAME
VALUES (value1, value2, value 3, .... Value N);
```

Query

```
INSERT INTO EMPLOYEE VALUES (6, 'Marry', 'Canada', 600000, 48);
```

Output: After executing this query, the EMPLOYEE table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29

5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

2. By specifying column name

To insert partial column values, you must have to specify the column names.

Syntax

INSERT INTO TABLE_NAME
[(col1, col2, col3,.... col N)]
VALUES (value1, value2, value 3, ... Value N);

Query

INSERT INTO EMPLOYEE (EMP_ID, EMP_NAME, AGE) VALUES (7, 'Jack', 40);

Output: After executing this query, the table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48
7	Jack	null	null	40

SQL Update Statement

The SQL UPDATE statement is used to modify the data that is already in the database. The condition in the WHERE clause decides that which row is to be updated.

Syntax

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

Sample Table

EMPLOYEE

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

Updating single record

Update the column EMP_NAME and set the value to 'Emma' in the row where SALARY is 500000.

Syntax

```
UPDATE table_name
SET column_name = value
WHERE condition;
```

Query

```
UPDATE EMPLOYEE
SET EMP_NAME = 'Emma'
WHERE SALARY = 500000;
```

Output: After executing this query, the EMPLOYEE table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Emma	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

Updating multiple records

If you want to update multiple columns, you should separate each field assigned with a comma. In the EMPLOYEE table, update the column EMP_NAME to 'Kevin' and CITY to 'Boston' where EMP_ID is 5.

Syntax

```
UPDATE table_name
SET column_name = value1, column_name2 = value2
WHERE condition;
```

Query

```
UPDATE EMPLOYEE
SET EMP_NAME = 'Kevin', City = 'Boston'
WHERE EMP_ID = 5;
```

Output

EMP_ID	**EMP_NAME**	**CITY**	**SALARY**	**AGE**
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Kevin	Boston	200000	36
6	Marry	Canada	600000	48

Without use of WHERE clause

If you want to update all row from a table, then you don't need to use the WHERE clause. In the EMPLOYEE table, update the column EMP_NAME as 'Harry'.

Syntax

```
UPDATE table_name
SET column_name = value1;
```

Query

```
UPDATE EMPLOYEE
SET EMP_NAME = 'Harry';
```

Output

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Harry	Chicago	200000	30
2	Harry	Austin	300000	26
3	Harry	Denver	100000	42
4	Harry	Washington	500000	29
5	Harry	Los angels	200000	36
6	Harry	Canada	600000	48

SQL DELETE Statement

The SQL DELETE statement is used to delete rows from a table. Generally, DELETE statement removes one or more records form a table.

Syntax

DELETE FROM table_name WHERE some_condition;

Sample Table

EMPLOYEE

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

Deleting Single Record

Delete the row from the table EMPLOYEE where EMP_NAME = 'Kristen'. This will delete only the fourth row.

Query

```
DELETE FROM EMPLOYEE
WHERE EMP_NAME = 'Kristen';
```

Output: After executing this query, the EMPLOYEE table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

Deleting Multiple Record

Delete the row from the EMPLOYEE table where AGE is 30. This will delete two rows(first and third row).

Query

```
DELETE FROM EMPLOYEE WHERE AGE= 30;
```

Output: After executing this query, the EMPLOYEE table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

Delete all of the records

Delete all the row from the EMPLOYEE table. After this, no records left to display. The EMPLOYEE table will become empty.

Syntax

```
DELETE * FROM table_name;
or
DELETE FROM table_name;
```

Query

```
DELETE FROM EMPLOYEE;
```

Output: After executing this query, the EMPLOYEE table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE

Views in SQL

- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database.
- A view can either have specific rows based on certain condition or all the rows of a table.

Sample table:

Student_Detail

STU_ID	**NAME**	**ADDRESS**
1	Stephan	Delhi
2	Kathrin	Noida
3	David	Ghaziabad
4	Alina	Gurugram

Student_Marks

STU_ID	**NAME**	**MARKS**	**AGE**
1	Stephan	97	19
2	Kathrin	86	21
3	David	74	18

4	Alina	90	20
5	John	96	18

1. Creating view

A view can be created using the **CREATE VIEW** statement. We can create a view from a single table or multiple tables.

Syntax:

7.5M

236

Java Tricky Program 16 - Autoboxing, Inheritance and Overriding

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE condition;
```

2. Creating View from a single table

In this example, we create a View named DetailsView from the table Student_Detail.

Query:

```
CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM Student_Details
WHERE STU_ID < 4;
```

Just like table query, we can query the view to view the data.

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Stephan	Delhi
Kathrin	Noida
David	Ghaziabad

3. Creating View from multiple tables

View from multiple tables can be created by simply include multiple tables in the SELECT statement.

In the given example, a view is created named MarksView from two tables Student_Detail and Student_Marks.

Query:

```
CREATE VIEW MarksView AS
SELECT Student_Detail.NAME, Student_Detail.ADDRESS, Student_Marks.MARKS
FROM Student_Detail, Student_Mark
WHERE Student_Detail.NAME = Student_Marks.NAME;
```

To display data of View MarksView:

```
SELECT * FROM MarksView;
```

NAME	ADDRESS	MARKS
Stephan	Delhi	97
Kathrin	Noida	86
David	Ghaziabad	74
Alina	Gurugram	90

4. Deleting View

A view can be deleted using the Drop View statement.

Syntax

```
DROP VIEW view_name;
```

Example:

If we want to delete the View **MarksView**, we can do this as:

```
DROP VIEW MarksView;
```

SQL Set Operation

The SQL Set operation is used to combine the two or more SQL SELECT statements.

Types of Set Operation

1. Union
2. Union All
3. Intersect
4. Minus

1. Union

- The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
- In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
- The union operation eliminates the duplicate rows from its resultset.

Syntax

```
SELECT column_name FROM table1
UNION
SELECT column_name FROM table2;
```

Example:

The First table

ID	NAME
1	Jack
2	Harry
3	Jackson

The Second table

ID	NAME
3	Jackson
4	Stephan
5	David

Union SQL query will be:

```
SELECT * FROM First
UNION
SELECT * FROM Second;
```

The resultset table will look like:

ID	NAME
1	Jack
2	Harry
3	Jackson
4	Stephan
5	David

2. Union All

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Syntax:

```
SELECT column_name FROM table1
UNION ALL
SELECT column_name FROM table2;
```

Example: Using the above First and Second table.

Union All query will be like:

```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

The resultset table will look like:

ID	NAME
1	Jack
2	Harry
3	Jackson
3	Jackson
4	Stephan
5	David

3. Intersect

- It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
- In the Intersect operation, the number of datatype and columns must be the same.
- It has no duplicates and it arranges the data in ascending order by default.

Syntax

SELECT column_name FROM table1

INTERSECT

SELECT column_name FROM table2;

Example:

Using the above First and Second table.

Intersect query will be:

SELECT * FROM First

INTERSECT

SELECT * FROM Second;

The resultset table will look like:

ID	NAME
3	Jackson

4. Minus

- It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates and data arranged in ascending order by default.

Syntax:

SELECT column_name FROM table1

MINUS

SELECT column_name FROM table2;

Example

Using the above First and Second table.

Minus query will be:

SELECT * FROM First

MINUS

SELECT * FROM Second;

The resultset table will look like:

ID	NAME
1	Jack
2	Harry

SQL Sub Query

A Subquery is a query within another SQL query and embedded within the WHERE clause.

Important Rule:

- A subquery can be placed in a number of SQL clauses like WHERE clause, FROM clause, HAVING clause.
- You can use Subquery with SELECT, UPDATE, INSERT, DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
- A subquery is a query within another query. The outer query is known as the main query, and the inner query is known as a subquery.
- Subqueries are on the right side of the comparison operator.
- A subquery is enclosed in parentheses.
- In the Subquery, ORDER BY command cannot be used. But GROUP BY command can be used to perform the same function as ORDER BY command.

1. **Subqueries with the Select Statement**

SQL subqueries are most frequently used with the Select statement.

Syntax

```
SELECT column_name
FROM table_name
WHERE column_name expression operator
( SELECT column_name  from table_name WHERE ... );
```

Example

Consider the EMPLOYEE table have the following records:

ID	NAME	AGE	ADDRESS	SALARY

1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
4	Alina	29	UK	6500.00
5	Kathrin	34	Bangalore	8500.00
6	Harry	42	China	4500.00
7	Jackson	25	Mizoram	10000.00

The subquery with a SELECT statement will be:

```
SELECT *
FROM EMPLOYEE
WHERE ID IN (SELECT ID
FROM EMPLOYEE
WHERE SALARY > 4500);
```

This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
4	Alina	29	UK	6500.00
5	Kathrin	34	Bangalore	8500.00
7	Jackson	25	Mizoram	10000.00

2. Subqueries with the INSERT Statement

- SQL subquery can also be used with the Insert statement. In the insert statement, data returned from the subquery is used to insert into another table.
- In the subquery, the selected data can be modified with any of the character, date functions.

Syntax:

```
INSERT INTO table_name (column1, column2, column3....)
SELECT *
FROM table_name
WHERE VALUE OPERATOR
```

Example

Consider a table EMPLOYEE_BKP with similar as EMPLOYEE.

Now use the following syntax to copy the complete EMPLOYEE table into the EMPLOYEE_BKP table.

```
INSERT INTO EMPLOYEE_BKP
SELECT * FROM EMPLOYEE
WHERE ID IN (SELECT ID
FROM EMPLOYEE);
```

3. Subqueries with the UPDATE Statement

The subquery of SQL can be used in conjunction with the Update statement. When a subquery is used with the Update statement, then either single or multiple columns in a table can be updated.

Syntax

```
UPDATE table
SET column_name = new_value
WHERE VALUE OPERATOR
(SELECT COLUMN_NAME
FROM TABLE_NAME
WHERE condition);
```

Example

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table. The given example updates the SALARY by .25 times in the EMPLOYEE table for all employee whose AGE is greater than or equal to 29.

```
UPDATE EMPLOYEE
```

```
SET SALARY = SALARY * 0.25
WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
WHERE AGE >= 29);
```

This would impact three rows, and finally, the EMPLOYEE table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
4	Alina	29	UK	1625.00
5	Kathrin	34	Bangalore	2125.00
6	Harry	42	China	1125.00
7	Jackson	25	Mizoram	10000.00

4. Subqueries with the DELETE Statement

The subquery of SQL can be used in conjunction with the Delete statement just like any other statements mentioned above.

Syntax

```
DELETE FROM TABLE_NAME
WHERE VALUE OPERATOR
(SELECT COLUMN_NAME
FROM TABLE_NAME
WHERE condition);
```

Example

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table. The given example deletes the records from the EMPLOYEE table for all EMPLOYEE whose AGE is greater than or equal to 29.

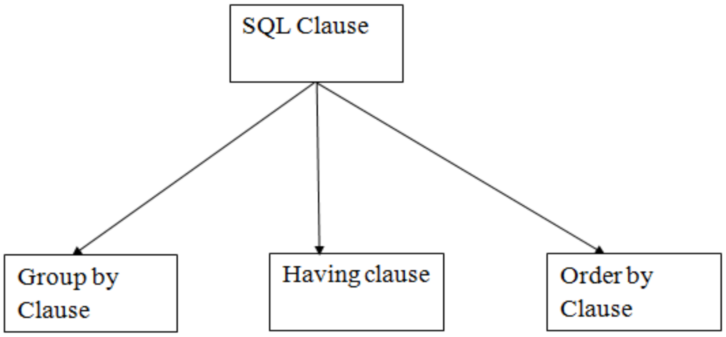
```
DELETE FROM EMPLOYEE
WHERE AGE IN (SELECT AGE FROM EMPLOYEE_BKP
WHERE AGE >= 29 );
```

This would impact three rows, and finally, the EMPLOYEE table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
7	Jackson	25	Mizoram	10000.00

SQL Clauses

The following are the various SQL clauses:



1. GROUP BY

- SQL GROUP BY statement is used to arrange identical data into groups. The GROUP BY statement is used with the SQL SELECT statement.
- The GROUP BY statement follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.
- The GROUP BY statement is used with aggregation function.

Syntax

```
SELECT column
FROM table_name
WHERE conditions
GROUP BY column
ORDER BY column
```

Sample table:

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30

Item9	Com2	2	25	50
Item10	Com3	4	30	120

Example:

```
SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP BY COMPANY;
```

Output:

```
Com1   5
Com2   3
Com3   2
```

2. HAVING

- HAVING clause is used to specify a search condition for a group or an aggregate.
- Having is used in a GROUP BY clause. If you are not using GROUP BY clause then you can use HAVING function like a WHERE clause.

Syntax:

```
SELECT column1, column2
FROM table_name
WHERE conditions
GROUP BY column1, column2
HAVING conditions
ORDER BY column1, column2;
```

Example:

```
SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP BY COMPANY
HAVING COUNT(*)>2;
```

Output:

```
Com1   5
```

3. ORDER BY

- The ORDER BY clause sorts the result-set in ascending or descending order.
- It sorts the records in ascending order by default. DESC keyword is used to sort the records in descending order.

Syntax:

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

Where

ASC: It is used to sort the result set in ascending order by expression.

DESC: It sorts the result set in descending order by expression.

Example: Sorting Results in Ascending Order

Table:

CUSTOMER

CUSTOMER_ID	NAME	ADDRESS
12	Kathrin	US
23	David	Bangkok
34	Alina	Dubai
45	John	UK
56	Harry	US

Enter the following SQL statement:

```
SELECT *
FROM CUSTOMER
ORDER BY NAME;
```

Output:

CUSTOMER_ID	NAME	ADDRESS
34	Alina	Dubai
23	David	Bangkok
56	Harry	US
45	John	UK
12	Kathrin	US

Example: Sorting Results in Descending Order

Using the above CUSTOMER table

```
SELECT *
FROM CUSTOMER
ORDER BY NAME DESC;
```

Output:

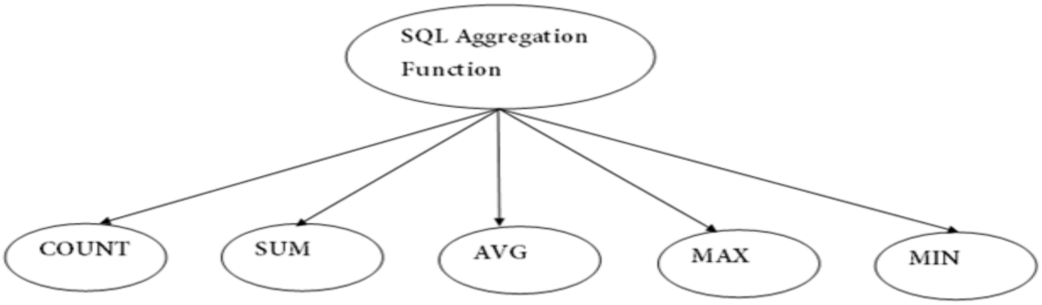
CUSTOMER_ID	NAME	ADDRESS

12	Kathrin	US
45	John	UK
56	Harry	US
23	David	Bangkok
34	Alina	Dubai

SQL Aggregate Functions

- SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
- It is also used to summarize the data.

Types of SQL Aggregation Function



1. COUNT FUNCTION

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

Syntax

COUNT(*)

or

COUNT([ALL|DISTINCT] expression)

Sample table:

PRODUCT_MAST

PRODUCT	**COMPANY**	**QTY**	**RATE**	**COST**
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Example: COUNT()

SELECT COUNT(*)

FROM PRODUCT_MAST;

Output:

```
10
```

Example: COUNT with WHERE

```
SELECT COUNT(*)
FROM PRODUCT_MAST;
WHERE RATE>=20;
```

Output:

```
7
```

Example: COUNT() with DISTINCT

```
SELECT COUNT(DISTINCT COMPANY)
FROM PRODUCT_MAST;
```

Output:

```
3
```

Example: COUNT() with GROUP BY

```
SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP BY COMPANY;
```

Output:

```
Com1    5
Com2    3
Com3    2
```

Example: COUNT() with HAVING

```
SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP BY COMPANY
HAVING COUNT(*)>2;
```

Output:

```
Com1    5
Com2    3
```

2. SUM Function

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

Syntax

```
SUM()
or
SUM( [ALL|DISTINCT] expression )
```

Example: SUM()

```
SELECT SUM(COST)
FROM PRODUCT_MAST;
```

Example: SUM() with WHERE

```
SELECT SUM(COST)
FROM PRODUCT_MAST
WHERE QTY>3;
```

Output:

```
320
```

Example: SUM() with GROUP BY

```
SELECT SUM(COST)
FROM PRODUCT_MAST
WHERE QTY>3
GROUP BY COMPANY;
```

Output:

```
Com1    150
Com2    170
```

Example: SUM() with HAVING

```
SELECT COMPANY, SUM(COST)
FROM PRODUCT_MAST
GROUP BY COMPANY
HAVING SUM(COST)>=170;
```

Output:

```
Com1    335
Com3    170
```

3. AVG function

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

Syntax

```
AVG()
or
AVG( [ALL|DISTINCT] expression )
```

Example:

```
SELECT AVG(COST)
FROM PRODUCT_MAST;
```

Output:

```
67.00
```

4. MAX Function

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

Syntax

```
MAX()
or
MAX( [ALL|DISTINCT] expression )
```

Example:

```
SELECT MAX(RATE)
FROM PRODUCT_MAST;
```

```
30
```

5. MIN Function

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

Syntax

```
MIN()
or
MIN( [ALL|DISTINCT] expression )
```

Example:

```
SELECT MIN(RATE)
FROM PRODUCT_MAST;
```

Output:

```
10
```

SQL JOIN

As the name shows, JOIN means to combine something. In case of SQL, JOIN means "to combine two or more tables".

In SQL, JOIN clause is used to combine the records from two or more tables in a database.

Types of SQL JOIN

1. INNER JOIN
2. LEFT JOIN
3. RIGHT JOIN
4. FULL JOIN

EMPLOYEE

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30

2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

PROJECT

PROJECT_NO	EMP_ID	DEPARTMENT
101	1	Testing
102	2	Development
103	3	Designing
104	4	Development

1. INNER JOIN

In SQL, INNER JOIN selects records that have matching values in both tables as long as the condition is satisfied. It returns the combination of all rows from both the tables where the condition satisfies.

Syntax

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

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
```

```
INNER JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development

2. LEFT JOIN

The SQL left join returns all the values from left table and the matching values from the right table. If there is no matching join value, it will return NULL.

Syntax

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

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
LEFT JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output

EMP_NAME	DEPARTMENT

Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development
Russell	NULL
Marry	NULL

3. RIGHT JOIN

In SQL, RIGHT JOIN returns all the values from the values from the rows of right table and the matched values from the left table. If there is no matching in both tables, it will return NULL.

Syntax

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

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
RIGHT JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output

EMP_NAME	DEPARTMENT
Angelina	Testing

Robert	Development
Christian	Designing
Kristen	Development

4. FULL JOIN

In SQL, FULL JOIN is the result of a combination of both left and right outer join. Join tables have all the records from both tables. It puts NULL on the place of matches not found.

Syntax

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

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
FULL JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing

Kristen	Development
Russell	NULL
Marry	NULL

NULL VALUES

The SQL **NULL** is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank.

A field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.

Syntax

The basic syntax of **NULL** while creating a table.

```
SQL> CREATE TABLE CUSTOMERS(
ID INT NOT NULL,
NAME VARCHAR (20) NOT NULL,
AGE INT NOT NULL,
ADDRESS CHAR (25) ,
SALARY DECIMAL (18, 2),
PRIMARY KEY (ID)
);
```

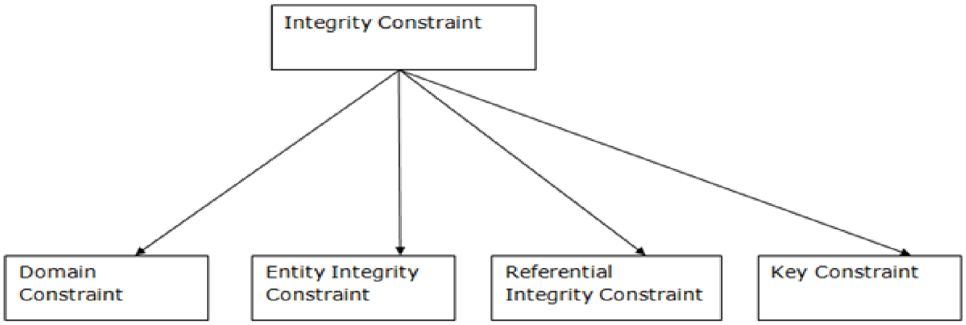
Here, **NOT NULL** signifies that column should always accept an explicit value of the given data type. There are two columns where we did not use NOT NULL, which means these columns could be NULL.

A field with a NULL value is the one that has been left blank during the record creation.

Integrity Constraints

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

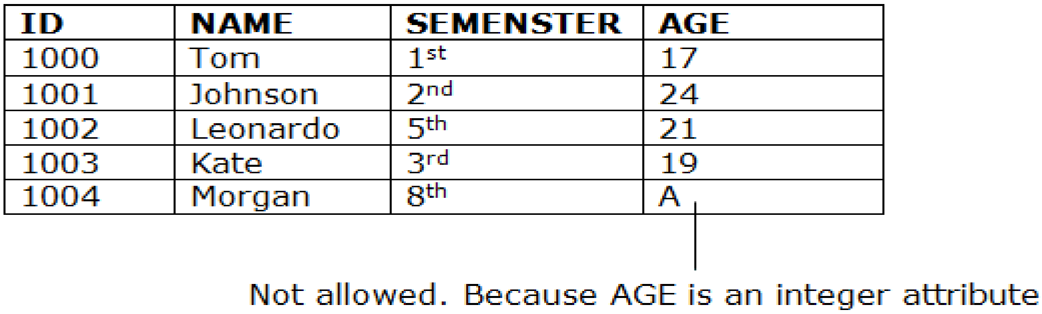
Types of Integrity Constraint



1. Domain constraints

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

Example:



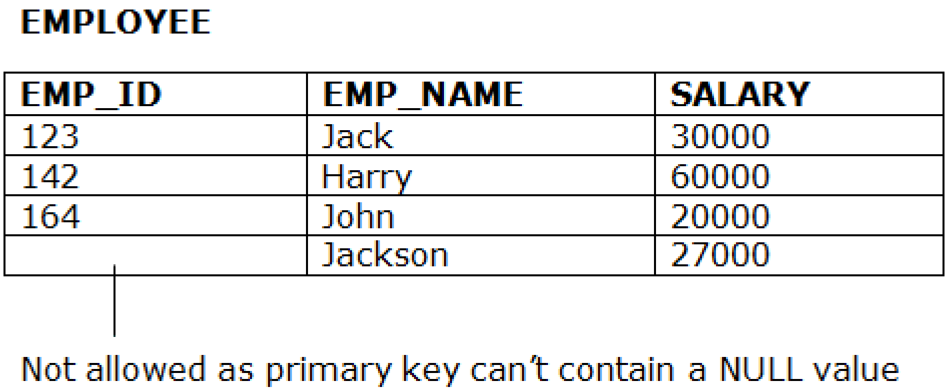
ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

2. Entity integrity constraints

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

Example:



EMPLOYEE

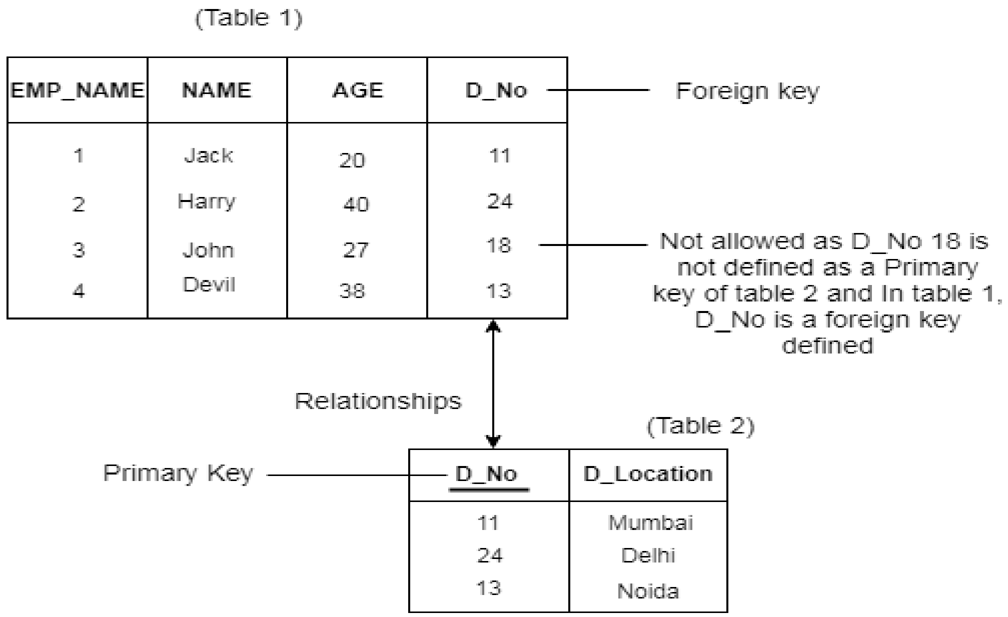
EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

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

3. Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Example



(Table 1)

EMP_NAME	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

Foreign key

Not allowed as D_No 18 is not defined as a Primary key of table 2 and In table 1, D_No is a foreign key defined

Relationships

(Table 2)

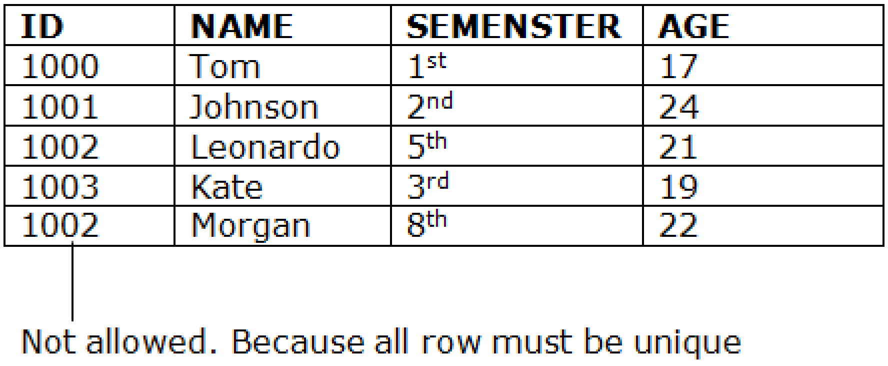
Primary Key

D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

4. Key constraints

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

Example:



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

Not allowed. Because all row must be unique

Trigger

Trigger is invoked by Oracle engine automatically whenever a specified event occurs.Trigger is stored into database and invoked repeatedly, when specific condition match.

Triggers are stored programs, which are automatically executed or fired when some event occurs.

Triggers are written to be executed in response to any of the following events.

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

Advantages of Triggers

These are the following advantages of Triggers:

- Trigger generates some derived column values automatically
- Enforces referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating a trigger:

Syntax for creating trigger:

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
  Declaration-statements
BEGIN
Executable-statements
EXCEPTION
Exception-handling-statements
END;
```

Here,

- CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.

- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

PL/SQL Trigger Example

Let's take a simple example to demonstrate the trigger. In this example, we are using the following CUSTOMERS table:

Create table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Create trigger:

Let's take a program to create a row level trigger for the CUSTOMERS table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
sal_diff number;
BEGIN
  sal_diff := :NEW.salary  - :OLD.salary;
  dbms_output.put_line('Old salary: ' || :OLD.salary);
  dbms_output.put_line('New salary: ' || :NEW.salary);
  dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

After the execution of the above code at SQL Prompt, it produces the following result.

```
Trigger created.
```

Check the salary difference by procedure:

Use the following code to get the old salary, new salary and salary difference after the trigger created.

```
DECLARE
total_rows number(2);
BEGIN
 UPDATE  customers
  SET salary = salary + 5000;
  IF sql%notfound THEN
      dbms_output.put_line('no customers updated');
  ELSIF sql%found THEN
     total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers updated ');
   END IF;
END;
/
```

Output:

```
Old salary: 20000
New salary: 25000
Salary difference: 5000
Old salary: 22000
New salary: 27000
Salary difference: 5000
Old salary: 24000
New salary: 29000
Salary difference: 5000
Old salary: 26000
New salary: 31000
Salary difference: 5000
Old salary: 28000
New salary: 33000
```

```
Salary difference: 5000
Old salary: 30000
New salary: 35000
Salary difference: 5000
6 customers updated
```

Note: As many times you executed this code, the old and new both salary is incremented by 5000 and hence the salary difference is always 5000.

After the execution of above code again, you will get the following result.

```
Old salary: 25000
New salary: 30000
Salary difference: 5000
Old salary: 27000
New salary: 32000
Salary difference: 5000
Old salary: 29000
New salary: 34000
Salary difference: 5000
Old salary: 31000
New salary: 36000
Salary difference: 5000
Old salary: 33000
New salary: 38000
Salary difference: 5000
Old salary: 35000
New salary: 40000
Salary difference: 5000
6 customers updated
```

Important Points

Following are the two very important point and should be noted carefully.

- OLD and NEW references are used for record level triggers these are not avialable for table level triggers.
- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.

UNIT – III

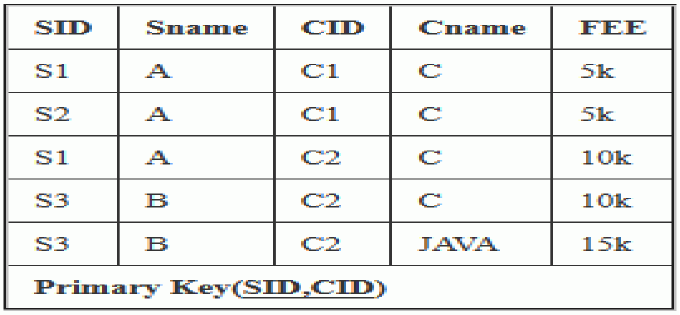
Schema Refinement

The Schema Refinement refers to refine the schema by using some technique. The best technique of schema refinement is decomposition.

Normalisation or Schema Refinement is a technique of organizing the data in the database. It is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies.

Redundancy refers to repetition of same data or duplicate copies of same data stored in different locations. Anomalies: Anomalies refers to the problems occurred after poorly planned and normalised databases where all the data is stored in one table which is sometimes called a flat file database.

Anomalies or problems facing without normalization(problems due to redundancy) : Anomalies refers to the problems occurred after poorly planned and unnormalised databases where all the data is stored in one table which is sometimes called a flat file database. Let us consider such type of schema –



SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k
Primary Key(SID,CID)				

Here all the data is stored in a single table which causes redundancy of data or say anomalies as SID and Sname are repeated once for same CID . Let us discuss anomalies one by one.

Due to redundancy of data we may get the following problems, those are-

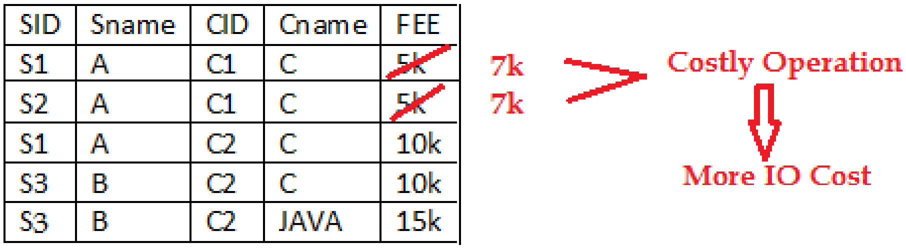
1. **Insertion anomalies :** It may not be possible to store some information unless some otherinformation is stored as well.

2.Redundant storage: some information is stored repeatedly

3.Update anomalies: If one copy of redundant data is updated, then inconsistency is createdunless all redundant copies of data are updated.

4.Deletion anomalies: It may not be possible to delete some information without losing someother information as well.

Problem in updation / updation anomaly – If there is updation in the fee from 5000 to 7000, then we have to update FEE column in all the rows, else data will become inconsistent.



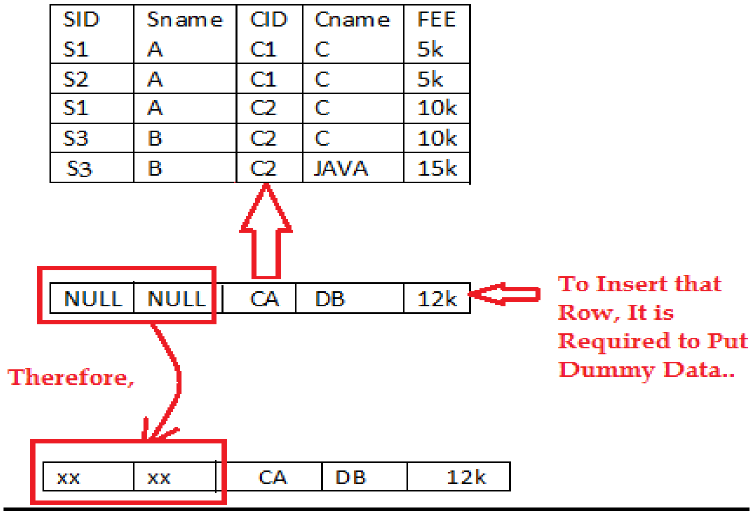
SID	Sname	CID	Cname	FEE
S1	A	C1	C	~~5k~~ 7k
S2	A	C1	C	~~5k~~ 7k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k

Costly Operation

More IO Cost

Insertion Anomaly and Deletion Anomaly- These anomalies exist only due to redundancy, otherwise they do not exist.

InsertionAnomalies: New course is introduced C4, But no student is there who is having C4subject.



SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k

NULL	NULL	CA	DB	12k

To Insert that Row, It is Required to Put Dummy Data..

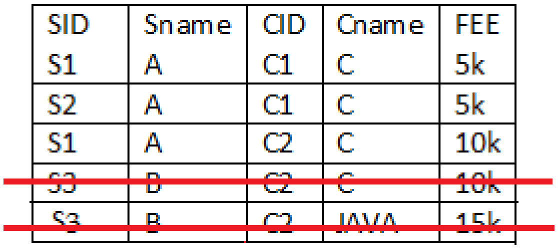
Therefore,

xx	xx	CA	DB	12k

Because of insertion of some data, It is forced to insert some other dummy data.

Deletion Anomaly :

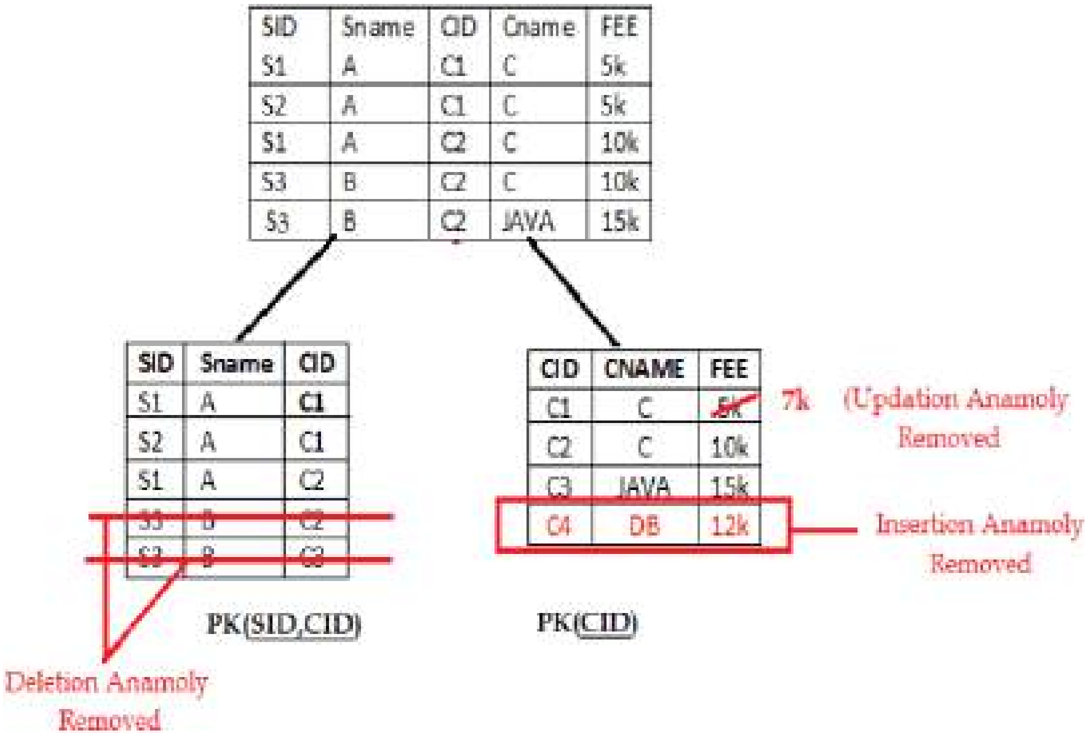
Deletion of S3 student cause the deletion of course.



SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
~~S3~~	~~B~~	~~C2~~	~~C~~	~~10k~~
~~S3~~	~~B~~	~~C2~~	~~JAVA~~	~~15k~~

Because of deletion of some data forced to delete some other useful data.

Solutions To Anomalies : Decomposition of Tables – Schema Refinement

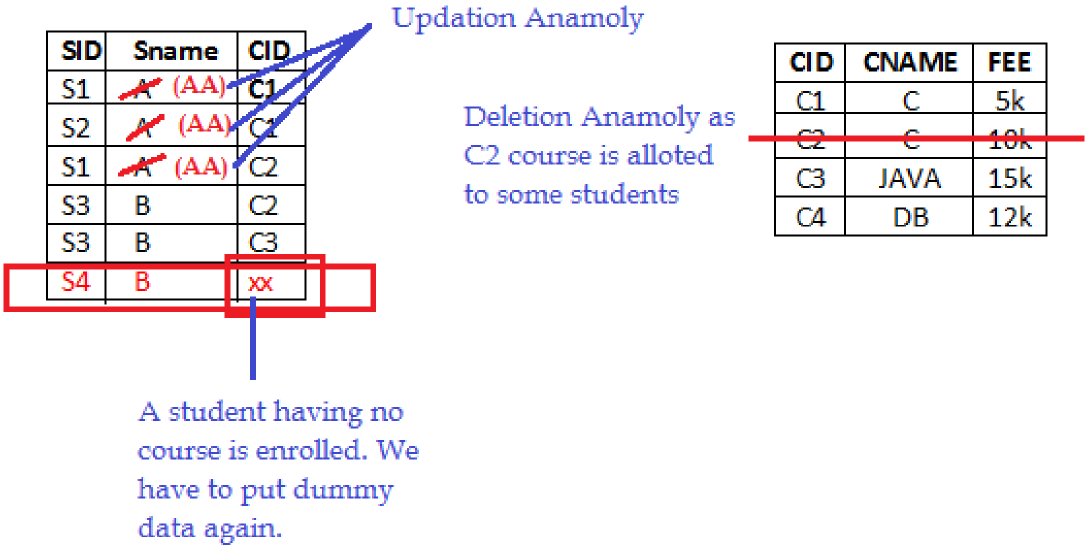


SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k

SID	Sname	CID
S1	A	C1
S2	A	C1
S1	A	C2
~~S3~~	~~B~~	~~C2~~
~~S3~~	~~B~~	~~C3~~

CID	CNAME	FEE
C1	C	~~5k~~
C2	C	10k
C3	JAVA	15k
C4	DB	12k

There are some Anomalies in this again –



SID	Sname	CID
S1	~~A~~ (AA)	C1
S2	~~A~~ (AA)	C1
S1	~~A~~ (AA)	C2
S3	B	C2
S3	B	C3
S4	B	xx

Updation Anamoly

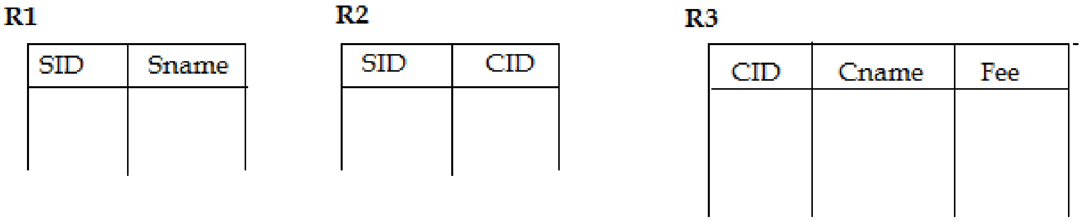
CID	CNAME	FEE
C1	C	5k
~~C2~~	~~C~~	~~10k~~
C3	JAVA	15k
C4	DB	12k

Deletion Anamoly as C2 course is alloted to some students

A student having no course is enrolled. We have to put dummy data again.

What is the Solution ??

Solution : decomposing into relations as shown below



R1

SID	Sname

R2

SID	CID

R3

CID	Cname	Fee

- **TO AVOID REDUNDANCY** and problems due to redundancy, we use refinement techniquecalled **DECOMPOSITION.**

Decomposition: - Process of decomposing a larger relation into smaller relations.

- Each of smaller relations contain subset of attributes of original relation.

Functional Dependency

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

1. X → Y

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example:

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

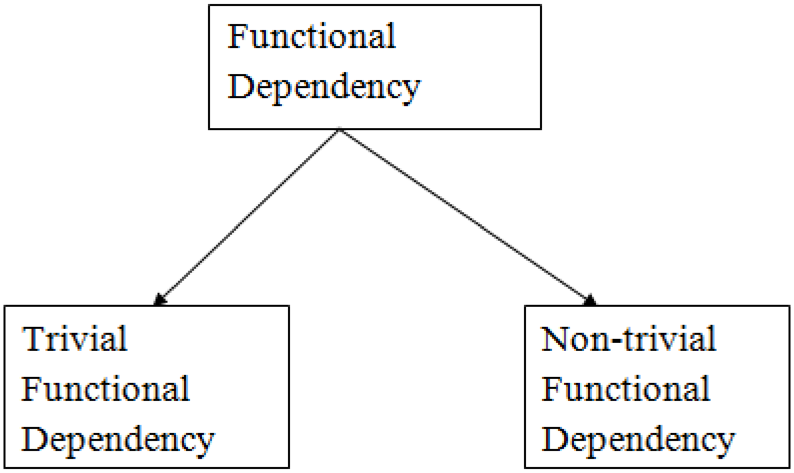
Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

1. Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id.

Types of Functional dependency


Functional Dependency
Trivial Functional Dependency
Non-trivial Functional Dependency

1. Trivial functional dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: A → A, B → B

Example:

1. Consider a table with two columns Employee_Id and Employee_Name.
2. {Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as
3. Employee_Id is a subset of {Employee_Id, Employee_Name}.
4. Also, Employee_Id → Employee_Id and Employee_Name → Employee_Name are trivial de pendencies too.

2. Non-trivial functional dependency

- A → B has a non-trivial functional dependency if B is not a subset of A.
- When A intersection B is NULL, then A → B is called as complete non-trivial.

Example:

1. ID → Name,

2. Name → DOB

Prime and non-prime attributes

Attributes which are parts of any candidate key of relation are called as prime attribute, othersare non-prime attributes.

Candidate Key:

Candidate Key is minimal set of attributes of a relation which can be used to identify a tupleuniquely.

Consider student table: student(sno, sname,sphone,age)

we can take **sno** as candidate key. we can have more than 1 candidate key in a table.types of candidate keys:

1. simple(having only one attribute)
2. composite(having multiple attributes as candidate key)

Super Key:

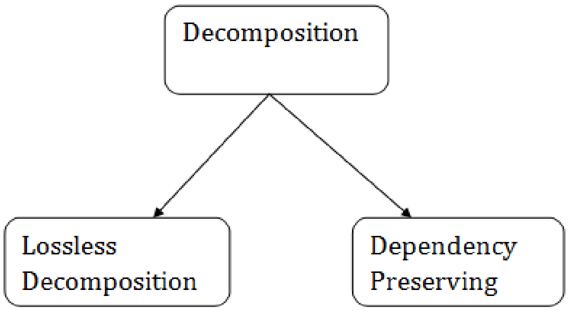
Super Key is set of attributes of a relation which can be used to identify a tuple uniquely.

- Adding zero or more attributes to candidate key generates super key.
- A candidate key is a super key but vice versa is not true.

Relational Decomposition

- When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.
- In a database, it breaks the table into multiple tables.
- If the relation has no proper decomposition, then it may lead to problems like loss of information.
- Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

Types of Decomposition



Lossless Decomposition

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

Example:

EMPLOYEE_DEPARTMENT table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY
22	Denim	28	Mumbai
33	Alina	25	Delhi
46	Stephan	30	Bangalore
52	Katherine	36	Mumbai
60	Jack	40	Noida

DEPARTMENT table

DEPT_ID	EMP_ID	DEPT_NAME
827	22	Sales
438	33	Marketing
869	46	Finance
575	52	Production
678	60	Testing

Now, when these two relations are joined on the common column "EMP_ID", then the resultant relation will look like:

Employee ⋈ Department

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

Hence, the decomposition is Lossless join decomposition.

Dependency Preserving

- It is an important constraint of the database.
- In the dependency preservation, at least one decomposed table must satisfy every dependency.
- If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.
- For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).

Multivalued Dependency

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

Example: Suppose there is a bike manufacturer company which produces two colors(white and black) of each model every year.

BIKE_MODEL	**MANUF_YEAR**	**COLOR**
M2011	2008	White
M2001	2008	Black
M3001	2013	White
M3001	2013	Black
M4006	2017	White
M4006	2017	Black

Here columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other.

In this case, these two columns can be called as multivalued dependent on BIKE_MODEL. The representation of these dependencies is shown below:

1. BIKE_MODEL → → MANUF_YEAR
2. BIKE_MODEL → → COLOR

This can be read as "BIKE_MODEL multidetermined MANUF_YEAR" and "BIKE_MODEL multidetermined COLOR".

Join Dependency

- Join decomposition is a further generalization of Multivalued dependencies.
- If the join of R1 and R2 over C is equal to relation R, then we can say that a join dependency (JD) exists.
- Where R1 and R2 are the decompositions R1(A, B, C) and R2(C, D) of a given relations R (A, B, C, D).
- Alternatively, R1 and R2 are a lossless decomposition of R.

- A JD ⋈ {R1, R2,..., Rn} is said to hold over a relation R if R1, R2,....., Rn is a lossless-join decomposition.
- The *(A, B, C, D), (C, D) will be a JD of R if the join of join's attribute is equal to the relation R.
- Here, *(R1, R2, R3) is used to indicate that relation R1, R2, R3 and so on are a JD of R.

What is Transitive Dependency?

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

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

To achieve 3NF, eliminate the Transitive Dependency.

What is Partial Dependency?

Partial Dependency occurs when a non-prime attribute is functionally dependent on part of a candidate key.

Normalization

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

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

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

What is Normalization?

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

Why do we need Normalization?

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

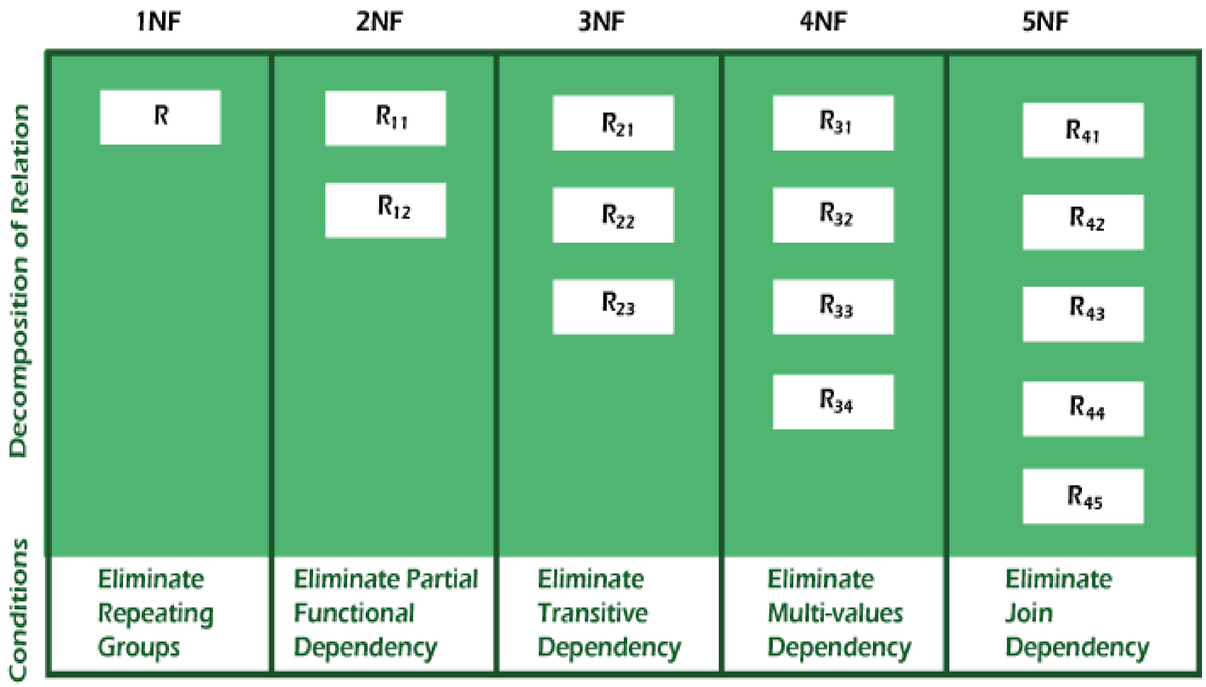
Data modification anomalies can be categorized into three types:

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

Types of Normal Forms:

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

Following are the various types of Normal forms:


1NF
2NF
3NF
4NF
5NF
Decomposition of Relation
Conditions
R
R_{11}
R_{12}
R_{21}
R_{22}
R_{23}
R_{31}
R_{32}
R_{33}
R_{34}
R_{41}
R_{42}
R_{43}
R_{44}
R_{45}
Eliminate Repeating Groups
Eliminate Partial Functional Dependency
Eliminate Transitive Dependency
Eliminate Multi-values Dependency
Eliminate Join Dependency

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

Advantages of Normalization

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

Disadvantages of Normalization

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

First Normal Form (1NF)

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

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

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP

20	Harry	8574783832	Bihar
12	Sam	7390372389,8589830302	Punjab

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

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

Second Normal Form (2NF)

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

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

TEACHER table

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

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

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

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

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

Third Normal Form (3NF)

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

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

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

Example:

EMPLOYEE_DETAIL table:

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

Super key in the table above:

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

Candidate key: {EMP_ID}

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

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

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

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

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

Boyce Codd normal form (BCNF)

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

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

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

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

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

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283

Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

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

Candidate keys:

Forthefirsttable: EMP_ID
Forthesecondtable: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}

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

Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Fifth normal form (5NF)

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).

Example

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

P1

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

P3

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

UNIT-IV

Transaction

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

Example: Suppose an employee of bank transfers Rs 800 from X's account to Y's account. This small transaction contains several low-level tasks:

X's Account

```
Open_Account(X)
Old_Balance = X.balance
New_Balance = Old_Balance - 800
X.balance = New_Balance
Close_Account(X)
```

Y's Account

```
Open_Account(Y)
Old_Balance = Y.balance
New_Balance = Old_Balance + 800
Y.balance = New_Balance
Close_Account(Y)
```

Operations of Transaction:

Following are the main operations of transaction:

Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.

Write(X): Write operation is used to write the value back to the database from the buffer.

Let's take an example to debit transaction from an account which consists of following operations:

1. R(X);
2. X = X - 500;
3. W(X);

Let's assume the value of X before starting of the transaction is 4000.

- The first operation reads X's value from database and stores it in a buffer.
- The second operation will decrease the value of X by 500. So buffer will contain 3500.
- The third operation will write the buffer's value to the database. So X's final value will be 3500.

But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

For example: If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.

To solve this problem, we have two important operations:

Commit: It is used to save the work done permanently.

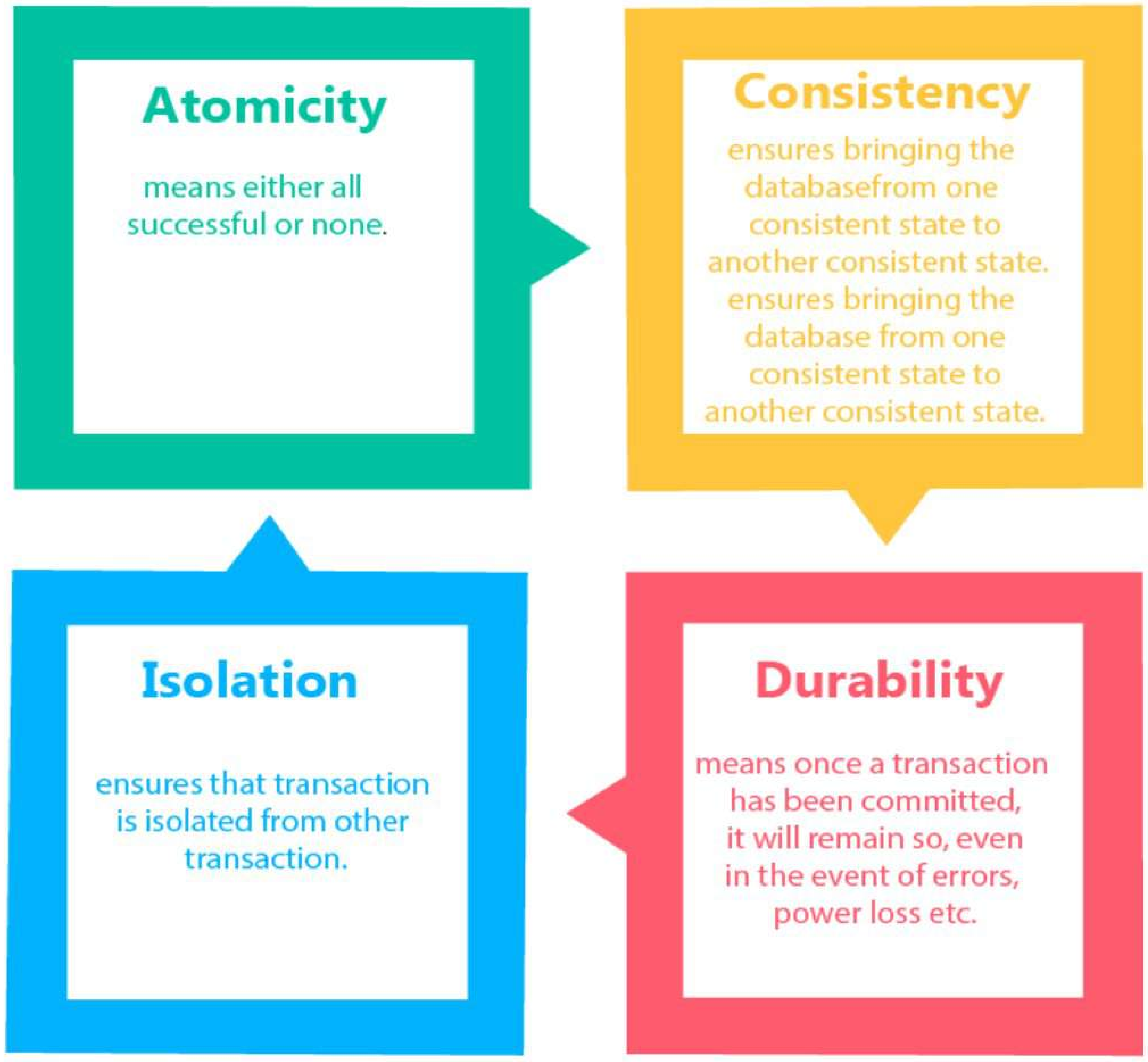
Rollback: It is used to undo the work done.

Transaction property

The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

Property of Transaction

1. Atomicity
2. Consistency
3. Isolation
4. Durability


Atomicity
means either all successful or none.
Consistency
ensures bringing the databasefrom one consistent state to another consistent state. ensures bringing the database from one consistent state to another consistent state.
Isolation
ensures that transaction is isolated from other transaction.
Durability
means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

Atomicity

- It states that all operations of the transaction take place at once if not, the transaction is aborted.
- There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.

Atomicity involves the following two operations:

Abort: If a transaction aborts then all the changes made are not visible.

Commit: If a transaction commits then all the changes made are visible.

Example: Let's assume that following transaction T consisting of T1 and T2. A consists of Rs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B.

T1	T2

Read(A)
A:=A-100
Write(A)
	Read(B)
	Y:=Y+100
	Write(B)

After completion of the transaction, A consists of Rs 500 and B consists of Rs 400.

If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B. This shows the inconsistent database state. In order to ensure correctness of database state, the transaction must be executed in entirety.

Consistency

- The integrity constraints are maintained so that the database is consistent before and after the transaction.
- The execution of a transaction will leave a database in either its prior stable state or a new stable state.
- The consistent property of database states that every transaction sees a consistent database instance.
- The transaction is used to transform the database from one consistent state to another consistent state.

For example: The total amount must be maintained before or after the transaction.

Total before T occurs = 600+300=900
Total after T occurs= 500+400=900

Therefore, the database is consistent. In the case when T1 is completed but T2 fails, then inconsistency will occur.

Isolation

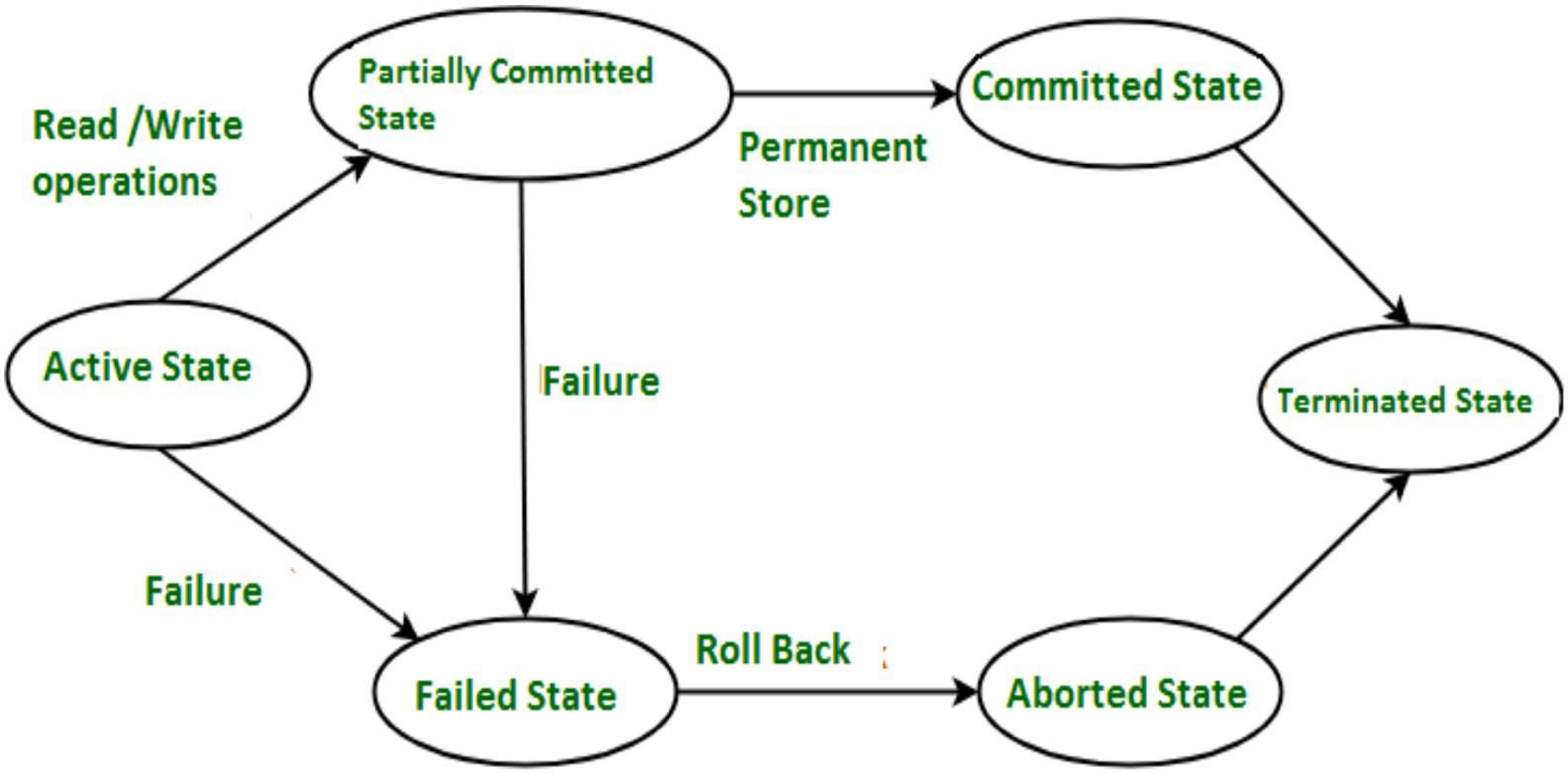
- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforced the isolation property.

Durability

- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.
- They cannot be lost by the erroneous operation of a faulty transaction or by the system failure. When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
- The recovery subsystem of the DBMS has the responsibility of Durability property.

States of Transaction

In a database, the transaction can be in one of the following states -



Transaction States in DBMS

Active state

- The active state is the first state of every transaction. In this state, the transaction is being executed.
- For example: Insertion or deletion or updating a record is done here. But all the records are still not saved to the database.

Partially committed

- In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.
- In the total mark calculation example, a final display of the total marks step is executed in this state.

Committed

A transaction is said to be in a committed state if it executes all its operations successfully. In this state, all the effects are now permanently saved on the database system.

Failed state

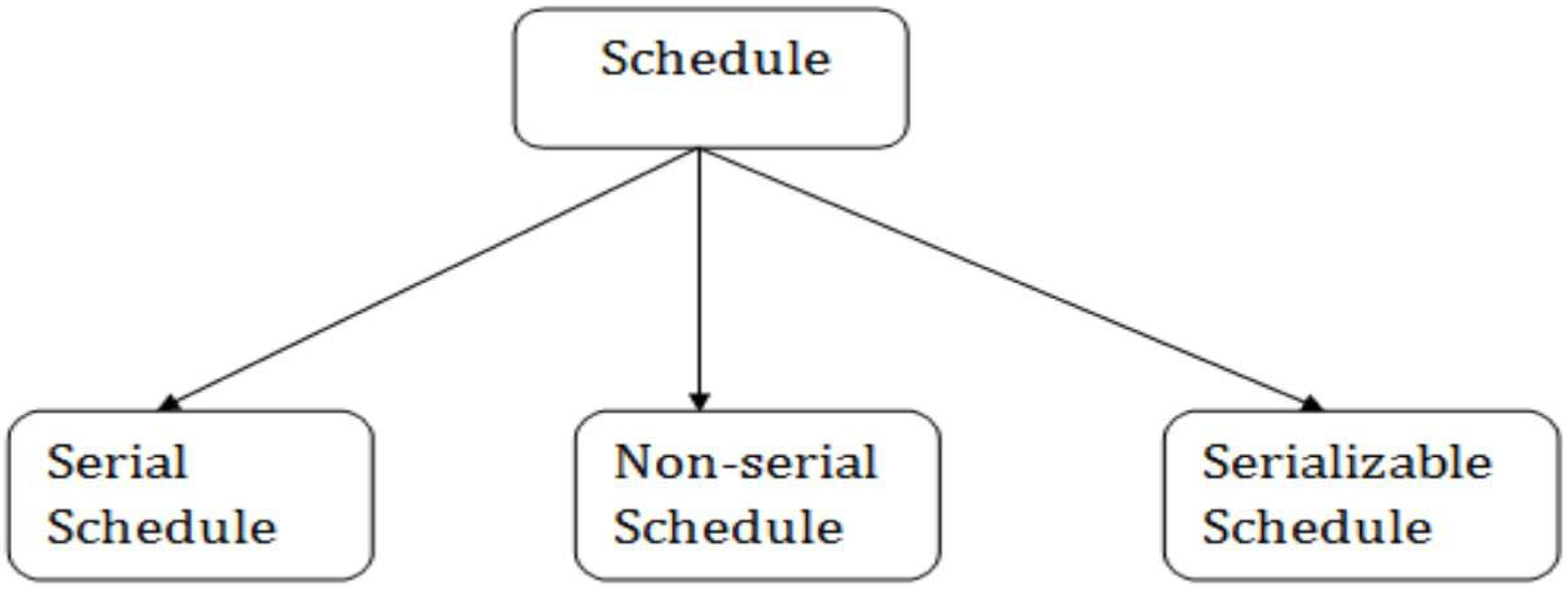
- If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.
- In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.

Aborted

- If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If not then it will abort or roll back the transaction to bring the database into a consistent state.
- If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:
    1. Re-start the transaction
    2. Kill the transaction

Schedule

A series of operation from one transaction to another transaction is known as schedule. It is used to preserve the order of the operation in each of the individual transaction.



1. Serial Schedule

The serial schedule is a type of schedule where one transaction is executed completely before starting another transaction. In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.

For example: Suppose there are two transactions T1 and T2 which have some operations. If it has no interleaving of operations, then there are the following two possible outcomes:

1. Execute all the operations of T1 which was followed by all the operations of T2.
2. Execute all the operations of T1 which was followed by all the operations of T2.

- In the given (a) figure, Schedule A shows the serial schedule where T1 followed by T2.
- In the given (b) figure, Schedule B shows the serial schedule where T2 followed by T1.

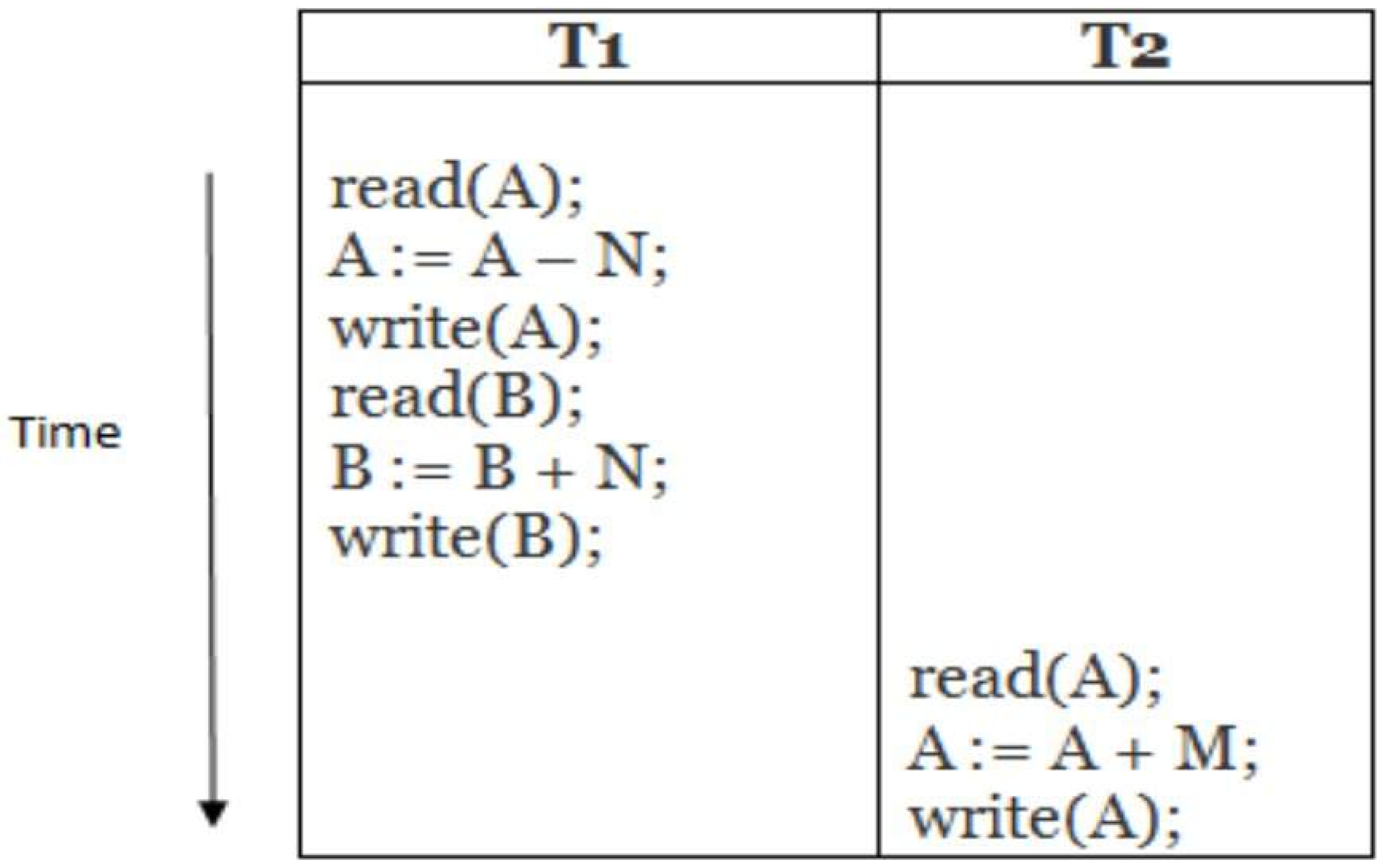
2. Non-serial Schedule

- If interleaving of operations is allowed, then there will be non-serial schedule.
- It contains many possible orders in which the system can execute the individual operations of the transactions.
- In the given figure (c) and (d), Schedule C and Schedule D are the non-serial schedules. It has interleaving of operations.

3. Serializable schedule

- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
- It identifies which schedules are correct when executions of the transaction have interleaving of their operations.
- A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.

(a)



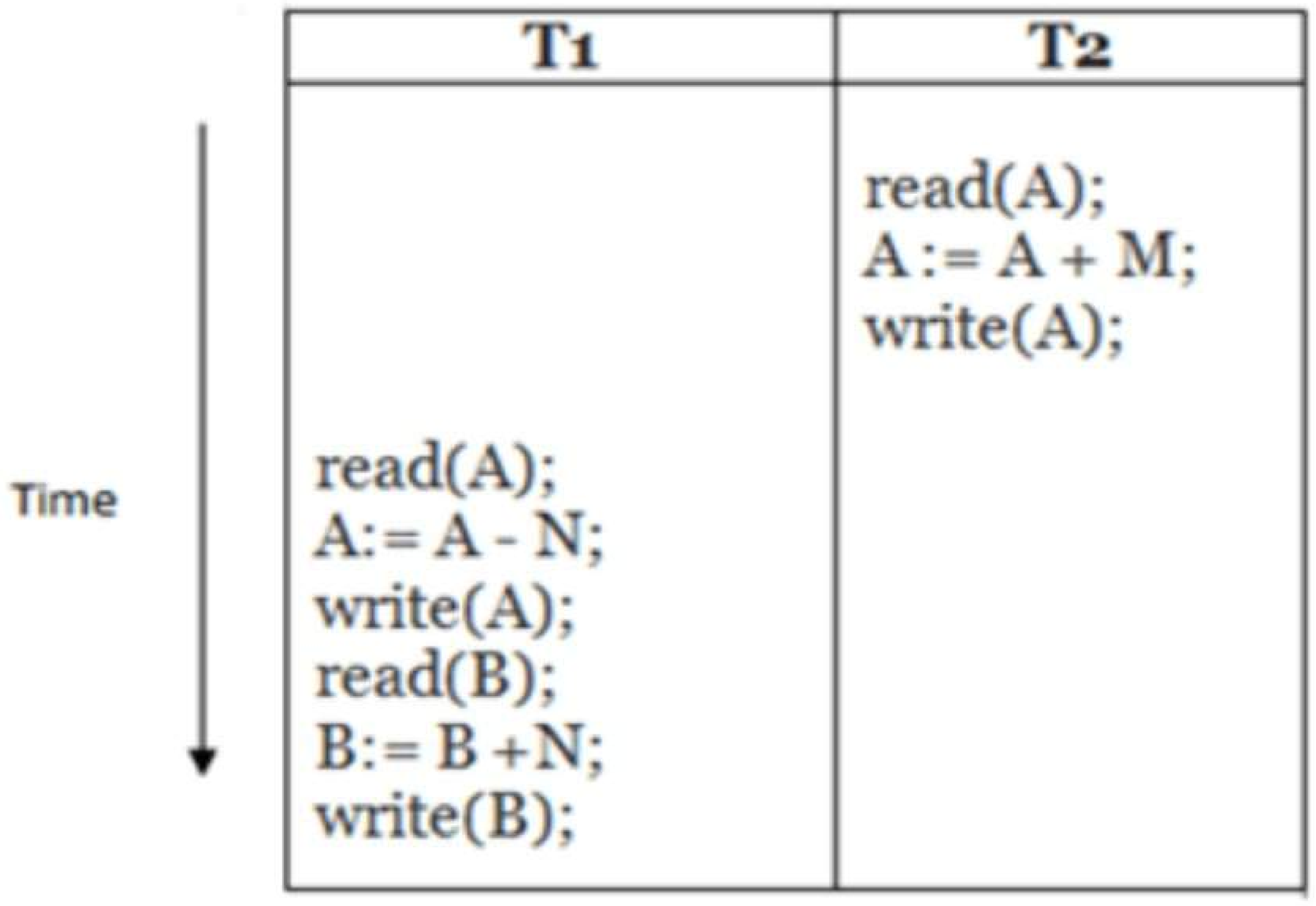
T1	T2

read(A);
A:= A – N;
write(A);
read(B);
B := B + N;
write(B);	

	read(A);
	A:= A + M;
	write(A);

Schedule A

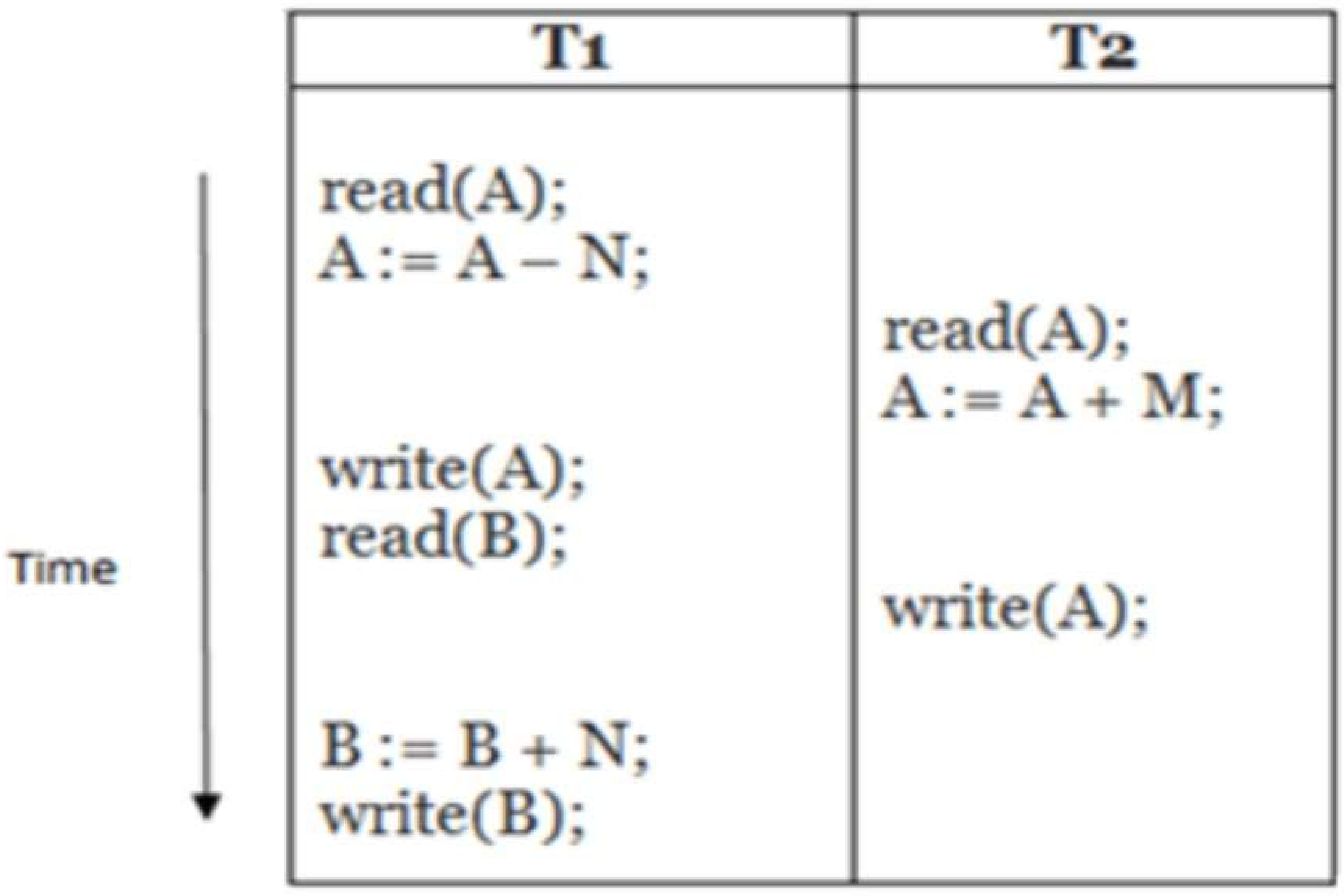
(b)



T1	T2
	read(A);
	A := A + M;
	write(A);
read(A);	
A:= A - N;	
write(A);	
read(B);	
B:= B +N;	
write(B);	

Schedule B

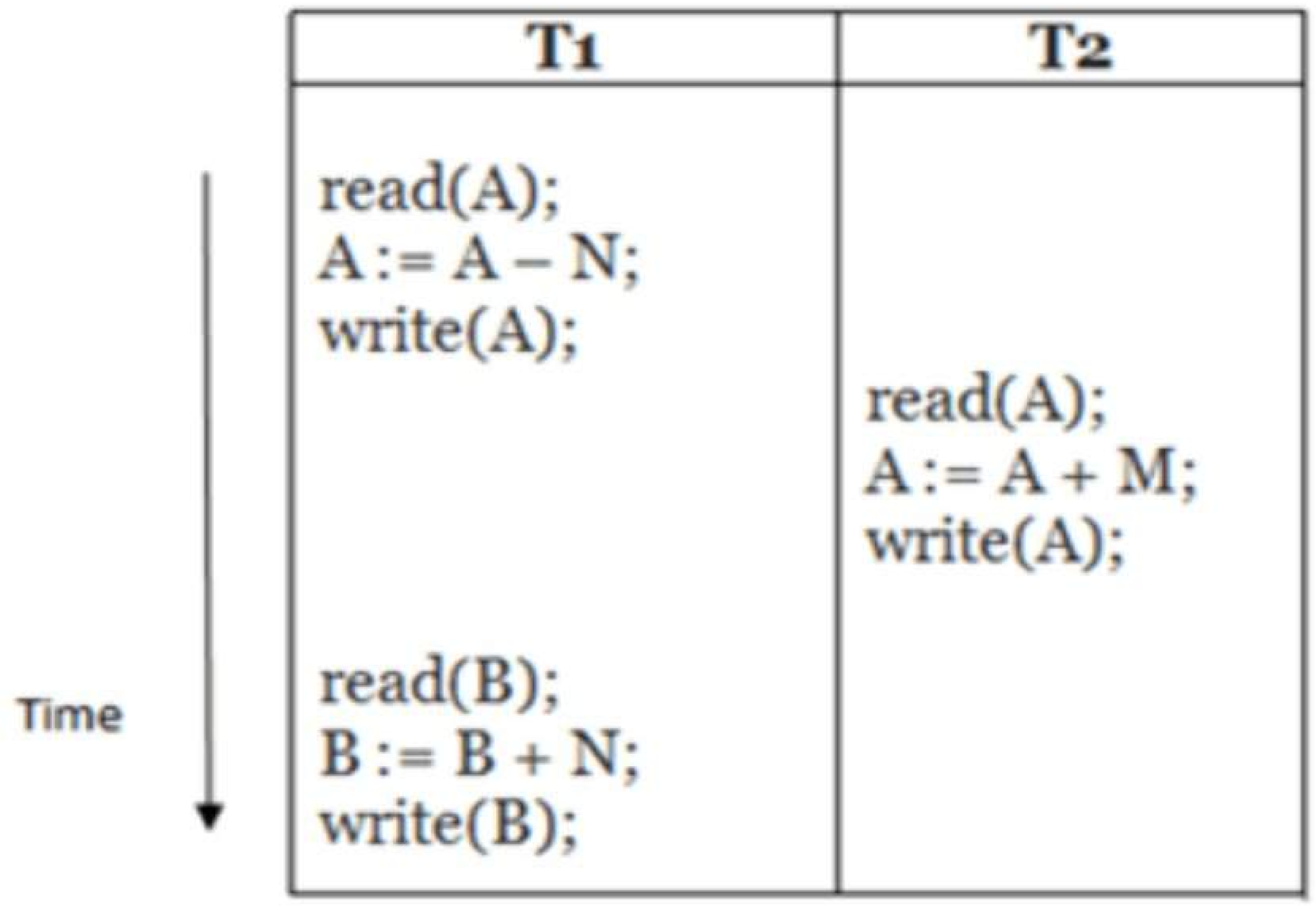
(c)



T1	T2
read(A);	
A := A – N;	
	read(A);
	A := A + M;
write(A);	
read(B);	
	write(A);
B := B + N;	
write(B);	

Schedule C

(d)



T1	T2

read(A);
A := A – N;
write(A);	

	read(A);
	A := A + M;
	write(A);

read(B);
B := B + N;
write(B);	

Schedule D

Here,

Schedule A and Schedule B are serial schedule.

Schedule C and Schedule D are Non-serial schedule.

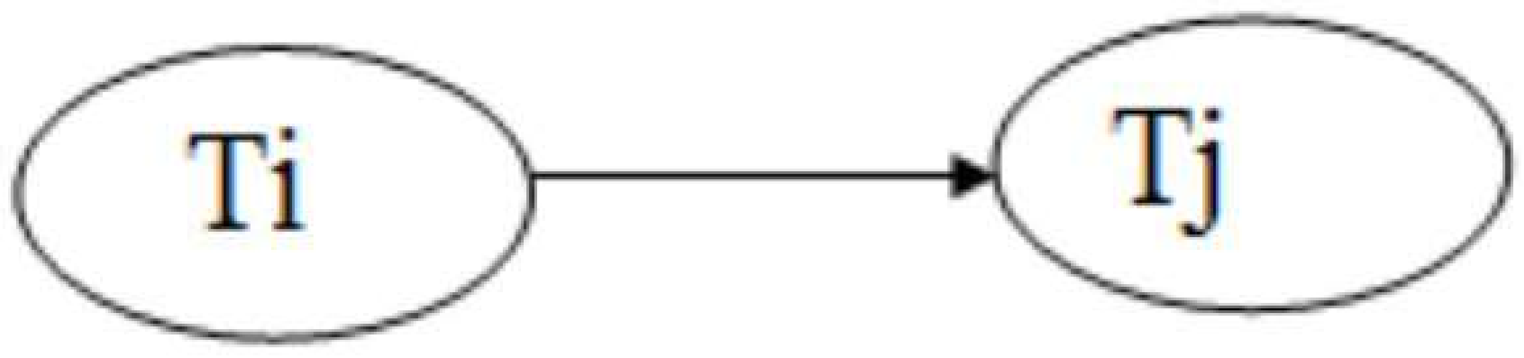
Testing of Serializability

Serialization Graph is used to test the Serializability of a schedule.

Assume a schedule S. For S, we construct a graph known as precedence graph. This graph has a pair G = (V, E), where V consists a set of vertices, and E consists a set of edges. The set of vertices is used to contain all the transactions participating in the schedule. The set of edges is used to contain all edges Ti ->Tj for which one of the three conditions holds:

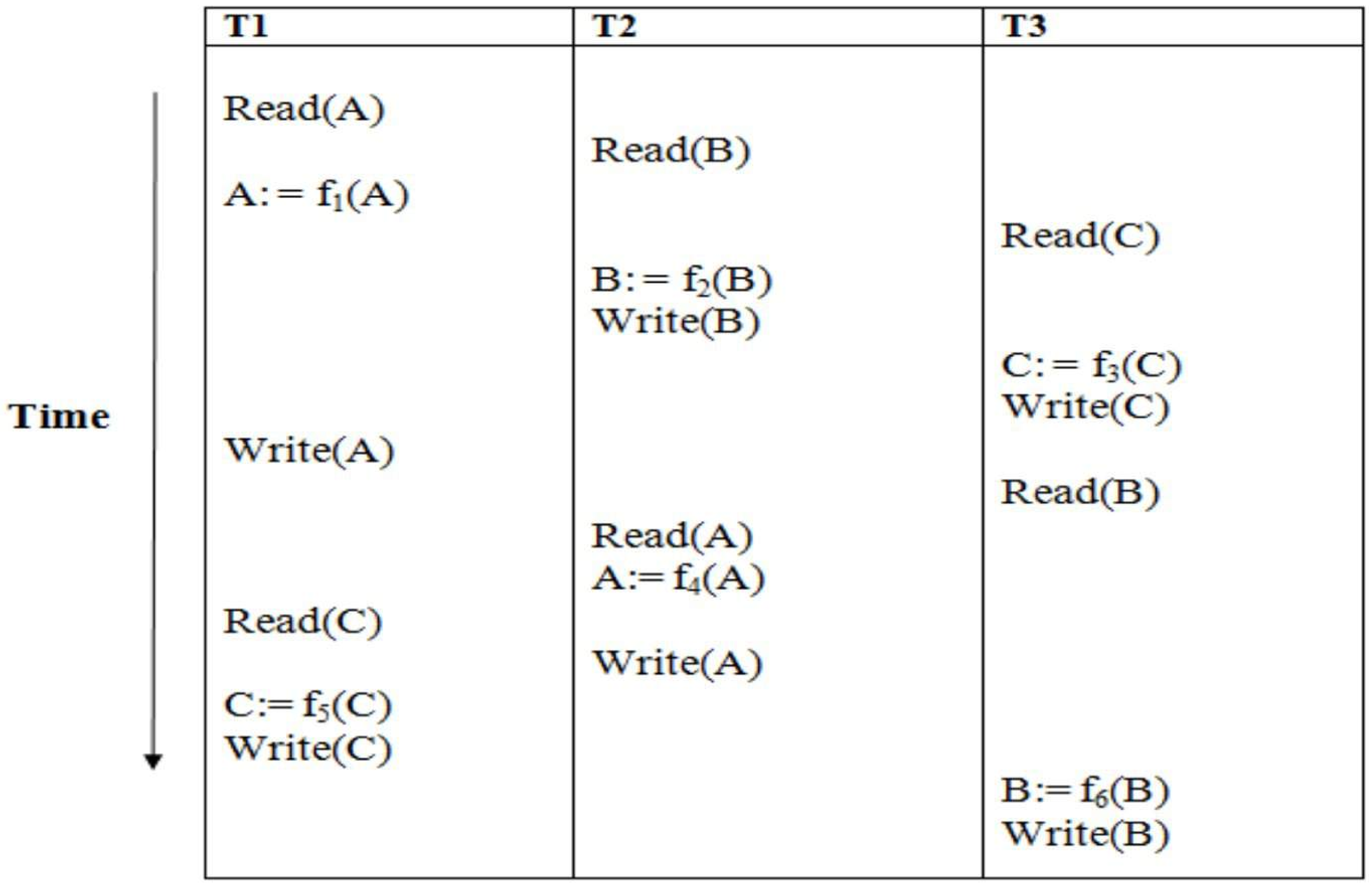
1. Create a node Ti → Tj if Ti executes write (Q) before Tj executes read (Q).
2. Create a node Ti → Tj if Ti executes read (Q) before Tj executes write (Q).
3. Create a node Ti → Tj if Ti executes write (Q) before Tj executes write (Q).

Precedence graph for Schedule S



- If a precedence graph contains a single edge Ti → Tj, then all the instructions of Ti are executed before the first instruction of Tj is executed.
- If a precedence graph for schedule S contains a cycle, then S is non-serializable. If the precedence graph has no cycle, then S is known as serializable.

For example:



T1	T2	T3
Read(A)		
	Read(B)	
A:= f_1(A)		
		Read(C)
	B:= f_2(B)	
	Write(B)	
		C:= f_3(C)
		Write(C)
Write(A)		
		Read(B)
	Read(A)	
	A:= f_4(A)	
Read(C)		
	Write(A)	
C:= f_5(C)		
Write(C)		
		B:= f_6(B)
		Write(B)

Schedule S1

Explanation:

Read(A): In T1, no subsequent writes to A, so no new edges

Read(B): In T2, no subsequent writes to B, so no new edges

Read(C): In T3, no subsequent writes to C, so no new edges

Write(B): B is subsequently read by T3, so add edge T2 → T3

Write(C): C is subsequently read by T1, so add edge T3 → T1

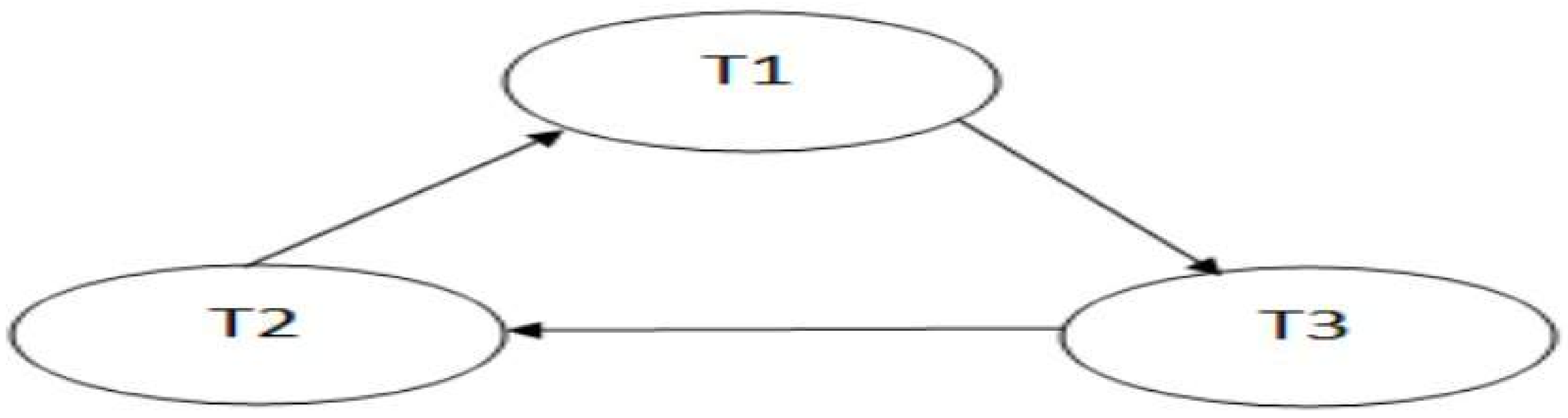
Write(A): A is subsequently read by T2, so add edge T1 → T2

Write(A): In T2, no subsequent reads to A, so no new edges

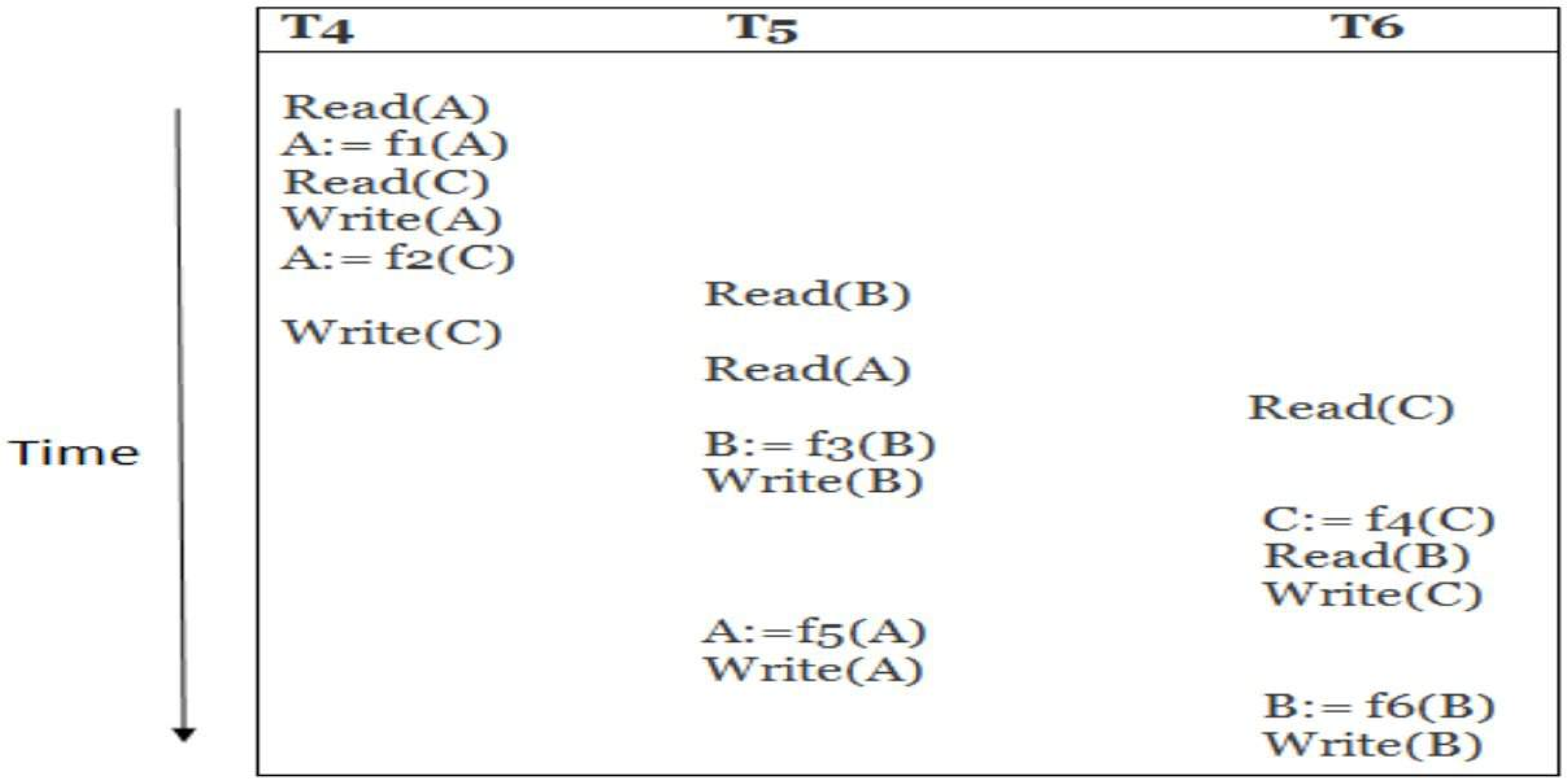
Write(C): In T1, no subsequent reads to C, so no new edges

Write(B): In T3, no subsequent reads to B, so no new edges

Precedence graph for schedule S1:



The precedence graph for schedule S1 contains a cycle that's why Schedule S1 is non-serializable.



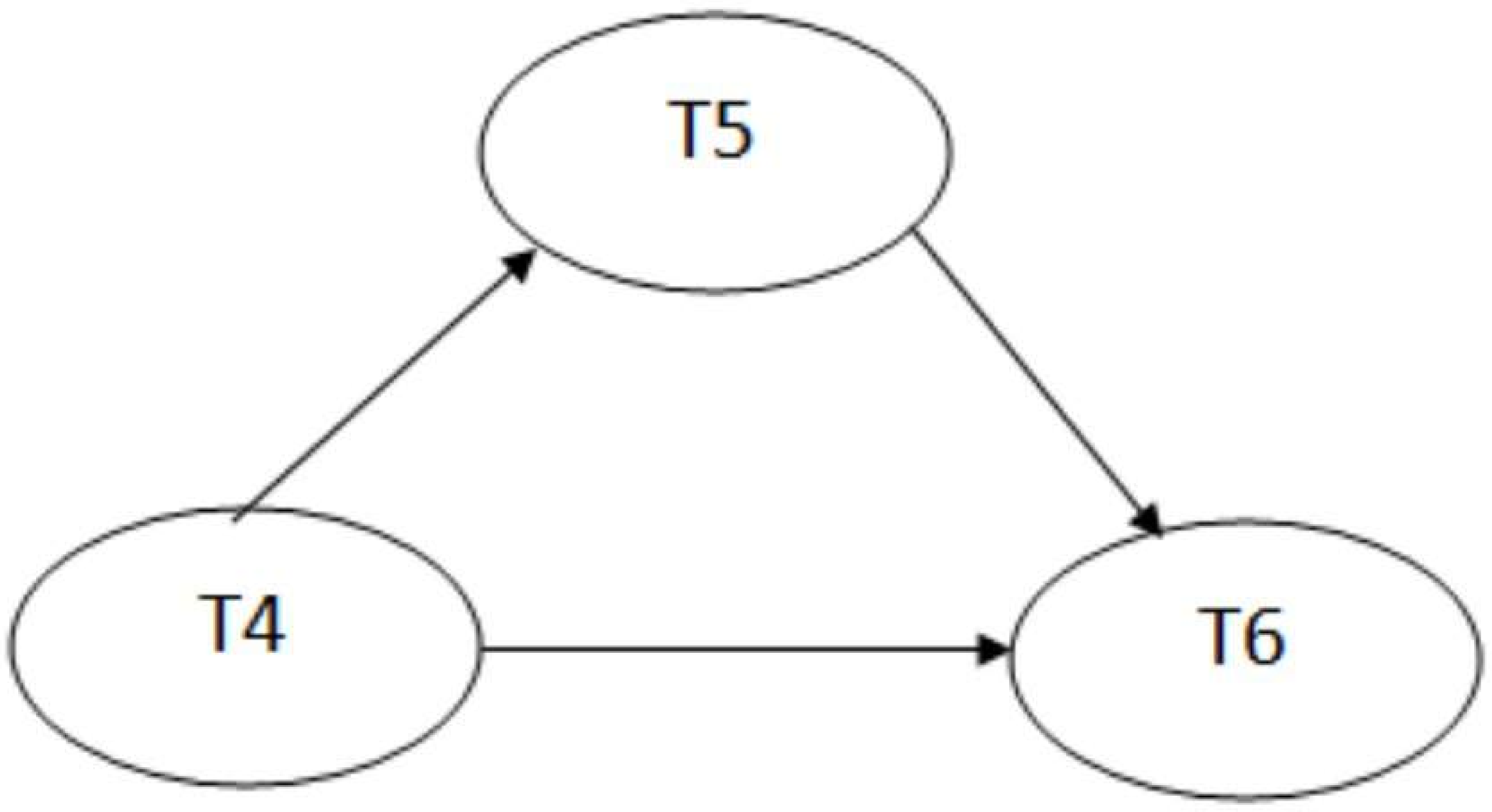
T4	T5	T6
Read(A)		
A:= f1(A)		
Read(C)		
Write(A)		
A:= f2(C)		
	Read(B)	
Write(C)		
	Read(A)	
		Read(C)
	B:= f3(B)	
	Write(B)	
		C:= f4(C)
		Read(B)
		Write(C)
	A:=f5(A)	
	Write(A)	
		B:= f6(B)
		Write(B)

Schedule S2

Explanation:

Read(A): In T4,no subsequent writes to A, so no new edges
Read(C): In T4, no subsequent writes to C, so no new edges
Write(A): A is subsequently read by T5, so add edge T4 → T5
Read(B): In T5,no subsequent writes to B, so no new edges
Write(C): C is subsequently read by T6, so add edge T4 → T6
Write(B): A is subsequently read by T6, so add edge T5 → T6
Write(C): In T6, no subsequent reads to C, so no new edges
Write(A): In T5, no subsequent reads to A, so no new edges
Write(B): In T6, no subsequent reads to B, so no new edges

Precedence graph for schedule S2:



The precedence graph for schedule S2 contains no cycle that's why ScheduleS2 is serializable.

Conflict Serializable Schedule

- A schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into a serial schedule.
- The schedule will be a conflict serializable if it is conflict equivalent to a serial schedule.

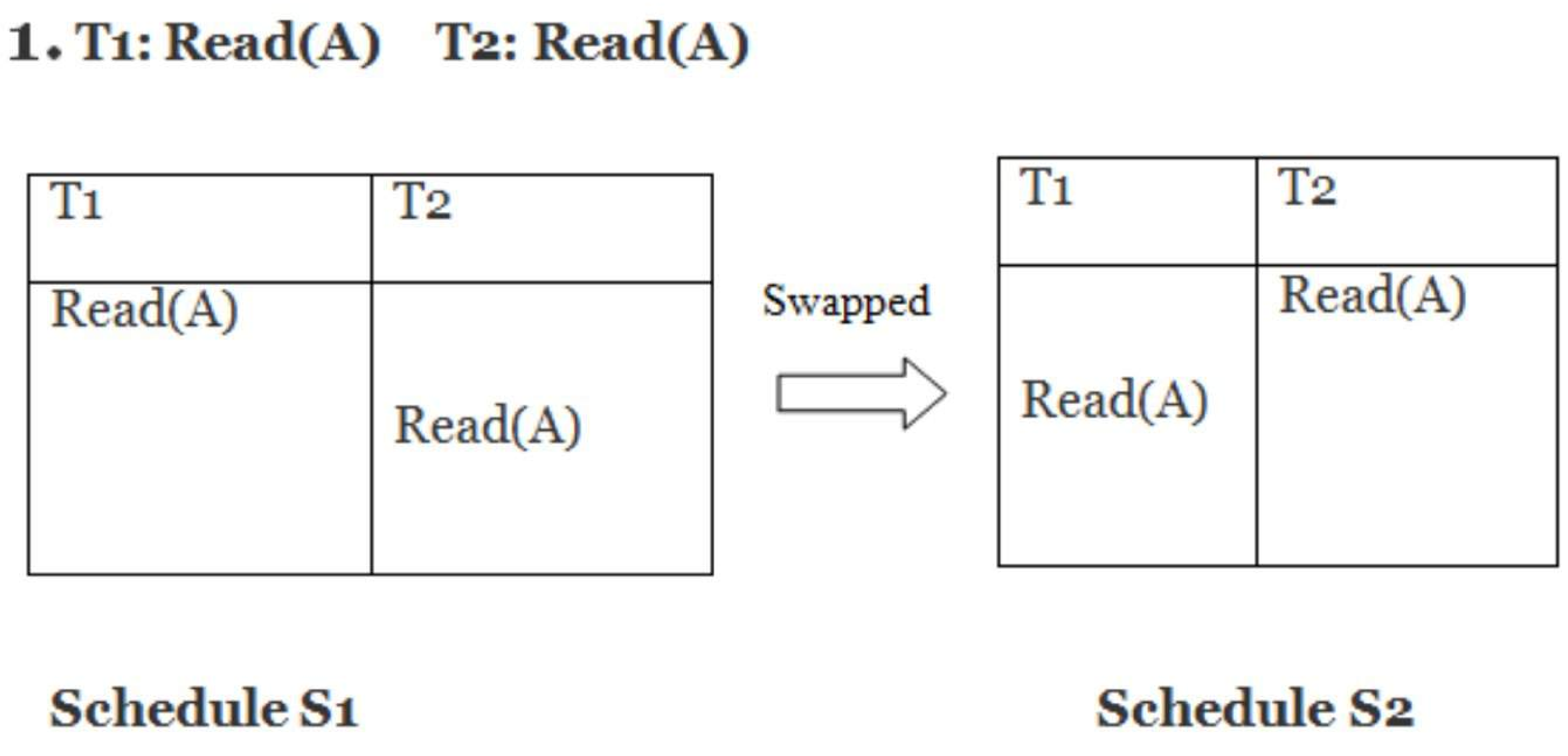
Conflicting Operations

The two operations become conflicting if all conditions satisfy:

1. Both belong to separate transactions.
2. They have the same data item.
3. They contain at least one write operation.

Example:

Swapping is possible only if S1 and S2 are logically equal.



1. T1: Read(A)   T2: Read(A)

T1	T2
Read(A)	
	Read(A)

T1	T2
	Read(A)
Read(A)	

Schedule S1

Schedule S2

Here, S1 = S2. That means it is non-conflict.

2. T1: Read(A) T2: Write(A)

T1	T2
Read(A)	
	Write(A)

Schedule S1

Swapped ⇒

T1	T2
	Write(A)
Read(A)	

Schedule S2

Here, S1 ≠ S2. That means it is conflict.

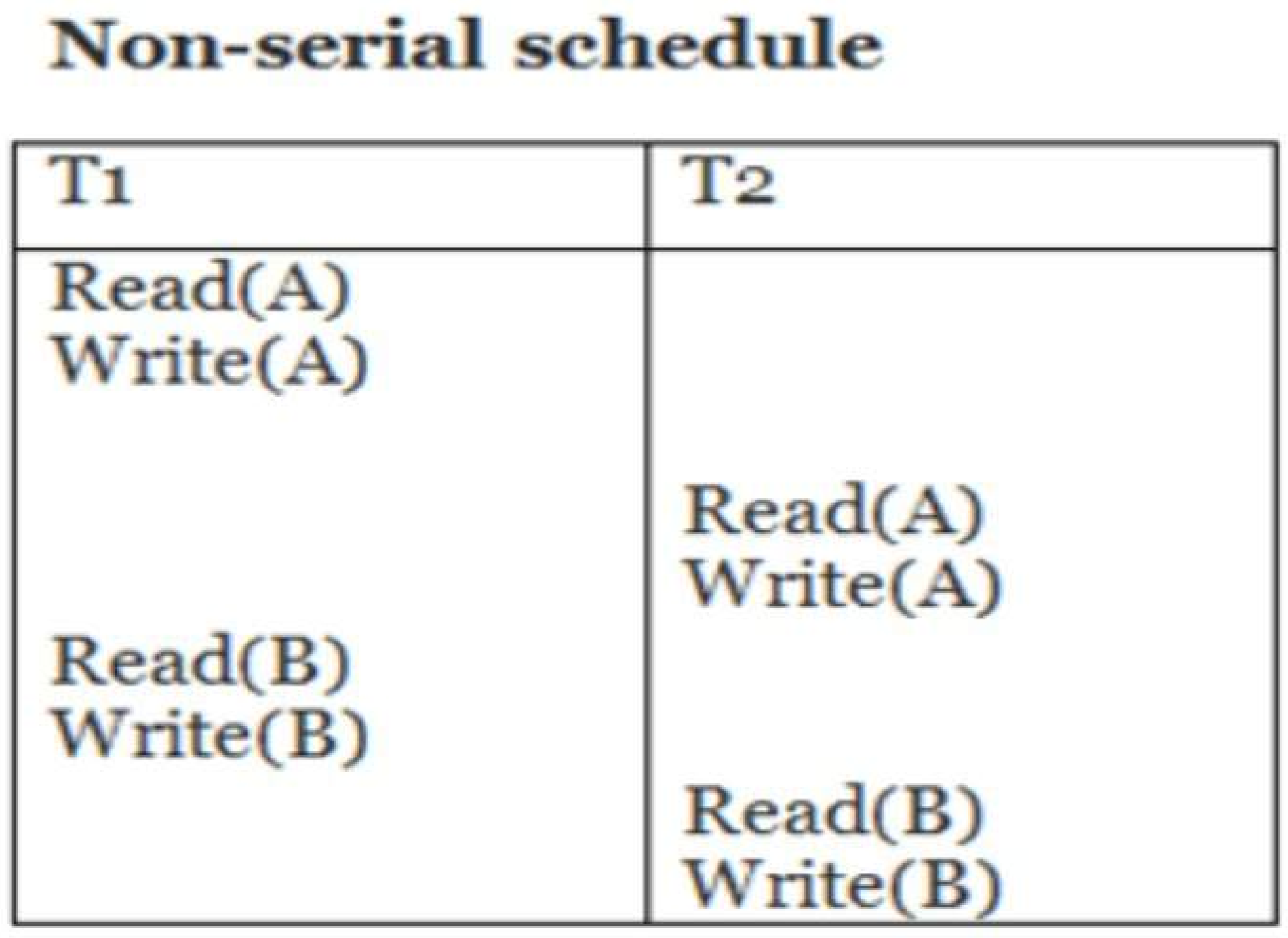
Conflict Equivalent

In the conflict equivalent, one can be transformed to another by swapping non-conflicting operations. In the given example, S2 is conflict equivalent to S1 (S1 can be converted to S2 by swapping non-conflicting operations).

Two schedules are said to be conflict equivalent if and only if:

1. They contain the same set of the transaction.
2. If each pair of conflict operations are ordered in the same way.

Example:



Non-serial schedule

T1	T2

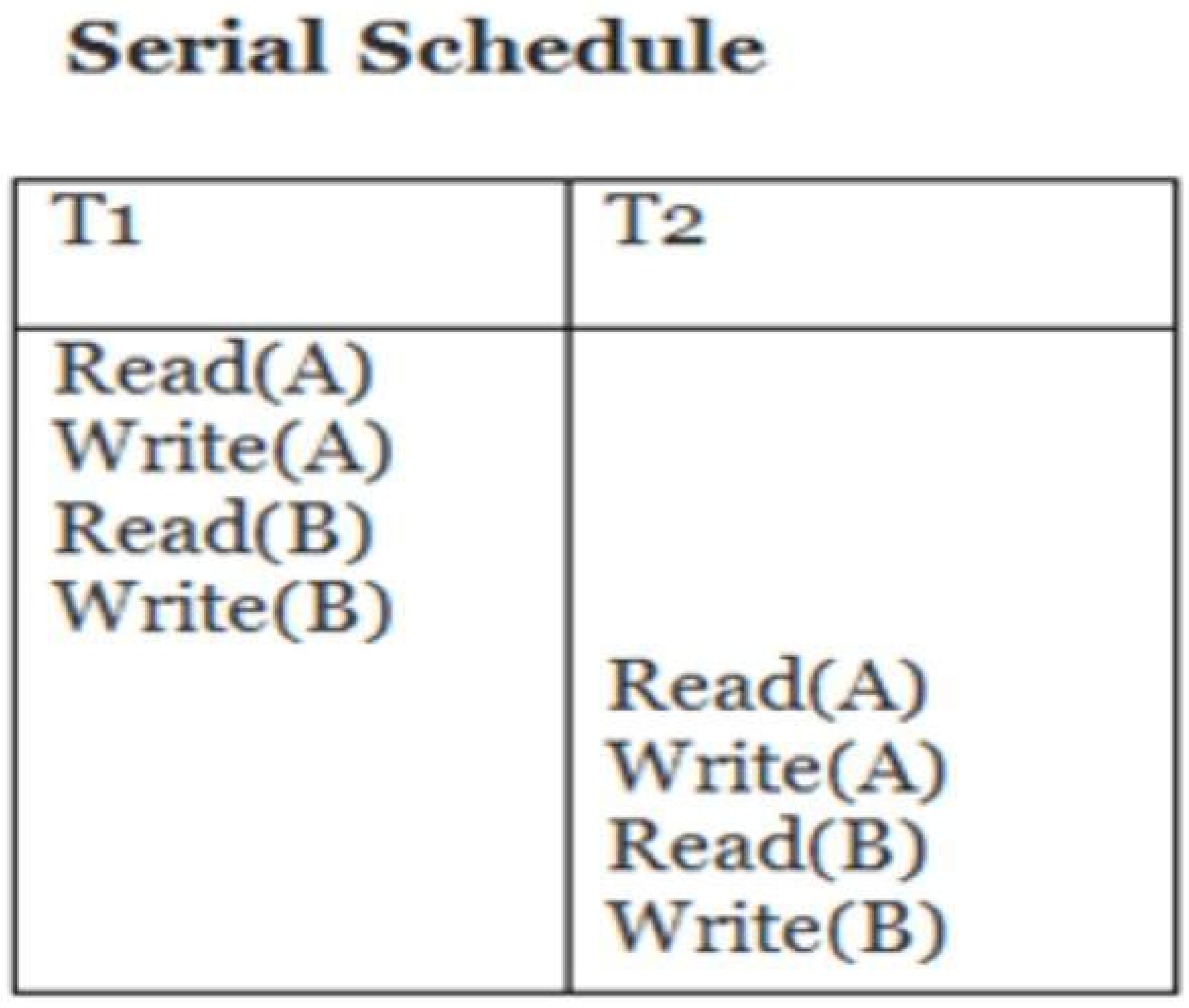
Read(A)
Write(A)	

	Read(A)
	Write(A)

Read(B)
Write(B)	

	Read(B)
	Write(B)

Schedule S1



Serial Schedule

T1	T2

Read(A)
Write(A)
Read(B)
Write(B)	

	Read(A)
	Write(A)
	Read(B)
	Write(B)

Schedule S2

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

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

T1	**T2**

Read(A)
Write(A)
Read(B)
Write(B)	

	Read(A)
	Write(A)
	Read(B)
	Write(B)

Since, S1 is conflict serializable.

View Serializability

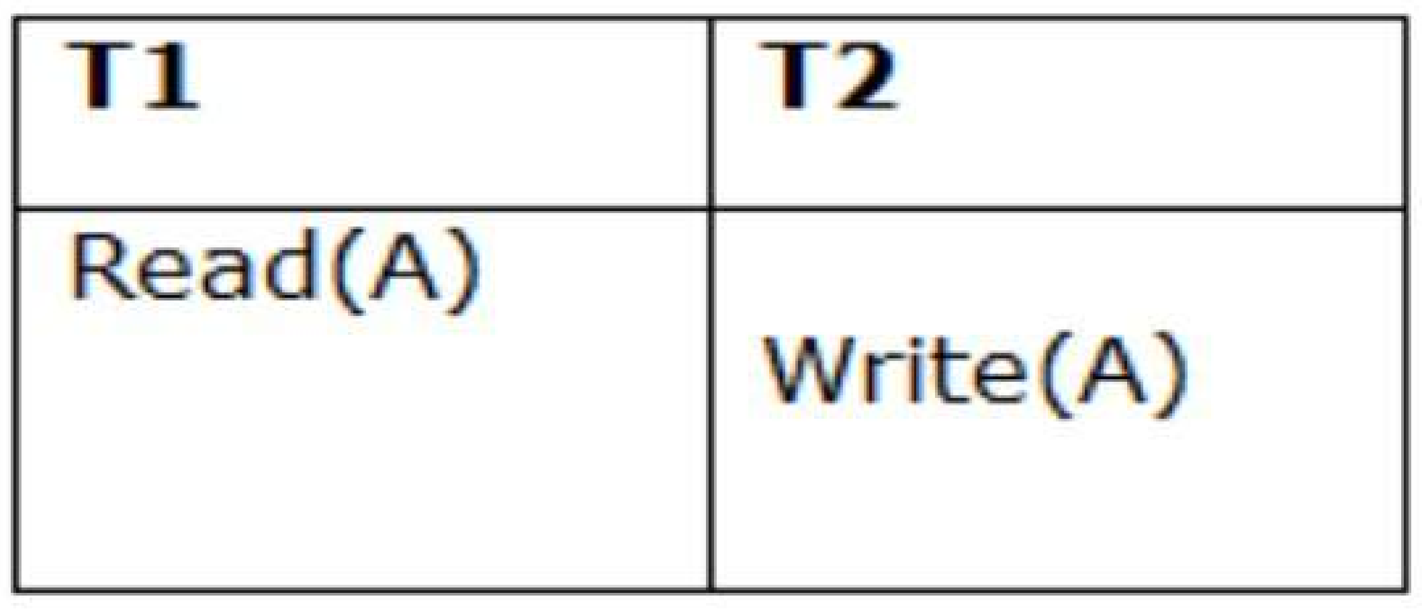
- A schedule will view serializable if it is view equivalent to a serial schedule.
- If a schedule is conflict serializable, then it will be view serializable.
- The view serializable which does not conflict serializable contains blind writes.

View Equivalent

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

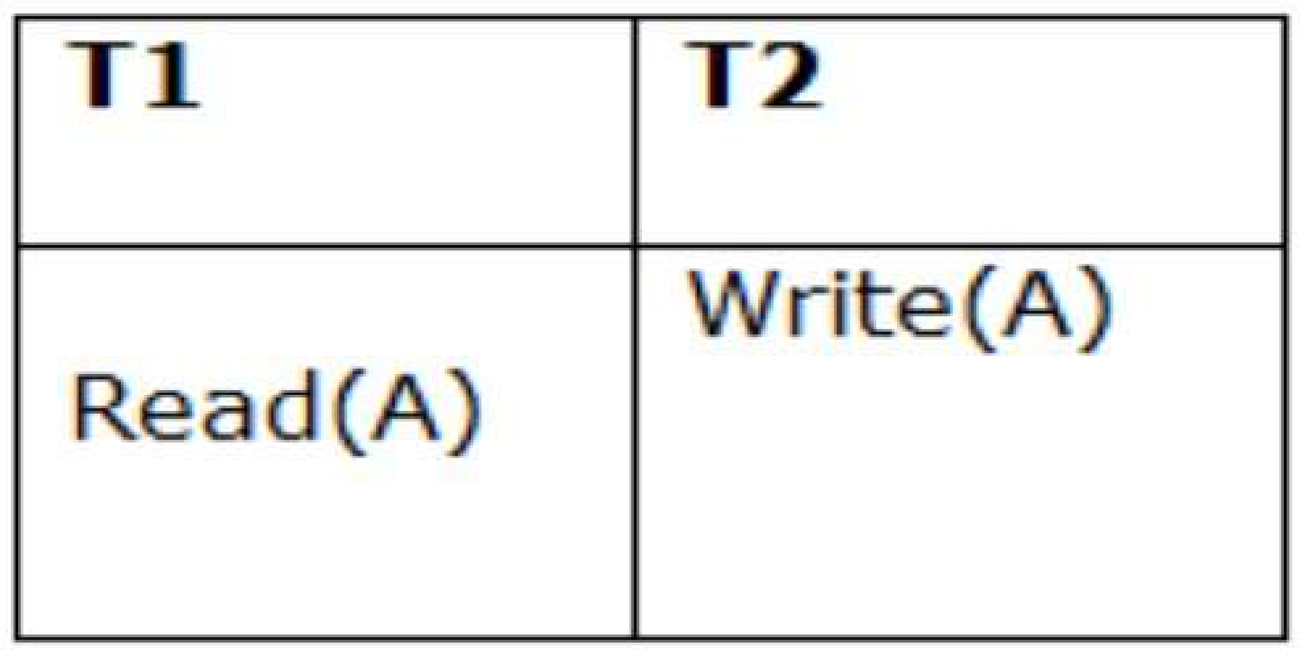
1. Initial Read

An initial read of both schedules must be the same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.



T1	T2
Read(A)	
	Write(A)

Schedule S1



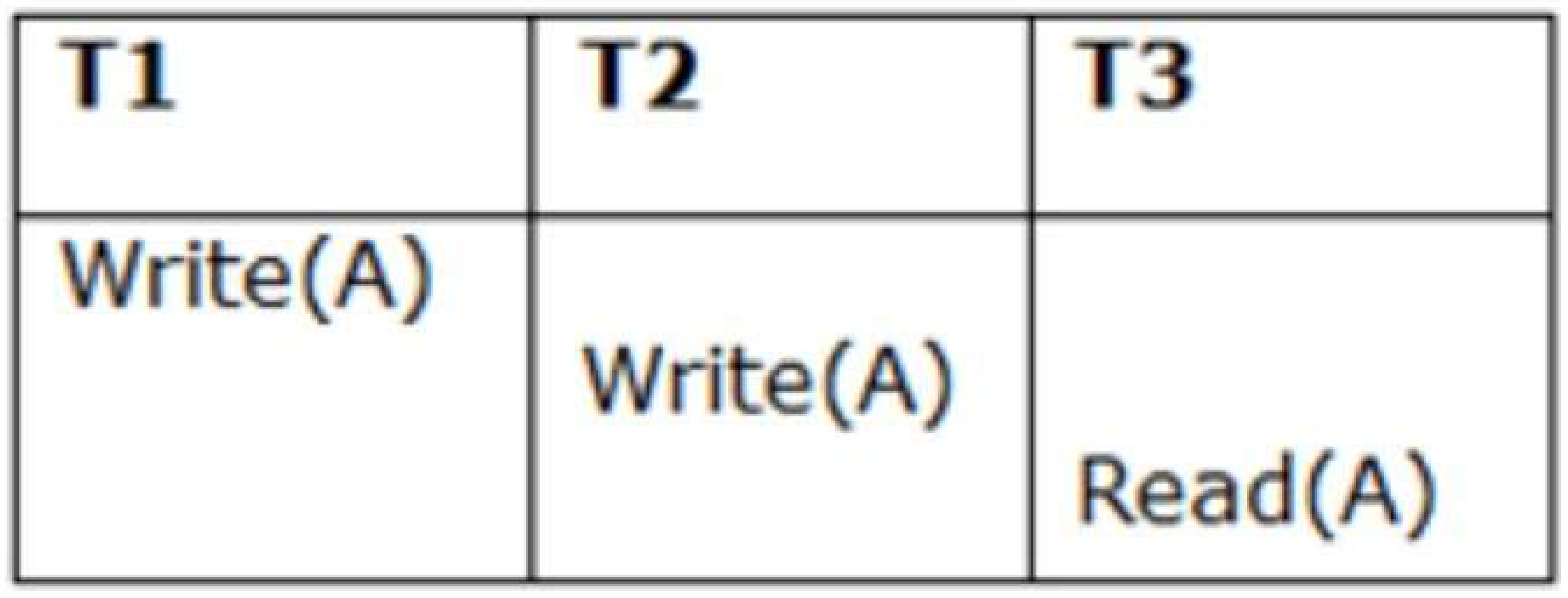
T1	T2
	Write(A)
Read(A)	

Schedule S2

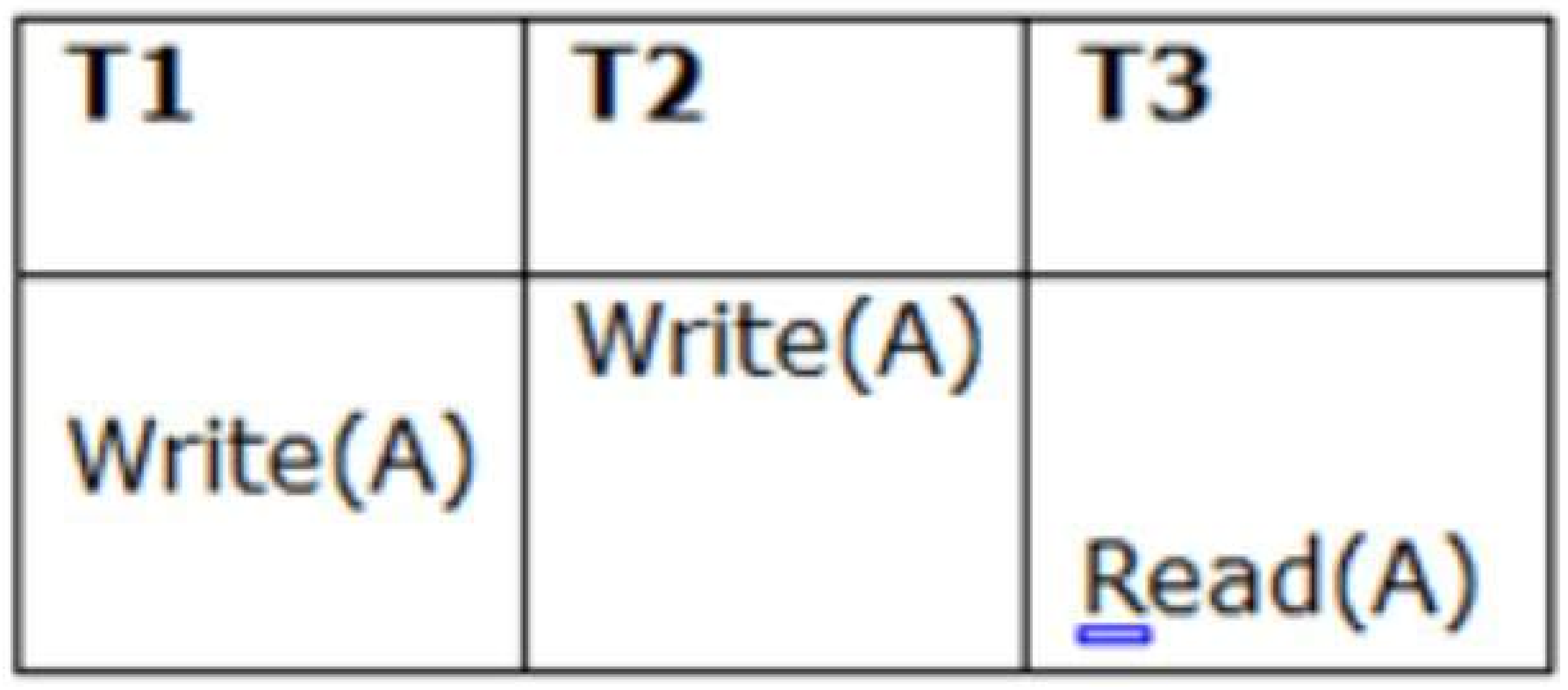
Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

2. Updated Read

In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj.



T1	T2	T3
Write(A)		
	Write(A)	
		Read(A)



T1	T2	T3
	Write(A)	
Write(A)		
		Read(A)

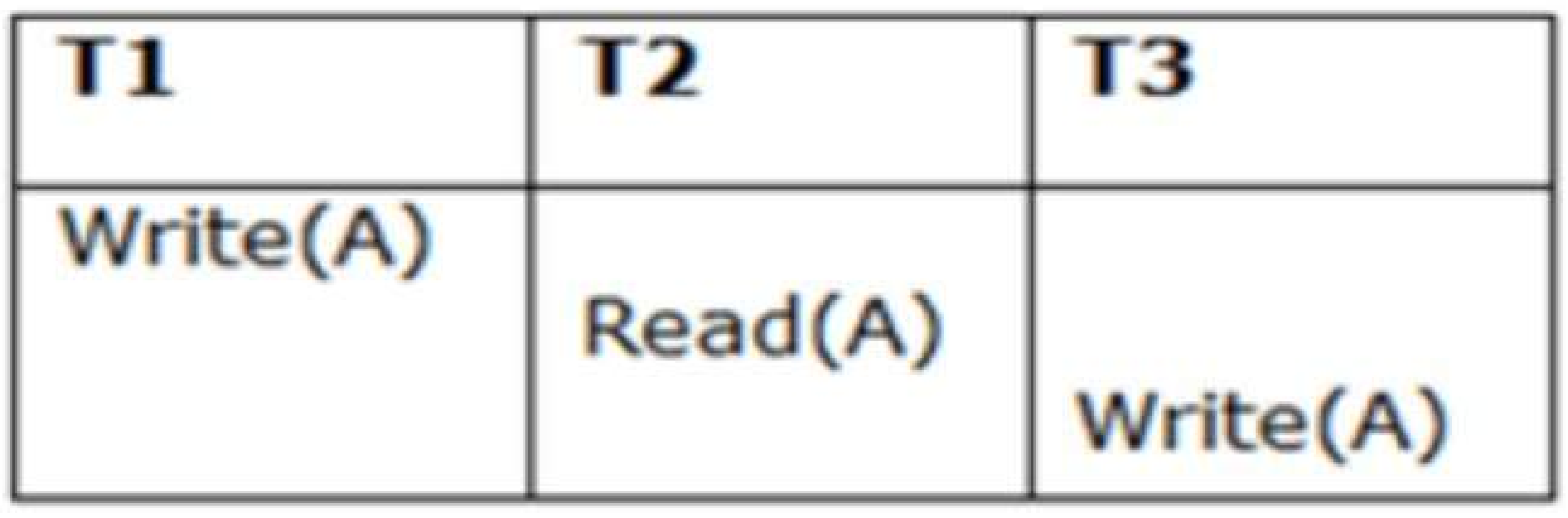
Schedule S1

Schedule S2

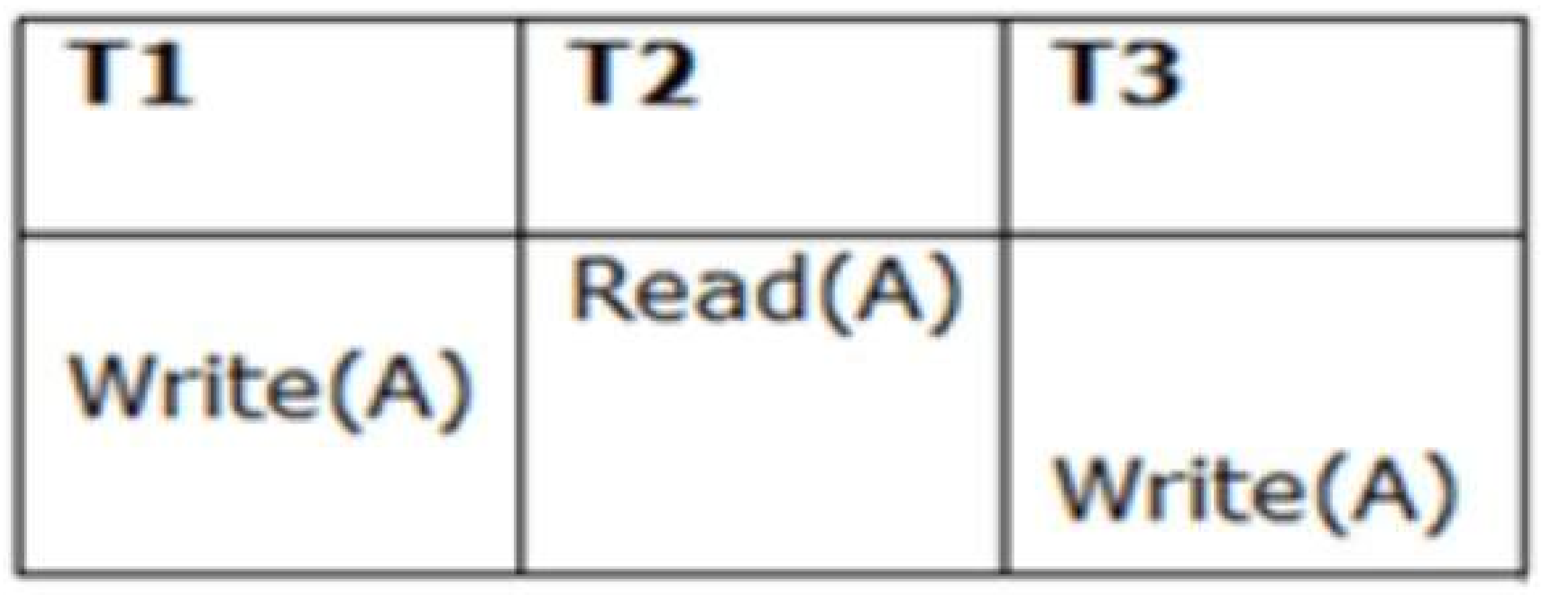
Above two schedules are not view equal because, in S1, T3 is reading A updated by T2 and in S2, T3 is reading A updated by T1.

3. Final Write

A final write must be the same between both the schedules. In schedule S1, if a transaction T1 updates A at last then in S2, final writes operations should also be done by T1.



T1	T2	T3
Write(A)		
	Read(A)	
		Write(A)



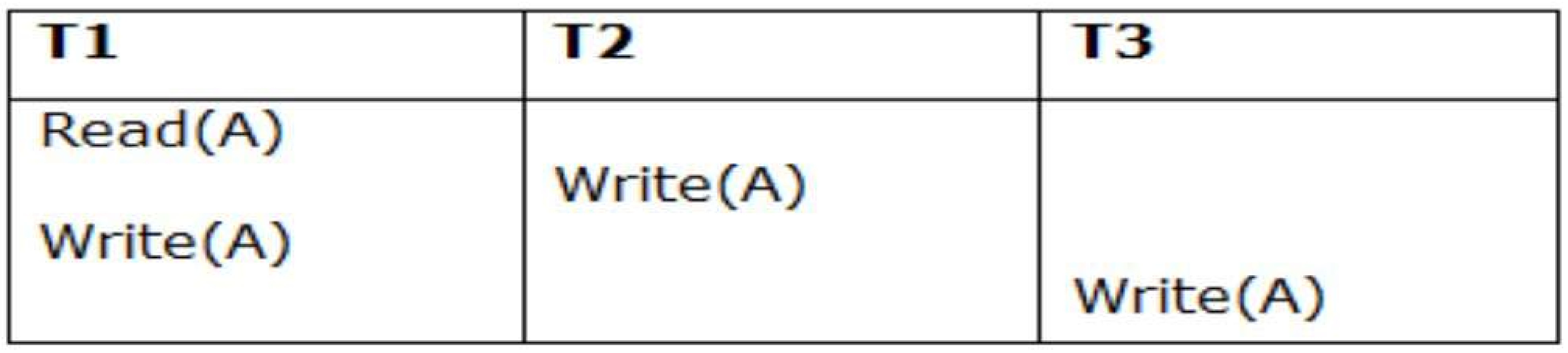
T1	T2	T3
	Read(A)	
Write(A)		
		Write(A)

Schedule S1

Schedule S2

Above two schedules is view equal because Final write operation in S1 is done by T3 and in S2, the final write operation is also done by T3.

Example:



T1	T2	T3
Read(A)		
	Write(A)	
Write(A)		
		Write(A)

Schedule S

With 3 transactions, the total number of possible schedule

= 3! = 6

S1 = <T1 T2 T3>

S2 = <T1 T3 T2>

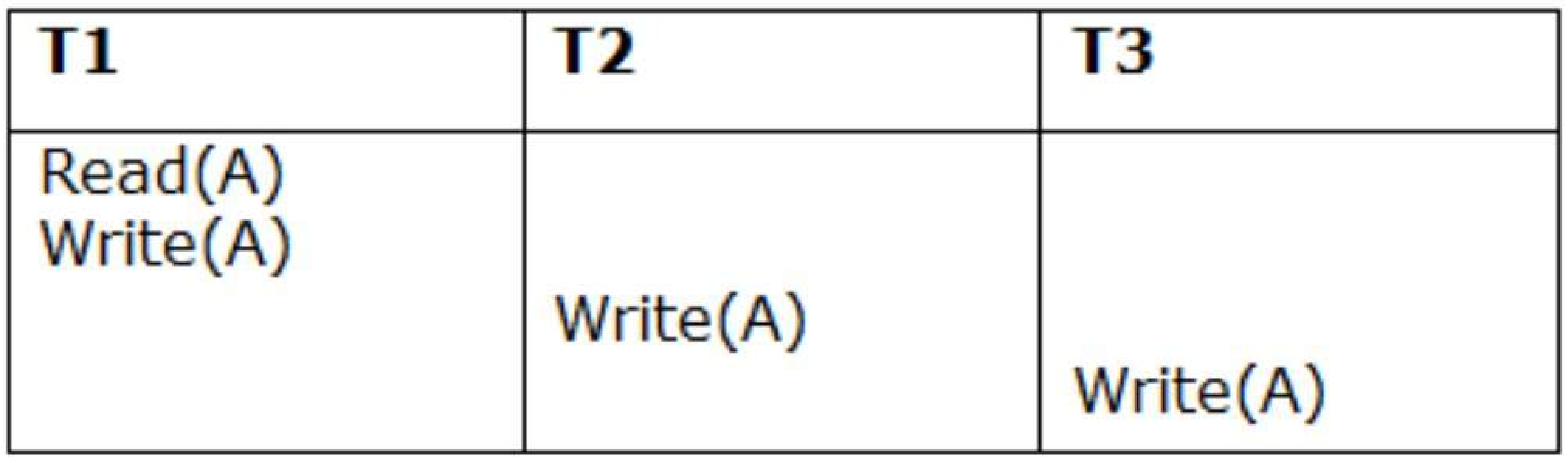
S3 = <T2 T3 T1>

S4 = <T2 T1 T3>

S5 = <T3 T1 T2>

S6 = <T3 T2 T1>

Taking first schedule S1:



T1	T2	T3
Read(A)		
Write(A)		
	Write(A)	
		Write(A)

Schedule S1

Step 1: final updation on data items

In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

Step 2: Initial Read

The initial read operation in S is done by T1 and in S1, it is also done by T1.

Step 3: Final Write

The final write operation in S is done by T3 and in S1, it is also done by T3. So, S and S1 are view Equivalent.

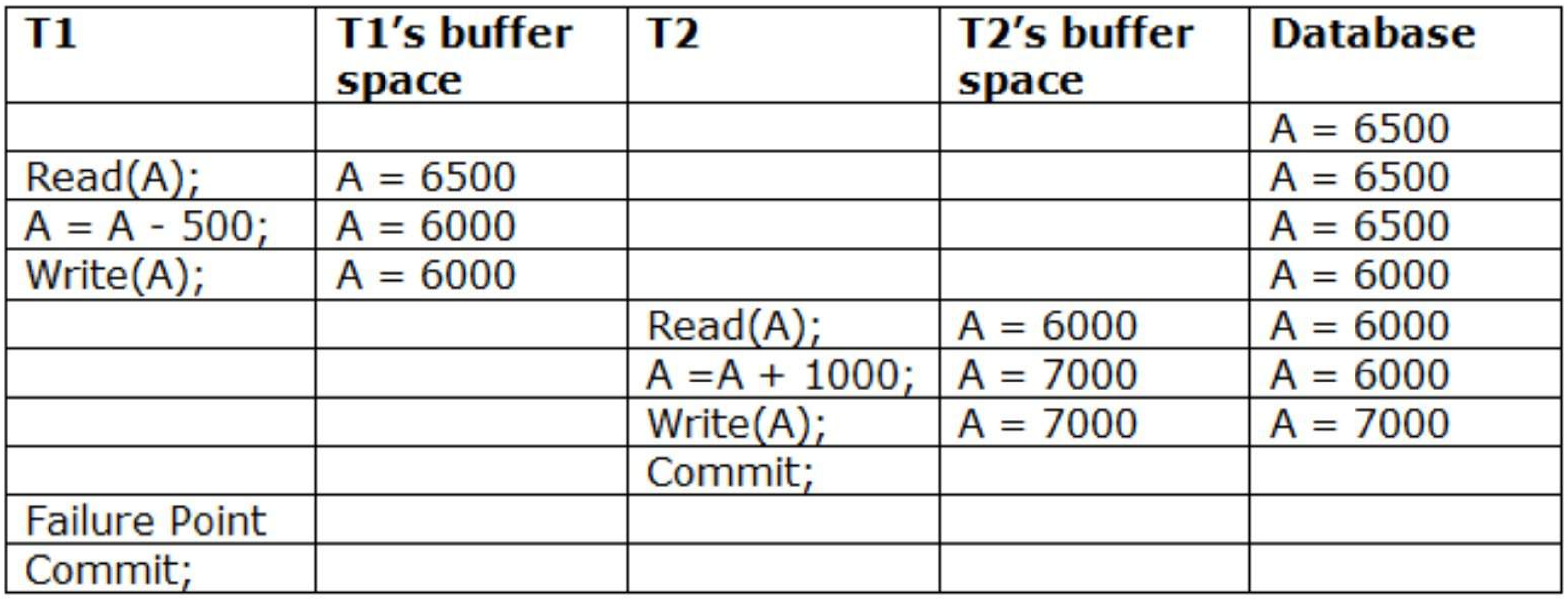
The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.

Hence, view equivalent serial schedule is:

1. T1 → T2 → T3

Recoverability of Schedule

Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure. In that case, the failed transaction has to be rollback. But some other transaction may also have used value produced by the failed transaction. So we also have to rollback those transactions.



T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		
Failure Point				
Commit;				

The above table 1 shows a schedule which has two transactions. T1 reads and writes the value of A and that value is read and written by T2. T2 commits but later on, T1 fails. Due to the failure, we have to rollback T1. T2 should also be rollback because it reads the value written by T1, but T2 can't be rollback because it already committed. So this type of schedule is known as irrecoverable schedule.

Irrecoverable schedule: The schedule will be irrecoverable if Tj reads the updated value of Ti and Tj committed before Ti commit.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
Failure Point				
Commit;				
		Commit;		

The above table 2 shows a schedule with two transactions. Transaction T1 reads and writes A, and that value is read and written by transaction T2. But later on, T1 fails. Due to this, we have to rollback T1. T2 should be rollback because T2 has read the value written by T1. As it has not committed before T1 commits so we can rollback transaction T2 as well. So it is recoverable with cascade rollback.

Recoverable with cascading rollback: The schedule will be recoverable with cascading rollback if Tj reads the updated value of Ti. Commit of Tj is delayed till commit of Ti.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
Commit;		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		

The above Table 3 shows a schedule with two transactions. Transaction T1 reads and write A and commits, and that value is read and written by T2. So this is a cascade less recoverable schedule.

Failure Classification

To find that where the problem has occurred, we generalize a failure into the following categories:

1. Transaction failure
2. System crash
3. Disk failure

1. Transaction failure

The transaction failure occurs when it fails to execute or when it reaches a point from where it can't go any further. If a few transaction or process is hurt, then this is called as transaction failure.

Reasons for a transaction failure could be -

- **Logical errors:** If a transaction cannot complete due to some code error or an internal error condition, then the logical error occurs.
- **Syntax error:** It occurs where the DBMS itself terminates an active transaction because the database system is not able to execute it. **For example,** The system aborts an active transaction, in case of deadlock or resource unavailability.

2. System Crash

- System failure can occur due to power failure or other hardware or software failure. **Example:** Operating system error.
- **Fail-stop assumption:** In the system crash, non-volatile storage is assumed not to be corrupted.

3. Disk Failure

- It occurs where hard-disk drives or storage drives used to fail frequently. It was a common problem in the early days of technology evolution.
- Disk failure occurs due to the formation of bad sectors, disk head crash, and unreachability to the disk or any other failure, which destroy all or part of disk storage.

Log-Based Recovery

- The log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.
- If any operation is performed on the database, then it will be recorded in the log.
- But the process of storing the logs should be done before the actual transaction is applied in the database.

Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.

- When the transaction is initiated, then it writes 'start' log.

1. <Tn, Start>

- When the transaction modifies the City from 'Noida' to 'Bangalore', then another log is written to the file.

1. <Tn, City, 'Noida', 'Bangalore' >

- When the transaction is finished, then it writes another log to indicate the end of the transaction.

1. <Tn, Commit>

There are two approaches to modify the database:

1. Deferred database modification:

- The deferred modification technique occurs if the transaction does not modify the database until it has committed.

- In this method, all the logs are created and stored in the stable storage, and the database is updated when a transaction commits.

2. Immediate database modification:

- The Immediate modification technique occurs if database modification occurs while the transaction is still active.
- In this technique, the database is modified immediately after every operation. It follows an actual database modification.

Recovery using Log records

When the system is crashed, then the system consults the log to find which transactions need to be undone and which need to be redone.

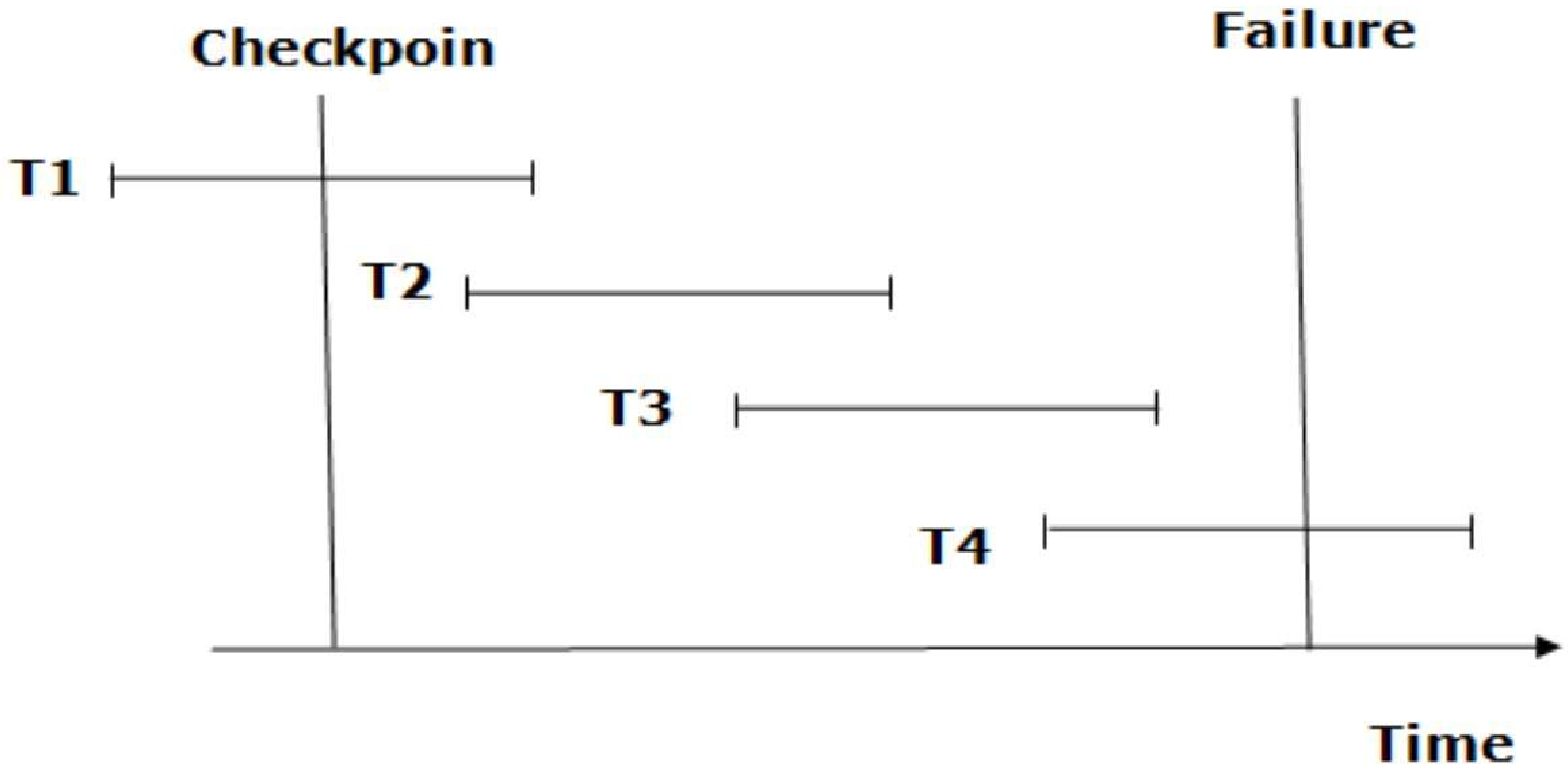
1. If the log contains the record <Ti, Start> and <Ti, Commit> or <Ti, Commit>, then the Transaction Ti needs to be redone.
2. If log contains record<T_n, Start> but does not contain the record either <Ti, commit> or <Ti, abort>, then the Transaction Ti needs to be undone.

Checkpoint

- The checkpoint is a type of mechanism where all the previous logs are removed from the system and permanently stored in the storage disk.
- The checkpoint is like a bookmark. While the execution of the transaction, such checkpoints are marked, and the transaction is executed then using the steps of the transaction, the log files will be created.
- When it reaches to the checkpoint, then the transaction will be updated into the database, and till that point, the entire log file will be removed from the file. Then the log file is updated with the new step of transaction till next checkpoint and so on.
- The checkpoint is used to declare a point before which the DBMS was in the consistent state, and all transactions were committed.

Recovery using Checkpoint

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



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

Deadlock in DBMS

A deadlock is a condition where two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as no task ever gets finished and is in waiting state forever.

For example: In the student table, transaction T1 holds a lock on some rows and needs to update some rows in the grade table. Simultaneously, transaction T2 holds locks on some rows in the grade table and needs to update the rows in the Student table held by Transaction T1.

Now, the main problem arises. Now Transaction T1 is waiting for T2 to release its lock and similarly, transaction T2 is waiting for T1 to release its lock. All activities come to a halt state and remain at a standstill. It will remain in a standstill until the DBMS detects the deadlock and aborts one of the transactions.

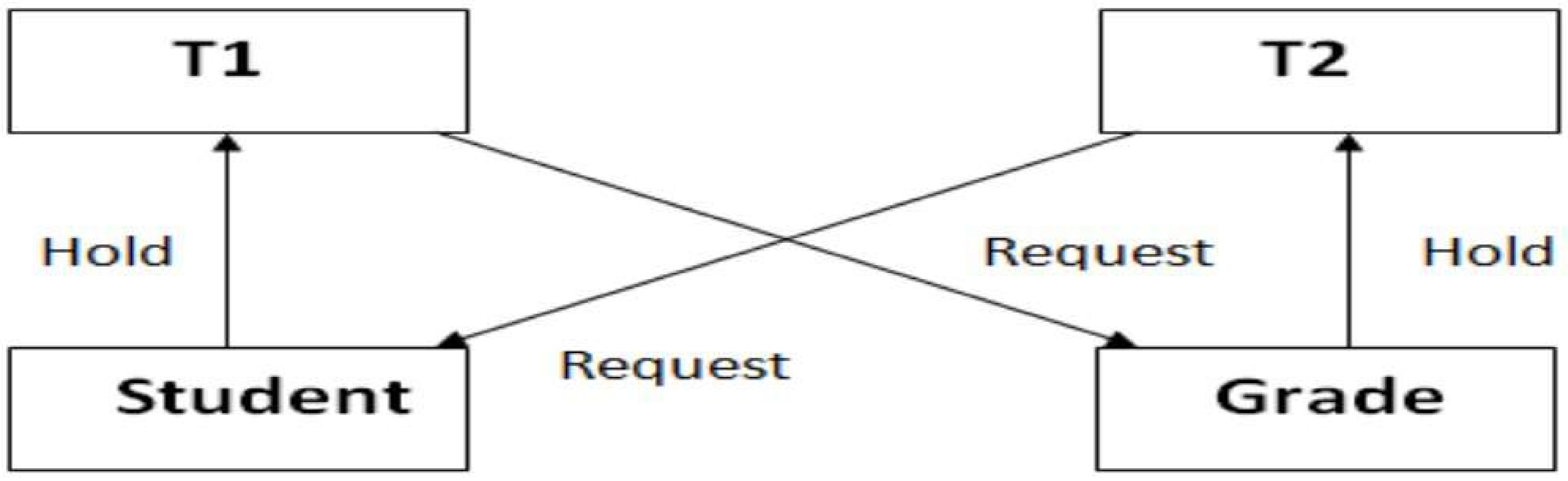


Figure: Deadlock in DBMS

Deadlock Avoidance

- When a database is stuck in a deadlock state, then it is better to avoid the database rather than aborting or restating the database. This is a waste of time and resource.
- Deadlock avoidance mechanism is used to detect any deadlock situation in advance. A method like "wait for graph" is used for detecting the deadlock situation but this method is suitable only for the smaller database. For the larger database, deadlock prevention method can be used.

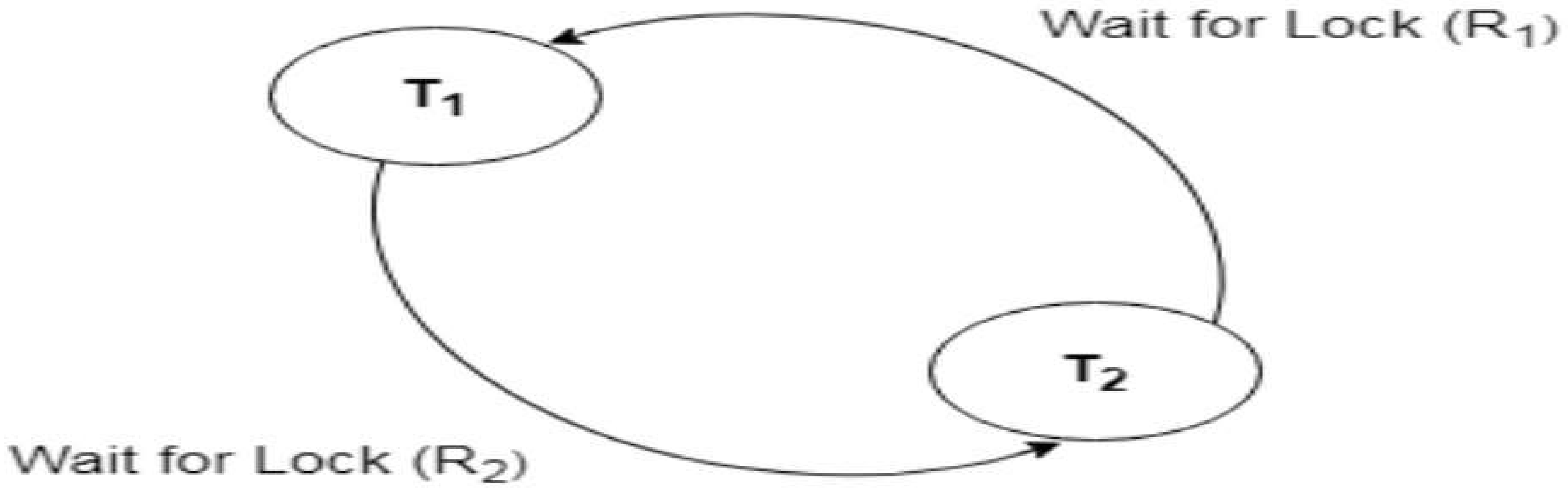
Deadlock Detection

In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not. The lock manager maintains a Wait for the graph to detect the deadlock cycle in the database.

Wait for Graph

- This is the suitable method for deadlock detection. In this method, a graph is created based on the transaction and their lock. If the created graph has a cycle or closed loop, then there is a deadlock.
- The wait for the graph is maintained by the system for every transaction which is waiting for some data held by the others. The system keeps checking the graph if there is any cycle in the graph.

The wait for a graph for the above scenario is shown below:



Deadlock Prevention

- Deadlock prevention method is suitable for a large database. If the resources are allocated in such a way that deadlock never occurs, then the deadlock can be prevented.
- The Database management system analyzes the operations of the transaction whether they can create a deadlock situation or not. If they do, then the DBMS never allowed that transaction to be executed.

Wait-Die scheme

In this scheme, if a transaction requests for a resource which is already held with a conflicting lock by another transaction then the DBMS simply checks the timestamp of both transactions. It allows the older transaction to wait until the resource is available for execution.

Let's assume there are two transactions Ti and Tj and let TS(T) is a timestamp of any transaction T. If T2 holds a lock by some other transaction and T1 is requesting for resources held by T2 then the following actions are performed by DBMS:

1. Check if TS(Ti) < TS(Tj) - If Ti is the older transaction and Tj has held some resource, then Ti is allowed to wait until the data-item is available for execution. That means if the older transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for resource until it is available.
2. Check if TS(T_i) < TS(Tj) - If Ti is older transaction and has held some resource and if Tj is waiting for it, then Tj is killed and restarted later with the random delay but with the same timestamp.

Wound wait scheme

- In wound wait scheme, if the older transaction requests for a resource which is held by the younger transaction, then older transaction forces younger one to kill the transaction and release the resource. After the minute delay, the younger transaction is restarted but with the same timestamp.
- If the older transaction has held a resource which is requested by the Younger transaction, then the younger transaction is asked to wait until older releases it.

DBMS Concurrency Control

Concurrency Control is the management procedure that is required for controlling concurrent execution of the operations that take place on a database.

But before knowing about concurrency control, we should know about concurrent execution.

Concurrent Execution in DBMS

- In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database. It means that the same database is executed simultaneously on a multi-user system by different users.
- While working on the database transactions, there occurs the requirement of using the database by multiple users for performing different operations, and in that case, concurrent execution of the database is performed.
- The thing is that the simultaneous execution that is performed should be done in an interleaved manner, and no operation should affect the other executing operations, thus maintaining the consistency of the database. Thus, on making the concurrent execution of the transaction operations, there occur several challenging problems that need to be solved.

Problems with Concurrent Execution

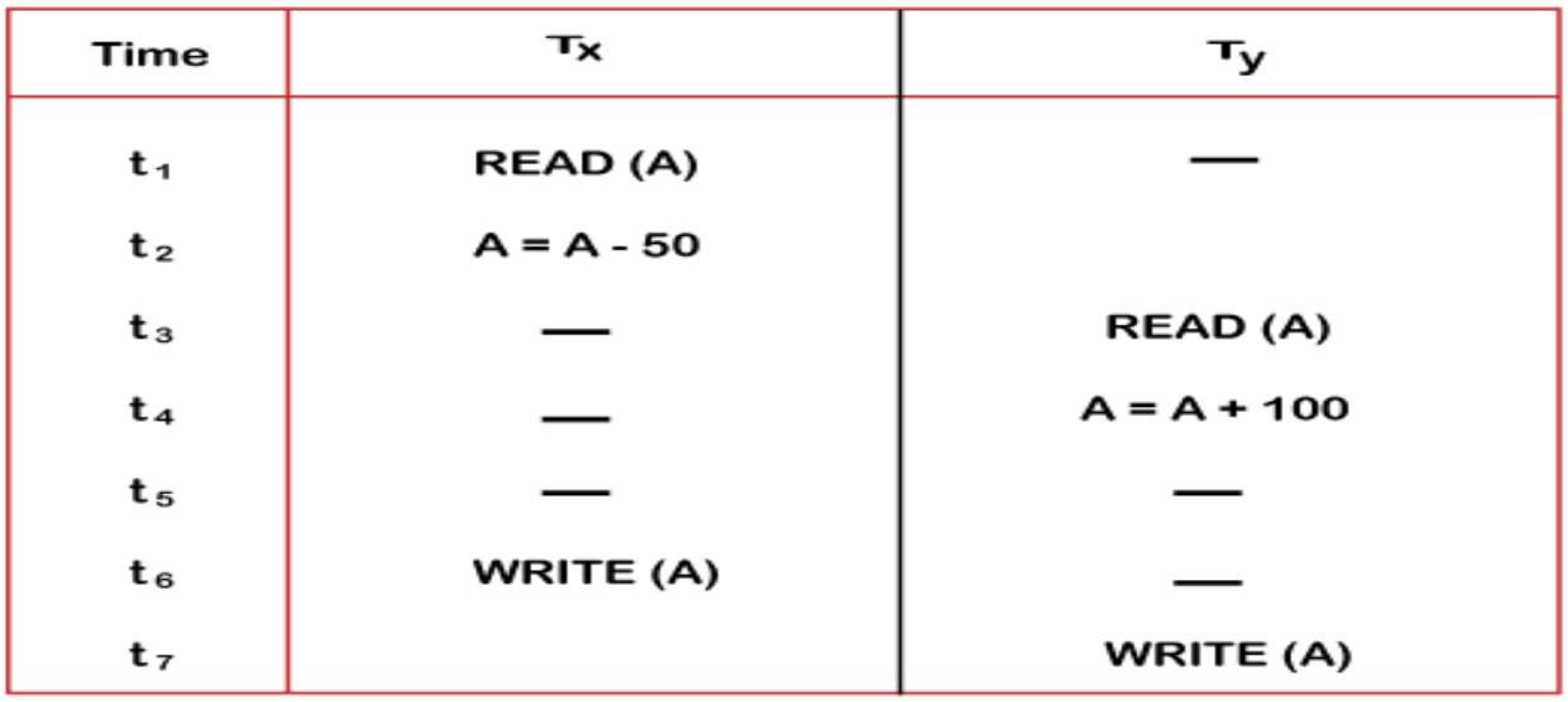
In a database transaction, the two main operations are **READ** and **WRITE** operations. So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, and the data may become inconsistent. So, the following problems occur with the Concurrent Execution of the operations:

Problem 1: Lost Update Problems (W - W Conflict)

The problem occurs *when two different database transactions perform the read/write operations on the same database items in an interleaved manner (i.e., concurrent execution) that makes the values of the items incorrect hence making the database inconsistent*.

For example:

Consider the below diagram where two transactions T_X and T_Y, are performed on the same account A where the balance of account A is $300.



Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A - 50	
t_3	—	READ (A)
t_4	—	A = A + 100
t_5	—	—
t_6	WRITE (A)	—
t_7		WRITE (A)

LOST UPDATE PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300 (only read).
- At time t2, transaction T_X deducts $50 from account A that becomes $250 (only deducted and not updated/write).
- Alternately, at time t3, transaction T_Y reads the value of account A that will be $300 only because T_X didn't update the value yet.
- At time t4, transaction T_Y adds $100 to account A that becomes $400 (only added but not updated/write).
- At time t6, transaction T_X writes the value of account A that will be updated as $250 only, as T_Y didn't update the value yet.
- Similarly, at time t7, transaction T_Y writes the values of account A, so it will write as done at time t4 that will be $400. It means the value written by T_X is lost, i.e., $250 is lost.

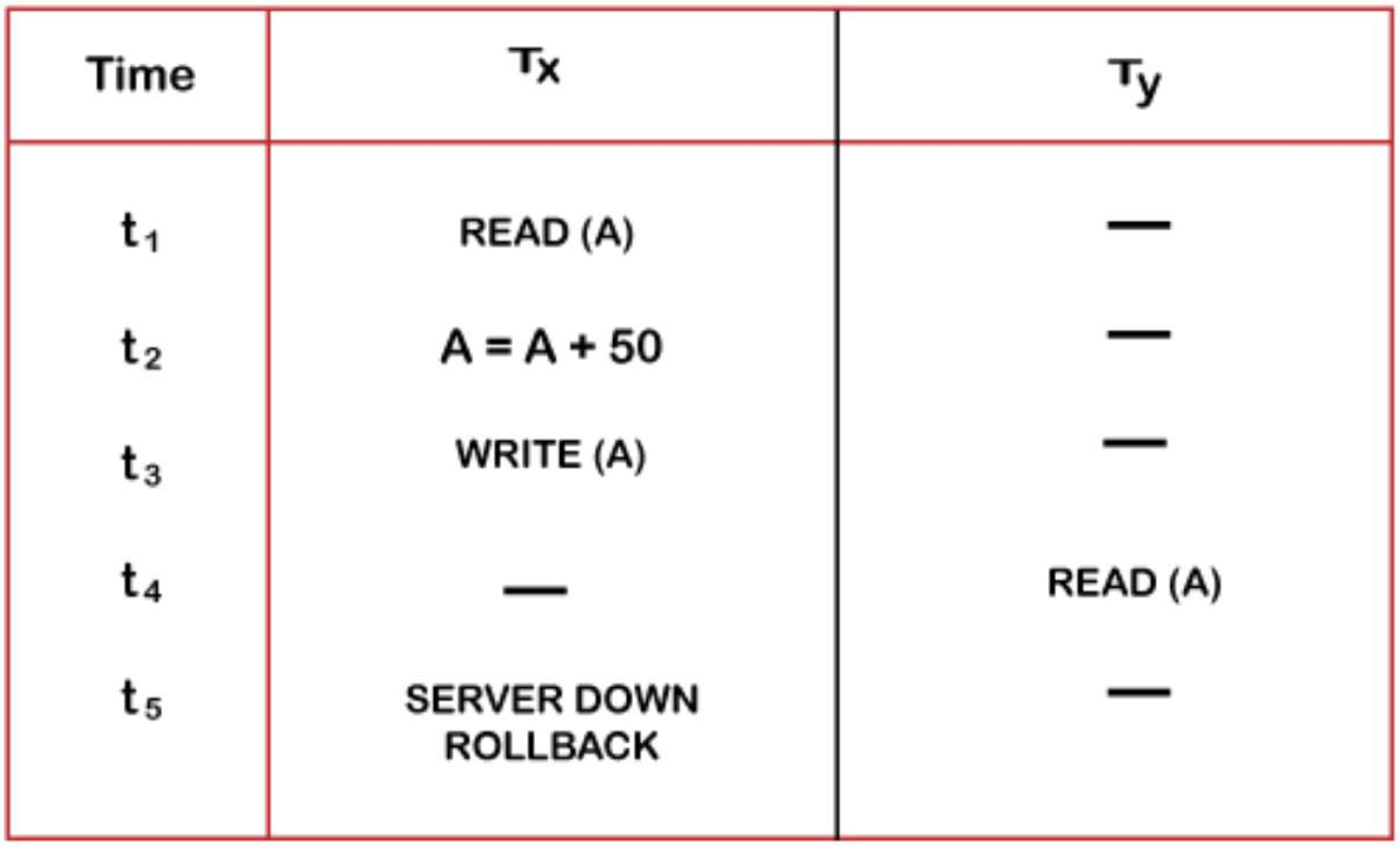
Hence data becomes incorrect, and database sets to inconsistent.

Dirty Read Problems (W-R Conflict)

The dirty read problem occurs when one transaction updates an item of the database, and somehow the transaction fails, and before the data gets rollback, the updated database item is accessed by another transaction. There comes the Read-Write Conflict between both transactions.

For example:

Consider two transactions T_X and T_Y in the below diagram performing read/write operations on account A where the available balance in account A is $300:



Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A + 50	—
t_3	WRITE (A)	—
t_4	—	READ (A)
t_5	SERVER DOWN ROLLBACK	—

DIRTY READ PROBLEM

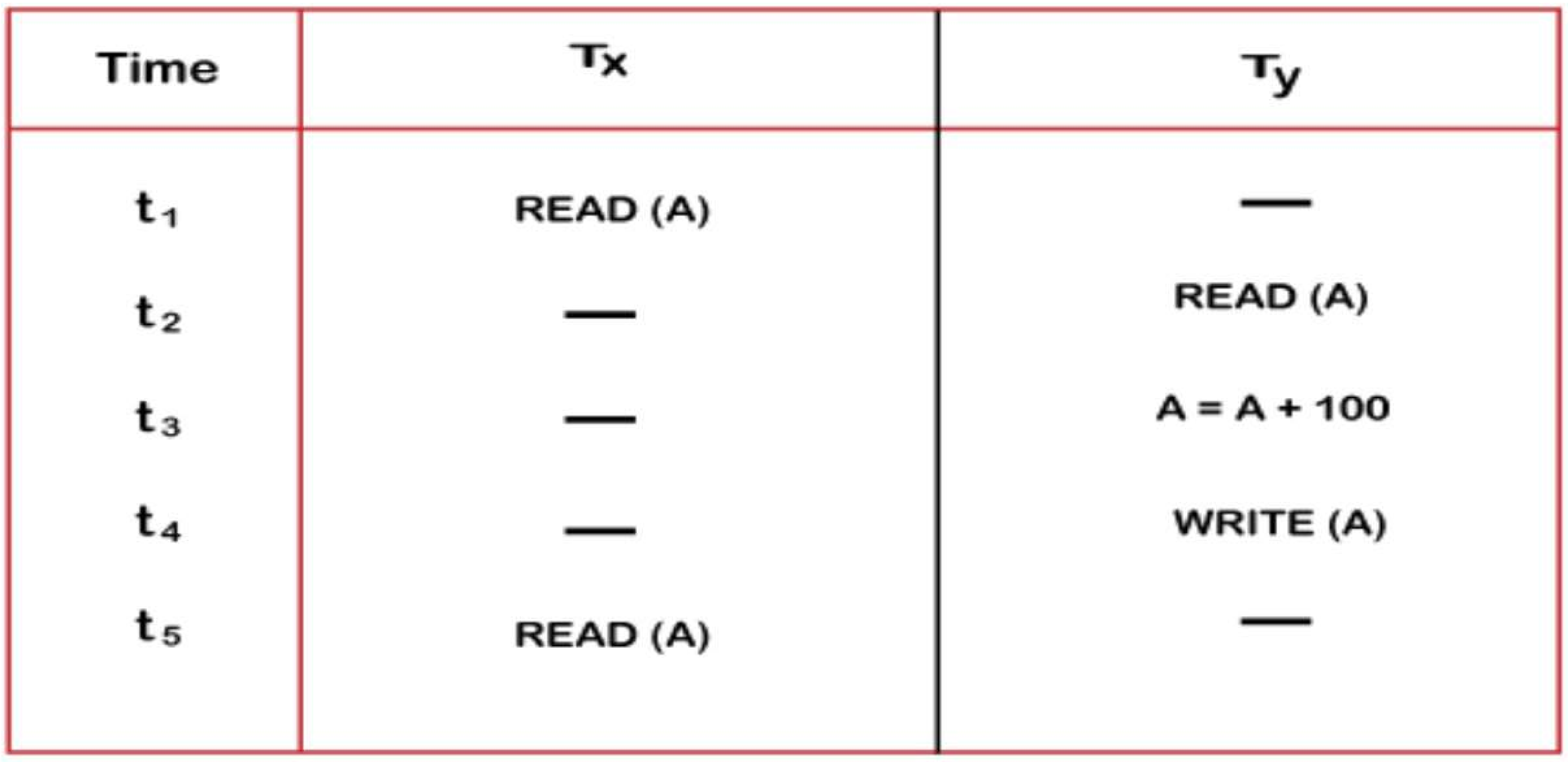
- At time t1, transaction T_X reads the value of account A, i.e., $300.
- At time t2, transaction T_X adds $50 to account A that becomes $350.
- At time t3, transaction T_X writes the updated value in account A, i.e., $350.
- Then at time t4, transaction T_Y reads account A that will be read as $350.
- Then at time t5, transaction T_X rollbacks due to server problem, and the value changes back to $300 (as initially).
- But the value for account A remains $350 for transaction T_Y as committed, which is the dirty read and therefore known as the Dirty Read Problem.

Unrepeatable Read Problem (W-R Conflict)

Also known as Inconsistent Retrievals Problem that occurs when in a transaction, two different values are read for the same database item.

For example:

Consider two transactions, T_X and T_Y, performing the read/write operations on account A, having an available balance = $300. The diagram is shown below:



Time	T_x	T_y
t_1	READ (A)	—
t_2	—	READ (A)
t_3	—	A = A + 100
t_4	—	WRITE (A)
t_5	READ (A)	—

UNREPEATABLE READ PROBLEM

- At time t1, transaction T_X reads the value from account A, i.e., $300.
- At time t2, transaction T_Y reads the value from account A, i.e., $300.
- At time t3, transaction T_Y updates the value of account A by adding $100 to the available balance, and then it becomes $400.
- At time t4, transaction T_Y writes the updated value, i.e., $400.
- After that, at time t5, transaction T_X reads the available value of account A, and that will be read as $400.
- It means that within the same transaction T_X, it reads two different values of account A, i.e., $ 300 initially, and after updation made by transaction T_Y, it reads $400. It is an unrepeatable read and is therefore known as the Unrepeatable read problem.

Thus, in order to maintain consistency in the database and avoid such problems that take place in concurrent execution, management is needed, and that is where the concept of Concurrency Control comes into role.

Concurrency Control

Concurrency Control is the working concept that is required for controlling and managing the concurrent execution of database operations and thus avoiding the inconsistencies in the database. Thus, for maintaining the concurrency of the database, we have the concurrency control protocols.

Concurrency Control Protocols

The concurrency control protocols ensure the atomicity, consistency, isolation, durability and serializability *o*f the concurrent execution of the database transactions. Therefore, these protocols are categorized as:

- Lock Based Concurrency Control Protocol
- Time Stamp Concurrency Control Protocol
- Validation Based Concurrency Control Protocol

We will understand and discuss each protocol one by one in our next sections.

Lock-Based Protocol

In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it. There are two types of lock:

1. Shared lock:

- It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction.
- It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.

2. Exclusive lock:

- In the exclusive lock, the data item can be both reads as well as written by the transaction.
- This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

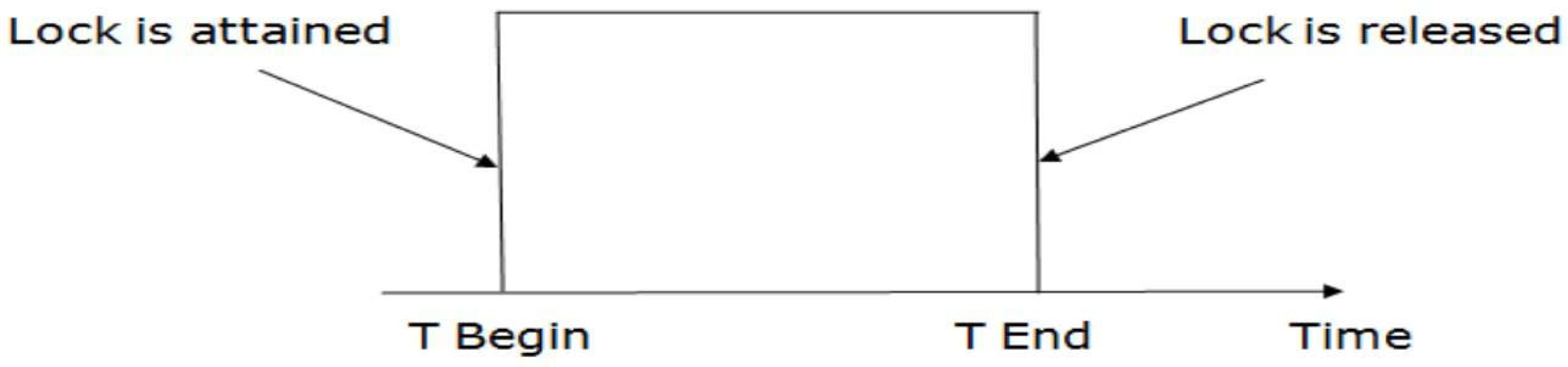
There are four types of lock protocols available:

1. Simplistic lock protocol

It is the simplest way of locking the data while transaction. Simplistic lock-based protocols allow all the transactions to get the lock on the data before insert or delete or update on it. It will unlock the data item after completing the transaction.

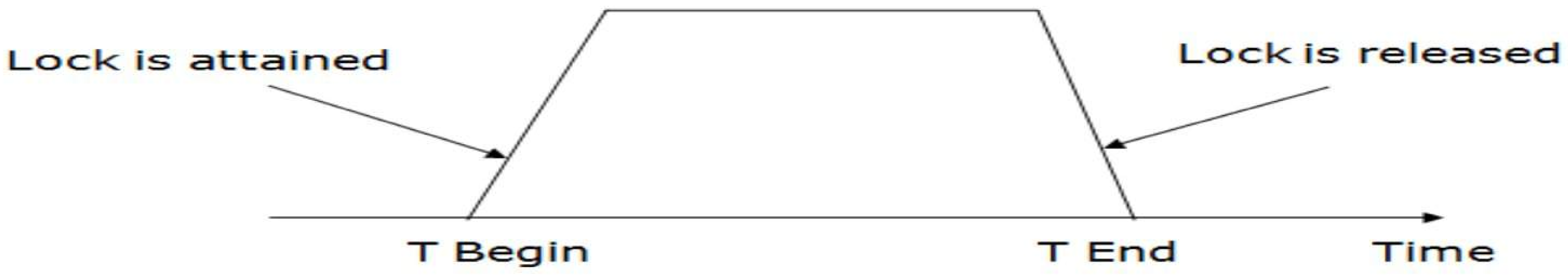
2. Pre-claiming Lock Protocol

- Pre-claiming Lock Protocols evaluate the transaction to list all the data items on which they need locks.
- Before initiating an execution of the transaction, it requests DBMS for all the lock on all those data items.
- If all the locks are granted then this protocol allows the transaction to begin. When the transaction is completed then it releases all the lock.
- If all the locks are not granted then this protocol allows the transaction to rolls back and waits until all the locks are granted.



3. Two-phase locking (2PL)

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



There are two phases of 2PL:

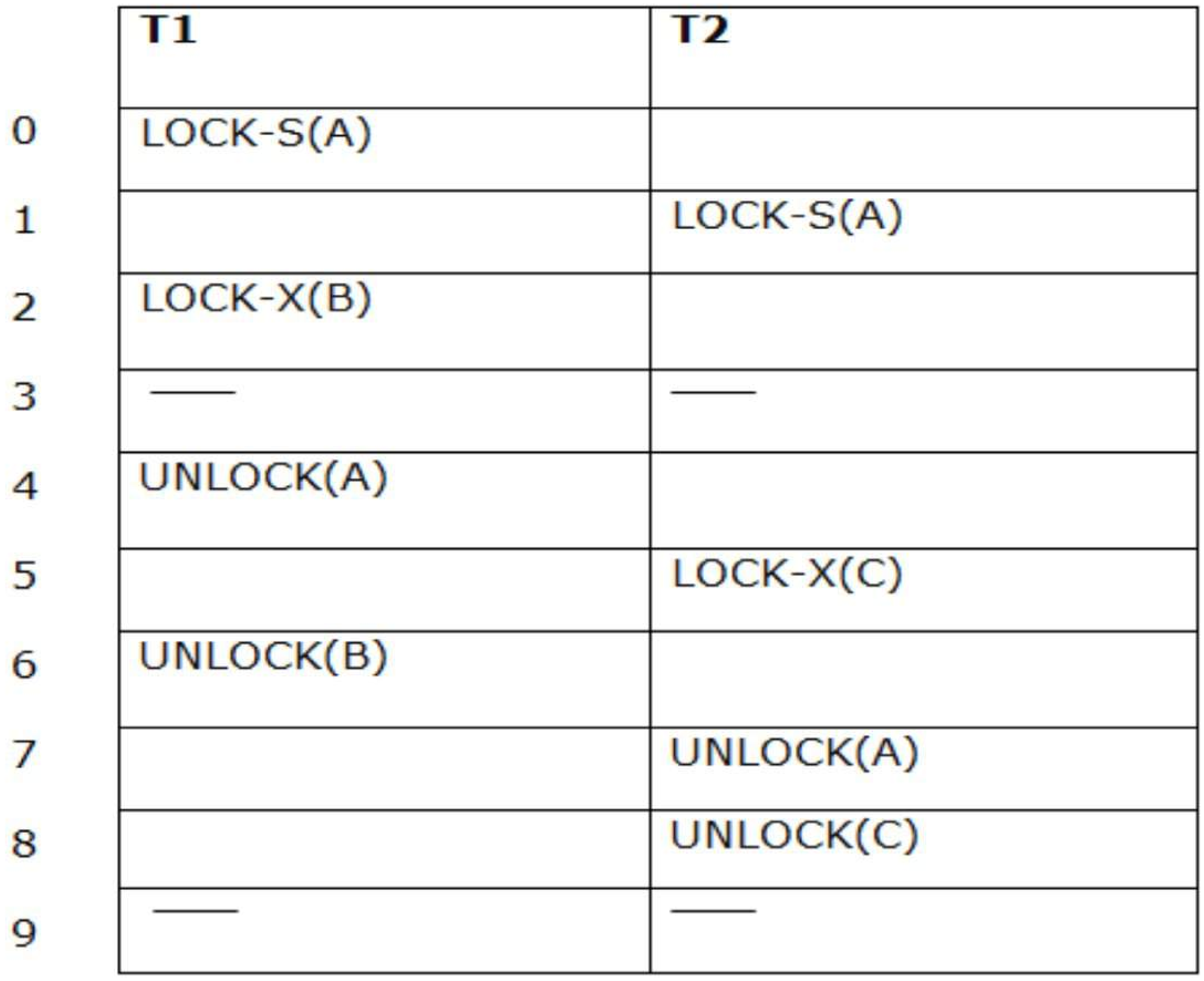
Growing phase: In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.

Shrinking phase: In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.

In the below example, if lock conversion is allowed then the following phase can happen:

1. Upgrading of lock (from S(a) to X (a)) is allowed in growing phase.
2. Downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

Example:



	T1	T2
0	LOCK-S(A)	
1		LOCK-S(A)
2	LOCK-X(B)	
3	——	——
4	UNLOCK(A)	
5		LOCK-X(C)
6	UNLOCK(B)	
7		UNLOCK(A)
8		UNLOCK(C)
9	——	——

The following way shows how unlocking and locking work with 2-PL.

Transaction T1:

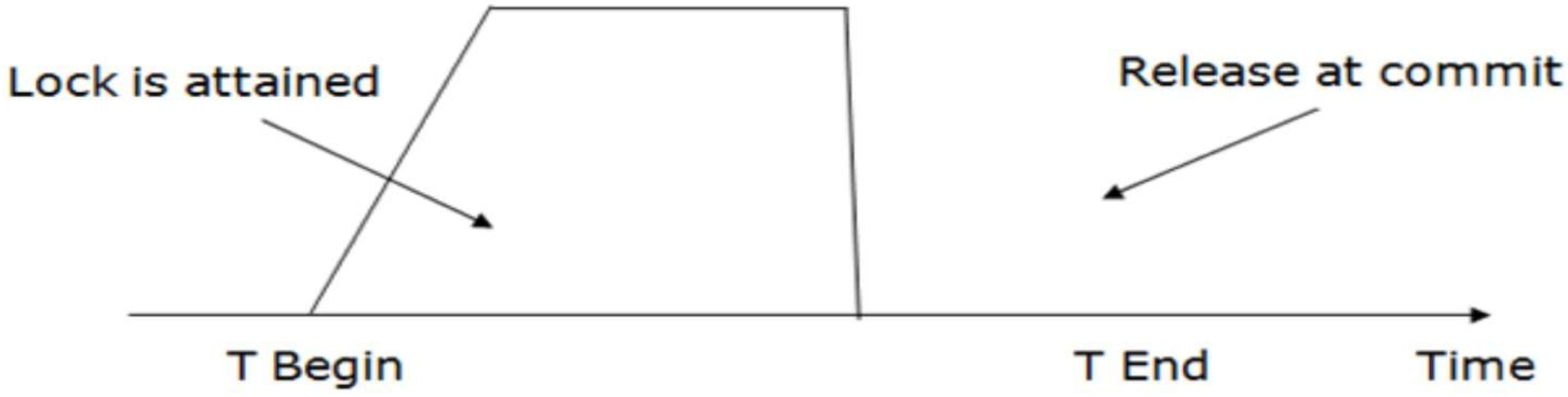
- **Growing phase:** from step 1-3
- **Shrinking phase:** from step 5-7
- **Lock point:** at 3

Transaction T2:

- **Growing phase:** from step 2-6
- **Shrinking phase:** from step 8-9
- **Lock point:** at 6

4. Strict Two-phase locking (Strict-2PL)

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



It does not have cascading abort as 2PL does.

Timestamp Ordering Protocol

- The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.
- The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.

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

Basic Timestamp ordering protocol works as follows:

1. Check the following condition whenever a transaction Ti issues a **Read (X)** operation:

- If W_TS(X) >TS(Ti) then the operation is rejected.
- If W_TS(X) <= TS(Ti) then the operation is executed.
- Timestamps of all the data items are updated.

2. Check the following condition whenever a transaction Ti issues a **Write(X)** operation:

- If TS(Ti) < R_TS(X) then the operation is rejected.
- If TS(Ti) < W_TS(X) then the operation is rejected and Ti is rolled back otherwise the operation is executed.

Where,

TS(TI) denotes the timestamp of the transaction Ti.

R_TS(X) denotes the Read time-stamp of data-item X.

W_TS(X) denotes the Write time-stamp of data-item X.

Advantages and Disadvantages of TO protocol:

- TO protocol ensures serializability since the precedence graph is as follows:

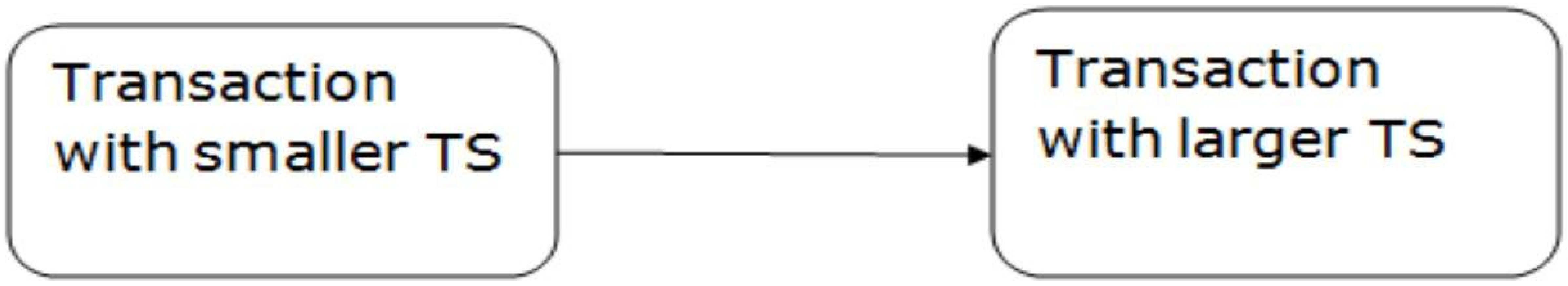


Image: Precedence Graph for TS ordering

- TS protocol ensures freedom from deadlock that means no transaction ever waits.
- But the schedule may not be recoverable and may not even be cascade- free.

Validation Based Protocol

Validation phase is also known as optimistic concurrency control technique. In the validation based protocol, the transaction is executed in the following three phases:

1. **Read phase:** In this phase, the transaction T is read and executed. It is used to read the value of various data items and stores them in temporary local variables. It can perform all the write operations on temporary variables without an update to the actual database.
2. **Validation phase:** In this phase, the temporary variable value will be validated against the actual data to see if it violates the serializability.
3. **Write phase:** If the validation of the transaction is validated, then the temporary results are written to the database or system otherwise the transaction is rolled back.

Here each phase has the following different timestamps:

Start(Ti): It contains the time when Ti started its execution.

Validation (T_i): It contains the time when Ti finishes its read phase and starts its validation phase.

Finish(Ti): It contains the time when Ti finishes its write phase.

- This protocol is used to determine the time stamp for the transaction for serialization using the time stamp of the validation phase, as it is the actual phase which determines if the transaction will commit or rollback.
- Hence TS(T) = validation(T).
- The serializability is determined during the validation process. It can't be decided in advance.
- While executing the transaction, it ensures a greater degree of concurrency and also less number of conflicts.
- Thus it contains transactions which have less number of rollbacks.

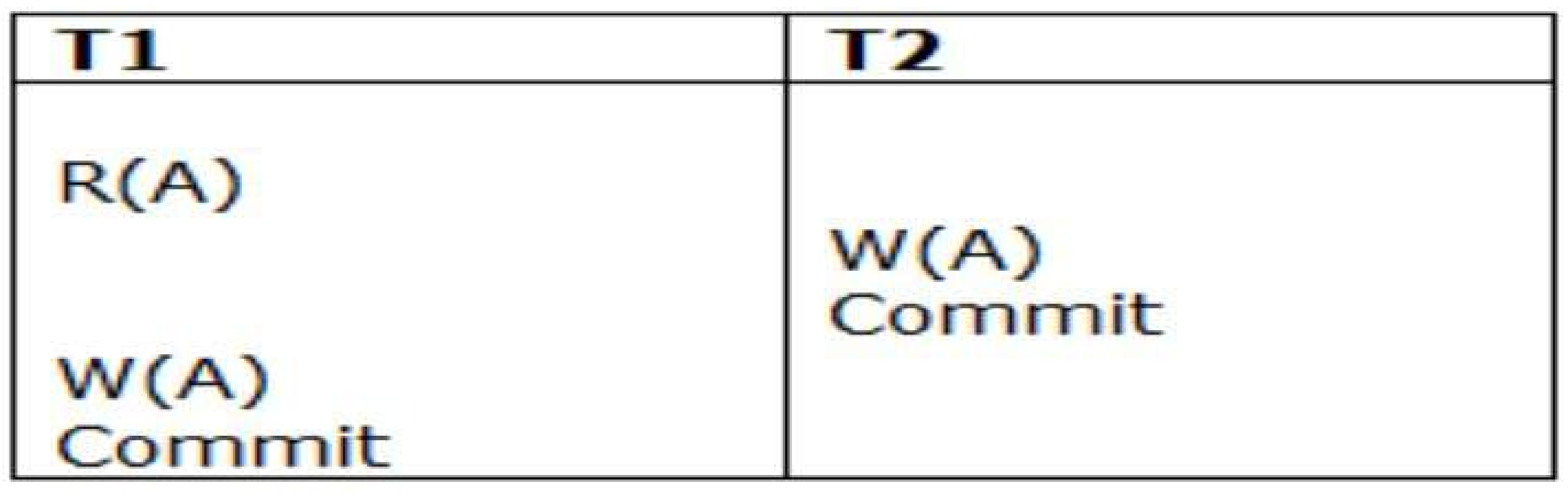
Thomas write Rule

Thomas Write Rule provides the guarantee of serializability order for the protocol. It improves the Basic Timestamp Ordering Algorithm.

The basic Thomas write rules are as follows:

- If TS(T) < R_TS(X) then transaction T is aborted and rolled back, and operation is rejected.
- If TS(T) < W_TS(X) then don't execute the W_item(X) operation of the transaction and continue processing.
- If neither condition 1 nor condition 2 occurs, then allowed to execute the WRITE operation by transaction Ti and set W_TS(X) to TS(T).

If we use the Thomas write rule then some serializable schedule can be permitted that does not conflict serializable as illustrate by the schedule in a given figure:

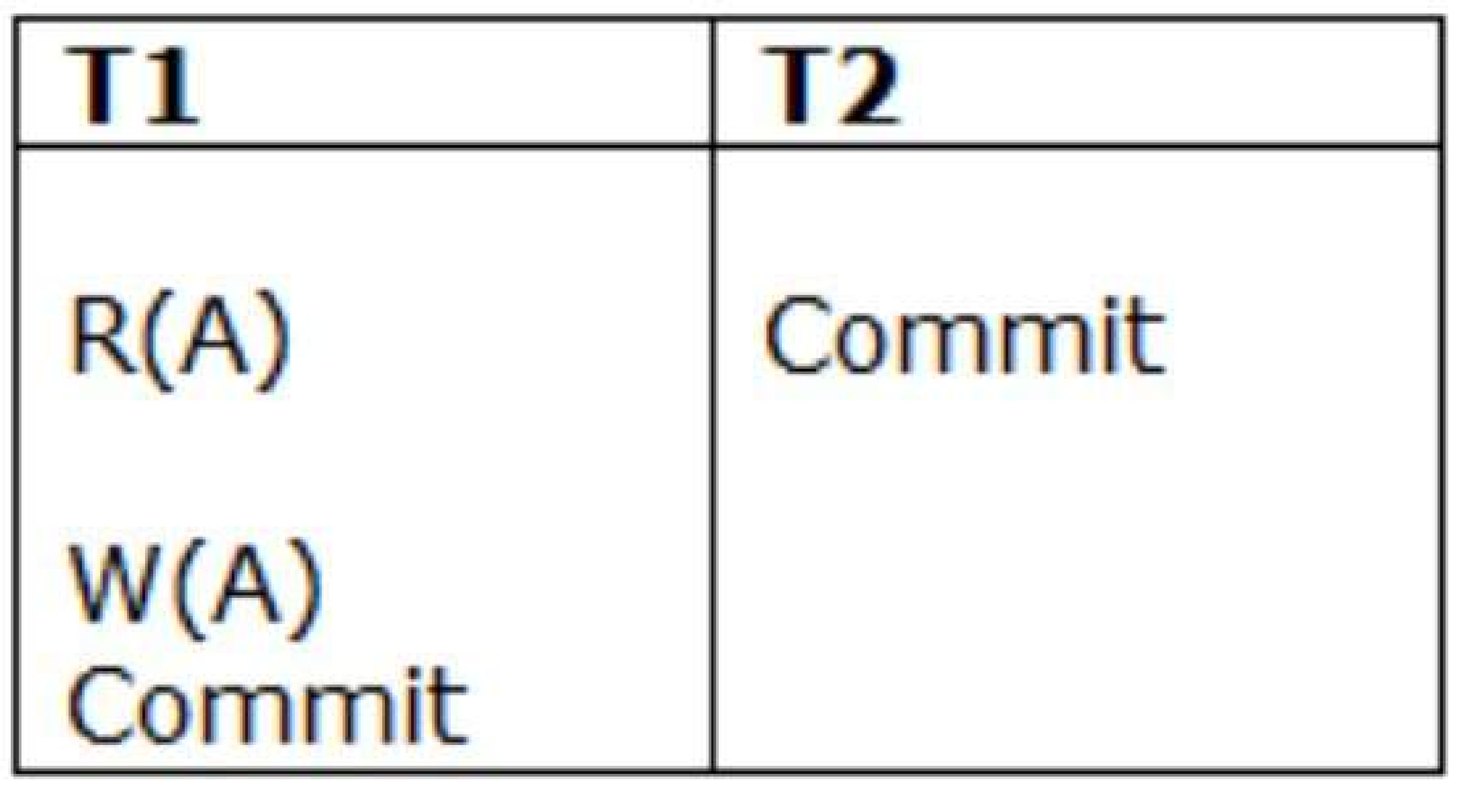


T1	T2
R(A)	
	W(A) Commit
W(A) Commit	

Figure: A Serializable Schedule that is not Conflict Serializable

In the above figure, T1's read and precedes T1's write of the same data item. This schedule does not conflict serializable.

Thomas write rule checks that T2's write is never seen by any transaction. If we delete the write operation in transaction T2, then conflict serializable schedule can be obtained which is shown in below figure.



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

Figure: A Conflict Serializable Schedule

Multiple Granularity

Let's start by understanding the meaning of granularity.

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

Multiple Granularity:

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

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

- The first level or higher level shows the entire database.
- The second level represents a node of type area. The higher level database consists of exactly these areas.
- The area consists of children nodes which are known as files. No file can be present in more than one area.
- Finally, each file contains child nodes known as records. The file has exactly those records that are its child nodes. No records represent in more than one file.
- Hence, the levels of the tree starting from the top level are as follows:
  1. Database
  2. Area
  3. File
  4. Record

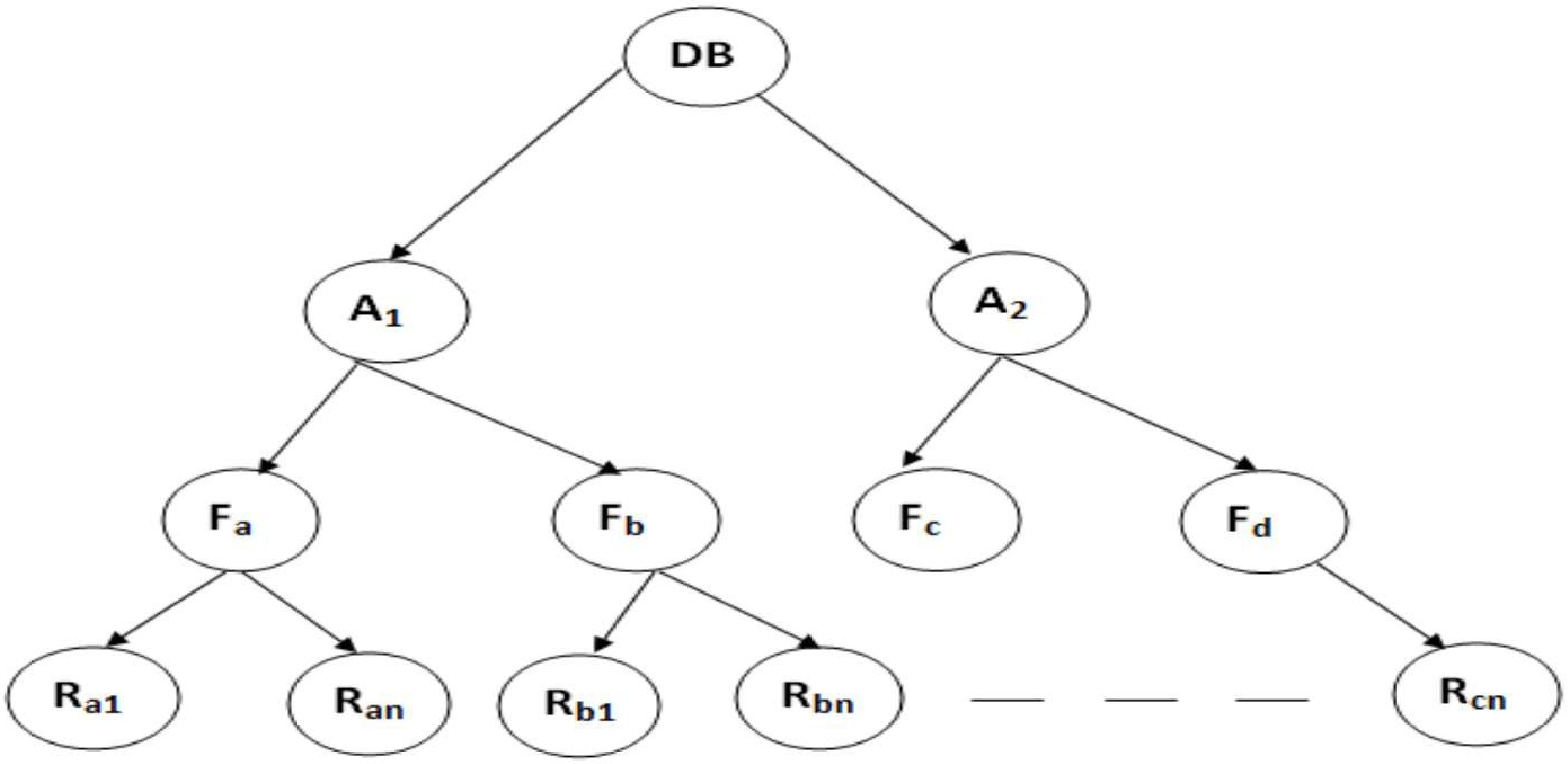


Figure: Multi Granularity tree Hierarchy

In this example, the highest level shows the entire database. The levels below are file, record, and fields.

There are three additional lock modes with multiple granularity:

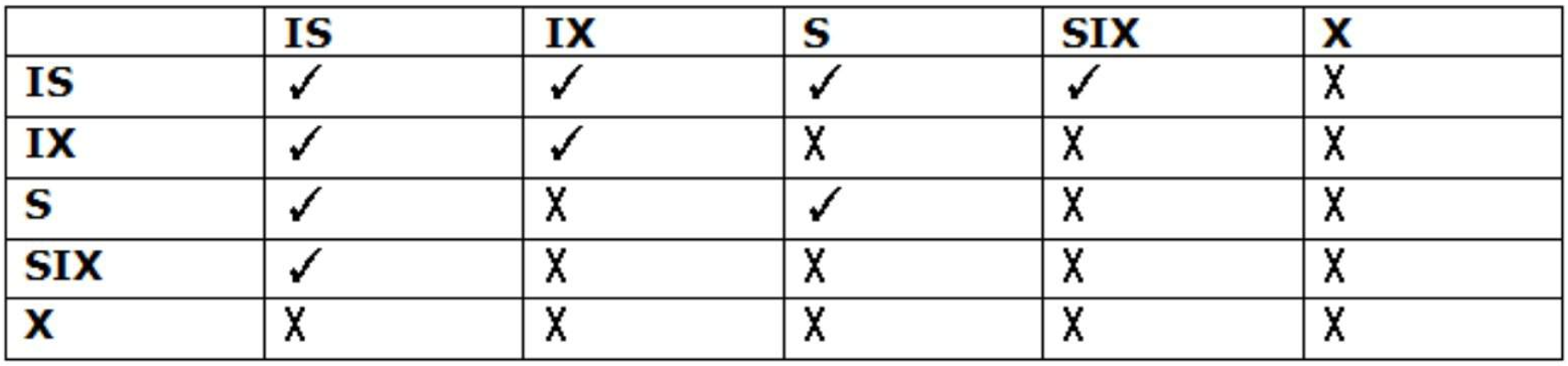
Intention Mode Lock

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

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

Shared & Intention-Exclusive (SIX): In this lock, the node is locked in shared mode, and some node is locked in exclusive mode by the same transaction.

Compatibility Matrix with Intention Lock Modes: The below table describes the compatibility matrix for these lock modes:



	IS	IX	S	SIX	X
IS	✓	✓	✓	✓	X
IX	✓	✓	X	X	X
S	✓	X	✓	X	X
SIX	✓	X	X	X	X
X	X	X	X	X	X

It uses the intention lock modes to ensure serializability. It requires that if a transaction attempts to lock a node, then that node must follow these protocols:

- Transaction T1 should follow the lock-compatibility matrix.
- Transaction T1 firstly locks the root of the tree. It can lock it in any mode.
- If T1 currently has the parent of the node locked in either IX or IS mode, then the transaction T1 will lock a node in S or IS mode only.

- If T1 currently has the parent of the node locked in either IX or SIX modes, then the transaction T1 will lock a node in X, SIX, or IX mode only.
- If T1 has not previously unlocked any node only, then the Transaction T1 can lock a node.
- If T1 currently has none of the children of the node-locked only, then Transaction T1 will unlock a node.

Observe that in multiple-granularity, the locks are acquired in top-down order, and locks must be released in bottom-up order.

- If transaction T1 reads record R_{a9} in file F_a, then transaction T1 needs to lock the database, area A_1 and file F_a in IX mode. Finally, it needs to lock R_{a2} in S mode.
- If transaction T2 modifies record R_{a9} in file F_a, then it can do so after locking the database, area A_1 and file F_a in IX mode. Finally, it needs to lock the R_{a9} in X mode.
- If transaction T3 reads all the records in file F_a, then transaction T3 needs to lock the database, and area A in IS mode. At last, it needs to lock F_a in S mode.
- If transaction T4 reads the entire database, then T4 needs to lock the database in S mode.

Recovery with Concurrent Transaction

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

UNIT-V

Physical Storage System in DBMS

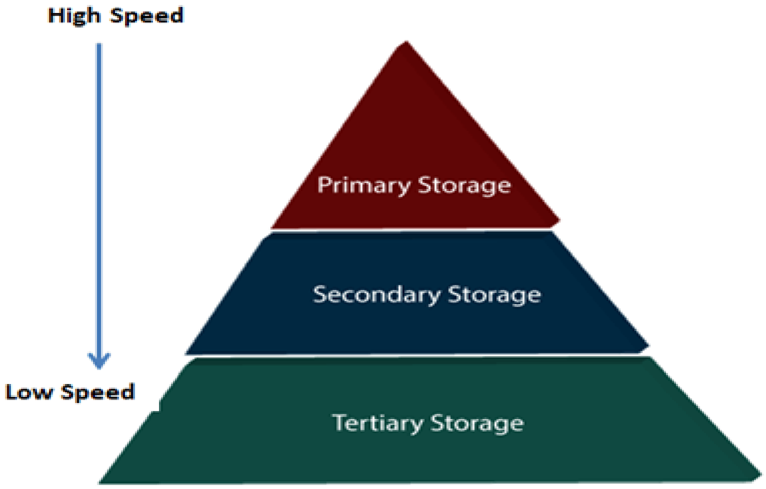
A database system provides an ultimate view of the stored data. However, data in the form of bits, bytes get stored in different storage devices.

In this section, we will take an overview of various types of storage devices that are used for accessing and storing data.

Types of Data Storage

For storing the data, there are different types of storage options available. These storage types differ from one another as per the speed and accessibility. There are the following types of storage devices used for storing the data:

- Primary Storage
- Secondary Storage
- Tertiary Storage



Primary Storage

It is the primary area that offers quick access to the stored data. We also know the primary storage as volatile storage. It is because this type of memory does not permanently store the data. As soon as the system leads to a power cut or a crash, the data also get lost. Main memory and cache are the types of primary storage.

- **Main Memory:** It is the one that is responsible for operating the data that is available by the storage medium. The main memory handles each instruction of a computer machine. This type of memory can store gigabytes of data on a system but is small enough to carry the entire database. At last, the main memory loses the whole content if the system shuts down because of power failure or other reasons.

1. **Cache:** It is one of the costly storage media. On the other hand, it is the fastest one. A cache is a tiny storage media which is maintained by the computer hardware usually. While designing the algorithms and query processors for the data structures, the designers keep concern on the cache effects.

Secondary Storage

Secondary storage is also called as Online storage. It is the storage area that allows the user to save and store data permanently. This type of memory does not lose the data due to any power failure or system crash. That's why we also call it non-volatile storage.

There are some commonly described secondary storage media which are available in almost every type of computer system:

- **Flash Memory:** A flash memory stores data in USB (Universal Serial Bus) keys which are further plugged into the USB slots of a computer system. These USB keys help transfer data to a computer system, but it varies in size limits. Unlike the main memory, it is possible to get back the stored data which may be lost due to a power cut or other reasons. This type of memory storage is most commonly used in the server systems for caching the frequently used data. This leads the systems towards high performance and is capable of storing large amounts of databases than the main memory.

- **Magnetic Disk Storage:** This type of storage media is also known as online storage media. A magnetic disk is used for storing the data for a long time. It is capable of storing an entire database. It is the responsibility of the computer system to make availability of the data from a disk to the main memory for further accessing. Also, if the system performs any operation over the data, the modified data should be written back to the disk. The tremendous capability of a magnetic disk is that it does not affect the data due to a system crash or failure, but a disk failure can easily ruin as well as destroy the stored data.

Tertiary Storage

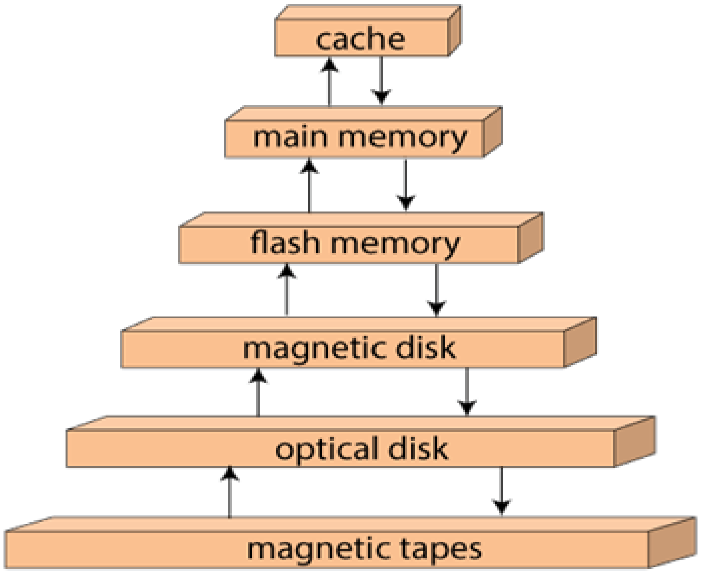
It is the storage type that is external from the computer system. It has the slowest speed. But it is capable of storing a large amount of data. It is also known as Offline storage. Tertiary storage is generally used for data backup. There are following tertiary storage devices available:

- **Optical Storage:** An optical storage can store megabytes or gigabytes of data. A Compact Disk (CD) can store 700 megabytes of data with a playtime of around 80 minutes. On the other hand, a Digital Video Disk or a DVD can store 4.7 or 8.5 gigabytes of data on each side of the disk.
- **Tape Storage:** It is the cheapest storage medium than disks. Generally, tapes are used for archiving or backing up the data. It provides slow access to data as it accesses data sequentially from the start. Thus, tape storage is also known as sequential-access storage. Disk storage is known as direct-access storage as we can directly access the data from any location on disk.

Storage Hierarchy

Besides the above, various other storage devices reside in the computer system. These storage media are organized on the basis of data accessing speed, cost per unit of data to buy the medium, and by medium's reliability. Thus, we can create a hierarchy of storage media on the basis of its cost and speed.

Thus, on arranging the above-described storage media in a hierarchy according to its speed and cost, we conclude the below-described image:



Storage device hierarchy

In the image, the higher levels are expensive but fast. On moving down, the cost per bit is decreasing, and the access time is increasing. Also, the storage media from the main memory to up represents the volatile nature, and below the main memory, all are non-volatile devices.

File Organization Storage

There are different ways of storing data in the database. Storing data in files is one of them. A user can store the data in files in an organized manner. These files are organized logically as a sequence of records and reside permanently on disks. Each file is divided into fixed-length storage units known as **Blocks**. These blocks are the units of storage allocation as well as data transfer. Although the default block size in the database is 4 to 8 kilobytes, many databases allow specifying the size at the time of creating the database instance.

Usually, the record size is smaller than the block size. But, for large data items such as images, the size can vary. For accessing the data quickly, it is required that one complete record should reside in one block only. It should not be partially divided between one or two blocks. In RDBMS, the size of tuples varies in different relations. Thus, we need to structure our files in multiple lengths for implementing the records. In file organization, there are two possible ways of representing the records:

- Fixed-length records
- Variable-length records

Fixed-Length Records

Fixed-length records means setting a length and storing the records into the file. If the record size exceeds the fixed size, it gets divided into more than one block. Due to the fixed size there occurs following two problems:

1. Partially storing subparts of the record in more than one block requires access to all the blocks containing the subparts to read or write in it.
2. It is difficult to delete a record in such a file organization. It is because if the size of the existing record is smaller than the block size, then another record or a part fills up the block.

However, including a certain number of bytes is the solution to the above problems. It is known as **File Header**. The allocated file header carries a variety of information about the file, such as the address of the first record. The address of the second record gets stored in the first record and so on. This process is similar to pointers. The method of insertion and deletion is easy in fixed-length records because the space left or freed by the deleted record is exactly similar to the space required to insert the new records. But this process fails for storing the records of variable lengths.

Variable-Length Record

Variable-length records are the records that vary in size. It requires the creation of multiple blocks of multiple sizes to store them. These variable-length records are kept in the following ways in the database system:

1. Storage of multiple record types in a file.
2. It is kept as Record types that enable repeating fields like multisets or arrays.
3. It is kept as Record types that enable variable lengths either for one field or more.

In variable-length records, there exist the following two problems:

1. Defining the way of representing a single record so as to extract the individual attributes easily.
2. Defining the way of storing variable-length records within a block so as to extract that record in a block easily.

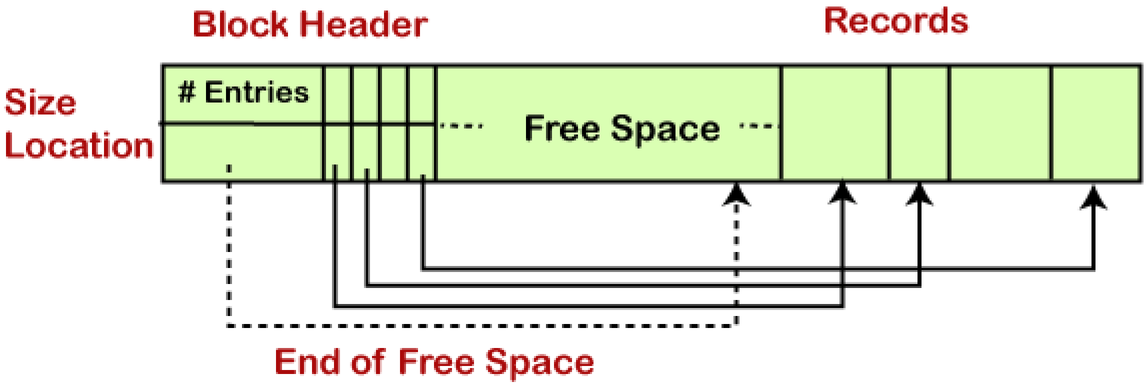
Thus, the representation of a variable-length record can be divided into two parts:

1. An initial part of the record with fixed-length attributes such as numeric values, dates, fixed-length character attributes for storing their value.
2. The data for variable-length attributes such as varchar type is represented in the initial part of the record by (offset, length) pair. The offset refers to the place where that record begins, and length refers to the length of the variable-size attribute. Thus, the initial part stores fixed-size information about each attribute, i.e., whether it is the fixed-length or variable-length attribute.

Slotted-page Structure

There occurs a problem to store variable-length records within the block. Thus, such records are organized in a slotted-page structure within the block. In the slotted-page structure, a header is present at the starting of each block. This header holds information such as:

1. The number of record entries in the header
2. No free space remaining in the block
3. An array containing the information on the location and size of the records.



Slotted-page structure

Inserting and Deleting Method

The variable-length records reside in a contiguous manner within the block.

When a new record is to be inserted, it gets the place at the end of the free space. It is because free space is contiguous as well. Also, the header fills an entry with the size and location information of the newly inserted record.

When an existing record is deleted, space is freed, and the header entry sets to deleted. Before deleting, it moves the record and occupies it to create the free space. The end-of-free-space gets the update. Then all the free space again sets between the first record and the final entry.

The primary technique of the slotted-page structure is that no pointer should directly point the record. Instead, it should point to the header entry that contains the information of its location. This stops fragmentation of space inside the block but supports indirect pointers to the record.

File Organization

- The **File** is a collection of records. Using the primary key, we can access the records. The type and frequency of access can be determined by the type of file organization which was used for a given set of records.
- File organization is a logical relationship among various records. This method defines how file records are mapped onto disk blocks.
- File organization is used to describe the way in which the records are stored in terms of blocks, and the blocks are placed on the storage medium.
- The first approach to map the database to the file is to use the several files and store only one fixed length record in any given file. An alternative approach is to structure our files so that we can contain multiple lengths for records.
- Files of fixed length records are easier to implement than the files of variable length records.

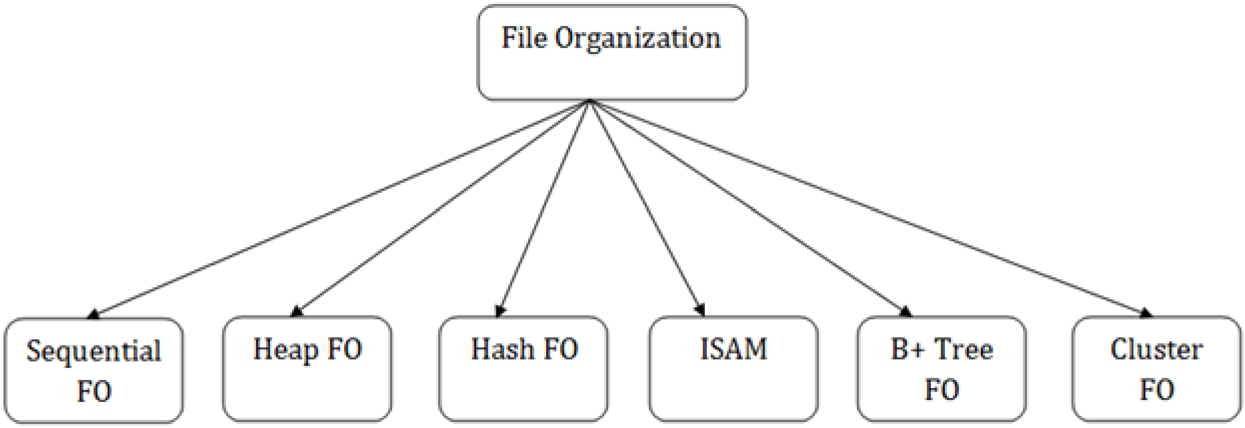
Objective of file organization

- It contains an optimal selection of records, i.e., records can be selected as fast as possible.
- To perform insert, delete or update transaction on the records should be quick and easy.
- The duplicate records cannot be induced as a result of insert, update or delete.
- For the minimal cost of storage, records should be stored efficiently.

Types of file organization:

File organization contains various methods. These particular methods have pros and cons on the basis of access or selection. In the file organization, the programmer decides the best-suited file organization method according to his requirement.

Types of file organization are as follows:



- Sequential file organization
- Heap file organization
- Hash file organization
- B+ file organization
- Indexed sequential access method (ISAM)
- Cluster file organization

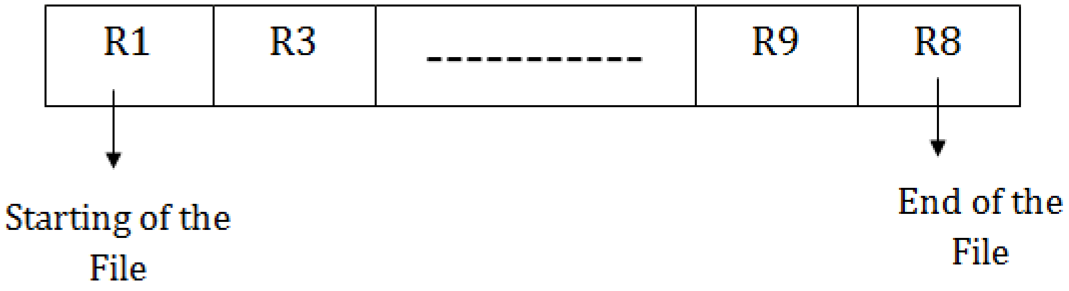
Sequential File Organization

This method is the easiest method for file organization. In this method, files are stored sequentially. This method can be implemented in two ways:

1. Pile File Method:

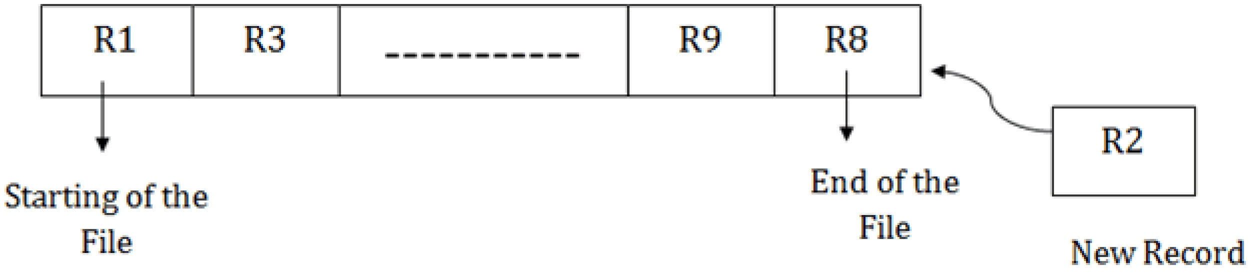
- It is a quite simple method. In this method, we store the record in a sequence, i.e., one after another. Here, the record will be inserted in the order in which they are inserted into tables.

- In case of updating or deleting of any record, the record will be searched in the memory blocks. When it is found, then it will be marked for deleting, and the new record is inserted.



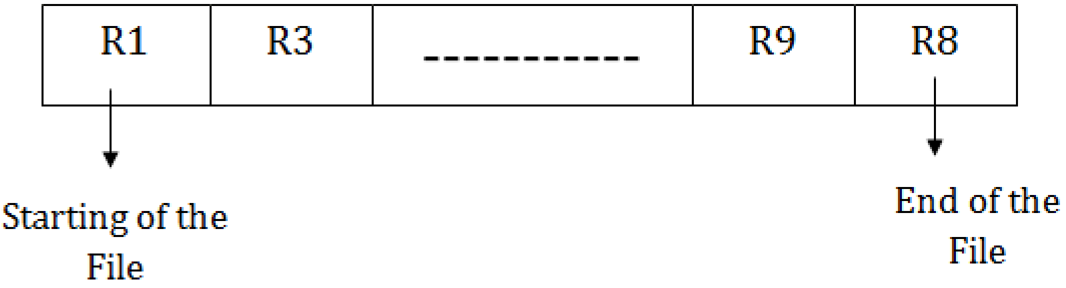
Insertion of the new record:

Suppose we have four records R1, R3 and so on upto R9 and R8 in a sequence. Hence, records are nothing but a row in the table. Suppose we want to insert a new record R2 in the sequence, then it will be placed at the end of the file. Here, records are nothing but a row in any table.



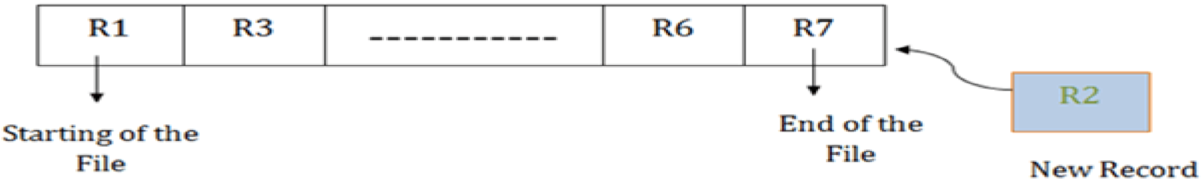
2. Sorted File Method:

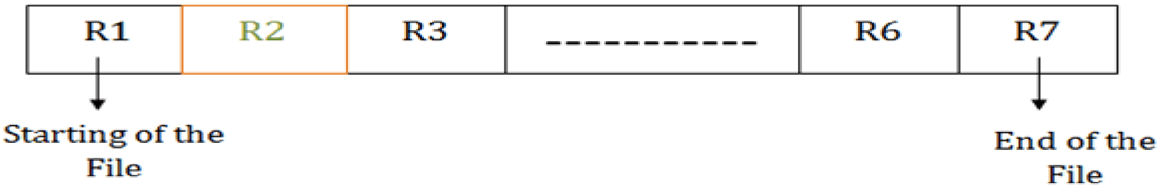
- In this method, the new record is always inserted at the file's end, and then it will sort the sequence in ascending or descending order. Sorting of records is based on any primary key or any other key.
- In the case of modification of any record, it will update the record and then sort the file, and lastly, the updated record is placed in the right place.



Insertion of the new record:

Suppose there is a preexisting sorted sequence of four records R1, R3 and so on upto R6 and R7. Suppose a new record R2 has to be inserted in the sequence, then it will be inserted at the end of the file, and then it will sort the sequence.





Pros of sequential file organization

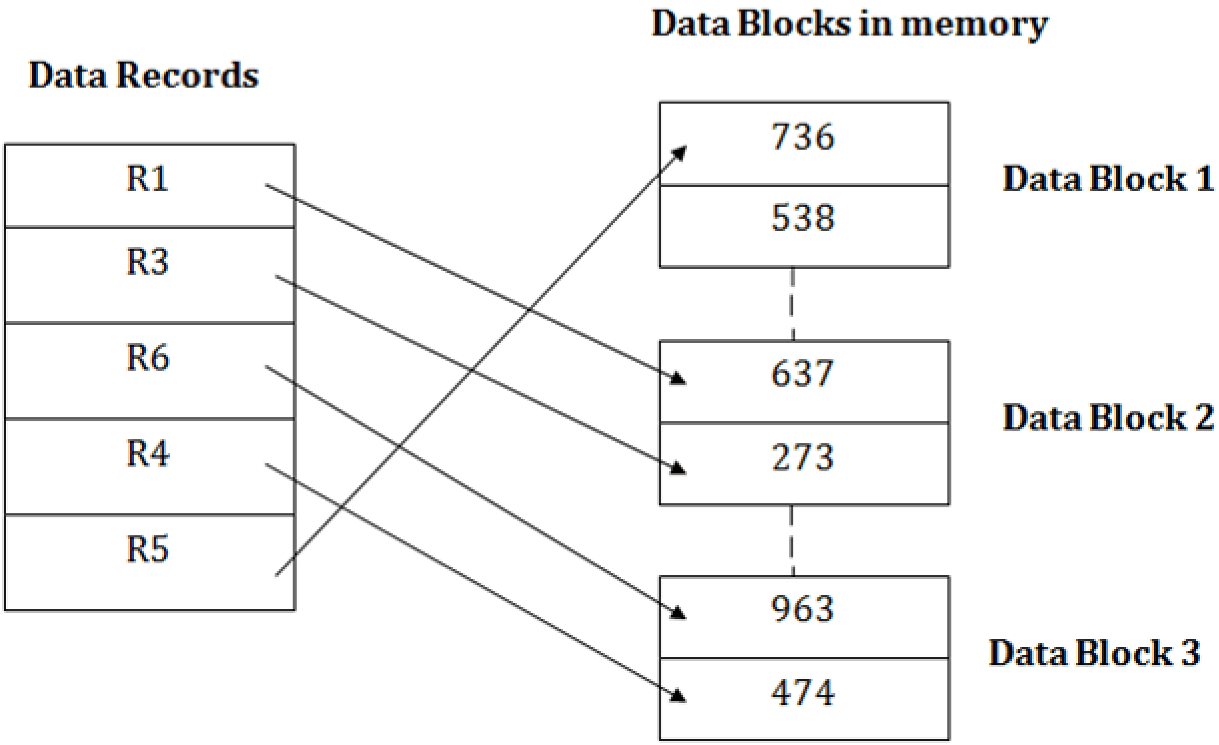
- It contains a fast and efficient method for the huge amount of data.
- In this method, files can be easily stored in cheaper storage mechanism like magnetic tapes.
- It is simple in design. It requires no much effort to store the data.
- This method is used when most of the records have to be accessed like grade calculation of a student, generating the salary slip, etc.
- This method is used for report generation or statistical calculations.

Cons of sequential file organization

- It will waste time as we cannot jump on a particular record that is required but we have to move sequentially which takes our time.
- Sorted file method takes more time and space for sorting the records.

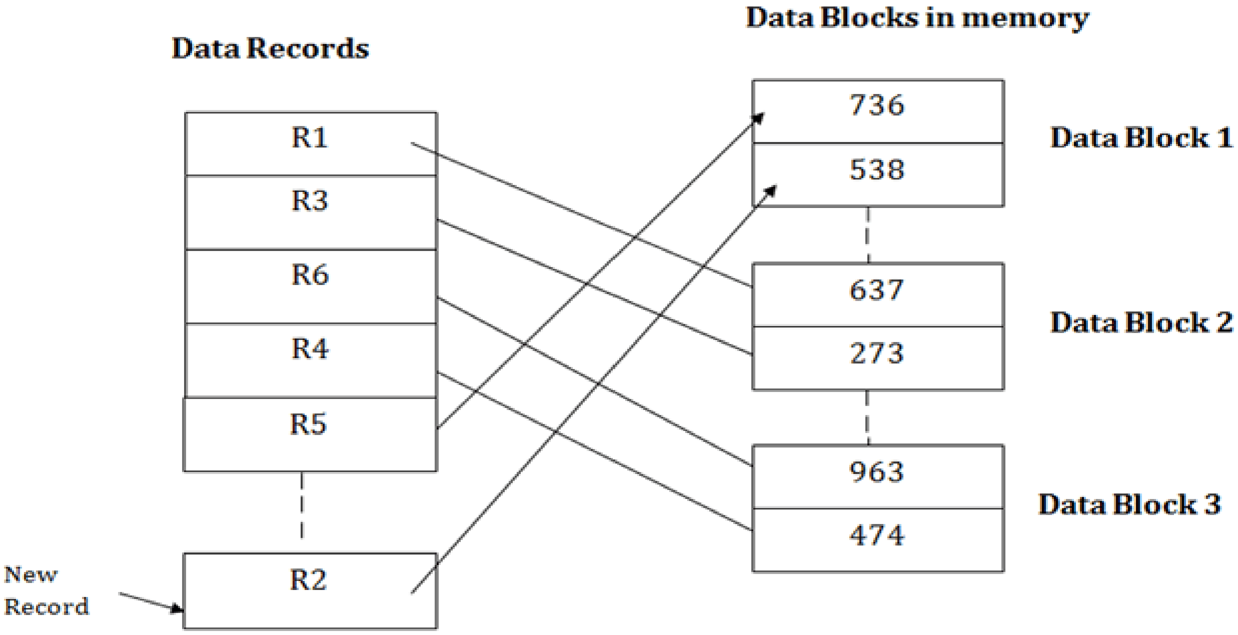
Heap file organization

- It is the simplest and most basic type of organization. It works with data blocks. In heap file organization, the records are inserted at the file's end. When the records are inserted, it doesn't require the sorting and ordering of records.
- When the data block is full, the new record is stored in some other block. This new data block need not to be the very next data block, but it can select any data block in the memory to store new records. The heap file is also known as an unordered file.
- In the file, every record has a unique id, and every page in a file is of the same size. It is the DBMS responsibility to store and manage the new records.



Insertion of a new record

Suppose we have five records R1, R3, R6, R4 and R5 in a heap and suppose we want to insert a new record R2 in a heap. If the data block 3 is full then it will be inserted in any of the database selected by the DBMS, let's say data block 1.



If we want to search, update or delete the data in heap file organization, then we need to traverse the data from staring of the file till we get the requested record.

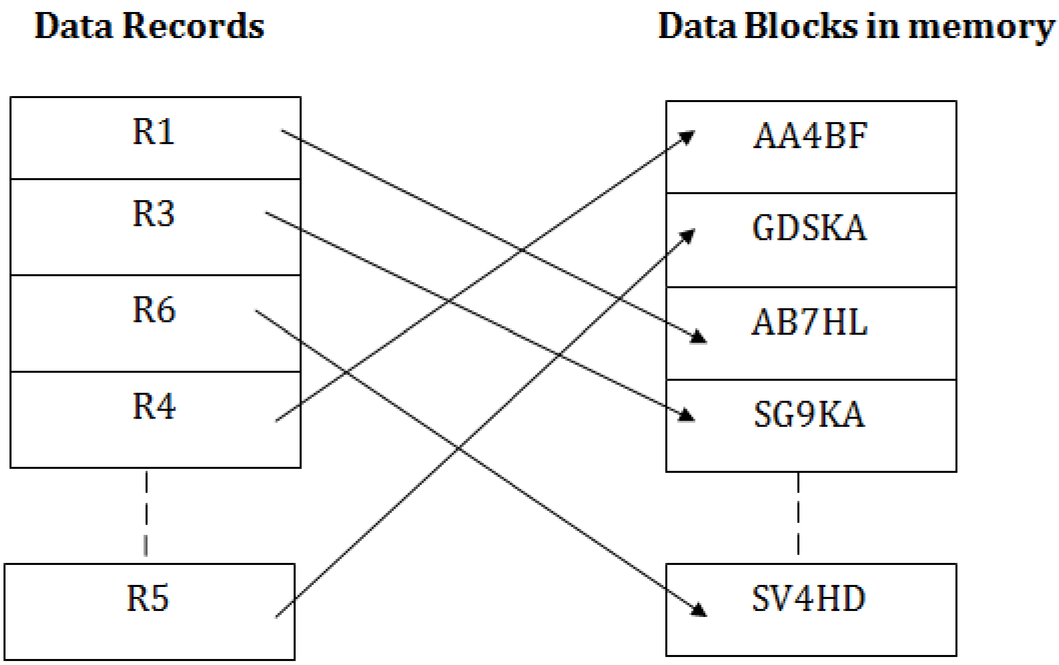
If the database is very large then searching, updating or deleting of record will be time-consuming because there is no sorting or ordering of records. In the heap file organization, we need to check all the data until we get the requested record.

Pros of Heap file organization

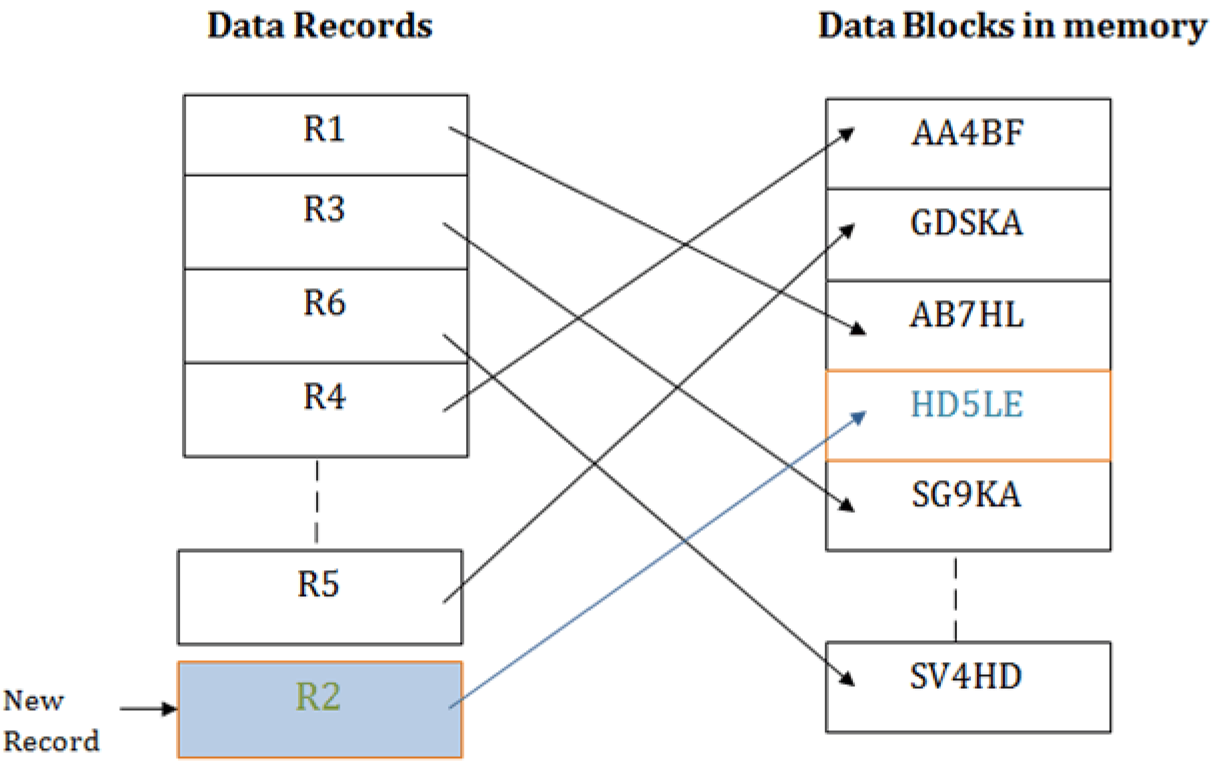
- It is a very good method of file organization for bulk insertion. If there is a large number of data which needs to load into the database at a time, then this method is best suited.
- In case of a small database, fetching and retrieving of records is faster than the sequential record.

Cons of Heap file organization

- This method is inefficient for the large database because it takes time to search or modify the record.
- This method is inefficient for large databases.
- Hash File Organization
- Hash File Organization uses the computation of hash function on some fields of the records. The hash function's output determines the location of disk block where the records are to be placed.

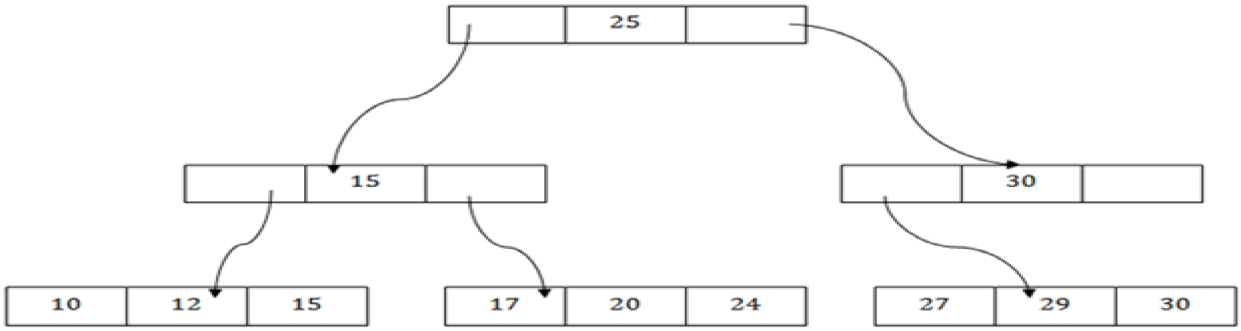


- When a record has to be received using the hash key columns, then the address is generated, and the whole record is retrieved using that address. In the same way, when a new record has to be inserted, then the address is generated using the hash key and record is directly inserted. The same process is applied in the case of delete and update.
- In this method, there is no effort for searching and sorting the entire file. In this method, each record will be stored randomly in the memory.

- 

B+ File Organization

- B+ tree file organization is the advanced method of an indexed sequential access method. It uses a tree-like structure to store records in File.
- It uses the same concept of key-index where the primary key is used to sort the records. For each primary key, the value of the index is generated and mapped with the record.
- The B+ tree is similar to a binary search tree (BST), but it can have more than two children. In this method, all the records are stored only at the leaf node. Intermediate nodes act as a pointer to the leaf nodes. They do not contain any records.



The above B+ tree shows that:

- There is one root node of the tree, i.e., 25.
- There is an intermediary layer with nodes. They do not store the actual record. They have only pointers to the leaf node.
- The nodes to the left of the root node contain the prior value of the root and nodes to the right contain next value of the root, i.e., 15 and 30 respectively.
- There is only one leaf node which has only values, i.e., 10, 12, 17, 20, 24, 27 and 29.
- Searching for any record is easier as all the leaf nodes are balanced.
- In this method, searching any record can be traversed through the single path and accessed easily.

Pros of B+ tree file organization

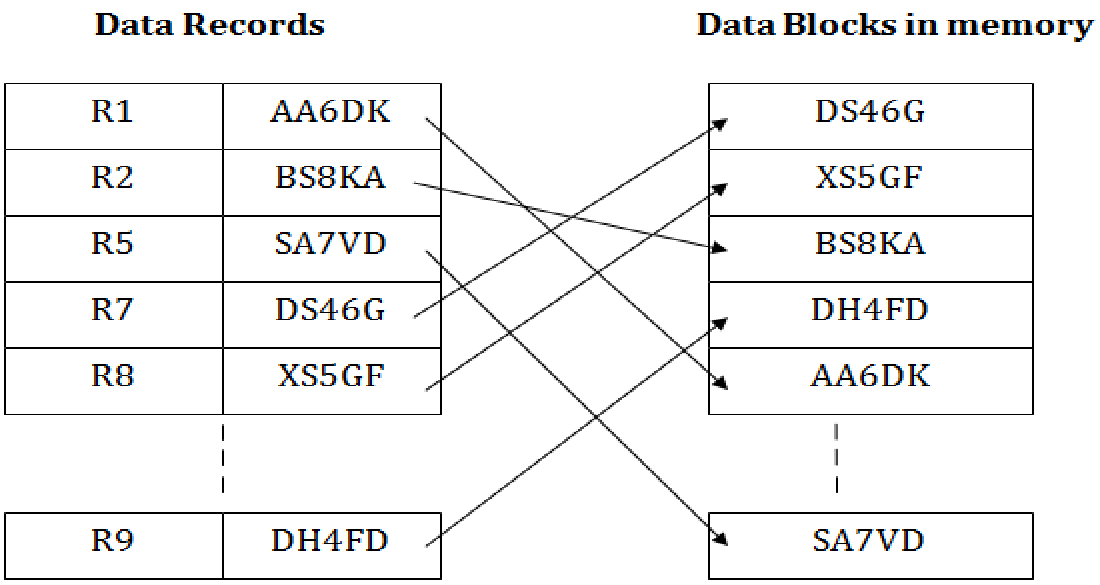
- In this method, searching becomes very easy as all the records are stored only in the leaf nodes and sorted the sequential linked list.
- Traversing through the tree structure is easier and faster.
- The size of the B+ tree has no restrictions, so the number of records can increase or decrease and the B+ tree structure can also grow or shrink.
- It is a balanced tree structure, and any insert/update/delete does not affect the performance of tree.

Cons of B+ tree file organization

- This method is inefficient for the static method.

Indexed sequential access method (ISAM)

ISAM method is an advanced sequential file organization. In this method, records are stored in the file using the primary key. An index value is generated for each primary key and mapped with the record. This index contains the address of the record in the file.



If any record has to be retrieved based on its index value, then the address of the data block is fetched and the record is retrieved from the memory.

Pros of ISAM:

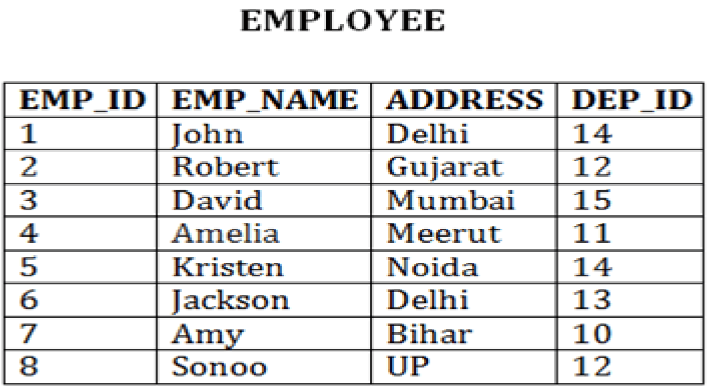
- In this method, each record has the address of its data block, searching a record in a huge database is quick and easy.
- This method supports range retrieval and partial retrieval of records. Since the index is based on the primary key values, we can retrieve the data for the given range of value. In the same way, the partial value can also be easily searched, i.e., the student name starting with 'JA' can be easily searched.

Cons of ISAM

- This method requires extra space in the disk to store the index value.
- When the new records are inserted, then these files have to be reconstructed to maintain the sequence.
- When the record is deleted, then the space used by it needs to be released. Otherwise, the performance of the database will slow down.

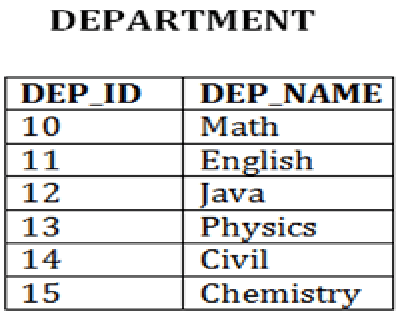
Cluster file organization

- When the two or more records are stored in the same file, it is known as clusters. These files will have two or more tables in the same data block, and key attributes which are used to map these tables together are stored only once.
- This method reduces the cost of searching for various records in different files.
- The cluster file organization is used when there is a frequent need for joining the tables with the same condition. These joins will give only a few records from both tables. In the given example, we are retrieving the record for only particular departments. This method can't be used to retrieve the record for the entire department.



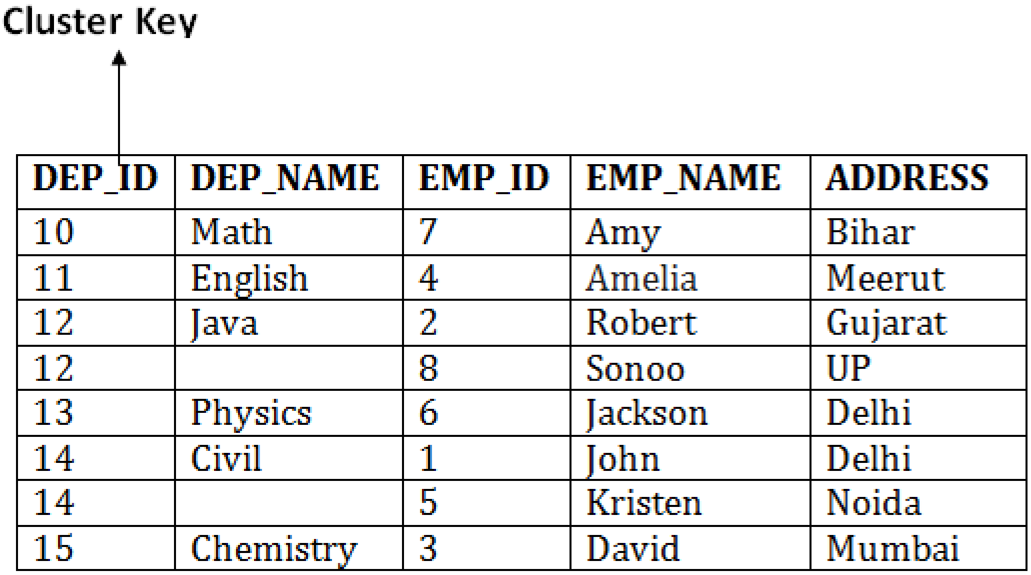
EMPLOYEE

EMP_ID	EMP_NAME	ADDRESS	DEP_ID
1	John	Delhi	14
2	Robert	Gujarat	12
3	David	Mumbai	15
4	Amelia	Meerut	11
5	Kristen	Noida	14
6	Jackson	Delhi	13
7	Amy	Bihar	10
8	Sonoo	UP	12



DEPARTMENT

DEP_ID	DEP_NAME
10	Math
11	English
12	Java
13	Physics
14	Civil
15	Chemistry


Cluster Key

DEP_ID	DEP_NAME	EMP_ID	EMP_NAME	ADDRESS
10	Math	7	Amy	Bihar
11	English	4	Amelia	Meerut
12	Java	2	Robert	Gujarat
12		8	Sonoo	UP
13	Physics	6	Jackson	Delhi
14	Civil	1	John	Delhi
14		5	Kristen	Noida
15	Chemistry	3	David	Mumbai

In this method, we can directly insert, update or delete any record. Data is sorted based on the key with which searching is done. Cluster key is a type of key with which joining of the table is performed.

Types of Cluster file organization:

Cluster file organization is of two types:

1. Indexed Clusters:

In indexed cluster, records are grouped based on the cluster key and stored together. The above EMPLOYEE and DEPARTMENT relationship is an example of an indexed cluster. Here, all the records are grouped based on the cluster key- DEP_ID and all the records are grouped.

2. Hash Clusters:

It is similar to the indexed cluster. In hash cluster, instead of storing the records based on the cluster key, we generate the value of the hash key for the cluster key and store the records with the same hash key value.

Pros of Cluster file organization

- The cluster file organization is used when there is a frequent request for joining the tables with same joining condition.
- It provides the efficient result when there is a 1:M mapping between the tables.

Cons of Cluster file organization

- This method has the low performance for the very large database.
- If there is any change in joining condition, then this method cannot use. If we change the condition of joining then traversing the file takes a lot of time.
- This method is not suitable for a table with a 1:1 condition.

Indexing in DBMS

- Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.
- The index is a type of data structure. It is used to locate and access the data in a database table quickly.

Index structure:

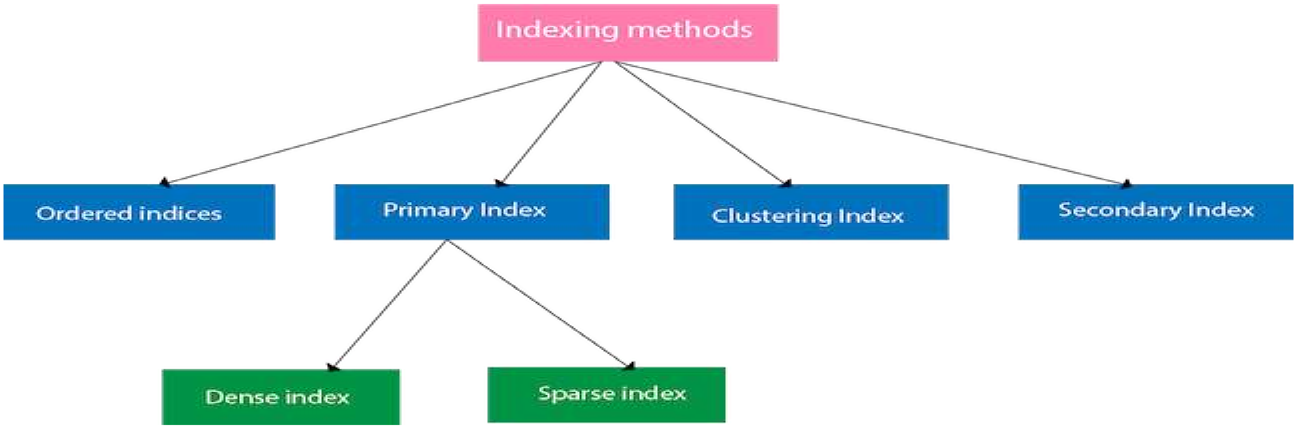
Indexes can be created using some database columns.

Search key	Data Reference

Fig: Structure of Index

- The first column of the database is the search key that contains a copy of the primary key or candidate key of the table. The values of the primary key are stored in sorted order so that the corresponding data can be accessed easily.
- The second column of the database is the data reference. It contains a set of pointers holding the address of the disk block where the value of the particular key can be found.

Indexing Methods



Ordered indices

The indices are usually sorted to make searching faster. The indices which are sorted are known as ordered indices.

Example: Suppose we have an employee table with thousands of record and each of which is 10 bytes long. If their IDs start with 1, 2, 3....and so on and we have to search student with ID-543.

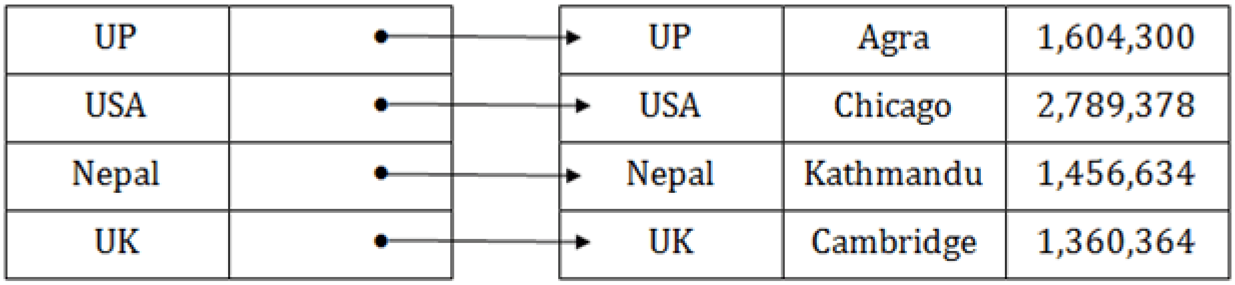
- In the case of a database with no index, we have to search the disk block from starting till it reaches 543. The DBMS will read the record after reading 543*10=5430 bytes.
- In the case of an index, we will search using indexes and the DBMS will read the record after reading 542*2= 1084 bytes which are very less compared to the previous case.

Primary Index

- If the index is created on the basis of the primary key of the table, then it is known as primary indexing. These primary keys are unique to each record and contain 1:1 relation between the records.
- As primary keys are stored in sorted order, the performance of the searching operation is quite efficient.
- The primary index can be classified into two types: Dense index and Sparse index.

Dense index

- The dense index contains an index record for every search key value in the data file. It makes searching faster.
- In this, the number of records in the index table is same as the number of records in the main table.
- It needs more space to store index record itself. The index records have the search key and a pointer to the actual record on the disk.

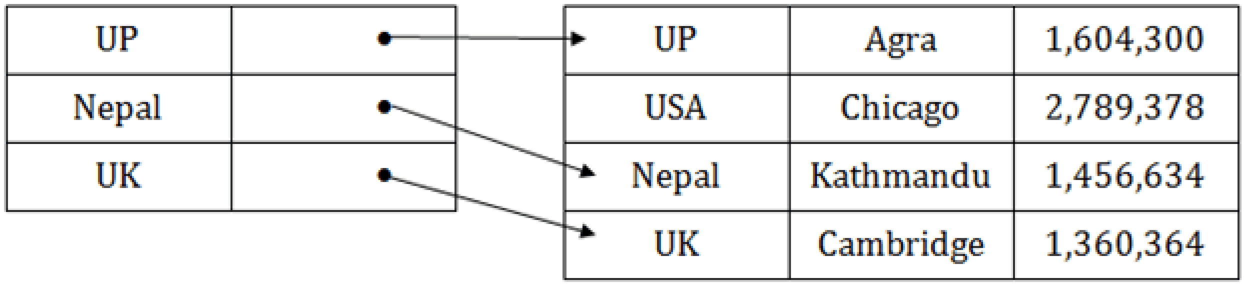


UP	●
USA	●
Nepal	●
UK	●

UP	Agra	1,604,300
USA	Chicago	2,789,378
Nepal	Kathmandu	1,456,634
UK	Cambridge	1,360,364

Sparse index

- In the data file, index record appears only for a few items. Each item points to a block.
- In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.



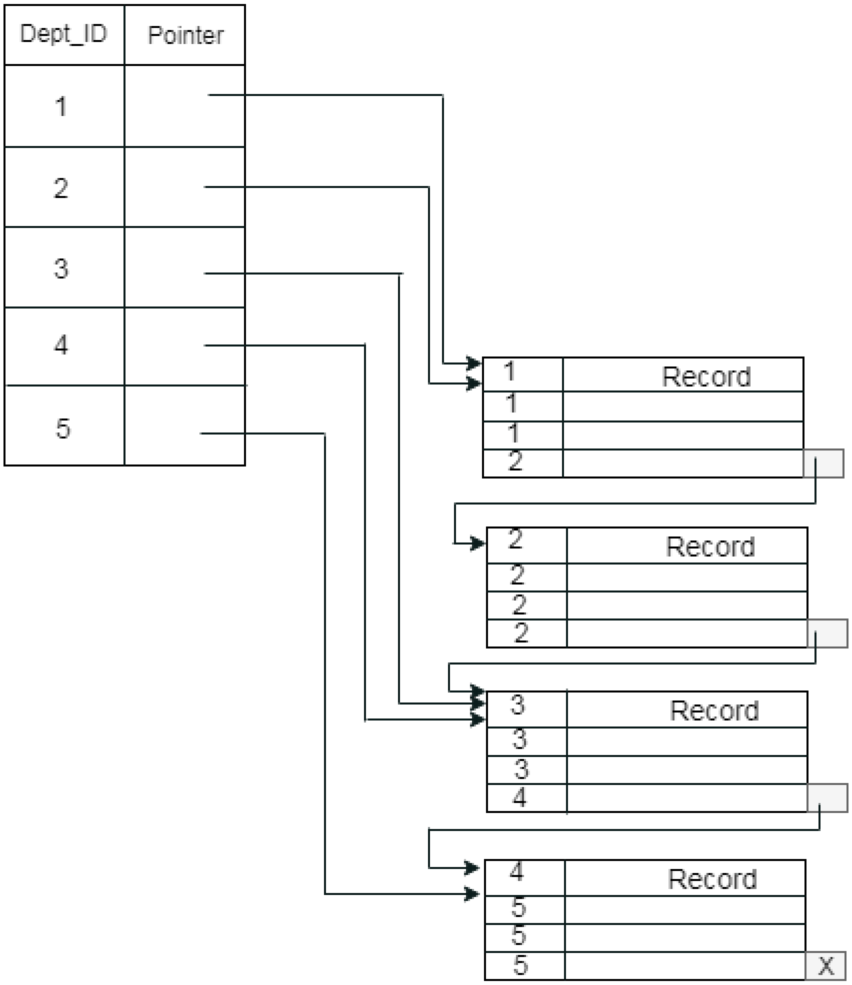
UP	●
Nepal	●
UK	●

UP	Agra	1,604,300
USA	Chicago	2,789,378
Nepal	Kathmandu	1,456,634
UK	Cambridge	1,360,364

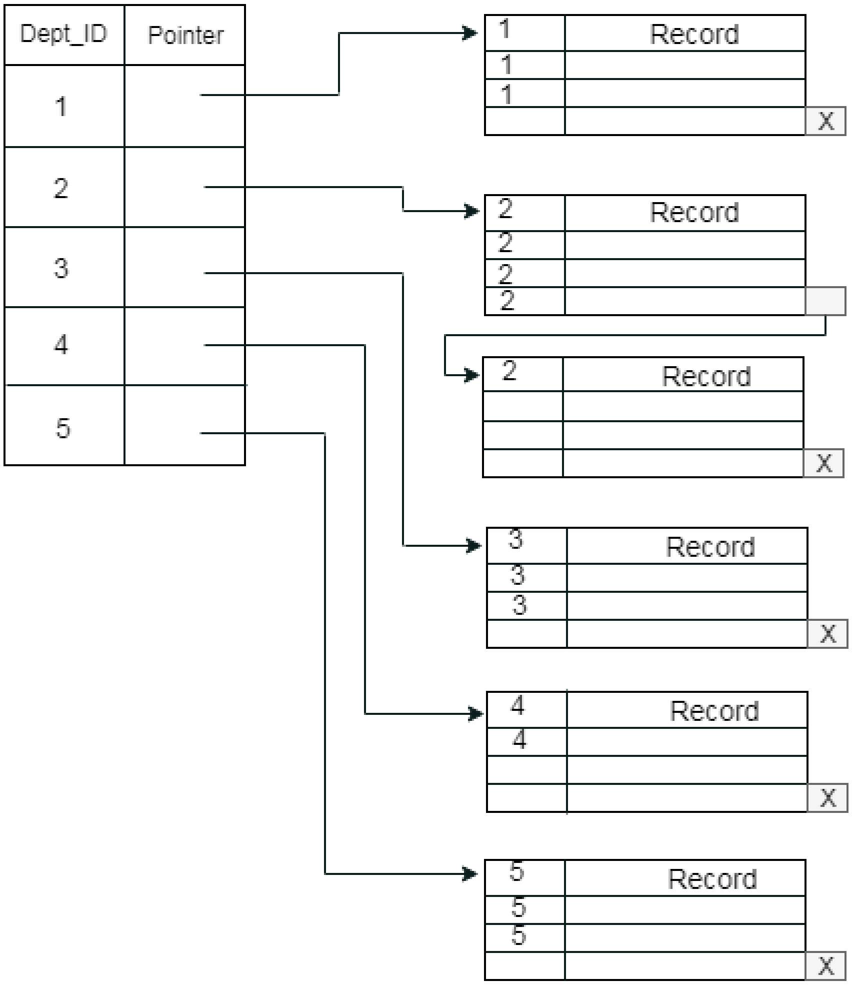
Clustering Index

- A clustered index can be defined as an ordered data file. Sometimes the index is created on non-primary key columns which may not be unique for each record.
- In this case, to identify the record faster, we will group two or more columns to get the unique value and create index out of them. This method is called a clustering index.
- The records which have similar characteristics are grouped, and indexes are created for these group.

Example: suppose a company contains several employees in each department. Suppose we use a clustering index, where all employees which belong to the same Dept_ID are considered within a single cluster, and index pointers point to the cluster as a whole. Here Dept_Id is a non-unique key.



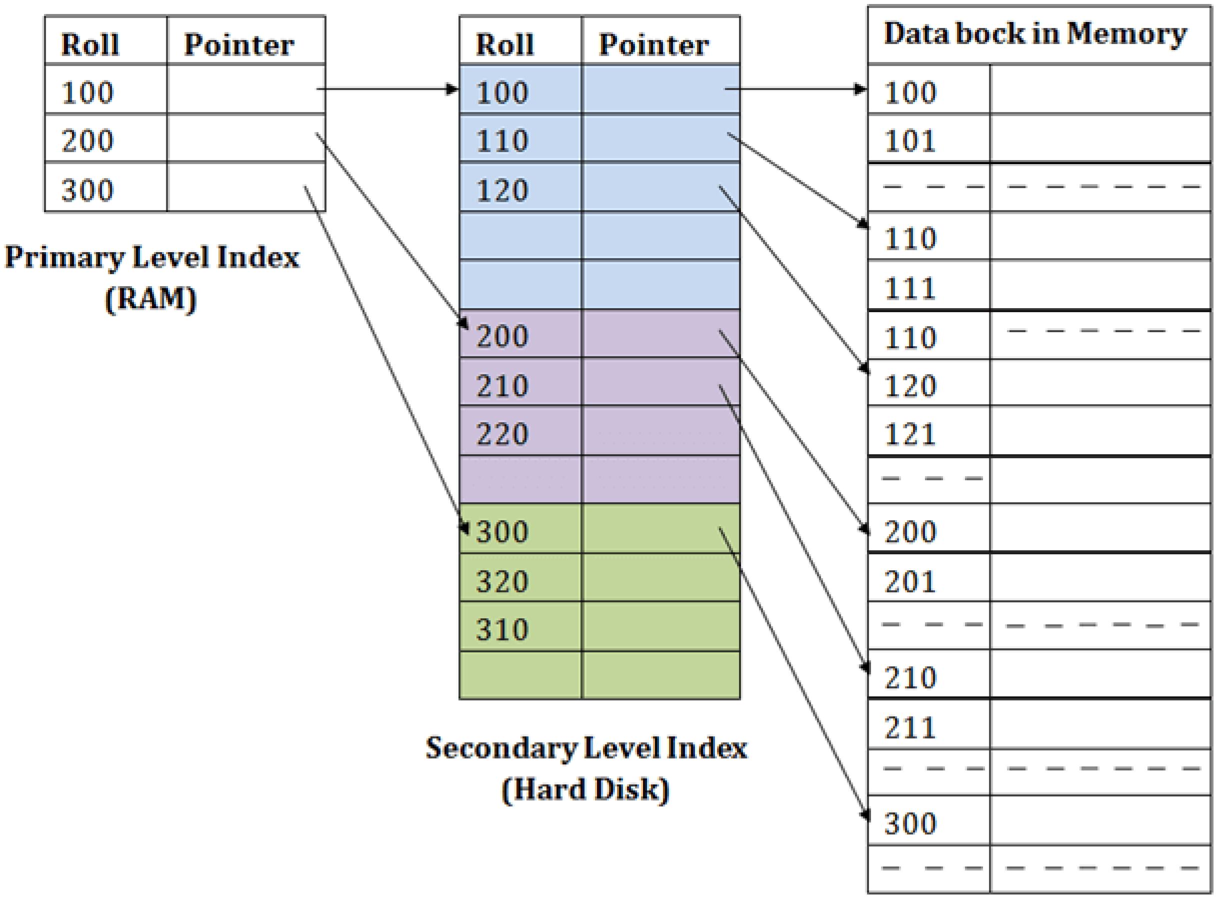
The previous schema is little confusing because one disk block is shared by records which belong to the different cluster. If we use separate disk block for separate clusters, then it is called better technique.



Secondary Index

In the sparse indexing, as the size of the table grows, the size of mapping also grows. These mappings are usually kept in the primary memory so that address fetch should be faster. Then the secondary memory searches the actual data based on the address got from mapping. If the mapping size grows then fetching the address itself becomes slower. In this case, the sparse index will not be efficient. To overcome this problem, secondary indexing is introduced.

In secondary indexing, to reduce the size of mapping, another level of indexing is introduced. In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small. Then each range is further divided into smaller ranges. The mapping of the first level is stored in the primary memory, so that address fetch is faster. The mapping of the second level and actual data are stored in the secondary memory (hard disk).



For example:

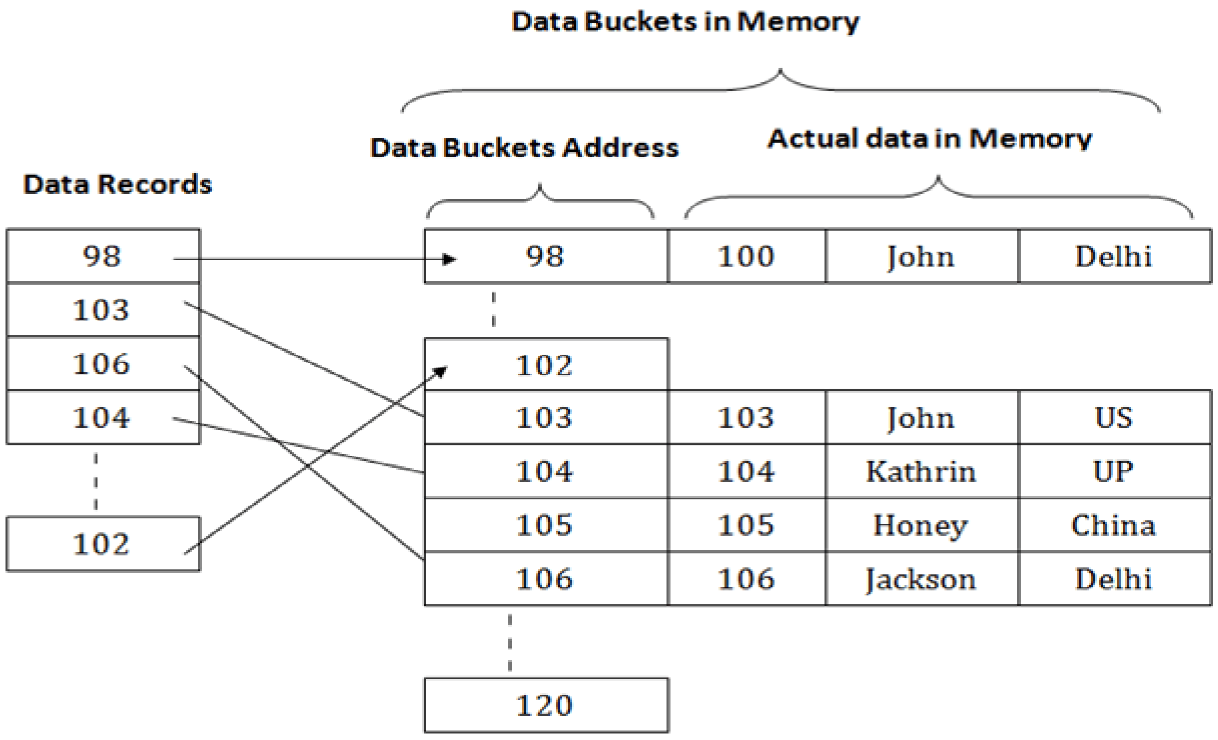
- If you want to find the record of roll 111 in the diagram, then it will search the highest entry which is smaller than or equal to 111 in the first level index. It will get 100 at this level.
- Then in the second index level, again it does max (111) <= 111 and gets 110. Now using the address 110, it goes to the data block and starts searching each record till it gets 111.
- This is how a search is performed in this method. Inserting, updating or deleting is also done in the same manner.

Hashing

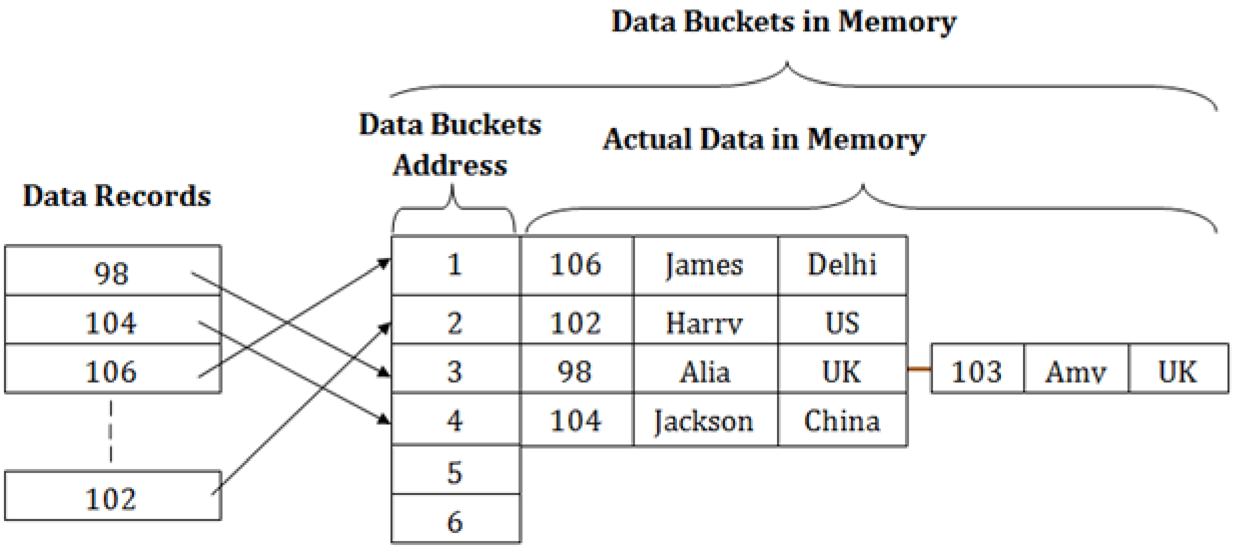
In a huge database structure, it is very inefficient to search all the index values and reach the desired data. Hashing technique is used to calculate the direct location of a data record on the disk without using index structure.

In this technique, data is stored at the data blocks whose address is generated by using the hashing function. The memory location where these records are stored is known as data bucket or data blocks.

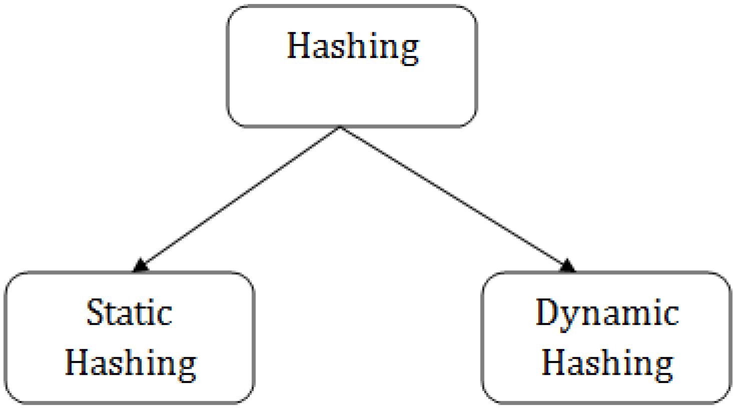
In this, a hash function can choose any of the column value to generate the address. Most of the time, the hash function uses the primary key to generate the address of the data block. A hash function is a simple mathematical function to any complex mathematical function. We can even consider the primary key itself as the address of the data block. That means each row whose address will be the same as a primary key stored in the data block.



The above diagram shows data block addresses same as primary key value. This hash function can also be a simple mathematical function like exponential, mod, cos, sin, etc. Suppose we have mod (5) hash function to determine the address of the data block. In this case, it applies mod (5) hash function on the primary keys and generates 3, 3, 1, 4 and 2 respectively, and records are stored in those data block addresses.



Types of Hashing:

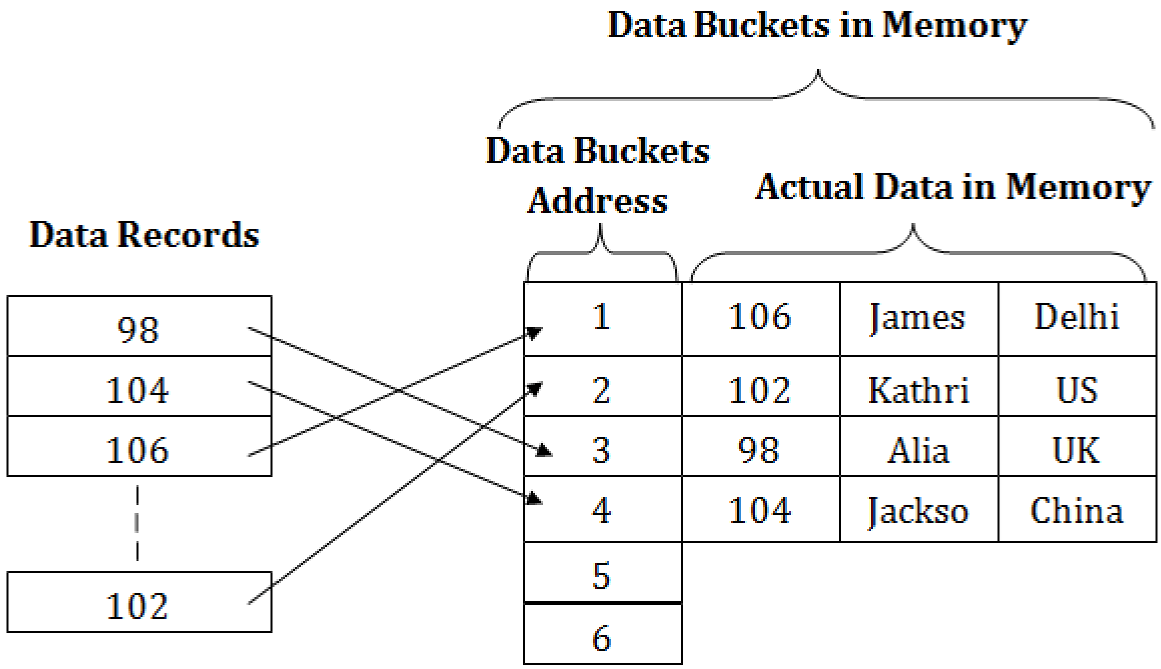


- Static Hashing
- Dynamic Hashing

Static Hashing

In static hashing, the resultant data bucket address will always be the same. That means if we generate an address for EMP_ID =103 using the hash function mod (5) then it will always result in same bucket address 3. Here, there will be no change in the bucket address.

Hence in this static hashing, the number of data buckets in memory remains constant throughout. In this example, we will have five data buckets in the memory used to store the data.



Operations of Static Hashing

- **Searching a record**

When a record needs to be searched, then the same hash function retrieves the address of the bucket where the data is stored.

- **Insert a Record**

When a new record is inserted into the table, then we will generate an address for a new record based on the hash key and record is stored in that location.

- **Delete a Record**

To delete a record, we will first fetch the record which is supposed to be deleted. Then we will delete the records for that address in memory.

- **Update a Record**

To update a record, we will first search it using a hash function, and then the data record is updated.

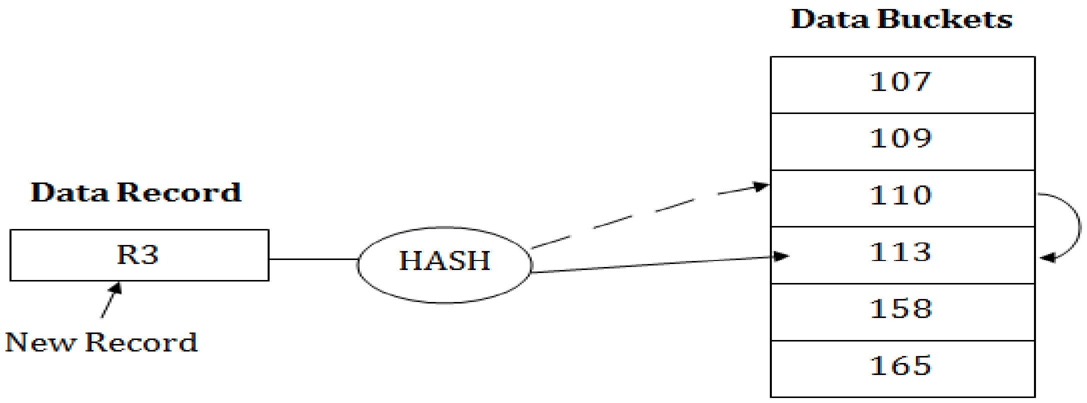
If we want to insert some new record into the file but the address of a data bucket generated by the hash function is not empty, or data already exists in that address. This situation in the static hashing is known as **bucket overflow**. This is a critical situation in this method.

To overcome this situation, there are various methods. Some commonly used methods are as follows:

1. **Open Hashing**

When a hash function generates an address at which data is already stored, then the next bucket will be allocated to it. This mechanism is called as **Linear Probing**.

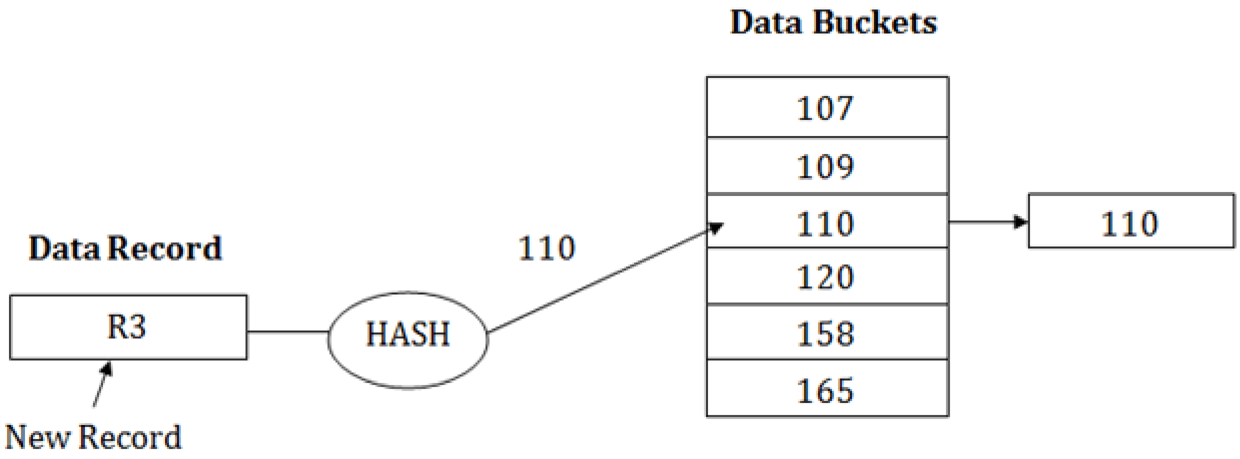
For example: suppose R3 is a new address which needs to be inserted, the hash function generates address as 112 for R3. But the generated address is already full. So the system searches next available data bucket, 113 and assigns R3 to it.



2. Close Hashing

When buckets are full, then a new data bucket is allocated for the same hash result and is linked after the previous one. This mechanism is known as **Overflow chaining**.

For example: Suppose R3 is a new address which needs to be inserted into the table, the hash function generates address as 110 for it. But this bucket is full to store the new data. In this case, a new bucket is inserted at the end of 110 buckets and is linked to it.



Dynamic Hashing

- The dynamic hashing method is used to overcome the problems of static hashing like bucket overflow.
- In this method, data buckets grow or shrink as the records increases or decreases. This method is also known as Extendable hashing method.
- This method makes hashing dynamic, i.e., it allows insertion or deletion without resulting in poor performance.

How to search a key

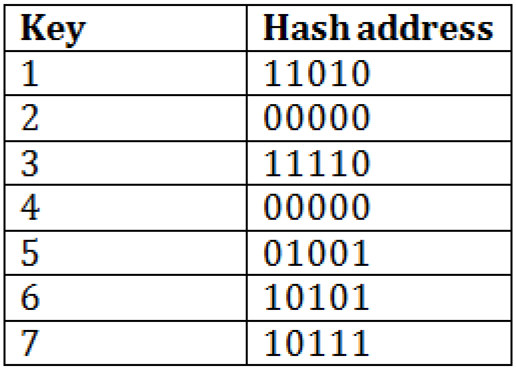
- First, calculate the hash address of the key.
- Check how many bits are used in the directory, and these bits are called as i.
- Take the least significant i bits of the hash address. This gives an index of the directory.
- Now using the index, go to the directory and find bucket address where the record might be.

How to insert a new record

- Firstly, you have to follow the same procedure for retrieval, ending up in some bucket.
- If there is still space in that bucket, then place the record in it.
- If the bucket is full, then we will split the bucket and redistribute the records.

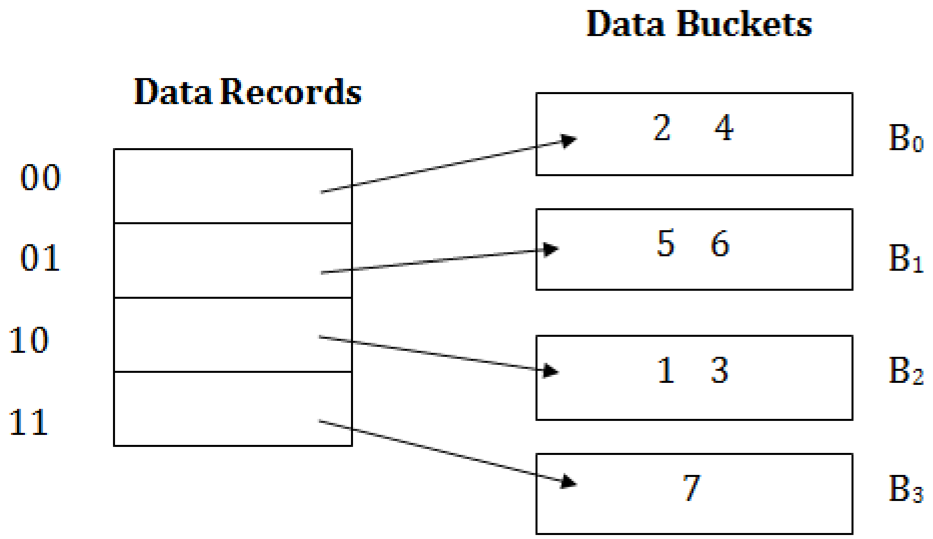
For example:

Consider the following grouping of keys into buckets, depending on the prefix of their hash address:



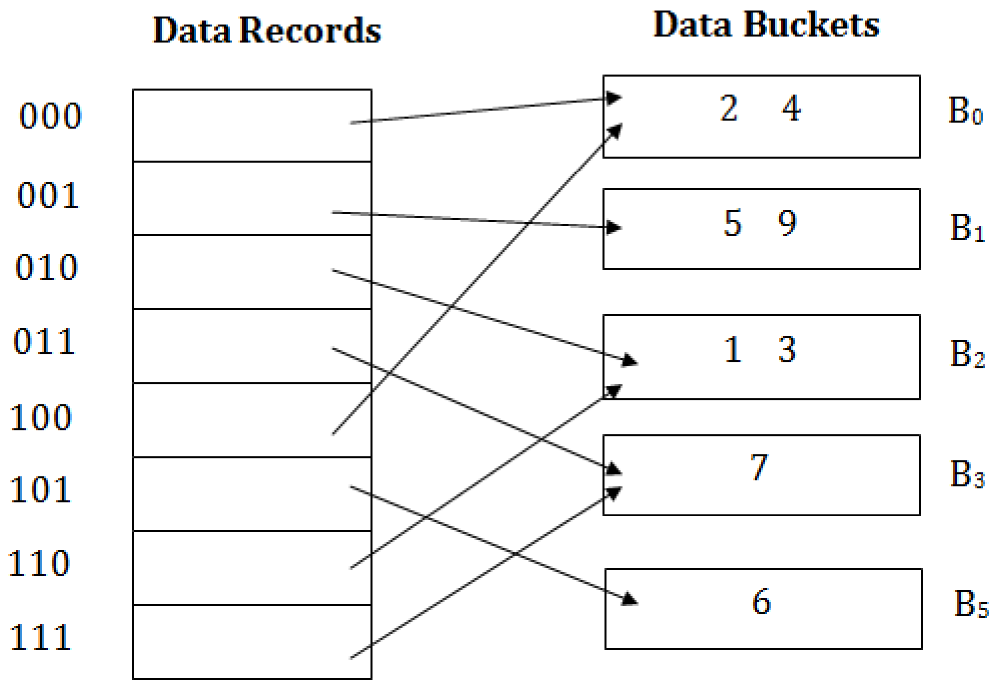
Key	Hash address
1	11010
2	00000
3	11110
4	00000
5	01001
6	10101
7	10111

The last two bits of 2 and 4 are 00. So it will go into bucket B0. The last two bits of 5 and 6 are 01, so it will go into bucket B1. The last two bits of 1 and 3 are 10, so it will go into bucket B2. The last two bits of 7 are 11, so it will go into B3.



Insert key 9 with hash address 10001 into the above structure:

- Since key 9 has hash address 10001, it must go into the first bucket. But bucket B1 is full, so it will get split.
- The splitting will separate 5, 9 from 6 since last three bits of 5, 9 are 001, so it will go into bucket B1, and the last three bits of 6 are 101, so it will go into bucket B5.
- Keys 2 and 4 are still in B0. The record in B0 pointed by the 000 and 100 entry because last two bits of both the entry are 00.
- Keys 1 and 3 are still in B2. The record in B2 pointed by the 010 and 110 entry because last two bits of both the entry are 10.
- Key 7 are still in B3. The record in B3 pointed by the 111 and 011 entry because last two bits of both the entry are 11.



Advantages of dynamic hashing

- In this method, the performance does not decrease as the data grows in the system. It simply increases the size of memory to accommodate the data.
- In this method, memory is well utilized as it grows and shrinks with the data. There will not be any unused memory lying.
- This method is good for the dynamic database where data grows and shrinks frequently.

Disadvantages of dynamic hashing

- In this method, if the data size increases then the bucket size is also increased. These addresses of data will be maintained in the bucket address table. This is because the data address will keep changing as buckets grow and shrink. If there is a huge increase in data, maintaining the bucket address table becomes tedious.
- In this case, the bucket overflow situation will also occur. But it might take little time to reach this situation than static hashing.

RAID

RAID refers to redundancy array of the independent disk. It is a technology which is used to connect multiple secondary storage devices for increased performance, data redundancy or both. It gives you the ability to survive one or more drive failure depending upon the RAID level used.

It consists of an array of disks in which multiple disks are connected to achieve different goals.

RAID technology

There are 7 levels of RAID schemes. These schemas are as RAID 0, RAID 1, ..., RAID 6.

These levels contain the following characteristics:

- It contains a set of physical disk drives.
- In this technology, the operating system views these separate disks as a single logical disk.
- In this technology, data is distributed across the physical drives of the array.
- Redundancy disk capacity is used to store parity information.
- In case of disk failure, the parity information can be helped to recover the data.

Standard RAID levels

RAID 0

- RAID level 0 provides data stripping, i.e., a data can place across multiple disks. It is based on stripping that means if one disk fails then all data in the array is lost.
- This level doesn't provide fault tolerance but increases the system performance.

Example:

Disk 0	**Disk 1**	**Disk 2**	**Disk 3**
20	21	22	23
24	25	26	27
28	29	30	31
32	33	34	35

In this figure, block 0, 1, 2, 3 form a stripe.

In this level, instead of placing just one block into a disk at a time, we can work with two or more blocks placed it into a disk before moving on to the next one.

Disk 0	**Disk 1**	**Disk 2**	**Disk 3**
20	22	24	26
21	23	25	27
28	30	32	34
29	31	33	35

In this above figure, there is no duplication of data. Hence, a block once lost cannot be recovered.

Pros of RAID 0:

- In this level, throughput is increased because multiple data requests probably not on the same disk.
- This level full utilizes the disk space and provides high performance.
- It requires minimum 2 drives.

Cons of RAID 0:

- It doesn't contain any error detection mechanism.
- The RAID 0 is not a true RAID because it is not fault-tolerance.
- In this level, failure of either disk results in complete data loss in respective array.

RAID 1

This level is called mirroring of data as it copies the data from drive 1 to drive 2. It provides 100% redundancy in case of a failure.

Example:

Disk 0	**Disk 1**	**Disk 2**	**Disk 3**
A	A	B	B
C	C	D	D
E	E	F	F
G	G	H	H

Only half space of the drive is used to store the data. The other half of drive is just a mirror to the already stored data.

Pros of RAID 1:

- The main advantage of RAID 1 is fault tolerance. In this level, if one disk fails, then the other automatically takes over.
- In this level, the array will function even if any one of the drives fails.

Cons of RAID 1:

- In this level, one extra drive is required per drive for mirroring, so the expense is higher.

RAID 2

- RAID 2 consists of bit-level striping using hamming code parity. In this level, each data bit in a word is recorded on a separate disk and ECC code of data words is stored on different set disks.
- Due to its high cost and complex structure, this level is not commercially used. This same performance can be achieved by RAID 3 at a lower cost.

Pros of RAID 2:

- This level uses one designated drive to store parity.
- It uses the hamming code for error detection.

Cons of RAID 2:

- It requires an additional drive for error detection.

RAID 3

- RAID 3 consists of byte-level striping with dedicated parity. In this level, the parity information is stored for each disk section and written to a dedicated parity drive.
- In case of drive failure, the parity drive is accessed, and data is reconstructed from the remaining devices. Once the failed drive is replaced, the missing data can be restored on the new drive.
- In this level, data can be transferred in bulk. Thus high-speed data transmission is possible.

Disk 0	**Disk 1**	**Disk 2**	**Disk 3**
A	B	C	P(A, B, C)
D	E	F	P(D, E, F)
G	H	I	P(G, H, I)
J	K	L	P(J, K, L)

Pros of RAID 3:

- In this level, data is regenerated using parity drive.
- It contains high data transfer rates.
- In this level, data is accessed in parallel.

Cons of RAID 3:

- It required an additional drive for parity.
- It gives a slow performance for operating on small sized files.

RAID 4

- RAID 4 consists of block-level stripping with a parity disk. Instead of duplicating data, the RAID 4 adopts a parity-based approach.
- This level allows recovery of at most 1 disk failure due to the way parity works. In this level, if more than one disk fails, then there is no way to recover the data.
- Level 3 and level 4 both are required at least three disks to implement RAID.

Disk 0	**Disk 1**	**Disk 2**	**Disk 3**
A	B	C	P0
D	E	F	P1
G	H	I	P2
J	K	L	P3

In this figure, we can observe one disk dedicated to parity.

In this level, parity can be calculated using an XOR function. If the data bits are 0,0,0,1 then the parity bits is XOR(0,1,0,0) = 1. If the parity bits are 0,0,1,1 then the parity bit is XOR(0,0,1,1)=

0. That means, even number of one results in parity 0 and an odd number of one results in parity 1.

C1	C2	C3	C4	Parity
0	1	0	0	1
0	0	1	1	0

Suppose that in the above figure, C2 is lost due to some disk failure. Then using the values of all the other columns and the parity bit, we can recompute the data bit stored in C2. This level allows us to recover lost data.

RAID 5

- RAID 5 is a slight modification of the RAID 4 system. The only difference is that in RAID 5, the parity rotates among the drives.
- It consists of block-level striping with DISTRIBUTED parity.
- Same as RAID 4, this level allows recovery of at most 1 disk failure. If more than one disk fails, then there is no way for data recovery.

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

This figure shows that how parity bit rotates.

This level was introduced to make the random write performance better.

Pros of RAID 5:

- This level is cost effective and provides high performance.
- In this level, parity is distributed across the disks in an array.
- It is used to make the random write performance better.

Cons of RAID 5:

- In this level, disk failure recovery takes longer time as parity has to be calculated from all available drives.
- This level cannot survive in concurrent drive failure.

RAID 6

- This level is an extension of RAID 5. It contains block-level stripping with 2 parity bits.
- In RAID 6, you can survive 2 concurrent disk failures. Suppose you are using RAID 5, and RAID 1. When your disks fail, you need to replace the failed disk because if simultaneously another disk fails then you won't be able to recover any of the data, so in this case RAID 6 plays its part where you can survive two concurrent disk failures before you run out of options.

Disk 1	**Disk 2**	**Disk 3**	**Disk 4**
A0	B0	Q0	P0
A1	Q1	P1	D1
Q2	P2	C2	D2
P3	B3	C3	Q3

Pros of RAID 6:

- This level performs RAID 0 to strip data and RAID 1 to mirror. In this level, stripping is performed before mirroring.
- In this level, drives required should be multiple of 2.

Cons of RAID 6:

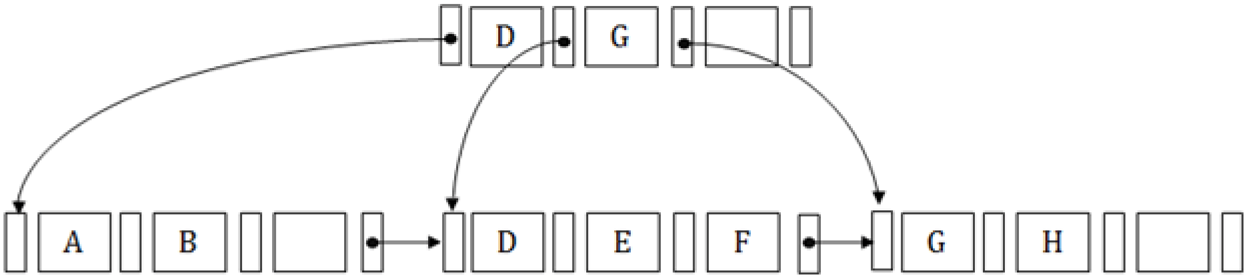
- It is not utilized 100% disk capability as half is used for mirroring.
- It contains very limited scalability.

B+ Tree

- The B+ tree is a balanced binary search tree. It follows a multi-level index format.
- In the B+ tree, leaf nodes denote actual data pointers. B+ tree ensures that all leaf nodes remain at the same height.
- In the B+ tree, the leaf nodes are linked using a link list. Therefore, a B+ tree can support random access as well as sequential access.

Structure of B+ Tree

- In the B+ tree, every leaf node is at equal distance from the root node. The B+ tree is of the order n where n is fixed for every B+ tree.
- It contains an internal node and leaf node.



Internal node

- An internal node of the B+ tree can contain at least n/2 record pointers except the root node.
- At most, an internal node of the tree contains n pointers.

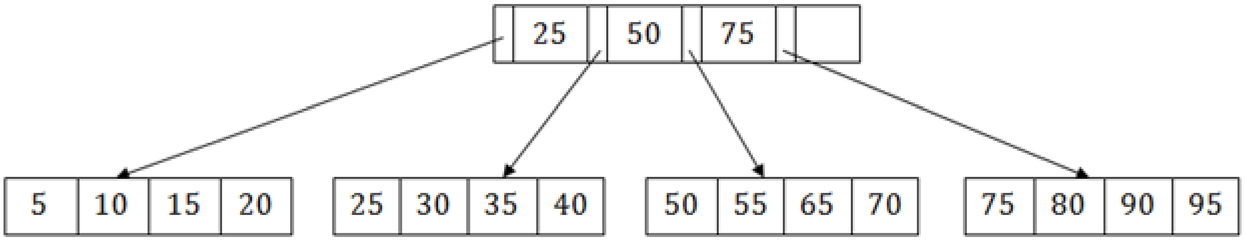
Leaf node

- The leaf node of the B+ tree can contain at least n/2 record pointers and n/2 key values.
- At most, a leaf node contains n record pointer and n key values.
- Every leaf node of the B+ tree contains one block pointer P to point to next leaf node.

Searching a record in B+ Tree

Suppose we have to search 55 in the below B+ tree structure. First, we will fetch for the intermediary node which will direct to the leaf node that can contain a record for 55.

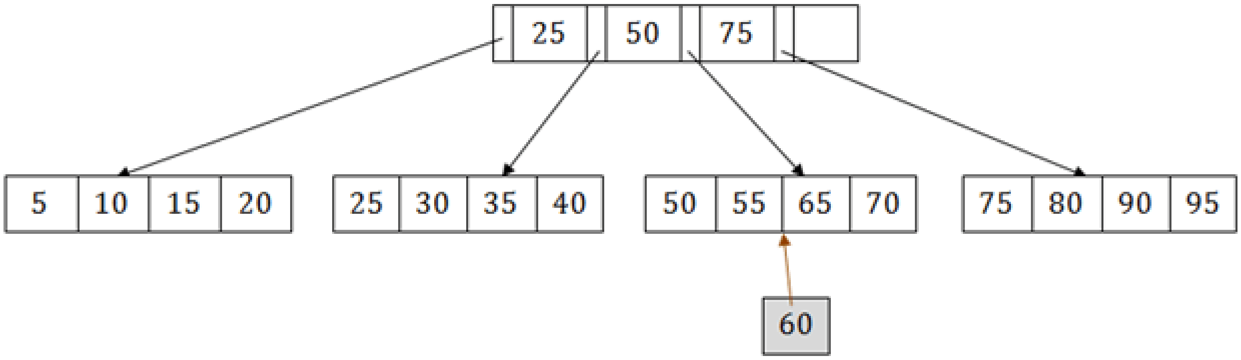
So, in the intermediary node, we will find a branch between 50 and 75 nodes. Then at the end, we will be redirected to the third leaf node. Here DBMS will perform a sequential search to find 55.



B+ Tree Insertion

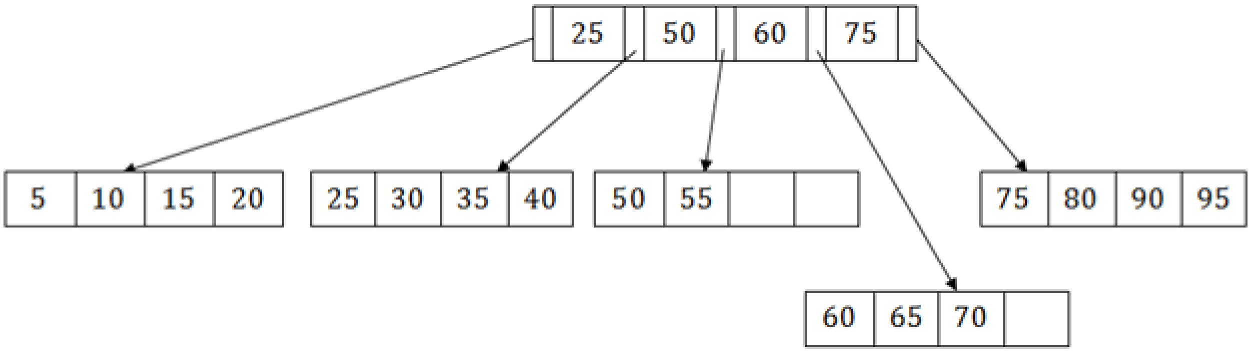
Suppose we want to insert a record 60 in the below structure. It will go to the 3rd leaf node after 55. It is a balanced tree, and a leaf node of this tree is already full, so we cannot insert 60 there.

In this case, we have to split the leaf node, so that it can be inserted into tree without affecting the fill factor, balance and order.



The 3rd leaf node has the values (50, 55, 60, 65, 70) and its current root node is 50. We will split the leaf node of the tree in the middle so that its balance is not altered. So we can group (50, 55) and (60, 65, 70) into 2 leaf nodes.

If these two has to be leaf nodes, the intermediate node cannot branch from 50. It should have 60 added to it, and then we can have pointers to a new leaf node.

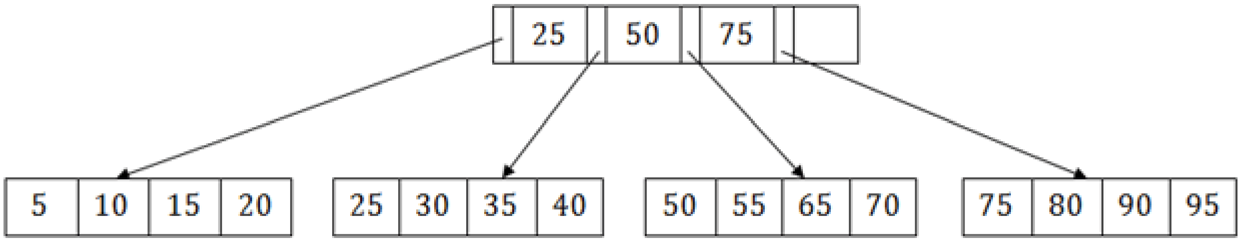


This is how we can insert an entry when there is overflow. In a normal scenario, it is very easy to find the node where it fits and then place it in that leaf node.

B+ Tree Deletion

Suppose we want to delete 60 from the above example. In this case, we have to remove 60 from the intermediate node as well as from the 4th leaf node too. If we remove it from the intermediate node, then the tree will not satisfy the rule of the B+ tree. So we need to modify it to have a balanced tree.

After deleting node 60 from above B+ tree and re-arranging the nodes, it will show as follows:


25
50
75
5
10
15
20
25
30
35
40
50
55
65
70
75
80
90
95