ANNAMACHARYA UNIVERSITY, RAJAMPET

(ESTD UNDER AP PRIVATE UNIVERSITIES (ESTABLISHMENT AND REGULATION) ACT, 2016
RAJAMPET, Annamayya District, AP, INDIA

Course	:	**Database Management Systems**
Course Code	:	**24FMCA12T**
Branch	:	**MCA**
Prepared by	:	**S.THABREEZ BASHA**
Designation	:	**Assistant Professor**
Department	:	**MCA**

ANNAMACHARYA UNIVERSITY, RAJAMPET

(ESTD UNDER AP PRIVATE UNIVERSITIES (ESTABLISHMENT AND REGULATION) ACT, 2016
RAJAMPET, Annamayya District, AP, INDIA

Title of the Course	**:**	**Database Management Systems**
Category	**:**	**PC**
Couse Code	**:**	**24FMCA12T**
Branch	**:**	**MCA**
Semester	**:**	**I Semester**

Lecture Hours	Tutorial Hours	Practice Hours	Credits
3	0	0	3

COURSE OBJECTIVES:

- Explain database concepts and structures and terms related to database design, transactions and management.
- Demonstrate data modeling, normalization and development of the database.
- Formulate SQL statements for data definition, modification and retrieval of data.
- Design and build a simple database system..
- Analyze how databases are affected by real-world transactions.

UNIT I **8 Hrs**

INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS: Data Vs Information, Purpose of databases, Views of data, Database languages, Data models, Database architecture -users and administrators. E-R Model, Entity Relationship diagrams, E-R diagrams design issues, Extended E-R features, Specialization, Generalization, Aggregation.

UNIT II **8 Hrs**

RELATIONAL MODEL: Structure of Relational database, Relational algebra, Tuple relational calculus, Domain relational calculus, QBE (Query-by-Example).

UNIT III **12 Hrs**

STRUCTURED QUERY LANGUAGE (SQL): Introduction to SQL, SQL Operators, SQL Functions, Join queries, Sub queries, Nested queries, Views, Integrity constraints, Functional Dependencies, Database design Normalization: Normal Forms-1st, 2nd, 3rd and BCNF, Multi - Valued Dependency-4th Normal Form, 5th NF/Projection Join Normal form and De- Normalization.

UNIT IV **12 Hrs**

PL/SQL: Introduction to PL/SQL, PL/SQL Block Structure, Conditional statements, Iterative Processing with Loops, Triggers, Cursor, exception handling, Procedures, Functions.
FILE STRUCTURE: File Organization, Organization of Records in Files and Data-Dictionary Storage. Indexing and Hashing: Ordered Indices, B+-Tree Index Files, B-Tree Index files, Static and Dynamic Hashing.

UNIT V **8 Hrs**

TRANSACTION MANAGEMENT: Transaction concept, ACID properties, Transaction state, concurrent execution. Recovery System: Storage structure, Recovery and atomicity, Log-Based Recovery, ARIES Recovery Technique and Remote Back systems.

TEXTBOOKS:

1. Abraham Silberschatz, Henry F. Korth and S. Sudarshan. Database system Concepts. McGraw Hill International Edition, 7thEd.
2. Raghurama Krishna, Johannes Gehrke, Database management systems,TMH.

REFERENCE BOOKS:

1. PS Deshpande, SQL/PLSQL for Oracle 9i, dreamtec Press.
2. Elmasri, Navate, Fundamentals of Database Systems, Person Education, 7thEd.

COURSE OUTCOMES:

The Student will be able to

1. Explain the significance of database languages, different data models.
2. Apply the Relational Algebra of Queries with TRC & DRC.
3. Apply SQL concepts, including operators, functions, and effectively create and analyze join queries, sub queries, and nested queries.
4. Comprehend the File management and also ordering concepts and PL/SQL.
5. Comprehend transaction management and Rollback.

CO-PO MAPPING:

Course Outcomes	Foundation Knowledge	Problem Analysis	Development of Solutions	Modern Tool Usage	Individual and Teamwork	Project Management and Finance	Ethics	Life-long Learning
24FMCA012T.1	2	2	1	-	-	-	-	-
24FMCA012T.2	3	2	1	-	-	-	-	-
24FMCA012T.3	3	2	1	-	-	-	-	-
24FMCA012T.4	2	2	1	-	-	-	-	-
24FMCA012T.5	2	2	1	-	-	-	-	-

UNIT I

INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS: Data Vs Information, Purpose of databases, Views of data, Database languages, Data models, Database architecture -users and administrators. E-R Model, Entity Relationship diagrams, E-R diagrams design issues, Extended E-R features, Specialization, Generalization, Aggregation.

Introduction To DBMS

A database management system (DBMS) refers to the technology for creating and managing databases. DBMS is a software tool to organize (create, retrieve, update, and manage) data in a database.

- To develop software applications In less time.
- Data independence and efficient use of data.
- For uniform data administration.
- For data integrity and security.
- For concurrent access to data, and data recovery from crashes.
- To use user-friendly declarative query language.

Components of DBMS

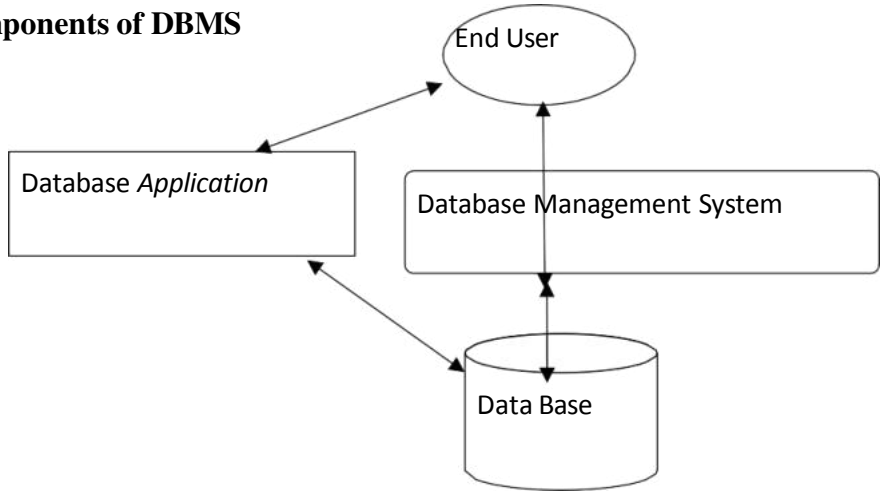


Fig : Components of DBMS

Users: Users may be of any kind such as DB administrator, System developer, or database users.

Database application: Database application may be Departmental, Personal, organization's and / or Internal.

DBMS: Software that allows users to create and manipulate database access,
Database: Collection of logical data as a single unit.

Hardware

- Here the hardware means the physical part of the DBMS. Here the hardware includes output devices like a printer, monitor, etc., and storage devices like a hard disk.
- In DBMS, information hardware is the most important visible part. The equipment which is used for the visibility of the data is the printer, computer, scanner, etc. This equipment is used to capture the data and present the output to the user.
- With the help of hardware, the DBMS can access and update the database.
- The server can store a large amount of data, which can be shared with the help of the user's own system.

Software

Software is defined as the collection of programs that are used to instruct the computer about its work. The software consists of a set of procedures, programs, and routines associated with the computer system's operation and performance. Also, we can say that computer software is a set of instructions that is used to instruct the computer hardware for the operation of the computers.

Data

The term data means the collection of any raw fact stored in the database. Here the data are any type of raw material from which meaningful information is generated.

The database can store any form of data, such as structural data, non-structural data, and logical data.

The structured data are highly specific in the database and have a structured format. But in the case of non-structural data, it is a collection of different types of data, and these data are stored in their native format.

Procedures

The procedure is a type of general instruction or guidelines for the use of DBMS. This instruction includes how to set up the database, how to install the database, how to log in and log out of the database, how to manage the database, how to take a backup of the database, and how to generate the report of the database.

Data VS Information:

Data

Data is a raw and unorganized fact that is required to be processed to make it meaningful. It can be considered as facts and statistics collected together for reference or analysis.

Information:

Information is delineate because the structured, organized, and processed data, conferred inside context, that makes it relevant and helpful to the one who desires it. Data suggests that raw facts and figures regarding individuals, places, or the other issue, that is expressed within the type of numbers letters or symbols.

Difference between Information and Data:

The major differences between Data and Information are as follows –

Data	Information
Data is the raw fact.	It is a processed form of data.
It is not significant to a business.	It is significant to a business.
Data is an atomic level piece of information.	It is a collection of data.
Example: Product name, Name of student.	Example: Report card of student.
It is a phenomenal fact.	It is organized data.
This is the primary level of intelligence.	It is a secondary level of intelligence.
May or may not be meaningful.	Always meaningful.
Understanding is difficult.	Understanding is easy.

Purpose of Databases:

It is a collection of tools that enable users to create and manage databases. In other words, it is general-purpose software that allows users to create, manipulate, and design databases for a number of purposes.

Database systems are design to deal with large volumes of data. Data management comprises both the construction of data storage systems and the provision of data manipulation methods. Furthermore, the database system must maintain the security of the information held despite system crashes or attempts at unauthorized access. The system must avoid any unexpected effects if data is to be shared across multiple users.

Characteristics of DBMS

- Firstly, It manages and stores information in a server-based digital repository.
- Secondly, It can logically and visibly represent the data transformation process.
- Automatic backup and recovery techniques are built into the database management system.
- It has ACID features, which ensure that data is safe even if the system fails.
- It has the ability to make complex data connections more understandable.
- It's utilise to help with data manipulation and processing.
- It is utilise to keep information safe.
- Lastly, It can examine the database from a variety of perspectives, depending on the needs of the user.

Advantages of DBMS

1. Data Structure

They organize data into tables with rows and columns, providing a structured and consistent way to store and manage data. This structure facilitates easy access, retrieval, and manipulation of data.

2. Data Integrity

They enforce data integrity through constraints, such as primary keys, foreign keys, and unique constraints. These constraints ensure that the data remains accurate and consistent, preventing the introduction of duplicate or invalid information.

3. Data Relationships

They enable the establishment of relationships between tables using primary and foreign keys. This feature allows data from different tables to be linked together, making it easier to query and analyze related information.

4. Querying and Reporting

They provide powerful query languages (e.g., SQL) that allow users to retrieve specific data from the database quickly. This simplicity and flexibility make it easier to generate reports and

gain insights from the data.

5. Data Security

They offer robust security mechanisms to protect sensitive data. Access control features allow administrators to define user permissions and restrict unauthorized access to data.

6. Scalability

They can handle large amounts of data and scale up to accommodate the growth of the dataset and user demand. Many RDBMS platforms offer high-availability options and clustering to ensure system availability even during peak usage.

7. ACID Transactions

They support ACID (Atomicity, Consistency, Isolation, Durability) transactions, which guarantee that database operations are reliable and that data remains in a consistent state even in the event of failures.

8. Data Backup and Recovery

They provide tools and mechanisms to perform regular backups of the database, enabling data recovery in case of hardware failures, accidental deletions, or other unforeseen issues.

9. Data Indexing

They use indexing techniques to optimize data retrieval. Indexes speed up query execution by reducing the need for full-table scans, especially for large datasets.

10. Data Normalization

They encourage data normalization, which helps eliminate data redundancy and inconsistencies. Normalized data ensures efficient storage and minimizes update anomalies.

11. Industry Standard

RDBMS, such as MySQL, Oracle, Microsoft SQL Server, and PostgreSQL, have been around for decades and have established themselves as reliable, widely used, and well-supported solutions.

12. Data Consistency

They enforce referential integrity, ensuring that relationships between data remain consistent. This prevents data inconsistencies and maintains the accuracy and reliability of the information stored in the database.

13. Data Independence

They offer data abstraction, separating the logical structure of the database from its physical implementation. This data independence allows developers to modify the database schema without affecting the applications built on top of it, promoting flexibility and reducing maintenance efforts.

14. Data Accessibility

They provide concurrent access to multiple users and applications. Users can read and write data simultaneously without interfering with each other's operations, promoting efficient collaboration and multi-user support.

15. Data Backup and Recovery

They typically offer various backup and recovery options, including full, incremental, and differential backups. These features ensure data protection and the ability to restore the database to a specific point in time.

Disadvantages of RDBMS

While Relational Database Management Systems (RDBMS) offer numerous advantages, they are not without their drawbacks. Here are some of the limitations of RDBMS:

1. Scalability Limitations

RDBMS may face challenges when dealing with extremely large datasets and high transaction volumes. As the data size grows, the performance of the RDBMS can degrade, requiring careful database design and optimization to maintain efficiency.

2. Complex Design

Designing a relational database schema can be complex, especially for large and intricate applications. Ensuring proper normalization and establishing appropriate relationships between tables can be time-consuming and require significant expertise.

3. Fixed Schema

They have a fixed schema, meaning the structure of the database is defined beforehand. Adding new columns or altering the schema often requires modifying existing applications, which can be cumbersome and may lead to downtime during updates.

4. Performance Bottlenecks

Certain operations, such as complex joins, can lead to performance bottlenecks in RDBMS, especially when dealing with large datasets. Proper indexing and query optimization are essential to mitigate these issues.

5. High Overhead

They typically have more overhead compared to some NoSQL databases. The need for data normalization, transactions, and referential integrity enforcement can result in increased storage requirements and slower performance.

6. Cost

Some commercial RDBMS solutions can be expensive, especially when considering licensing, maintenance, and hardware requirements. While open-source options like MySQL and PostgreSQL exist, implementing and managing RDBMS still incurs costs.

7. Replication Complexity

While RDBMS supports replication, setting up and managing replication can be complex, particularly in distributed and multi-data center environments.

8. Single Point of Failure

In traditional RDBMS setups, the database server can become a single point of failure. Ensuring high availability often requires implementing clustering or failover mechanisms, which can add complexity to the system.

9. Data Modeling Challenges

Representing certain types of data, such as hierarchical or unstructured data, can be challenging in a relational database. NoSQL databases may be better suited for handling these specific data types.

10. Data Type Limitations

RDBMS have predefined data types, and accommodating certain data formats or unstructured data can be difficult without resorting to workarounds.

12. Complex Joins and Performance

While relational databases are excellent at handling structured data and enforcing data integrity, complex joins involving multiple tables can become a performance challenge. As the number of tables and the complexity of relationships increase, query execution times may also increase significantly.

13. Difficulty with Semi-Structured and Unstructured Data

RDBMS is not well-suited for handling semi-structured and unstructured data like JSON, XML, or documents. Some RDBMS provide support for storing and querying such data through specialized data types or extensions.

Views of Data:

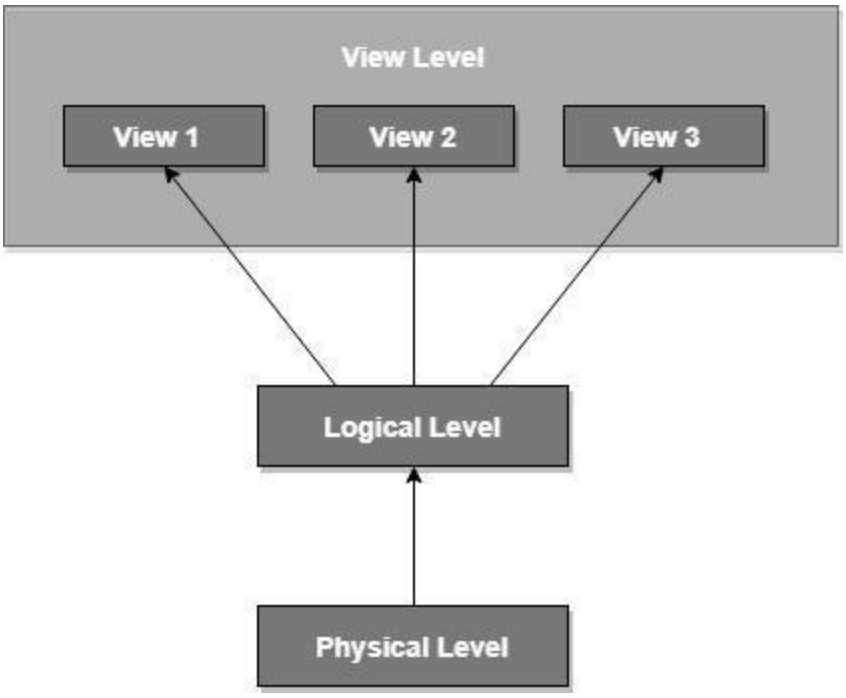
Database systems comprise complex data structures. In order to make the system efficient in terms of retrieval of data, and reduce complexity in terms of usability of users, developers use abstraction. The data should hide irrelevant details from the users. This approach simplifies database design.

There are mainly **3** levels of data abstraction:

Physical: This is the lowest level of data abstraction. It tells us how the data is actually stored in memory. The access methods like sequential or random access and file organization methods like B+ trees and hashing are used for the same. Usability, size of memory, and the number of times the records are factors that we need to know while designing the database. Suppose we need to store the details of an employee. Blocks of storage and the amount of memory used for these purposes are kept hidden from the user.

Logical: This level comprises the information that is actually stored in the database in the form of tables. It also stores the relationship among the data entities in relatively simple structures. At this level, the information available to the user at the view level is unknown. We can store the various attributes of an employee and relationships, e.g. with the manager can also be stored.

View: This is the highest level of abstraction. Only a part of the actual database is viewed by the users. This level exists to ease the accessibility of the database by an individual user. Users view data in the form of rows and columns. Tables and relations are used to store data. Multiple views of the same database may exist. Users can just view the data and interact with the database, storage and implementation details are hidden from them.



The main purpose of data abstraction is to achieve data independence in order to save the time and cost required when the database is modified or altered.

We have namely two levels of data independence arising from these levels of abstraction.

Physical level data independence: It refers to the characteristic of being able to modify the physical schema without any alterations to the conceptual or logical schema, done for optimization purposes, e.g., the Conceptual structure of the database would not be affected by any change in storage size of the database system server. Changing from sequential to random access files is one such example. These alterations or modifications to the physical structure may include:

- Utilizing new storage devices.
- Modifying data structures used for storage.
- Altering indexes or using alternative file organization techniques etc.

Logical level data independence: It refers characteristic of being able to modify the logical schema without affecting the external schema or application program. The user view of the data would not be affected by any changes to the conceptual view of the data. These changes may include insertion or deletion of attributes, altering table structures entities or relationships to the logical schema, etc.

Data Schema and Instances:

- The data which is stored in the database at a particular moment of time is called an instance of the database.
- The overall design of a database is called schema.
- A database schema is the skeleton structure of the database. It represents the logical view of the entire database.
- A schema contains schema objects like table, foreign key, primary key, views, columns, data types, stored procedure, etc.
- A database schema can be represented by using the visual diagram. That diagram shows the database objects and relationship with each other.
- A database schema is designed by the database designers to help programmers whose software will interact with the database. The process of database creation is called data modeling.

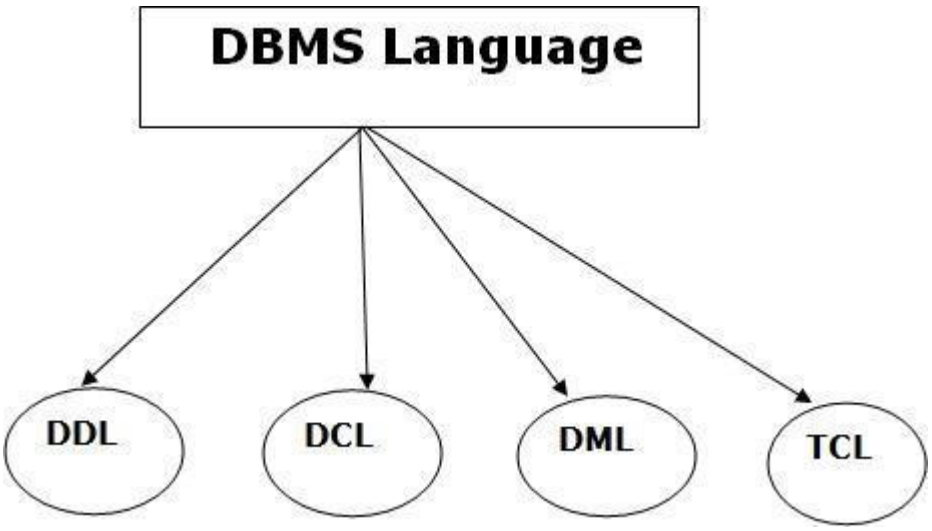
A schema diagram can display only some aspects of a schema like the name of record type, data type, and constraints. Other aspects can't be specified through the schema diagram. For example, the given figure neither shows the data type of each data item nor the relationship among various files.

In the database, actual data changes quite frequently. For example, in the given figure, the database changes whenever we add a new grade or add a student. The data at a particular moment of time is called the instance of the database.

Database Languages (OR) Basic SQL commands

- A DBMS has appropriate languages and interfaces to express database queries and updates.
- Database languages can be used to read, store and update the data in the database.

Types of Database Language



1. Data Definition Language

- **DDL** stands for **D**ata **D**efinition **L**anguage. It is used to define database structure or pattern.
- It is used to create schema, tables, indexes, constraints, etc. in the database.
- Using the DDL statements, you can create the skeleton of the database.
- Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Here are some tasks that come under DDL:

- **Create:** It is used to create objects in the database.
- **Alter:** It is used to alter the structure of the database.
- **Drop:** It is used to delete objects from the database.
- **Truncate:** It is used to remove all records from a table.

- **Rename:** It is used to rename an object.
- **Comment:** It is used to comment on the data dictionary.

These commands are used to update the database schema that's why they come under Data definition language.

2. Data Manipulation Language

DML stands for **D**ata **M**anipulation **L**anguage. It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

- **Select:** It is used to retrieve data from a database.
- **Insert:** It is used to insert data into a table.
- **Update:** It is used to update existing data within a table.
- **Delete:** It is used to delete all records from a table.
- **Merge:** It performs UPSERT operation, i.e., insert or update operations.
- **Call:** It is used to call a structured query language or a Java subprogram.
- **Explain Plan:** It has the parameter of explaining data.
- **Lock Table:** It controls concurrency.

3. Data Control Language

- **DCL** stands for **D**ata **C**ontrol **L**anguage. It is used to retrieve the stored or saved data.
- The DCL execution is transactional. It also has rollback parameters.
  (But in Oracle database, the execution of data control language does not have the feature of rolling back.)

Here are some tasks that come under DCL:

- **Grant:** It is used to give user access privileges to a database.
- **Revoke:** It is used to take back permissions from the user.

There are the following operations which have the authorization of Revoke:

CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

4. Transaction Control Language

TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

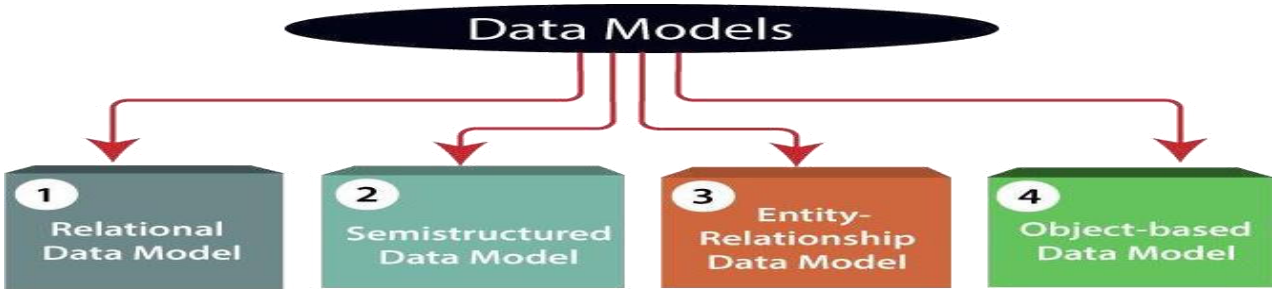
Here are some tasks that come under TCL:

- **Commit:** It is used to save the transaction on the database.

- **Rollback:** It is used to restore the database to original since the last Commit.

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:



1) Relational Data Model: This type of model designs the data in the form of rows and columns within a table. Thus, a relational model uses tables for representing data and in-between relationships. Tables are also called relations. This model was initially described by Edgar F. Codd, in 1969. The relational data model is the widely used model which is primarily used by commercial data processing applications.

2) Entity-Relationship Data Model: An ER model is the logical representation of data as objects and relationships among them. These objects are known as entities, and relationship is an association among these entities. This model was designed by Peter Chen and published in 1976 papers. It was widely used in database designing. A set of attributes describe the entities. For example, student_name, student_id describes the 'student' entity. A set of the same type of entities is known as an 'Entity set', and the set of the same type of relationships is known as 'relationship set'.

3) Object-based Data Model: An extension of the ER model with notions of functions, encapsulation, and object identity, as well. This model supports a rich type system that includes structured and collection types. Thus, in 1980s, various database systems following the object-oriented approach were developed. Here, the objects are nothing but the data carrying its properties.

4) Semi structured Data Model:

This type of data model is different from the other three data models. The semi structured data model allows the data specifications at places where the individual data items of the same type may have different attributes sets. The Extensible Markup Language, also known as XML, is widely used for representing the semi structured data. Although XML was initially designed for including the markup information to the text document, it gains importance because of its application in the exchange of data.

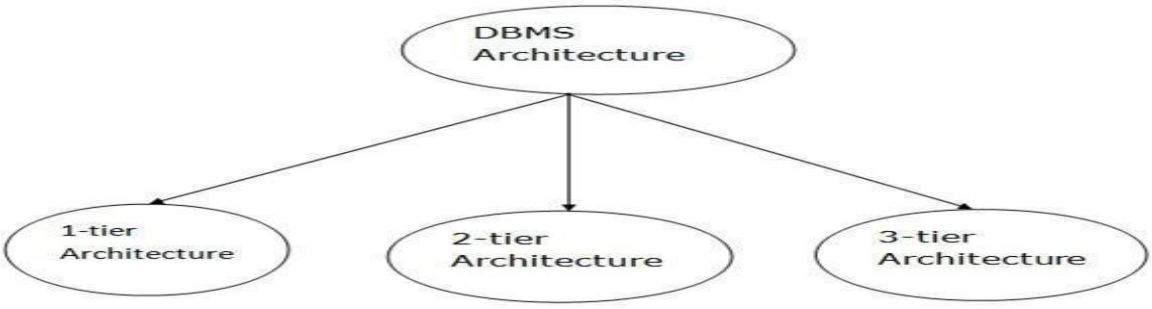
Database Architecture User and Administrator:

The DBMS design depends upon its architecture. The basic client/server architecture is used to deal with a large number of PCs, web servers, database servers and other components that are connected with networks.

The client/server architecture consists of many PCs and a workstation which are connected via the network.

DBMS architecture depends upon how users are connected to the database to get their request done.

Types of DBMS Architecture



Database architecture can be seen as a single tier or multi-tier. But logically, database architecture is of two types like: **2-tier architecture** and **3-tier architecture**.

1- Tier Architecture

- In this architecture, the database is directly available to the user. It means the user can directly sit on the DBMS and uses it.

- Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.
- The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.

2- Tier Architecture

- The 2-Tier architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like: **ODBC**, **JDBC** are used.
- The user interfaces and application programs are run on the client-side.
- The server side is responsible to provide the functionalities like: query processing and transaction management.
- To communicate with the DBMS, client-side application establishes a connection with the server side.

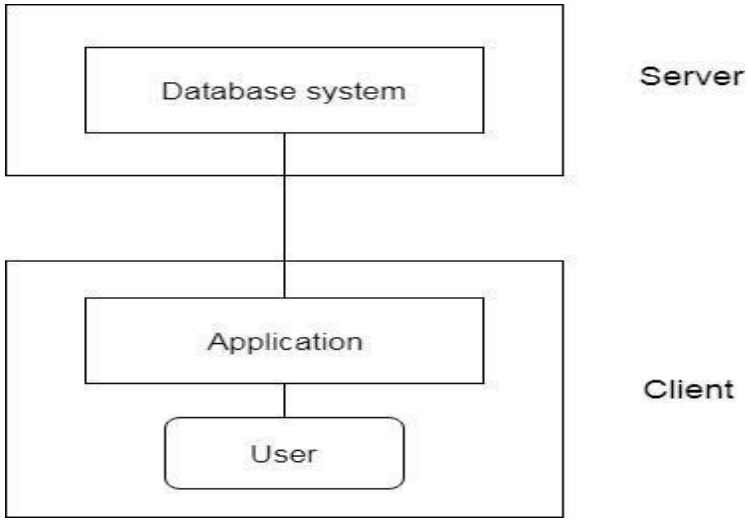


Fig: 2-tier Architecture

3- Tier Architecture

- The 3-Tier architecture contains another layer between the client and server. In this architecture, client can't directly communicate with the server.
- The application on the client-end interacts with an application server which further communicates with the database system.

- End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.
- The 3-Tier architecture is used in case of large web application.

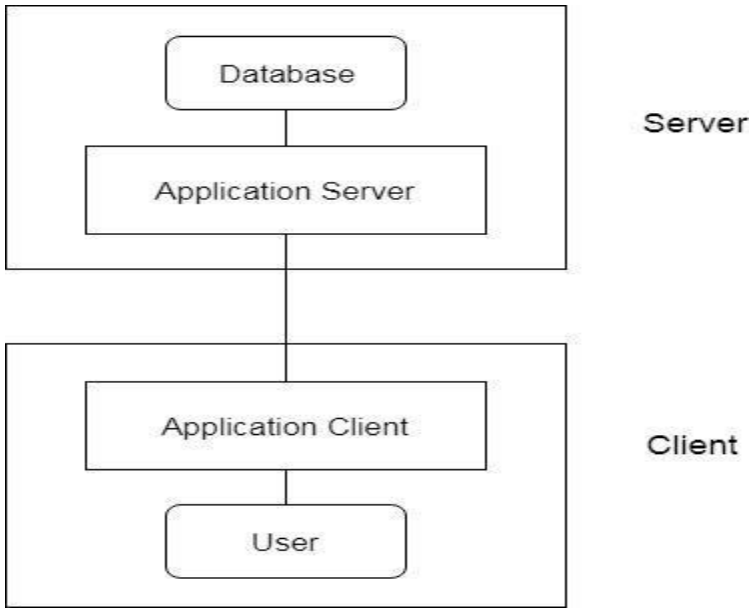


Fig: 3-tier Architecture

User:

1. **Naive / Parametric End Users :** Parametric End Users are the unsophisticated who don't have any DBMS knowledge but they frequently use the database applications in their daily life to get the desired results. For examples, Railway's ticket booking users are naive users. Clerks in any bank is a naive user because they don't have any DBMS knowledge but they still use the database and perform their given task.
2. **System Analyst:**

System Analyst is a user who analyzes the requirements of parametric end users. They check whether all the requirements of end users are satisfied.

3. **Sophisticated Users:**

Sophisticated users can be engineers, scientists, business analyst, who are familiar with the database. They can develop their own database applications according to their requirement. They don't write the program code but they interact the database by writing SQL queries directly through the query processor.

4. **Database Designers:** Data Base Designers are the users who design the structure of database which includes tables, indexes, views, triggers, stored procedures and constraints which are usually enforced before the database is created or populated with data. He/she controls what data must be stored and how the data items to be related. It is responsibility of Database Designers to understand the requirements of different user groups and then create a design which satisfies the need of all the user groups.
5. **Application Programmers:** Application Programmers also referred as System Analysts or simply Software Engineers, are the back-end programmers who writes the code for the application programs. They are the computer professionals. These programs could be written in Programming languages such as Visual Basic, Developer, C, FORTRAN, COBOL etc. Application programmers design, debug, test, and maintain set of programs called "canned transactions" for the Naive (parametric) users in order to interact with database.
6. **Casual Users / Temporary Users:** Casual Users are the users who occasionally use/access the database but each time when they access the database they require the new information, for example, Middle or higher level manager.

Administrator:

A **Database Administrator (DBA)** is an individual or person responsible for controlling, maintaining, coordinating, and operating a database management system. Managing, securing, and taking care of the database systems is a prime responsibility. They are responsible and in charge of authorizing access to the database, coordinating, capacity, planning, installation, and monitoring uses, and acquiring and gathering software and hardware resources as and when needed. Their role also varies from configuration, database design, migration, security, troubleshooting, backup, and data recovery. Database administration is a major and key function in any firm or organization that is relying on one or more databases. They are overall commanders of the Database system.

Types of Database Administrator (DBA) :

- **AdministrativeDBA** –
  Their job is to maintain the server and keep it functional. They are concerned with data backups, security, troubleshooting, replication, migration, etc.
- **DataWarehouseDBA** –
  Assigned earlier roles, but held accountable for merging data from various sources into the data warehouse. They also design the warehouse, with cleaning and scrubs data prior to loading.
- **CloudDBA**
  Nowadays companies are preferring to save their workpiece on cloud storage. As it reduces the chance of data loss and provides an extra layer of data security and integrity.
- **DevelopmentDBA**
  They build and develop queries, stores procedure, etc. that meets firm or organization needs. They are par at programming.
- **ApplicationDBA**
  They particularly manage all requirements of application components that interact with the database and accomplish activities such as application installation and coordination, application upgrades, database cloning, data load process management, etc.
- **Architect** –
  They are held responsible for designing schemas like building tables. They work to build a structure that meets organizational needs. The design is further used by developers and development DBAs to design and implement real applications.
- **OLAPDBA**
  They design and build multi-dimensional cubes for determination support or OLAP systems.
- **DataModeler**
  In general, a data modeler is in charge of a portion of a data architect's duties. A data modeler is typically not regarded as a DBA, but this is not a hard and fast rule.
- **Task-OrientedDBA**
  To concentrate on a specific DBA task, large businesses may hire highly specialised DBAs. They are quite uncommon outside of big corporations. Recovery and backup DBA, whose responsibility it is to guarantee that the databases of businesses can be recovered, is an example of a task-oriented DBA. However, this specialism is not present in the majority of firms. These task-oriented DBAs will make sure that highly qualified professionals are working on crucial DBA tasks when it is possible.

- **DatabaseAnalyst**

  This position doesn't actually have a set definition. Junior DBAs may occasionally be referred to as database analysts. A database analyst occasionally performs functions that are comparable to those of a database architect. The term "Data Administrator" is also used to describe database analysts and data analysts. Additionally, some businesses occasionally refer to database administrators as data analysts.

Importance of Database Administrator (DBA) :

- Database Administrator manages and controls three levels of database internal level, conceptual level, and external level of Database management system architecture and in discussion with the comprehensive user community, gives a definition of the world view of the database. It then provides an external view of different users and applications.
- Database Administrator ensures held responsible to maintain integrity and security of database restricting from unauthorized users. It grants permission to users of the database and contains a profile of each and every user in the database.
- Database Administrators are also held accountable that the database is protected and secured and that any chance of data loss keeps at a minimum.
- Database Administrator is solely responsible for reducing the risk of data loss as it backup the data at regular intervals.

Role and Duties of Database Administrator (DBA):

1. **Database backup**:

   A database administrator has the responsibility to back up every data in the database, recurrently. This is necessary, so that operations can be restored in times of disaster or downtime.

2. **Database availability**:

   A database administrator has the responsibility of ensuring database accessibility to users from time to time.

3. **Database restore**:

   A database administrator has the responsibility of restoring a file from a backup state, when there is a need for it.

4. **Database design**:

A database administrator has the responsibility of designing a database that meets the demands of users. Hence, having knowledge of database design is crucial for an administrator.

5. **Data move**:

   A database administrator has the responsibility of moving a database set, say from a physical base to a cloud base, or from an existing application to a new application.

6. **Database upgrade**:

   A database administrator has the responsibility of upgrading database software files when there is a new update for them, as this protects software from security breaches.

7. **Database patch**:

   In times of new upgrades for database software, the database administrator has the responsibility of ensuring that the database system functions perfectly and works to close up any gaps in the new update.

8. **Database security**:

   Datasets are assets, and one major responsibility of database administrators is to protect the data and ensure adequate security in an organization's database.

9. **Capacity planning**:

   A database administrator has the responsibility of planning for increased capacity, in case of sudden growth in database need.

10. **Database monitoring**:

    A database administrator has the responsibility of monitoring the database and the movement of data in the database. Administrators provide access for users who require access to the database.

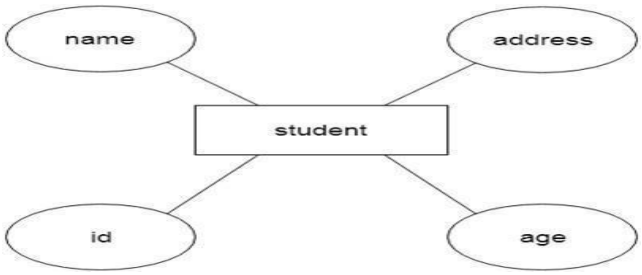
11. **Error log review**:

    A database administrator has the responsibility of interpreting the error messages sent by a database when there is a fault or bridge.

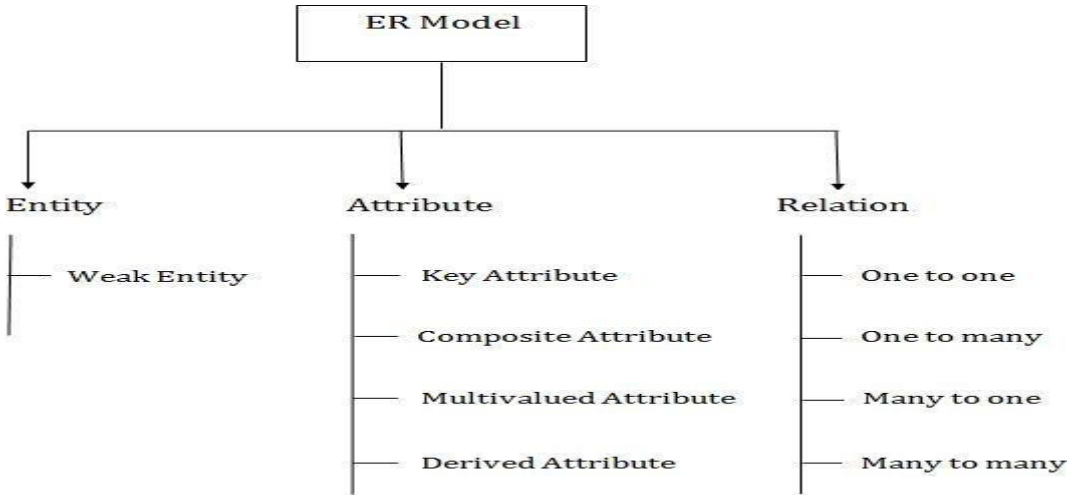
E-R Model:

- 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.



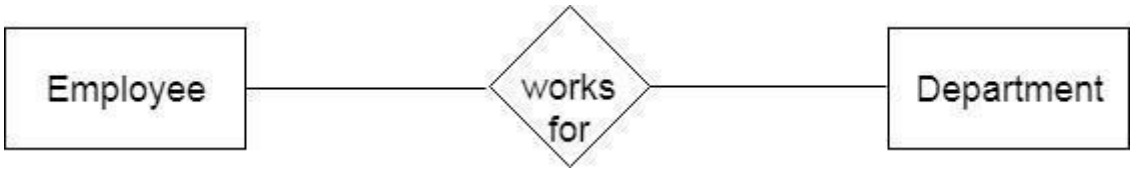
Component of ER Diagram



1. Entity:

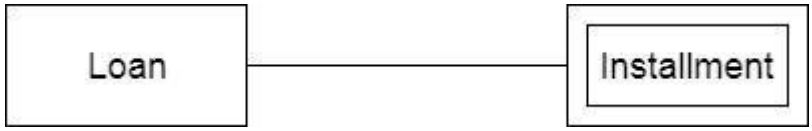
An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.

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



a. Weak Entity

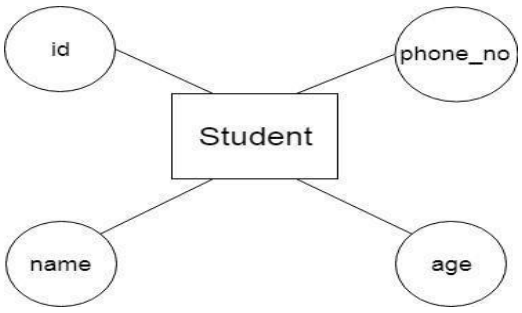
An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.



2. Attribute

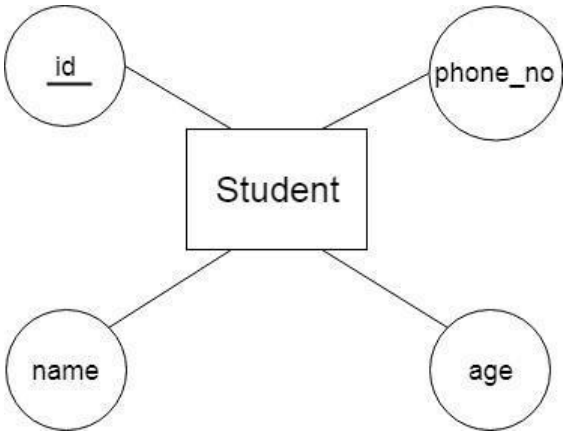
The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.

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



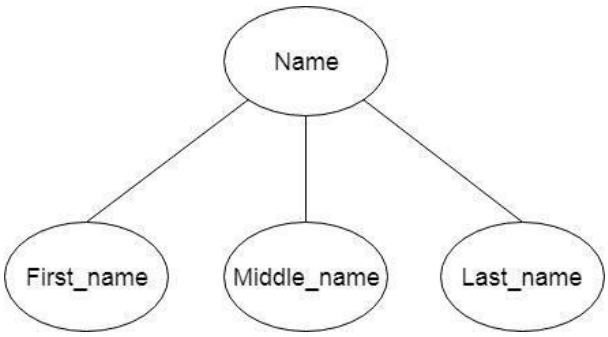
a. Key Attribute

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

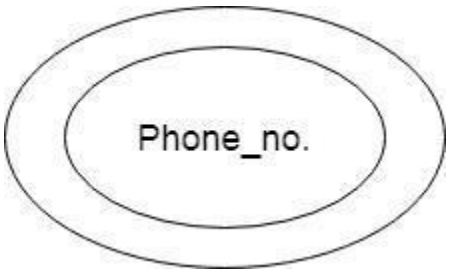
An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.



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.

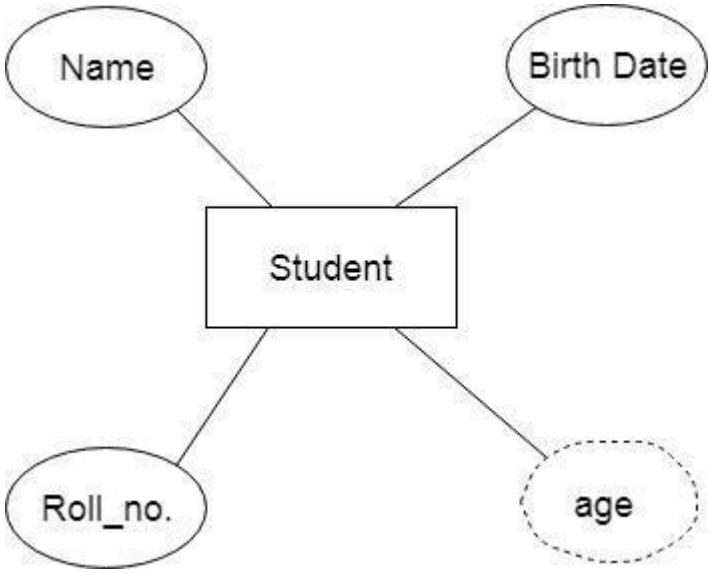
For example, a student can have more than one phone number.



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

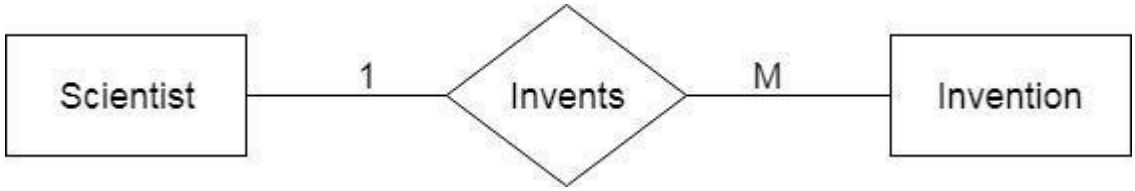
When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.

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

b. One-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.

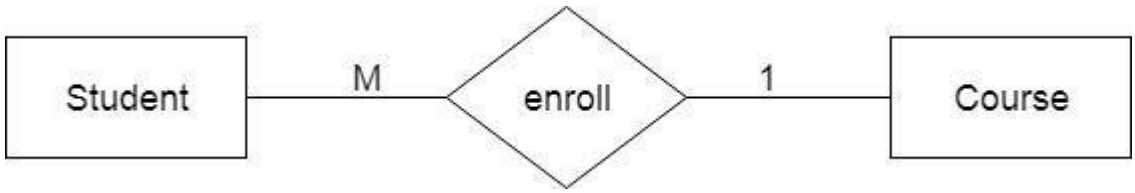
For example, Scientist can invent many inventions, but the invention is done by the only specific scientist.



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.

ER Diagram:

How to Draw (Or) steps an Entity Relation Diagram (ERD)

A step-by-step process to draw an entity relation diagram (ERD) is:

Step 1: Identifying Entities

Determine the main objects you want to represent in the database. Eg, "students", "courses", or "products".

Step 2: Defining Attributes

Identify the properties(attributes) of properties of each entity. These attributes provide more details about an entity.

Step 3: Specifing Relationships

Create relationships between entities to specify how entities interact with each other.

Relationships are verbs like "teaches", "studies", or "sells".

Step 4: Drawing Entities

Draw entities as rectangle and write the name.

Step 5: Adding Attributes

To add attributes of a entitity write attributes inside the rectangle or connect them with lines.

Step 6: Connecting Entities

Draw lines between the related entities to represent their connection.

Step 7: Specifying Cardinality

Indicate the minimum and maximum number of relationship instances associated with an entity using notations like crow's foot.

Step 8: Organizing ER Diagram

Organize all entities and relationships in a clean way for better readibility and understanding.

Draw Entity Relationship Diagram Example

Entity Relationship Diagram for BANK

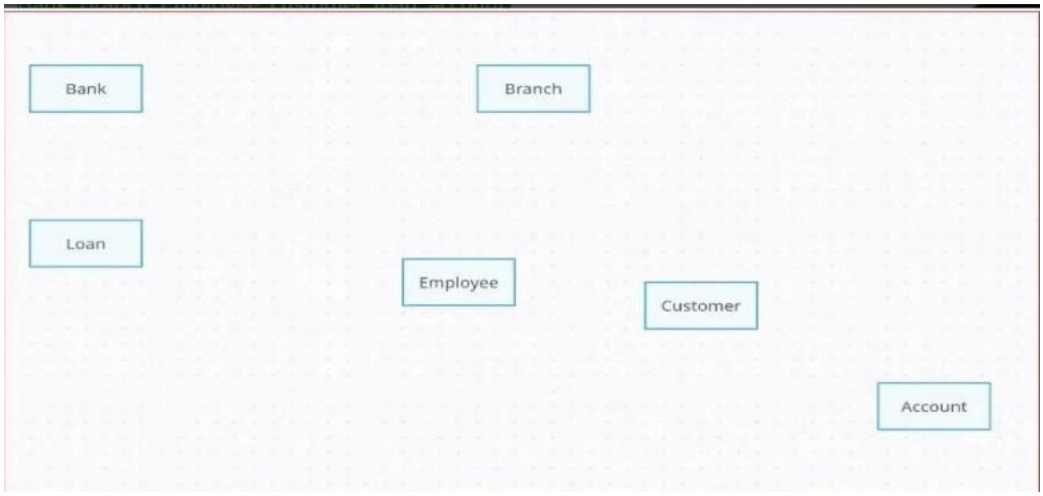
We will follow the steps mentioned above, to draw entity relationship diagram for bank.

Defining Entities

A thing in the **real world** with an independent existence. It is may be an object with physical existence (ex**: house, person)** or with a conceptual existence (ex: **course, job**). The are represented by **rectangle**.

Entities for Bank are:

Bank, Branch, Employee, customer, loan, account.



Adding Attributes

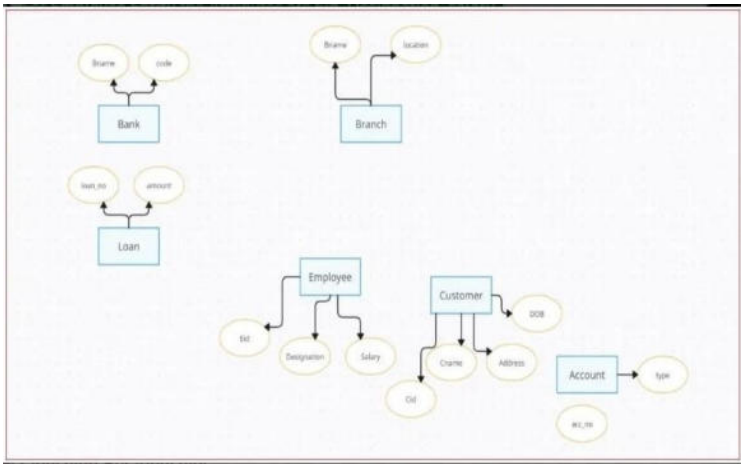
Attributes are the kind of properties that describe the entities. They are represented by **ovals**.

Attributes for Bank are:

- For **Bank Entity** the Attributes are **Bname, code.**
- For **Branch Entity** the Attributes are **Blocation, Bname.**
- For **Employee Entity** the Attributes are **Eid, Designation, salary.**
- For **Customer Entity** the Attributes are **Cid, Cname, Address, DOB.**
- For **Loan Entity** the Attributes are **Loan_no, amount, rate.**
- For **Account Entity** the Attributes are **acc_no, type.**

Establishing Relationships

Entities have some relationships with each other. Relationships define **how entities are**



associated with each other.

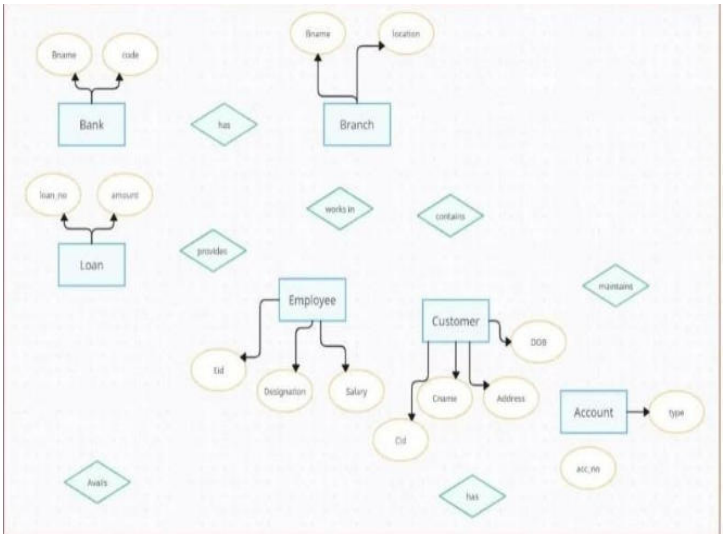
Let's Establishing Relationships between them are:

- The **Bank** has **branches**.
- The **Branch** provides **loan**.
- The **Employee** works in **branch**.
- The **Branch** contains **customers**.

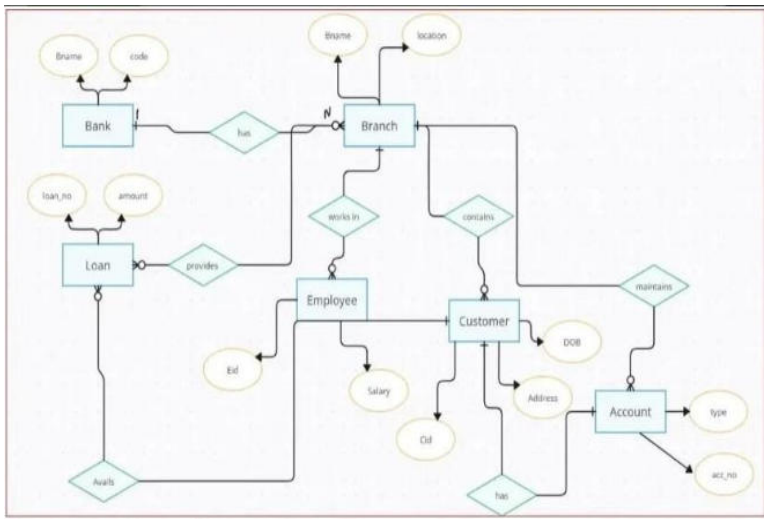
- The **Customers** has **account**.
- The **Branch** maintains **account**.
- The **Customer** avails **loan**.

Specify cardinality for Bank:

- **Bank** and **branch** has **One to Many** relationship (a bank has multiple branches).

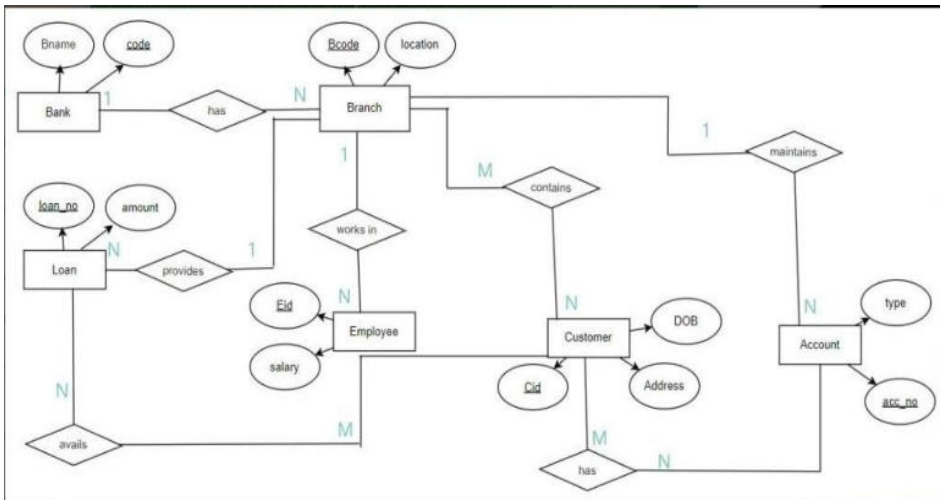


- **Branch** and **loan** has also **One to Many** relationship(a branch can provide many loans).
- **Branch** and **employee** has **One to Many** relationship(one branch has many employees).
- **Branch** and **account** has **One to Many** relationship(one branch has many accounts).
- **Branch** and **customer** has **Many to Many** relationship(multiple branches have multiple customers).
- **Customer** and **account** has **Many to Many** relationship(multiple customers have multiple accounts).
- **Customer** and **loan** has **Many to Many** relationships(multiple customers have multiple loans)



Final ER Diagram

The below diagram is our final entity relationship diagram for bank with all **entities**, their **attributes** and the relationship between them with the PRIMARY KEY and **Cardinality ratio**.

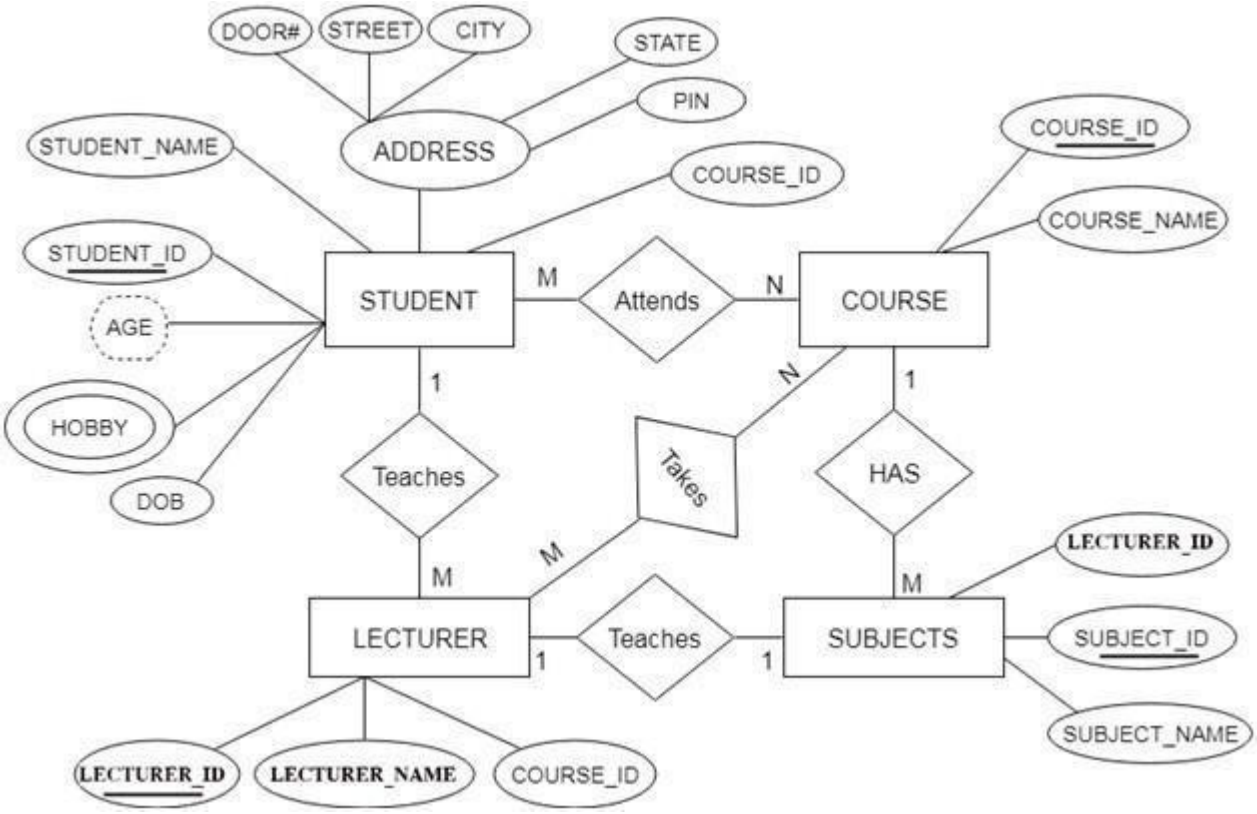


University diagram

The database can be represented using the notations, and these notations can be reduced to a collection of tables.

In the database, every entity set or relationship set can be represented in tabular form.

The ER diagram is given below:



ER Diagram Design Issues:

the various designing shapes that represent a relationship, an entity, and its attributes. However, users often mislead the concept of the elements and the design process of the ER diagram. Thus, it leads to a complex structure of the ER diagram and certain issues that does not meet the characteristics of the real-world enterprise model.

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

1) Use of Entity Set vs Attributes

The use of an entity set or attribute depends on the structure of the real-world enterprise that is being modeled and the semantics associated with its attributes. It leads to a mistake when the user use the primary key of an entity set as an attribute of another entity set. Instead, he should

use the relationship to do so. Also, the primary key attributes are implicit in the relationship set, but we designate it in the relationship sets.

2) Use of Entity Set vs. Relationship Sets

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

3) Use of Binary vs n-ary Relationship Sets

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

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

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

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

Extended ER Features

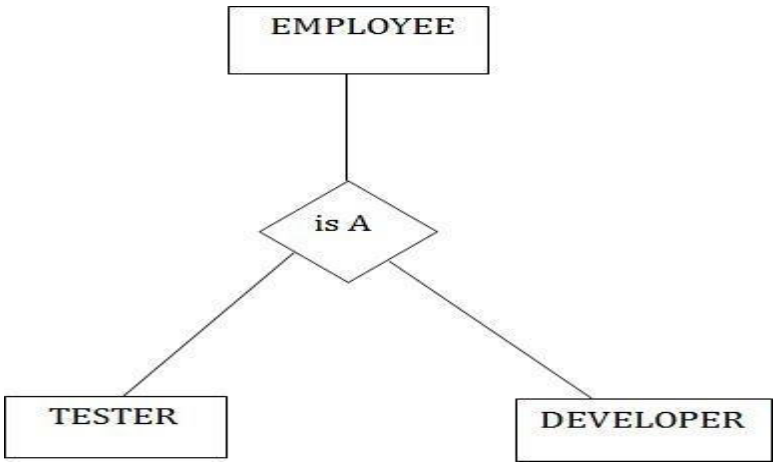
Specialization:

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.

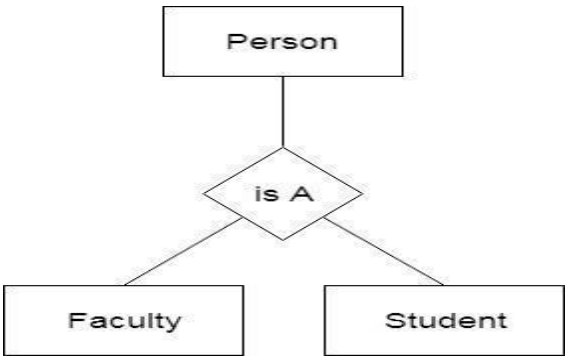
For example: In an Employee management system, EMPLOYEE entity can be specialized as TESTER or DEVELOPER based on what role they play in the company.



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.

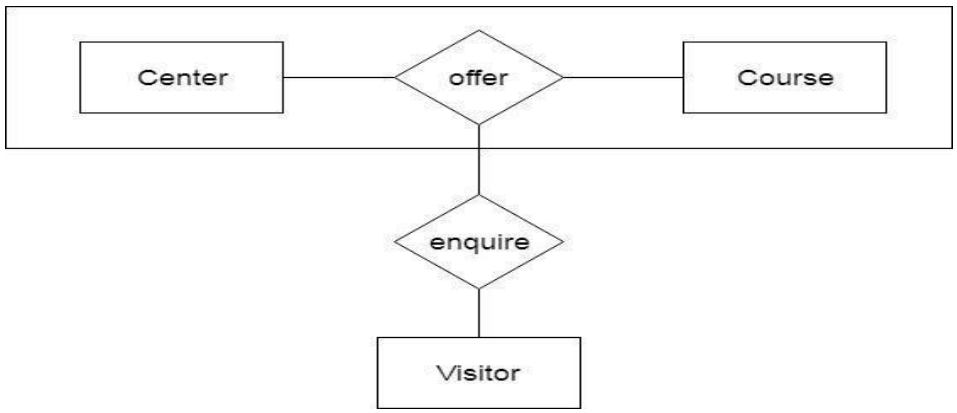
For example, Faculty and Student entities can be generalized and create a higher level entity Person.



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.



Attributes in DBMS

Attributes are properties or characteristics of an entity. Attributes are used to describe the entity. The attribute is nothing but a piece of data that gives more information about the entity. Attributes are used to distinguish one entity from the other entity. Attributes help to categorize the entity and the entity can be easily retrieved and manipulate the entity. Attributes can help the database to be more structural and hierarchical. An entity with no attribute is of no use in the database.

There are 8 types of attributes in DBMS.

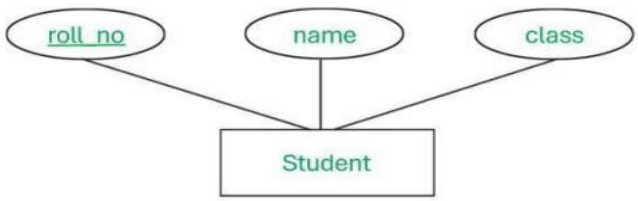
- **Simple Attribute.**
- **Composite Attribute.**
- **Single Valued Attribute.**
- **Multivalued Attribute.**
- **Key Attribute.**
- **Derived Attribute.**
- **Stored Attribute.**
- **Complex Attribute.**

Simple Attribute

- Simple attributes are those attributes that cannot be divided into more attributes. Simple attributes state the simple information about the entity such as name, roll_no, class, age, etc.
- Simple attributes are widely used for storing information about the entity.

Example

- Here in the below example, Student has roll_no, class, and name as attributes that cannot be divided into more sub-attributes.
- These types of attributes are called **simple attributes.**
- **Simple attributes are mainly used to create all other types of attributes.**

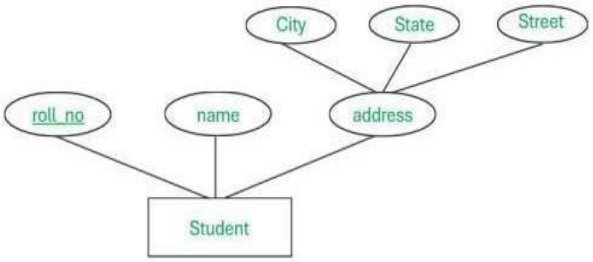


Composite Attribute

- When 2 or more than 2 simple attributes are combined to make an attribute then that attribute is called a **Composite attribute.**
- The composite attribute is made up of multiple attributes. After combining these attributes, the composed attributes are formed.
- Complex attributes are used where data is complex and needs to be stored in a complex structure.

Example

- Here if we look at the below example, address is the attribute derived from the 3 simple attributes i.e. City, State, and Street.
- To get the value of the address attribute, we have first to know those city, state, and street attributes.
- This type of attribute is known as a **composite attribute.**



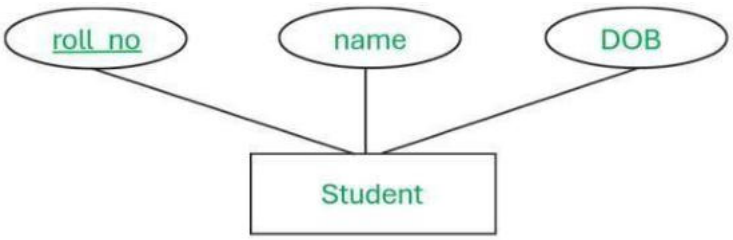
Single Valued Attribute

- The attribute with only a single value is known as a **single-valued attribute.** These attributes have a single value for each instance of a given entity.
- Mostly these attributes are used to provide the unique identity to the multiple instances of attributes.

Example

- In the given example, we know that the DOB attribute will have only one value. So we can say that the DOB attribute is nothing but a single Valed attribute and it cannot have multiple attributes.
- Here roll_no and name will also have mostly one value only.

- We can say that all 3 attributes of the student are **single-valued.**

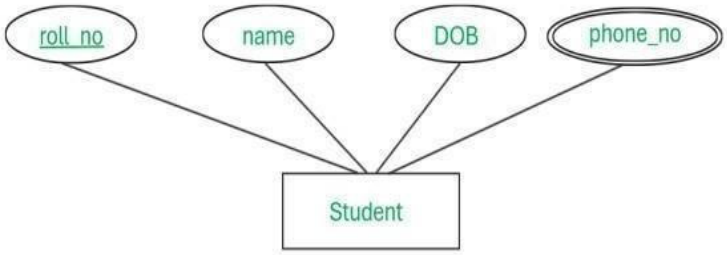


Multivalued Attribute

- An attribute which can have multiple values is known as a **multivalued attribute.** Multivalued attributes have multiple values for the single instance of an entity.
- Keu of entity is associated with multiple values. It does not have only one value. It is the opposite of the single-valued attribute.

Example

- Here the student has an attribute named phone_no. One student can have multiple phone_no, so we can say that phone_no can have multiple values.
- These types of attributes are known as **multi-valued attributes.**
- Multi-valued attributes are used when more than 1 entries for one attribute need to be stored in the Database.

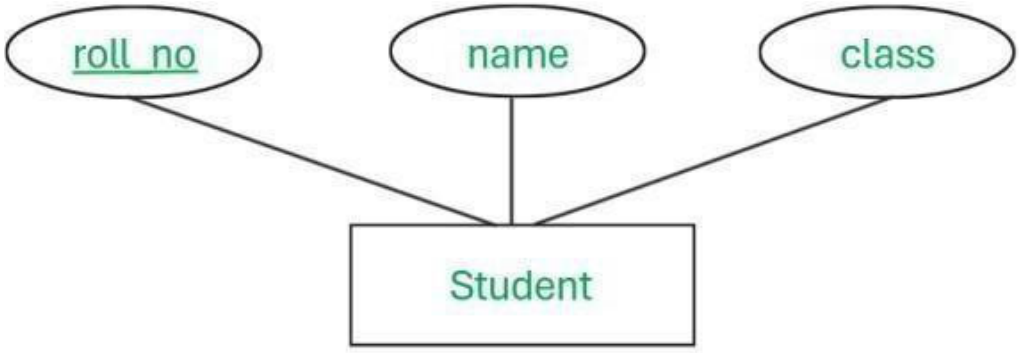


Key Attribute

- The attribute which has unique values for every row in the table is known as a **Key Attribute**. The key attribute has a very crucial role in the database.
- The key attribute is a distinct and unique characteristic of the entity that can be used to identify the entity uniquely.

Example

- For students, we can identify every student with roll_no because each student will have a unique roll_no.
- This indicates that roll_no will be a Key attribute for the Student entity.
- All operations on the database can be performed only using Key Attributes.

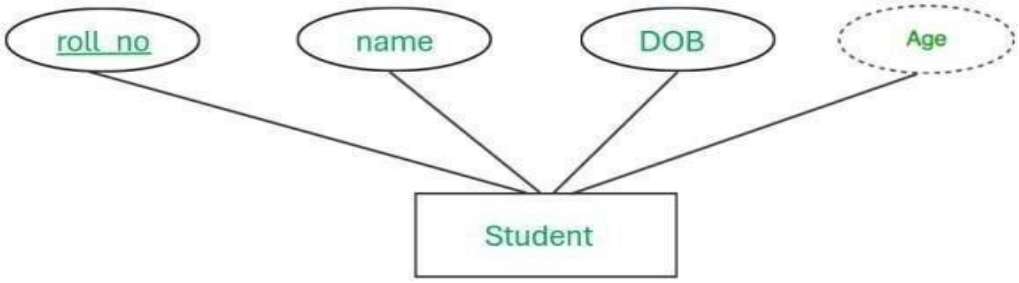


Derived Attribute

- The attribute that can be derived from the other attributes and does not require to be already present in the database is called a **Derived Attribute**.
- Derived attributes are not stored in the Database directly. It is calculated by using the stored attributes in the database.

Example

- Here the student has multiple attributes including DOB and age. It is observed that age can be calculated with the help of the DOB attribute.
- So age is a derived attribute that is derived from an attribute named DOB.

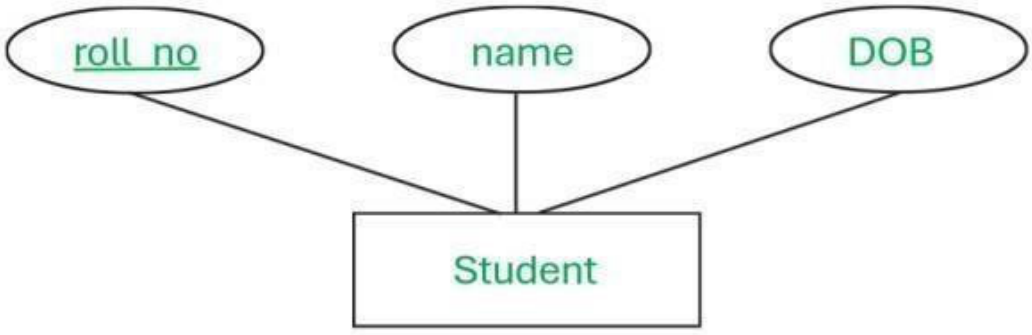


Stored Attribute

- If the data of the attribute remains constant for every instance of entity then it is called a **Stored Attribute.**
- The value of the attribute present in the database does not get updated and it remains constant once it is stored.
- These attributes are used to store permanent information about an entity which will remain constant throughout the lifetime of the entity.

Example

- The student has 3 attributes as shown above. Her name and DOB will remain the same throughout his/her education. So the student has a fixed value attribute that will never change in the future.
- These attributes are known as **stored attributes**.



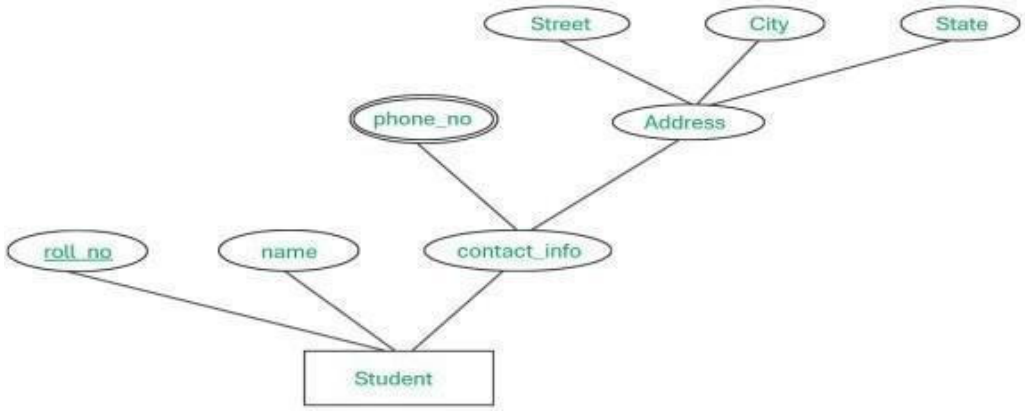
omplex Attribute

- When multi-valued and composite attributes together form an attribute then it is called a **Complex attribute.**
- Complex attributes can have an unlimited number of sub-attributes.

Example

- Here for the student, we created an attribute named contact_info which further decomposed into phone_no + Address.
- The address is a composite attribute which is further divided into simple attributes and phone_no is a multivalued attribute.

- This indicates that the contact_info attribute is made from the multi-valued and composite attribute.



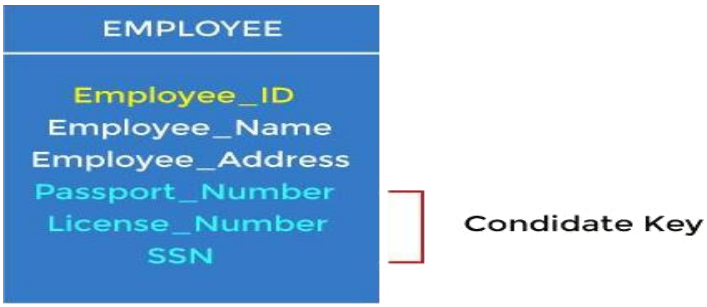
Types of Keys in Relational Model:

- Candidate Key
- Primary Key
- Super Key
- Alternate Key
- Foreign Key
- Composite Key

1. Candidate key

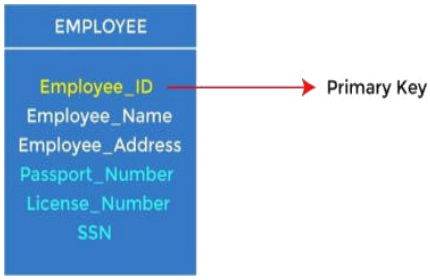
- A candidate key is an attribute or set of attributes that can uniquely identify a tuple.
- Except for the primary key, the remaining attributes are considered a candidate key. The candidate keys are as strong as the primary key.

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.



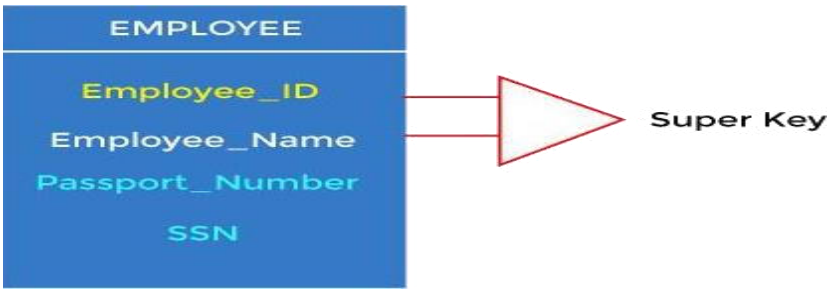
2. Primary key

- 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.



3. Super Key

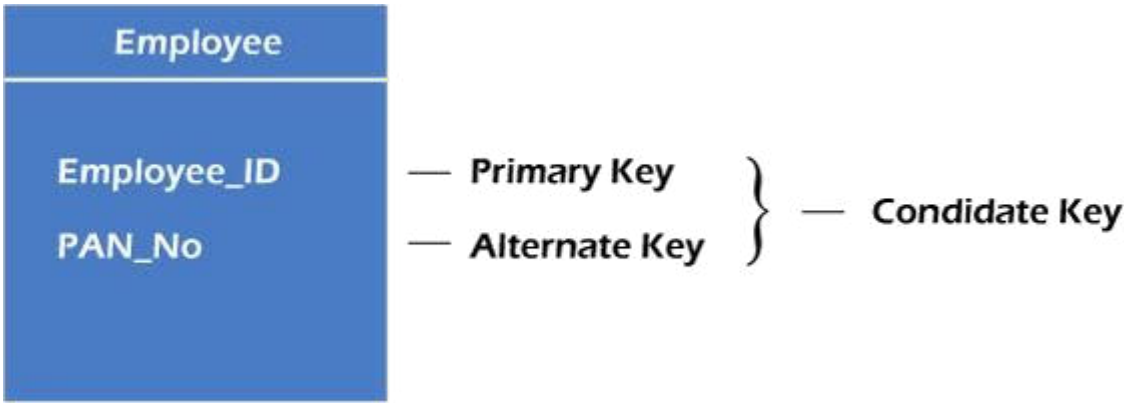
Super key is an attribute set that can uniquely identify a tuple. A super key is a superset of a candidate key.



4. 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

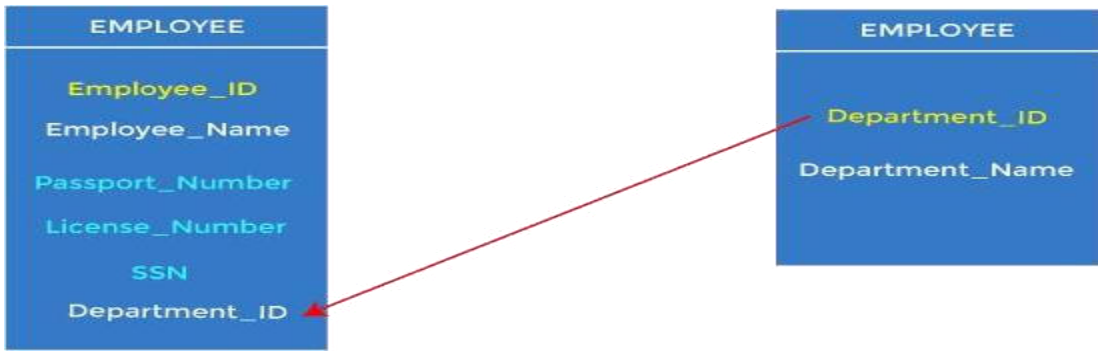
For example, employee relation has two attributes, Employee_Id and PAN_No, that act as candidate keys. In this relation, Employee_Id is chosen as the primary key, so the other candidate key, PAN_No, acts as the Alternate key.



5. Foreign key

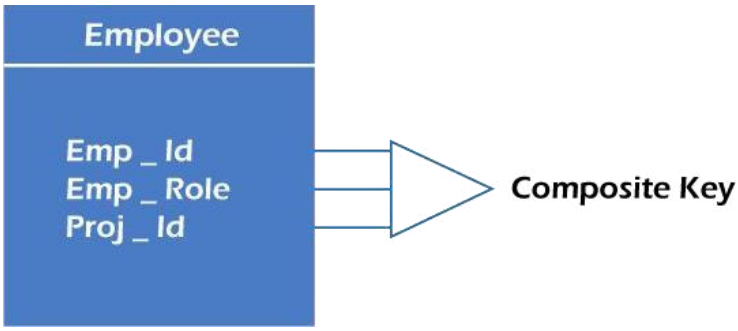
- Foreign keys are the column of the table used to point to the primary key of another table.
- Every employee works in a specific department in a company, and employee and department are two different entities. So we can't store the department's information in the employee table. That's why we link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



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.



Unit II: Relational Model

RELATIONAL MODEL: Structure of Relational database, Relational algebra, Tuple relational calculus, Domain relational calculus, QBE (Query-by-Example).

Relational model in DBMS

The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are also known as relations.

Structure of Relational model:

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

Tuple – It is nothing but a single row of a table, which contains a single record.

Relations- are in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.

Relation schema- A relational schema is the design for the table. It includes none of the actual data, but is like a blueprint or design for the table, so describes what columns are on the table and the data types. It may show basic table constraints.

Degree- (or arity) of a relation is the number of attributes n of its relation schema.

Domains: A domain is a set of values permitted for an attribute in a table. Domain is atomic. For example, age can only be a positive integer.

Cardinality: Total number of rows present in the Table.

Relation instance – Relation instance is a finite set of tuples at a given time. The current relationstate reflects only the valid tuples that represent a particular state of the real world.

Null value: A field with a NULL value is a field with no value. Primary key can‟t be a null value.

Relation Name: **STUDENT**

Attributes

Tuples

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

Relational Algebra

RELATIONAL ALGEBRA is a widely used procedural query language. It collects instances of relations as input and gives occurrences of relations as output. It uses various operations to perform this action. SQL Relational algebra query operations are performed recursively on a relation. The output of these operations is a new relation, which might be formed from one or more input relations.

Relational Algebra divided in various types :

Unary Relational Operations

- SELECT (symbol: σ)
- PROJECT (symbol: π)
- RENAME (symbol: ρ)

Set Operations

- **UNION (∪)**
- **INTERSECTION (∩),**
- **SET DIFFERENCE (-)**
- **CARTESIAN PRODUCT (x)**

Binary Relational Operations

- **JOIN**

Unary Relational Operations

SELECT (σ): The SELECT operation is used for selecting a subset of the tuples according to agiven selection condition. Sigma (σ) Symbol denotes it.

$$\sigma_{\text{condition}} \text{ (Relation)}$$

STUDENT

ID	NAME	MARK
1	Jisy	70
2	vishnu	75
3	Dwayne	80

1. To retrieve entire details from student

   σ (STUDENT)

2. To retrieve details from STUDENT where ID=2

$$\sigma_{ID=2}(STUDENT)$$

Projection (π): The project operation is used for selecting attributes according to a given selection condition.

$$\Pi_{Attribute}(Relation)$$

1. To retrieve ID and NAME from STUDENT.

$$\Pi_{ID, NAME}(STUDENT)$$

Combination of Select and Project:

2. Retrieve ID and Name from STUDENT where mark>=75

$$\Pi_{ID, NAME}(\sigma_{Mark>=75}(STUDENT))$$

Rename Operation (ρ):

. The rename operation allows us to rename the old t relation to new. „Rename" operation isdenoted with small Greek letter rho ρ.

$$\rho\ New\ (Old\ Relation\ Name)$$

Set Operations:

Union operation (υ): UNION is symbolized by ∪ symbol. It includes all tuples that are in tables Aor in B. It also eliminates duplicate tuples A ∪B.

For a union operation to be valid, the following conditions must hold -

- R and S must be the same number of attributes.
- Attribute domains need to be compatible.
- Duplicate tuples should be automatically removed.

Retrieve students name **either** participant in arts **or** sports:

Π NAME (ARTS) ∪ Π NAME (SPORTS)

ARTS

ID	NAME
1	A
2	B
3	C

SPORTS

ID	NAME
3	C
2	B
4	D

ARTS **U** SPORTS

NAME
A
B
C
D

Intersection: $A \cap B$ of two sets A and B is the set that contains all elements of A that also belongto B (or equivalently, all elements of B that also belong to A), but no other elements.

Retrieve students name those who participant in both arts and sports:

Query 1 : ARTS ∩SPORTS

ID	
2	B
3	C

Query 2 : Π NAME (ARTS) ∩ Π NAME (SPORTS)

NAME
B
C

Set Difference (−): The result of set difference operation is tuples, which are present in onerelation but are not in the second relation.

Retrieve students name those who participant **only** in arts **and not** in sports:

Query 1 : ARTS - SPORTS

ID	NAME
1	A

Retrieve students name those who participant **only** in sports **and not** in arts:

Query 2 : Π NAME (SPORTS) - Π NAME (ARTS)

NAME
D

Cartesian Product (X): Combines information of two different relations into one.

Table A

K	Y
1	A
2	B

Table B

K	Y
1	C
3	D

A **X** B

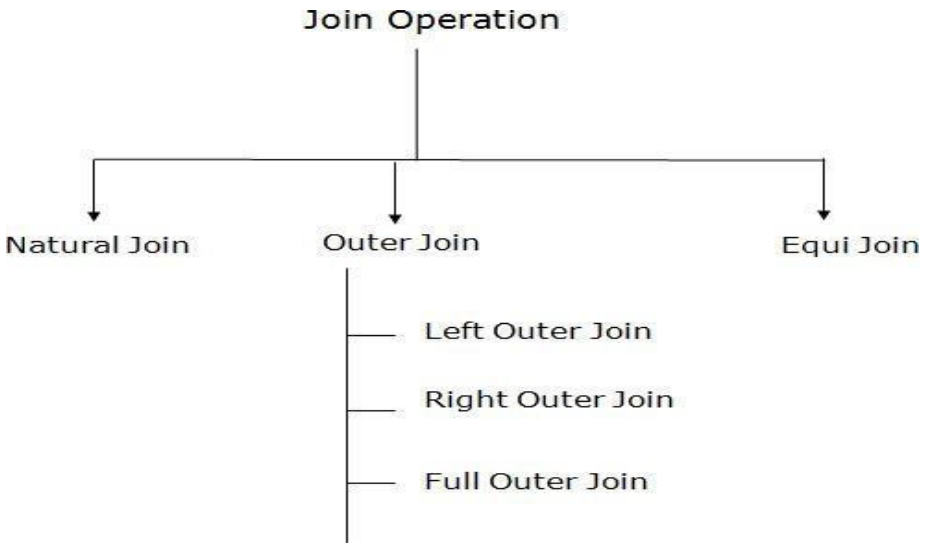
K	Y	K	Y
1	A	1	C
1	A	3	D
2	B	1	C
2	B	3	D

Join Operations

JOIN operation also allows joining variously related tuples from different relations.

Join operation denoted by ⋈.

Types of JOIN:



Natural Join:

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

Example:

Let‟s consider two relations:

Relation 1: Employees

emp_id	emp_name	dept_id
101	Alice	10
102	Bob	20
103	Charlie	10

Relation 2: Departments

dept_id	dept_name
10	HR
20	IT

Natural Join Example:

The **Natural Join** of **Employees** and **Departments** based on the common attribute `dept_id` will result in:

Employees ⋈ DepartmentS

The resulting relation will combine tuples where `dept_id` matches in both relations. The result is:

emp_id	emp_name	dept_id	dept_name
101	Alice	10	HR
102	Bob	20	IT
103	Charlie	10	HR

Outer Join

The Join operation in relational algebra can be extended to include rows from one or both relations that do not have matching values in the other relation. This is known as an outer join. When necessary, it fills non-matching tuples with null values and includes them in the result. There are three types of **Outer Joins**:

1. **Left Outer Join** ( )
2. **Right Outer Join** (⋈)
3. **Full Outer Join** ( )

Example:

Let‟s use the following two relations:

Relation 1: Employees

emp_id	emp_name	dept_id
101	Alice	10
102	Bob	20
103	Charlie	30
104	David	40

Relation 2: Departments

dept_id	dept_name
10	**HR**
20	**IT**
30	**Finance**

1. Left Outer Join ( )

Includes all rows from the **left relation** (R) and the matching rows from the **right relation** (S).

If there is no match, the missing side (right relation) will have **null** values.

We will perform a **Left Outer Join** between **Employees** and **Departments** on dept_id:

Employees Departments Result:

emp_id	emp_name	dept_id	dept_name
101	Alice	10	HR
102	Bob	20	IT
103	Charlie	30	Finance
104	David	40	NULL

2. Right Outer Join (⋈)

Includes all rows from the **right relation** (S) and the matching rows from the **left relation** (R).
If there is no match, the missing side (left relation) will have **null** values.

Now, we will perform a **Right Outer Join** between **Employees** and **Departments**:

Employees ⋈ Departments

Result:

emp_id	emp_name	dept_id	dept_name
101	**Alice**	**10**	**HR**
102	**Bob**	**20**	**IT**
103	**Charlie**	**30**	**Finance**
NULL	**NULL**	**40**	**NULL**

Full Outer Join ( ):

Includes all rows from both relations. If there is no match on either side, **null** values are used for the missing side.

Finally, we perform a **Full Outer Join** between **Employees** and **Departments**:

Employees Departments

Result:

emp_id	emp_name	dept_id	dept_name
101	**Alice**	**10**	**HR**
102	**Bob**	**20**	**IT**
103	**Charlie**	**30**	**Finance**
104	**David**	**40**	**NULL**
NULL	**NULL**	**40**	**NULL**

Equi Join

An **Equi Join** in relational algebra is a type of **join** operation where two relations are combined based on an **equality condition**. Specifically, an **Equi Join** involves matching rows from two relations where the values in the common attribute(s) are equal.

Equi Join Syntax:

The general form of an **Equi Join** operation between two relations R and S on a common attribute A is:

$$R \bowtie R.A = S.A \quad S$$

Where:

- R and S are two relations (tables).
- A is the common attribute (column) that exists in both relations.
- $\bowtie R.A=S.A$ is the join condition, indicating that the tuples will be joined where the values of R.A are equal to S.A

Equi Join Query in Relational Algebra:

Employees ⋈Employees.deptid=Departments.deptid Departmentss

Result:

emp_id	emp_name	dept_id	dept_name
101	Alice	10	HR
102	Bob	20	IT
103	Charlie	10	HR
104	David	30	Finance

.

Tuple Relational Calculus (TRC) in DBMS

Tuple Relational Calculus is a **non-procedural query language** unlike relational algebra. Tuple Calculus provides only the description of the query but it does not provide the methods to solve it. Thus, it explains what to do but not how to do.

In Tuple Calculus, a query is expressed as

$\{t| P(t)\}$

where t = resulting tuples,

P(t) = known as Predicate and these are the conditions that are used to fetch t

Thus, it generates set of all tuples t, such that Predicate P(t) is true for t.

P(t) may have various conditions logically combined with OR (∨), AND (∧), NOT(¬).

It also uses quantifiers:

∃ t ∈ r (Q(t)) = ”there exists” a tuple in t in relation r such that predicate Q(t) is true.

∀ t ∈ r (Q(t)) = Q(t) is true “for all” tuples in relation r.

Example:

Consider a database with a relation (table) named `Students`, which has the following schema:

Students(StudentID, Name, Age, Major)

Assume the table contains the following data:

StudentID	Name	Age	Major
1	Alice	20	CS
2	Bob	22	Math
3	Carol	21	CS

StudentID	Name	Age	Major
4	Dave	23	Physics

Query Example

Let's say we want to retrieve the names of all students who are majoring in "CS". The TRC expression would look like this:

Query 1: $\{ t.Name \mid t \in Students \wedge t.Major = 'CS' \}$

Result

For the above query, the result would be:

Name
Alice
Carol

Domain Relational Calculus

This is a non-procedural query language equivalent in power to Tuple Relational Calculus. Domain Relational Calculus provides only the description of the query but it does not provide the methods to solve it. In Domain Relational Calculus, a query is expressed as,

{ < x1, x2, x3, ..., xn > | P (x1, x2, x3, ..., xn) }

where, < x1, x2, x3, …, xn > represents resulting domains variables and P (x1, x2, x3, …, xn) represents the condition or formula equivalent to the Predicate calculus.

Predicate Calculus Formula:

1. Set of all comparison operators
2. Set of connectives like and, or, not
3. Set of quantifiers (Universal Quantifier (∀),Existential Quantifier (∃))

Syntax of DRC:

{ <attributes> | P(x1, x2, ..., xn) }

Where:

- <attributes> are the output attributes.
- P(x1, x2, ..., xn) is a predicate (a condition) that describes the relationships between the variables and the data.

Example

Example 1: Find the loan number, branch, and amount of loans greater than or equal to 10000 amount.

Schema:

- **Loan**(LoanNumber, Branch, Amount)

Sample Data:

LoanNumber	Branch	Amount
L001	Downtown	5000
L002	Uptown	12000
L003	Suburban	15000
L004	Central	8000
L005	Uptown	20000

Query :

To find the `LoanNumber`, `Branch`, and `Amount` of loans with an amount greater than or equal to 10,000, we select rows where the `Amount` is at least 10,000.

Result:

LoanNumber	Branch	Amount
L002	Uptown	12000
L003	Suburban	15000
L005	Uptown	20000

Differences betweem tuple and domain relation calculus

S. No.	Basis of Comparison	Tuple Relational Calculus (TRC)	Domain Relational Calculus (DRC)
1.	**Definition**	The Tuple Relational Calculus (TRC) is used to select tuples from a relation. The tuples with specific range values, tuples with certain attribute values, and so on can be selected.	The Domain Relational Calculus (DRC) employs a list of attributes from which to choose based on the condition. It‟s similar to TRC, but instead of selecting entire tuples, it selects attributes.
2.	**Representation of variables**	In TRC, the variables represent the tuples from specified relations.	In DRC, the variables represent the value drawn from a specified domain.
3.	**Tuple/ Domain**	A tuple is a single element of relation. In database terms, it is a row.	A domain is equivalent to column data type and any constraints on the value of data.
4.	**Filtering**	This filtering variable uses a tuple of relations.	This filtering is done based on the domain of attributes.
5.	**Return Value**	The predicate expression condition associated with the TRC is used to test every row using a tuple variable and return those tuples that met the condition.	DRC takes advantage of domain variables and, based on the condition set, returns the required attribute or column that satisfies the criteria of the condition.
5.	**Membership**	The query cannot be	The query can be expressed

S. No.	Basis of Comparison	Tuple Relational Calculus (TRC)	Domain Relational Calculus (DRC)
	condition	expressed using a membership condition.	using a membership condition.
6.	**Query Language**	The QUEL or Query Language is a query language related to it,	The QBE or Query-By-Example is query language related to it.
7.	**Similarity**	It reflects traditional pre-relational file structures.	It is more similar to logic as a modeling language.
8.	**Syntax**	Notation: {T | P (T)} or {T | Condition (T)}	Notation: { a1, a2, a3, …, an | P (a1, a2, a3, …, an)}
9.	**Example**	{T | EMPLOYEE (T) AND T.DEPT_ID = 10}	{ | < EMPLOYEE > DEPT_ID = 10 }

QBE(Querry By Example):

If we talk about normal queries we fire on the database they should be correct and in a well-defined structure which means they should follow a proper syntax if the syntax or query is wrong definitely we will get an error and due to that our application or calculation definitely going to stop. So to overcome this problem QBE was introduced. QBE stands for **Query By Example** and it was developed in 1970 by Moshe Zloof at IBM.

It is a graphical query language where we get a user interface and then we fill some required fields to get our proper result.

In SQL we will get an error if the query is not correct but in the case of QBE if the query is wrong either we get a wrong answer or the query will not be going to execute but we will never get any error.

Note-:

In QBE we don‟t write complete queries like SQL or other database languages it comes with some blank so we need to just fill that blanks and we will get our required result.

Example

Consider the example where a table „SAC‟ is present in the database with Name, Phone_Number, and Branch fields. And we want to get the name of the SAC-Representative name who belongs to the MCA Branch. If we write this query in SQL we have to write it like

```
SELECT NAME
FROM SAC
WHERE BRANCH = 'MCA'"
```

And definitely, we will get our correct result. But in the case of QBE, it may be done as like there is a field present and we just need to fill it with “MCA” and then click on the SEARCH button we will get our required result.

Points to be:

- Supported by most of the database programs.
- It is a Graphical Query Language.
- Created in parallel to SQL development.

UNIT-III

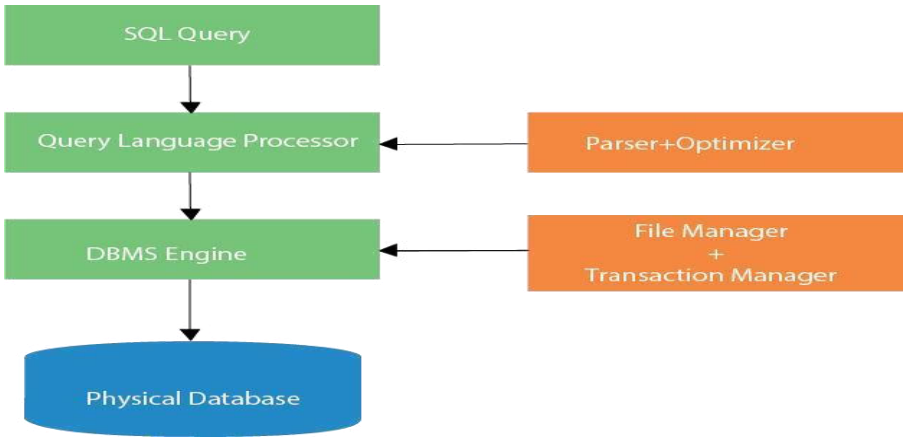
STRUCTURED QUERY LANGUAGE (SQL): Introduction to SQL, SQL Operators, SQL Join queries, Sub queries, Nested queries, Views, Integrity constraints, Functional Dependenc Database design Normalization: Normal Forms-1st, 2nd, 3rd and BCNF, Multi - Valued Depe Normal Form, 5th NF/Projection Join Normal form and De- Normalization.

Topic1: Introduction to SQL:

- QL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a number of ways, using English-like statements.
- Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line. Using the SQL statements, you can perform most of the actions in a database. SQL depends on tuple relational calculus and relational algebra.

SQL process:

- When an SQL command is executing for any RDBMS, then the system figure out the best way to carry out the request and the SQL engine determines that how to interpret the task.
- In the process, various components are included. These components can be optimization Engine, Query engine, Query dispatcher, classic, etc.
- All the non-SQL queries are handled by the classic query engine, but SQL query engine won't handle logical files.



Topic 2: SQL Operator

Every database administrator and user uses SQL queries for manipulating and accessing the data of database tables and views.

The manipulation and retrieving of the data are performed with the help of reserved words and characters, which are used to perform arithmetic operations, logical operations, comparison operations, compound operations, etc.

Types of Operator

SQL operators are categorized in the following categories:

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

Arithmetic Operators

We can use various arithmetic operators on the data stored in the tables. Arithmetic Operators are:

Operator	Description
+	The addition is used to perform an addition operation on the data values.
–	This operator is used for the subtraction of the data values.
/	This operator works with the 'ALL' keyword and it calculates division operations.
*	This operator is used for multiplying data values.
%	Modulus is used to get the remainder when data is divided by another.

Comparison Operators

Another important operator in SQL is a comparison operator, which is used to compare one expression's value to other expressions. SQL supports different types of comparison operator, which is described below:

Operator	Description
=	Equal to.
>	Greater than.
<	Less than.
>=	Greater than equal to.
<=	Less than equal to.
<>	Not equal to.

Logical Operators

The Logical operators are those that are true or false. They return true or false values to combine one or more true or false values.

Operator	Description
AND	Logical AND compares two Booleans as expressions and returns true when both expressions are true.
OR	Logical OR compares two Booleans as expressions and returns true when one of the expressions is true.
NOT	Not takes a single Boolean as an argument and change its value from false to true or from true to false.

Special Operators

Operators	Description
ALL	ALL is used to select all records of a SELECT STATEMENT. It compares a value to every value in a list of results from a query. The ALL must be preceded by the comparison operators and evaluated to TRUE if the query returns no rows.
ANY	ANY compares a value to each value in a list of results from a query and evaluates to true if the result of an inner query contains at least one row.
BETWEEN	The SQL BETWEEN operator tests an expression against a range. The range consists of a beginning, followed by an AND keyword and an end expression.
IN	The IN operator checks a value within a set of values separated by commas and retrieves the rows from the table that match.

SET Operations

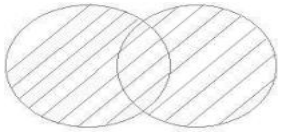
SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions.

In this tutorial, we will cover 4 different types of SET operations, along with example:

1. **UNION**
2. **UNION ALL**
3. **INTERSECT**
4. **MINUS**

UNION Operation

UNION is used to combine the results of two or more SELECT statements. However it will eliminate duplicate rows from its result set. In case of union, number of columns and data type must be same in both the tables, on which UNION operation is being applied.



Example:

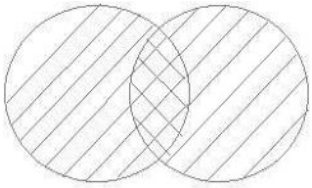
SELECT * FROM First table

UNION

SELECT * FROM Second table;

UNION ALL

This operation is similar to Union. But it also shows the duplicate rows.



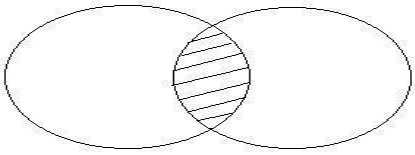
SELECT * FROM First

UNION ALL

SELECT * FROM Second;

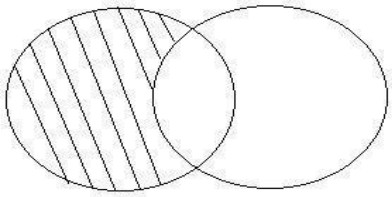
INTERSECT

Intersect operation is used to combine two SELECT statements, but it only retunes the records which are common from both SELECT statements. In case of **Intersect** the number of columns and data type must be same.



MINUS

The Minus operation combines results of two SELECT statements and return only those in the final result, which belongs to the first set of the result.



Topic 3: SQL FUNCTION

SQL, or **Structured Query Language**, is a programming language used for managing and manipulating relational databases. One of the most powerful features of SQL is the ability to use functions to perform various operations on the data in a database.

Types of SQL functions

- Aggregate Function
- Number Function
- Character Function
- Conversion Function
- Date Function

1. 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.

1. COUNT

2. SUM

3. AVG

4. MAX

5. MIN

1. COUNT ( ):

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.

Syntax: COUNT(*)

Example: SELECT COUNT(*) FROM employee;

Output: COUNT(*)

21

2. SUM ( )

Sum function is used to calculate the sum of all selected columns.

Syntax: SUM ( )

Example: SELECT sum(salary) FROM employee;

Output: SUM (salary)

65440

3. AVG ( )

The AVG function is used to calculate the average value of the numeric type.

Syntax: AVG( )

Example: SELECT AVG(salary) FROM employee;

Output:

AVG(salary)

3116.1904

4. MAX ( )

MAX function is used to find the maximum value of a certain column.

Syntax: MAX( )

Example: SELECT MAX(salary) FROM employee;

Output:

MAX (salary)

7000

5. MIN ( )

MIN function is used to find the minimum value of a certain column.

Syntax: MIN ( )

Example: SELECT MIN (salary) FROM employee;

Output:

MIN (salary)

500

QUERIES USING CONVERSION FUNCTIONS:

The DUAL table is a special one-row, one-column table present by default in Oracle and other database installations. This is used to perform mathematical calculations without using a table.

Example: select * from dual;

"DUMMY"

"X"

Numeric Functions:

Numeric Functions are used to perform operations on numbers and return numbers.

Following are the numeric functions defined in SQL:

ABS( ): It returns the absolute value of a number.

Syntax: ABS(number)

Example: SELECT ABS(-12) FROM DUAL;

Output:

ABS (-12)

12

CEIL( ): It returns the smallest integer value that is greater than or equal to a number.

Syntax: CEIL(NUMBER)

Example: SELECT CEIL(25.75) FROM DUAL;

Output:

CEIL (25.75)

26

FLOOR( ): It returns the largest integer value that is less than or equal to a number.

Syntax: FLOOR(NUMBER)

Example: SELECT FLOOR(25.75) FROM DUAL;

Output:

FLOOR (25.75)

25

MOD( ): It returns the remainder (aka. modulus) of n divided by m.

Syntax: MOD(m, n)

Example: SELECT MOD(10,2) FROM DUAL;

Output:

MOD(10,2)

0

ROUND( ): It returns a number rounded to a certain number of decimal places.

Syntax: ROUND(NUMBER,[DECIMAL PLACES])

Example: SELECT ROUND(15.253, 1) FROM DUAL;

Output:

ROUND (15.253, 1)

15.3

SIN( ): It returns the sine of a number in radians.

Syntax: SELECT SIN(2)FROM DUAL;

Example: SELECT SIN(2)FROM DUAL;

Output:

SIN(2)

0.909297

COS( ): It returns the cosine of a number, in radians.

Syntax: SELECT COS(30);

Example: SELECT COS(30) FROM DUAL;

Output:

COS(30)

0.154251

TAN( ): It returns the tangent of a number in radians.

Syntax: SELECT TAN(1.75);

Example: SELECT TAN(1.75) FROM DUAL;

Output:

TAN(1.75)

-5.52037992250933

SQRT( ): It returns the square root of a number.

Syntax: SELECT SQRT(25);

Example: SELECT SQRT(25) FROM DUAL;

Output:
SQRT(25)

5

POWER( ):

This numeric function is used to return the power of a given expression.

Syntax: POWER(m, n)

Example: SELECT POWER(2, 3) FROM DUAL;

Output:

POWER(2, 3)

8

CHARACTER FUNCTIONS:

SQL provides a rich set of character functions that allow you to get information about strings and modify the contents of those strings in multiple ways.

1. ASCII

ASCII is a String function of Oracle. This function returns the numeric value of given character.

Syntax

ASCII(**character**)

Example: SELECT **ASCII('J')** FROM **DUAL;**

Output:

"ASCII('J')"
---

2. CHR

This function returns all the characters of the given ASCII code.

Syntax: **CHR(number)**

Example: SELECT **CHR(69)** FROM **DUAL;**

Output:

"CHR(69)"

E

3. **CONCAT :** This function always appends (concatenates) string2 to the end of string1.

Syntax:

CONCAT('String1', 'String2')

Example: SELECT CONCAT('computer' ,'science') FROM DUAL;

Output:

"CONCAT('computer' ,'science') "

---

computerscience

4. **INITCAP:** This function converts alpha character values to uppercase for the first letter of each word and all others in lowercase.

Syntax:

INITCAP('computer science')

Example: SELECT INITCAP('computer science') FROM DUAL;

Output:

INITCAP('computer science')

Computer Science

5. **LOWER( ):** This function converts alpha character values to lowercase.

Syntax:

LOWER(SQL course)

Example: SELECT LOWER('GEEKSFORGEEKS') FROM DUAL;

Output:

LOWER('GEEKSFORGEEKS')

Geeksforgeeks

6. UPPER( ): This function converts alpha character values to uppercase.

Syntax: UPPER(SQL)

Example: SELECT UPPER('geeksforgeeks') FROM DUAL;

Output:

UPPER('geeksforgeeks')

GEEKSFORGEEKS

7. **LPAD( ):** function is used to padding the left side of a string with a specific set of characters.

Syntax: LPAD(string1, padded_length)

Example: Select LPAD ('ORACLE', 4) from dual;

Output:

LPAD ('ORACLE')

ORAC

8. RPAD( )

This function returns the right-padded to the given length.

Syntax:

RPAD(string1, padded_length)

Example: SELECT **RPAD('ORACLE',2)** FROM **dual;**

Output:

RPAD('ORACLE',2)

OR

9. RTRIM( )

This function returns the string by removing given characters from the right side.

Syntax: RTRIM(string1 [, trim_string])

Parameters

Example: SELECT RTRIM ('javatpoint', 'tpoint') FROM dual;

Output:
RTRIM ('javatpoint', 'tpoint')

Java

10. LTRIM( )

This function returns the string by removing given characters from left side.

Syntax:

LTRIM(string1 [, trim_string])

Example: SELECT LTRIM ('JAVA','J') FROM DUAL;

OUTPUT:

LTRIM ('JAVA','J')

AVA

Conversion Function:

1. TO_CHAR

TO_CHAR function is used to typecast a numeric or date input to a character type with a format model (optional).

Syntax

TO_CHAR(date, 'format')

Example: select TO_CHAR(sysdate,'year') from dual;

OUTPUT:
TO_CHAR(sysdate,'year')
---
twenty twenty-four

2. TO_DATE

This function takes character values and returns the output in the date format.

syntax: TO_DATE(number, format)

Example: select TO_DATE('090623','mm dd yy') from dual;

OUTPUT:

TO_DATE('090623','mm dd yy')

06-09-23

DATE/TIME FUNCTION:

1. SYSDATE( ) Function

This function is used to get the current date of the system.

Syntax:

SYSDATE

Example:

SELECT SYSDATE FROM dual;

Output:

SYSDATE

08-01-24

(OR)

CURRENT_DATE ( )

It is used to get the current date.

Syntax:

CURRENT_DATE

Example:

SELECT CURRENT_DATE FROM DUAL;

Output:

SYSDATE

08-01-24

2. SYSTIMESTAMP ( )

This function is used to get the current date and time of the system.

Syntax:

SYSTIMESTAMP

Example:

SELECT SYSTIMESTAMP FROM dual;

Output:

SYSTIMESTAMP

08-01-24 9:57:18.607000000 PM +05:30

3. ADD_MONTHS ( )

It is used to get the date in which some specified months are added.

Syntax:

ADD_MONTHS(date1, number_months)

Example:

SELECT ADD_MONTHS ('20-NOV-2018', 2) FROM DUAL;

Output:

ADD_MONTHS ('20-NOV-2018', 2)

20-01-19

4. MONTHS_BETWEEN ( )

It is used to get the difference between given two dates, i.e., date1 and date2.

Syntax:

MONTHS_BETWEEN(date1, date2)

Example:

SELECT MONTHS_BETWEEN('05-JAN-23','05-JAN-22') FROM DUAL;

OUTPUT:

MONTHS_BETWEEN('05-JAN-23','05-JAN-22')

12

5. NEXT_DAY ( )

This function returns the first weekday from the given date and week name.

Syntax

NEXT_DAY(date, weekday)

Example

SELECT NEXT_DAY('08-JAN-24', 'MONDAY') FROM DUAL;

OUTPUT:

NEXT_DAY('08-JAN-24', 'MONDAY')

15-01-24

6. LAST_DAY ( )

It is used to get the last date of the given month in the date.

Syntax:

LAST_DAY(date)

Example:

SELECT LAST_DAY(SYSDATE) FROM DUAL;

OUTPUT:

LAST_DAY(SYSDATE)

31-01-24

Topic : 4 SQL JOIN

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

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

Types of SQL JOIN

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

1. INNER JOIN

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

Syntax

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

FROM table1

INNER JOIN table2

ON table1.matching_column = table2.matching_column;

Query

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

2. LEFT JOIN

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

Syntax

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

Query

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

3. RIGHT JOIN

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

Syntax

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

Query

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

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

4. FULL JOIN

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

Syntax

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

Query

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

Topic 5: SQl Views

A view is a virtual table that is created from an existing table. The CREATE VIEW statement is used to create views. Views can be constructed from a single table or multiple tables. Similar to a real table, a view consists of rows and columns jest like real table. SQL functions, WHERE clauses and JOIN statements can be added to a view. A view does not store data in the database.

There are two types of views:

1. Simple View: These views can only contain a single base table or can be created only from one table. Group functions such as MAX(), COUNT(), etc., cannot be used here. By using Simple View, DML operations can be performed. Insert, delete, and update are directly possible

2. Complex View: The view is created on multiple tables then it is called as complex view.

1. Simple View:

Creating a view from a single table

Syntax:

```
CREATE VIEW view_name
AS
SELECT *.
FROM  table_name
WHERE [condition]; (optional)
```

Example:

```
SQL>  CREATE VIEW studentview
        AS
        SELECT * FROM Student;
Sql> SELECT * FROM studentview;
```

Inserting a row into a view

Syntax:

```
INSERT view_name VALUES (value1, value2, value3..);
```

Example:

```
SQL> insert into studentview values(7,'Anu','Bangalore');
```

Deleting a row from a view

Syntax:

```
DELETE  FROM  view_name  WHERE condition;
```

Example:

```
SQL> DELETE FROM studentview WHERE sid = 5;
```

Updating Views:

Syntax:

```
UPDATE view_name
SET column1 = value1,....
WHERE [condition];
```

Example:

```
UPDATE studentview
  SET address = 'hyd'
  WHERE sid = 1;
```

Dropping A View In DBMS:

Syntax:

drop view tablename;

Example : drop view studentview;

2. Complex view: Creating a view from multiple tables

Updating a View

A view can be updated with the CREATE OR REPLACE VIEW statement.

Syntax:

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1,coulmn2,..
FROM table_name
WHERE condition;
```

Example:

```
CREATE OR REPLACE VIEW MarksView
AS
SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS,
StudentMarks. AGE
FROM StudentDetails, StudentMarks
WHERE StudentDetails.NAME = StudentMarks.NAME;
```

Dropping a View

A view is deleted with the DROP VIEW statement.

Syntax:

```
DROP VIEW view_name;
```

Example: DROP VIEW MarksView;

Topic 6: SQL Constraints

1. NOT NULL:

The NOT NULL constraint doesn't accept null values but it accepts duplicate values. The NOT NULL constraint is only allowed at the column level.

Example:

```
CREATE TABLE student1 (
                        Sno NUMBER (3) NOT NULL,
                        Name CHAR(10)
                    );
```

2. **UNIQUE:** A unique constraint doesn't allow duplicate values in a column but they accept null values. UNIQUE constraint applied only on a particular column.

Example:

```
CREATE TABLE student2 (
                        sno number(3) unique,
                        name char(10)
                    );
```

3. **CHECK:** Specifies a condition that each row in the table must satisfy. To satisfy the constraint, each row in the table must make the condition either TRUE or FALSE.

Example:

```
CREATE TABLE student3 (
                        StudentID INT,
                        Name VARCHAR(30),
                        Age INT,
                        Check(Age >= 17)
                    );
```

4. DEFAULT :

The DEFAULT constraint is used to insert a default value into a column. The default value will be added to all new records, if no other value is specified.

Example:

```
Sql> CREATE TABLE student5 (
                    id INT PRIMARY KEY,
                    country VARCHAR(20) DEFAULT 'mca'
                );
```

PRIMARY KEY:

In SQL, the PRIMARY KEY constraint is used to uniquely identify rows. It is combinations of NOT NULL and UNIQUE constraints i.e. it cannot contain duplicate or NULL values.

Example:

```
CREATE TABLE student7 (
                    id int,
                    college_name varchar(50),
                    primary key (id)
                );
```

FOREIGN KEY:

The FOREIGN KEY constraint is used to create a relationship between two tables. A foreign key is defined using the FOREIGN KEY and REFERENCES keywords. The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Ex:

```
CREATE TABLE DEPT11(
                    DEPTNO NUMBER,
                    DNAME VARCHAR2(100),
                     PRIMARY KEY (DEPTNO) );
```

```
CREATE TABLE EMP11(
                                    EMPID NUMBER PRIMARY KEY,
                                    ENAME VARCHAR2(100),
                                    DEPTNO NUMBER,
                            FOREIGN KEY (DEPTNO) REFERENCES DEPT11(DEPTNO)
                              );
```

Topic 7: SUB QUERIES

SQL Sub queries:

An SQL **Sub query**, is a SELECT query within another query. It is also known as **Inner query** or **Nested query** and the query containing it is the outer query.

The outer query can contain the SELECT, INSERT, UPDATE, and DELETE statements. We can use the sub query as a column expression, as a condition in SQL clauses, and with operators like =, >, <, >=, <=, IN, BETWEEN, etc.

Following are the rules to be followed while writing sub queries –

- subqueries must be enclosed within parentheses.
- Subqueries can be nested within another subquery.
- A subquery must contain the SELECT query and the FROM clause always.
- A subquery consists of all the clauses an ordinary SELECT clause can contain: GROUP BY, WHERE, HAVING, DISTINCT, TOP/LIMIT, etc
- A subquery can return a single value, a single row, a single column, or a whole table.

The basic syntax is as follows –

```
SELECT column_name [, column_name ]
FROM table1 [, table2 ]
WHERE  column_name
OPERATOR (SELECT column_name [,column_name ] FROM table1 [, table2 ] [WHERE]);
```

Now, let us check the following subquery with a SELECT statement.

ex;

SELECT * FROM CUSTOMERS

WHERE ID IN (SELECT ID FROM CUSTOMERS WHERE SALARY > 4500);

This would produce the following result −

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
7	Muffy	24	Indore	10000.00

Subqueries with the INSERT Statement:

We can also use the subqueries along with the INSERT statements. The data returned by the subquery is inserted into another table.

Now to copy the complete records of CUSTOMERS table into the CUSTOMERS_BKP table, we can use the following query −

The basic syntax is as follows −

```
INSERT INTO table_name [ (column1 [, column2 ]) ]
   SELECT [ *|column1 [, column2 ] FROM table1 [, table2 ]
   [ WHERE VALUE OPERATOR ]
```

Ex:

```
INSERT INTO CUSTOMERS_BKP
SELECT * FROM CUSTOMERS
WHERE ID IN (SELECT ID FROM CUSTOMERS);
```

Subqueries with the UPDATE Statement:

A subquery can also be used with the UPDATE statement. You can update single or multiple columns in a table using a subquery.

The basic syntax is as follows −

UPDATE table

SET column_name = new_value

[WHERE OPERATOR [VALUE](SELECT COLUMN_NAME FROM TABLE_NAME [WHERE]);

Example:

UPDATE CUSTOMERS

SET SALARY = SALARY * 0.25

WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP WHERE AGE >= 27);

TOPIC 8: NESTED QUERY

A **nested query** is a query that has another query embedded within it. The embedded query is called a subquery.

A subquery typically appears within the **WHERE clause** of a query. It can sometimes appear in the FROM clause or **HAVING clause**.

Subqueries can be used in various sections of an SQL statement, such as:

- WHERE clause
- FROM clause
- SELECT clause

When a subquery is used in a WHERE clause, it is often to filter the results based on a comparison to an expression or column value from the main query. For example, you can use a subquery to find all products whose price is above average by comparing the price column with the average value generated by the subquery.

Example: SELECT product_name, priceFROM productsWHERE price > (SELECT AVG(price) FROM products);

TOPIC 9: FUNCTIONAL DEPENDENCY

A functional dependency is a constraint that specifies the relationship between two sets of attributes where one set can accurately determine the value of other sets. It is denoted as **X → Y**, where X is a set of attributes that is capable of determining the value of Y. The attribute set on the left side of the arrow, **X** is called **Determinant**, while on the right side, **Y** is called the **Dependent**. Functional dependencies are used to mathematically express relations among database entities.

Types of Functional dependencies in DBMS:

1. Trivial functional dependency
2. Non-Trivial functional dependency
3. Multivalued functional dependency
4. Transitive functional dependency

1. Trivial Functional Dependency

In **Trivial Functional Dependency**, a dependent is always a subset of the determinant. i.e. If **X → Y** and **Y is the subset of X**, then it is called trivial functional dependency

For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

Here, **{roll_no, name} → name** is a trivial functional dependency, since the dependent **name** is a subset of determinant set **{roll_no, name}** Similarly, **roll_no → roll_no** is also an example of trivial functional dependency.

2. Non-trivial Functional Dependency

In **Non-trivial functional dependency**, the dependent is strictly not a subset of the determinant.

i.e. If **X → Y** and **Y is not a subset of X**, then it is called Non-trivial functional dependency.

For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

Here, **roll_no → name** is a non-trivial functional dependency, since the dependent **name** is **not a subset of** determinant **roll_no.**

Similarly, **{roll_no, name} → age** is also a non-trivial functional dependency, since **age** is **not a subset of {roll_no, name}**

3. Multivalued Functional Dependency

In **Multivalued functional dependency**, entities of the dependent set are **not dependent on each other.**

i.e. If **a → {b, c}** and there exists **no functional dependency** between **b and c**, then it is called a **multivalued functional dependency.**

For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18
45	abc	19

Here, **roll_no → {name, age}** is a multivalued functional dependency, since the dependents **name** & **age** are **not dependent** on each other(i.e. **name → age** or **age → name doesn't exist !**)

4. Transitive Functional Dependency

In transitive functional dependency, dependent is indirectly dependent on determinant. i.e. If **a → b** & **b → c**, then according to axiom of transitivity, **a → c**. This is a **transitive functional dependency**

For example,

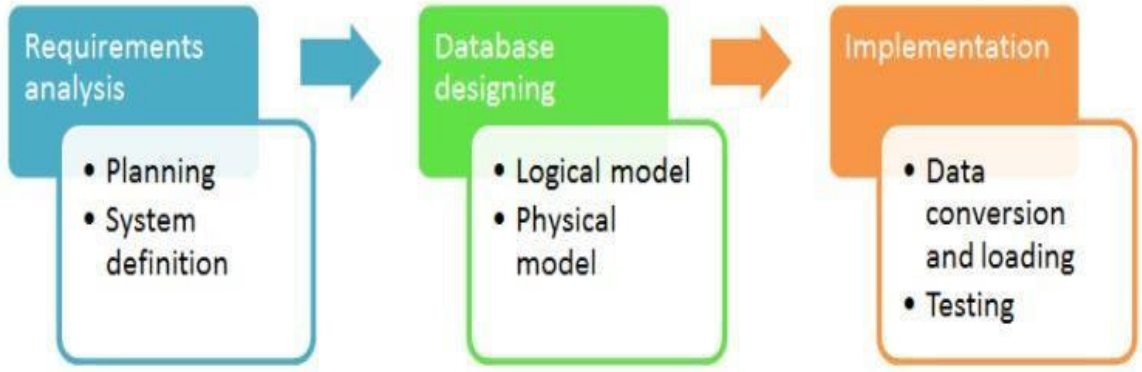
enrol_no	name	dept	building_no
42	abc	CO	4
43	pqr	EC	2
44	xyz	IT	1
45	abc	EC	2

Here, **enrol_no → dept** and **dept → building_no**,

Hence, according to the axiom of transitivity, **enrol_no → building_no** is a valid functional dependency. This is an indirect functional dependency, hence called Transitive functional dependency.

Topic 10: Database Design:

Database designs provide the blueprints of how the data is going to be stored in a system. A proper design of a database highly affects the overall performance of any application. The designing principles defined for a database give a clear idea of the behavior of any application and how the requests are processed. Another instance to emphasize the database design is that a proper database design meets all the requirements of users. Lastly, the processing time of an application is greatly reduced if the constraints of designing a highly efficient database are properly implemented.



Although, the life cycle of a database is not an important discussion that has to be taken forward in this article because we are focused on the database design. But, before jumping directly on the designing models constituting database design it is important to understand the overall workflow and life-cycle of the database.

Requirement Analysis

First of all, the planning has to be done on what are the basic requirements of the project under which the design of the database has to be taken forward. Thus, they can be defined as:-

Planning - This stage is concerned with planning the entire DDLC (Database Development Life Cycle). The strategic considerations are taken into account before proceeding.

System definition - This stage covers the boundaries and scopes of the proper database after planning.

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.

Topic 11: What is Normalization?

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

Why do we need Normalization?

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

Data modification anomalies can be categorized into three types:

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

NORMAL FORMS

Forms of normalization are given below:

1. First Normal Form (1NF)
2. Second Normal Form (2NF)
3. Third Normal Form (3NF)
4. Boyce-Codd Normal Form (BCNF)
5. Fourth Normal Form (4NF)
6. Fifth Normal Form (5NF)

1.FIRST NORMAL FORM (1NF):

"A relation schema R is in 1NF, if it does not have any **composite** attributes, **multivalued** attribute or their combination."

The objective of first normal form is that the table should contain no repeating groups of data.Data is divided into logical units called entities or tables
All attributes (column) in the entity (table) must be single valued.

Repeating or multi valued attributes are moved into a separate entity (table) & a relationship is established between the two tables or entities.

Example of the first normalform:
Consider the following **Customer** table.
Customer:

cid	name	address		contact_no
		society	city	
C01	aaa	Amul avas,Anand		{1234567988}
C02	bbb	near parimal garden,abad		{123,333,4445}
C03	ccc	sardar colony , surat		

Here, address is a composite attribute , which is further subdivided into two column society and city. And attribute contact_no is multivalued attribute.

Problems with this relation are -

- It is not possible to store multiple values in a single field in a relation. So, if any customer has more than one contact number, it is not possible to store those numbers.
- Another problem is related to information retrieval. Suppose, here, if there is a need to find out all customers belonging to some particular city, it is very difficult to retrieve. The reason is: city name for all customers are combined with society names and stored whole as an address.

Solution for composite attribute

Insert separate attributes in a relation for each sub-attribute of a composite attribute.

In our example, insert two separate attributes for Society and city in a relation in place of single composite attributes address. Now, insert data values separately for Society and City for all tuples.

Customer :

cid	name	Society	city	contact_no
C01	aaa	Amul avas	Anand	{1234567988}
C02	bbb	near parimal garden	abad	{123,333,4445}
C03	ccc	sardar colony	surat	

Solution for Multi-valued attribute

Two approaches are available to solve problem of multi-valued attribute

1. First Approach:

In a First approach, determine maximum allowable values for a multi-valued attribute. In our case, if maximum two numbers are allowed to store, insert two separate attributes attributes to store contact numbers as shown.

Customer:

cid	name	Society	city	contact_no1	contact_no2	contact_no3
C01	aaa	Amul avas	Anand	1234567988		
C02	bbb	near parimal garden	abad	123	333	4445
C03	ccc	sardar colony	surat			

Now,if customer has only one contact number or no any contact number, then keep the related field empty for tupple of that customer. If customer has two contact numbers, store both number in related fields. If customer has more than two contact numbers, store two numbers and ignore all other numbers.

2. Second Approach:

In a second approach, remove the multi-valued attribute that violates 1NF and place it in a separate relation along with the primary key of given original relation. The primary key of new relation is the combination of multi-valued attribute and primary key of old relation. for example, in our case, remove the contact_no attribute and place it with cid in a separate relation customer_contact. Primary Key for relation Customer_contact will be combination of cid and contact_no.

customer:

cid	name	Society	city
C01	aaa	Amul avas	Anand
C02	bbb	near parimal garden	abad
C03	ccc	sardar colony	surat

Customer_contact

cid	contact_no
C01	1234567988
C02	123
C02	333
C02	4445

First approach is simple. But, it is not always possible to put restriction on maximum allowable values. It also introduces null values for mant fields.

Second approach is superior as it does not suffer from draw backs of first approach. But, it is somewhat complicated one. For example, to display all information about any/all customers, two relations - Customer and Customer_contact - need to be accessed.

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:

{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

4. Boyce-Codd Normal Form (BCNF)

Boyce–Codd Normal Form (BCNF) is based on functional dependencies that take into account all candidate keys in a relation; however, BCNF also has additional constraints compared with the general definition of 3NF.

Rules for BCNF

Rule 1: *The table should be in the 3rd Normal Form.*

Rule 2: *X should be a superkey for every functional dependency (FD) X−>Y in a given relation.*

Note: To test whether a relation is in BCNF, we identify all the determinants and make sure that they are candidate keys.

Examples

Here, we are going to discuss some basic examples which let you understand the properties of BCNF. We will discuss multiple examples here.

Example 1

Let us consider the student database, in which data of the student are mentioned.

Stu_ID	Stu_Branch	Stu_Course	Branch_Number	Stu_Course_No
101	Computer Science & Engineering	DBMS	B_001	201
101	Computer Science & Engineering	Computer Networks	B_001	202
102	Electronics & Communication Engineering	VLSI Technology	B_003	401
102	Electronics & Communication Engineering	Mobile Communication	B_003	402

Functional Dependency of the above is as mentioned:

Stu_ID −> Stu_Branch

Stu_Course −> {Branch_Number, Stu_Course_No}

Candidate Keys of the above table are: **{Stu_ID, Stu_Course}**

Why this Table is Not in BCNF?

The table present above is not in BCNF, because as we can see that neither Stu_ID nor Stu_Course is a Super Key. As the rules mentioned above clearly tell that for a table to be in BCNF, it must follow the property that for functional dependency X−>Y, X must be in Super Key and here this property fails, that's why this table is not in BCNF.

How to Satisfy BCNF?

For satisfying this table in BCNF, we have to decompose it into further tables. Here is the full procedure through which we transform this table into BCNF. Let us first divide this main table into two tables **Stu_Branch** and **Stu_Course** Table.

Stu_Branch Table

Stu_ID	Stu_Branch
101	Computer Science & Engineering
102	Electronics & Communication Engineering

Candidate Key for this table: **Stu_ID**.

Stu_Course Table

Stu_Course	Branch_Number	Stu_Course_No
DBMS	B_001	201
Computer Networks	B_001	202
VLSI Technology	B_003	401
Mobile Communication	B_003	402

Candidate Key for this table: **Stu_Course**.

Stu_ID to Stu_Course_No Table

Stu_ID	Stu_Course_No
101	201
101	202
102	401
102	402

Candidate Key for this table: **{Stu_ID, Stu_Course_No}.**

After decomposing into further tables, now it is in BCNF, as it is passing the condition of Super Key, that in functional dependency X−>Y, X is a **Super Key.**

5. Fourth Normal Form (4NF)

The Fourth Normal Form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key. It builds on the first three normal forms (1NF, 2NF, and 3NF) and the Boyce-Codd Normal Form (BCNF). It states that, in addition to a database meeting the requirements of BCNF, it must not contain more than one multivalued dependency.

Properties

A relation R is in 4NF if and only if the following conditions are satisfied:

1. It should be in the Boyce-Codd Normal Form (BCNF).
2. The table should not have any Multi-valued Dependency.

A table with a multivalued dependency violates the normalization standard of the Fourth Normal Form (4NF) because it creates unnecessary redundancies and can contribute to

inconsistent data. To bring this up to 4NF, it is necessary to break this information into two tables.

Example: Consider the database table of a class that has two relations R1 contains student ID(SID) and student name (SNAME) and R2 contains course id(CID) and course name (CNAME).

Table R1

SID	SNAME
S1	A
S2	B

Table R2

CID	CN AME
C1	C
C2	D

When their cross-product is done it resulted in multivalued dependencies.

Table R1 X R2

SID	SNAME	CID	CNAME
S1	A	C1	C
S1	A	C2	D
S2	B	C1	C
S2	B	C2	D

Multivalued dependencies (MVD) are:

```
SID->->CID; SID->->CNAME; SNAME->->CNAME
```

Fifth Normal Form/Projected Normal Form (5NF)

A relation R is in Fifth Normal Form if and only if everyone joins dependency in R is implied by the candidate keys of R. A relation decomposed into two relations must have lossless join Property, which ensures that no spurious or extra tuples are generated when relations are reunited through a natural join.

Properties

A relation R is in 5NF if and only if it satisfies the following conditions:

1. R should be already in 4NF.
2. It cannot be further non loss decomposed (join dependency).

Example – Consider the above schema, with a case as "if a company makes a product and an agent is an agent for that company, then he always sells that product for the company". Under these circumstances, the ACP table is shown as:

Table ACP

Agent	Company	Product
A1	PQR	Nut
A1	PQR	Bolt
A1	XYZ	Nut
A1	XYZ	Bolt
A2	PQR	Nut

The relation ACP is again decomposed into 3 relations. Now, the natural Join of all three relations will be shown as:

Table R1

Agent	Company
A1	PQR
A1	XYZ
A2	PQR

Table R2

Agent	Product
A1	Nut
A1	Bolt
A2	Nut

Table R3

Company	Product
PQR	Nut
PQR	Bolt

Company	Product
XYZ	Nut
XYZ	Bolt

The result of the Natural Join of R1 and R3 over ‘Company’ and then the Natural Join of R13 and R2 over ‘Agent’and ‘Product’ will be **Table ACP**.

Hence, in this example, all the redundancies are eliminated, and the decomposition of ACP is a lossless join decomposition. Therefore, the relation is in 5NF as it does not violate the property of lossless join.

UNIT-IV

PL/SQL: Introduction to PL/SQL, PL/SQL Block Structure, Conditional statements, Iterative Processing with Loops, Triggers, Cursor, exception handling, Procedures, Functions.

FILE STRUCTURE: File Organization, Organization of Records in Files and Data-Dictionary Storage. Indexing and Hashing: Ordered Indices, B+-Tree Index Files, B-Tree Index files, Static and Dynamic Hashing.

Introduction to PL/SQL

PL/SQL (Procedural Language/Structured Query Language) is an extension of SQL (Structured Query Language) designed by Oracle. It enables developers to write code that combines the power of SQL queries with the procedural capabilities of a programming language. PL/SQL is mainly used to interact with Oracle databases, providing a way to execute SQL statements in a procedural manner, which allows for more control over data processing, including logic flow and error handling.

Structure of PL/SQL in DBMS

PL/SQL programs can be structured into different blocks. The basic structure of a PL/SQL block is as follows:

1. Declaration Section (optional):

This section is used to declare variables, constants, cursors, and exceptions that will be used in the PL/SQL block.

```
DECLARE
   variable_name datatype;
   CONSTANT_NAME CONSTANT datatype := value;
BEGIN
```

2. Execution Section (mandatory):

This is the section where the actual SQL statements and procedural code are written. It is the core of the PL/SQL block and is mandatory. SQL commands like SELECT, INSERT, UPDATE, or DELETE are executed here, along with procedural logic like loops and conditions.

Syntax:

```
BEGIN
   -- SQL and PL/SQL code
END;
```

3. Exception Handling Section (optional):

- This section is used to handle runtime errors or exceptions that occur during the execution of the PL/SQL block. It allows programmers to gracefully handle errors instead of having the program crash or stop.

Syntax:

```
EXCEPTION
   WHEN exception_name THEN
      -- Handle error
   WHEN OTHERS THEN
      -- Handle all other errors
END;
```

Example of a Simple PL/SQL Block:

```
DECLARE
   v_emp_name VARCHAR2(50);
   v_emp_salary NUMBER(10,2);
BEGIN
   SELECT employee_name, salary INTO v_emp_name, v_emp_salary
   FROM employees
   WHERE employee_id = 100;

   DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_emp_name);
   DBMS_OUTPUT.PUT_LINE('Salary: ' || v_emp_salary);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('No employee found with the given ID.');
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('An unexpected error occurred.');
END;
```

Topic 1:

Decision making statements

Or

Conditional statements

Or

Control statements:

Decision making or conditional or Control statements are those statements which are in charge of executing a statement out of multiple given statements based on some condition. The condition will return either true or false. Based on what the condition returns, the associated statement is executed.

Conditional Statements available in PL/SQL are defined below:

1. **IF THEN**
2. **IF THEN ELSE**
3. **NESTED-IF-THEN**
4. **IF THEN ELSIF-THEN-ELSE**

1. IF THEN

if then the statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

Syntax:

```
IF condition THEN
   -- Code to execute if the condition is true
END IF;
```

Example:

```
DECLARE
   v_salary NUMBER := 3000;
BEGIN
   IF v_salary > 2500 THEN
      DBMS_OUTPUT.PUT_LINE('Salary is above 2500');
   END IF;
END;
```

2. IF THEN ELSE

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else if the condition is false. Here comes the else statement. We can use the else statement with if statement to execute a block of code when the condition is false.

Syntax:

```
IF condition THEN
   -- Code to execute if the condition is true
ELSE
   -- Code to execute if the condition is false
END IF;
```

Example:

```
DECLARE
   v_age NUMBER := 20;
BEGIN
   IF v_age >= 18 THEN
      DBMS_OUTPUT.PUT_LINE('Adult');
   ELSE
      DBMS_OUTPUT.PUT_LINE('Minor');
   END IF;
END;
```

3. NESTED-IF-THEN

A **nested IF-THEN** statement occurs when an IF-THEN statement is placed inside another IF-THEN statement.

Syntax:

```
IF condition1 THEN
   -- Code to execute if condition1 is true
   IF condition2 THEN
      -- Code to execute if condition2 is true
   END IF;
END IF;
```

Here, the second IF statement (the nested one) will only be evaluated if the first IF statement is true.

Example:

```
declare
num1 number:= 10;
num2 number:= 20;
num3 number:= 20;

begin
if num1 < num2 then
dbms_output.put_line('num1 small num2');
if num1 < num3 then
  dbms_output.put_line('num1 small num3 also');
  end if;

end if;

dbms_output.put_line('after end if');
end;
```

4. if...then...elsif...else statement

It is used to check multiple conditions. Sometimes it is required to test more than one condition in that case if...then...else statement cannot be used. For this purpose, if...then...elsif...else statement is suitable in which all the conditions are tested one by one and whichever condition is found to be **TRUE**, that block of code is executed. And if all the conditions result in **FALSE** then the else part is executed.

Syntax:

```
if <test_condition1> then
        body of action
elsif <test_condition2>then
        body of action
elsif<test_condition3>then
        body of action
...
...
```

...

else

body of action

end if;

set serveroutput on;

Example:

```
DECLARE
        a int;
        b int;
BEGIN
        a := &a;
        b := &b;
        if(a>b) then
                dbms_output.put_line('a is greater than b');
        elsif(b>a) then
                dbms_output.put_line('b is greater than a');
        else
                dbms_output.put_line('Both a and b are equal');
        end if;
END;
```

Topic 2:

PL/SQL Loops:

Loops Statements (or) Iterative Statements

Loops in PL/SQL provides a way of repeating a particular part of any program or any code statement as many times as required. In PL/SQL we have three different loop options to choose from when we want to execute a statement repeatedly in our code block.

They are:

1. Basic Loop
2. While Loop
3. For Loop

1. Basic Loop

Basic loop or simple loop is preferred in PL/SQL code when there is no surety about how many times the block of code is to be repeated. When we use the basic loop the code block will be executed at least once.

While using it, following two things must be considered:

- Simple loop always begins with the keyword LOOP and ends with a keyword END LOOP.
- A basic/simple loop can be terminated at any given point by using the exit statement or by specifying certain condition by using the statement exit when.

Syntax:

```
LOOP
  -- Statements to be executed repeatedly
  EXIT WHEN condition; -- Optional exit condition
END LOOP;
```

Example:

```
DECLARE
  counter NUMBER := 1;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE('Counter: ' || counter);
    counter := counter + 1;
    EXIT WHEN counter > 5;
  END LOOP;
END;
```

/

2. While Loop

It is an entry controlled loop which means that before entering in a while loop first the condition is tested, if the condition is **TRUE** the statement or a group of statements get executed and if the condition is **FALSE** the control will move out of the while loop.

Syntax:

```
WHILE <test_condition> LOOP
        <action>
END LOOP;
```

Example:

```
set serveroutput on;
DECLARE
        num int:=1;
BEGIN
        while(num <= 10) LOOP
                dbms_output.put_line('|| no);
                num := num+2;
        END LOOP;
END;
```

3. For Loop

This loop is used when some statements in PL/SQL code block are to be repeated for a fixed number of times.

When we use the for loop we are supposed to define a counter variable which decides how many time the loop will be executed based on a starting and ending value provided at the beginning of the loop. The for loop automatically increments the value of the counter variable by 1 at the end of each loop cycle.

Syntax:

```
FOR counter_variable IN start_value..end_value LOOP
        statement to be executed
END LOOP;
```

Example:

```
set serveroutput on;
DECLARE
        i number(2);
```

```
BEGIN
        FOR i IN 1..10 LOOP
                dbms_output.put_line(i);
        END LOOP;
END;
```

Topic 3:

Triggers in PL/SQL

A **trigger** in PL/SQL is a type of stored procedure that automatically executes (or "fires") in response to a specific event occurring in the database. Triggers are used to enforce business rules, validate data, maintain audit trails, or automatically modify data in a table based on changes made to that table.

Triggers are defined to execute either **before** or **after** a specific event on a database object such as an **INSERT**, **UPDATE**, or **DELETE** operation. The event can be a change in a table or view, and triggers are bound to that table or view.

Types of Triggers

1. BEFORE Trigger
2. AFTER Trigger
3. INSTEAD OF Trigger
4. Level Triggers

There are 2 different types of level triggers, they are:

1. ROW LEVEL TRIGGERS
2. STATEMENT LEVEL TRIGGERS

Trigger Syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name
   {BEFORE | AFTER | INSTEAD OF}
   {INSERT | UPDATE | DELETE}
   ON table_name
   [FOR EACH ROW]
   DECLARE
      -- Variable declarations (optional)
   BEGIN
      -- Trigger code
```

END;

1. BEFORE Trigger: This trigger fires before the triggering event (INSERT, UPDATE, DELETE) is executed on the table.

Syntax:

```
CREATE OR REPLACE TRIGGER trigger_name
BEFORE {INSERT | UPDATE | DELETE}
ON table_name
[FOR EACH ROW]
DECLARE
   -- Variable declarations (optional)
BEGIN
   -- Trigger code
END;
```

Example:

```
CREATE OR REPLACE TRIGGER before_employee_insert
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
   -- Set the hire_date to the current system date before the new record is inserted
   :NEW.hire_date := SYSDATE;
END;
```

2. AFTER trigger in PL/SQL is a type of trigger that executes **after** a specified event (such as INSERT, UPDATE, or DELETE) has been performed on a table. This trigger is typically used for actions that need to occur after the data has been modified, such as logging changes, auditing data, or updating related tables.

Syntax:

```
CREATE OR REPLACE TRIGGER trigger_name
AFTER {INSERT | UPDATE | DELETE}
ON table_name
[FOR EACH ROW]
DECLARE
   -- Variable declarations (optional)
BEGIN
   -- Trigger code
```

```
END;
```

Example:

```
CREATE OR REPLACE TRIGGER after_employee_insert
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
   -- Insert a record into the audit_log table after a new employee is inserted
   INSERT INTO audit_log (log_id, action_type, employee_id, action_date)
   VALUES (audit_log_seq.NEXTVAL, 'INSERT', :NEW.employee_id, SYSDATE);
END;
```

3. INSTEAD OF Trigger in PL/SQL

An **INSTEAD OF trigger** in PL/SQL is used to define the behavior that should occur **instead of** the usual DML (Data Manipulation Language) operation such as INSERT, UPDATE, or DELETE. This type of trigger is typically used on **views** rather than tables. The purpose of an INSTEAD OF trigger is to replace the default action with a custom action when an operation is performed on a view.

Syntax:

```
CREATE OR REPLACE TRIGGER trigger_name
INSTEAD OF {INSERT | UPDATE | DELETE}
ON view_name
[FOR EACH ROW]
DECLARE
   -- Variable declarations (optional)
BEGIN
   -- Trigger code
END;
```

Example:

```
CREATE OR REPLACE TRIGGER update_employee_department
INSTEAD OF UPDATE ON employee_department_view
FOR EACH ROW
BEGIN
   -- Update the employees table
```

```
    UPDATE employees
    SET employee_name = :NEW.employee_name
    WHERE employee_id = :OLD.employee_id;

    -- Update the departments table
    UPDATE departments
    SET department_name = :NEW.department_name
    WHERE department_id = :OLD.department_id;
END;
```

4. Level Triggers in PL/SQL

Triggers are special procedures in PL/SQL that automatically execute in response to certain events on a particular table or view. Based on the scope of their operation, triggers are categorized into **Row-Level Triggers** and **Statement-Level Triggers**.

1. Row-Level Triggers

- A **Row-Level Trigger** is executed once for each row affected by a DML (INSERT, UPDATE, DELETE) statement.
- They are used when you need to work with or track changes to specific rows in a table.

Example of a Row-Level Trigger:

```
CREATE OR REPLACE TRIGGER row_level_trigger_example
AFTER INSERT OR UPDATE ON employees
FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('Row-Level Trigger fired for Employee ID: '
|| :NEW.employee_id);
END;
/
```

2. Statement-Level Triggers

- A **Statement-Level Trigger** is executed once for the entire DML statement, regardless of the number of rows affected.
- These triggers are used when the operation affects the table as a whole, not specific rows.

Example of a Statement-Level Trigger:

```
CREATE OR REPLACE TRIGGER statement_level_trigger_example
```

```
AFTER DELETE ON employees
BEGIN
   DBMS_OUTPUT.PUT_LINE('Statement-Level Trigger fired after a DELETE operation on employees.');
END;
/
```

Topic 4:

Cursor in PL/SQL

A **cursor** in PL/SQL is a pointer to a context area that stores the result set of a query. It allows row-by-row processing of query results.

Cursor Attributes

PL/SQL provides cursor attributes for obtaining information about the cursor state:

- **%FOUND**: Returns TRUE if a row was fetched, otherwise FALSE.
- **%NOTFOUND**: Returns TRUE if no row was fetched, otherwise FALSE.
- **%ISOPEN**: Returns TRUE if the cursor is open, otherwise FALSE.
- **%ROWCOUNT**: Returns the number of rows fetched so far.

PL/SQL provides two types of cursors:

1. **Implicit Cursors**
2. **Explicit Cursors**

1. Implicit Cursor

- Automatically created by PL/SQL for SELECT INTO, INSERT, UPDATE, and DELETE statements that affect a single row or multiple rows.
- You don't need to declare or control it.

Example of Implicit Cursor:

```
DECLARE
   emp_count NUMBER;
BEGIN
   SELECT COUNT(*) INTO emp_count FROM employees;
   DBMS_OUTPUT.PUT_LINE('Total Employees: ' || emp_count);
END;
/
```

2. Explicit Cursor

- Defined and controlled explicitly by the programmer.
- Useful for queries that return multiple rows.

- Consists of four main steps:
    1. **Declaration**: Declare the cursor.
    2. **Opening**: Open the cursor to execute the query and establish the result set.
    3. **Fetching**: Retrieve rows from the result set one by one.
    4. **Closing**: Close the cursor to release resources.

Example of Explicit Cursor:

```
DECLARE
   CURSOR emp_cursor IS
      SELECT employee_id, name FROM employees;
   v_emp_id employees.employee_id%TYPE;
   v_name employees.name%TYPE;
BEGIN
   -- Open the cursor
   OPEN emp_cursor;
   -- Loop through rows
   LOOP
      FETCH emp_cursor INTO v_emp_id, v_name;
      -- Exit the loop when no more rows are available
      EXIT WHEN emp_cursor%NOTFOUND;
      -- Process the fetched data
      DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_emp_id || ', Name: ' || v_name);
   END LOOP;
   -- Close the cursor
   CLOSE emp_cursor;
END;
/
```

Exception Handling in PL/SQL

An exception is an error. An error accrued during the program execution is called exception.

Types of Exceptions

1. **Pre-defined or System Exceptions**: These are built-in exceptions in PL/SQL, such as `NO_DATA_FOUND`, `TOO_MANY_ROWS`, and `ZERO_DIVIDE`.
2. **User-Defined Exceptions**: These are explicitly defined by the programmer to handle specific business logic errors.

Syntax for Exception Handling

Syntax:

```
DECLARE
   <exception_name> EXCEPTION;
BEGIN
   ...
   RAISE <exception_name>;
   ...
END;
```

1. System-defined Exceptions

System-defined exceptions are predefined in programming languages to handle runtime errors.

```
DECLARE
    -- Declaration statements;
BEGIN
    -- SQL statements;
    -- Procedural statements;
EXCEPTION
    -- Exception handling statements;
END;
```

Exception	Raised when....
LOGIN_DENIED	At the time when user login is denied.
TOO_MANY_ROWS	When a select query returns more than one row and the destination variable can take only single value.
VALUE_ERROR	When an arithmetic, value conversion, truncation, or constraint error occurs.

Example:

```
set serveroutput on;
DECLARE
        a int;
        b int;
        c int;
BEGIN
        a := &a;
        b := &b;
        c := a/b;
        dbms_output.put_line('RESULT=' || c);
EXCEPTION
        when ZERO_DIVIDE then
                dbms_output.put_line('Division by 0 is not possible');
END;
```

User-defined Exception

In any program, there is a possibility that a number of errors can occur that may not be considered as exceptions by oracle. In that case, an exception can be defined by the programmer while writing the code such type of exceptions are called User-defined exception. User defined exceptions are in general defined to handle special cases where our code can generate exception due to our code logic.

Also, in your code logic, you can explicitly specify to genrate an exception using the RAISE keyword and then handle it using the EXCEPTION block.

Syntax:

```
DECLARE
        <exception name> EXCEPTION
BEGIN
        <sql sentence>
        If <test_condition> THEN
                RAISE <exception_name>;
```

```
        END IF;
        EXCEPTION
                WHEN <exception_name> THEN
                        -- some action
END;
```

Example:

```
DECLARE
   v_salary NUMBER := 4000; -- Employee salary
   e_low_salary EXCEPTION; -- User-defined exception
BEGIN
   -- Check if salary is below the threshold
   IF v_salary < 5000 THEN
      RAISE e_low_salary; -- Raise the user-defined exception
   END IF;

   DBMS_OUTPUT.PUT_LINE('Salary is acceptable.');
EXCEPTION
   WHEN e_low_salary THEN
      DBMS_OUTPUT.PUT_LINE('Error: Salary is below the minimum threshold.');
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('An unexpected error occurred.');
END;
/
```

Procedures in PL/SQL

A **procedure** in PL/SQL is a named block of code that performs a specific task. It is similar to a function but does not necessarily return a value. Procedures are used to encapsulate and reuse business logic, reducing redundancy and enhancing maintainability.

Parameters in Procedures

In PL/SQL, **parameters** in procedures allow you to pass and retrieve values between the calling program and the procedure. There are three types of parameters:

1. **IN Parameter**:
    - Used to pass input values to the procedure.
    - Acts as a constant and cannot be modified within the procedure.
2. **OUT Parameter**:

- Used to return a value from the procedure to the calling program.
- Must be assigned a value within the procedure.

3. **IN OUT Parameter**:
    - Used to pass a value into the procedure, modify it within the procedure, and return it to the calling program.

Parameters in Procedures

In PL/SQL, **parameters** in procedures allow you to pass and retrieve values between the calling program and the procedure. There are three types of parameters:

1. **IN Parameter**:
    - Used to pass input values to the procedure.
    - Acts as a constant and cannot be modified within the procedure.
2. **OUT Parameter**:
    - Used to return a value from the procedure to the calling program.
    - Must be assigned a value within the procedure.
3. **IN OUT Parameter**:
    - Used to pass a value into the procedure, modify it within the procedure, and return it to the calling program.

Syntax for Parameters in Procedures

```
CREATE OR REPLACE PROCEDURE procedure_name (
   parameter1 IN datatype,       -- Input parameter
   parameter2 OUT datatype,      -- Output parameter
   parameter3 IN OUT datatype    -- Input and output parameter
) IS
   -- Declarations
BEGIN
   -- Executable statements
EXCEPTION
   -- Exception handling
END procedure_name;
/
```

Example: Procedure with Parameters

Creating the Procedure

```
CREATE OR REPLACE PROCEDURE manage_salary (
    p_emp_id IN NUMBER,        -- Input parameter
    p_salary IN OUT NUMBER,    -- Input and output parameter
    p_bonus OUT NUMBER         -- Output parameter
) IS
BEGIN
    -- Calculate bonus as 10% of salary
    p_bonus := p_salary * 0.10;

    -- Update salary by adding bonus
    p_salary := p_salary + p_bonus;

    DBMS_OUTPUT.PUT_LINE('Salary updated for Employee ID: ' || p_emp_id);
END;
/
```

Dropping a Procedure

To remove an existing procedure, use the DROP PROCEDURE statement.

Syntax:

```
DROP PROCEDURE procedure_name;
```

Example:

```
DROP PROCEDURE manage_salary;
```

PL/SQL Functions

PL/SQL functions are reusable blocks of code that can be used to perform specific tasks. They are similar to **procedures** but must always return a value.

A function in PL/SQL contains:

- **Function Header**: The function header includes the function name and an optional parameter list. It is the first part of the function and specifies the name and parameters.
- **Function Body**: The function body contains the executable statements that implement the specific logic. It can include declarative statements, executable statements, and exception-handling statements.

Create Function in PL/SQL

To create a procedure in PL/SQL, use the **CREATE FUNCTION statement.**

Syntax

```
CREATE [OR REPLACE] FUNCTION function_name
(parameter_name type [, …])

— This statement is must for functions
RETURN return_datatype

{IS | AS}

BEGIN
— program code

[EXCEPTION
exception_section;

END [function_name];
```

Example

```
CREATE OR REPLACE FUNCTION factorial(x NUMBER)
  RETURN NUMBER
IS
  f NUMBER;
BEGIN
  IF x = 0 THEN
    f := 1;
  ELSE
    f := x * factorial(x - 1);
  END IF;
  RETURN f;
END;
/
```

Dropping a Function

To remove an existing function, use the DROP FUNCTION statement.

Syntax:

```
DROP FUNCTION function_name;
```

Example:

```
DROP FUNCTION calculate_bonus;
```

File Organization

A database consists of a huge amount of data. The data is grouped within a table in RDBMS, and each table have related records. A user can see that the data is stored in form of tables, but in actual this huge amount of data is stored in physical memory in form of files.

Types of File Organizations –

Various methods have been introduced to Organize files. These particular methods have advantages and disadvantages on the basis of access or selection . Thus it is all upon the programmer to decide the best suited file Organization method according to his requirements.

Some types of File Organizations are:

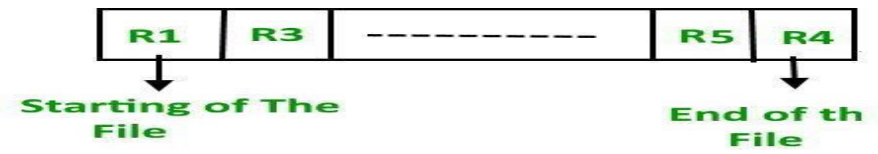
- Sequential File Organization
- Heap File Organization
- Hash File Organization
- B+ Tree File Organization
- Indexed sequential access method (ISAM)
- Clustered File Organization

We will be discussing each of the file Organizations in further sets of this article along with differences and advantages/ disadvantages of each file Organization methods.

Sequential File Organization –

The easiest method for file Organization is Sequential method. In this method the file are stored one after another in a sequential manner. There are two ways to implement this method:

1. **Pile File Method –** This method is quite simple, in which we store the records in a sequence i.e one after other in the order in which they are inserted into the tables.



Insertion of new record –

Let the R1, R3 and so on upto R5 and R4 be four records in the sequence. Here, records are nothing but a row in any table. Suppose a new record R2 has to be inserted in the sequence, then it is simply placed at the end of the file.

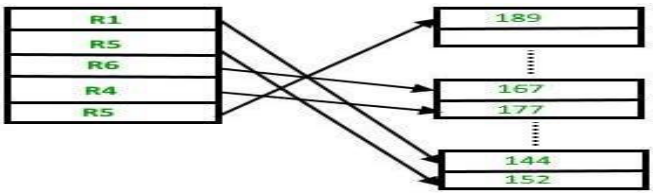
2. **Sorted File Method** –In this method, As the name itself suggest whenever a new record has to be inserted, it is always inserted in a sorted (ascending or descending) manner. Sorting of records may be based on any primary key or any other key.

Insertion of new record –

Let us assume that there is a preexisting sorted sequence of four records R1, R3, and so on upto R7 and R8. Suppose a new record R2 has to be inserted in the sequence, then it will be inserted at the end of the file and then it will sort the sequence .

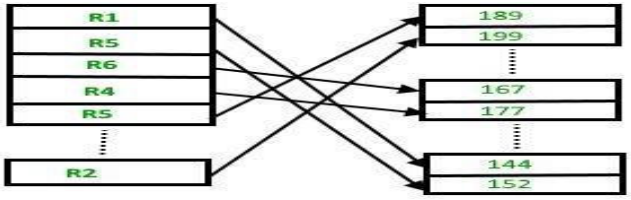
Heap File Organization –

Heap File Organization works with data blocks. In this method records are inserted at the end of the file, into the data blocks. No Sorting or Ordering is required in this method. If a data block is full, the new record is stored in some other block, Here the other data block need not be the very next data block, but it can be any block in the memory. It is the responsibility of DBMS to store and manage the new records.



Insertion of new record –

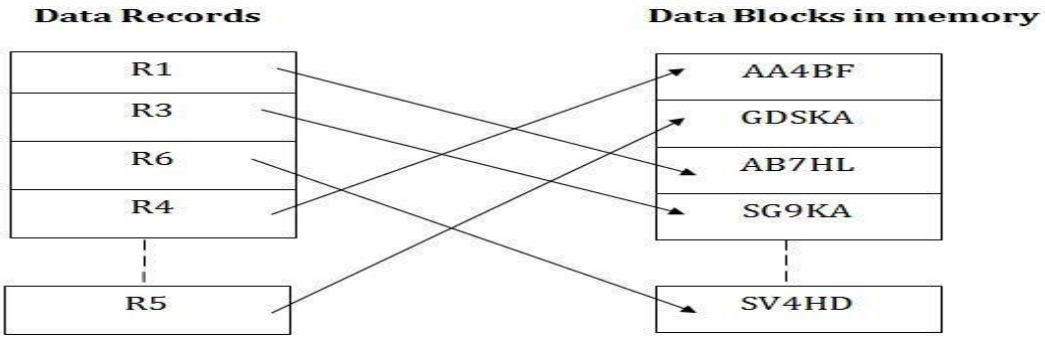
Suppose we have four records in the heap R1, R5, R6, R4 and R3 and suppose a new record R2 has to be inserted in the heap then, since the last data block i.e data block 3 is full it will be inserted in any of the data blocks selected by the DBMS, lets say data block 1.



If we want to search, delete or update data in heap file Organization the we will traverse the data from the beginning of the file till we get the requested record. Thus if the database is very huge, searching, deleting or updating the record will take a lot of time.

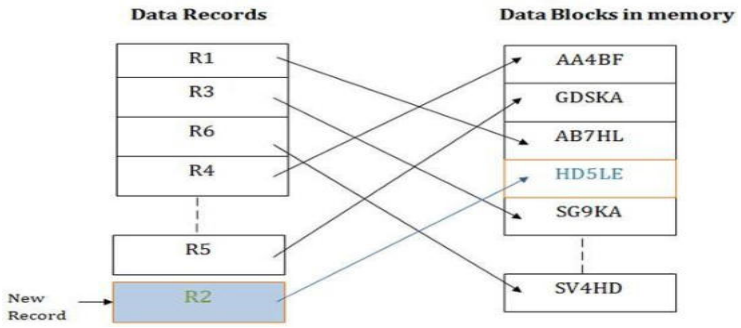
Hash File Organization

Hash File Organization uses the computation of hash function on some fields of the records. The hash function's output determines the location of disk block where the records are to be placed.



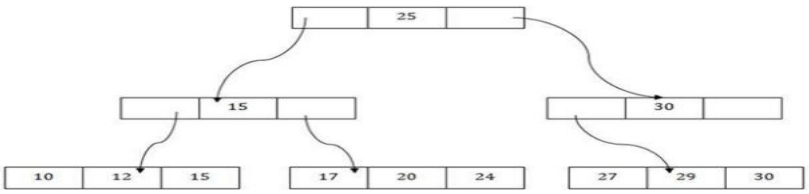
When a record has to be received using the hash key columns, then the address is generated, and the whole record is retrieved using that address. In the same way, when a new record has to be inserted, then the address is generated using the hash key and record is directly inserted. The same process is applied in the case of delete and update.

In this method, there is no effort for searching and sorting the entire file. In this method, each record will be stored randomly in the memory.



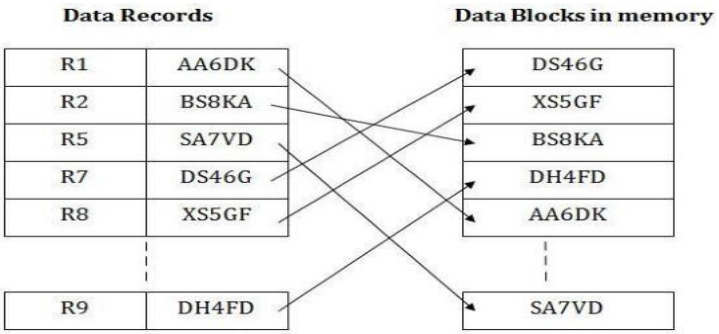
B+ File Organization

- B+ tree file organization is the advanced method of an indexed sequential access method. It uses a tree-like structure to store records in File.
- It uses the same concept of key-index where the primary key is used to sort the records. For each primary key, the value of the index is generated and mapped with the record.
- The B+ tree is similar to a binary search tree (BST), but it can have more than two children. In this method, all the records are stored only at the leaf node. Intermediate nodes act as a pointer to the leaf nodes. They do not contain any records.



Indexed sequential access method (ISAM)

ISAM method is an advanced sequential file organization. In this method, records are stored in the file using the primary key. An index value is generated for each primary key and mapped with the record. This index contains the address of the record in the file.



If any record has to be retrieved based on its index value, then the address of the data block is fetched and the record is retrieved from the memory.

Cluster file organization

- When the two or more records are stored in the same file, it is known as clusters. These files will hae two or more tables in the same data block, and key attributes which are used to map these tables together are stored only once.
- This method reduces the cost of searching for various records in different files.
- The cluster file organization is used when there is a frequent need for joining the tables with the same condition. These joins will give only a few records from both tables.

EMPLOYEE

EMP_ID	EMP_NAME	ADDRESS	DEP_ID
1	John	Delhi	14
2	Robert	Gujarat	12
3	David	Mumbai	15
4	Amelia	Meerut	11
5	Kristen	Noida	14
6	Jackson	Delhi	13
7	Amy	Bihar	10
8	Sonoo	UP	12

DEPARTMENT

DEP_ID	DEP_NAME
10	Math
11	English
12	Java
13	Physics
14	Civil
15	Chemistry

Cluster Key

DEP_ID	DEP_NAME	EMP_ID	EMP_NAME	ADDRESS
10	Math	7	Amy	Bihar
11	English	4	Amelia	Meerut
12	Java	2	Robert	Gujarat
12		8	Sonoo	UP
13	Physics	6	Jackson	Delhi
14	Civil	1	John	Delhi
14		5	Kristen	Noida
15	Chemistry	3	David	Mumbai

In this method, we can directly insert, update or delete any record. Data is sorted based on the key with which searching is done. Cluster key is a type of key with which joining of the table is performed.

Data Dictionary storage

Till now, we learned and understood about relations and its representation. In the relational database system, it maintains all information of a relation or table, from its schema to the applied constraints. All the metadata is stored. In general, metadata refers to the data about data. So, storing the relational schemas and other metadata about the relations in a structure is known as **Data Dictionary** or **System Catalog**.

1. Integrated Data Dictionary:

An **Integrated Data Dictionary** is a data dictionary that is tightly coupled with the DBMS. This means that the metadata (information about database structure, schema, and other objects) is automatically stored and managed by the DBMS itself. Every time the database schema or structure is modified (such as when a new table is created, a column is added, or a constraint is changed), the data dictionary is updated automatically by the DBMS.

Characteristics:

- **Automatic Updates**: Whenever any change is made to the database (like creating tables, adding columns, etc.), the integrated data dictionary is updated automatically.
- **Part of DBMS**: It is an internal component of the DBMS and is not a separate system. It exists as system tables or views within the database.
- **Real-Time Access**: It is directly accessed by the DBMS for operations like query optimization, integrity checks, and enforcing constraints.
- **No Manual Intervention**: Users or administrators do not need to manually update the dictionary. It is managed by the DBMS system.

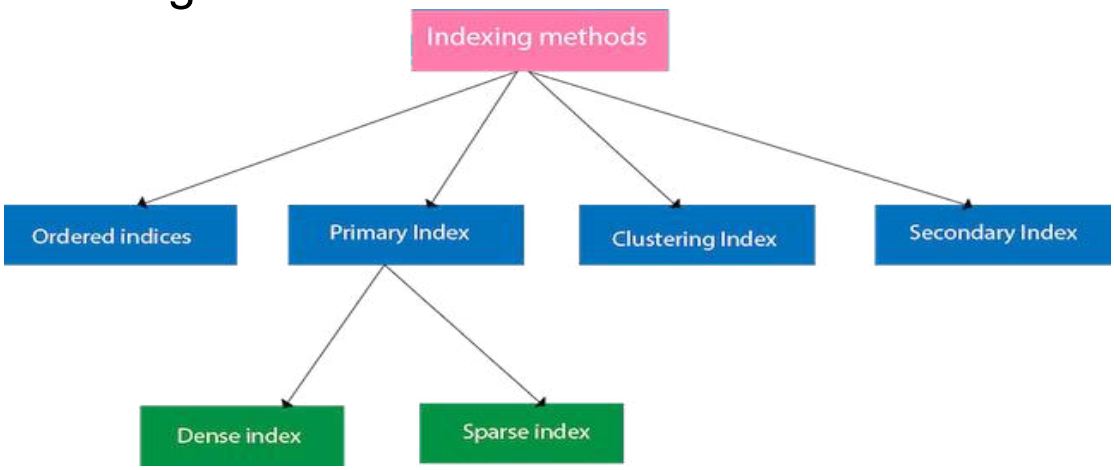
2. Stand-alone Data Dictionary:

A **Stand-alone Data Dictionary** is a separate system or repository used to store metadata about the database schema, objects, and structures. It is not integrated with the DBMS. This type of data dictionary must be manually updated whenever the database schema or structure is modified. The stand-alone data dictionary can exist outside of the DBMS and is often used for auditing or documentation purposes.

Characteristics:

- **Manual Updates**: Whenever a change is made to the database, such as creating new tables, adding columns, or changing relationships, the stand-alone data dictionary needs to be manually updated by database administrators or users.
- **Separate from DBMS**: It exists outside the DBMS and is not automatically updated by the system. It may be maintained in a separate file, document, or external database.
- **Used for Documentation and Auditing**: Often used for purposes like database documentation, auditing, and ensuring consistency across different environments.
- **No Direct Access by DBMS**: The DBMS does not use the stand-alone data dictionary for operations like query optimization or constraint enforcement.

Indexing Methods:



Ordered indices

The indices are usually sorted to make searching faster. The indices which are sorted are known as ordered indices.

Example: Suppose we have an employee table with thousands of record and each of which is 10 bytes long. If their IDs start with 1, 2, 3....and so on and we have to search student with ID-543.

- In the case of a database with no index, we have to search the disk block from starting till it reaches 543. The DBMS will read the record after reading 543*10=5430 bytes.
- In the case of an index, we will search using indexes and the DBMS will read the record after reading 542*2= 1084 bytes which are very less compared to the previous case.

Primary Index

- If the index is created on the basis of the primary key of the table, then it is known as primary indexing. These primary keys are unique to each record and contain 1:1 relation between the records.
- As primary keys are stored in sorted order, the performance of the searching operation is quite efficient.

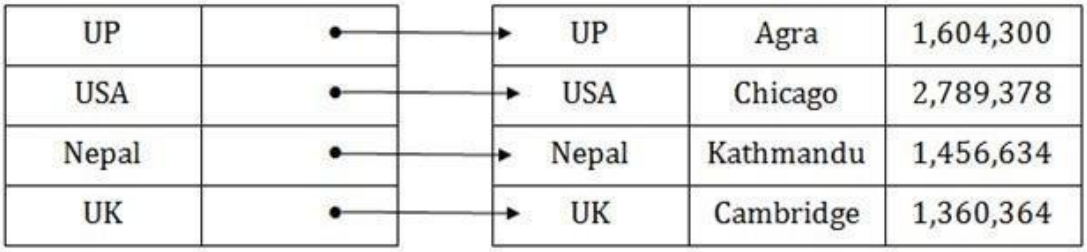
The primary index can be classified into two types:

Dense index and Sparse index.

Dense index

- The dense index contains an index record for every search key value in the data file. It makes searching faster.
- In this, the number of records in the index table is same as the number of records in the main table.

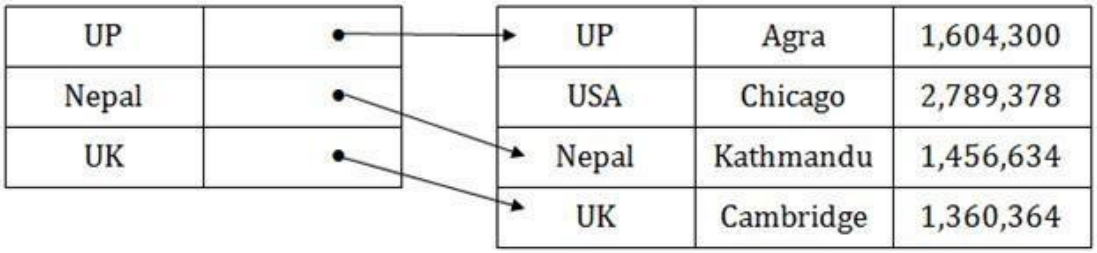
- It needs more space to store index record itself. The index records have the search key and a pointer to the actual record on the disk.



UP		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.

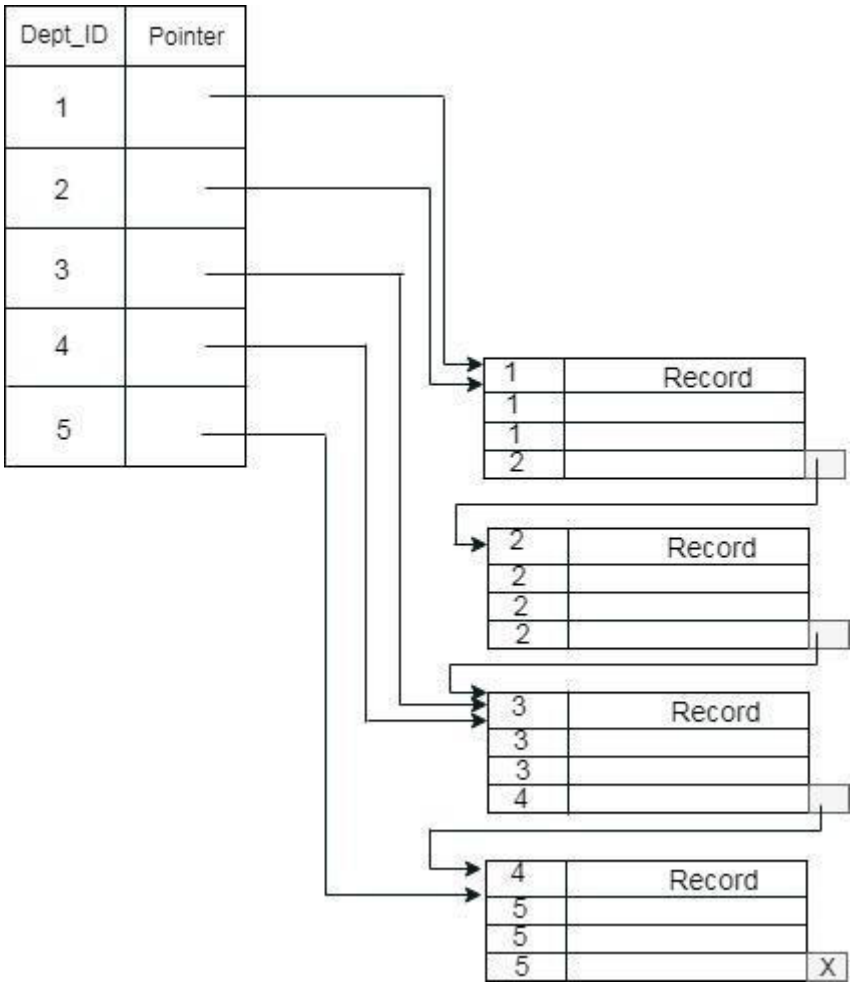


UP		UP	Agra	1,604,300
Nepal		USA	Chicago	2,789,378
UK		Nepal	Kathmandu	1,456,634
		UK	Cambridge	1,360,364

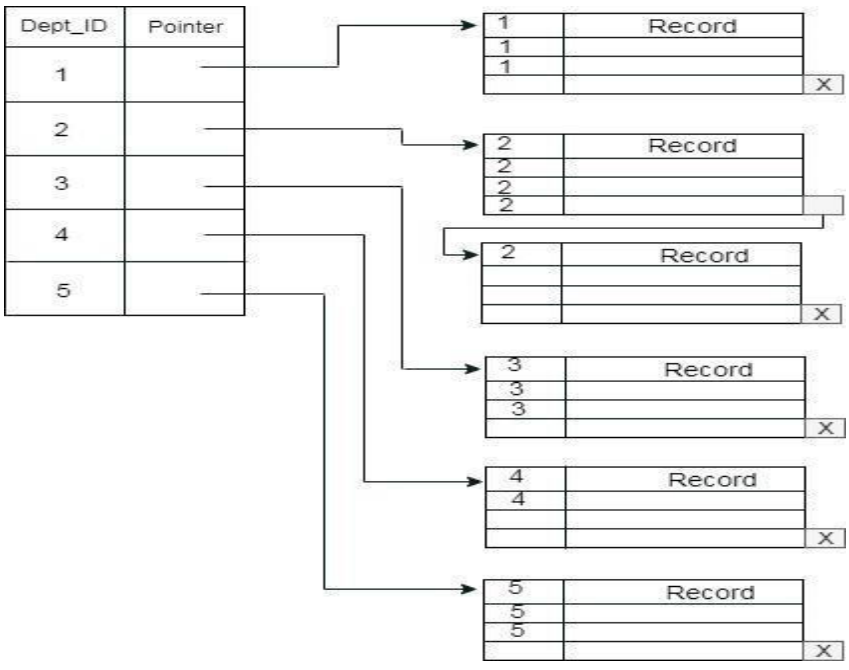
Clustering Index

- 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.



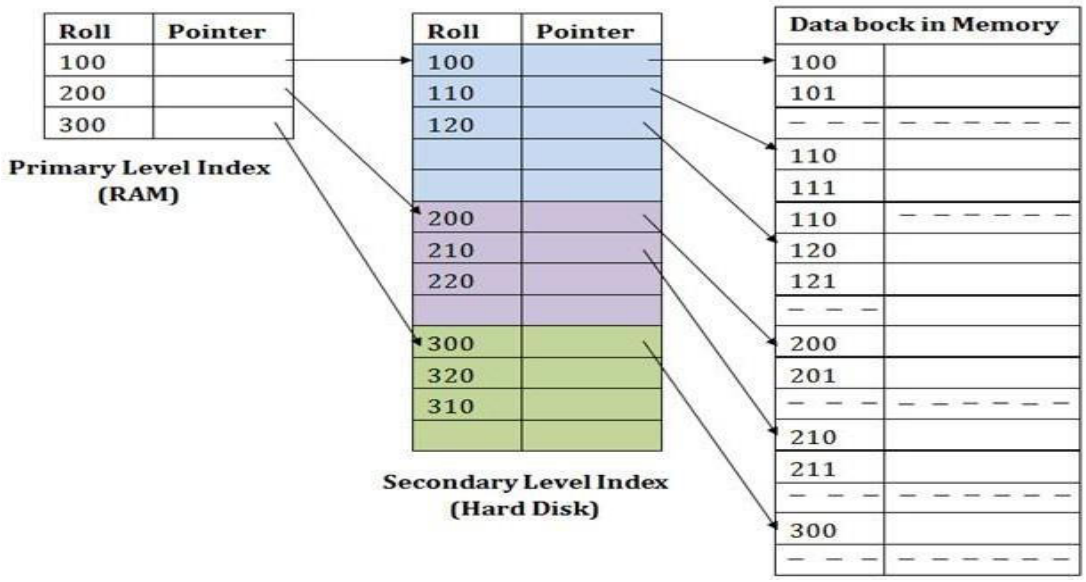
The previous schema is little confusing because one disk block is shared by records which belong to the different cluster. If we use separate disk block for separate clusters, then it is called better technique.



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).

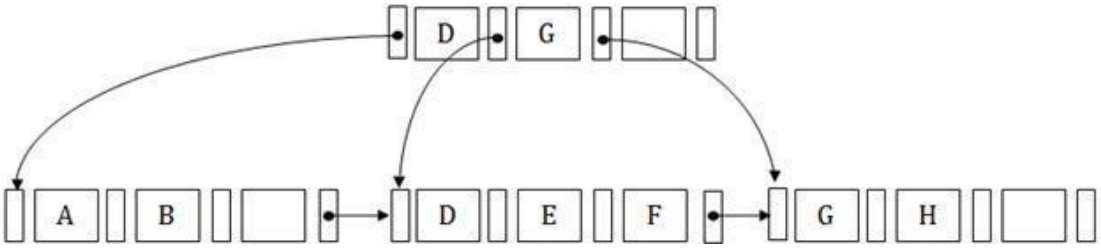


B+ Tree

- 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.

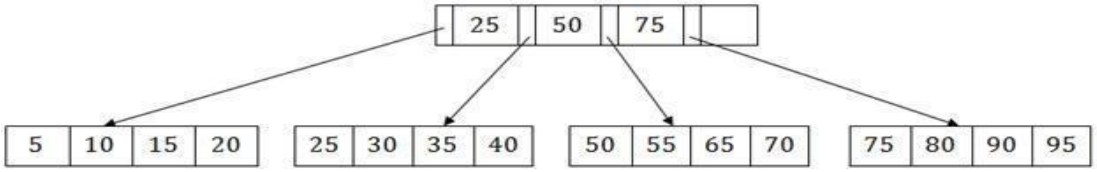
Leaf node

- 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.

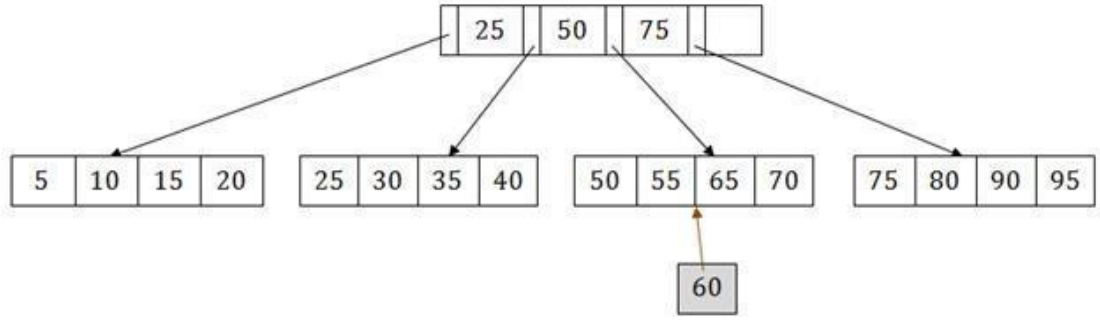
So, in the intermediary node, we will find a branch between 50 and 75 nodes. Then at the end, we will be redirected to the third leaf node. Here DBMS will perform a sequential search to find 55.



B+ Tree Insertion

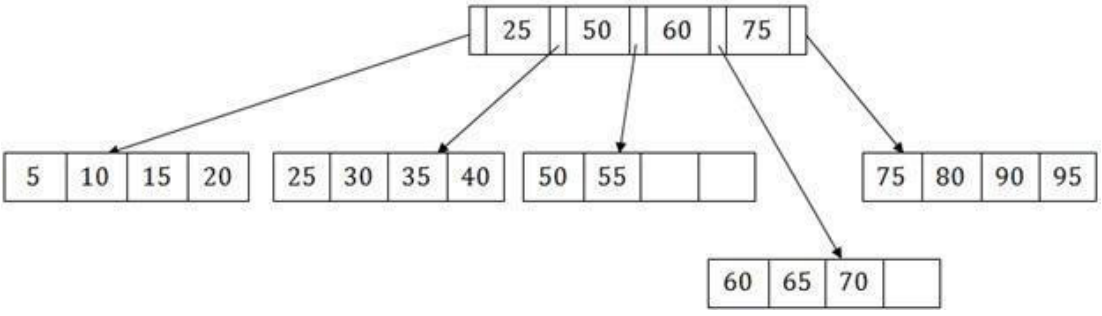
Suppose we want to insert a record 60 in the below structure. It will go to the 3rd leaf node after 55. It is a balanced tree, and a leaf node of this tree is already full, so we cannot insert 60 there.

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.



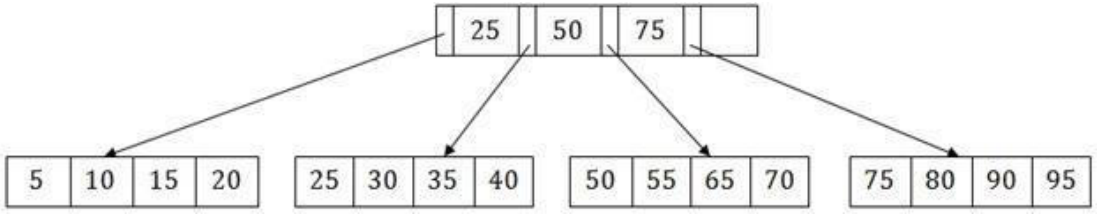
This is how we can insert an entry when there is overflow. In a normal scenario, it is very easy to find the node where it fits and then place it in that leaf node.

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:



Operationsin B+treein DBMS

In the B+ tree, we can perform so many operations of insertion, deletion, and searching a node in a B+ tree.

- Insertion
- Deletion
- Searching

Insertionin B+Tree

Insertion is a process in which we can insert any data in the B+ tree in the form of nodes.

We can insert value in the leaf node of the B+ tree. There are 2 cases of insertion as follows:

- **Case 1:** In the first case of insertion, if the leaf node is empty then we can simply insert the value into the leaf node in the B+ tree in increasing order if the leaf node is not full.
- **Case 2:** In the second case of insertion, if the leaf node is full then we need to follow some procedure or steps as follows:

Step 1: first we need to insert the new node in the leaf node in increasing order but the leaf node is already full, we need some balancing in the node to store the value.

Step 2: now we need to break the node in the B+ tree at the m/2 position.

Step 3: After breaking the node, we add the m/2 value to the parent node of the B+ tree.

Step 4: if the parent node is full then in this case we just repeat steps 2 and 3.

Let's understand by an example:

Example:

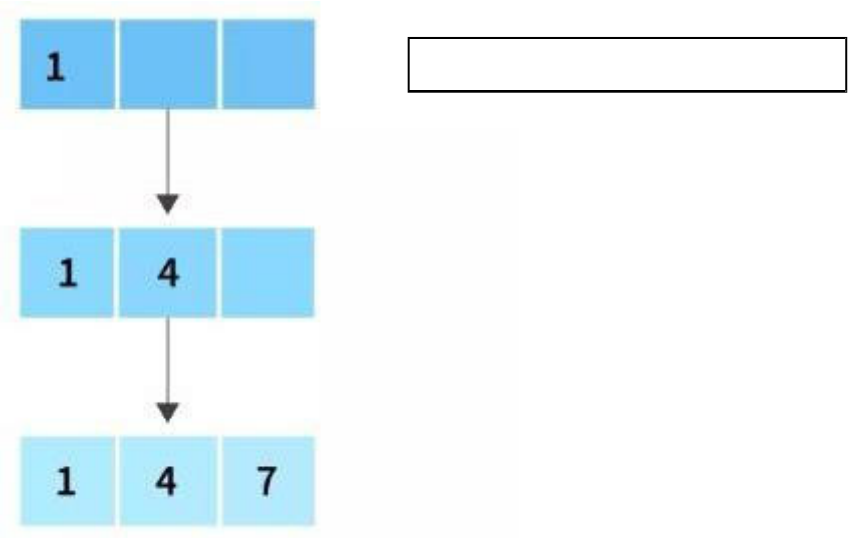
First, we need to create the B+ tree with the help of this data: 1,4,7,10,17,21,31

Let's suppose our B+ tree is in the order(m) that is 4.

Now max children= 4,

and minimum children: m/2 =2 and to find the max keys(m-1):4-1=3 and to find the min keys(m/2)-1: (4/2)-1= 1

Now, first we will insert the values 1,4 and 7 into the first node of the B+ tree.

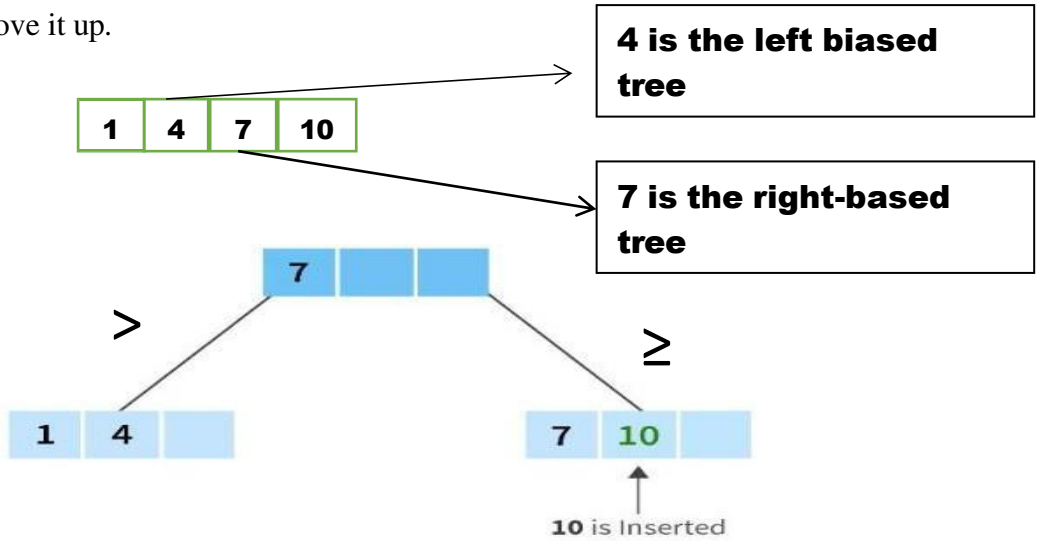


After the insertion of 1,4 and 7, now we will insert the next element 10, but you can see there is no space in the current node(because we can only store a maximum of 3 keys in a node), we need to split the node.

Now, we need to spilt between 4 or 7 nodes.

If we choose 4 then the tree will be **left biased tree.** And if we choose 7 then the tree will be the **right-biased tree.** We can choose one of them.

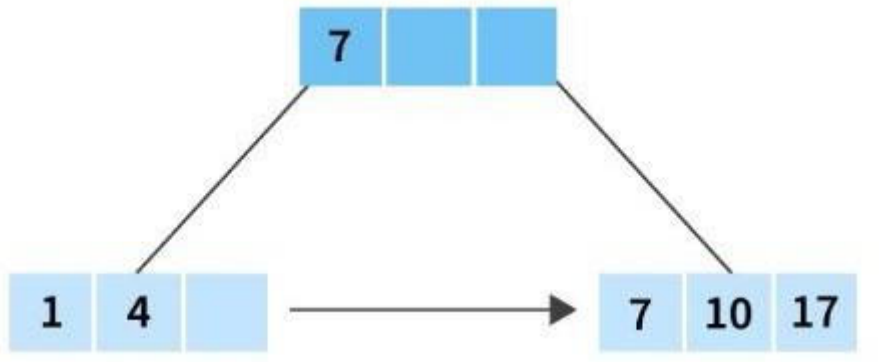
Let's choose a **right-based tree** for this example. Now, we split node 7 and after that, we move it up.



Note : b+ tree

After inserting 10, now we will split it into:

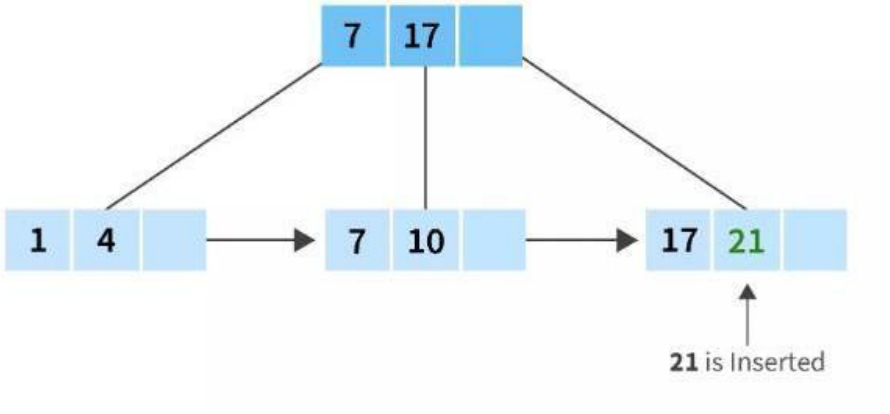
I. 1,4 and a blank space (because the maximum number of keys in a node can be 3) and,
II. We will insert the next value 17 into the 7 and 10 nodes and it will come after 10 in the ascending order.



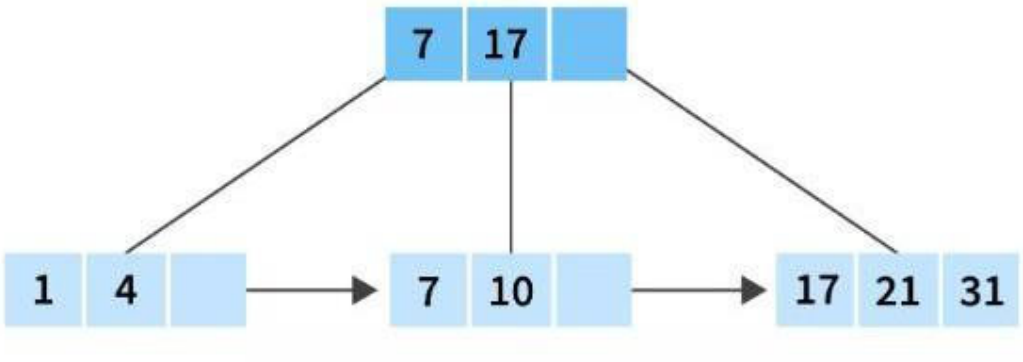
Now, we will insert 21 and preferably, it should be placed after 17. But the node is full because of 3 keys. We need to split the node, as we all know that we choose the right biased tree after step 2, so we will split the node in the same way.

I. 7,10 and a blank space

II. And 17,21 and a blank space



And all the nodes have a minimum of 2 children, which satisfies the fact that we deduced the B+ tree before the insertion in the B+ tree.



Deletionin B+tree

Deletion is a process when we delete the data or value/keys from the tree. In other words, if we want to delete some value from the node in the B+ tree, we use the deletion operation to perform that task.

Similar to insertion, deletion also follows two cases for deleting any value from the B+ tree.

Case 1: Let's suppose we want to delete the value that is present only in the leaf node of the B+ tree.

If the value to be deleted is present in the leaf node only then we simply delete the value easily.

Case 2: Let's suppose the value to be deleted is present in the leaf node and also has a pointer to it as well then first, we need to locate the node that we want to delete and remove it from the leaf node, but we also need to remove it from the index of the B+ tree that points to this node.

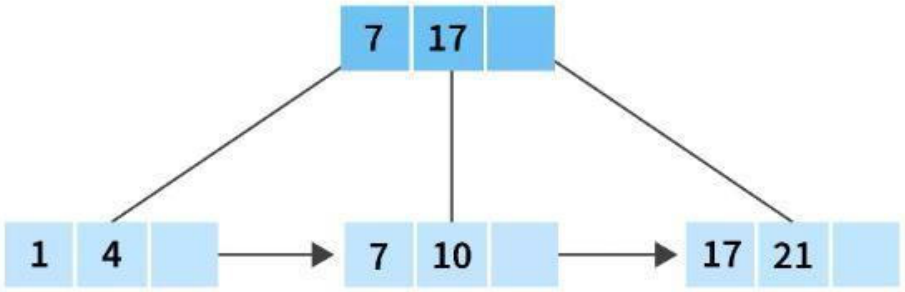
Let’s take an example to understand the deletion in the B+ tree.

Example:

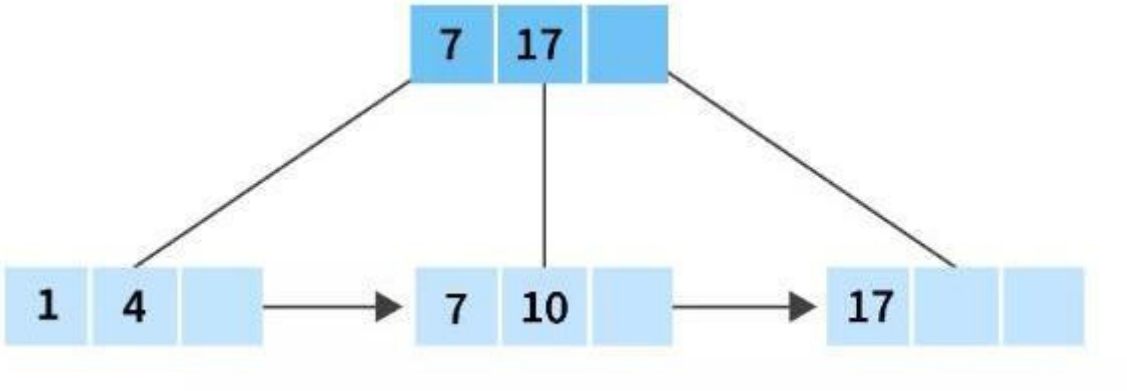
Suppose we want to delete 21 from the form of the leaf node in the B+ tree.

We will simply delete the value from the B+ tree.

Before deletion:



After the deletion:

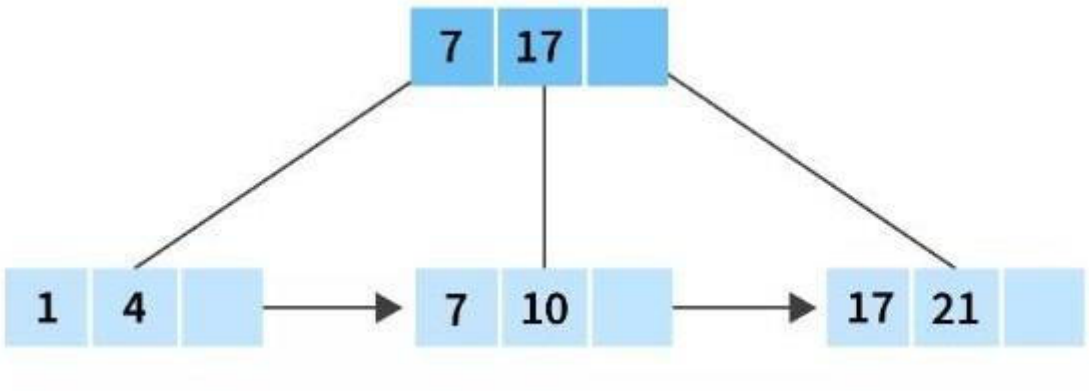


So 21 is present in the leaf node of the B+ tree and if we delete the value it will not invalidate the minimum number of keys which is 1, so we can simply delete the value from the leaf node.

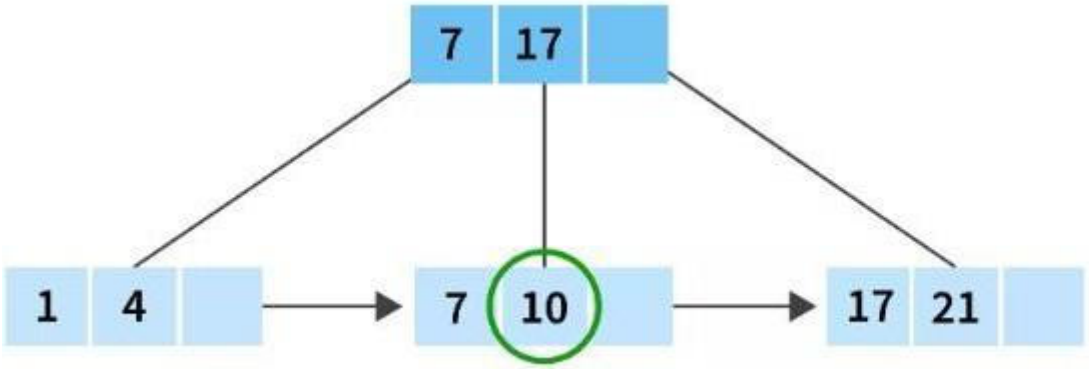
Let's understand searching by an example:

Example 1:

Suppose we want to search the element 10 from the B+ tree.



First, we compare element 10 with the starting node which is 7 and 17. as we all know 10 lies between them, so we simply need to search in the records between 7 and 17. Since 7 and 17 are leaf nodes we will get the desired value which is 10 in this case.



B- trees

B trees are extended binary search trees that are specialized in m-way searching, since the order of B trees is 'm'. Order of a tree is defined as the maximum number of children a node can accommodate. Therefore, the height of a b tree is relatively smaller than the height of AVL tree and RB tree.

B-Tree of Order m has the following properties...

The various properties of B trees include –

- Every node in a B Tree will hold a maximum of m children and (m-1) keys, since the order of the tree is m.
- Every node in a B tree, except root and leaf, can hold at least m/2 children
- The root node must have no less than two children.
- All the paths in a B tree must end at the same level, i.e. the leaf nodes must be at the same level.
- A B tree always maintains sorted data.

Operations on a B-Tree

The following operations are performed on a B-Tree...

1. **Search**
2. **Insertion**
3. **Deletion**

Search Operation in B-Tree

The search operation in B-Tree is similar to the search operation in Binary Search Tree. In a Binary search tree, the search process starts from the root node and we

make a 2-way decision every time (we go to either left subtree or right subtree). In B-Tree also search process starts from the root node but here we make an n-way decision every time. Where 'n' is the total number of children the node has. In a B-Tree, the search operation is performed with **O(log n)** time complexity. The search operation is performed as follows...

- **Step 1 -** Read the search element from the user.
- **Step 2 -** Compare the search element with first key value of root node in the tree.
- **Step 3 -** If both are matched, then display "Given node is found!!!" and terminate the function
- **Step 4 -** If both are not matched, then check whether search element is smaller or larger than that key value.
- **Step 5 -** If search element is smaller, then continue the search process in left subtree.
- **Step 6 -** If search element is larger, then compare the search element with next key value in the same node and repeate steps 3, 4, 5 and 6 until we find the exact match or until the search element is compared with last key value in the leaf node.
- **Step 7 -** If the last key value in the leaf node is also not matched then display "Element is not found" and terminate the function.

Insertion Operation in B-Tree

In a B-Tree, a new element must be added only at the leaf node. That means, the new keyValue is always attached to the leaf node only. The insertion operation is performed as follows...

- **Step 1 -** Check whether tree is Empty.
- **Step 2 -** If tree is **Empty**, then create a new node with new key value and insert it into the tree as a root node.
- **Step 3 -** If tree is **Not Empty**, then find the suitable leaf node to which the new key value is added using Binary Search Tree logic.
- **Step 4 -** If that leaf node has empty position, add the new key value to that leaf node in ascending order of key value within the node.
- **Step 5 -** If that leaf node is already full, **split** that leaf node by sending middle value to its parent node. Repeat the same until the sending value is fixed into a node.

- **Step 6 -** If the spilting is performed at root node then the middle value becomes new root node for the tree and the height of the tree is increased by one.

Example

Construct a B Tree of order 3 by inserting numbers from

1, 2, 3,4,5,6,7,8,9, 10.

The insertion is done using the following procedure −

Step 1: Calculate the maximum (m−1)(m−1) and,

minimum (⌈m2⌉−1)(⌈m2⌉−1) number of keys a node can hold, where m is denoted by the order of the B Tree.

Order (M) =3

Maximum keys (m-1)=2

Minimum key (m/2)-1=1

Maximum chidren =3

Minimum children (m/2)=2

Insert (1):

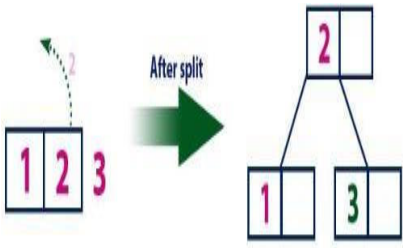
Since '1'is the first element into the tree that is inserted into a new node. It acts as the root node.

Insert (2)

Element '2'is added to existing leaf node. Here, we have only one node and that node acts as root and also leaf. This leaf node has an empty position. So, new element (2) can be inserted at that empty position.

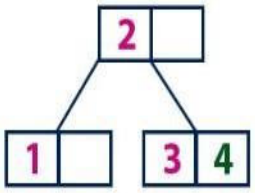
Insert (3)

Element '3'is added to existing leaf node. Here, we have only one node and that node acts as root and also leaf. This leaf node doesn't has an empty position. So, we split that node by sending middle value (2) to its parent node. But here, this node doesn't has parent. So, this middle value becomes a new root node for the tree.



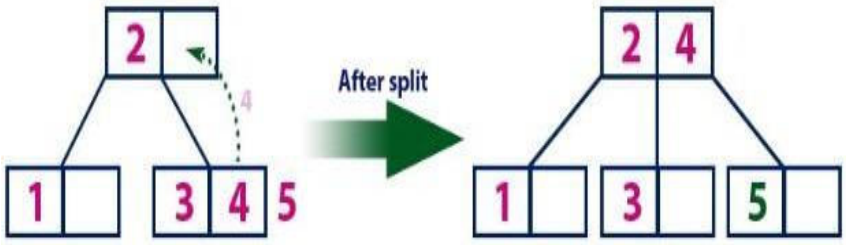
Insert (4)

Element '4'is larger than root node '2' and it is not a leaf node. So, we move to the right of '2. We reach to a leaf node with value '3' and it has an empty position. So, new element (4) can be inserted at that empty position.



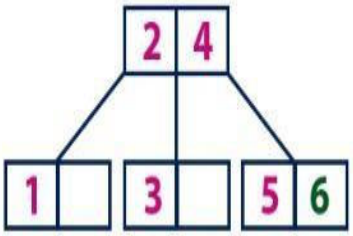
Insert (5)

Element '5'is larger than root node'2' and it is not a leaf node. So, we move to the right of '2. We reach to a leaf node and it is already full. So, we split that node by sending middle value (4) to its parent node (2). There is an empty position in its parent node. So, value '4'is added to node with value '2' and new element'5' added as new leaf node.



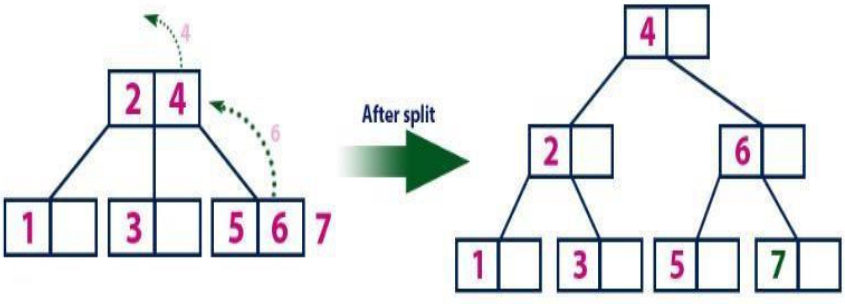
Insert (6)

Element '6'is larger than root node'2' &'4' and it is not a leaf node. So, we move to the right of '4. We reach to a leaf node with value'5' and it has an empty position. So, new element (6) can be inserted at that empty position.



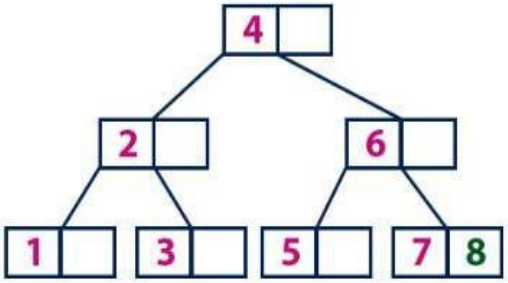
Insert (7)

Element '7'is larger than root node '2'&'4' and it is not a leaf node. So, we move to the right of '4. We reach to a leaf node and it is already full. So, we split that node by sending middle value (6) to its parent node (2&4). But the parent (2&4) is also full. So, again we split the node (2&4) by sending middle value '4' to its parent but this node doesn't have parent. So, the element '4' becomes new root node for the tree.



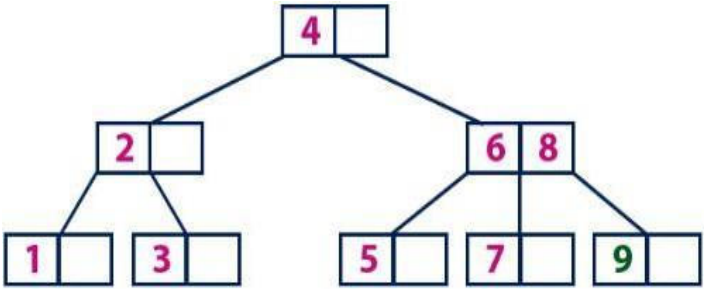
Insert (8)

Element '8'is larger than root node'4' and it is not a leaf node. So, we move to the right of '4. We reach to a node with value '6''8' is larger than '6' and it is also not a leaf node. So, we move to the right of '6: We reach to a leaf node (7) and it has an empty position. So, new element (8) can be inserted at that empty position.



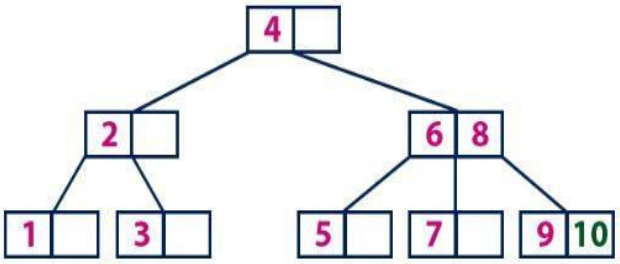
Insert (9)

Element '9'is larger than root node'4' and it is not a leaf node. So, we move to the right of '4. We reach to a node with value'6. '9' is larger than '6' and it is also not a leaf node. So, we move to the right of '6: We reach to a leaf node (7 & 8). This leaf node is already full. So, we split this node by sending middle value (8) to its parent node. The parent node (6) has an empty position. So, '8' is added at that position. And new element is added as a new leaf node.



Insert (10)

Element '10' is larger than root node '4' and it is not a leaf node. So, we move to the right of'4. We reach to a node with values '6 & 8 '10' is larger than '6 & 8' and it is also not a leaf node. So, we move to the right of'8. We reach to a leaf node (9). This leaf node has an empty position. So, new element '10' is added at that empty position.

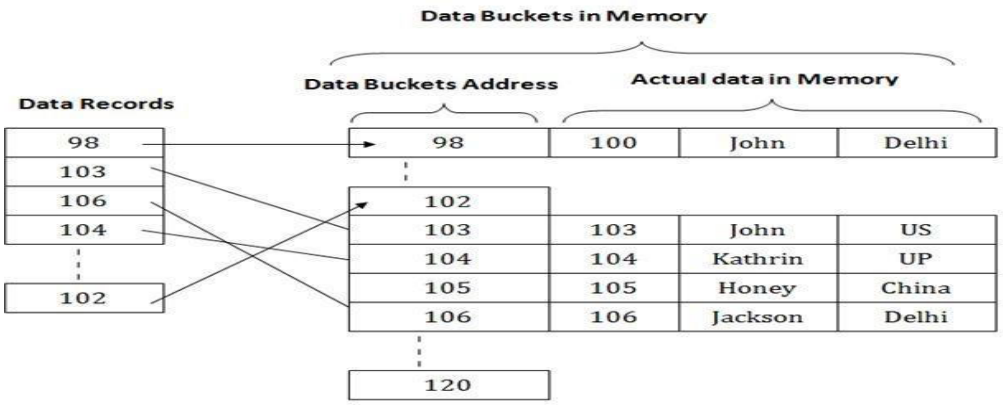


Hashing in DBMS

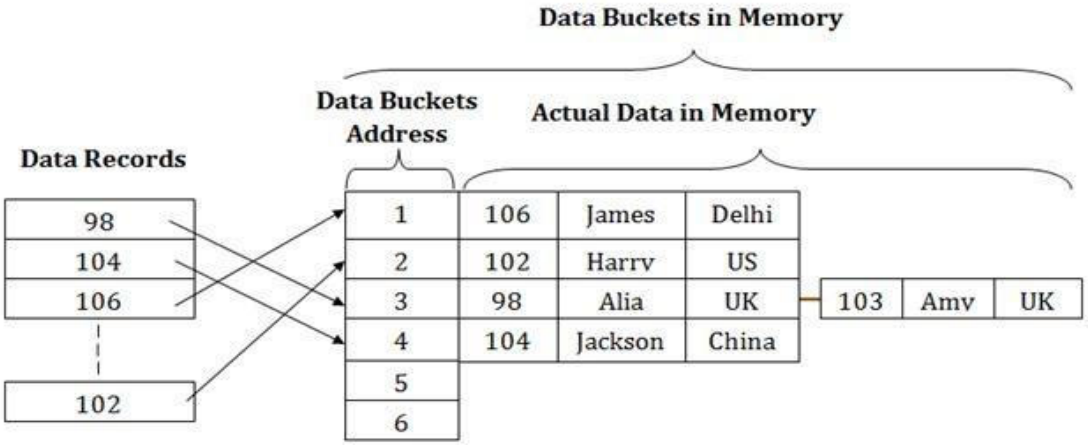
In a huge database structure, it is very inefficient to search all the index values and reach the desired data. Hashing technique is used to calculate the direct location of a data record on the disk without using index structure.

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.

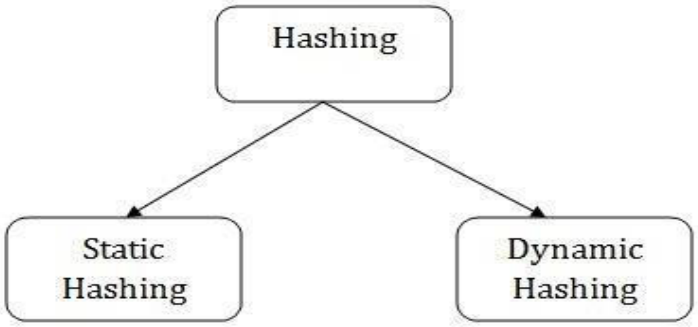
In this, a hash function can choose any of the column value to generate the address. Most of the time, the hash function uses the primary key to generate the address of the data block. A hash function is a simple mathematical function to any complex mathematical function. We can even consider the primary 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.



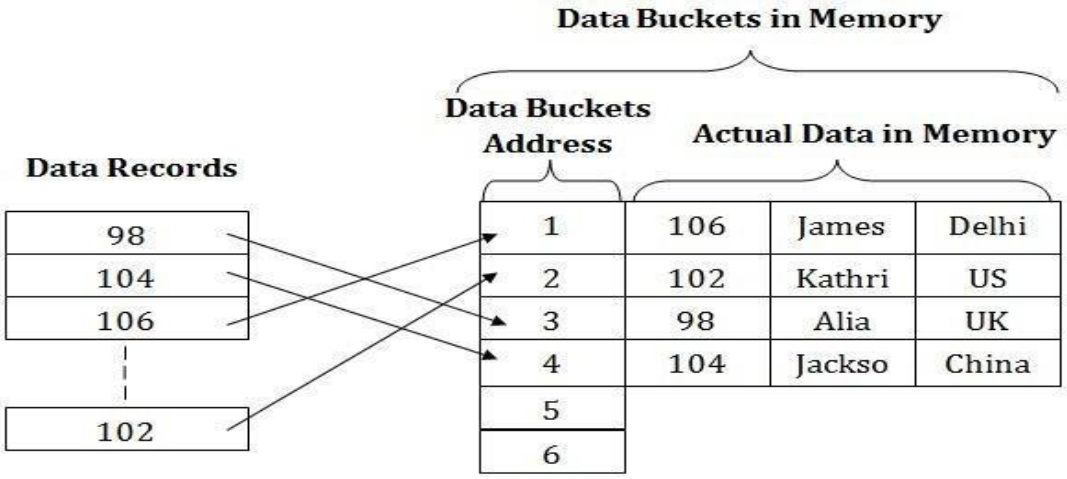
Types of Hashing:



Static Hashing

In static hashing, the resultant data bucket address will always be the same. That means if we generate an address for EMP_ID =103 using the hash function mod (5) then it will always result in same bucket address 3. Here, there will be no change in the bucket address.

Hence in this static hashing, the number of data buckets in memory remains constant throughout. In this example, we will have five data buckets in the memory used to store the data.



Operations of Static Hashing

- **Searching a record**

When a record needs to be searched, then the same hash function retrieves the address of the bucket where the data is stored.

- **Insert a Record**

When a new record is inserted into the table, then we will generate an address for a new record based on the hash key and record is stored in that location.

- **Delete a Record**

To delete a record, we will first fetch the record which is supposed to be deleted. Then we will delete the records for that address in memory.

- **Update a Record**

To update a record, we will first search it using a hash function, and then the data record is updated.

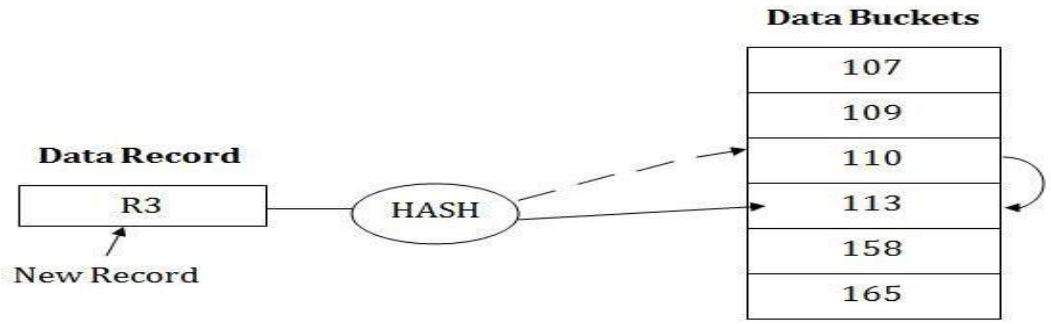
If we want to insert some new record into the file but the address of a data bucket generated by the hash function is not empty, or data already exists in that address. This situation in the static hashing is known as **bucket overflow**. This is a critical situation in this method.

To overcome this situation, there are various methods. Some commonly used methods are as follows:

1. Open Hashing

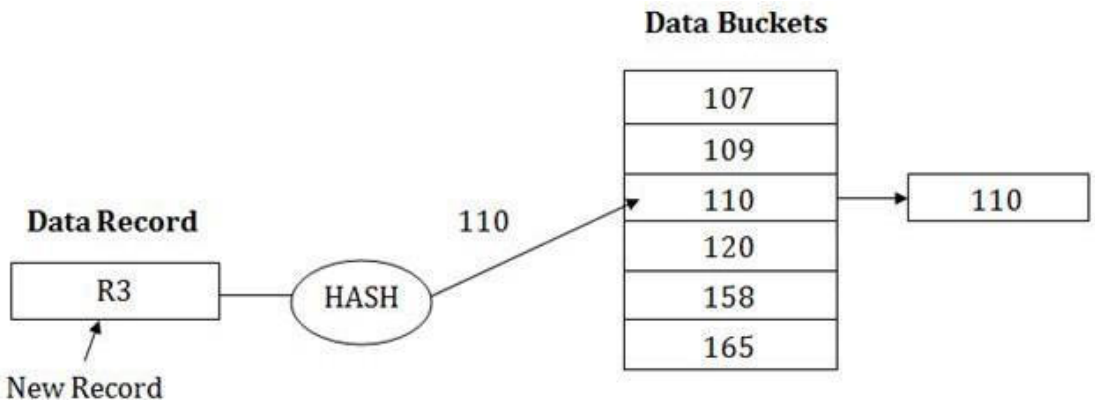
When a hash function generates an address at which data is already stored, then the next bucket will be allocated to it. This mechanism is called as **Linear Probing**.

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



2. Close Hashing

When buckets are full, then a new data bucket is allocated for the same hash result and is linked after the previous one. This mechanism is known as Overflow chaining. For example: Suppose R3 is a new address which needs to be inserted into the table, the hash function generates address as 110 for it. But this bucket is full to store the new data. In this case, a new bucket is inserted at the end of 110 buckets and is linked to it.



Dynamic Hashing

- The dynamic hashing method is used to overcome the problems of static hashing like bucket overflow.
- In this method, data buckets grow or shrink as the records increases or decreases. This method is also known as Extendable hashing method.
- This method makes hashing dynamic, i.e., it allows insertion or deletion without resulting in poor performance.

How to search a key

- First, calculate the hash address of the key.
- Check how many bits are used in the directory, and these bits are called as i.
- Take the least significant i bits of the hash address. This gives an index of the directory.
- Now using the index, go to the directory and find bucket address where the record might be.

How to insert a new record

- Firstly, you have to follow the same procedure for retrieval, ending up in some bucket.

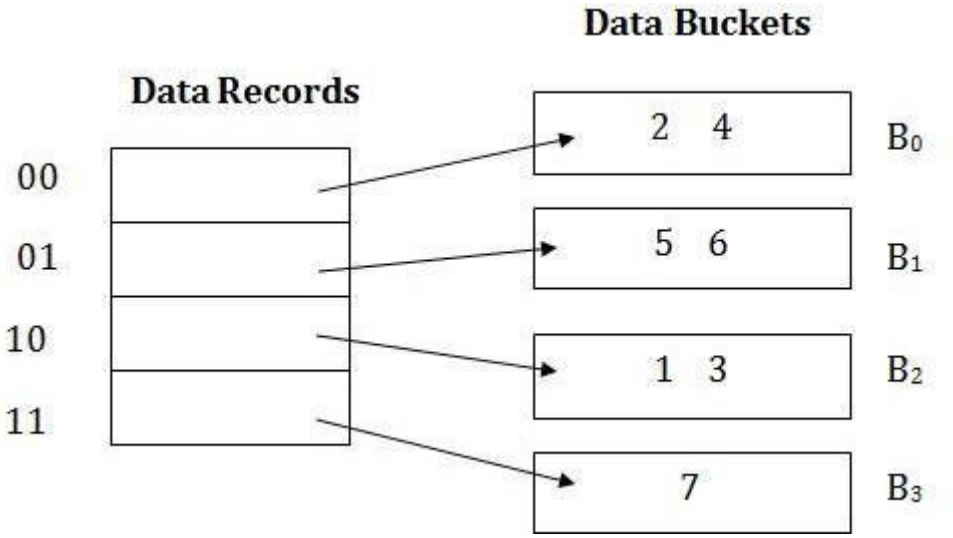
- If there is still space in that bucket, then place the record in it.
- If the bucket is full, then we will split the bucket and redistribute the records.

For example:

Consider the following grouping of keys into buckets, depending on the prefix of their hash address:

Key	Hash address
1	11010
2	00000
3	11110
4	00000
5	01001
6	10101
7	10111

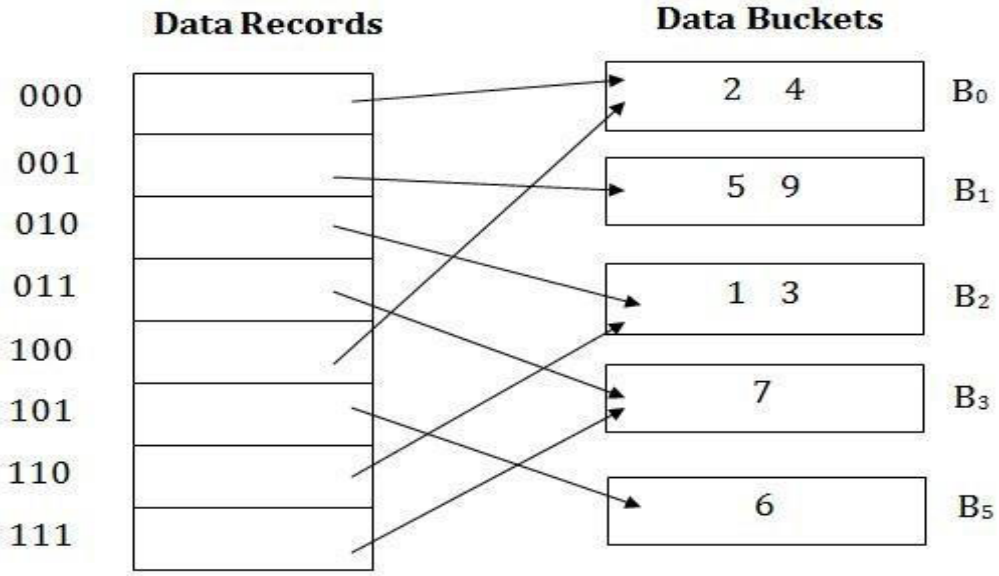
The last two bits of 2 and 4 are 00. So it will go into bucket B0. The last two bits of 5 and 6 are 01, so it will go into bucket B1. The last two bits of 1 and 3 are 10, so it will go into bucket B2. The last two bits of 7 are 11, so it will go into B3.



Insert key 9 with hash address 10001 into the above structure:

- Since key 9 has hash address 10001, it must go into the first bucket. But bucket B1 is full, so it will get split.
- The splitting will separate 5, 9 from 6 since last three bits of 5, 9 are 001, so it will go into bucket B1, and the last three bits of 6 are 101, so it will go into bucket B5.

- Keys 2 and 4 are still in B0. The record in B0 pointed by the 000 and 100 entry because last two bits of both the entry are 00.
- Keys 1 and 3 are still in B2. The record in B2 pointed by the 010 and 110 entry because last two bits of both the entry are 10.
- Key 7 are still in B3. The record in B3 pointed by the 111 and 011 entry because last two bits of both the entry are 11.



Advantages of dynamic hashing

- In this method, the performance does not decrease as the data grows in the system. It simply increases the size of memory to accommodate the data.
- In this method, memory is well utilized as it grows and shrinks with the data. There will not be any unused memory lying.
- This method is good for the dynamic database where data grows and shrinks frequently.

Disadvantages of dynamic hashing

- In this method, if the data size increases then the bucket size is also increased. These addresses of data will be maintained in the bucket address table. This is because the data address will keep changing as buckets grow and shrink. If there is a huge increase in data, maintaining the bucket address table becomes tedious.
- In this case, the bucket overflow situation will also occur. But it might take little time to reach this situation than static hashing.

Differences between B-Tree and B+ Tree

B-Tree	B+ Tree
All internal nodes and leaf nodes contain data pointers along with keys.	Only leaf nodes contain data pointers along with keys, internal nodes contain keys only.
There are no duplicate keys.	Duplicate keys are present in this, all internal nodes are also present at leaves.
Leaf nodes are not linked to each other.	Leaf nodes are linked to each other.
Sequential access of nodes is not possible.	All nodes are present at leaves, so sequential access is possible just like a linked list.
Searching for a key is slower.	Searching is faster.
For a particular number of entries, the height of the B-tree is larger.	The height of the B+ tree is lesser than B-tree for the same number of entries.

Unit - V

TRANSACTION MANAGEMENT: Transaction concept, ACID properties, Transaction state, concurrent execution. Recovery System: Storage structure, Recovery and atomicity, Log-Based Recovery, ARIES Recovery Technique and Remote Back systems.

Topic I: Transaction Management Concept:

Transactions are a set of operations used to perform a logical set of work. A transaction usually means that the data in the database has changed. One of the major uses of DBMS is to protect the user's data from system failures. It is done by ensuring that all the data is restored to a consistent state when the computer is restarted after a crash. The transaction is any one execution of the user program in a DBMS. Executing the same program multiple times will generate multiple transactions.

Example –

Transaction to be performed to withdraw cash from an ATM vestibule.

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'accont:

```
Open_Account(X)
Old_Balance = X.balance
New_Balance = Old_Balance - 800
X.balance = New_Balance
Close_Account(X)
```

Y's Account

```
Open_Account(Y)
Old_Balance = Y.balance
New_Balance = Old_Balance + 800
Y.balance = New_Balance
Close_Account(Y)
```

Operations of Transaction:

Following are the main operations of transaction:

Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.

Write(X): Write operation is used to write the value back to the database from the buffer.

Let's take an example to debit transaction from an account which consists of following operations:

```
R(X);  X =
X - 500;
W(X);
```

Let's assume the value of X before starting of the transaction is 4000.

- The first operation reads X's value from database and stores it in a buffer.
- The second operation will decrease the value of X by 500. So buffer will contain 3500.
- The third operation will write the buffer's value to the database. So X's final value will be 3500.

But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

For example: If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.

To solve this problem, we have two important operations:

Commit: It is used to save the work done permanently.

Rollback: It is used to undo the work done.

Topic II: ACID Properties:

A **transaction** is a single logical unit of work that accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.

In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.

Atomicity:

By this, we mean that either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations.

—**Abort**: If a transaction aborts, changes made to the database are not visible.

—**Commit**: If a transaction commits, changes made are visible.

Atomicity is also known as the 'All or nothing rule'.

Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.

Before: X : 500	Y: 200
Transaction T	
T1	T2
Read (X) X: = X – 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

If the transaction fails after completion of **T1** but before completion of **T2**.(say, after **write(X)** but before **write(Y)**), then the amount has been deducted from **X** but not added to **Y**. This results in an inconsistent database state. Therefore, the transaction must be executed in its entirety in order to ensure the correctness of the database state.

Consistency:

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database. Referring to the example above,

the total amount before and after the transaction must be maintained.

Total **before T** occurs = **500 + 200 = 700**.

Total **after T occurs = 400 + 300 = 700**.

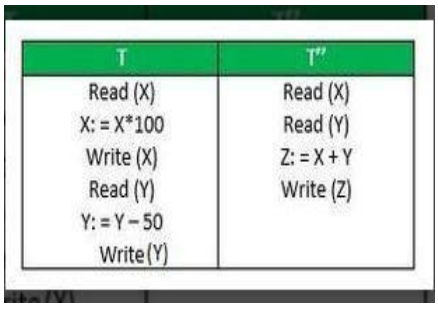
Therefore, the database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result, T is incomplete.

Isolation:

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of the database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

Let **X**= 500, **Y** = 500.

Consider two transactions **T** and **T".**



T	T''
Read (X) X: = X*100 Write (X) Read (Y) Y: = Y – 50 Write (Y)	Read (X) Read (Y) Z: = X + Y Write (Z)

Suppose **T** has been executed till **Read (Y)** and then **T''** starts. As a result, interleaving of operations takes place due to which **T''** reads the correct value of **X** but the incorrect value of **Y** and sum computed by

T'': (X+Y = 50, 000+500=50, 500)

is thus not consistent with the sum at end of the transaction:

T: (X+Y = 50, 000 + 450 = 50, 450).

This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take

place in isolation and changes should be visible only after they have been made to the main memory.

Durability:

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.

Some important points:

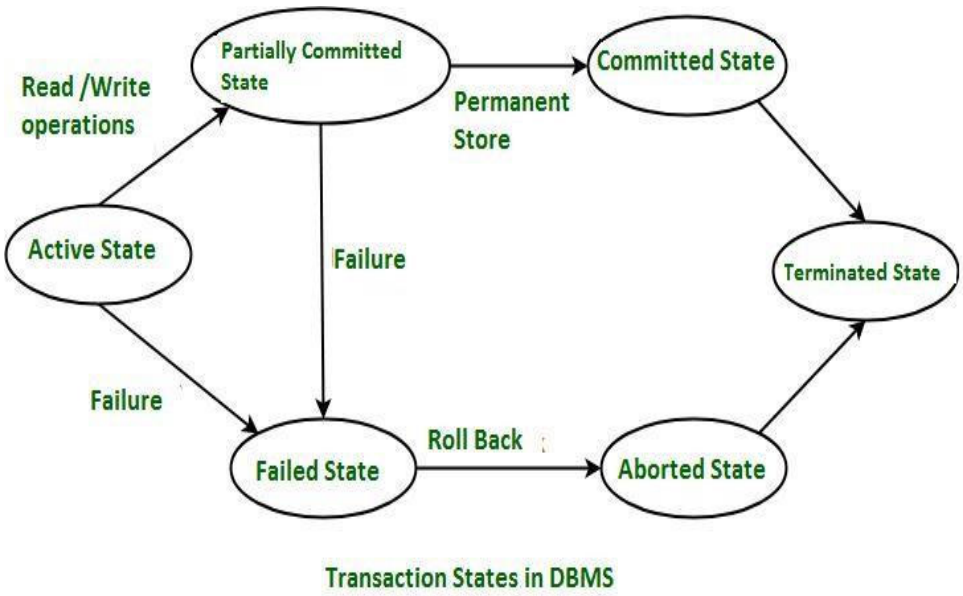
Property	**Responsibility for maintaining properties**
Atomicity	Transaction Manager
Consistency	Application programmer
Isolation	Concurrency Control Manager
Durability	Recovery Manager

The **ACID** properties, in totality, provide a mechanism to ensure the correctness and consistency of a database in a way such that each transaction is a group of operations that acts as a single unit, produces consistent results, acts in isolation from other operations, and updates that it makes are durably stored.

Topic III: Transaction states:

States through which a transaction goes during its lifetime. These are the states which tell about the current state of the Transaction and also tell how we will further do the processing in the transactions. These states govern the rules which decide the fate of the transaction whether it will commit or abort.

They also use **Transaction log.** Transaction log is a file maintain by recovery management component to record all the activities of the transaction. After commit is done transaction log file is removed.



Transaction States in DBMS

These are different types of Transaction States :

1. **Active State –**

   When the instructions of the transaction are running then the transaction is in active state. If all the 'read and write' operations are performed without any error then it goes to the "partially committed state"; if any instruction fails, it goes to the "failed state".

2. **Partially Committed –**

   After completion of all the read and write operation the changes are made in main memory or local buffer. If the changes are made permanent on the DataBase then the state will

change to “committed state” and in case of failure it will go to the “failed state”.

3. **Failed State –**
   When any instruction of the transaction fails, it goes to the “failed state” or if failure occurs in making a permanent change of data on Data Base.

4. **Aborted State –**
   After having any type of failure the transaction goes from “failed state” to “aborted state” and since in previous states, the changes are only made to local buffer or main memory and hence these changes are deleted or rolled-back.

5. **Committed State –**
   It is the state when the changes are made permanent on the Data Base and the transaction is complete and therefore terminated in the “terminated state”.

6. **Terminated State –**
   If there isn’t any roll-back or the transaction comes from the “committed state”, then the system is consistent and ready for new transaction and the old transaction is terminated.

Topic 4: Concurrent Execution:

Concurrency Control is the management procedure that is required for controlling concurrent execution of the operations that take place on a database.

But before knowing about concurrency control, we should know about concurrent execution.

Concurrent Execution in DBMS

- In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database. It means that the same database is executed simultaneously on a multi-user system by different users.

- While working on the database transactions, there occurs the requirement of using the database by multiple users for performing different operations, and in that case, concurrent execution of the database is performed.
- The thing is that the simultaneous execution that is performed should be done in an interleaved manner, and no operation should affect the other executing operations, thus maintaining the consistency of the database. Thus, on making the concurrent execution of the transaction operations, there occur several challenging problems that need to be solved.

Problems with Concurrent Execution:

In a database transaction, the two main operations are **READ** and **WRITE** operations. So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, and the data may become inconsistent. So, the following problems occur with the Concurrent Execution of the operations:

Problem 1: Lost Update Problems (W - W Conflict)

The problem occurs when two different database transactions perform the read/write operations on the same database items in an interleaved manner (i.e., concurrent execution) that makes the values of the items incorrect hence making the database inconsistent.

For example:

Consider the below diagram where two transactions T_X and T_Y, are performed on the same account A where the balance of account A is $300.

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A - 50	
t_3	—	READ (A)
t_4	—	A = A + 100
t_5	—	—
t_6	WRITE (A)	—
t_7		WRITE (A)

LOST UPDATE PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300 (only read).
- At time t2, transaction T_X deducts $50 from account A that becomes $250 (only deducted and not updated/write).
- Alternately, at time t3, transaction T_Y reads the value of account A that will be $300 only because T_X didn't update the value yet.
- At time t4, transaction T_Y adds $100 to account A that becomes $400 (only added but not updated/write).
- At time t6, transaction T_X writes the value of account A that will be updated as $250 only, as T_Y didn't update the value yet.
- Similarly, at time t7, transaction T_Y writes the values of account A, so it will write as done at time t4 that will be $400. It means the value written by T_X is lost, i.e., $250 is lost.

Hence data becomes incorrect, and database sets to inconsistent.

Problem 2: Dirty Read Problems (W-R Conflict)

The dirty read problem occurs *when one transaction updates an item of the database, and somehow the transaction fails, and before the data gets rollback, the updated database item is accessed by another transaction. There comes the Read-Write Conflict between both transactions.*

For example:

Consider two transactions T_X and T_Y in the below diagram performing read/write operations on account A where the available balance in account A is $300:

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A + 50	—
t_3	WRITE (A)	—
t_4	—	READ (A)
t_5	SERVER DOWN ROLLBACK	—

DIRTY READ PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300.

- At time t2, transaction T_X adds $50 to account A that becomes $350.
- At time t3, transaction T_X writes the updated value in account A, i.e., $350.
- Then at time t4, transaction T_Y reads account A that will be read as $350.
- Then at time t5, transaction T_X rollbacks due to server problem, and the value changes back to $300 (as initially).
- But the value for account A remains $350 for transaction T_Y as committed, which is the dirty read and therefore known as the Dirty Read Problem.

Problem 3: 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.

Chapter 2: Recovery System

Topic1: Storage Structure:

A database system provides an ultimate view of the stored data. However, data in the form of bits, bytes get stored in different storage devices.

In this section, we will take an overview of various types of storage devices that are used for accessing and storing data.

Storage Structure

We have already described the storage system. In brief, the storage structure can be divided into two categories −

- **Volatile storage** − As the name suggests, a volatile storage cannot survive system crashes. Volatile storage devices are placed very close to the CPU; normally they are embedded onto the chipset itself. For example, main memory and cache memory are examples of volatile storage. They are fast but can store only a small amount of information.
- **Non-volatile storage** − These memories are made to survive system crashes. They are huge in data storage capacity, but slower in accessibility. Examples may include hard-disks, magnetic tapes, flash memory, and non-volatile (battery backed up) RAM.

Topic 2: Recovery And Atomicity

Recovery and Atomicity

When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items. Transactions are made of various operations, which are atomic in nature. But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.

When a DBMS recovers from a crash, it should maintain the following −

- It should check the states of all the transactions, which were being executed.
- A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
- It should check whether the transaction can be completed now or it needs to be rolled back.
- No transactions would be allowed to leave the DBMS in an inconsistent state.

There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction −

- Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
- Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

Topic 3: 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.

<Tn, Start>

When the transaction modifies the City from 'Noida' to 'Bangalore', then another log is written to the file.

<Tn, City, 'Noida', 'Bangalore' >

When the transaction is finished, then it writes another log to indicate the end of the transaction.

<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.

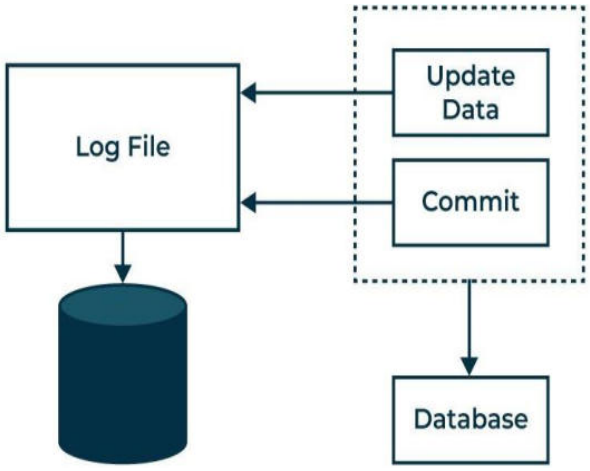
2. Immediate database modification:

- 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.

Topic 4 :

Log-Based Recovery in DBMS

Log-based recovery is a technique used in database management systems (DBMS) to ensure database consistency and durability in case of system failures. It relies on maintaining a **transaction log**, which records all changes made to the database, and uses this log to recover the database to a consistent state.



Phases of Log-Based Recovery

1. **Log Analysis**:
    - Determines which transactions were committed and which were active at the time of the failure.
    - Marks transactions as:
        - **Committed**: Changes will be redone.
        - **Active/Uncommitted**: Changes will be undone.
2. **Redo Phase**:
    - Reapplies changes of committed transactions to ensure durability.
    - Scans the log forward from the last checkpoint to apply the **after image**.
3. **Undo Phase**:
    - Rolls back the changes of uncommitted transactions to maintain atomicity.
    - Scans the log backward from the end and applies the **before image**.

Example of Log-Based Recovery

Scenario:

- A database contains a data item A with an initial value of 50.
- Two transactions, **T1** and **T2**, update the value of A:
    - **T1** changes A to 60.
    - **T2** changes A to 70.
- A system crash occurs after **T1 commits** but before **T2 commits**.

Log Entries:

Log Sequence Number (LSN)	**Transaction ID**	**Operation**	**Data Item**	**Before Image**	**After Image**	**Status**
1	T1	Update	A	50	60	Committed
2	T2	Update	A	60	70	Uncommitted

Recovery Process:

1. **Analysis Phase**:
    - The log is analyzed to determine that:
        - **T1** is committed.
        - **T2** is active/uncommitted at the time of the crash.

2. **Redo Phase**:
    - **T1's changes** (A = 60) are reapplied because **T1** was committed.
    - **T2's changes** (A = 70) are not redone because **T2** was not committed.
3. **Undo Phase**:
    - Rolls back **T2's changes** using the **before image** in the log, restoring A to 60.

Final State:

- After recovery, A = 60, reflecting the committed changes of **T1** and ignoring the uncommitted changes of **T2**.

Topic 4:

RIES Recovery Technique in DBMS

ARIES (Algorithm for Recovery and Isolation Exploiting Semantics) is a widely used recovery technique in database management systems (DBMS) for ensuring database consistency and fault tolerance. It efficiently handles failures by supporting atomicity and durability, two key properties of database transactions in the ACID model.

Steps of ARIES Recovery Technique

ARIES performs recovery in three phases:

1. Analysis Phase:

- Scans the log to identify transactions that were active at the time of failure.
- Determines the point where recovery should begin.
- Reconstructs transaction and dirty page information.

2. Redo Phase:

- Redoes all changes made by committed transactions since the last checkpoint.
- Ensures all committed changes are applied to the database.

3. Undo Phase:

- Reverts the effects of uncommitted transactions to maintain database consistency.
- Uses the log to roll back uncommitted changes in reverse order.

Key Features of Aries Recovery Algorithm in DBMS

1. **Write-Ahead Logging (WAL)**: This make sure that all changes are logged before they are applied to the database.

2. **Checkpointing**: To create a stable point in the database from which recovery can start.
3. **Three Phases of Recovery**: To ensure database recovery Analysis, Redo, and Undo phases are crucial.