

UNIT-1

What is Database

The database is a collection of inter-related data which is used to retrieve, insert and delete the data efficiently. It is also used to organize the data in the form of a table, schema, views, and reports, etc.

For example: The college Database organizes the data about the admin, staff, students and faculty etc.

Using the database, you can easily retrieve, insert, and delete the information.

Database Management System

- Database management system is a software which is used to manage the database. For example: **MySQL**, **Oracle**, etc are a very popular commercial database which is used in different applications.
- DBMS provides an interface to perform various operations like database creation, storing data in it, updating data, creating a table in the database and a lot more.
- It provides protection and security to the database. In the case of multiple users, it also maintains data consistency.

DBMS allows users the following tasks:

- **Data Definition:** It is used for creation, modification, and removal of definition that defines the organization of data in the database.
- **Data Updation:** It is used for the insertion, modification, and deletion of the actual data in the database.
- **Data Retrieval:** It is used to retrieve the data from the database which can be used by applications for various purposes.
- **User Administration:** It is used for registering and monitoring users, maintain data integrity, enforcing data security, dealing with concurrency control, monitoring performance and recovering information corrupted by unexpected failure.

Characteristics of DBMS

- It uses a digital repository established on a server to store and manage the information.
- It can provide a clear and logical view of the process that manipulates data.

- DBMS contains automatic backup and recovery procedures.
- It contains ACID properties which maintain data in a healthy state in case of failure.
- It can reduce the complex relationship between data.
- It is used to support manipulation and processing of data.
- It is used to provide security of data.
- It can view the database from different viewpoints according to the requirements of the user.

Advantages of DBMS

- **Controls database redundancy:** It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.
- **Data sharing:** In DBMS, the authorized users of an organization can share the data among multiple users.
- **Easily Maintenance:** It can be easily maintainable due to the centralized nature of the database system.
- **Reduce time:** It reduces development time and maintenance need.
- **Backup:** It provides backup and recovery subsystems which create automatic backup of data from **hardware** and **software** failures and restores the data if required.
- **multiple user interface:** It provides different types of user interfaces like graphical user interfaces, application program interfaces

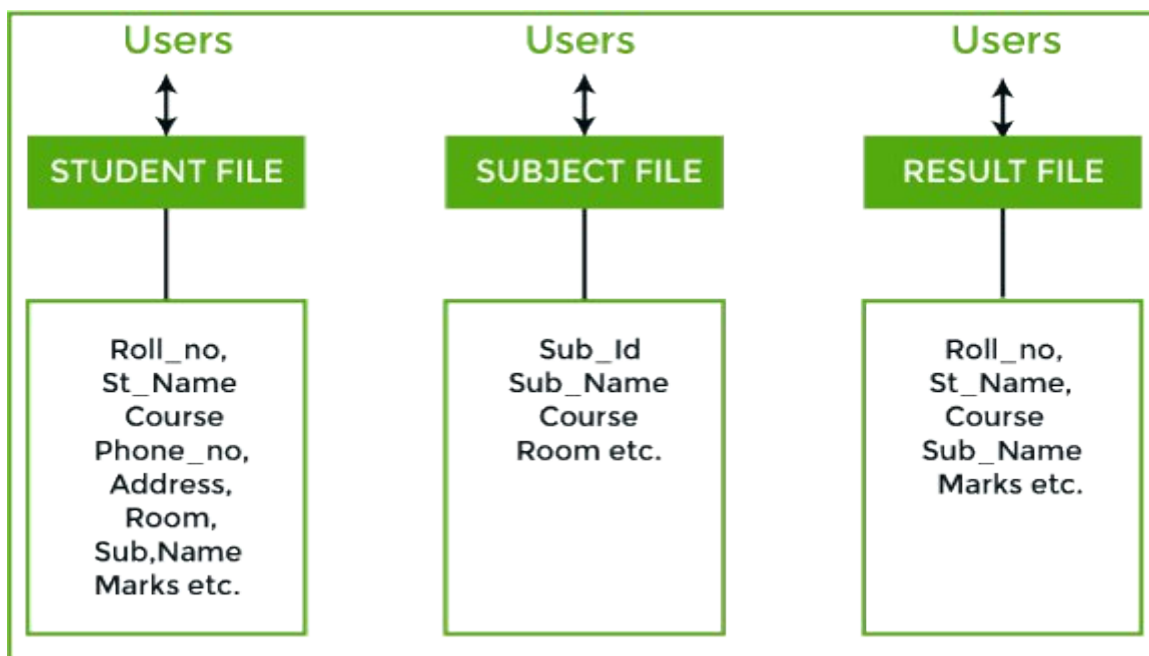
Disadvantages of DBMS

- **Cost of Hardware and Software:** It requires a high speed of data processor and large memory size to run DBMS software.
- **Size:** It occupies a large space of disks and large memory to run them efficiently.
- **Complexity:** Database system creates additional complexity and requirements.

Higher impact of failure: Failure is highly impacted the database because in most of the organization, all the data stored in a single database and if the database is damaged due to electric failure or database corruption then the data may be lost forever. **Difference between File System and DBMS**

File System Approach

File based systems were an early attempt to computerize the manual system. It is also called a traditional based approach in which a decentralized approach was taken where each department stored and controlled its own data with the help of a data processing specialist. The main role of a data processing specialist was to create the necessary computer file structures, and also manage the data within structures and design some application programs that create reports based on file data.



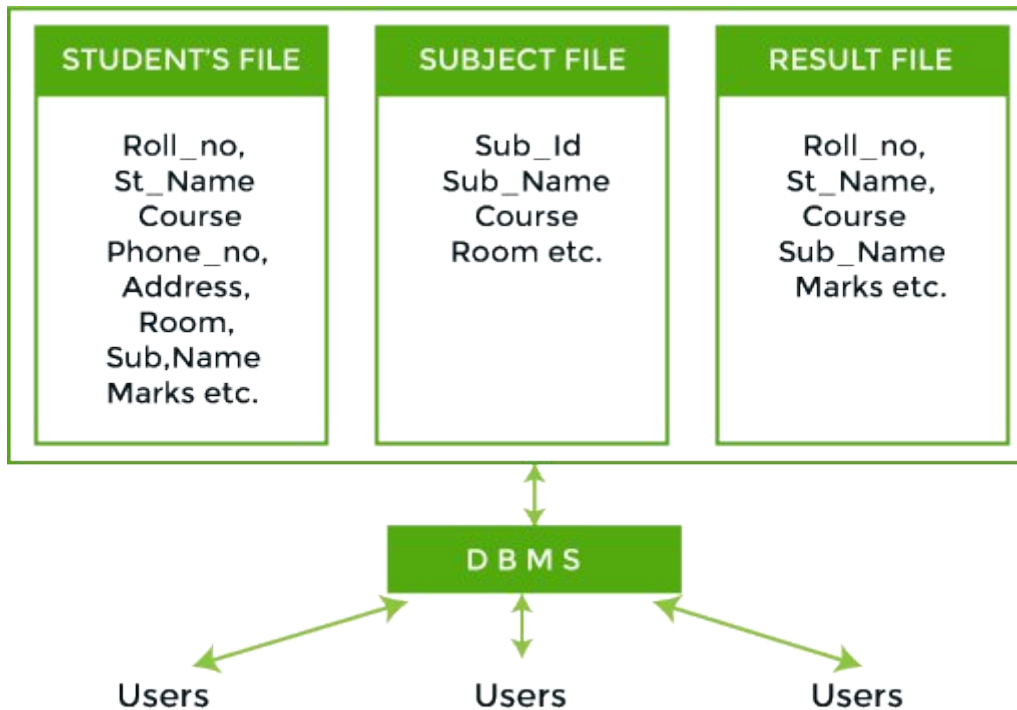
In the above figure:

Consider an example of a student's file system. The student file will contain information regarding the student (i.e. roll no, student name, course etc.). Similarly, we have a subject file that contains information about the subject and the result file which contains the information regarding the result.

Some fields are duplicated in more than one file, which leads to data redundancy. So to overcome this problem, we need to create a centralized system, i.e. DBMS approach.

DBMS:

A database approach is a well-organized collection of data that are related in a meaningful way which can be accessed by different users but stored only once in a system. The various operations performed by the DBMS system are: Insertion, deletion, selection, sorting etc.



In the above figure,

In the above figure, duplication of data is reduced due to centralization of data.

There are the following differences between DBMS and File systems:

Basis	DBMS Approach	File System Approach
Meaning	DBMS is a collection of data. In DBMS, the user is not required to write the procedures.	The file system is a collection of data. In system, the user has to write the proced for managing the database.
Sharing of data	Due to the centralized approach, data sharing is easy.	Data is distributed in many files, and it be of different formats, so it isn't eas share data.
Data Abstraction	DBMS gives an abstract view of data that hides the details.	The file system provides the detail of the representation and storage of data.
Security and Protection	DBMS provides a good protection mechanism.	It isn't easy to protect a file under the system.
Recovery Mechanism	DBMS provides a crash recovery mechanism, i.e., DBMS protects the user from system failure.	The file system doesn't have a c mechanism, i.e., if the system crashes w

		entering some data, then the content of file will be lost.
Manipulation Techniques	DBMS contains a wide variety of sophisticated techniques to store and retrieve the data.	The file system can't efficiently store retrieve the data.
Concurrency Problems	DBMS takes care of Concurrent access of data using some form of locking.	In the File system, concurrent access many problems like redirecting the file w deleting some information or updating s information.
Where to use	Database approach used in large systems which interrelate many files.	File system approach used in large syst which interrelate many files.
Cost	The database system is expensive to design.	The file system approach is cheaper design.
Data Redundancy and Inconsistency	Due to the centralization of the database, the problems of data redundancy and inconsistency are controlled.	In this, the files and application programs created by different programmers so there exists a lot of duplication of data w may lead to inconsistency.
Structure	The database structure is complex to design.	The file system approach has a sim structure.
Data Independence	In this system, Data Independence exists, and it can be of two types. ○ Logical Data Independence ○ Physical Data Independence	In the File system approach, there exists Data Independence.
Integrity Constraints	Integrity Constraints are easy to apply.	Integrity Constraints are difficult implement in file system.
Data Models	In the database approach, 3 types of data models exist: ○ Hierarchal data models ○ Network data models ○ Relational data models	In the file system approach, there is concept of data models exists.

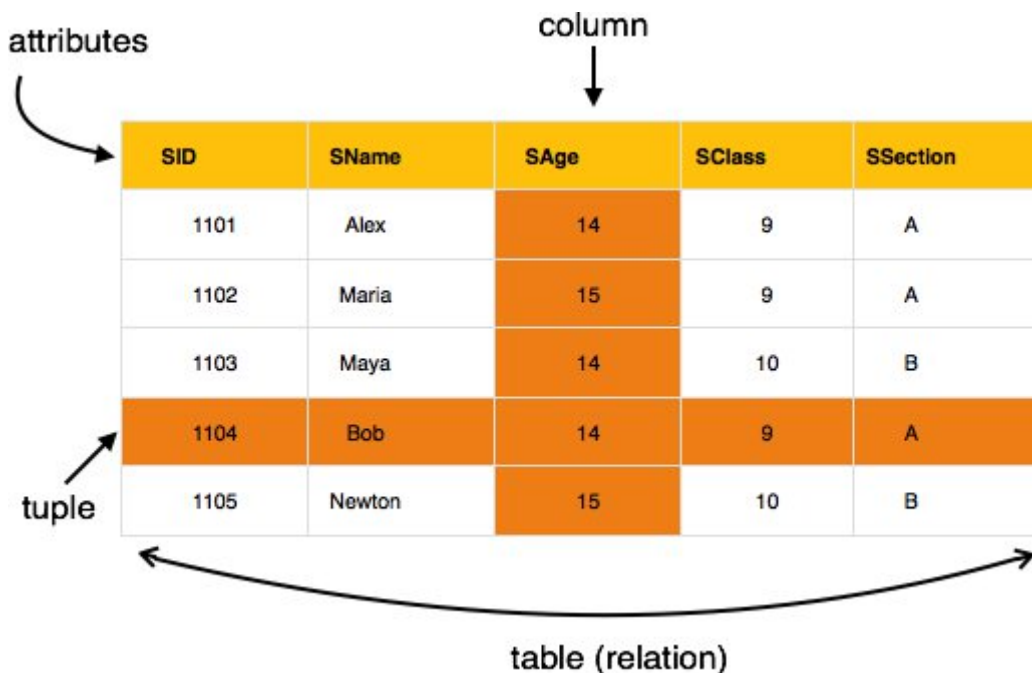
Flexibility	Changes are often a necessity to the content of the data stored in any system, and these changes are more easily with a database approach.	The flexibility of the system is less compared to the DBMS approach.
Examples	Oracle, SQL Server, Sybase etc.	Cobol, C++ etc.

Data Models

Data Model is the modeling of the data description, data semantics, and consistency constraints of the data. It provides the conceptual tools for describing the design of a database at each level of data abstraction. Therefore, there are following four data models used for understanding the structure of the database:

Relational Model

The most popular data model in DBMS is the Relational Model. It is more scientific a model than others. This model is based on first-order predicate logic and defines a table as an **n-ary relation**.



The main highlights of this model are –

- Data is stored in tables called **relations**.
- Relations can be normalized.
- In normalized relations, values saved are atomic values.

- Each row in a relation contains a unique value.
- Each column in a relation contains values from a same domain.

Entity-Relationship Model

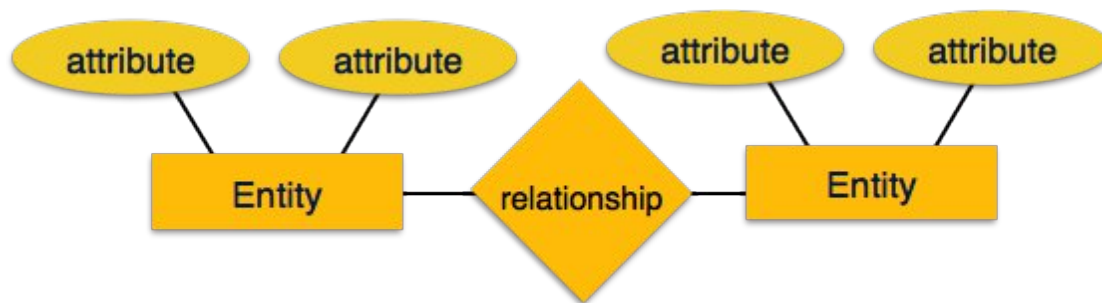
Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates entity set, relationship set, general attributes and constraints.

ER Model is best used for the conceptual design of a database.

ER Model is based on –

- **Entities** and their *attributes*.
- **Relationships** among entities.

These concepts are explained below.



- **Entity** – An entity in an ER Model is a real-world entity having properties called **attributes**. Every **attribute** is defined by its set of values called **domain**. For example, in a school database, a student is considered as an entity. Student has various attributes like name, age, class, etc.
- **Relationship** – The logical association among entities is called **relationship**. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of association between two entities.

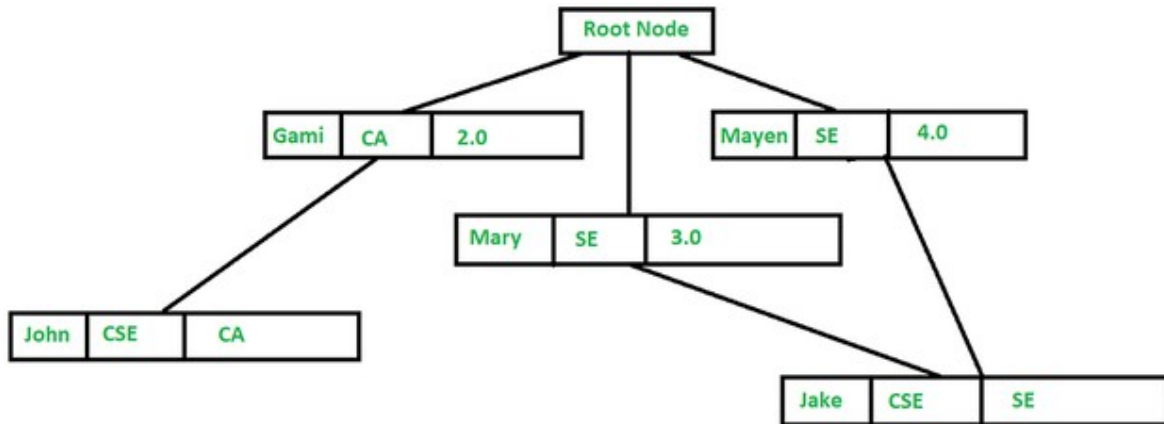
Mapping cardinalities –

- one to one
- one to many
- many to one
- many to many

hierarchical model,

In a hierarchical model, data are viewed as a collection of tables, or we can say segments that form a hierarchical relation. In this, the data is organized into a tree-like structure where each record consists of one parent record and many children. Even if the segments are connected as a chain-like structure by logical associations, then the instant structure

can be a fan structure with multiple branches. We call the illogical associations as directional associations.



2. Network Model

The [Network Model](#) was formalized by the Database Task group in the 1960s. This model is the generalization of the hierarchical model. This model can consist of multiple parent segments and these segments are grouped as levels but there exists a logical association between the segments belonging to any level. Mostly, there exists a many-to-many logical association between any of the two segments.



3. Object-Oriented Data Model

Object oriented data model is based upon real world situations. These situations are represented as objects, with different attributes. All these object have multiple relationships between them.

Elements of Object oriented data model

Objects

The real world entities and situations are represented as objects in the Object oriented database model.

Attributes and Method

Every object has certain characteristics. These are represented using Attributes. The behaviour of the objects is represented using Methods.

Class

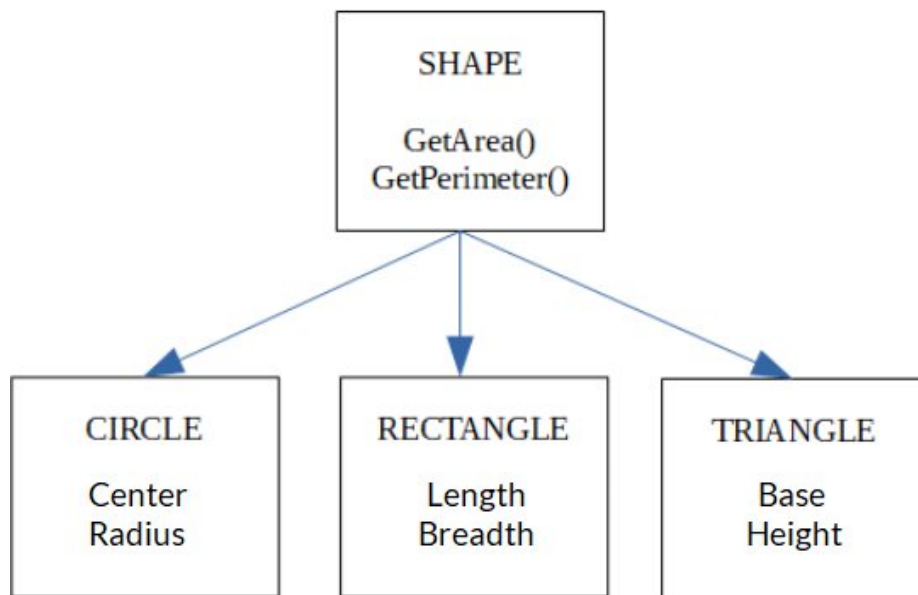
Similar attributes and methods are grouped together using a class. An object can be called as an instance of the class.

Inheritance

A new class can be derived from the original class. The derived class contains attributes and methods of the original class as well as its own.

Example

An Example of the Object Oriented data model is –



Shape, Circle, Rectangle and Triangle are all objects in this model.

Circle has the attributes Center and Radius.

Rectangle has the attributes Length and Breadth

Triangle has the attributes Base and Height.

The objects Circle, Rectangle and Triangle inherit from the object Shape.

Levels of abstraction

Data Abstraction is a process of hiding unwanted or irrelevant details from the end user. It provides a different view and helps in achieving data independence which is used to enhance the security of data.

The database systems consist of complicated data structures and relations. For users to access the data easily, these complications are kept hidden, and only the relevant part of the database is made accessible to the users through data abstraction.

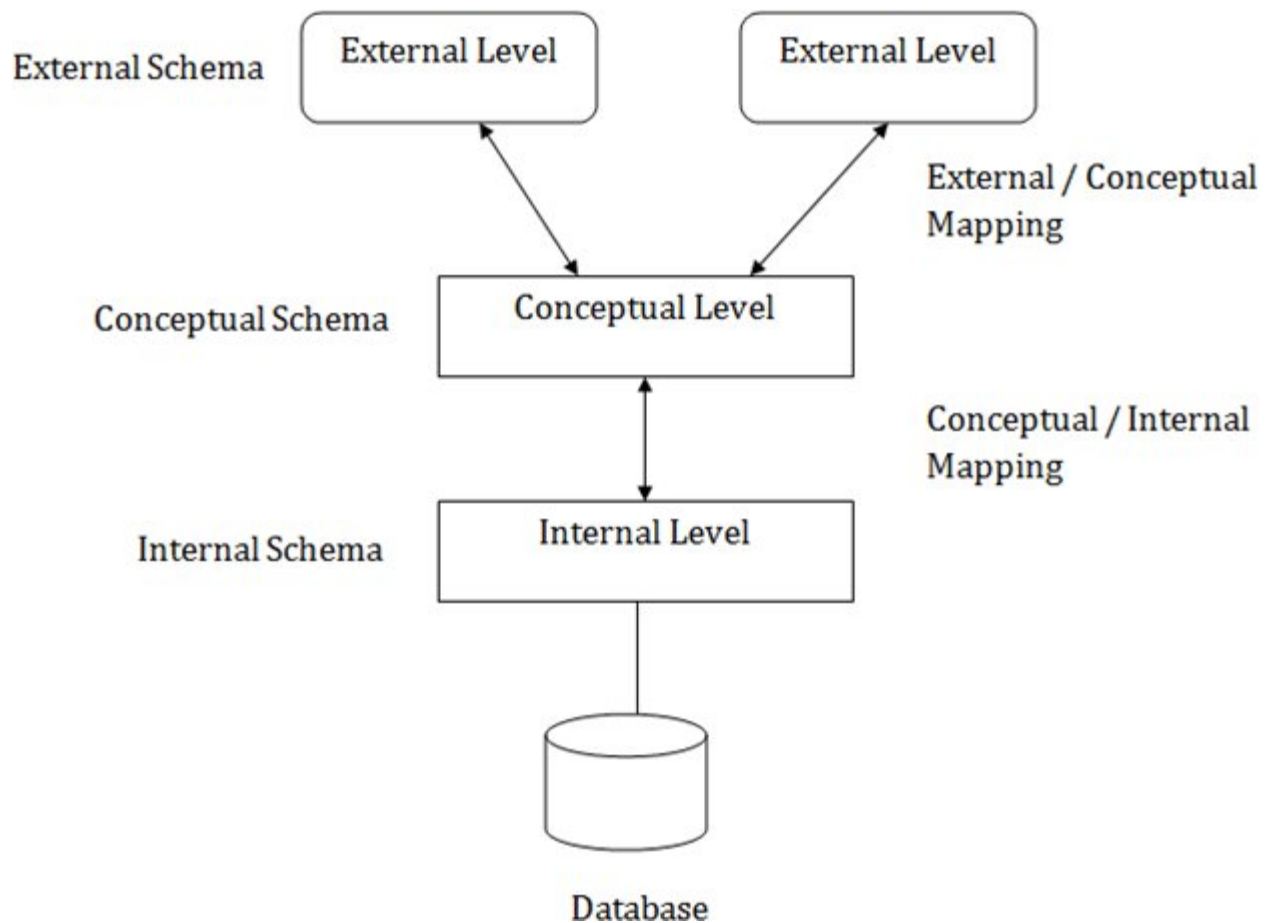
Levels of abstraction for DBMS

Database systems include complex data-structures. In terms of retrieval of data, reduce complexity in terms of usability of users and in order to make the system efficient, developers use levels of abstraction that hide irrelevant details from the users. Levels of abstraction simplify database design.

Mainly there are three levels of abstraction for DBMS, which are as follows –

- Physical or Internal Level
- Logical or Conceptual Level
- View or External Level

These levels are shown in the diagram below –



Data Independence

- Data independence can be explained using the three-schema architecture.
- Data independence refers to the characteristic of being able to modify the schema at one level of the database system without altering the schema at the next higher level.

There are two types of data independence:

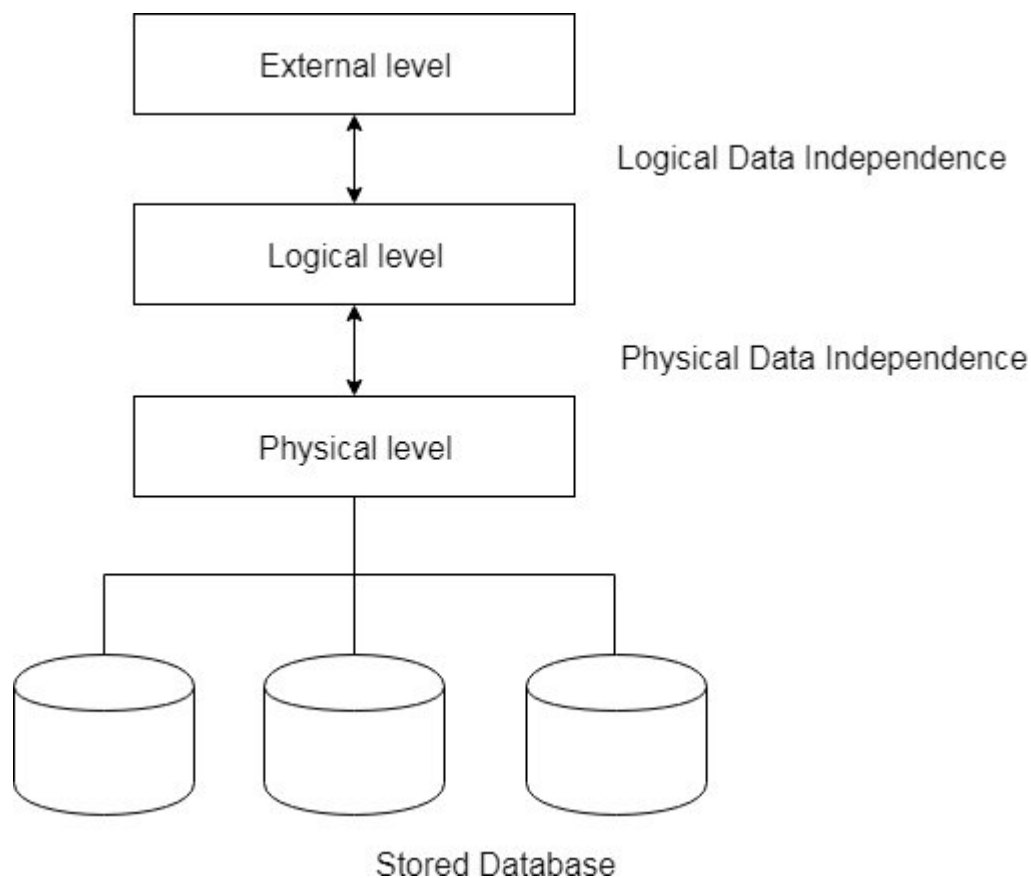
1. Logical Data Independence

- Logical data independence refers to the characteristic of being able to change the conceptual schema without having to change the external schema.
- Logical data independence is used to separate the external level from the conceptual view.
- If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.

- Logical data independence occurs at the user interface level.

2. Physical Data Independence

- Physical data independence can be defined as the capacity to change the internal schema without having to change the conceptual schema.
- If we do any changes in the storage size of the database system server, then the Conceptual structure of the database will not be affected.
- Physical data independence is used to separate conceptual levels from the internal levels.
- Physical data independence occurs at the logical interface level.



Structure of DBMS

DBMS

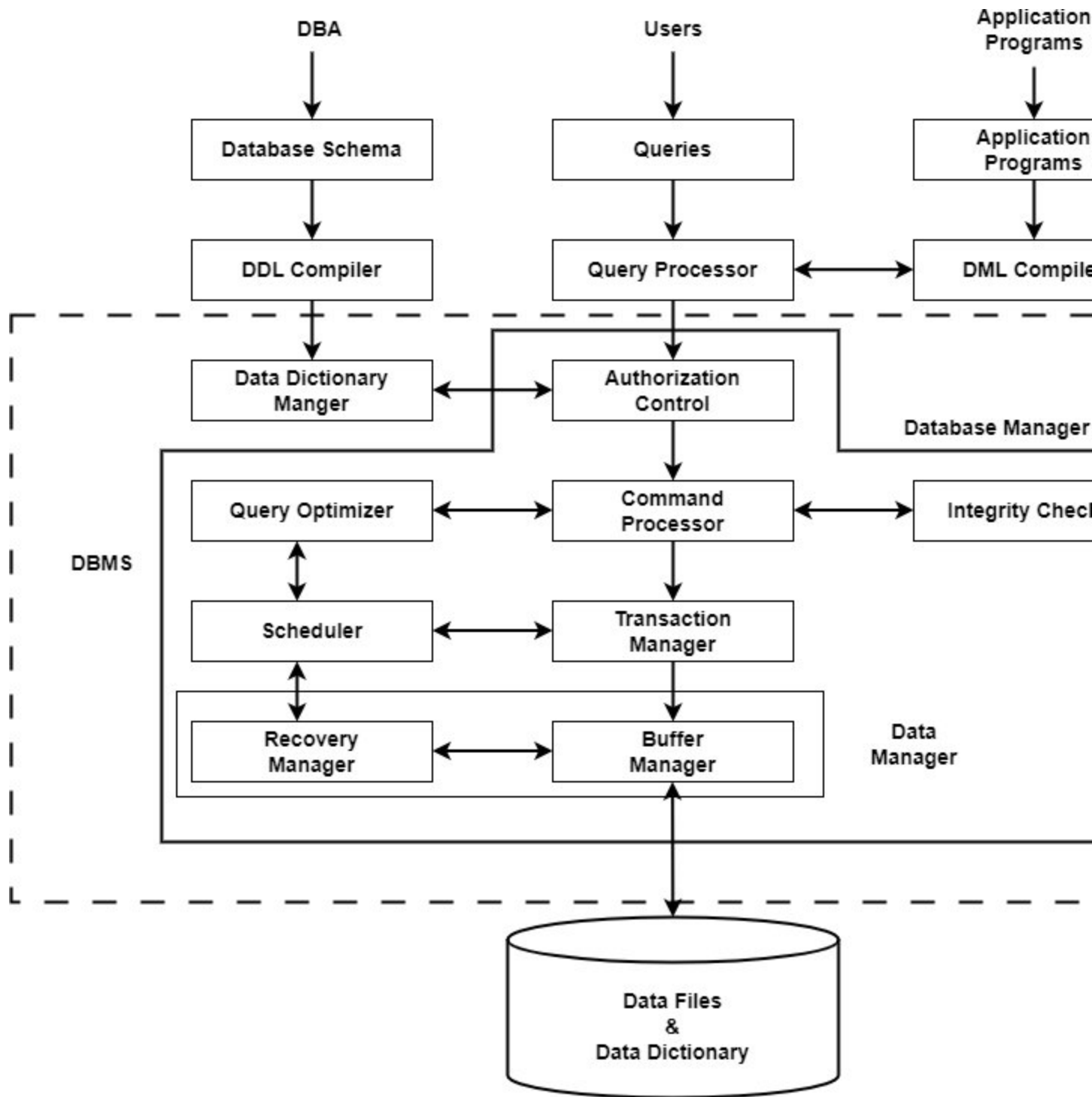
DBMS means Database Management System, which is a tool or software used to create the database or delete or manipulate the database. A software programme created to store, retrieve, query, and manage data is known as a Database Management System (DBMS). Data can be generated, read, updated, and destroyed by authorized entities thanks to user interfaces (UIs).

Because they give programmers, Database Managers, and end users a consolidated view of the data, Database Management Systems are crucial because they relieve applications and end users of the need to comprehend the physical location of the data. Application Programme Interfaces (APIs) manage internet requests and responses for particular sorts of data.

In marketing materials, the phrase "database as a service" (DBaaS) may be used to refer to both relational and non-relational DBMS components that are given via the internet.

Users of DBMSs include application programmers, Database Administrators (DBAs), and end users.

Database Administrators are typically the only people who work directly with a DBMS. Today, end users read and write to databases using front-end interfaces made by programmers, while programmers use cloud APIs to connect with DBMSs.



Three Parts that make up the Database System are:

- Query Processor

- Storage Manager
- Disk Storage

The explanations for these are provided below:

1. Query Processor

The query processing is handled by the query processor, as the name implies. It executes the user's query, to put it simply. In this way, the query processor aids the database system in making data access simple and easy. The query processor's primary duty is to successfully execute the query. The Query Processor transforms (or interprets) the user's application program-provided requests into instructions that a computer can understand.

Components of the Query Processor

- **DDL Interpreter:**

Data Definition Language is what DDL stands for. As implied by the name, the DDL Interpreter interprets DDL statements like those used in schema definitions (such as create, remove, etc.). This interpretation yields a set of tables that include the meta-data (data of data) that is kept in the data dictionary. Metadata may be stored in a data dictionary. In essence, it is a part of the disc storage that will be covered in a later section of this article.

- **DML Compiler:**

Compiler for DML Data Manipulation Language is what DML stands for. In keeping with its name, the DML Compiler converts DML statements like select, update, and delete into low-level instructions or simply machine-readable object code, to enable execution. The optimization of queries is another function of the DML compiler. Since a single question can typically be translated into a number of evaluation plans. As a result, some optimization is needed to select the evaluation plan with the lowest cost out of all the options. This process, known as query optimization, is exclusively carried out by the DML compiler. Simply put, query optimization determines the most effective technique to carry out a query.

- **Embedded DML Pre-compiler:**

Before the query evaluation, the embedded DML commands in the application program (such as SELECT, FROM, etc., in SQL) must be pre-compiled into standard procedural calls (program instructions that the host language can understand). Therefore, the DML statements which are embedded in an application program must be converted into routine calls by the Embedded DML Pre-compiler.

- **Query Optimizer:**

It starts by taking the evaluation plan for the question, runs it, and then returns the result. Simply said, the query evaluation engine evaluates the SQL commands used to access the database's contents before returning the result of the query. In a nutshell, it is in charge of analyzing the queries and running the object code that the DML Compiler produces. Apache Drill, Presto, and other Query Evaluation Engines are a few examples.

2. Storage Manager:

An application called Storage Manager acts as a conduit between the queries made and the data kept in the database. Another name for it is Database Control System. By applying the restrictions and running the DCL instructions, it keeps the database's consistency and integrity. It is in charge of retrieving, storing, updating, and removing data from the database.

Components of Storage Manager

Following are the components of Storage Manager:

- **Integrity Manager:**

Whenever there is any change in the database, the Integrity manager will manage the integrity constraints.

- **Authorization Manager:**

Authorization manager verifies the user that he is valid and authenticated for the specific query or request.

- **File Manager:**

All the files and data structure of the database are managed by this component.

- **Transaction Manager:**

It is responsible for making the database consistent before and after the transactions. Concurrent processes are generally controlled by this component.

- **Buffer Manager:**

The transfer of data between primary and main memory and managing the cache memory is done by the buffer manager.

3. Disk Storage

A DBMS can use various kinds of Data Structures as a part of physical system implementation in the form of disk storage.

Components of Disk Storage

Following are the components of Disk Manager:

- **Data Dictionary:**

It contains the metadata (data of data), which means each object of the database has some information about its structure. So, it creates a repository which contains the details about the structure of the database object.

- **Data Files:**

- This component stores the data in the files.

- **Indices:** These indices are used to access and retrieve the data in a very fast and efficient way.

Database Designing

The next step involves designing the database considering the user-based requirements and splitting them out into various models so that load or heavy dependencies on a single aspect are not imposed. Therefore, there has been some model-centric approach and that's where logical and physical models play a crucial role.

Physical Model - The physical model is concerned with the practices and implementations of the logical model.

Logical Model - This stage is primarily concerned with developing a model based on the proposed requirements. The entire model is designed on paper without any implementation or adopting DBMS considerations.

Implementation

The last step covers the implementation methods and checking out the behavior that matches our requirements. It is ensured with continuous integration testing of the database with different data sets and conversion of data into machine understandable language. The manipulation of data is primarily focused on these steps where queries are made to run and check if the application is designed satisfactorily or not.

Data conversion and loading - This section is used to import and convert data from the old to the new system.

Testing - This stage is concerned with error identification in the newly implemented system. Testing is a crucial step because it checks the database directly and compares the requirement specifications.

Database Design Process

The process of designing a database carries various conceptual approaches that are needed to be kept in mind. An ideal and well-structured database design must be able to:

1. Save disk space by eliminating redundant data.
2. Maintains data integrity and accuracy.
3. Provides data access in useful ways.
4. Comparing Logical and Physical data models.

Logical

A logical data model generally describes the data in as many details as possible, without having to be concerned about the physical implementations in the database. Features of logical data model might include:

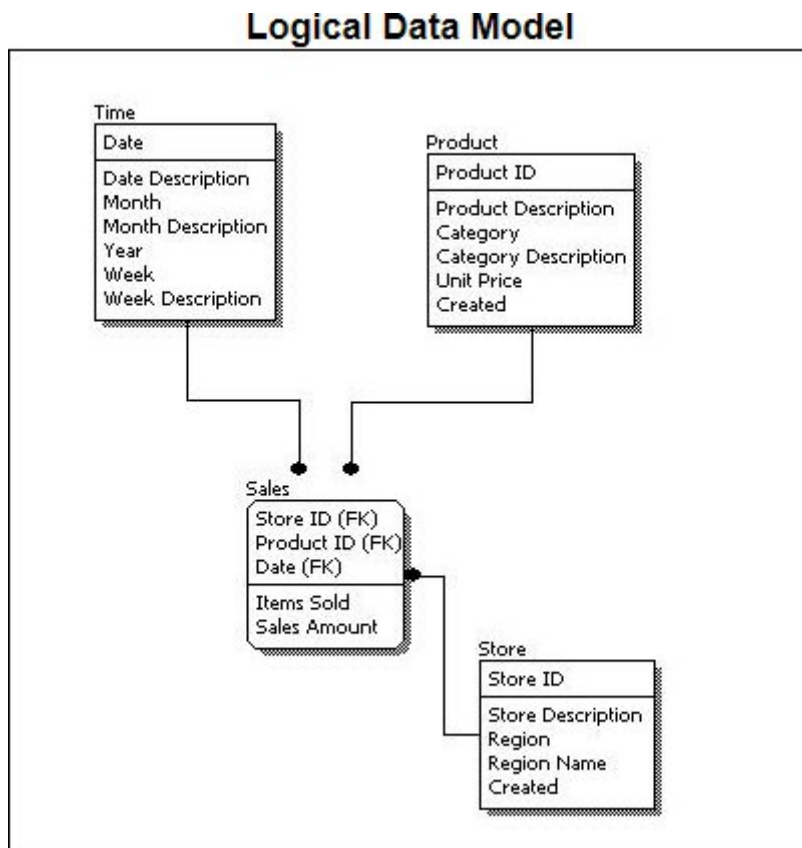
1. All the entities and relationships amongst them.
2. Each entity has well-specified attributes.
3. The primary key for each entity is specified.
4. Foreign keys which are used to identify a relationship between different entities are specified.
5. Normalization occurs at this level.

A logical model can be designed using the following approach:

1. Specify all the entities with primary keys.
2. Specify concurrent relationships between different entities.
3. Figure out each entity attributes
4. Resolve many-to-many relationships.
5. Carry out the process of normalization.

Also, one important factor after following the above approach is to critically examine the design based on requirement gathering. If the above steps are strictly followed, there are chances of creating a highly efficient database design that follows the native approach.

To understand these points, see the image below to get a clear picture.



If we compare the logical data model as shown in the figure above with some sample data in the diagram, we can come up with facts that in a conceptual data model there are no presence of a primary key whereas a logical data model has primary keys for all of its attributes. Also, logical data model the cover relationship between different entities and carries room for foreign keys to establish relationships among them.

Physical

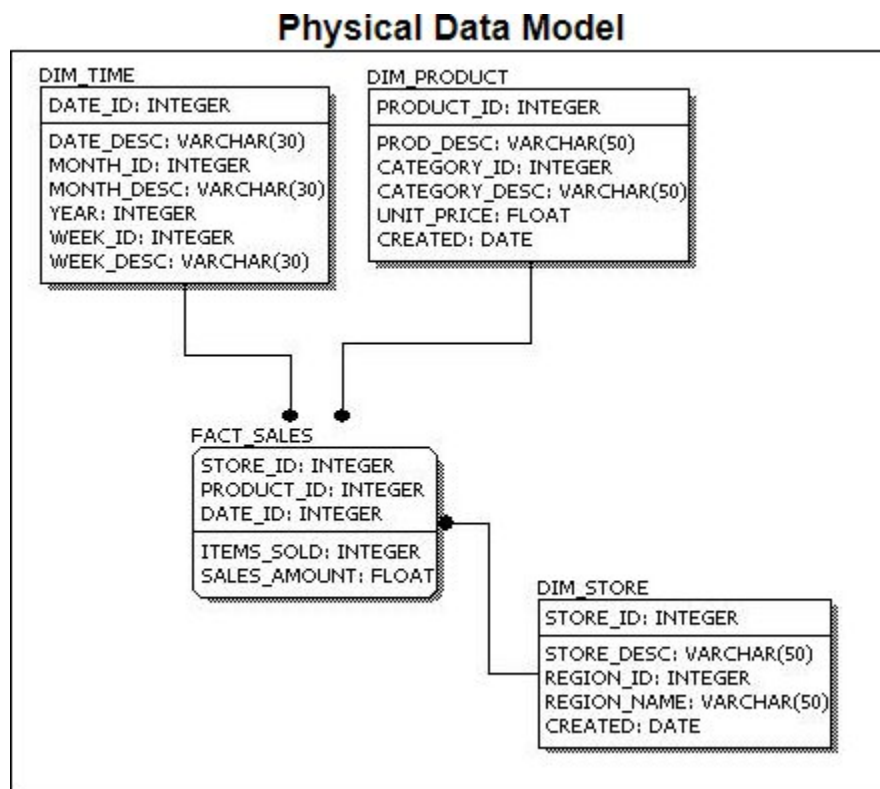
A Physical data mode generally represents how the approach or concept of designing the database. The main purpose of the physical data model is to show all the **structures** of the table including the **column name, column data type, constraints, keys(primary and foreign)**, and the relationship among tables. The following are the features of a physical data model:

1. Specifies all the columns and tables.
2. Specifies foreign keys that usually define the relationship between tables.
3. Based on user requirements, de-normalization might occur.

4. Since the physical consideration is taken into account so there will straightforward reasons for difference than a logical model.
5. Physical models might be different for different RDBMS. For example, the data type column may be different in MySQL and SQL Server.

While designing a physical data model, the following points should be taken into consideration:

1. Convert the entities into tables.
2. Convert the defined relationships into foreign keys.
3. Convert the data attributes into columns.
4. Modify the data model constraints based on physical requirements.

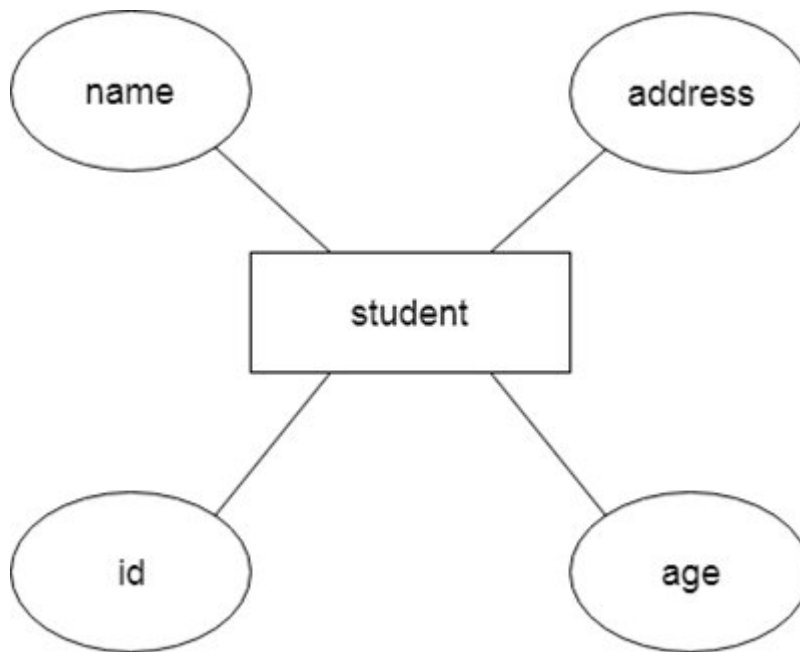


Comparing this physical data model with the logical with the previous logical model, we might conclude the differences that in a physical database entity names are considered table names and attributes are considered column names. Also, the data type of each column is defined in the physical model depending on the actual database used.

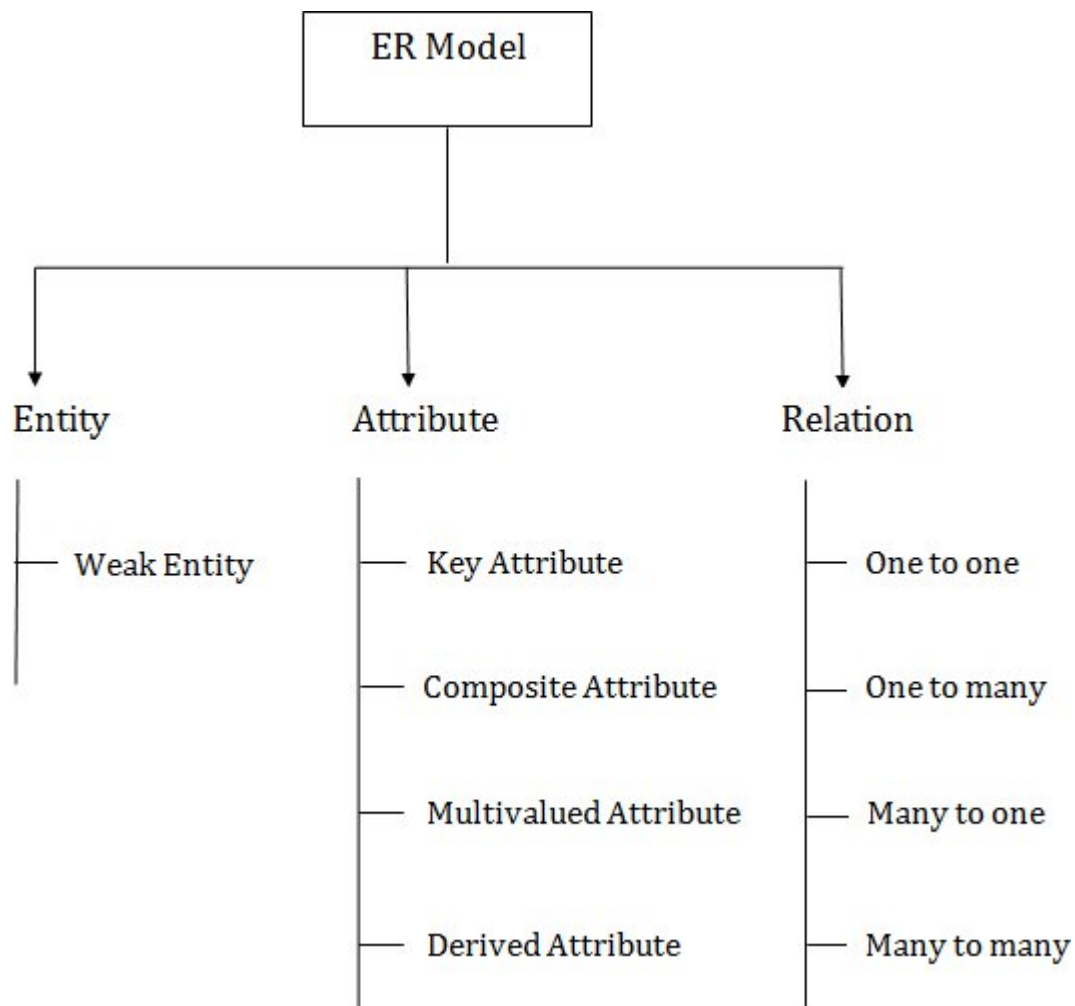
ER (Entity Relationship) Diagram in DBMS

- ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
- In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

For example, Suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc and there will be a relationship between them.



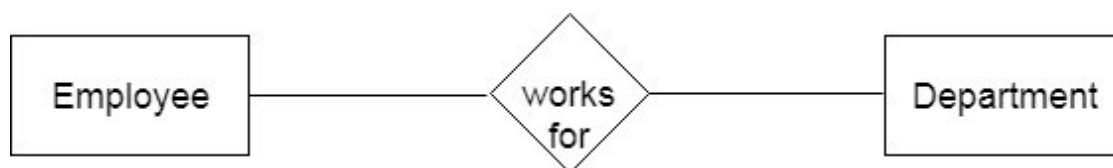
Component of ER Diagram



1. Entity:

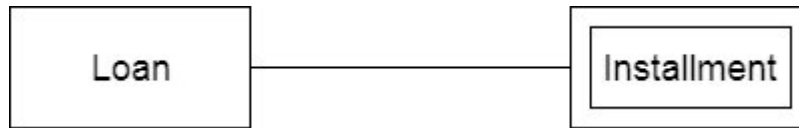
An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.

Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity.



a. Weak Entity

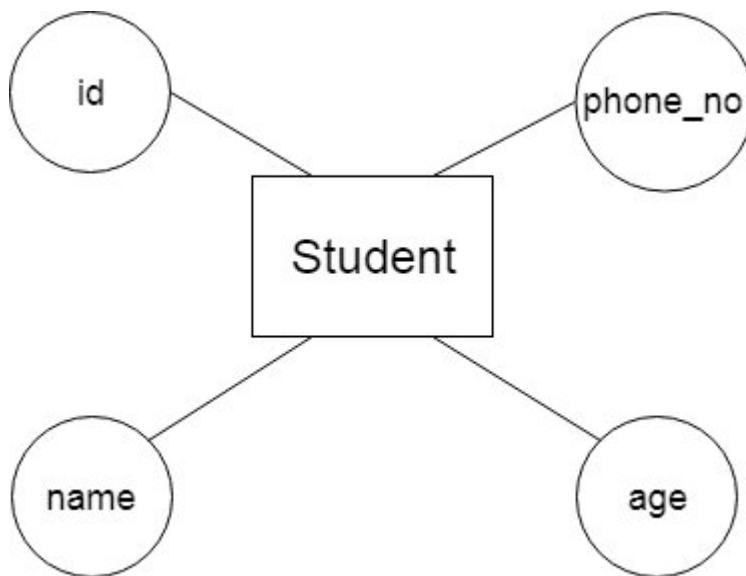
An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.



2. Attribute

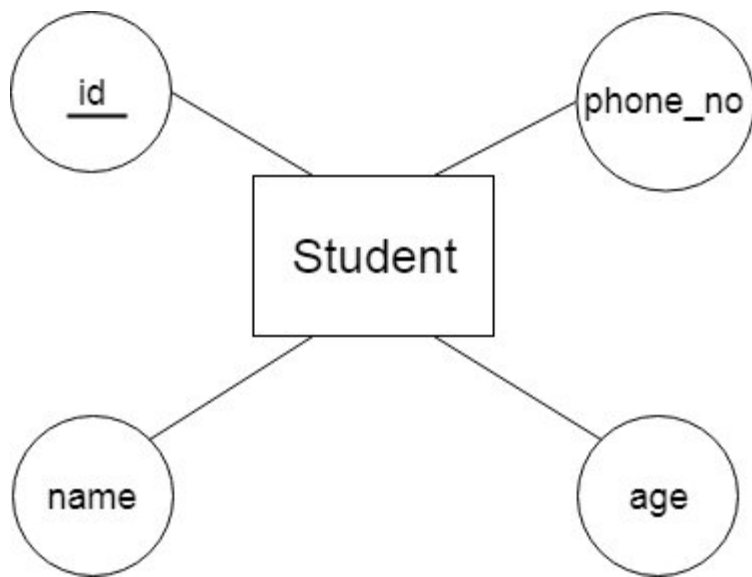
The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.

For example, id, age, contact number, name, etc. can be attributes of a student.



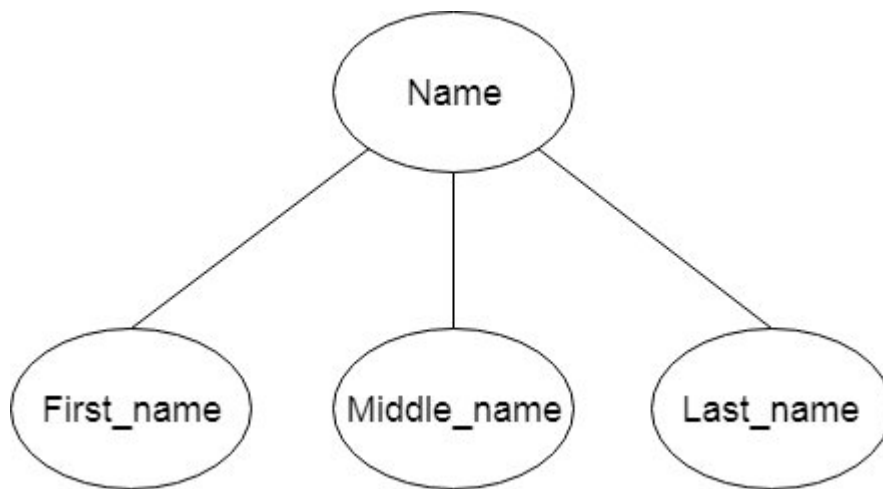
a. Key Attribute

The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.



b. Composite Attribute

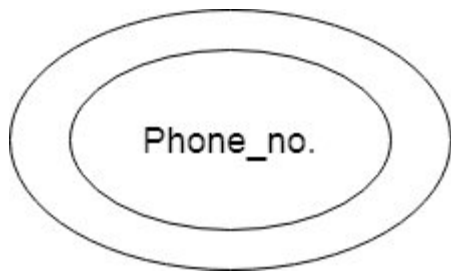
An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.



c. Multivalued Attribute

An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.

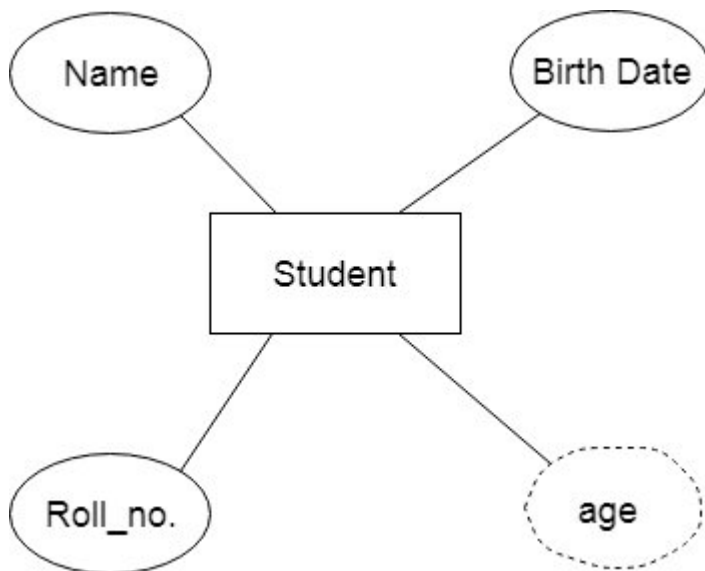
For example, a student can have more than one phone number.



d. Derived Attribute

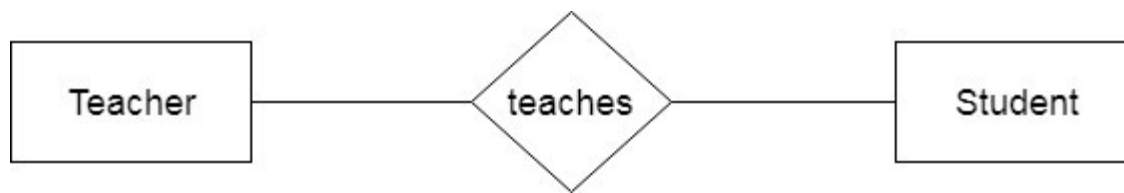
An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.

For example, A person's age changes over time and can be derived from another attribute like Date of birth.



3. Relationship

A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.



Types of relationship are as follows:

a. One-to-One Relationship

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

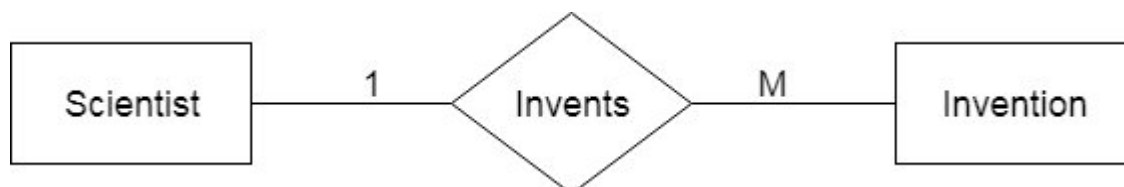
For example, A female can marry to one male, and a male can marry to one female.



to-many relationship

When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a one-to-many relationship.

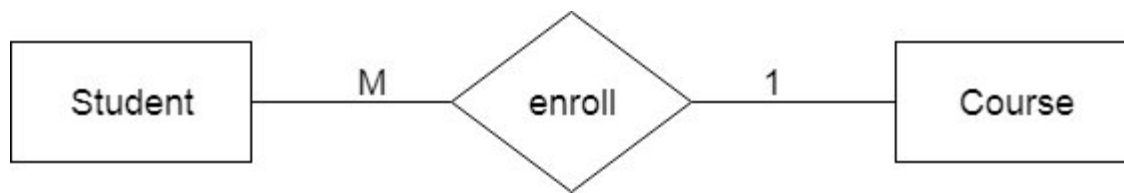
For example, Scientist can invent many inventions, but the invention is done by the only specific scientist



c. Many-to-one relationship

When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.

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



d. Many-to-many relationship

When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.

For example, Employee can assign by many projects and project can have many employees.



Cardinality in DBMS (Mapping Constraints)

DBMS

DBMS stands for Database Management System, which is a tool, or a software used to do various operations on a Database like the Creation of the Database, Deletion of the Database, or Updating the current Database. To simplify processing and data querying, the most popular types of Databases currently in use typically model their data as rows and columns in a set of tables. The data may then be handled, updated, regulated, and structured with ease. For writing and querying data, most Databases employ Structured Query Language (SQL).

Cardinality

Cardinality means how the entities are arranged to each other or what is the relationship structure between entities in a relationship set. In a Database Management System, Cardinality represents a number that denotes how many times an entity is participating with another entity in a relationship set. The Cardinality of DBMS is a very important attribute in representing the structure of a Database. In a table, the number of rows or tuples represents the Cardinality.

Cardinality Ratio

Cardinality ratio is also called **Cardinality Mapping**, which represents the mapping of one entity set to another entity set in a relationship set. We generally take the example of a binary relationship set where two entities are mapped to each other.

Cardinality is very important in the Database of various businesses. For example, if we want to track the purchase history of each customer then we can use the one-to-many cardinality to find the data of a specific customer. The Cardinality model can be used in Databases by Database Managers for a variety of purposes, but corporations often use it to evaluate customer or inventory data.

There are four types of Cardinality Mapping in Database Management Systems:

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

One to One

One to one cardinality is represented by a **1:1** symbol. In this, there is at most one relationship from one entity to another entity. There are a lot of examples of one-to-one cardinality in real life databases.

For example, one student can have only one student id, and one student id can belong to only one student. So, the relationship mapping between student and student id will be one to one cardinality mapping.

Another example is the relationship between the director of the school and the school because one school can have a maximum of one director, and one director can belong to only one school.

There are four types of Cardinality Mapping in Database Management Systems:

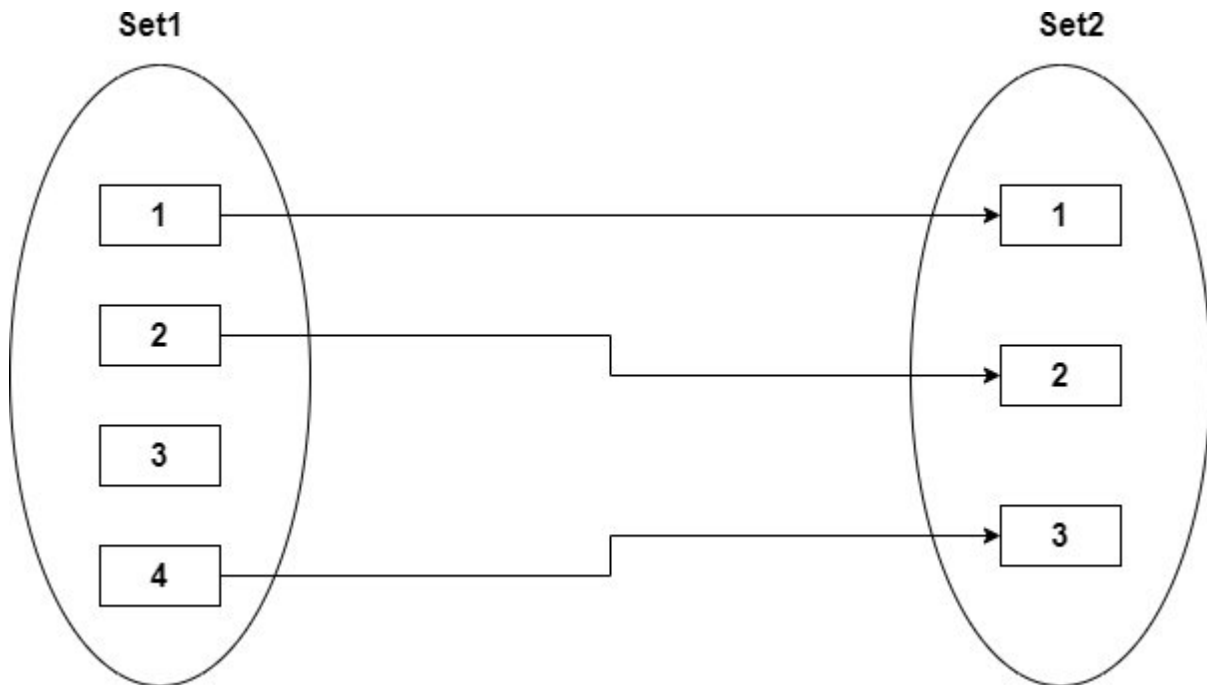
1. One to one
2. Many to one
3. One to many
4. Many to many

One to One

One to one cardinality is represented by a 1:1 symbol. In this, there is at most one relationship from one entity to another entity. There are a lot of examples of one-to-one cardinality in real life databases.

For example, one student can have only one student id, and one student id can belong to only one student. So, the relationship mapping between student and student id will be one to one cardinality mapping.

Another example is the relationship between the director of the school and the school because one school can have a maximum of one director, and one director can belong to only one school. .

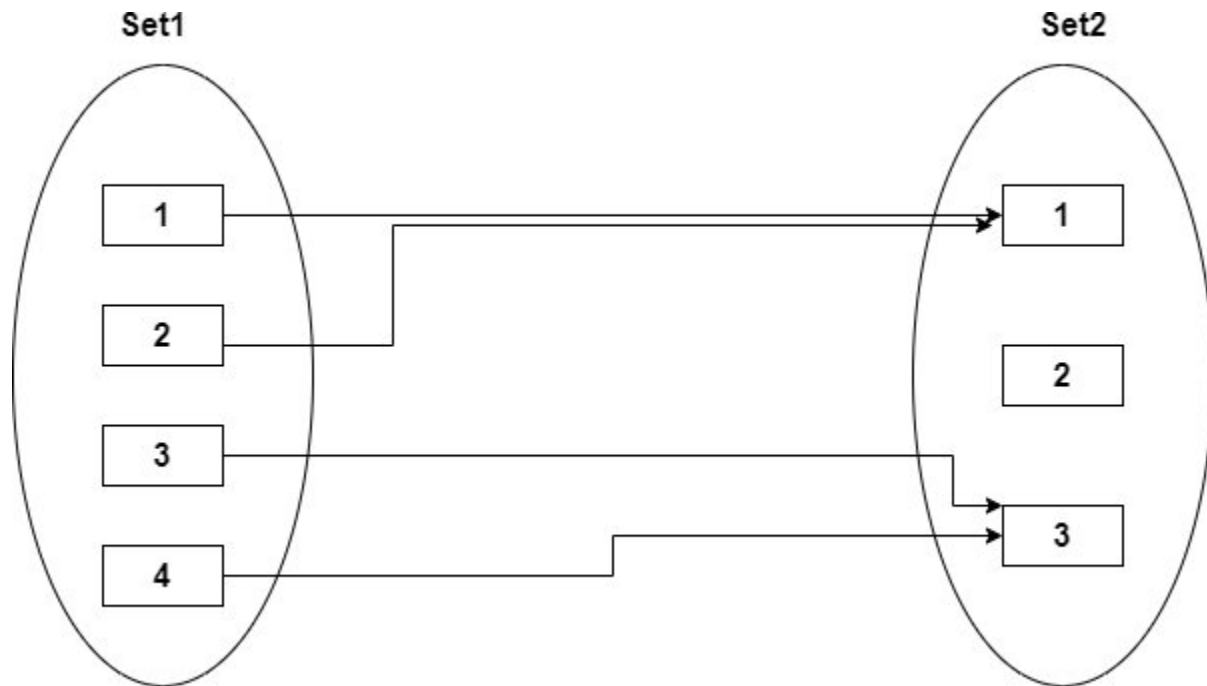


Many to One Cardinality:

In many to one cardinality mapping, from set 1, there can be multiple sets that can make relationships with a single entity of set 2. Or we can also describe it as from set 2, and one entity can make a relationship with more than one entity of set 1.

One to one Cardinality is the subset of Many to one Cardinality. It can be represented by **M:1**.

For example, there are multiple patients in a hospital who are served by a single doctor, so the relationship between patients and doctors can be represented by Many to one Cardinality.

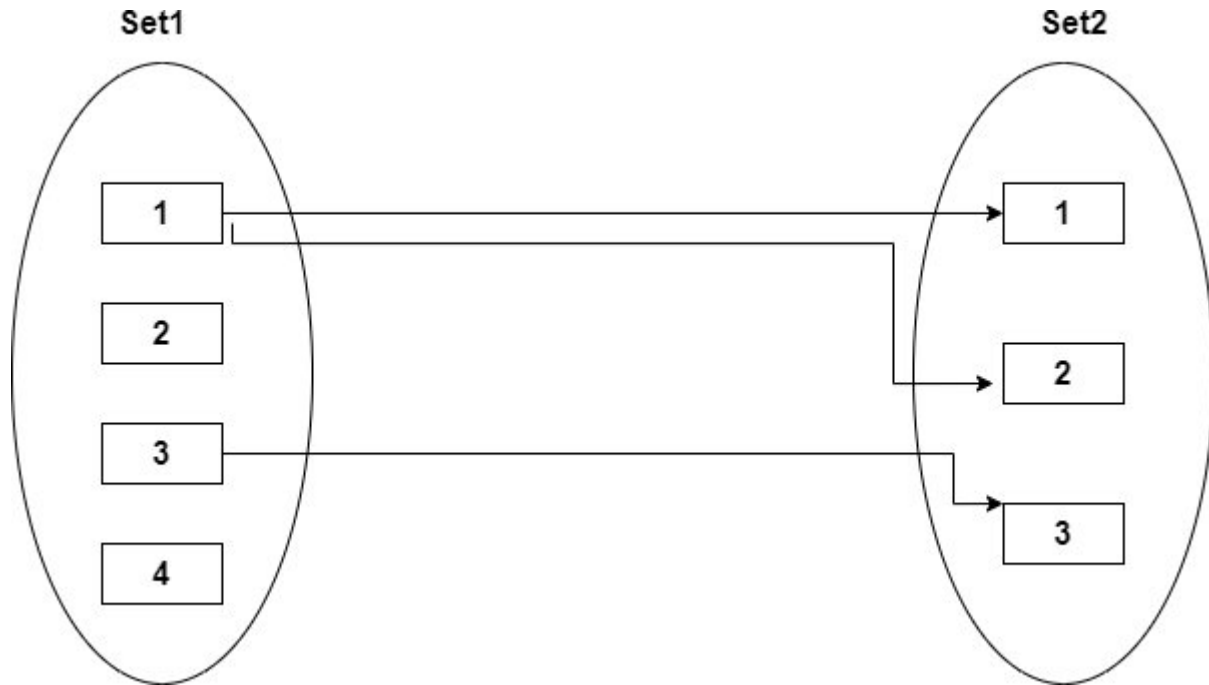


One to Many Cardinalities:

In One-to-many cardinality mapping, from set 1, there can be a maximum single set that can make relationships with a single or more than one entity of set 2. Or we can also describe it as from set 2, more than one entity can make a relationship with only one entity of set

ne to one cardinality is the subset of One-to-many Cardinality. It can be represented by **1: M**.

For Example, in a hospital, there can be various compounders, so the relationship between the hospital and compounders can be mapped through One-to-many Cardinality.



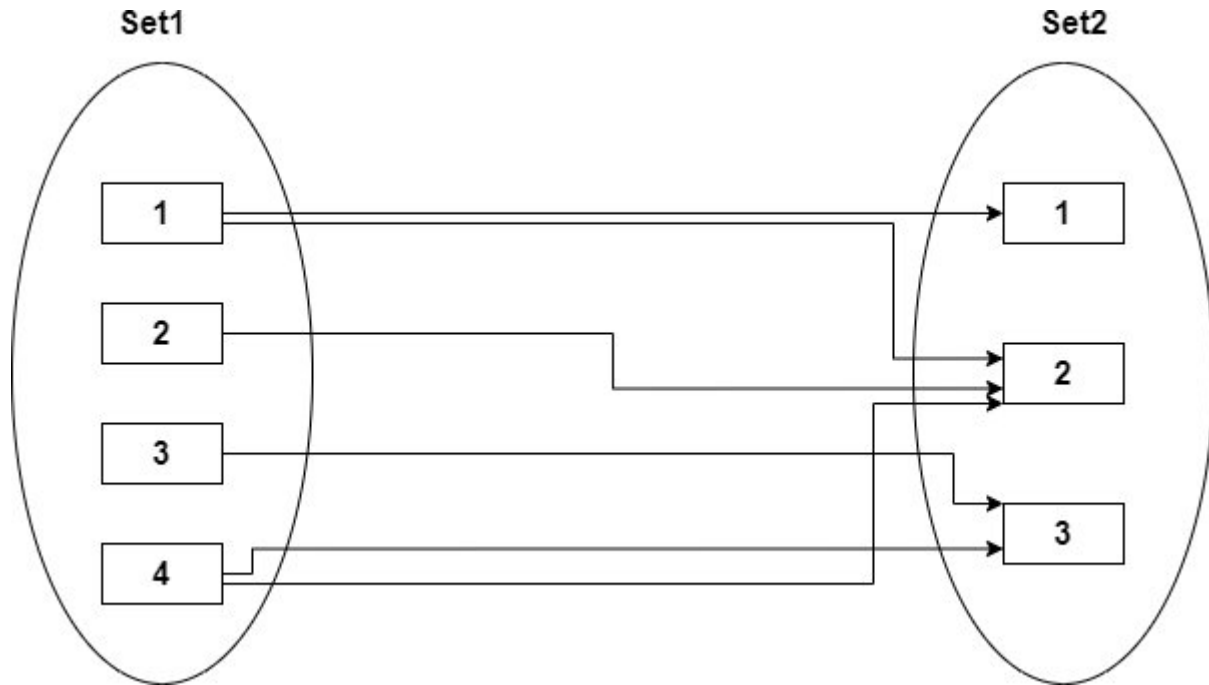
Many to Many Cardinalities:

In many, many cardinalities mapping, there can be one or more than one entity that can associate with one or more than one entity of set 2. In the same way from the end of set 2, one or more than one entity can make a relation with one or more than one entity of set 1.

It is represented by **M: N** or **N: M**.

One to one cardinality, One to many cardinalities, and Many to one cardinality is the subset of the many to many cardinalities.

For Example, in a college, multiple students can work on a single project, and a single student can also work on multiple projects. So, the relationship between the project and the student can be represented by many to many cardinalities.



Entity Set: An Entity is an object of Entity Type and a set of all entities is called an entity set. For Example, E1 is an entity having Entity Type Student and the set of all students is called Entity Set. In ER diagram, Entity Type is represented as:



Entity Type

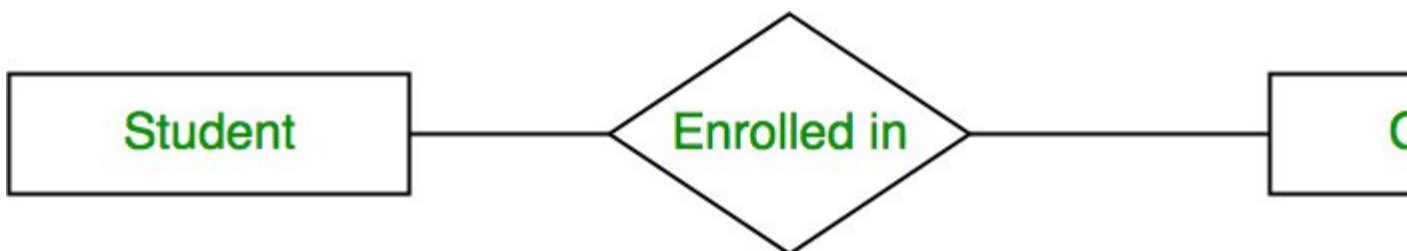


Entity Set

Entity Set

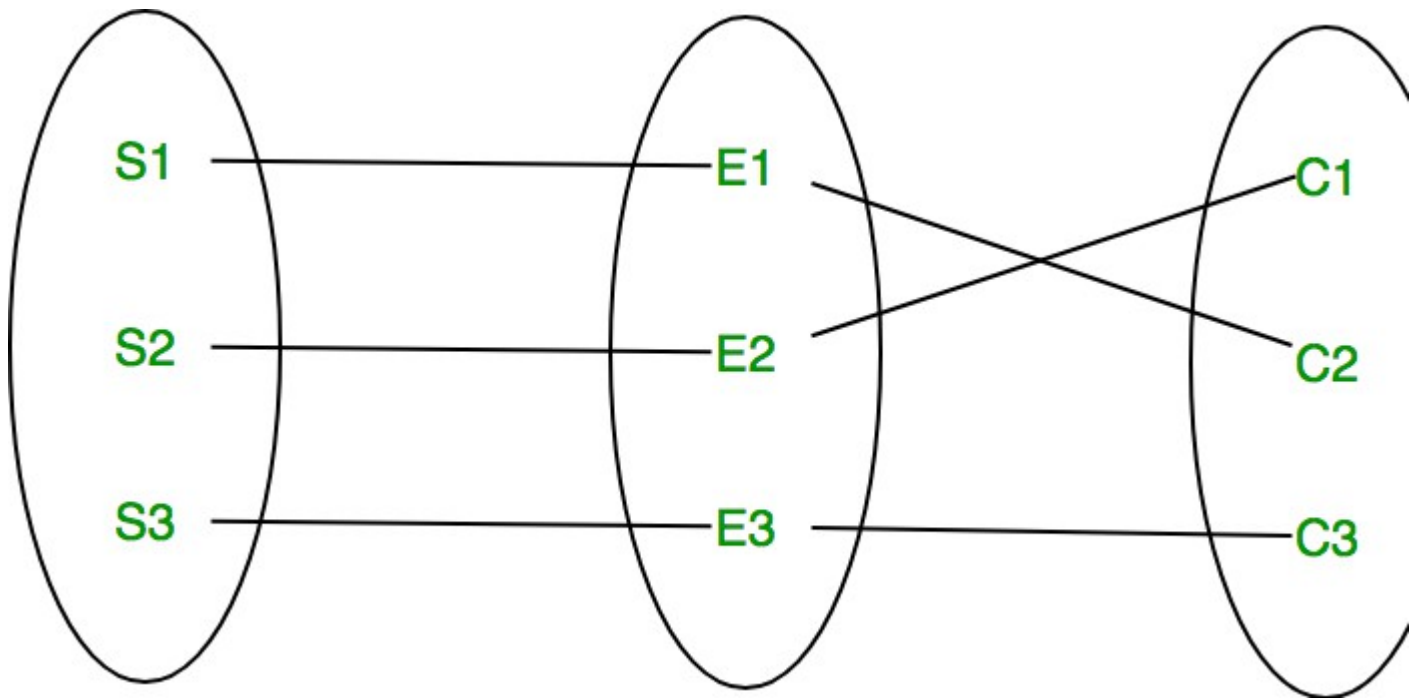
Relationship Type and Relationship Set

A Relationship Type represents the association between entity types. For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, the relationship type is represented by a diamond and connecting the entities with lines.



Entity-Relationship Set

A set of relationships of the same type is known as a relationship set. The following relationship set depicts S1 as enrolled in C2, S2 as enrolled in C1, and S3 as registered in C3.

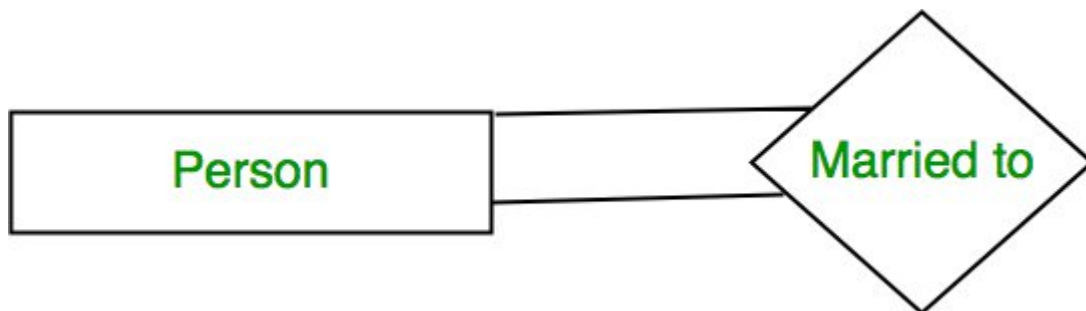


Relationship Set

Degree of a Relationship Set

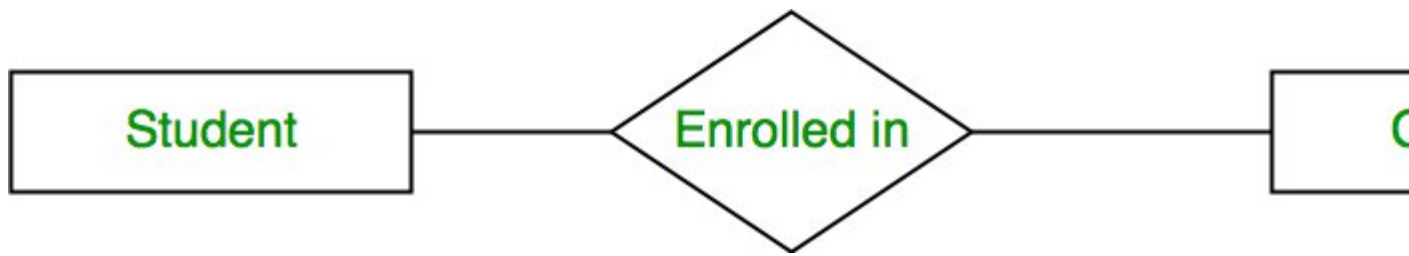
The number of different entity sets participating in a relationship set is called the [degree of a relationship set](#).

1. Unary Relationship: When there is only ONE entity set participating in a relation, the relationship is called a unary relationship. For example, one person is married to only one person.



Unary Relationship

2. Binary Relationship: When there are TWO entities set participating in a relationship, the relationship is called a binary relationship. For example, a Student is enrolled in a Course.



Binary Relationship

3. n-ary Relationship: When there are n entities set participating in a relation, the relationship is called an n-ary relationship.

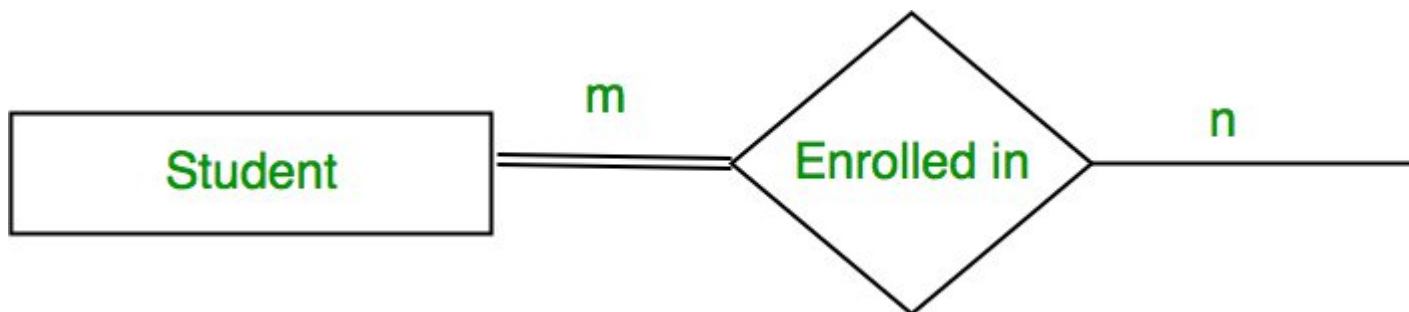
Participation Constraint

Participation Constraint is applied to the entity participating in the relationship set.

1. Total Participation – Each entity in the entity set must participate in the relationship. If each student must enroll in a course, the participation of students will be total. Total participation is shown by a double line in the ER diagram.

2. Partial Participation – The entity in the entity set may or may NOT participate in the relationship. If some courses are not enrolled by any of the students, the participation in the course will be partial.

The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation.



Symbols Used in ER Model

ER Model is used to model the logical view of the system from a data perspective which consists of these symbols:

- **Rectangles:** Rectangles represent Entities in ER Model.
- **Ellipses:** Ellipses represent Attributes in ER Model.
- **Diamond:** Diamonds represent Relationships among Entities.

- **Lines:** Lines represent attributes to entities and entity sets with other relationship types.
- **Double Ellipse:** Double Ellipses represent [Multi-Valued Attributes](#).
- **Double Rectangle:** Double Rectangle represents a Weak Entity.

Keys

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

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

STUDENT
ID
Name
Address
Course

PERSON
Name
DOB
Passport, Number
License_Number
SSN

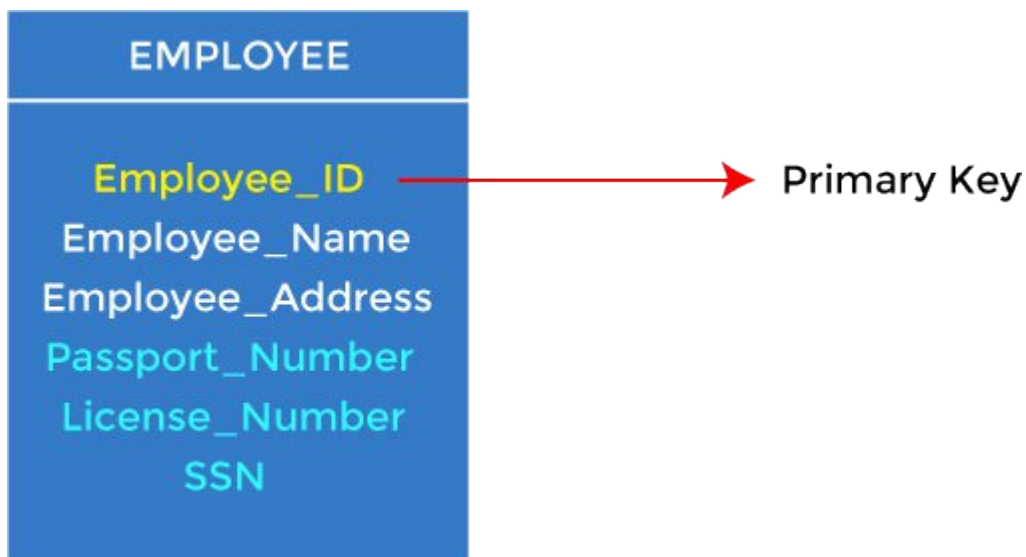
Types of keys:

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.

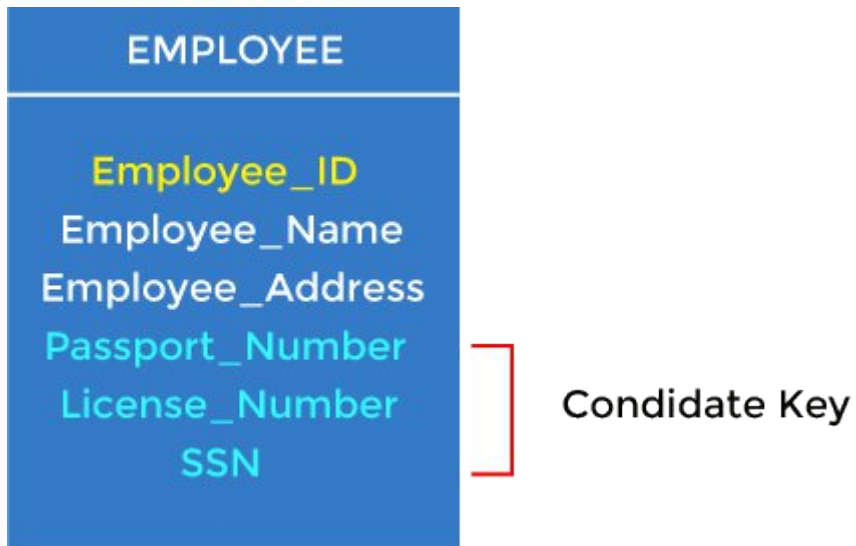


2. Candidate key

- A candidate key is an attribute or set of attributes that can uniquely identify a tuple.

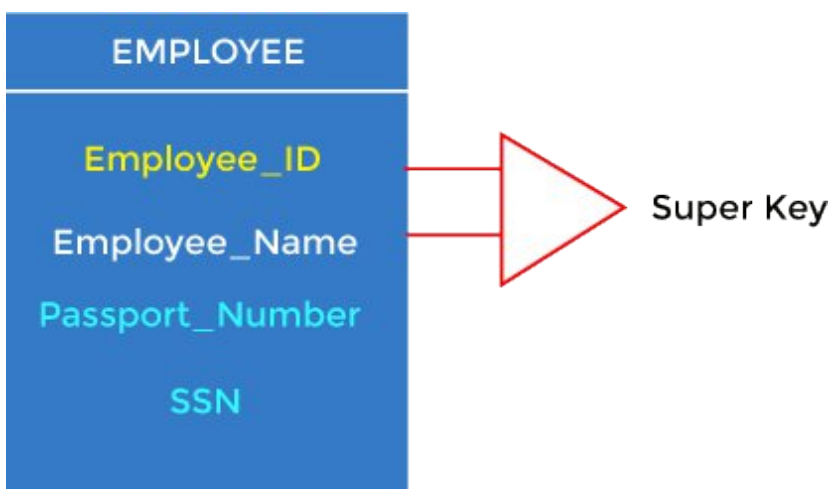
- Except for the primary key, the remaining attributes are considered a candidate key. The candidate keys are as strong as the primary key.

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.

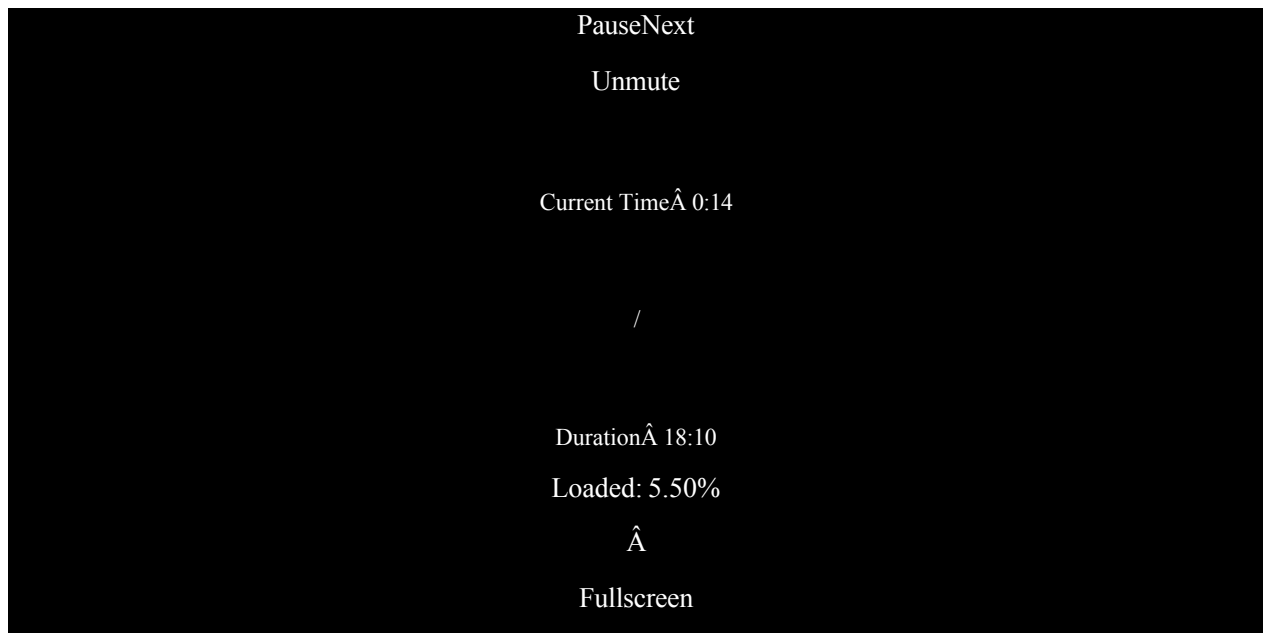


3. Super Key

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



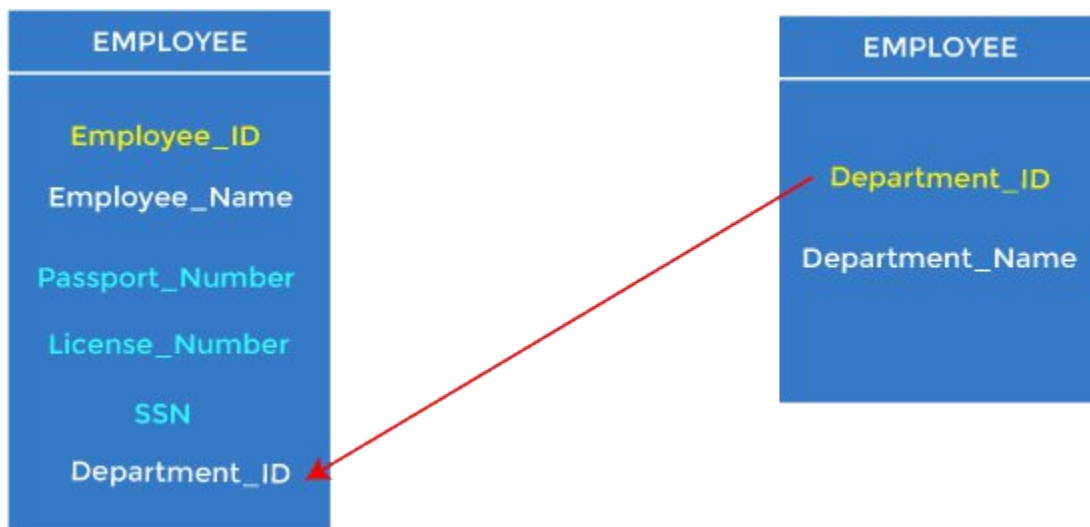
For example: In the above EMPLOYEE table, for(EMPLOYEE_ID, EMPLOYEE_NAME), the name of two employees can be the same, but their EMPLOYEE_ID can't be the same. Hence, this combination can also be a key.



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

4. Foreign key

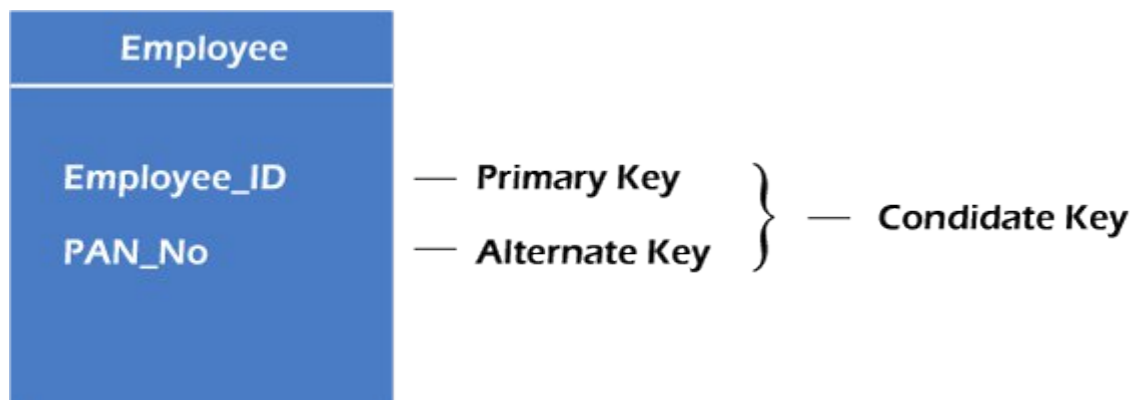
- Foreign keys are the column of the table used to point to the primary key of another table.
- Every employee works in a specific department in a company, and employee and department are two different entities. So we can't store the department's information in the employee table. That's why we link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

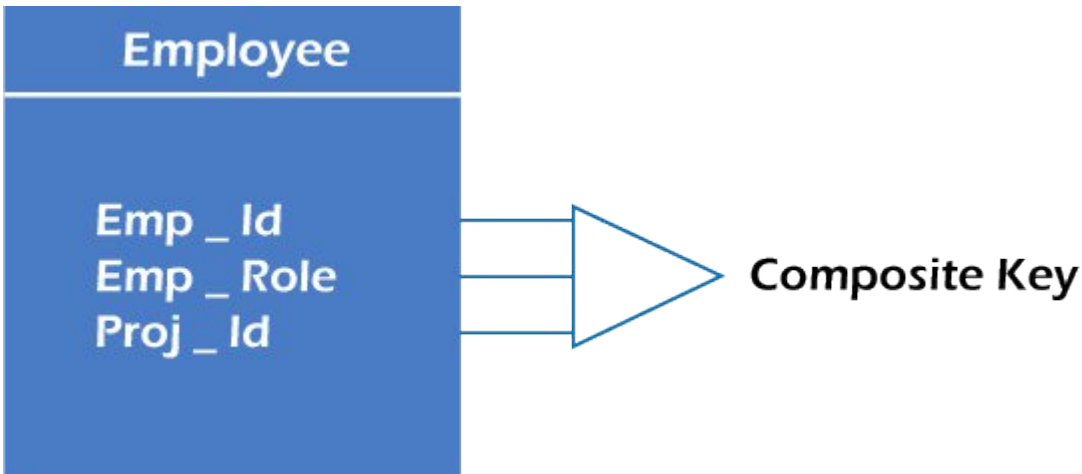
There may be one or more attributes or a combination of attributes that uniquely identify each tuple in a relation. These attributes or combinations of the attributes are called the candidate keys. One key is chosen as the primary key from these candidate keys, and the remaining candidate key, if it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

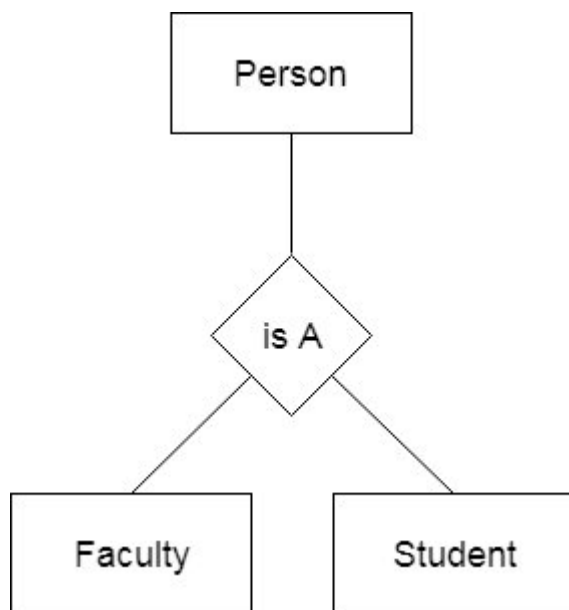
The key created using arbitrarily assigned data are known as artificial keys. These keys are created when a primary key is large and complex and has no relationship with many other relations. The data values of the artificial keys are usually numbered in a serial order.

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.

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 superclass 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 superclass.

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

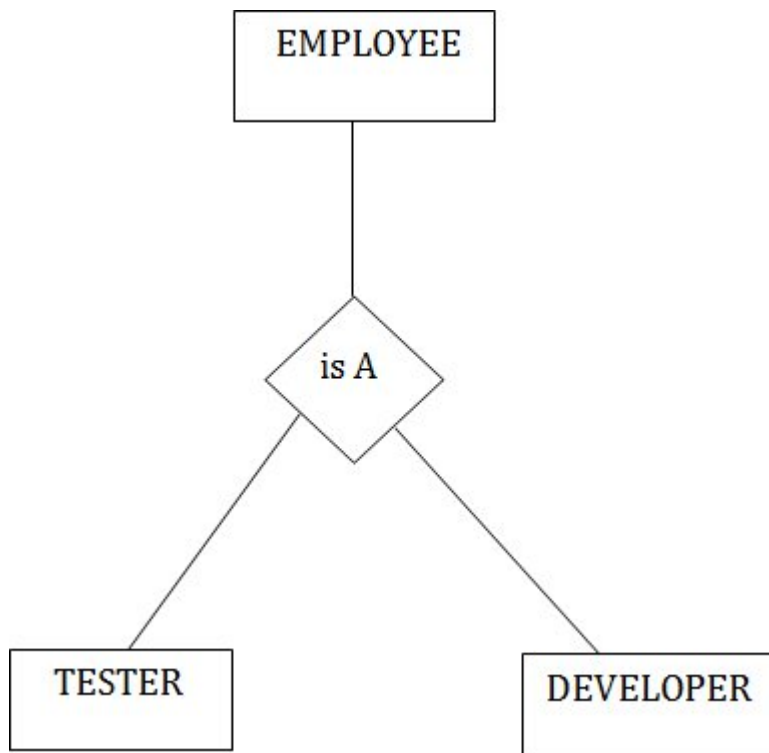


Specialization

- 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 superclass is defined first, the subclass and its related attributes are defined next, and relationship set are then added.

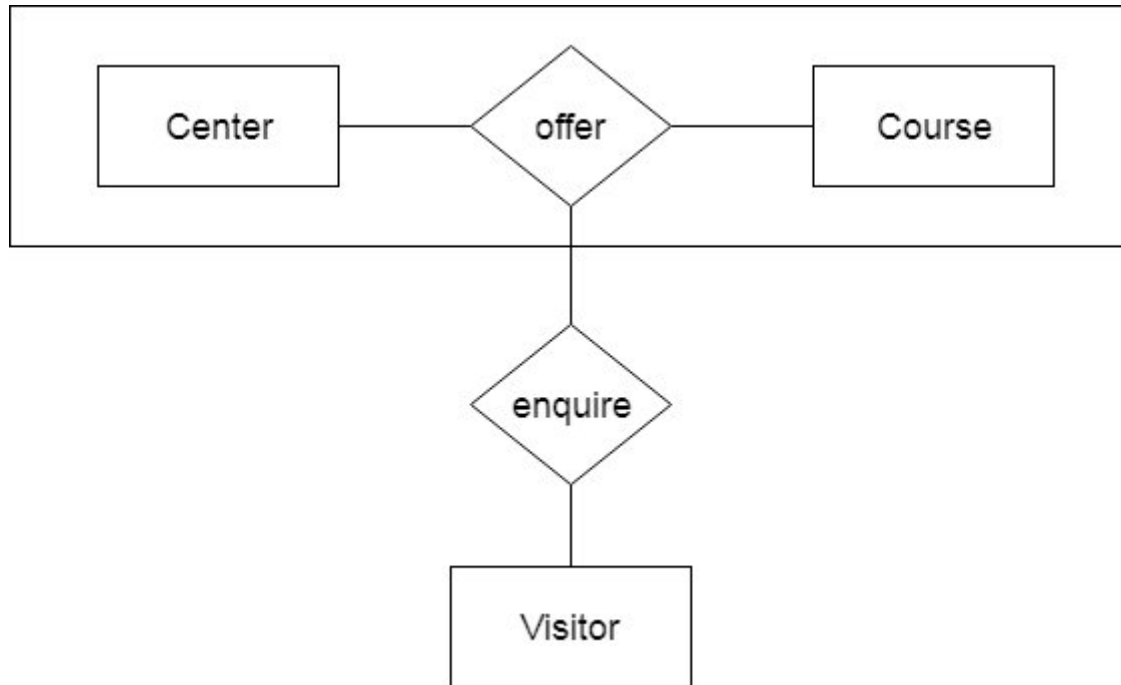
For example: In an Employee management system, EMPLOYEE entity can be specialized as TESTER or DEVELOPER based on what role they play in the company.



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

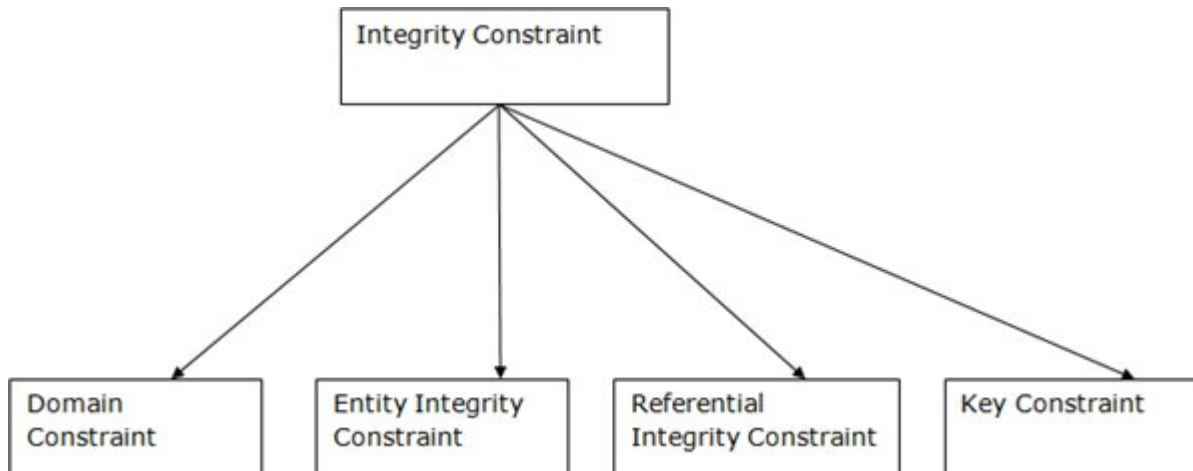


UNIT 2

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	1 st	17
1001	Johnson	2 nd	24
1002	Leonardo	5 th	21
1003	Kate	3 rd	19
1004	Morgan	8 th	A

Not allowed. Because AGE is an integer attribute

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)

<u>D_No</u>	D_Location
11	Mumbai
24	Delhi
13	Noida

Primary Key

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	1 st	17
1001	Johnson	2 nd	24
1002	Leonardo	5 th	21
1003	Kate	3 rd	19
1002	Morgan	8 th	22

Not allowed. Because all row must be unique

Ways to Enforce Integrity Constraints

There are several ways to enforce integrity constraints in a DBMS:

- **Declarative referential integrity:** This method involves specifying the integrity constraints at the time of database design and allowing the DBMS to enforce them automatically.
- **Triggers:** A trigger is a special type of stored procedure that is executed automatically by the DBMS when certain events occur (such as inserting, updating, or deleting data). Triggers can be used to enforce integrity constraints by checking for and rejecting invalid data.
- **Stored procedures:** A stored procedure is a pre-defined set of SQL statements that can be executed as a single unit. Stored procedures can be used to enforce integrity constraints by performing checks on the data before it is inserted, updated, or deleted.
- **Application-level code:** Integrity constraints can also be enforced at the application level by writing code to check for and reject invalid data before it is entered into the database.

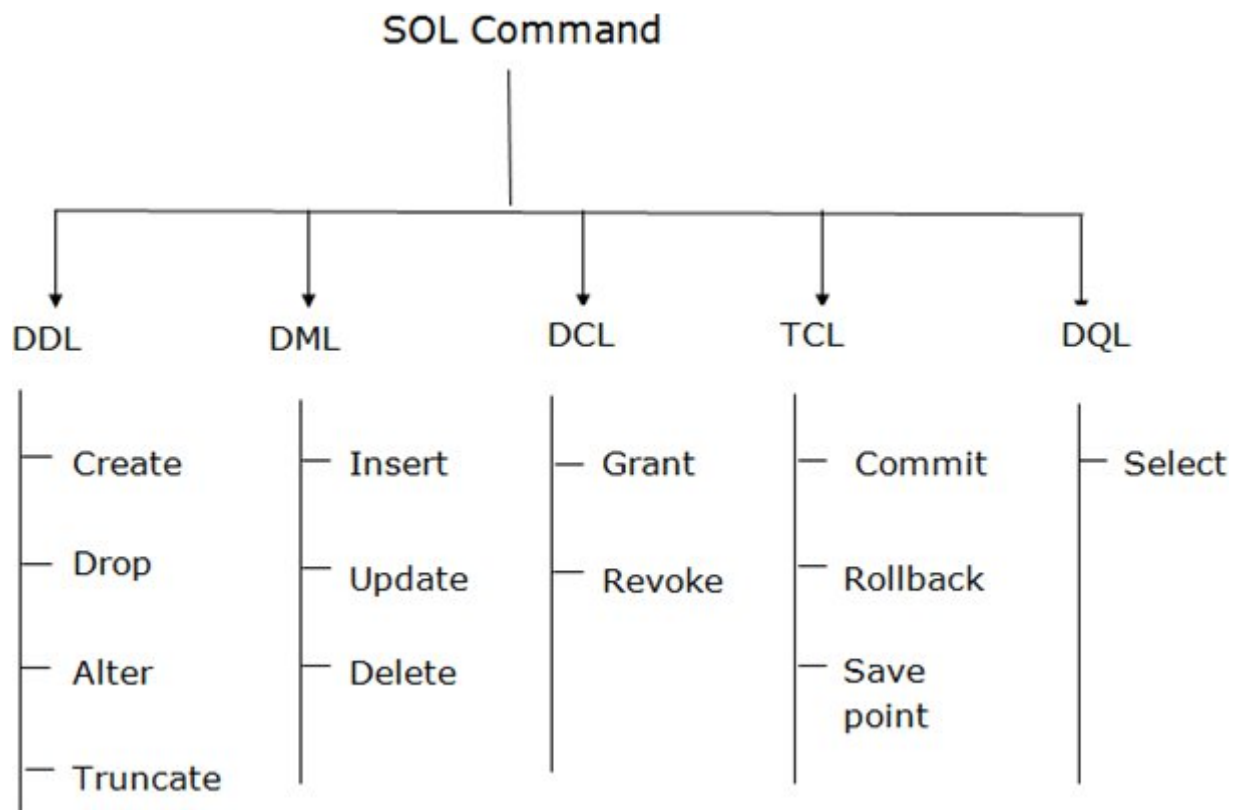
It is important to carefully consider the appropriate method for enforcing integrity constraints in a DBMS in order to ensure the accuracy and consistency of the data.

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:

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

Example:

1. 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

1. DROP TABLE table_name;

Example

1. 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

1. ALTER TABLE table_name ADD column_name COLUMN-definition;

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

EXAMPLE

1. ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));

2. 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:

1. TRUNCATE TABLE table_name;

Example:

1. 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:

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

Or

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

For example:

1. 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:

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

For example:

1. UPDATE students

2. SET User_Name = 'Sonoo'
3. WHERE Student_Id = '3'

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

Syntax:

1. DELETE FROM table_name [WHERE condition];

For example:

1. DELETE FROM javatpoint
2. 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

1. GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

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

Example

1. 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:

1. COMMIT;

Example:

1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. COMMIT;

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

Syntax:

1. ROLLBACK;

Example:

1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. ROLLBACK;

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

Syntax:

1. 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:

1. SELECT expressions
2. FROM TABLES
3. WHERE conditions;

For example:

1. SELECT emp_name
2. FROM employee
3. WHERE age > 20;

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:

1. **CREATE VIEW** view_name **AS**
2. **SELECT** column1, column2.....
3. **FROM** table_name
4. **WHERE** condition;

2. Creating View from a single table

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

Query:

1. **CREATE VIEW** DetailsView **AS**
2. **SELECT NAME**, ADDRESS
3. **FROM** Student_Details
4. **WHERE** STU_ID < 4;

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

1. **SELECT * FROM** DetailsView;

Output:

NAME	ADDRESS
Stephan	Delhi
Kathrin	Noida
David	Ghaziabad

Deleting View

A view can be deleted using the Drop View statement.

Syntax

1. **DROP VIEW** view_name;

Example:

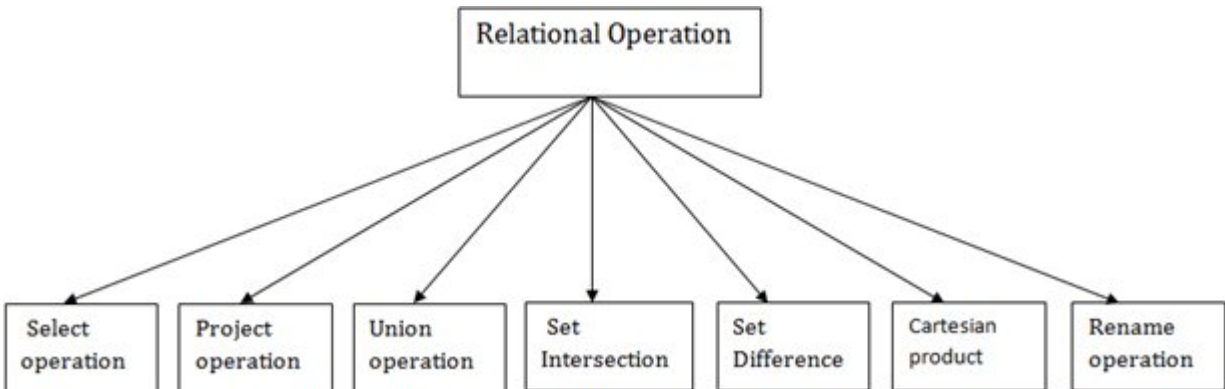
If we want to delete the View **MarksView**, we can do this as:

1. **DROP VIEW** MarksView;

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).

1. Notation: $\sigma p(r)$

Where:

σ is used for selection
 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 $=, \neq, \geq, <, >, \leq$.

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:

1. 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 π .

1. Notation: π 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:

1. Π 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 $\cup$.

1. Notation: $R \cup S$

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:

- Π CUSTOMER_NAME (BORROW) $\cup$ Π 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 $\cap$.

1. Notation: $R \cap S$

Example: Using the above DEPOSITOR table and BORROW table

Input:

1. $\Pi \text{ CUSTOMER_NAME (BORROW)} \cap \Pi \text{ CUSTOMER_NAME (DEPOSITOR)}$

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 (-).

1. Notation: $R - S$

Example: Using the above DEPOSITOR table and BORROW table

Input:

1. $\Pi \text{ CUSTOMER_NAME (BORROW) - } \Pi \text{ CUSTOMER_NAME (DEPOSITOR)}$

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.

1. Notation: $E \times D$

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).

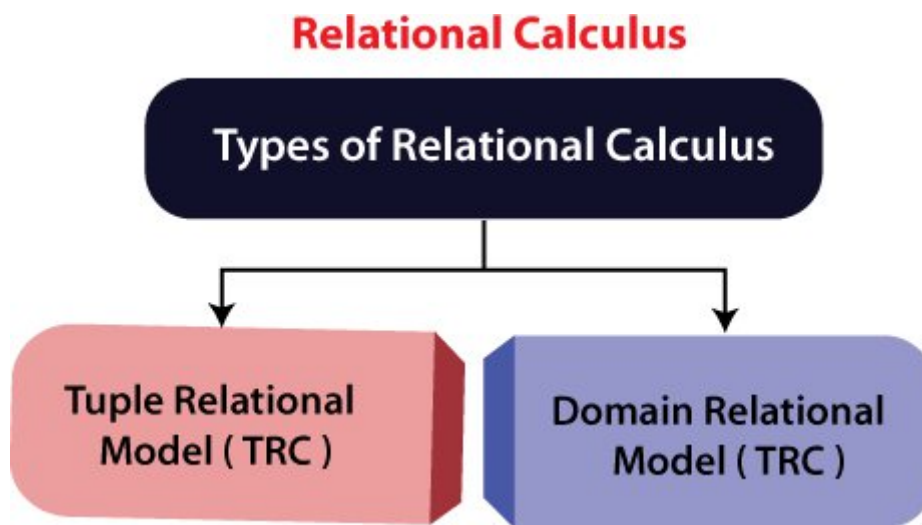
Example: We can use the rename operator to rename STUDENT relation to STUDENT1.

1. $\rho(\text{STUDENT1}, \text{STUDENT})$

Relational Calculus

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

Types of Relational calculus:



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

1. $\{T \mid P(T)\}$ or $\{T \mid \text{Condition}(T)\}$

Where

T is the resulting tuples

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

For example:

1. $\{T.name \mid \text{Author}(T) \text{ AND } T.article = 'database'\}$

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 ($\exists$) and Universal Quantifiers ($\forall$).

For example:

1. $\{R \mid \exists T \in \text{Authors}(T.article = 'database' \text{ AND } R.name = T.name)\}$

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 $\wedge$ (and), $\vee$ (or) and $\neg$ (not). It uses Existential ($\exists$) and Universal Quantifiers ($\forall$) to bind the variable. The QBE or Query by example is a query language related to domain relational calculus.

Notation:

1. $\{a_1, a_2, a_3, \dots, a_n \mid P(a_1, a_2, a_3, \dots, a_n)\}$

Where

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

For example:

1. $\{ \langle article, page, subject \rangle \mid \in \text{javatpoint} \wedge subject = 'database' \}$

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

UNIT 3

SQL UNION

UNION is an SQL operator which combines the result of two or more SELECT queries and provides the single set in the output.

Syntax of UNION in SQL:

1. **SELECT** Column_Name_1, Column_Name_2....., Column_NameN **FROM** Table_Name_1
2. **UNION**
3. **SELECT** Column_Name1, Column_Name_2, Column_Name_N **FROM** Table_Name_2
4. **UNION.....UNION**
5. **SELECT** Column_Name1, Column_Name_2....., Column_Name_N **FROM** Table_Name_N ;

The data type and the number of fields must be same for every SELECT statement connected with the UNION operator. The database system uses the UNION operator for removing the duplicate values from the combined result set.

Example of UNION operator in SQL

1. **SELECT * FROM** Old_Employee;

Employee_Id	Employee_Name	Emp_Age	Emp_Salary
101	Akhil	28	25000
102	Abhay	27	26000
103	Sorya	26	29000
104	Abhishek	27	26000
105	Ritik	26	29000

106	Yash	29	28000
-----	------	----	-------

1. **SELECT * FROM** New_Employee;

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Jack	28	45000
202	Berry	29	35000
105	Ritik	26	29000
203	Shyam	27	26000
204	Ritika	28	38000
106	Yash	29	28000

1. **SELECT * FROM** Old_Employee**UNION SELECT * FROM** New_Employee;

Output:

Employee_Id	Employee_Name	Emp_Age	Emp_Salary
101	Akhil	28	25000
102	Abhay	27	26000
103	Sorya	26	29000
104	Abhishek	27	26000
105	Ritik	26	29000
106	Yash	29	28000
201	Jack	28	45000
202	Berry	29	35000
203	Shyam	27	26000
204	Ritika	28	38000

SQL INTERSECT

The INTERSECT is an operator in Structured Query Language that combines the rows of two SELECT statements and returns only those rows from the first SELECT statement, which are the same as the rows of the second SELECT statement.

In simple words, we can say that this operator shows common rows from both the SELECT statement.

This operator does not work in the MySQL database.

1. **SELECT * FROM** Old_Faculty_Info;

Faculty_Id	Faculty_First_Name	Faculty_Last_Name	Faculty_Dept_Id	Faculty_Joini
1001	Arush	Sharma	4001	2020-01-02
1002	Bulbul	Roy	4002	2019-12-31
1004	Saurabh	Roy	4001	2020-10-10
1005	Shivani	Singhania	4001	2019-07-15
1006	Avinash	Sharma	4002	2019-11-11
1007	Shyam	Besas	4003	2021-06-21

1. **SELECT * FROM** New_Faculty_Info;

Faculty_Id	Faculty_First_Name	Faculty_Last_Name	Faculty_Dept_Id	Faculty_Joini
1010	Ankush	Roy	4004	2018-10-02
1001	Arush	Sharma	4001	2020-01-02
1009	Raj	Singhania	4005	2021-05-10
1005	Shivani	Singhania	4001	2019-07-15
1008	Avinabh	Chetya	4002	2018-11-11
1007	Shyam	Besas	4003	2021-06-21

1. **SELECT * FROM** Old_Faculty_Info**INTERSECT SELECT * FROM** New_Faculty_Info;

Output:

Faculty_Id	Faculty_First_Name	Faculty_Last_Name	Faculty_Dept_Id	Faculty_Joini
1001	Arush	Sharma	4001	2020-01-02
1005	Shivani	Singhania	4001	2019-07-15
1007	Shyam	Besas	4003	2021-06-21

SQL | Except Clause

In SQL, EXCEPT returns those tuples that are returned by the first SELECT operation, and not returned by the second SELECT operation.

This is the same as using a subtract operator in relational algebra.

Example:

Say we have two relations, Students and TA (Teaching Assistant). We want to return all those students who are not teaching assistants. The query can be formulated as:

Students Table:

StudentID	Name	Course
1	Rohan	DBMS
2	Kevin	OS
3	Mansi	DBMS
4	Mansi	ADA
5	Rekha	ADA
6	Megha	OS

TA Table:

StudentID	Name	Course
1	Kevin	TOC

2	Sita	IP
3	Manik	AP
4	Rekha	SNS

```
SELECT Name
    FROM Students
EXCEPT
SELECT NAME
    FROM TA;
```

Output:

Rohan

Mansi

Megha

SQL Sub Query

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

Syntax

1. SELECT column_name
2. FROM table_name
3. WHERE column_name expression operator
4. (SELECT column_name from table_name WHERE ...);
 5. **Example**
 6. 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:

1. SELECT *
2. FROM EMPLOYEE
3. WHERE ID IN (SELECT ID
4. FROM EMPLOYEE
5. WHERE SALARY > 4500);

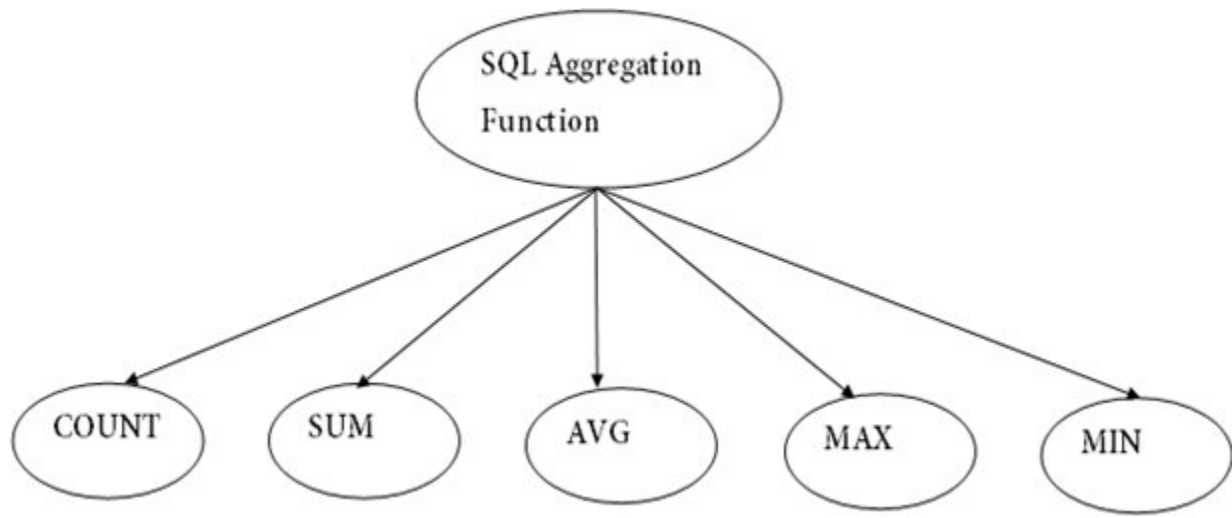
This would produce the following result:

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

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

1. COUNT(*)
2. or
3. 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()

1. SELECT COUNT(*)
2. FROM PRODUCT_MAST;

Output:

10

Example: COUNT with WHERE

1. SELECT COUNT(*)
2. FROM PRODUCT_MAST;
3. WHERE RATE >= 20;

Output:

7

Example: COUNT() with DISTINCT

1. SELECT COUNT(DISTINCT COMPANY)
2. FROM PRODUCT_MAST;

Output:

```
3
```

Example: COUNT() with GROUP BY

1. SELECT COMPANY, COUNT(*)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY;

Output:

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

Example: COUNT() with HAVING

1. SELECT COMPANY, COUNT(*)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY
4. 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

1. SUM()
2. or
3. SUM([ALL|DISTINCT] expression)

Example: SUM()

1. SELECT SUM(COST)
2. FROM PRODUCT_MAST;

Output:

```
670
```

Example: SUM() with WHERE

1. SELECT SUM(COST)
2. FROM PRODUCT_MAST
3. WHERE QTY>3;

Output:

```
320
```

Example: SUM() with GROUP BY

1. SELECT SUM(COST)
2. FROM PRODUCT_MAST
3. WHERE QTY>3
4. GROUP BY COMPANY;

Output:

```
Com1  150
Com2  170
```

Example: SUM() with HAVING

1. SELECT COMPANY, SUM(COST)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY
4. HAVING SUM(COST)>=170;

5. Output:

```
6. Com1  335
7. Com3  170
```

8. 3. AVG function

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

10. Syntax

1. AVG()
2. or

3. AVG([ALL|DISTINCT] expression)

Example:

1. SELECT AVG(COST)
2. 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

1. MAX()
2. or
3. MAX([ALL|DISTINCT] expression)

Example:

1. SELECT MAX(RATE)
2. 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

1. MIN()
2. or
3. MIN([ALL|DISTINCT] expression)

Example:

1. SELECT MIN(RATE)
2. FROM PRODUCT_MAST;

Output:

10

SQL SELECT NULL

First of all we should know that what null value is? Null values are used to represent missing unknown data.

There can be two conditions:

1. Where SQL is NULL
2. Where SQL is NOT NULL

If in a table, a column is optional, it is very easy to insert data in column or update an existing record without adding a value in this column. This means that field has null value.

Where SQL is NULL:

How to select records with null values only? (in the marks column)

There is an example of student table:

SIR_NAME	NAME	MARKS
TYAGI	SEEMA	
SINGH	RAMAN	5.5
SHARMA	AMAR	
JAISWAL	VICKY	6.2

Let's see the query to get all the records where marks is NULL:

1. **SELECT** SIR_NAME, **NAME**, MARKS **FROM** STUDENTS

2. **WHERE** MARKS **IS** NULL

It will return the following records:

SIR_NAME	NAME	MARKS
SHARMA	AMAR	
TYAGI	SEEMA	

Where SQL is NOT NULL:

How to select records with no null values(in marks column)? Let's see the query to get all the records where marks is NOT NULL

1. **SELECT** SIR_NAME, FIRSTNAME, MARKS **FROM** STUDENTS
2. **WHERE** MARKS **IS** NOT NULL

SIR_NAME	NAME	MARKS
SINGH	RAMAN	5.5
JAISWAL	VICKY	6.2

Trigger in SQL

In this article, you will learn about the trigger and its implementation with examples.

A **Trigger** in Structured Query Language is a set of procedural statements which are executed automatically when there is any response to certain events on the particular table in the database. Triggers are used to protect the data integrity in the database.

In SQL, this concept is the same as the trigger in real life. For example, when we pull the gun trigger, the bullet is fired.

To understand the concept of trigger in SQL, let's take the below hypothetical situation:

Suppose Rishabh is the human resource manager in a multinational company. When the record of a new employee is entered into the database, he has to send the 'Congrats' message to each

new employee. If there are four or five employees, Rishabh can do it manually, but if the number of new Employees is more than the thousand, then in such condition, he has to use the trigger in the database.

Thus, now Rishabh has to create the trigger in the table, which will automatically send a 'Congrats' message to the new employees once their record is inserted into the database.

The trigger is always executed with the specific table in the database. If we remove the table, all the triggers associated with that table are also deleted automatically.

In Structured Query Language, triggers are called only either before or after the below events:

1. **INSERT Event:** This event is called when the new row is entered in the table.
2. **UPDATE Event:** This event is called when the existing record is changed or modified in the table.
3. **DELETE Event:** This event is called when the existing record is removed from the table.

Types of Triggers in SQL

Following are the six types of triggers in SQL:

1. **AFTER INSERT Trigger**
This trigger is invoked after the insertion of data in the table.
2. **AFTER UPDATE Trigger**
This trigger is invoked in SQL after the modification of the data in the table.
3. **AFTER DELETE Trigger**
This trigger is invoked after deleting the data from the table.
4. **BEFORE INSERT Trigger**
This trigger is invoked before the inserting the record in the table.
5. **BEFORE UPDATE Trigger**
This trigger is invoked before the updating the record in the table.
6. **BEFORE DELETE Trigger**
This trigger is invoked before deleting the record from the table.

Syntax of Trigger in SQL

1. **CREATE TRIGGER** Trigger_Name

2. [BEFORE | **AFTER**] [**Insert** | **Update** | **Delete**]
3. **ON** [Table_Name]
4. [**FOR EACH ROW** | **FOR EACH COLUMN**]
5. **AS**
6. **Set of** SQL Statement

In the trigger syntax, firstly, we have to define the name of the trigger after the CREATE TRIGGER keyword. After that, we have to define the BEFORE or AFTER keyword with anyone event.

Then, we define the name of that table on which trigger is to occur.

After the table name, we have to define the row-level or statement-level trigger.

And, at last, we have to write the SQL statements which perform actions on the occurring of event.

Example of Trigger in SQL

To understand the concept of trigger in SQL, first, we have to create the table on which trigger is to be executed.

The following query creates the **Student_Trigger** table in the SQL database:

1. **CREATE TABLE** Student_Trigger
2. (
3. Student_RollNo **INT** NOT NULL **PRIMARY KEY**,
4. Student_FirstName **Varchar** (100),
5. Student_EnglishMarks **INT**,
6. Student_PhysicsMarks **INT**,
7. Student_ChemistryMarks **INT**,
8. Student_MathsMarks **INT**,
9. Student_TotalMarks **INT**,
10. Student_Percentage);

The following query shows the structure of the **Student_Trigger** table:

1. **DESC** Student_Trigger;

Output:

Field	Type	NULL	Key	Default	Ex
Student_RollNo	INT	NO	PRI	NULL	
Student_FirstName	Varchar(100)	YES		NULL	
Student_EnglishMarks	INT	YES		NULL	
Student_PhysicsMarks	INT	YES		NULL	
Student_ChemistryMarks	INT	YES		NULL	
Student_MathsMarks	INT	YES		NULL	
Student_TotalMarks	INT	YES		NULL	
Student_Percentage	INT	YES		NULL	

The following query fires a trigger before the insertion of the student record in the table:

1. **CREATE TRIGGER** Student_Table_Marks
2. **BEFORE INSERT**
3. **ON**
4. Student_Trigger
5. **FOR EACH ROW**
6. **SET** new.Student_TotalMarks = new.Student_EnglishMarks + new.Student_PhysicsMarks + new.Student_ChemistryMarks + new.Student_MathsMarks,
7. new.Student_Percentage = (new.Student_TotalMarks / 400) * 100;

The following query inserts the record into Student_Trigger table:

1. **INSERT INTO** Student_Trigger (Student_RollNo, Student_FirstName, Student_EnglishMarks, Student_PhysicsMarks, Student_ChemistryMarks, Student_MathsMarks, Student_TotalMarks, Student_Percentage) **VALUES** (201, Sorya, 88, 75, 69, 92, 0, 0);

To check the output of the above INSERT statement, you have to type the following SELECT statement:

1. **SELECT * FROM** Student_Trigger;

Output:

Student_RollNo	Student_FirstName	Student_EnglishMarks	Student_PhysicsMarks	
201	Sorya	88	75	6

Advantages of Triggers in SQL

Following are the three main advantages of triggers in Structured Query Language:

1. SQL provides an alternate way for maintaining the data and referential integrity in the tables.
2. Triggers helps in executing the scheduled tasks because they are called automatically.
3. They catch the errors in the database layer of various businesses.
4. They allow the database users to validate values before inserting and updating.

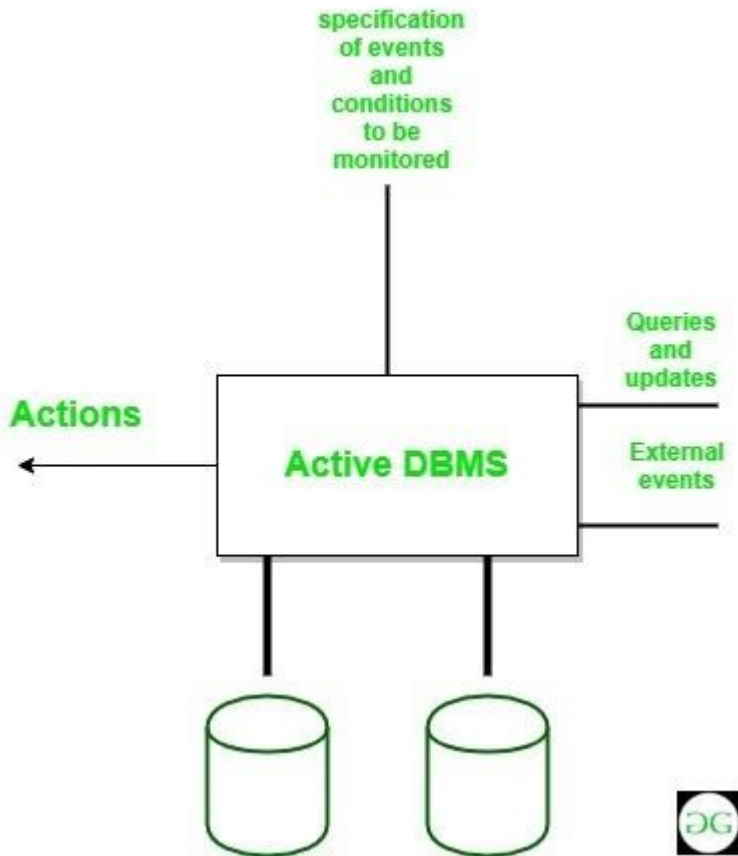
Disadvantages of Triggers in SQL

Following are the main disadvantages of triggers in Structured Query Language:

1. They are not compiled.
2. It is not possible to find and debug the errors in triggers.
3. If we use the complex code in the trigger, it makes the application run slower.
4. Trigger increases the high load on the database system.

Active Databases

An active Database is a database consisting of a set of triggers. These databases are very difficult to be maintained because of the complexity that arises in understanding the effect of these triggers. In such database, DBMS initially verifies whether the particular trigger specified in the statement that modifies the database is activated or not, prior to executing the statement. If the trigger is active then DBMS executes the condition part and then executes the action part only if the specified condition is evaluated to true. It is possible to activate more than one trigger within a single statement. In such situation, DBMS processes each of the trigger randomly. The execution of an action part of a trigger may either activate other triggers or the same trigger that Initialized this action. Such types of trigger that activates itself is called as 'recursive trigger'. The DBMS executes such chains of trigger in some pre-defined manner but it effects the concept of understanding.

**Features of Active Database:**

1. It possess all the concepts of a conventional database i.e. data modelling facilities, query language etc.
2. It supports all the functions of a traditional database like data definition, data manipulation, storage management etc.
3. It supports definition and management of ECA rules.
4. It detects event occurrence.
5. It must be able to evaluate conditions and to execute actions.
6. It means that it has to implement rule execution.

Advantages :

1. Enhances traditional database functionalities with powerful rule processing capabilities.
2. Enable a uniform and centralized description of the business rules relevant to the information system.
3. Avoids redundancy of checking and repair operations.
4. Suitable platform for building large and efficient knowledge base and expert systems.

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 satisfy desirable properties. Normalization is a process of decomposing the relations into relations with fewer attributes.

What is Normalization?

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

Why do we need Normalization?

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

Data modification anomalies can be categorized into three types:

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

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	R	R₁₁ R₁₂	R₂₁ R₂₂ R₂₃	R₃₁ R₃₂ R₃₃ R₃₄	R₄₁ R₄₂ R₄₃ R₄₄ R₄₅
Conditions	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 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 $X \rightarrow Y$.

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 $X \rightarrow Y$, 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. $EMP_ID \rightarrow EMP_COUNTRY$

2. $EMP_DEPT \rightarrow \{DEPT_TYPE, EMP_DEPT_NO\}$

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. $EMP_ID \rightarrow EMP_COUNTRY$
2. $EMP_DEPT \rightarrow \{DEPT_TYPE, EMP_DEPT_NO\}$

Candidate keys:

For the first table: EMP_ID
 For the second table: 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 $A \twoheadrightarrow B$, 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
-----------	---------

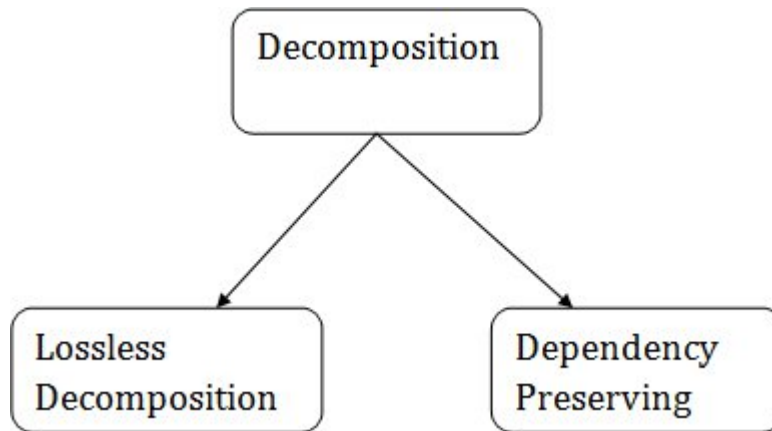
P3

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

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 $\twoheadrightarrow$ MANUF_YEAR
2. BIKE_MODEL $\twoheadrightarrow$ COLOR

This can be read as "BIKE_MODEL multidetermined MANUF_YEAR" and "BIKE_MODEL multidetermined C

UNIT-4".

Transaction

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

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

1. Open_Account(X)

2. $\text{Old_Balance} = \text{X.balance}$
3. $\text{New_Balance} = \text{Old_Balance} - 800$
4. $\text{X.balance} = \text{New_Balance}$
5. $\text{Close_Account}(\text{X})$

Y's Account

1. $\text{Open_Account}(\text{Y})$
2. $\text{Old_Balance} = \text{Y.balance}$
3. $\text{New_Balance} = \text{Old_Balance} + 800$
4. $\text{Y.balance} = \text{New_Balance}$
5. $\text{Close_Account}(\text{Y})$

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. $\text{R}(\text{X});$
2. $\text{X} = \text{X} - 500;$
3. $\text{W}(\text{X});$

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 database from 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-100	Read(B)	Y+
A:=		Y:=	
Write(A)		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.

1. Total before T occurs = $600+300=900$
2. 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

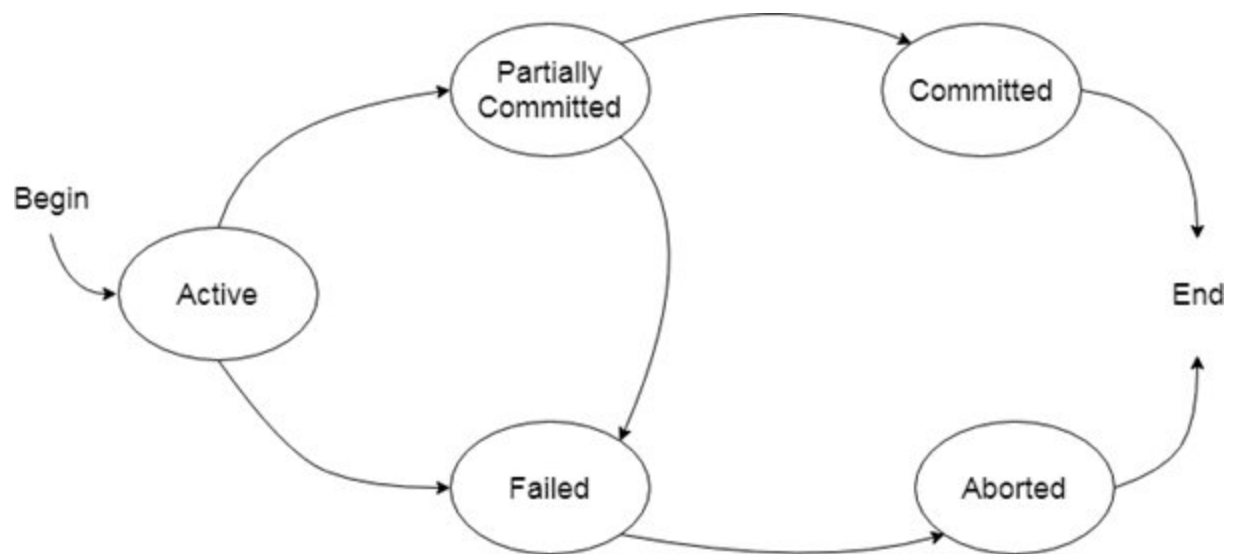
- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforces 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 -



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.

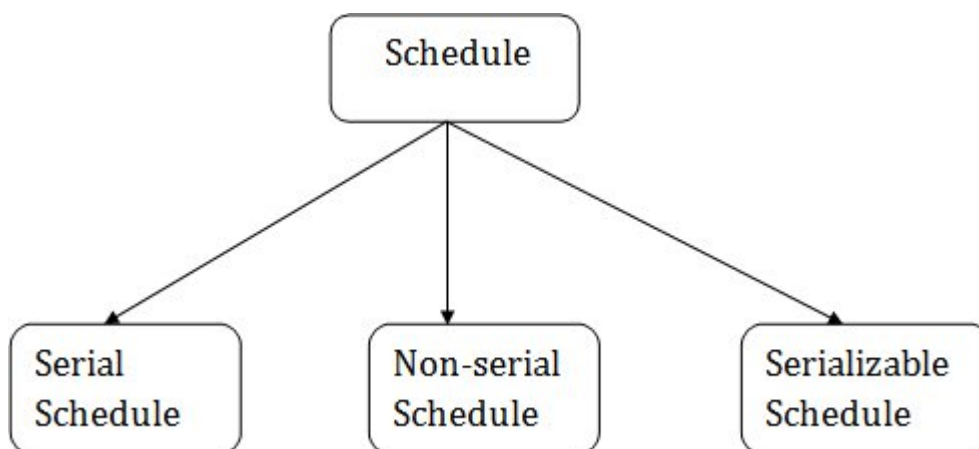
- In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.

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 T2 which was followed by all the operations of T1.
- 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.

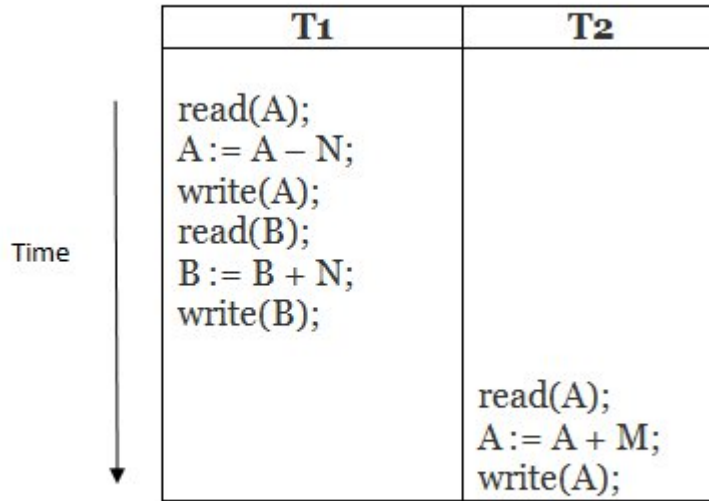
2. Non-serial Schedule

- 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

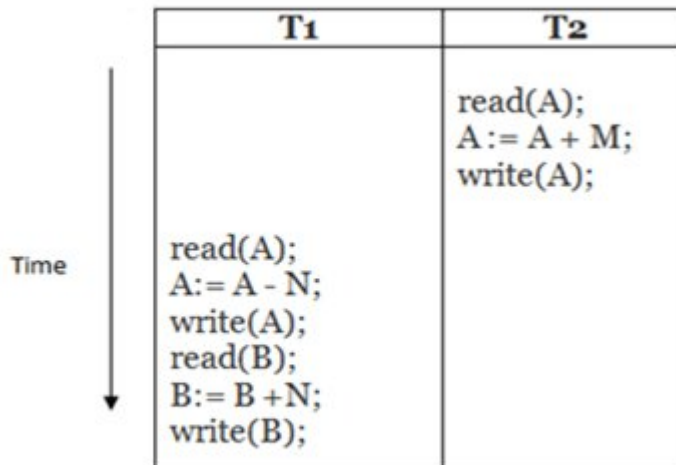
- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
- It identifies which schedules are correct when executions of the transaction have interleaving of their operations.
- A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.

(a)



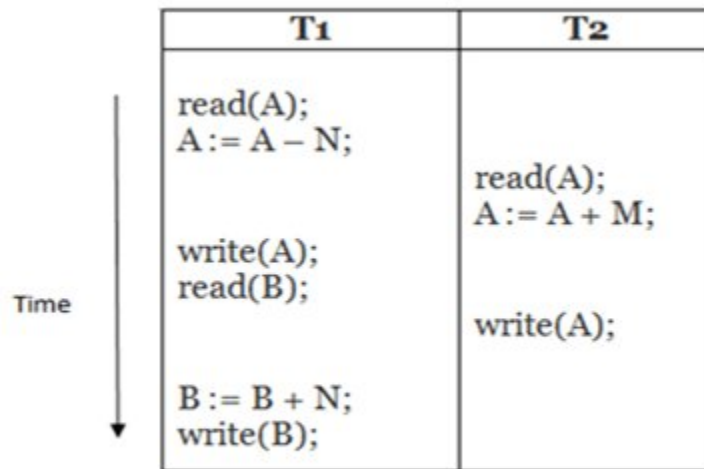
Schedule A

(b)



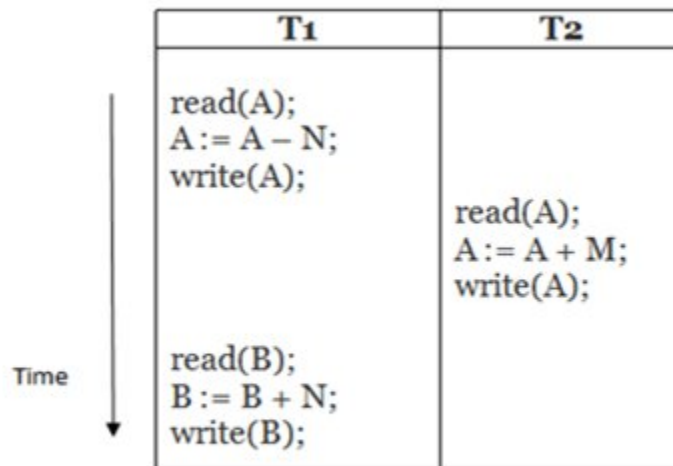
Schedule B

(c)



Schedule C

(d)



Schedule D

Here,

Schedule A and Schedule B are serial schedule.

Schedule C and Schedule D are Non-serial schedule.

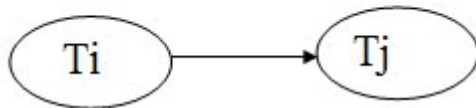
Testing of Serializability

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 $T_i \rightarrow T_j$ for which one of the three conditions holds:

1. Create a node $T_i \rightarrow T_j$ if T_i executes write (Q) before T_j executes read (Q).
2. Create a node $T_i \rightarrow T_j$ if T_i executes read (Q) before T_j executes write (Q).
3. Create a node $T_i \rightarrow T_j$ if T_i executes write (Q) before T_j executes write (Q).

Precedence graph for Schedule S



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

For example:

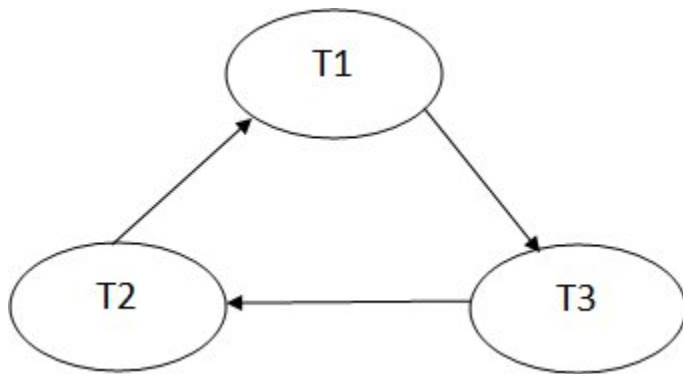
	T1	T2	T3
Time ↓	Read(A)	Read(B)	
	A := f ₁ (A)		Read(C)
		B := f ₂ (B) Write(B)	C := f ₃ (C) Write(C)
	Write(A)		Read(B)
		Read(A) A := f ₄ (A)	
	Read(C) C := f ₅ (C) Write(C)	Write(A)	B := f ₆ (B) Write(B)

Schedule S1

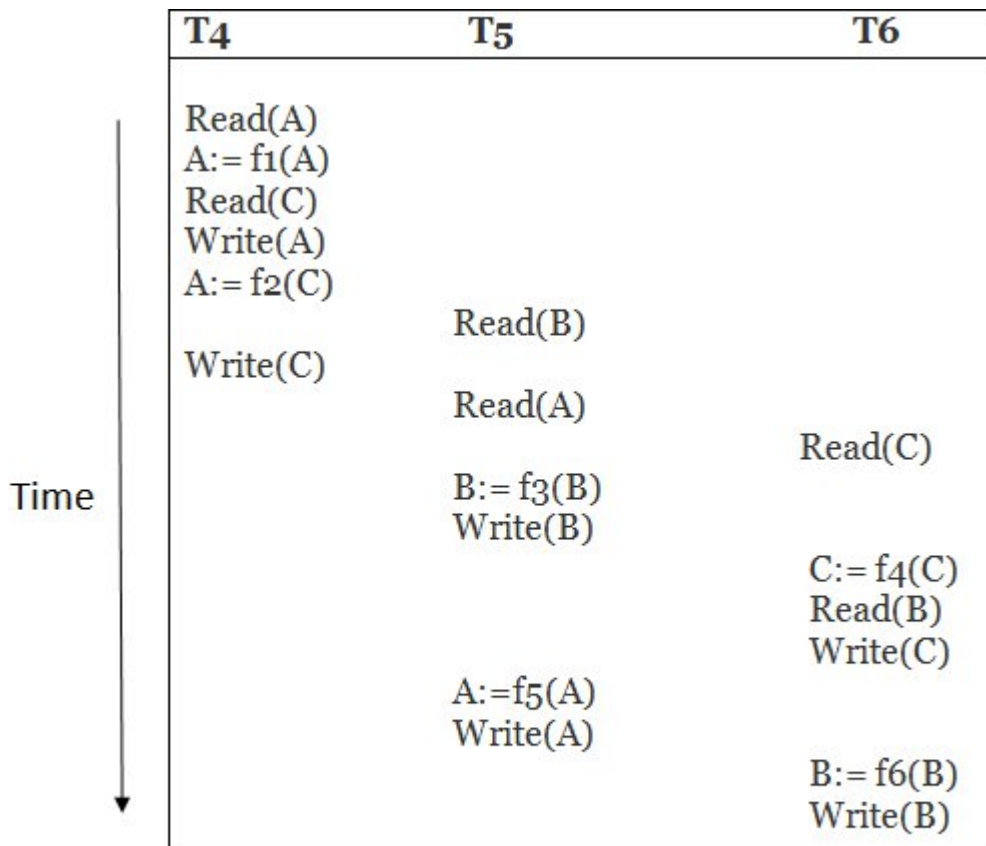
Explanation:

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.

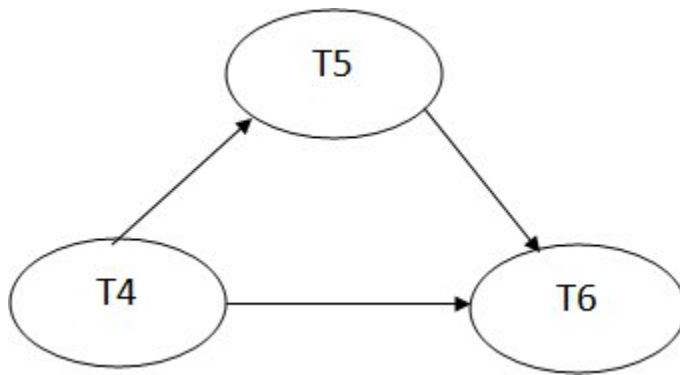


Schedule S2

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 Schedule S2 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

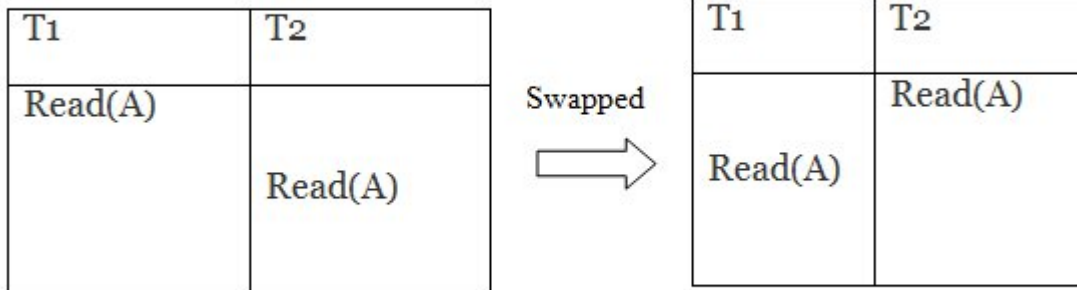
The two operations become conflicting if all conditions satisfy:

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)

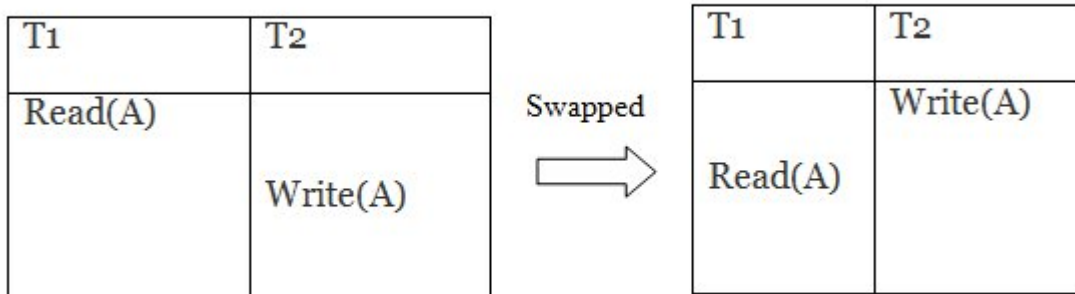


Schedule S1

Schedule S2

Here, $S1 = S2$. That means it is non-conflict.

2. T1: Read(A) T2: Write(A)



Schedule S1

Schedule S2

Here, $S1 \neq S2$. 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
Read(A)	Write(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 T_i is reading A which is updated by T_j then in S2 also, T_i should read A which is updated by T_j .

T1	T2	T3
Write(A)	Write(A)	Read(A)

Schedule S1

T1	T2	T3
Write(A)	Write(A)	<u>Read(A)</u>

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)

Schedule S1

T1	T2	T3
Write(A)	Read(A)	Write(A)

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

1. $= 3! = 6$
2. $S1 = \langle T1 \ T2 \ T3 \rangle$
3. $S2 = \langle T1 \ T3 \ T2 \rangle$
4. $S3 = \langle T2 \ T3 \ T1 \rangle$
5. $S4 = \langle T2 \ T1 \ T3 \rangle$
6. $S5 = \langle T3 \ T1 \ T2 \rangle$
7. $S6 = \langle T3 \ T2 \ T1 \rangle$

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.

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.

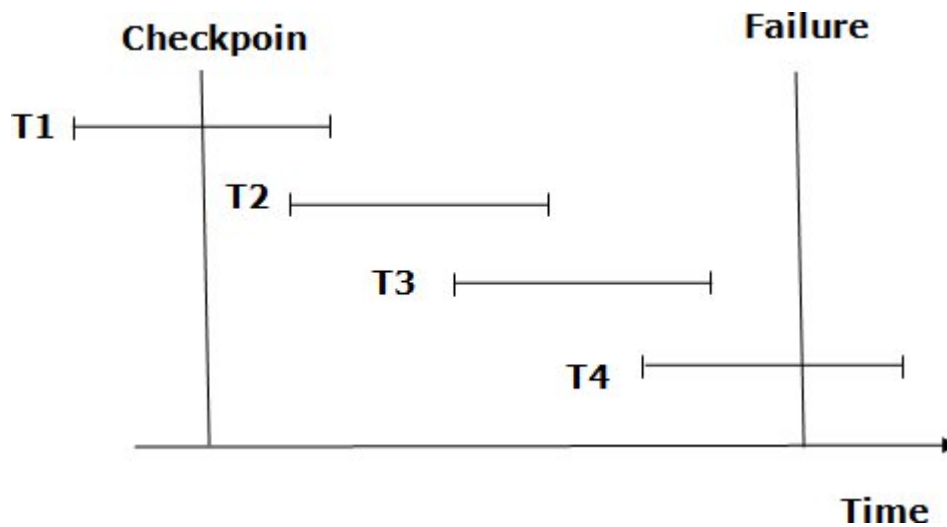
1. If the log contains the record $\langle T_i, \text{Start} \rangle$ and $\langle T_i, \text{Commit} \rangle$ or $\langle T_i, \text{Commit} \rangle$, then the Transaction T_i needs to be redone.
2. If log contains record $\langle T_n, \text{Start} \rangle$ but does not contain the record either $\langle T_i, \text{commit} \rangle$ or $\langle T_i, \text{abort} \rangle$, then the Transaction T_i 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 $\langle T_n, \text{Start} \rangle$ and $\langle T_n, \text{Commit} \rangle$ or just $\langle T_n, \text{Commit} \rangle$. In the redo-list and their previous list, all the transactions are removed and then redone before saving their logs.
- **For example:** In the log file, transaction T2 and T3 will have $\langle T_n, \text{Start} \rangle$ and $\langle T_n, \text{Commit} \rangle$. The T1 transaction will have only $\langle T_n, \text{commit} \rangle$ in the log file. That's why the transaction is committed after the checkpoint is crossed. Hence it puts T1, T2 and T3 transaction into redo list.
- The transaction is put into undo state if the recovery system sees a log with $\langle T_n, \text{Start} \rangle$ but no commit or abort log found. In the undo-list, all the transactions are undone, and their logs are removed.
- **For example:** Transaction T4 will have $\langle T_n, \text{Start} \rangle$. So T4 will be put into undo list since this transaction is not yet complete and failed 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.

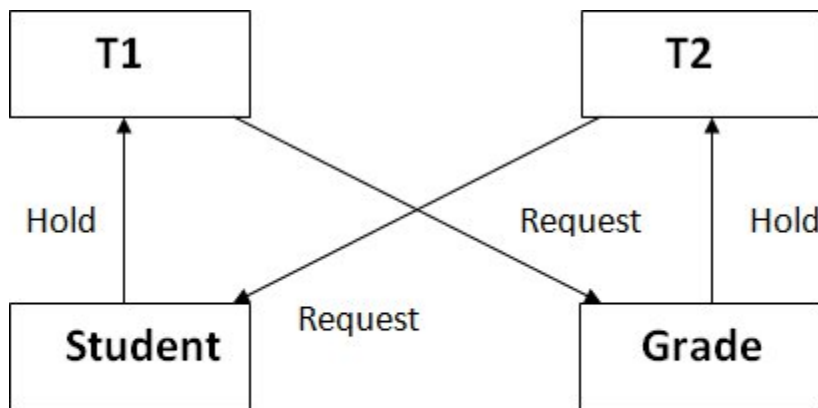


Figure: Deadlock in DBMS

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.

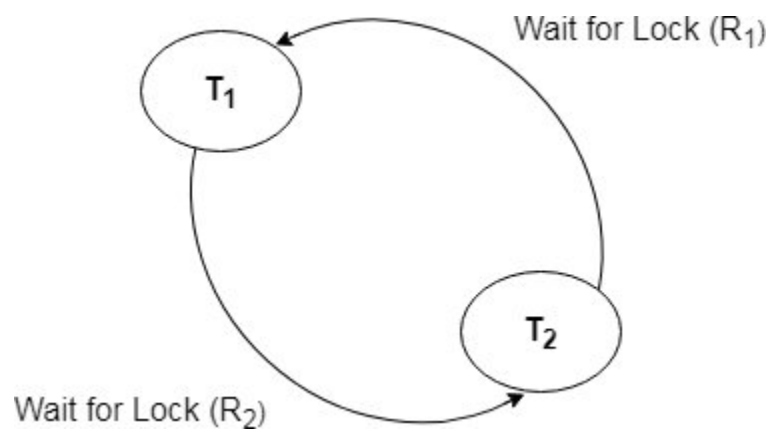
Deadlock Detection

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 T_i and T_j and let $TS(T)$ is a timestamp of any transaction T . If T_2 holds a lock by some other transaction and T_1 is requesting for resources held by T_2 then the following actions are performed by DBMS:

1. Check if $TS(T_i) < TS(T_j)$ - If T_i is the older transaction and T_j has held some resource, then T_i 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 $TS(T_i) < TS(T_j)$ - If T_i is older transaction and has held some resource and if T_j is waiting for it, then T_j 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 t_1 , 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 t_1 , transaction T_x reads the value of account A, i.e., \$300.
- At time t_2 , transaction T_x adds \$50 to account A that becomes \$350.
- At time t_3 , transaction T_x writes the updated value in account A, i.e., \$350.
- Then at time t_4 , transaction T_y reads account A that will be read as \$350.
- Then at time t_5 , transaction T_x rolls back 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* of 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.

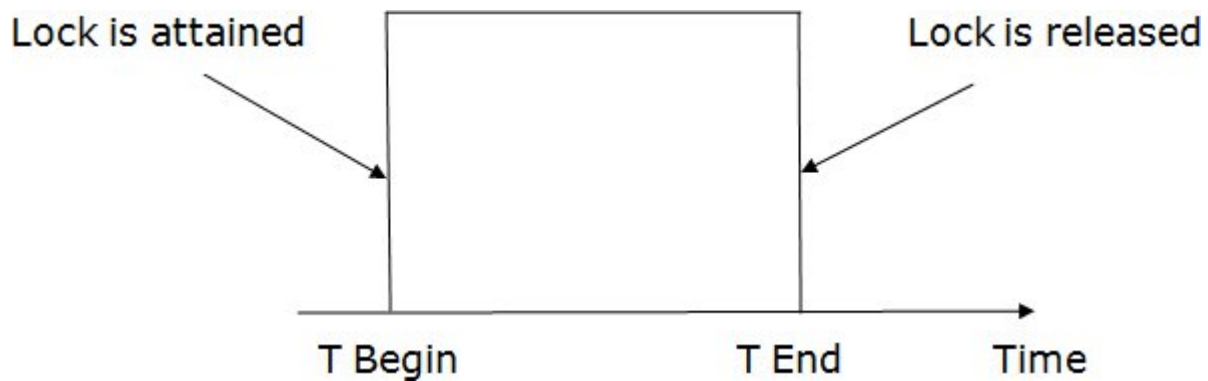
There are four types of lock protocols available:

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.

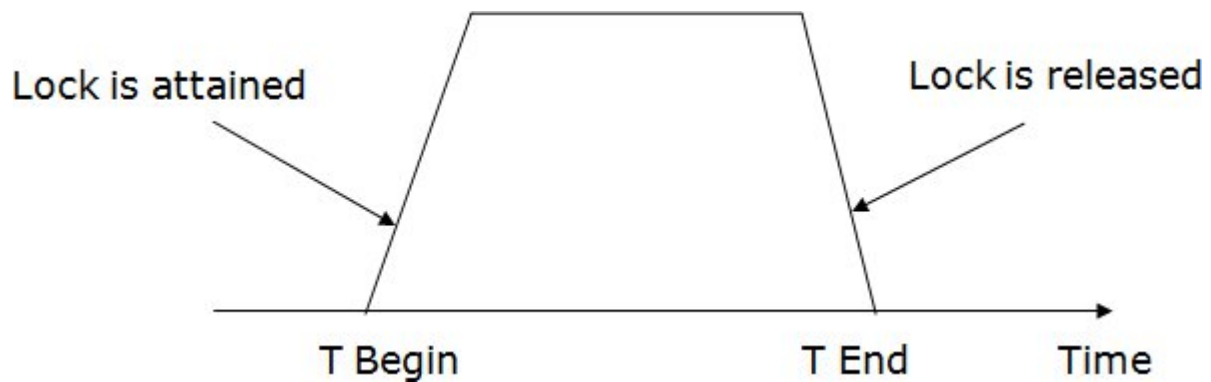
2. Pre-claiming Lock Protocol

- 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:

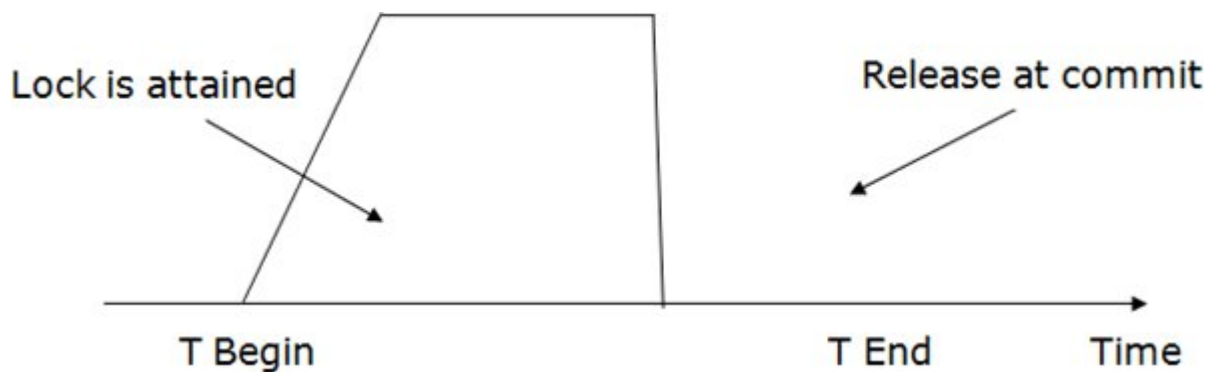
- **Growing phase:** from step 1-3
- **Shrinking phase:** from step 5-7
- **Lock point:** at 3

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 T_i issues a **Read (X)** operation:

- If $W_TS(X) > TS(T_i)$ then the operation is rejected.
- If $W_TS(X) \leq TS(T_i)$ then the operation is executed.
- Timestamps of all the data items are updated.

2. Check the following condition whenever a transaction T_i issues a **Write(X)** operation:

- If $TS(T_i) < R_TS(X)$ then the operation is rejected.
- If $TS(T_i) < W_TS(X)$ then the operation is rejected and T_i is rolled back otherwise the operation is executed.

Where,

$TS(T_i)$ denotes the timestamp of the transaction T_i .

$R_TS(X)$ 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(T_i): It contains the time when T_i started its execution.

Validation (T_i): It contains the time when T_i finishes its read phase and starts its validation phase.

Finish(T_i): It contains the time when T_i 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 $TS(T) = \text{validation}(T)$.
- 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.

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

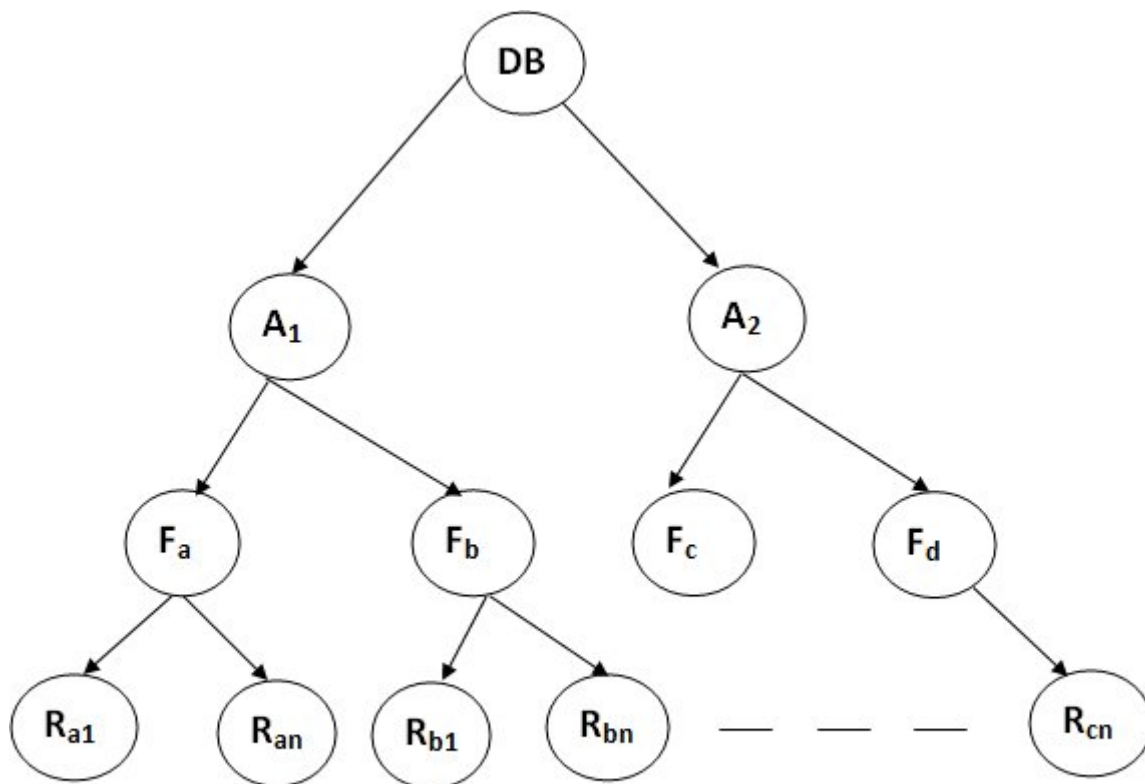


Figure: Multi Granularity tree Hierarchy

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.

As we have discussed [checkpoint](#) in Transaction Processing Concept of this tutorial, so you can go through the concepts again to make things more clear.

UNIT 5

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.

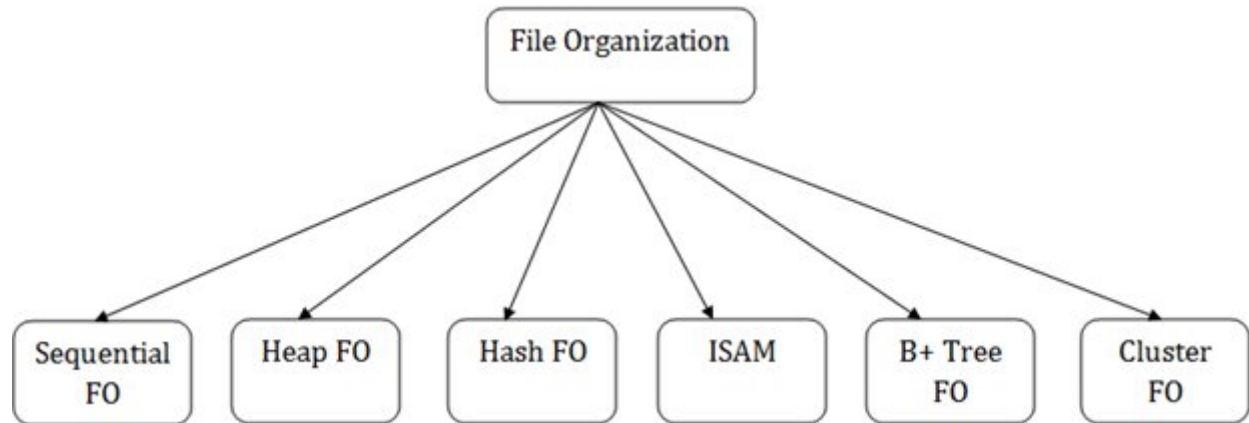
Objective of file organization

- 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.

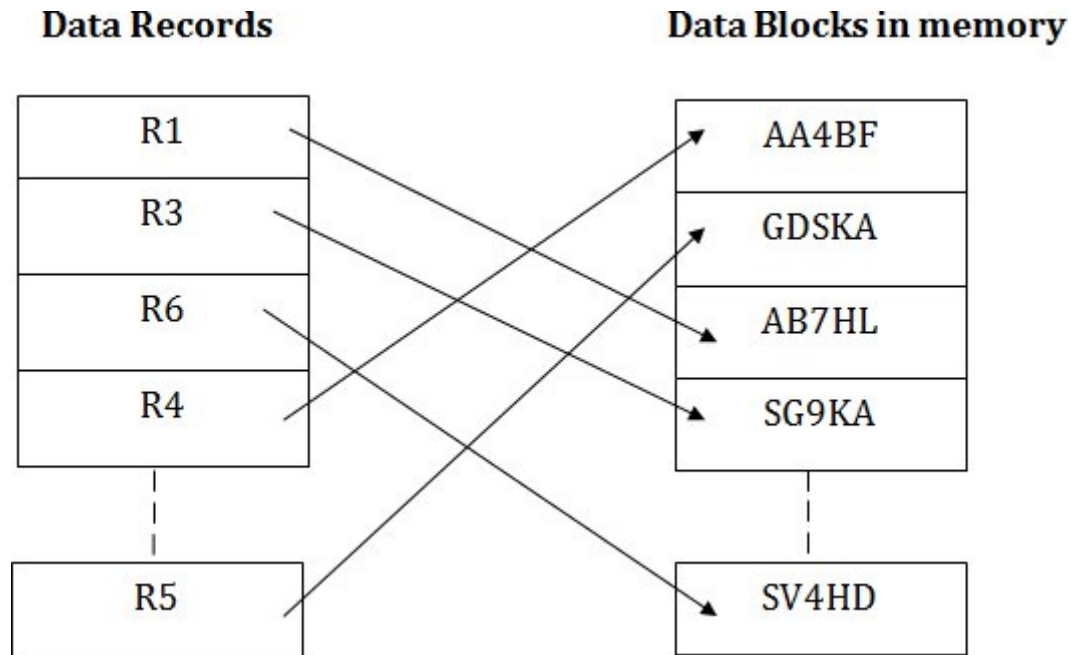
Types of file organization are as follows:



- Sequential file organization
- Heap file organization
- Hash file organization
- B+ file organization
- Indexed sequential access method (ISAM)
- Cluster file organization

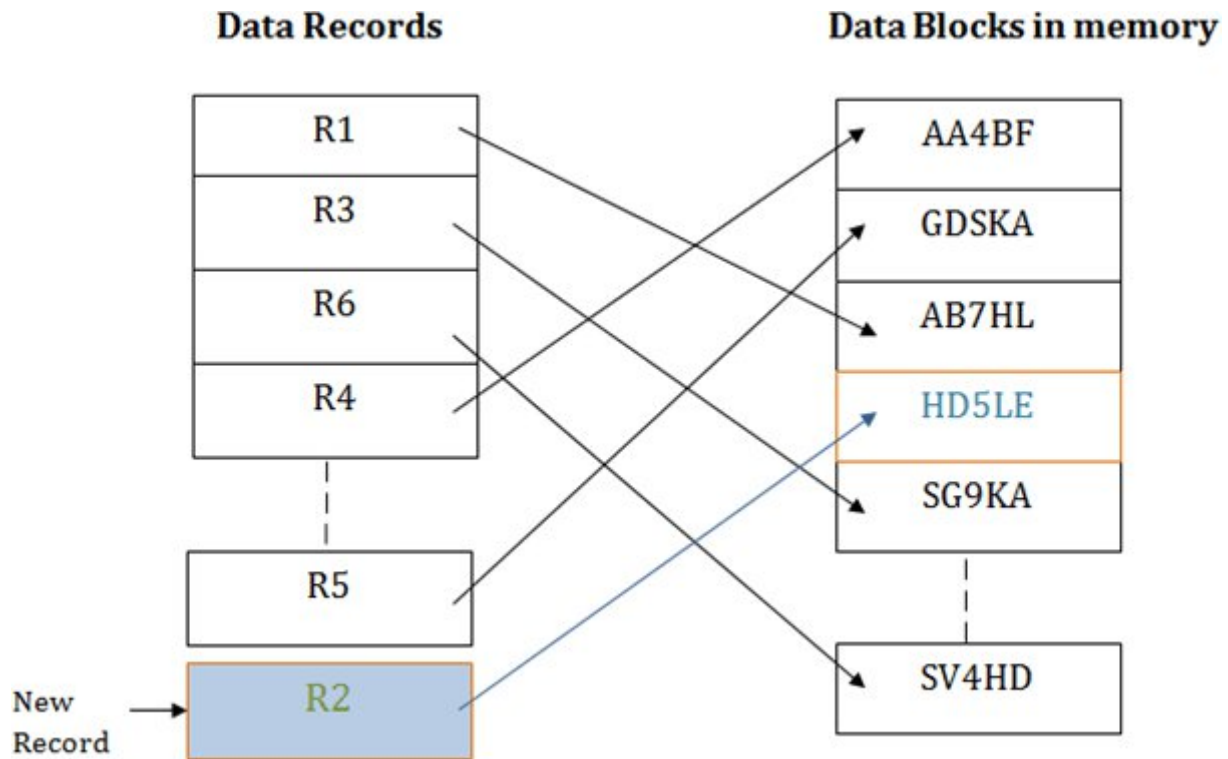
Hash File Organization

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.



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.

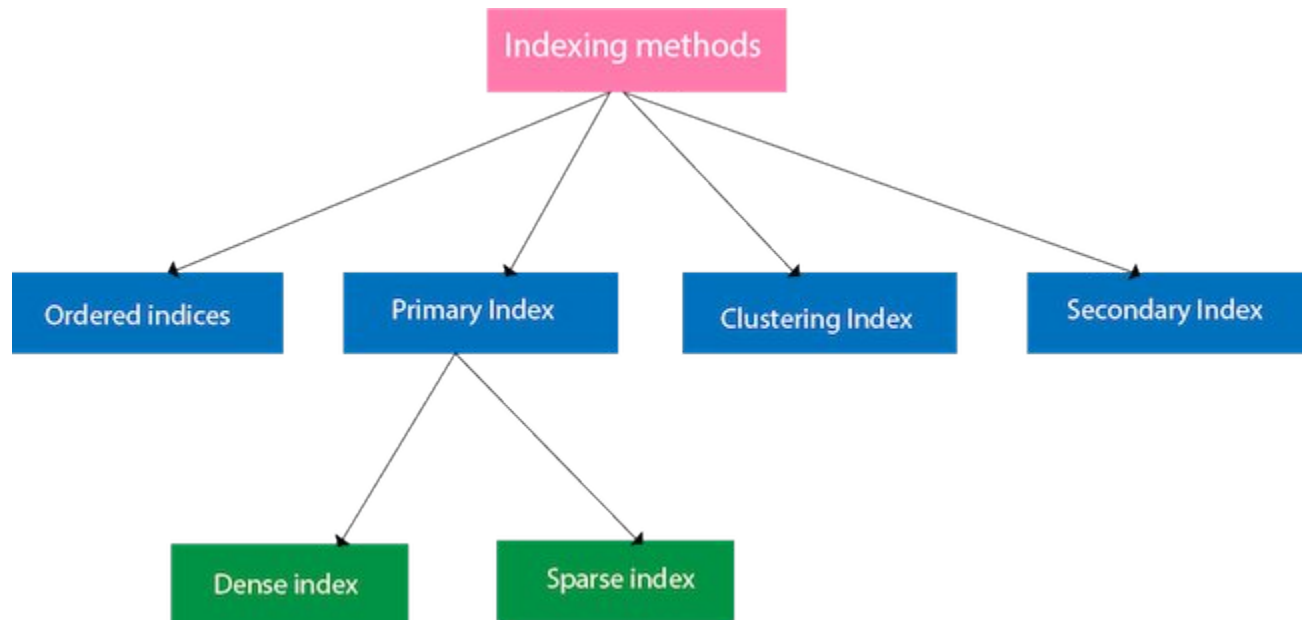
Search key	Data Reference
------------	----------------

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 $543 \times 10 = 5430$ bytes.
- In the case of an index, we will search using indexes and the DBMS will read the record after reading $542 \times 2 = 1084$ 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	•	→	UP	Agra	1,604,300
USA	•	→	USA	Chicago	2,789,378
Nepal	•	→	Nepal	Kathmandu	1,456,634
UK	•	→	UK	Cambridge	1,360,364

Sparse index

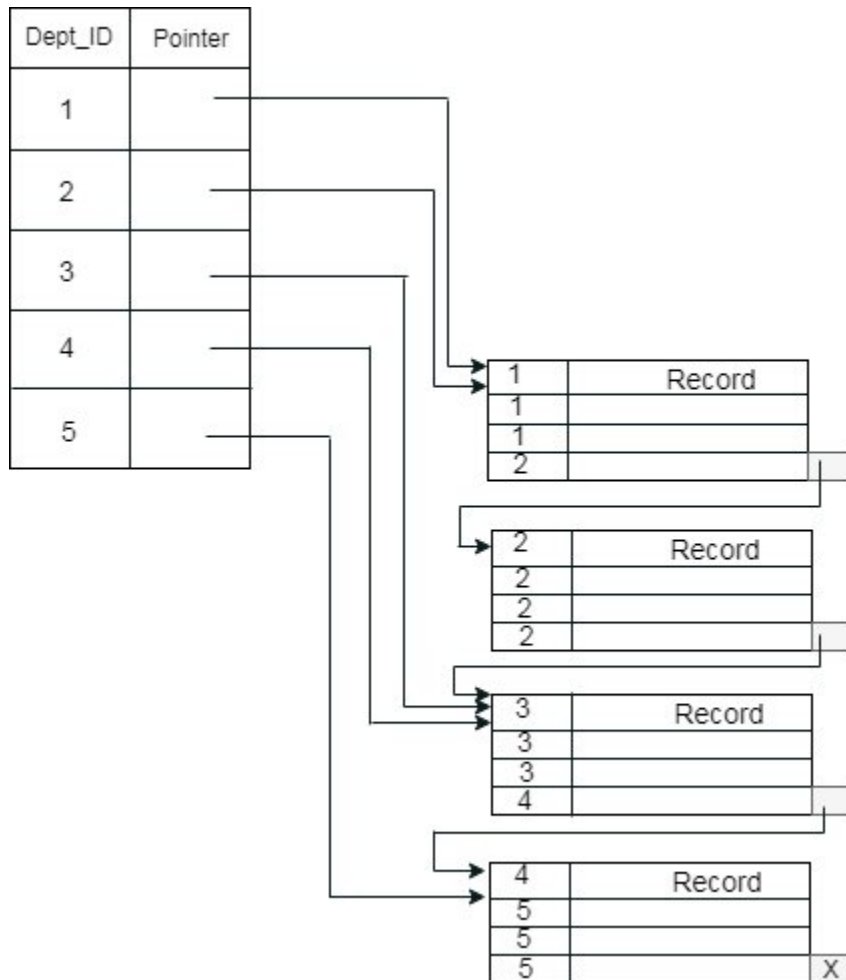
- In the data file, index record appears only for a few items. Each item points to a block.
- In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.

UP	•	→	UP	Agra	1,604,300
Nepal	•	→	USA	Chicago	2,789,378
UK	•	→	Nepal	Kathmandu	1,456,634
		→	UK	Cambridge	1,360,364

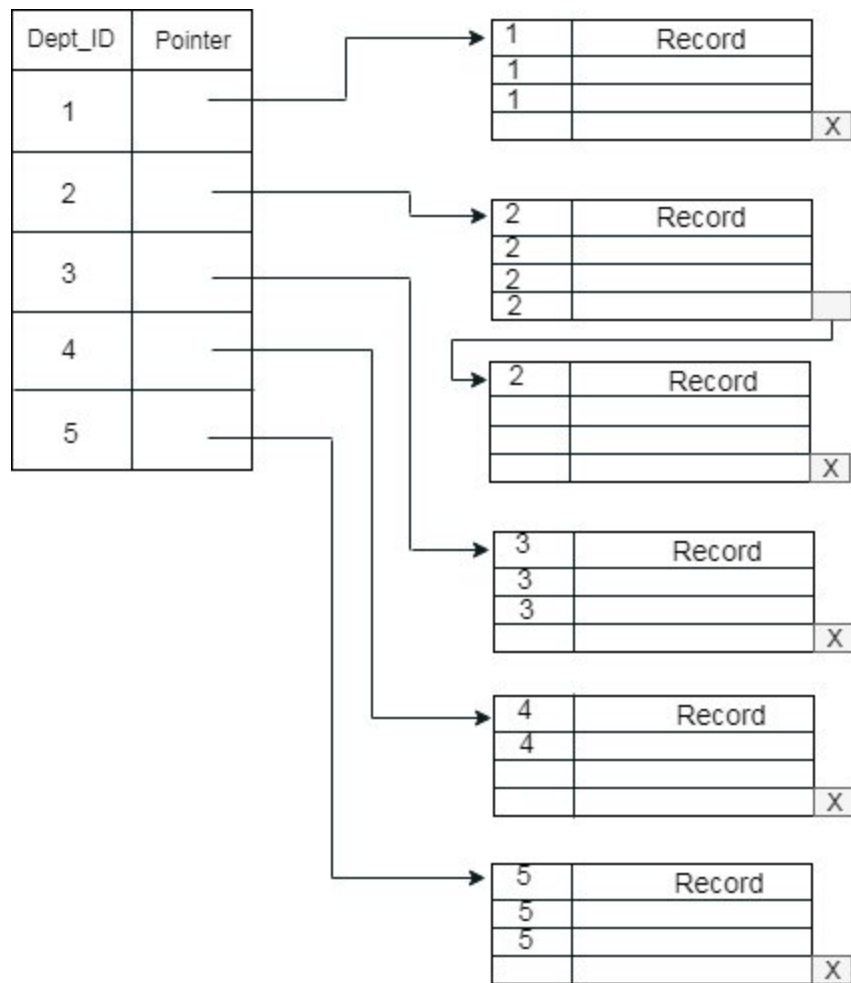
Clustering Index

- 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.



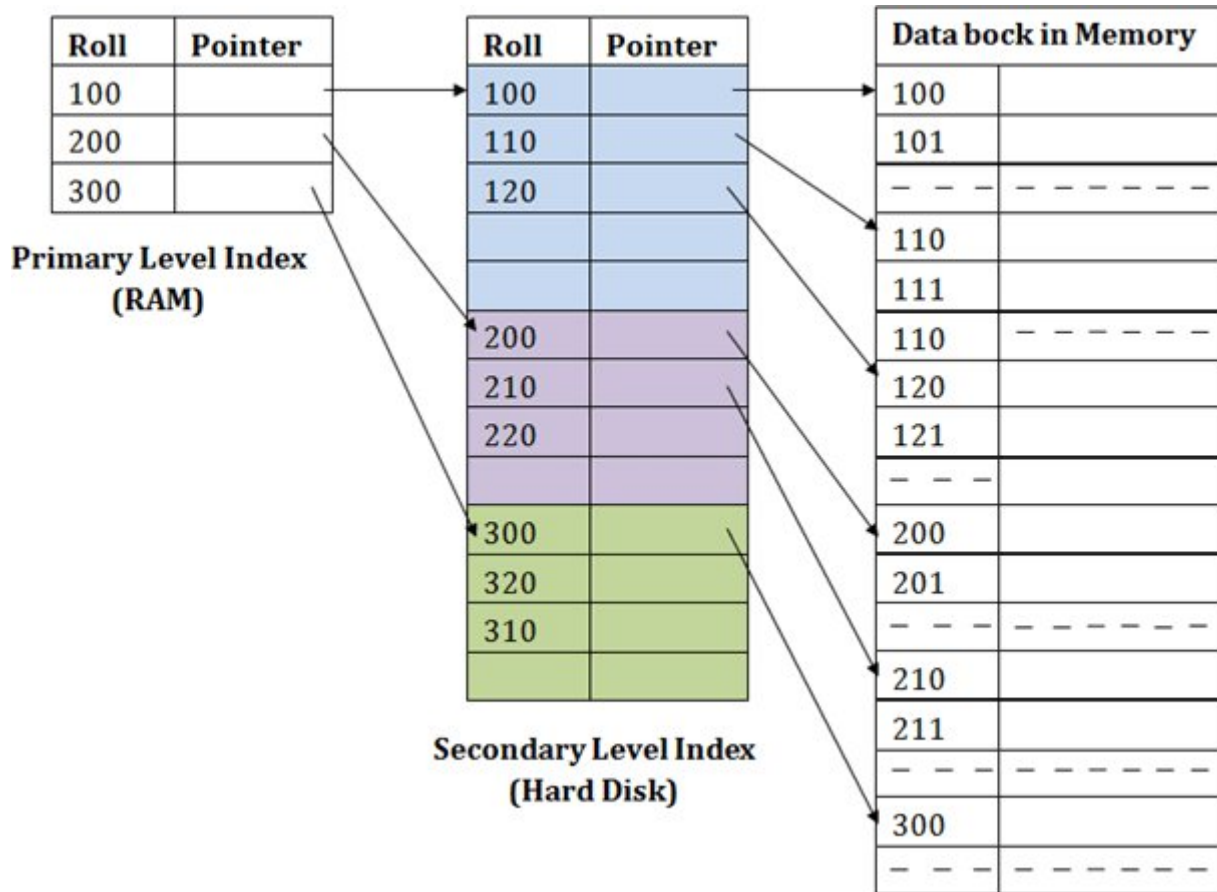
The previous schema is little confusing because one disk block is shared by records which belong to the different cluster. If we use separate disk block for separate clusters, then it is called better technique.



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 $\max(111) \leq 111$ 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.

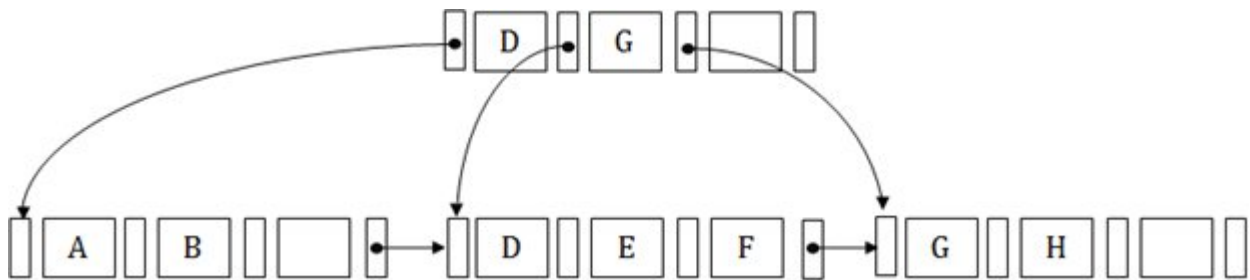
B+ Tree

- 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.

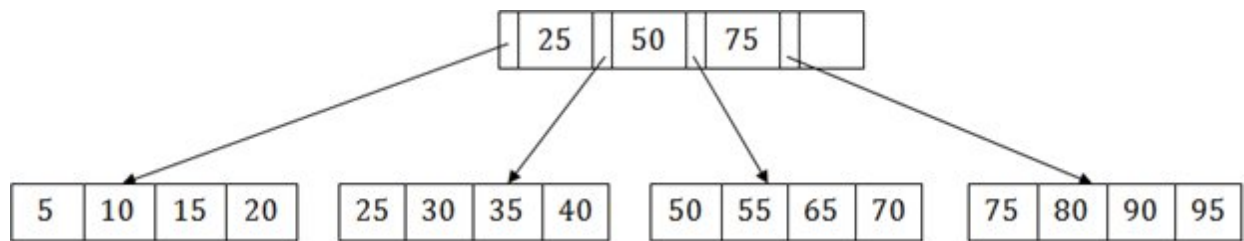
Leaf node

- 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.

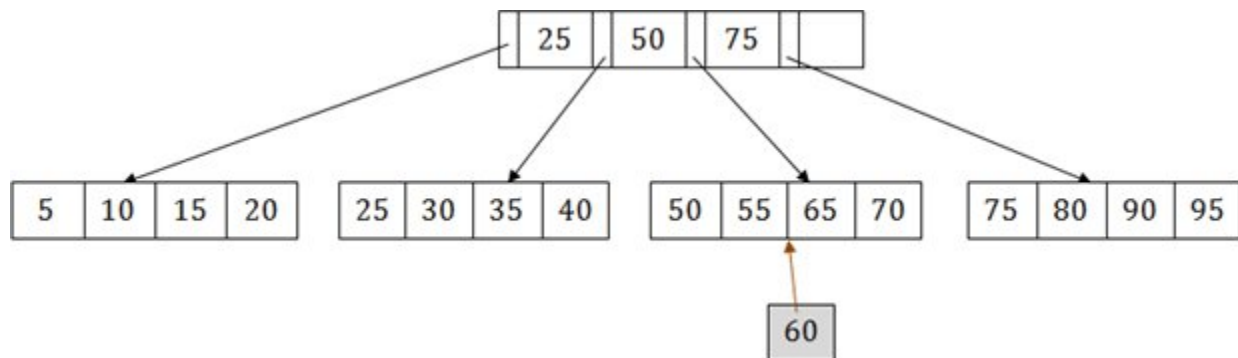
So, in the intermediary node, we will find a branch between 50 and 75 nodes. Then at the end, we will be redirected to the third leaf node. Here DBMS will perform a sequential search to find 55.



B+ Tree Insertion

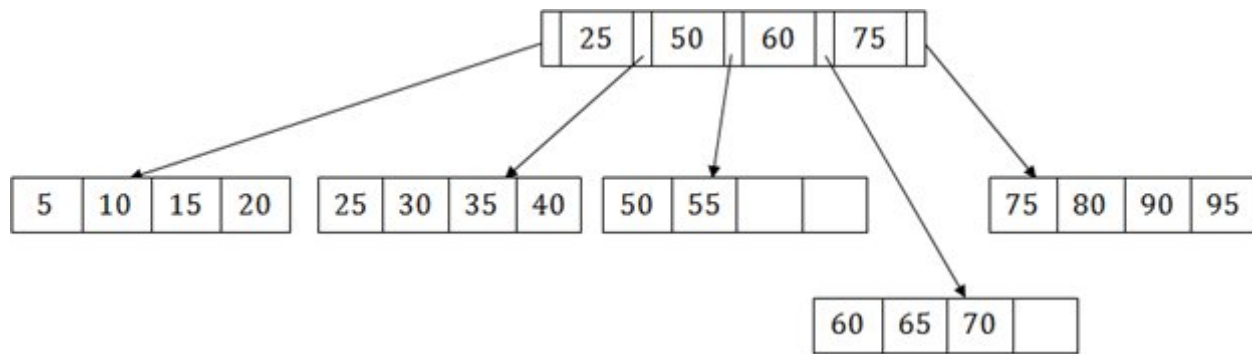
Suppose we want to insert a record 60 in the below structure. It will go to the 3rd leaf node after 55. It is a balanced tree, and a leaf node of this tree is already full, so we cannot insert 60 there.

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.

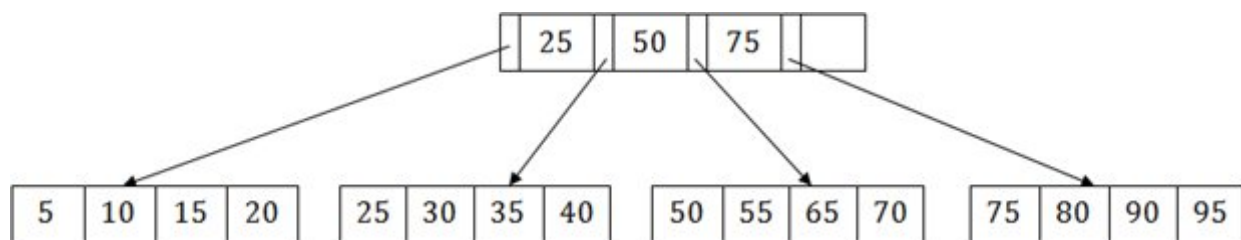


This is how we can insert an entry when there is overflow. In a normal scenario, it is very easy to find the node where it fits and then place it in that leaf node.

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:



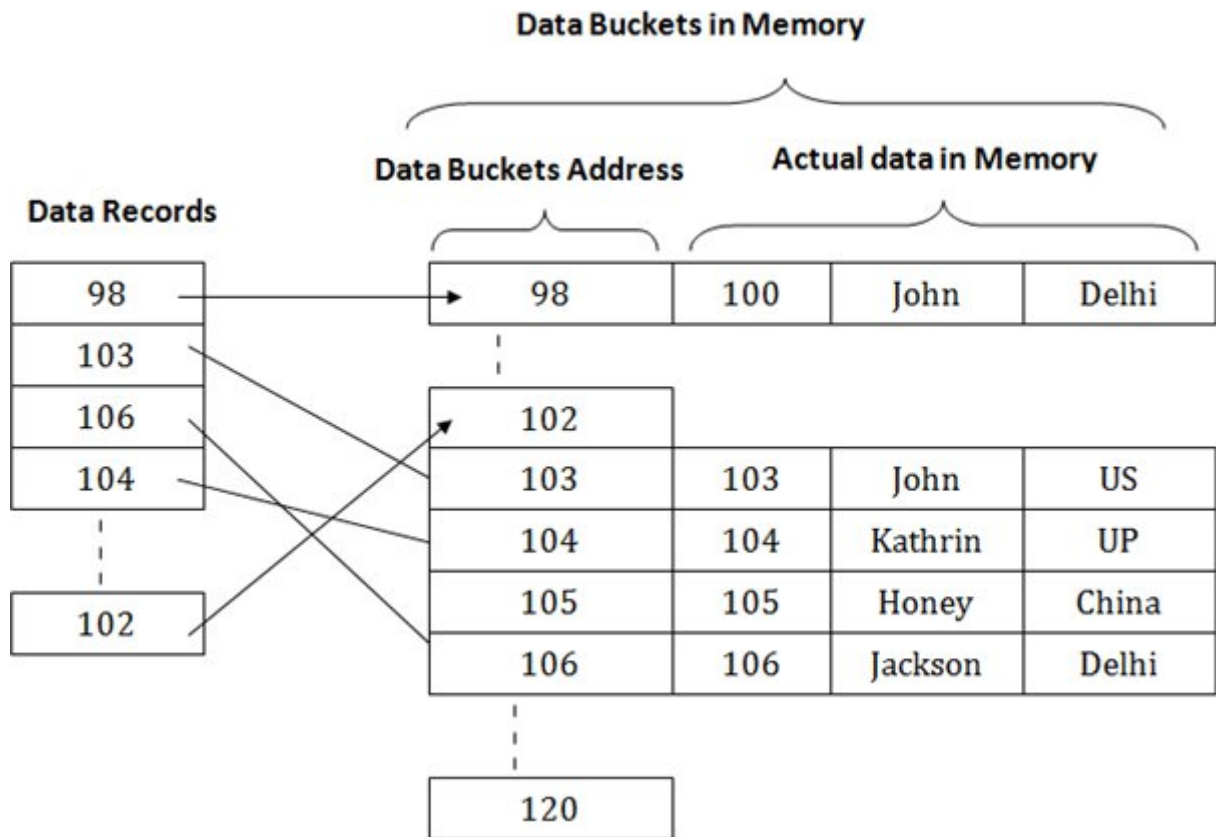
Hashing in DBMS

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.

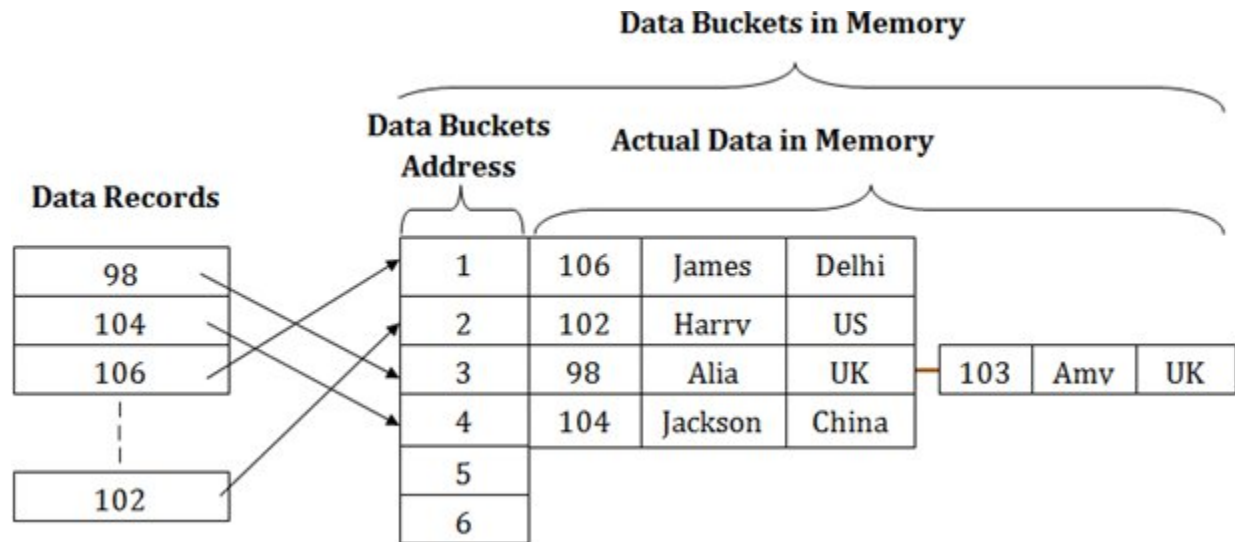
In this technique, data is stored at the data blocks whose address is generated by using the hashing function. The memory location where these records are stored is known as data bucket or data blocks.

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

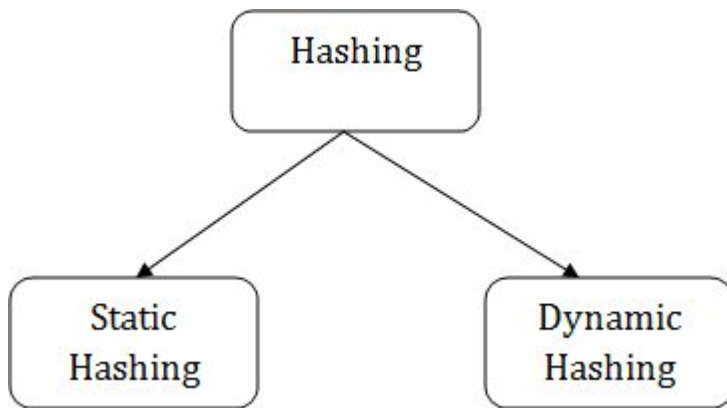
consider the primary key itself as the address of the data block. That means each row whose address will be the same as a primary key stored in the data block.



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:

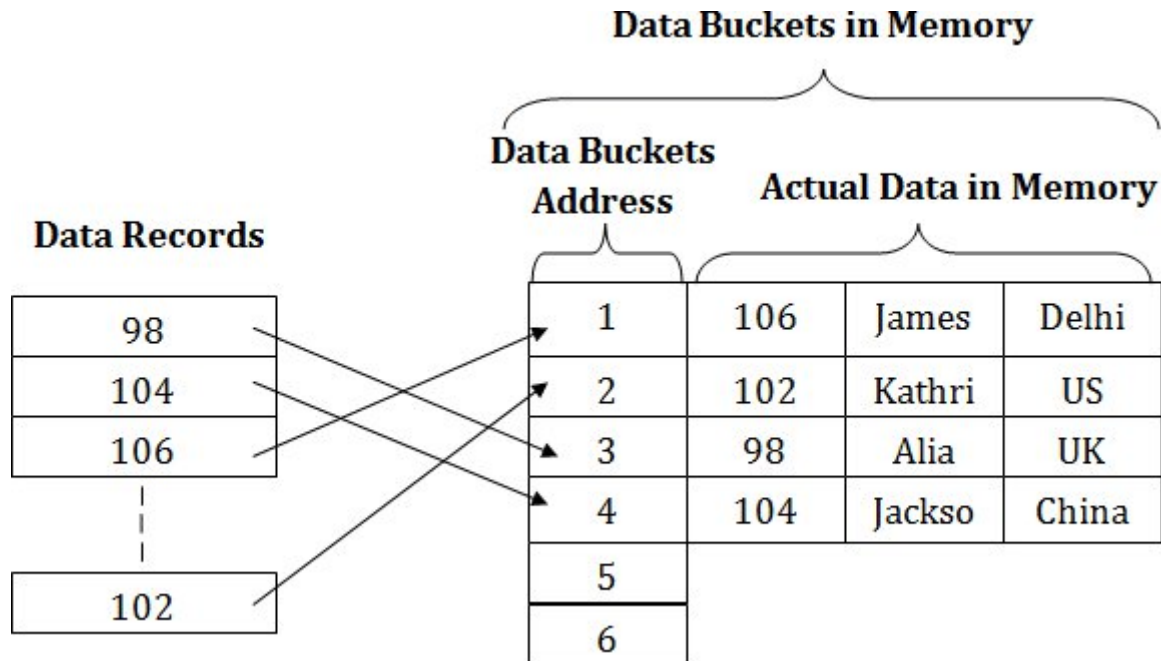


- [Static Hashing](#)
- [Dynamic Hashing](#)

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.

Backward Skip 10sPlay VideoForward Skip 10s

- **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.

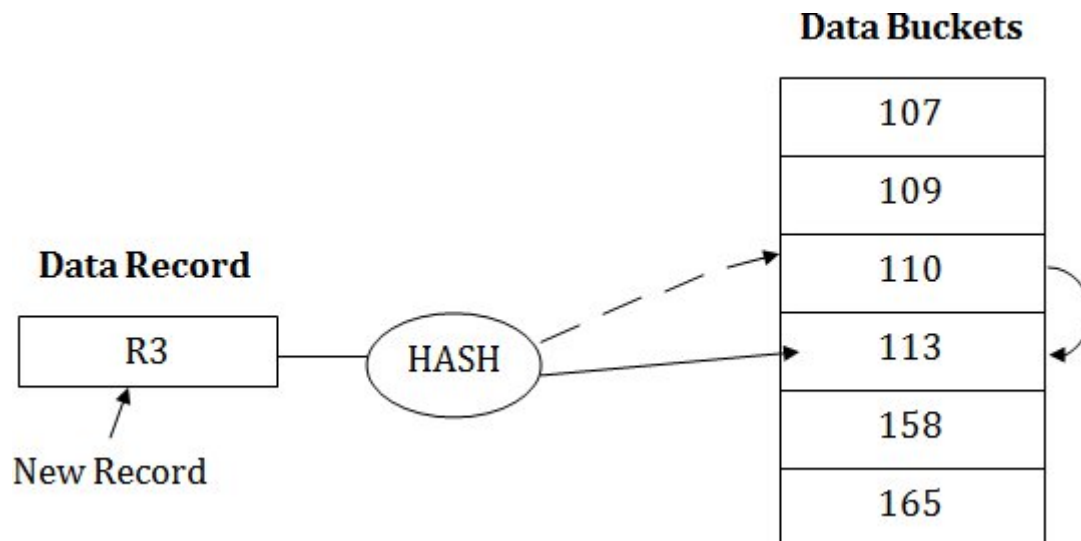
If we want to insert some new record into the file but the address of a data bucket generated by the hash function is not empty, or data already exists in that address. This situation in the static hashing is known as **bucket overflow**. This is a critical situation in this method.

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**.

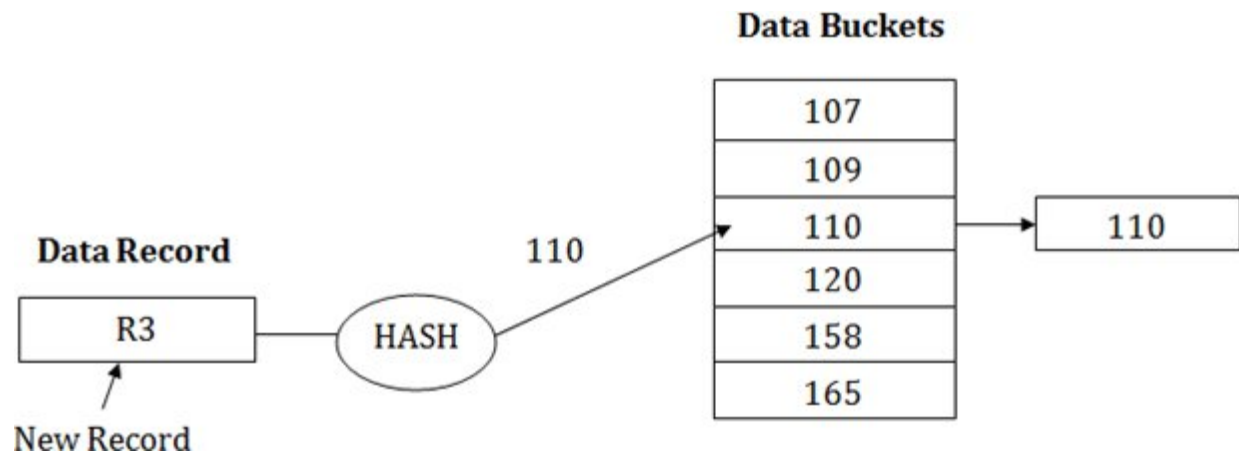
For example: suppose R3 is a new address which needs to be inserted, the hash function generates address as 112 for R3. But the generated address is already full. So the system searches next available data bucket, 113 and assigns R3 to it.



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 increase or decrease. 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

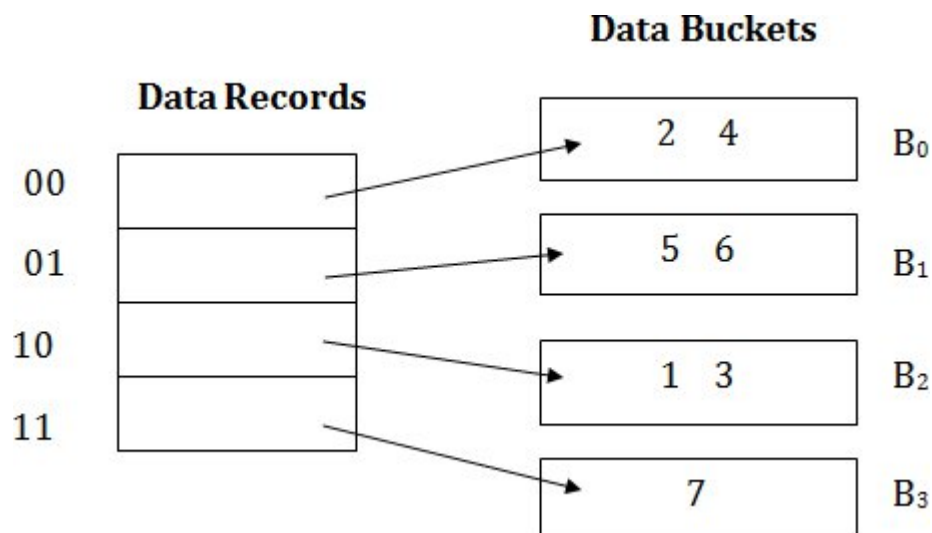
- Firstly, you have to follow the same procedure for retrieval, ending up in some bucket.
- If there is still space in that bucket, then place the record in it.
- If the bucket is full, then we will split the bucket and redistribute the records.

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

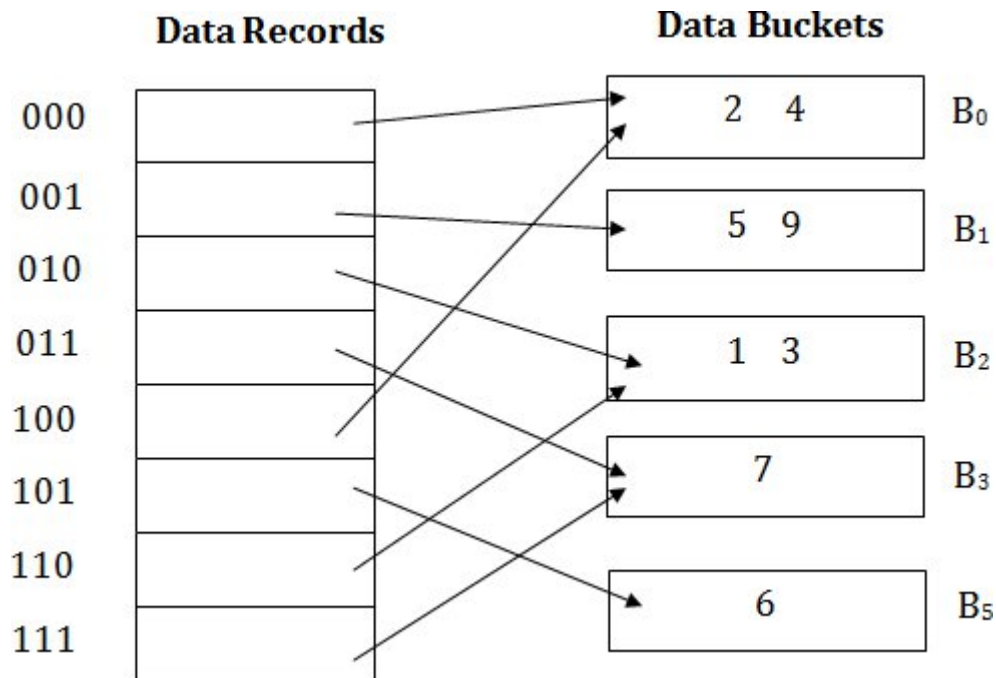
The last two bits of 2 and 4 are 00. So it will go into bucket B₀. The last two bits of 5 and 6 are 01, so it will go into bucket B₁. The last two bits of 1 and 3 are 10, so it will go into bucket B₂. The last two bits of 7 are 11, so it will go into B₃.



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.

Symbols used in ER Diagram