GEETHANJALI INSTITUTE OF SCIENCE & TECHNOLOGY

Unit of USHODAYA EDUCATIONAL SOCIETY

An ISO 9001:2015 certified Institution: Recognized under Sec. 2(f)& 12(B) of UGC Act, 1956

3rd Mile, Bombay Highway, Gangavaram (V), Kovur(M), SPSR Nellore (Dt), Andhra Pradesh, India- 524137 Ph. No. 08622-212769, E-Mail: geethanjali@gist.edu.in, Website: www.gist.edu.in

(23A0512T) DATABASE MANAGEMENT SYSTEMS

SYLLABUS

UNIT I: Introduction: Database system, Characteristics (Database Vs File System), Database Users, Advantages of Database systems, Database applications. Brief introduction of different Data Models; Concepts of Schema, Instance and data independence; Three tier schema architecture for data independence; Database system structure, environment, Centralized and Client Server architecture for the database. **Entity Relationship Model:** Introduction, Representation of entities, attributes, entity set, relationship, relationship set, constraints, sub classes, super class, inheritance, specialization, generalization using ER Diagrams.

Unit II: Relational Model: Introduction to relational model, concepts of domain, attribute, tuple, relation, importance of null values, constraints (Domain, Key constraints, integrity constraints) and their importance, Relational Algebra, Relational Calculus. BASIC SQL: Simple Database schema, data types, table definitions (create, alter), different DML operations (insert, delete, update).

UNIT III: SQL: Basic SQL querying (select and project) using where clause, arithmetic & logical operations, SQL functions (Date and Time, Numeric, String conversion). Creating tables with relationship, implementation of key and integrity constraints, nested queries, sub queries, grouping, aggregation, ordering, implementation of different types of joins, view(updatable and non-updatable), relational set operations.

UNIT IV: Schema Refinement (Normalization): Purpose of Normalization or schema refinement, concept of functional dependency, normal forms based on functional dependency Lossless join and dependency preserving decomposition, (1NF, 2NF and 3 NF), concept of surrogate key, Boyce-Codd normal form(BCNF), MVD, Fourth normal form(4NF), Fifth Normal Form (5NF).

UNIT V:Transaction Concept: Transaction State, ACID properties, Concurrent Executions, Serializability, Recoverability, Implementation of Isolation, Testing for Serializability, lock based, time stamp based, optimistic, concurrency protocols, Deadlocks, Failure Classification, Storage, Recovery and Atomicity, Recovery algorithm.

Introduction to Indexing Techniques: B+ Trees, operations on B+Trees, Hash Based Indexing:

Textbooks:

1. Database System Concepts,5th edition, Silberschatz, Korth, Sudarsan, TMH (For Chapter 1 and Chapter5)

Reference Books:

1. Introduction to Database Systems, 8thedition, C J Date, Pearson.
2. Database Management System, 6th edition, Ramez Elmasri, Shamkant B. Navathe, Pearson
3. Database Principles Fundamentals of Design Implementation and Management, Corlos Coronel, Steven Morris, Peter Robb, Cengage Learning.
4. Database Management Systems, 3rd edition, Raghurama Krishnan, Johannes Gehrke, TMH

INTRODUCTION:

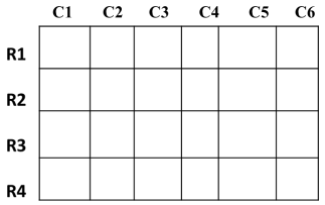
- Database is a collection of interrelated information of any organization or institute.

Example: The complete information of students such as roll no, name, branch, marks, address, etc.. is called as student database.

- Database does not support to store much information it has some limit. So now-a-days we are using data warehousing instead of database. Different database are combined into one is called as **Data Warehousing** it is 3D view. They are side view, top, front views in data warehousing.
- To combine two tables into one then it is called relational database. Here we can exchange into from one table to other. Where as normal data deals with one table. It is only one dimensional.
- It is very difficult to store the database into human memory. Hence we need system to store the database. When we need to retrieve, Alter any database information (or) To insert (or) Eliminate any information into database, Then we can do it through management. So, hence we call it as **Database Management System (DBMS).**

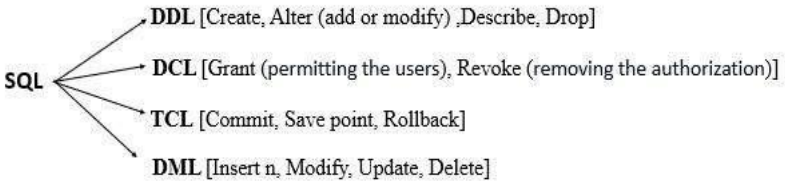
Database Vs File System

- File system is the old version (or) before version of DBMS. To create a file structure, we need to design dummy structure or skeleton. So that we need any high level languages such as C, C++, Java, .net etc.. If the user does not know the high level languages, then it is very difficult to create the dummy structure of file. The dummy structure of the file has rows and columns.

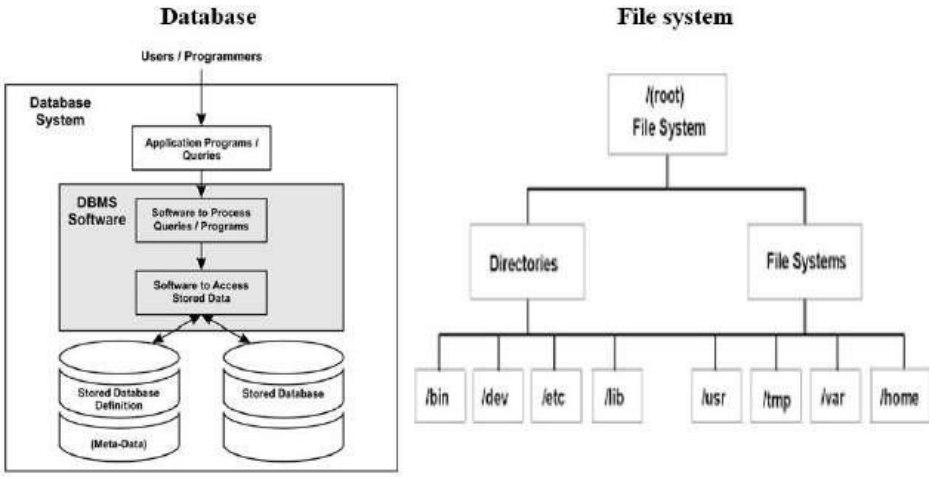


- The columns can also be known as **attribute or field**. Each and every row is called as **record**. Is nothing but collection of fields or attributes or columns.
- R1 = C1+C2+C3 (or) A1+A2+A3 (or) F1+F2+F3.
- Once created a dummy structure, Then we should name the columns. Then we need to insert the data in rows or record by using any high level language.
- DBMS always use a **Structure Query Language(SQL).** SQL is again partitioned as Data Definition Language **(DDL)**, Data Control Language **(DCL)** or Transition Control Language **(TCL),** Data Manipulation Language **(DML)**.

  Here a table has to be created first by using SQL Through single line syntax it is very simple than file system.



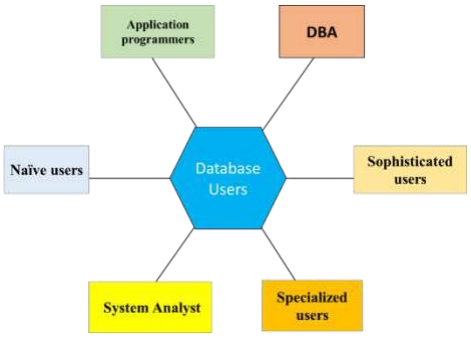
Features	Database	File System
Definition	A software system designed to store, manage, and retrieve structured data.	A method for organizing and storing files on a computer's storage medium.
Purpose	Store, manage, and retrieve structured data efficiently.	Store and manage files and directories.



Data Organization	Structured in tables with rows and columns, relationships between data.	Hierarchical structure of files and directories.
Data Access	Accessed through SQL queries.	Accessed through file paths and names.
Data Redundancy	Minimized through normalization.	Can have redundant data across multiple files.

DATABASE USERS:

- A primary goal of a database system is to retrieve information from and store new information into the database. People who work with a database can be categorized as database users or database administrators.
- There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.
- Users are differentiated by the way they expect to interact with the system:
- Application programmers – interact with system through DML calls.
- Database Administrator (DBA) – A Database Administrator (DBA) is a person/team who defines the schema and also controls the 3 levels of the database. The DBA will then create a new account ID and password for the user if he/she needs to access the database.
- Sophisticated users – form requests in a database query into language.
- Specialized users – write specialized database applications that do not fit the traditional data processing framework.
- System Analyst – A system Analyst is a user who analyzes the requirements of parametric end users. They check whether all the requirements of end users are satisfied.



- Naïve users – invoke one of the permanent application programs that have been written previously.

Examples, people accessing database over the web, bank tellers, clerical staff.

ADVANTAGES OF DATABASE USERS:

- **Data Integration or Data Integrity:** To add new information to file or Table, we use data integrity. If we forgot to add one column in table or file, then we use Data Integrity.
- **Data Security:** To provide the security to the stored data, password mechanism is acquired. Due to trail methods, others can easily hack the files of user and corrupt the files. So, the password mechanism has became a drawback in filesystem, So we shifted file system to DBMS for Data Security.
- **Data Isolation:** Data installation is the process of dividing larger application to formal attributes or columns first up to divide any larger table into small column, we go for data isolation.
- **Atomicity:** According to the file system, When your data is inserting into the table, If there is any power failures, Then after rebooting the system, it is not sure that we may regain the entered data. So, it takes much time to re enter the data. If any mechanical or hardware problems arises, then also it is very difficult to retain information. The file system does not have any type of Prevention, avoidance mechanisms. DBA (Database Administrator) detects the problems and automatically corrected.
- **Data Consistency:** DBMS maintains data consistency across the database, preventing conflicting updates.
- **Reduced Data Redundancy:** DBMS eliminates redundant data, saving storage space and reducing the risk of inconsistencies.
- **Efficient Data Access:** DBMS provides efficient data retrieval and manipulation mechanisms, allowing users to quickly access and analyze data.
- **Data Independence:** DBMS separates the logical and physical views of data, making data more portable and adaptable to changes in hardware or software. Enhanced Security and Control
- **Access Control:** DBMS allows administrators to define access permissions for different users, limiting access to authorized personnel.
- **Data Backup and Recovery**: DBMS offers backup and recovery mechanisms to safeguard data against accidental loss or corruption. Improved Collaboration and Decision-Making:
- **Data Sharing**: DBMS enables multiple users to access and share data simultaneously, facilitating collaboration and teamwork.
- **Data Analysis:** DBMS supports complex data analysis and reporting, providing valuable insights for decision-making.
- **Scalability:** DBMS can handle large amounts of data and support a growing number of users, making it suitable for organizations of all sizes.
- **Reduced Development Time:** DBMS provides a structured framework for data management, reducing the time and effort required to develop database applications.
- **Improved Data Quality**: DBMS helps maintain data quality through validation rules and data cleansing tools.

DATA BASE APPLICATIONS:

Database Management Systems (DBMS) are widely used in various sectors due to their ability to efficiently store, manage, and retrieve data. Here are some of the key applications of DBMS:

1. <u>Financial Systems:</u>

--**Banking:** Managing customer accounts, transactions, loans, and financial statements.

--**Insurance:** Storing and processing claims, policy information, and customer data.

--**Accounting:** Handling financial records, generating reports, and tracking expenses.

2. <u>Healthcare:</u>

--**Patient Records:** Storing medical history, test results, prescriptions, and appointments.
--**Hospital Management:** Managing patient admissions, discharges, and billing.
--**Research:** Analyzing medical data for research purposes.

3. <u>Retail:</u>

--**Inventory Management:** Tracking product stock levels and sales data.
--**Customer Relationship Management (CRM):** Storing customer information, preferences, and purchase history.
--**Point-of-Sale (POS) Systems:** Processing sales transactions and generating reports.

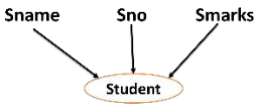
4. <u>Government:</u>

--**Citizen Records:** Maintaining records of citizens' personal information, addresses, and tax details.
--**Land Records:** Managing land ownership and property information.
--**Election Management:** Storing voter information and election results.

5. <u>Education</u>

--**Student Records:** Managing student information, grades, attendance, and transcripts.
--**Library Management:** Tracking books, journals, and other library resources.
--**Course Management:** Scheduling classes, assigning instructors, and managing course materials.

DATA MODELS:



- Data model is the process of describing of any data over database in graphical or pictorial representation.
- **Schema:** The process of combining different attributes into one form is called as schema.

Example: Student (sno: int, sname: varchar2, smarks: long);

- Schema is the overall database information.
- **Instance:** The process of storing any schema at the particular moment in the memory is called as instance.
- Data models are of four types:

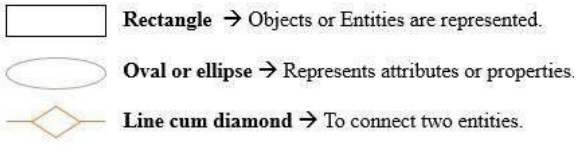
1. Object based logical model
2. Object oriented model
3. Record based model
4. Physical base model

➔ **Object based logical model:**

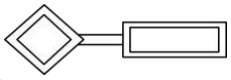
- Anything or entity that exists in real world is called **object**. Property of an entity is called **attribute.** The object can be represented in graphical or pictorial form. That form is called as diagram (Entity Relationship Model). By using ER model, we can connect more than one entity.

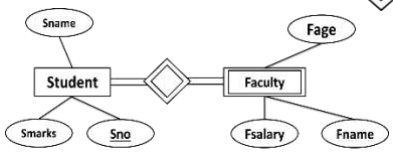
- ER diagram provides some symbols for users that are used for construct n number of ER diagrams.
- If the attribute is underlined with straight-line then it is called as primary attribute or strong attribute.

- If the attribute is underline with dotted-line then it is said to be weak attribute.



- To specify weak entity under ER diagram, we use (double diamond cum double line cum double rectangle).





This ER diagram shows the above rules.

- If we have seen primary key to the attribute, then the particular attribute must contain unique values. There must not be any repeatation of any value. It must not contain null or invalid values. Primary key attribute is a integrity constraint i.e., It does not accept null or invalid values into it.
- If the entity consists of strong attribute, Then the entity is stated as strong attribute.
- The attribute with dotted line is called weak attribute it is secondary key attribute in this duplication is possible. The entity with weak attribute is called as weak entity. • **Exercise:** construct year diagram for university.

✈ Object Oriented Model (OOM):

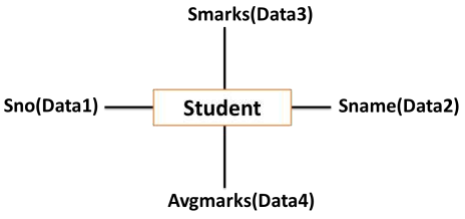
- C, C++, Java Can be used to construct OOM it is nothing but to combine the code into single.
- " The combination of data into a single one by using high level language like C, C++, Java Is known as **Object Oriented Model**". According to high level language the attributes are named as code or data.
- In Java the program will be follow classes.
- Method is nothing but a collection of statements and instructions.

EXAMPLE Java program for showing classes and Methods:

```
class Student {     int Sno;  String Sname;  float Smarks;
   float Savgmarks;

   // Method to set data      void setData() {
Sno = 1;
      Sname = "John";
      Smarks = 80.6f;
      Savgmarks = 90.1f;
   }

   // Method to display student details
   void displayData() {
      System.out.println("Student Number: " + Sno);
      System.out.println("Student Name: " + Sname);
```

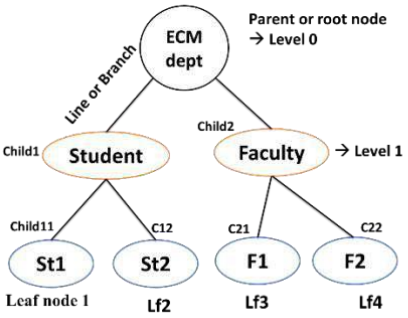


```
        System.out.println("Student Marks: " + Smarks);
        System.out.println("Student Average Marks: " + Savgmarks);
    }
}

public class Main {
    public static void main(String[ ] args) {
        // Create an object of the Student class
        Student student = new Student();

        // Set data for the student
student.setData();

        // Display the student data
        student.displayData();
    }
}
```



✈ Record based logical model:

Record is a collection of attributes or fields or columns. To combine more than one record, we need a table or relation. Relation is nothing but the arrangement of records in sequential or sorting order.

Record this sometimes called as row or tuple. Record based logical level always a fixed length record model. If first record length is 12, then on the records must maintain the same size as of the first record.

Record formats are of fixed and variable. Here we use only fixed length A record can be mathematically represented by the formula: **Record** = F1 + F2 + F3+ F4

Record = Sno + Sname + Smarks + Savg

The record based logical model is basically divided into 3 types they are:

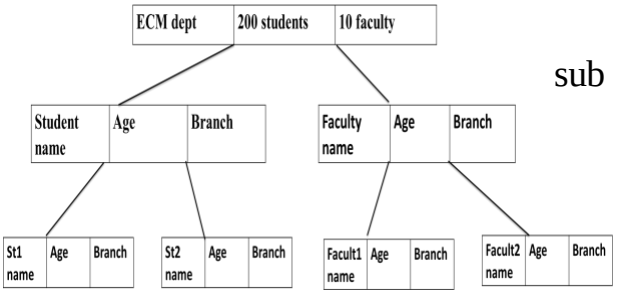
i) Hierarchical model ii) Network model iii) Relational model

1. Hierarchical model:

- It always follows tree structure. To construct this model initially we must have a root node or parent node. The root or parent node must have the child nodes and that child nodes and so on. To connect the notes with each other. We use branch or link or line. The root node is level zero. The chain nodes are of level one. As we go deep to the tree. The levels will be increasing. If a Hierarchical model is drawn, Then the nodes specified as the leaf node. As the hierarchical

  model. Is the desired model of record based logical model, We need to convert. Each node into a record.
- The total nodes can be converted as follows, into records.

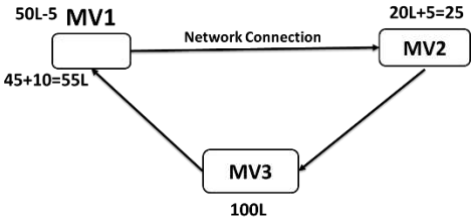
- The main drawbacks of hierarchal model are, if any information has to be loaded into sub node or subnodes, root node must give permission. This structure does not represent this pointers(i.e., direction of flow of information is not known). If the hierarchy is increases, then complexity also increases.



- To overcome this drawback, we switch to network model.

2. Network Model:

A network model is a collection of more than one record or node. It allows interconnections between records or nodes, forming a communication path known as a link, network connection, or interface.
Each link has pointers, which indicate the direction of data flow, making the relationships between records clear.

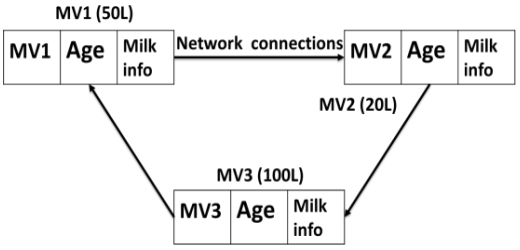


- **Example:** Consider 3 milk vendors i.e, MV1, MV2, MV3, contains
- MV1= 50L milk; MV2=20L; MV3=100L milk information.
- If MV1 Wants to exchange milk information with MV 2 of five liters,

then the connection path is laid in forward direction now MV2 has

20L+ 5L=25L milk information. Now MV3 send 10L to MV1 which has 45L now

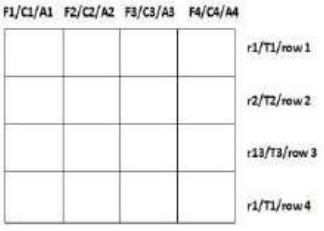
MV1 has 45+10=55L.



- The main drawback is, If the nodes are increased then complexity or confusion will be increased.
- Here one to one and one to many communication can be achieved but not many to many..
- The information of network model can be converted to the records as shown in the picture.

3. Relational Model:

- Relation of rows and columns are collection of attributes or records combining number of records into single one of fixed length is called as relation. As the records are increased then there won't be complexity. Relational model maintains interface between a minimum number of two relations or more than two relations.
- If we want to connect two relations, Then there must be one common column.
- Example: let us consider two relations i.e, IVECM & IVCSE Both of them has a common column as SNO, IVECM is primary table and IVCSE as secondary table common column. Similarly, the secondary table common column is named as foreign key. Now the information exchange between IVECM & IVCSE. When attribute is named as foreign key, then there exists a possibility of inserting the same information more than once. If any information is modified in secondary table that won't affect primary table.



- If the secondary key/ foreign key consists of adverb called as "**References"** keyword. If the foreign key wants to refer the original data from primary key, then we use reference keyword. To construct the 1st table i.e, IVECM

SQL> Create Table IVECM(SNO number(5) Primary Key, Sname varchar2(7), Smarks number(4, 1));

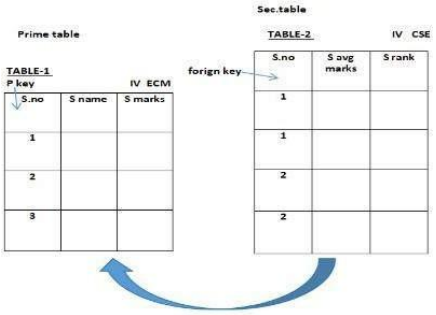
To create a 2nd table

SQL> Create Table IVCSE(SNO number(5) References IVECM(SNO), SAvgmarks number(6,2), Srank number(3));

- If we want to extend the relations, we can create communication between any tables. Table3 , (if Savg marks is common column in Table2, Table3).

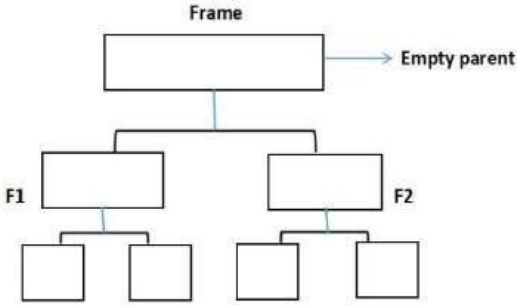
SQL> Create Table IVECE(Sno number (5) , Savg marks number (6,2) reference IVCSE(savg marks), Spercentage number (4,2));

- Only one table will be as primary table & primary key but foreign keys may be more than 1 each table must have only one primary key.



4. Physical Based Model:

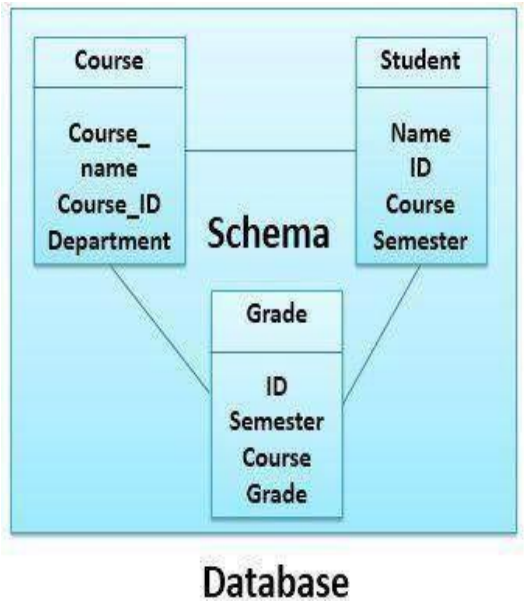
- It is process of describing no of frames implemented in physical level. This model consists of hierarchy. It accepts empty parent or parent. This model concerns of empty frames. The physical level accepts any type of data given by users. It accepts even null values of dummy models. The physical level sends the data to physical based model. As this model has empty frames, it can easily accept any type of data. The physical based model supports frame model Unified Model.
- Frame model has dummy frames in it. The physical based model is represented mostly by frame model.
- Unified model has use case diagrams, UML Diagrams. It is graphical model.



Concepts of Schema:

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

- **Physical schema:** The design of a database at physical level is called physical schema, how the data stored in blocks of storage is described at this level.
- **Logical schema:** Design of database at logical level is called logical schema, programmers and database administrators work at this level, at this level data can be described as certain types of data records gets stored in data structures, however the internal details such as implementation of data structure is hidden at this level (available at physical level).
- **View schema:** Design of database at view level is called view schema. This generally describes end user interaction with database systems.

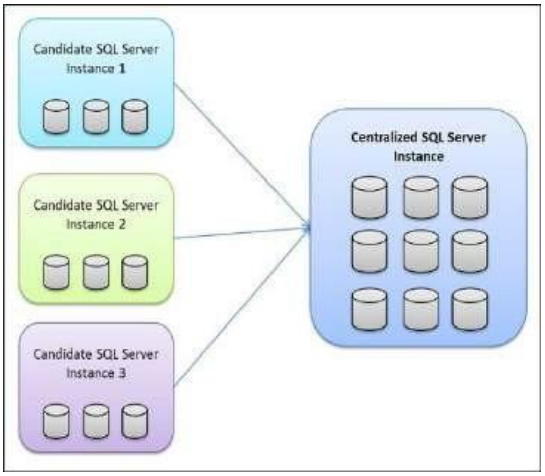


Instance:

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

1. Database Instance:

- Represents the entire state of the database at a specific point in time.
- Includes data in all tables, relationships between tables, and any other database objects.
- Can change dynamically as data is added, modified, or deleted.



2. Table Instance:

- A specific instance of a table within a database.
- Represents the data currently stored in a particular table.
- Changes as rows are inserted, updated, or deleted.

3. Attribute Instance:

- The value of an attribute for a specific record in a table.
- Can change independently within a table instance.

It's important to note that these are not distinct categories but rather different levels of granularity within a database instance. They are interconnected and form the overall state of the database.

Additionally, you might encounter the concept of "instance" in the context of objectoriented databases:

Object Instance: A specific object created from a class definition.
Class Instance: A specific class that is an instance of a meta class.

DATA INDEPENDENCE:

It is defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

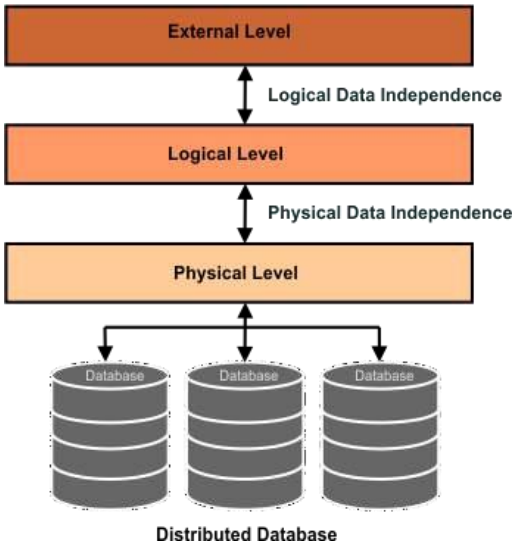
1. Logical Data Independence, 2. Physical Data Independence

Logical Data Independence:

- The ability to change the conceptual schema without impacting external schemas or application programs.
- Enables structural and constraint changes without impacting applications.
- More challenging to achieve than physical data independence.
- Requires DBMS to handle mappings between schema levels.
- Catalog plays a crucial role in managing mapping information.

Physical Data Independence:

- Ability to change the internal schema without affecting the conceptual or
- external schemas.
- Improves performance by optimizing physical storage and access paths.
- Does not require changes to application programs.
- Achieved by hiding physical details from users.



Key Difference:

- Physical data independence focuses on physical storage and access optimization.
- Logical data independence focuses on structural and semantic changes to the data.

Three Tier Schema Architecture for Data Independence:

- Three of the four important characteristics of the database approach are:

The external: The external or view level includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group

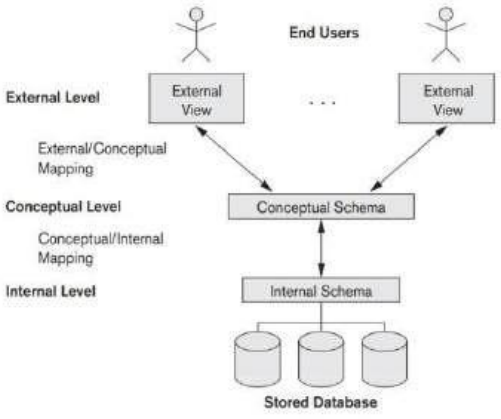
is interested in and hides the rest of the database from that user group. As in the previous level, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level data model.

(1) Use of a catalog to store the database description (schema) so as to make it self describing.

(2) Insulation of programs and data (program-data and program-operation independence

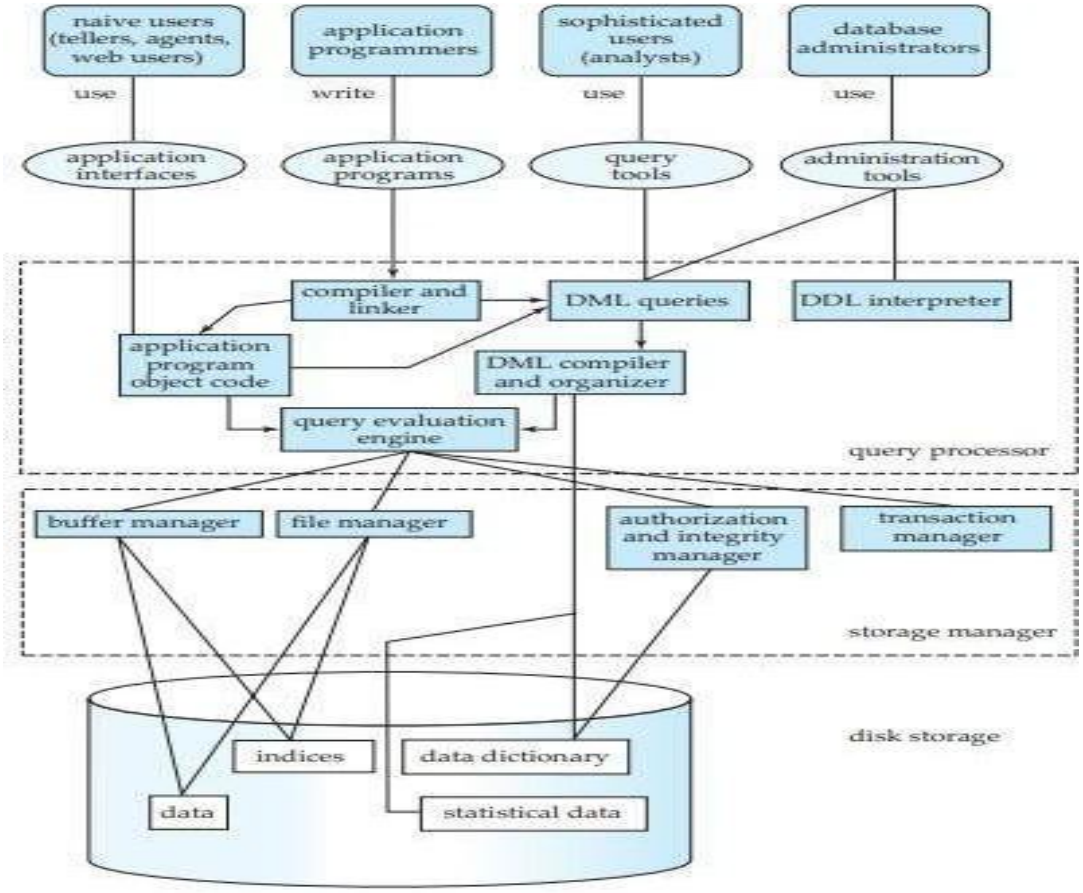
(3) Support of multiple user views. In this section we specify an architecture for database systems, called the three-schema architecture.

- The goal of the three-schema architecture, illustrated in Figure, is to separate the user applications from the physical database. In this architecture, schemas can be defined at the following three levels:
- **1.The internal level:** The internal level has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
- **2. The conceptual level:** The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. Usually, a representational data model is used to describe the conceptual schema when a database system is implemented. This implementation conceptual schema is often based on a conceptual schema design in a high-level data model.
- **3. The external:** The external or view level includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. As in the previous level, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level data model.
- The three-schema architecture is a convenient tool with which the user can visualize the schema levels in a database system. Most DBMSs do not separate the three levels completely and explicitly, but support the three schema architecture to some extent. Some older DBMSs may include physical-level details in the conceptual schema. The three-level ANSI (**American National Standards Institute**) architecture has an important place in database technology development because it clearly separates the users' external level, the database's conceptual level, and the internal storage level for designing a database. It is very much applicable in the design of DBMSs, even today. In most DBMSs that support user views, external schemas are



specified in the same data model that describes the conceptual-level information (for example, a relational DBMS like Oracle uses SQL for this). Some DBMSs allow different data models to be used at the conceptual and external levels.

✈ DATABASE SYSTEM STRUCTURE:



DATABASE SYSTEM ARCHITECTURE:

- **Centralized vs. Client-Server:** Database systems can be centralized or distributed across multiple machines.
- **Storage Manager:** Handles data storage and retrieval.
- **Query Processor:** Processes user queries and updates.
- **Translation:** Converts high-level queries into low-level operations.

DATABASE APPLICATION ARCHITECTURES:

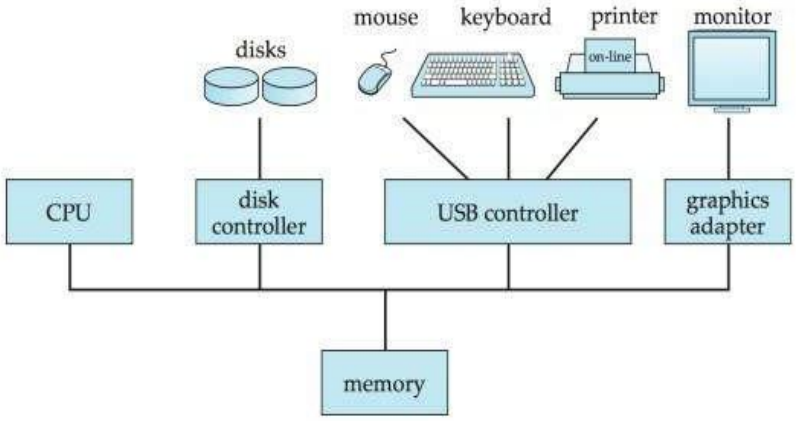
- **Two-Tier Architecture:** Application directly interacts with the database server.
- **Three-Tier Architecture:** Application server mediates between the client and database server.
- **Three-Tier Advantages:** Improved scalability, security, and maintainability.
- **Three-Tier Use Cases:** Large-scale and web-based applications.

Environmental Factors For Database System Architecture:

Category	Factors	Description
User Environment	Naive Users	These users (tellers, agents, web users) interact with the system through application interfaces.
	Sophisticated Users (Analysts)	Availability and usability of query tools, query engine performance, support for advanced analytics.
	Database Administrators	Availability and functionality of admin tools, complexity of tasks, documentation /training.
Software Environment	Application Programs	Supported languages, libraries/APIs, compiler/linker performance.
	Database Management System (DBMS)	Type (relational, NoSQL), features, performance, reliability.
	Query Processor	Efficiency, accuracy.
Hardware Environment	Storage Manager	Storage device type/capacity, data access speed, storage space.
	Buffer Manager	Available memory, allocation strategies, data caching efficiency.
	File Manager	File system organization, data access methods, file operation efficiency.
Data Environment	Data Dictionary	Metadata accuracy/completeness, ease of querying/updating, DBMS support.
	Data	Volume, types, access patterns.
Security and Authorization Environment	Authorization and Integrity Manager	Security measures effectiveness, user permission management ease, data validation robustness.
Additional Considerations	Transaction Management	Transaction processing efficiency, concurrency support, recovery mechanisms.
	Indices	Indexing techniques, index maintenance efficiency, storage space impact.
	Statistical Data	Availability, accuracy, use for optimization

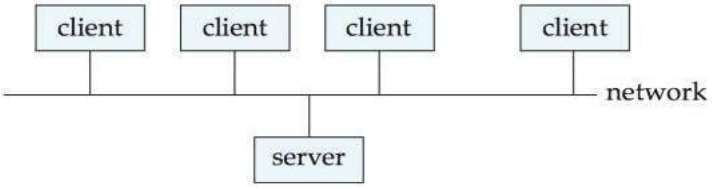
✈ Centralized Systems Architecture for the Database:

- Run on a single computer system and do not interact with other computer systems.
- **General-purpose computer system:** one to a few CPUs and a number of device controllers that are connected through a common bus that provides access to shared memory.
- **Single-user system** (e.g., personal computer or workstation): desk-top unit, single user, usually has only one CPU and one or two hard disks; the OS may support only one user.
- **Multi-user system:** more disks, more memory, multiple CPUs, and a multi-user OS. Serve a large number of users who are connected to the system via terminals. Often called server systems.



Client Server Architecture for the Database:

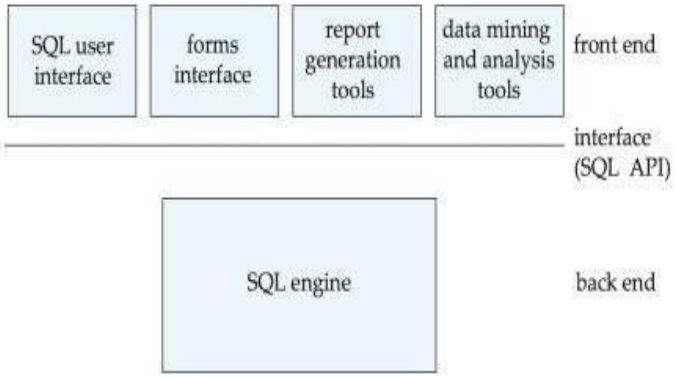
- Server systems satisfy requests generated at client systems, whose general structure is shown below:



 Database functionality can be divided into:

- **Back-end:** manages access structures, query evaluation and optimization, concurrency control and recovery.
- **Front-end:** consists of tools such as forms, report-writers, and graphical user interface facilities.

The interface between the front-end and the back-end is through SQL or through an application program interface.



PART 2: ENTITY RELATIONSHIP MODEL:

Entity Relationship Model:

In the entity-relationship model (or E /R model), the structure of data is represented graphically, as an "entity-relationship diagram," using three principal element types:

1. Entity sets,
2. Attributes, 3. Relationships.

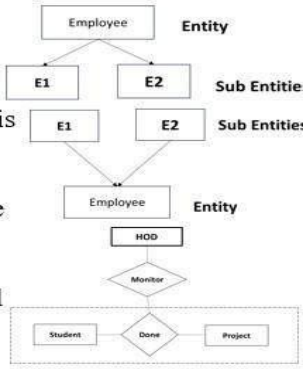
Entity-Relationship Diagrams

An E /R diagram is a graph representing entity sets, attributes, and relationships. Elements of each of these kinds are represented by nodes of the graph, and we use a special shape of node to indicate the kind, as follows:

- Entity sets are represented by rectangles.
- Attributes are represented by ovals.
- Relationships are represented by diamonds.

REPRESENTATION OF ENTITIES:

To create ER diagram some concepts are involved. Line

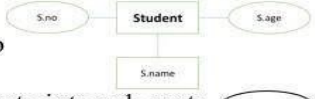


- **Specialization:** It is the process of dividing the available entity into different sub entities. It follows the property of inheritance. The sub entities inheritance the properties of super entity or large entity.
- **Generalization:** It is a process of deriving the sub entities to super entities. It is the reverse process of specialization.
- **Aggregation:** To represent the aggregation in ER Diagram at that time we represent it in **dashed box**. To get the aggregation of one or more attributes we use aggregation operation by this we can reduce the number of attributes into single attribute.

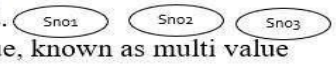
Suppose students want to show the project report to the project guide then that all information will be kept in the dotted lines is that it is hidden. →

- **Entity:** Anything in the real world is called as an entity or object.

- **Attribute:** Property of an entity is called as attribute. →



- **Simple Attribute:** The attribute which is not divided into different parts this is a single attribute.
- **Composite Attribute:** It is the process of available attribute into sub parts.



- **Multi value attribute:** The attribute which consists of more than one value, known as multi value attribute.

- **ER Diagrams:** When a user collects any data base, that has to be represented in either graphical or pictorial format. So, it is very easy to implement ER diagram.
- The ER diagram mainly has Entity. Entity means any thing in real world. To construct ER diagram, we must have some symbols. They are:

i) rectangle → Entity **Example:** Student

ii) oval → Property of an Entity (attribute)

iii) line → Communication path between entity and attribute.

iv) diamond → To create interface between 2 Entities.

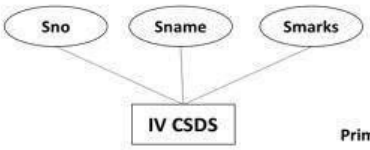
v) Line under attribute → Strong Entity or Primary Entity.

vi) Dotted line under attribute → Weak Entity or secondary attribute.

vii) →Double diamond cum double line cum double rectangle is used to represent weak entity.

- **Key attribute:** it is a collection of number of attributes this represent each and every rows in the unique manner.

➢ The key attribute is also called as primary attribute or primary key attribute.



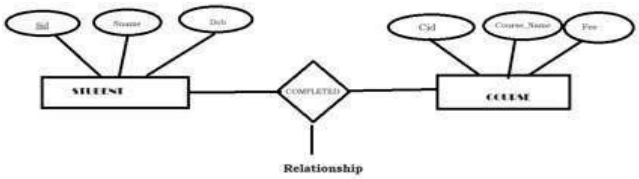
Primary Key <--

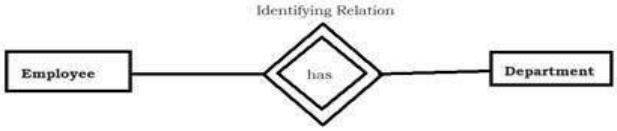
Sno	Sname	Smarks
1	X	12
2	Y	12
3	Z	12

❑ Relationship:

- A relationship is an association between the instances of one or more entity types. In an ER Diagram the relationship is represented by a diamond symbol containing the name of the relationship.
- These are two types 1. Relationship 2.Has/ Identity relationship.

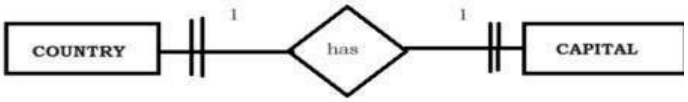
1. 2.





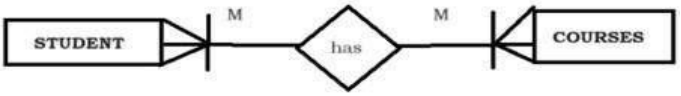
❑ Relationship Classification:

One to One Relationship (1:1): In **1:1** relationship one entity can be related only one other entity. **Ex:** Consider **COUNTRY** has only one **CAPITAL.**



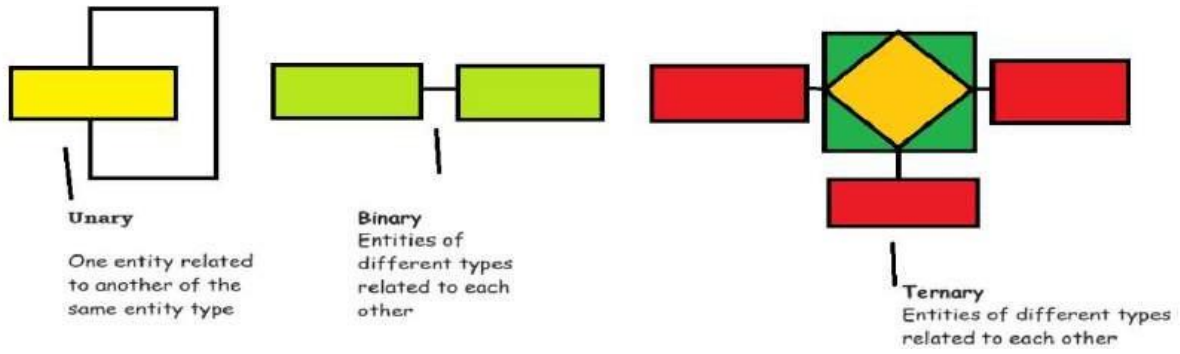
One to Many Relationship(1:M): **1:M** relationship is the standard type in the relational model. Consider AUTHOR wrote **BOOKS.**

Many to Many Relationship(M:M): In M:M relationship is not supported directly in the relational environment. Therefore M:N relationships can be implemented by creating a new entity in 1:M relationships.



✈ Relationship Set:

- A relationship set is a collection of similar relationships. Each relationship in the set connects the same types of entities. For example, if you have a relationship set called "Enrollment," it would consist of all the relationships that link students to the courses they are enrolled in. In ER diagrams, there are 3 most common relationship degrees are available. They are:
  1. Unary Relationship.
  2. Binary Relationship.
  3. Ternary Relationship.



✈ **Constraints:**

- **An integrity constraint is a rule that cannot be violated by the user. The constraints are used to prevent invalid data entry into the table. It enforces rules for the columns in a table. In SQL the integrity constraints are classified into 3 types. They are**

1) Domain integrity constraints

2) Entity Integrity constraints 3) Referential integrity constraints

1. Domain Integrity Constraints:

- A domain is a set of values that may assign to a column. Domain Integrity Constraints are 2 types. They are
- i) Not Null
- ii) Check

i) Not Null:

- The value which is absent is called NULL. The NOT NULL constraint is used to avoid NULL values into a column of a table. The NOT NULL constraint allows duplicate values.
- **Eg: SQL**> Create table student(Sno number(10) NOT NULL, sname varchar2(20));
  In the above example, Sno column should not allow null values.

ii) Check:

- The Check constraint is used to specify a particular condition to a column.
- It checks the condition satisfied or not.
  SQL> Create table student(Sno number(10), Sname varchar2(20) Sfee number(8,2) CHECK (Sfee>10000));
- In the above example, the column 'Sfee' is allows only the values, which are greater than 10000.

2. Entity Integrity Constraints:

- The entity integrity constraints are used to avoid invalid data into column of a table. There are 2 types

i) Unique

ii) Primary Key

i) Unique:

- It is used to prevent duplicate values to a particular column. i.e., It does not allow duplicate values, but allows NULL values.

Eg: SQL> Create table emp(E_no number(3) UNIQUE, Ename varchar2(30), Deptno number(5));

ii) Primary Key:

- A filed (or) combination of fields used to identify a single record will be called as Primary key. It does not allow duplicate values and null values.

Eg: SQL> Create table emp(E_no number(3) PRIMARY KEY, Ename varchar2(30), Deptno number(5));

3. Referential Integrity Constraints:

- The referential integrity constraint is used to validation of a rule between two tables. A filed which references the primary key of another table will be called as Foreign key.
- To implement this type of constraint the Master (parent) table must contain primary key and the same column in the Detailed (child) table as a reference of primary key known as foreign key. **Eg: SQL>** Create table Dept(Deptno number(5) PRIMARY KEY)

3. Master Table:

Eg: SQL> Create table Dept(Deptno number(5) PRIMARY KEY
Dname varchar2(20), Locaction varchar2(10));

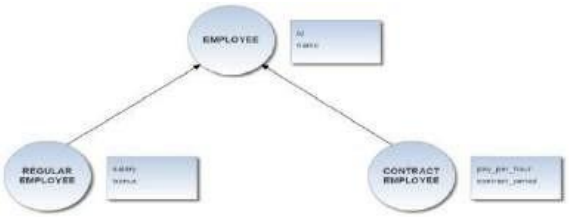
Detailed Table:

Eg: SQL> Create table Emp(E_no number(3) PRIMARY KEY, Ename varchar2(20), Deptno number(5) references Dept(Deptno));

The records from the Master table will not be deleted until the corresponding records are deleted from the detail table. Similarly new records cannot be inserted into the detail table, until the corresponding values are inserted in the master table.

✈ Inheritance:

Inheritance in DBMS refers to the concept derived from object-oriented programming where a child entity inherits properties and behaviors from a parent entity. This is primarily used in object-oriented and objectrelational databases.



Single level Inheritance:

Single Table Inheritance, also known as Table Per Hierarchy(TPH), stores all entities—superclass and subclass—in a single table. Every record in the table has a unique marker, known as a discriminator, that designates the particular kind of entity it represents. Because fewer tables and intricate connections are required, this method makes data storage and retrieval easier. But it could lead to sparse tables with a lot of null values for characteristics that don't apply to some kinds of entities.

Class Table Inheritance:

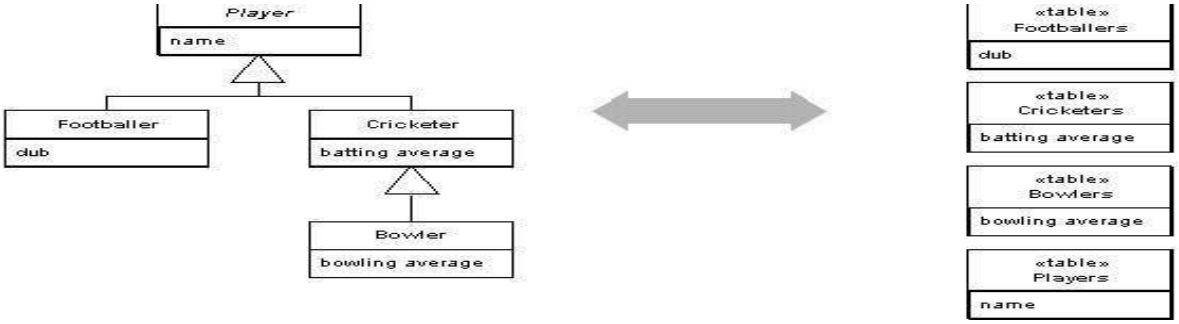


Table per Type (TPT), another name for Class Table Inheritance, is the process of making a separate table for every entity type, including the superclass and all of its subclasses. Only the properties pertinent to that particular entity type are contained in each table, and foreign keys are used to form associations between the tables. By cutting down on redundancy, this method preserves data integrity while providing improved data organization. To extract data from many tables, however, more intricate queries requiring joins can be needed.

Sub classes and Super classes

The Enhanced Entity Relationship Model contains all the features of the Entity Relationship Model. In addition to all that, it also contains features of Subclasses, Super classes. A subclass is a class derived from the superclass. It inherits the properties of the superclass and also contains attributes of its own. An example is:

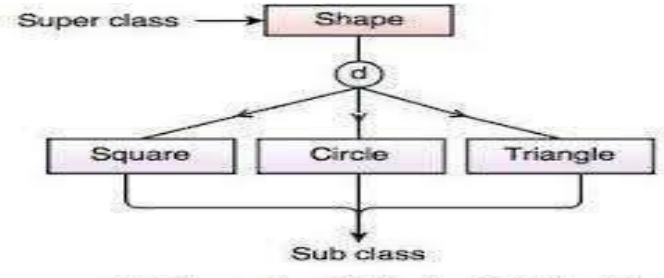


Fig. Super class/Sub class Relationship

Square, Circle and Triangle are the subclasses of the superclass shape. They all inherit common attributes from shape such as number of sides etc. A superclass is the class from which many subclasses can be created. The subclasses inherit the characteristics of a superclass. The superclass is also known as the parent class or base class.

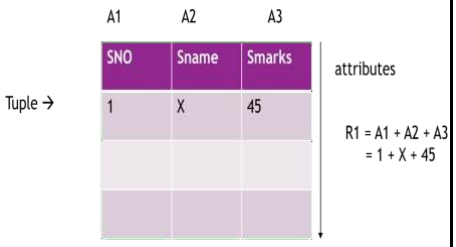
In the above example, Shape is the Superclass and its subclasses are Square, Circle and Triangle.

Unit II: Relational Model

Introduction to relational model, concepts of domain, attribute, tuple, relation, importance of null values, constraints (Domain, Key constraints, integrity constraints) and their importance, Relational Algebra, Relational Calculus. BASIC SQL: Simple Database schema, data types, table definitions (create, alter), different DML operations (insert, delete, update).

Introduction to relational model:

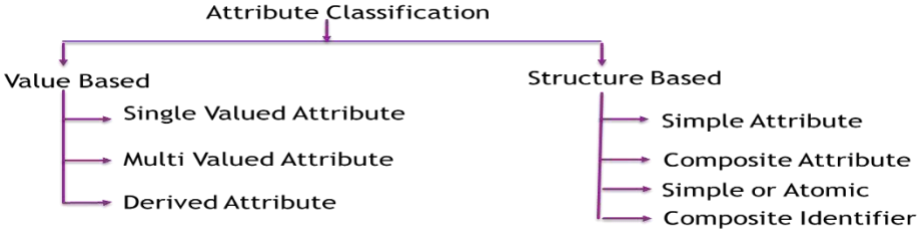
- **Relation**: Combination or rows/ records and columns / attributes is called as Relation. Entity can also be called as object or relation. If the entity is having any properties, then those properties has to be substituted into the attributes.
  Sometimes, a row can be called as Record or Tuple.



- Relational Database is the process of connecting more than one table.
- Relational model was developed by **"codd"** in **1970s** it was developed by IBM organization. IBM uses languages like Fox pro, Sybase, DB/2, MS-access, oracle etc.. and several different versions of language. By using these languages IBM can be Update, Modify the tables. Before inserting the data into relation some required conditions has to be applied. Those conditions may be done at the entry level itself. After insertion it is not possible to frame the conditions to the relation.
- To restrict any relation, IBM developed a phenomenon/ concept called Integrity constraint over relations.

Attribute:

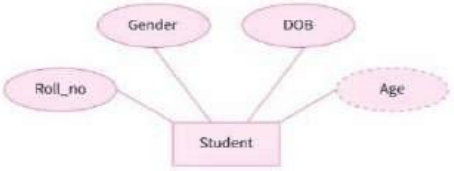
- An attribute is a property or characteristic of an entity type. Each entity type has a set of attributes. The attribute is represented in a Ellipse Symbol.
- **ATTRIBUTE CLASSIFICATION:**



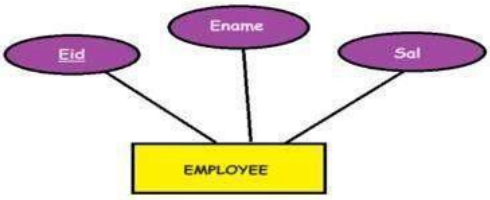
- **Simple Attribute:** Single-valued attributes are those attributes that consist of a

single value for each entity instance and can't store more than one value. The value of these single-valued attributes always remains the same, just like the name of a person.

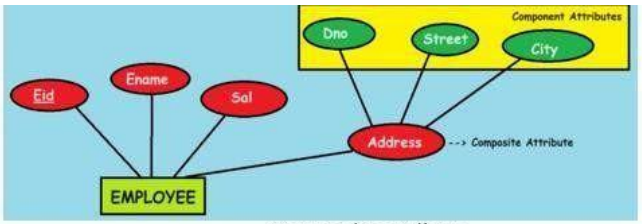
- **Composite Attribute:** An attribute that is a combination of other attributes is known as composite attribute. For example, In employee entity, the employee address is a composite attribute as an address is composed of other attributes such as pin code, state, country. In the given ER diagram, student address is a composite attribute. It contains CITY, PIN, DOOR#, STREET, and STATE. In the STUDENT table, these attributes can merge as an individual column.
- **Derived Attribute**: If an attribute's value can be derived from the values of other attributes, then the attribute is derivable, and is said to be a derived attribute. Derived attributes are shown with a **dotted lined**.



- In the STUDENT table, Age is the derived attribute. It can be calculated at any point of time by calculating the difference between current date and Date of Birth.



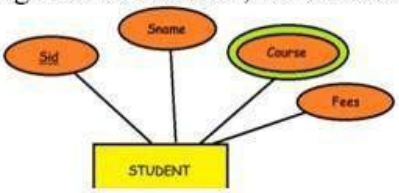
Simple attribute



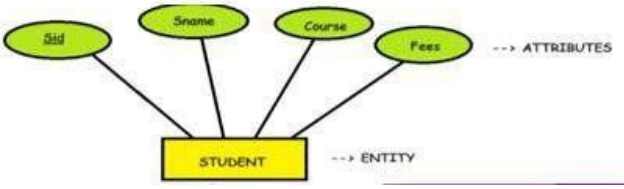
Composite attribute

- **Multi valued Attribute:** Multivalued attributes in DBMS are used when we need to store one or more values for an attribute. The name of the attribute is contained inside a double oval shape that is used to symbolize a multivalued attribute.

In the student table, a hobby is a multi valued attribute. So it is not possible to represent multiple values in a single column of STUDENT table. Hence we create a table STUD_HOBBY with column name STUDENT_ID and HOBBY. Using both the column, we create a composite key.



- **Single Valued Attribute**: Those attributes which can have exactly one value are known as single valued attributes. They contain singular values, so more than one value is not allowed.
- In the STUDENT entity, STUDENT_NAME and STUDENT_ID form the column of STUDENT table. Similarly, COURSE_NAME and COURSE_ID form the column of COURSE table .



TUPLE:

A tuple, also known as a record or row, is a basic unit of data in a relational database management system (DBMS). A tuple represents a single instance of a relation, or table, in the database. Each tuple contains a set of values, or attributes, that correspond to the columns and rows.

EXAMPLE:

In an un-normalized relation, an "ORDER" relation may include attributes such as "order_id", "customer_id", "product_id", and "quantity". In the process of normalization, the relation may be broken down into two separate relations, one called "orders" containing attributes such as "order_id" and "customer_id", and another called "order_details" containing attributes such as "product_id" and "quantity".

Sno.	order_id	customer_id	product_id	quantity
0	A	1	AAA1	5
1	B	2	BBB1	6
2	C	3	CCC1	7

Types of Tuples:

There are two types of tuples in a database management system:

- **Physical Tuples:** Physical Tuples are the actual data stored in the storage media of a database. It is also known as a record or row.
- **Logical Tuples:** Logical Tuples are the data representation in memory, where data is temporarily stored before being written to disk or during a query operation.

CONCLUSION:

A tuple is a basic unit of data in a relational database management system. It represents a single instance of a relation and contains a set of values that correspond to the attributes of that relation. Tuples are used in the process of normalization, as well as querying a relational database, to retrieve specific data. They play a vital role in organizing and structuring data in a database, improving data integrity, and making data retrieval easy.

KEY CONSTRAINTS:

A 'Key Constraint' in computer science refers to a rule implemented by a database management system to ensure that each constraint is enforced by only one program module, such as the DBMS itself, stored procedures, triggers, or other executable modules.

Key Type	Definition
Primary Key	Uniquely identifies each record in a table.
Foreign Key	Establishes a relationship between tables.
Candidate Key	Alternative unique keys that could be primary keys.
Super Key	Combination of attributes that uniquely identify

Integrity constraints:

An integrity constraint is a rule that cannot be violated by the user. The constraints are used to prevent invalid data entry into the table. It enforces rules for the columns in a table. In SQL the integrity constraints are classified into 3 types. They are:

1) Domain integrity constraints
2) Entity Integrity constraints
3) Referential integrity constraints

1. Domain Integrity Constraints: A domain is a set of values that may assign to a column. Domain Integrity Constraints are 2 types. They are:

i) Not Null ii) Check

i) Not Null: The value which is absent is called NULL. The NOT NULL

constraint is used to avoid NULL values into a column of a table. The NOT NULL constraint allows duplicate values.

Eg: SQL> Create table student(Sno number(10) NOT NULL, Sname varchar2(20));

In the above example, Sno column should not allow null values.

ii) Check: The Check constraint is used to specify a particular condition to a column. It checks the condition satisfied or not.

Eg: **SQL>** Create table student(Sno number(10), Sname varchar2(20), Sfee number(8,2) CHECK (Sfee>10000));

In the above example, the column 'Sfee' is allows only the values, which are greater than 10000.

Entity Integrity Constraints: The entity integrity constraints are used to avoid invalid data into column of a table. There are 2 types

i) Unique ii) Primary Key

i) Unique: It is used to prevent duplicate values to a particular column. i.e., It does not allow, but allows NULL values.

Eg: SQL> Create table emp(E_no number(3) UNIQUE, Ename varchar2(30), Deptno number(5));

ii) Primary Key: A filed (or) combination of fields used to identify a single record will be called as Primary key. It does not allow duplicate values and null values.

Eg: SQL> Create table emp(E_no number(3) PRIMARY KEY, Ename varchar2(30), Deptno number(5));

3. Referential Integrity Constraints:

The referential integrity constraint is used to validation of a rule between two tables. A filed which references the primary key of another table will be called as Foreign key.

To implement this type of constraint the Master (parent) table must contain primary key and the same column in the Detailed (child) table as a reference of primary key known as foreign key.

Master Table:

Eg: SQL> Create table Dept(Deptno number(5) PRIMARY KEY, Dname varchar2(20), Locaction varchar2(10));

Detailed Table:

Eg: SQL> Create table Emp(E_no number(3) PRIMARY KEY, Ename varchar2(20), Deptno number(5) references Dept(Deptno));

The records from the Master table will not be deleted until the corresponding records are deleted from the detail table. Similarly new records cannot be inserted into the detail table, until the corresponding values are inserted in the master table.

Relational Algebra:

The Relational Algebra is a Procedural Query Language. It consists of a set of operations that takes one or two relations as input and produce a new relation as their result. The functional operations in the relational algebra are

1. Selection(σ)
2. Projection(Π)
3. Rename(p)
4. Cartesian Product(X)
5. Join Operations

1. **SELECTION Operation():** It is the process of retrieving any desired row from the relation/table. Selection operation is represented as 'σ' (sigma). Syntax for selection is:

$\sigma_{attribute\ name=value}$ (Relation name) (or) $\sigma_{Condition}$ (Table name or Relation name)

Eg: Query → $\sigma_{x=2}$ (STUDENTS); Query → $\sigma_{y=c}$ (STUDENTS);

O/P:

2	B	B

O/P:

3	C	14

→ Select X from STUDENTS where Y = C;

STUDENTS

X	Y	Z
1	A	12
2	B	13
3	C	14

ii) **Projection:** It is the process of selecting any no of columns from the relation / table.

→ It is represented as Π(pi).

→ Πattribute name1, attribute name 2, attribute n. (Relation / Table name).

→ Eg: Query = Π_x (STUDENTS)

→ O/P:

X
1
2
3

→ Select * from STUDENTS;

STUDENTS

X	Y	Z
1	A	12
2	B	13
3	C	14

Eg: Query = $\Pi_{x,z}$ (STUDENTS)

X	Z
1	12
2	13
3	14

iii) **Cross Product Operation:** It is the process of multiplying 2 relations into one relation. It is represented as 'X'. To multiply any relations, they must have one same column.

Eg:

STUDENTS

X	Y	Z
1	A	12
2	B	13
3	C	14

EMPLOYEES

X	B	C
1	Venkat	Fork
2	Likhith	Bell
3	Vishnu	Box

- By applying many to many relation, we get, the following tables.
- $R_{STUDENTS} \times R_{EMPLOYEES}$ →
- The result of the cross product operation has duplication values.

Cross product related output

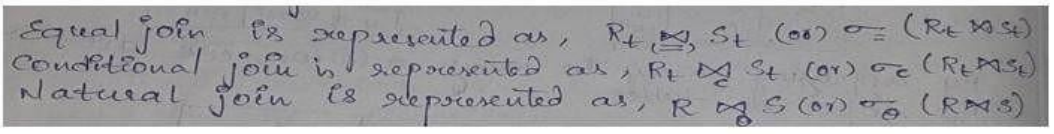
X	Y	Z	X	B	C
1	A	12	1	Venkat	Fork
1	A	12	2	Likhith	Bell
1	A	12	3	Vishnu	Box
2	B	13	1	Venkat	Fork
2	B	13	2	Likhith	Bell
2	B	13	3	Vishnu	Box
3	C	14	1	Venkat	Fork
3	C	14	2	Likhith	Bell
3	C	14	3	Vishnu	Box

- **Join Operation:** To avoid the duplication values from the cross product operation, we go for join operation. It is Symbolically as, '⋈'.
- From the O/P of above relation, the first row is original because it has same Sno. So, the other are duplicated records has to be eliminated.

1	A	12	1	Venkat	Fork
1	A	12	2	Likhith	Bell
1	A	12	3	Vishnu	Box

- So, the join operation O/P can be written as follows.
- Join operation are 3 types:

1) Natural join
2) Equal join
3) Conditional join



- **Union Operation:** It is represented as 'U'. We can union relations from a minimum of 2. Two relations, for which union operations has to be performed, must has same column (i.e, same column name, but the records/ tuples may vary from each other).
- **Eg:**

R

Sno	Smar
1	12
2	13
3	14
4	15

S

Sno	Percentage(Sp)
1	50
2	60
3	70
4	80
5	90

R U S →

R U S

Sno	Sm	Sp
1	12	50
2	13	60
3	14	70
4	15	80
5		90

When any record or column is empty, then the system by default entries Null value.

- **Rename:** It is the process of converting of old table name to another name. This is represented with " ρ" (lower case sigma). The syntax for rename operation is:

$\rho_{oldname=new\ name}$ (Relation name)

(ρ_{Temp} (Boats) ⋈ Reserves ⋈ Sailors)

- **Intersection:** It is represented as ' ∩ '. To perform this, we need minimum of 2 tables and those must contain same column, this is the process of extracting common touple values from both relations.
- Eg:

R

Sno	Smar
1	12
2	13
3	14
4	15

S

Sno	Percentage(Sp)
1	50
2	60
3	70
4	80
5	90

R ∩ S

Sno
1
2
3
4

Here only Sno is common column in both tables.

- **Subtraction or Set difference:**

It is represented as '-' (minus). A minimum of 2 tables with one same column is required.
'R-S' means tuples contain in 'R' but not 'S'.
'S-R' means tuples contain in 'S' but not in 'R'.

R

Sm
15
16
17
18

S

Sm
50
60
17
18

R-S

Sm
50
60

S-R

Sm
50
60

Relational Calcus:

▶ It is a non-procedural language. When user wants to get desired information get desired into from database without using any operation, then it is non-procedural language. Here we use different mathematical symbols instead of operation that are used in Relational Algebra.

▶ The mathematical symbols are, <, >, =, ∀ (for all), ∃ (There exists), E(Belongs) ¬ (not), ^(and), V (or), <=, >=. ▶ There are 2 types in RC:

1) TRC (Tuple Relational Calculus)
2) DRC (Domain Relational Calculus)

▶ **TRC (Tuple Relational Calculus):** **TRC** is a non-procedural language. So, we use the mathematical symbols. Any given query has to be implemented, TRC maintains a formulae:

❼ TRC = { T/ p(T)} where T❼ Tuple of given relation P(T) ❼ Predict of Tuples.

▶ **DRC (Domain Relational Calculus):** DRC is a non procedural language. So, here no operation are used, only mathematical symbols i.e, <, >, =, ∀ (for all), ∃ (There exists), E(Belongs) ¬ (not), ^(and) are used. To implement any Queries, DRC has provided some formulae,

❼DRC = {<x1, x2, x3…xn>/p< x1, x2, x3…xn >}

❼ <x1, x2, x3…xn> are domains; p< x1, x2, x3…xn > are predict of the domain. Domain means process of renaming attributes of the relations, Domain Relational Calculus (DRC) is always applied on domains.

Simple Database Schema

When we talk about a database, we must differentiate between the database schema, which is the logical design of the database, and the database instance, which is a snapshot of the data in the database at a given instant in time. The concept of a relation corresponds to the programming language notion of a variable, while the concept of a relation schema corresponds to the programming language notion of type definition. In general, a relation schema consists of a list of attributes and their corresponding domains. The concept of a relation instance corresponds to the programming-language notion of a value of a variable. The value of a given variable may change with time

dept_name	building	budget
Biology	Watson	90000
Comp.Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

Similarly the contents of a relation instance may change with time as the relation is updated. In contrast, the schema of a relation does not generally change. Although it is important to know the difference between a relation schema and a relation instance, we often use the same name, such as instructor, to refer to both the schema and the instance. Where required, we explicitly refer to the schema or to the instance, for example "the instructor schema," or "an instance of the instructor relation." However, where it is clear whether we mean the schema or the instance, we simply use the relation name.

Consider the department relation of Figure. The schema for that relation is department (dept name, building, budget) Note that the attribute dept name appears in both the instructor schema and the department schema. This duplication is not a coincidence. Rather, using common attributes in relation schemas is one way of relating tuples of distinct relations.

For example, suppose we wish to find the information about all the instructors who work in the Watson building. We look first at the department relation to find the dept name of all the departments housed in Watson. Then, for each such department, we look in the instructor relation to find the information about the instructor associated with the corresponding dept name.

Table Definitions:

1) Create: In SQL the database primary object is "table". The table contains rows and columns in which the data is stored. In a table each column is identified by a column name and a given data type with specified width.

In SQL the tables are created by using "Create table" command. This command creates only the structure of the table.

Syntax: CREATE TABLE <table name>(Column name1 data type(width), Columnname2 data type(width).);

Ex: create table student(htno number(8), sname varchar2(15), course varchar(5));

ii) Alter: The alter command is used to modify the structure of the existing table. By using the alter command we can do the following.

→ We can add new columns to an existing table.

→ We can remove the columns from an existing table

→We can add and drop constraints

Syntax: Alter table <table name> add/modify(column name data type(width),

Ex-1: Alter table student add(dob date);

The above command adds new column "dob" to the Student table.

Ex 2: Alter table student modify(htno varchar(10)); The above command modify the htno "number" data type to "varchar."

Syntax: Alter table <table name> drop column <column name>;

Ex 1: Alter table student drop column (course);

The above command removes the existing column "course" from the STUDENT table

Syntax: Alter table <table name> add constraint;

The above syntax is used to add a table constraint.

Ex 2: Alter table student add primary key(htno);

The above command add a primary key constraint to the 'htno' column on Student table

Ex 3: Alter table emp add foreign key(deptno) references dept; The above command add a foreign key constraint to the 'deptno' column to the 'Emp' table.

Datatypes:

The data type specifies what type of data is stored into a specific variable. The following are the different types of data types.

1. Number
2. Char
3. Varchar2
4. Long
5. Date
6. Boolean

1. Number: This data type is used to store integer or real values.

Syntax: Number(l,d);

Here l-length and d-no.of decimal places in length

Ex: (1) sno number(6);
(2) sal number(7,2);

sal: 19000.60;

2. Char: This data type is used to store only characters or alphabets.

Syntax: char(size);

Here size no.of characters.

If the size is not specified, then it occurs 1 byte of the memory. **Ex:**
gender char;

gender: 'M';

3. Varchar2: This data type is used to store alphanumeric vales.

Syntax: varchar2(size); Here
size no.of characters. **Ex:**
emp_id varchar2(10);
emp_id: 'E_123';

The maximum no.of characters size is 4000.

4. Long: This data type is similar to varchar2 data type. The range of this data type is "2GB".

Syntax: long;

Ex: biodata long;

5. Date: This data type is used to store date values. It reserves 7 bytes of memory. But the values are assigned through built-in functions only. The default date format is "DD-MON- YYYY". This date format is called "Julian date format". The date value must be in the range from "01-JAN-4612 BC" to "31-DEC-4612 AD".

Syntax: date;

Ex: s_dob date;

s_dob:-TO_DATE ('21-Nov-1993', 'DD-MON-YYYY');

6. Boolean: The Boolean data type can store only "True or false". The default value is "false".

Syntax: boolean;

Ex: a Boolean;

Different DML Operations:

The DML means "Data Manipulation Language". The DML commands are used to manipulate data in a database. i.e., The DML commands are used to insert data, modify data, delete data and retrieve data in the table. In SQL the DML commands are classified into mainly 4 types. They are i) Insert ii) Update iii) Delete iv) Select

i) **Insert:** This command is used to add records into a table. This command is also used for inserting data into particular columns of a table.

Syntax:- insert into <table name> values (val1,val2.....;

Ex:- insert into student values(101,'venu','B.Tech');

The above insert command is used for inserting values into all the columns. For inserting values for a particular columns we have to use the following syntax.

Syntax:- insert into <table name> (field1,field2,…….....) values(val1, val2. .....); **Ex:-**
insert into student (sno, sname) values(101,'kumar');

The above insert commands inserts only "sno, sname" values to "student" table.

Syntax for inserting Multiple rows

Syntax: insert into <table name> values ('&field1', '&field"

Ex: insert into student values['&sno', '&sname', '&course');

The above insert command takes the values for "Sno, Sname, Phno" at the time of query execution. This insertion is also called "bulk insertion".

ii) Update: The update command is used to modify the existing data with new data in a table..

This command used "Set" clause.

Syntax: update<table name> set <field1>=vall, <field2>-val2.....

Ex: Update student set course= 'B.Tech';

The above update command modifies the all records(row) in a specified table. For modifying only the specified records(rows), we have to use "where" clause.

Syntax: update <table name> set <field>-vall,<field2>-val2.....where <condition>;

Ex: update student set course='B.Tech' where sno=101;

The above command modifies only particular records that will satisfy the condition sno=101 from the table "student".

iii) Delete: The delete command is used to delete the specific records or all records from the table. For deleting a particular record in the table, we have to use "Where" clause.

Syntax: delete from <table name>(where <condition>);

Ex 1:- delete from student;

The above command deletes all rows from the table

Ex 2:- delete from student where sno = 101;

The above command deletes specific rows that will satisfy the condition sno = 101 from the table "student".

iv) Select: This command is also called "DQL" command. It is used to retrieve (or) display the data from the

table. By using this command and we can select all columns (or) specific columns all rows (or) specific rows from the table.

Syntax: select *from <table name> [Where <condition>];

The above syntax *" represents all columns in the table. Here 'where' clause is optional.

Ex 1: Select *from student;

The above command displays all columns and all rows from the "student" table.

Ex 2: select *from student where sno=101;

The above command displays all columns and specific rows that will satisfy the condition sno=101;

Ex 3: Select sno, sname from student;

The above command displays specific columns i.e., sno, sname and all rows from the table "student"

Ex 4: Select sno, sname, course from student where sno=101;

The above command displays particular column values i.e, sno, sname, course and specific rows that will satisfy the condition sno=101 in the table "student".

DBMS UNIT 3

Syllabus:

SQL: Basic SQL querying (select and project) using where clause, arithmetic & logical operations, SQL functions (Date and Time, Numeric, String conversion). Creating tables with relationship, implementation of key and integrity constraints, nested queries, sub queries, grouping, aggregation, ordering, implementation of different types of joins, view (updatable and non-updatable), relational set operations.

----------Basic SQL Querying Concepts----------

In SQL (Structured Query Language), the ability to query databases involves two primary operations: selecting (retrieving data) and projecting (filtering columns). The WHERE clause is crucial in refining these operations by dictating the conditions under which data should be retrieved.

Sailors

Sid	Sname	rating	age
22	Dustin	7	45
29	Brutal	1	45
31	Lubber	8	15
32	Andy	8	16
58	Rusty	10	17
64	H	7	18
71	Z	10	15
74	X	10	15
85	Andrew	9	18
95	Bob	3	19
		3	20

Reserves

Sid	bid	date
22	101	1/1/12
22	102	2/1/12
22	103	3/1/12
22	104	4/1/12
31	102	5/1/12
31	103	6/1/12
31	104	7/1/12
64	101	8/1/12
64	102	9/1/12
74	103	10/1/12

Boats

bid	bname	bcolor
101	Titanic	Blue
102	Interlake	Red
103	Clipper	Green
104	Marine	Yellow

2. SELECT Statement

- The **SELECT** statement is used to specify and retrieve specific columns from a database table.
- It can be combined with other clauses to filter and sort results.

3. Projection

- **Projection** refers to the operation of selecting specific columns from a table.
- For example, if you want to retrieve only the names and ratings of sailors, you can project those specific attributes.

4. WHERE Clause

- The **WHERE** clause is used to filter results based on specific conditions.
- Only records that meet the criteria specified in the WHERE clause will be included in the result set.

Example using the Provided Tables

Assuming you want to perform queries on the given tables **Sailors**, **Reserves**, and **Boats**, here's how you can apply **SELECT**, **PROJECTION**, and the **WHERE** clause:

5. Sample Queries

- **Query 1: Select Sailors with a Rating Higher than 5 SQL**>

SELECT Sname, rating FROM Sailors WHERE rating > 5; O/P:

Sname	Rating
Dustin	7
Lubber	8
Andy	8
Rusty	10
Z	10
X	10
Andrew	9

- **Query 2: Select All Columns for Sailors Aged under 20**

SQL> SELECT * FROM Sailors WHERE age < 20; O/P:

Sid	Sname	Rating	Age
32	Andy	8	16
58	Rusty	10	17
64	H	7	18
71	Z	10	15
74	X	10	15
95	Bob	3	19

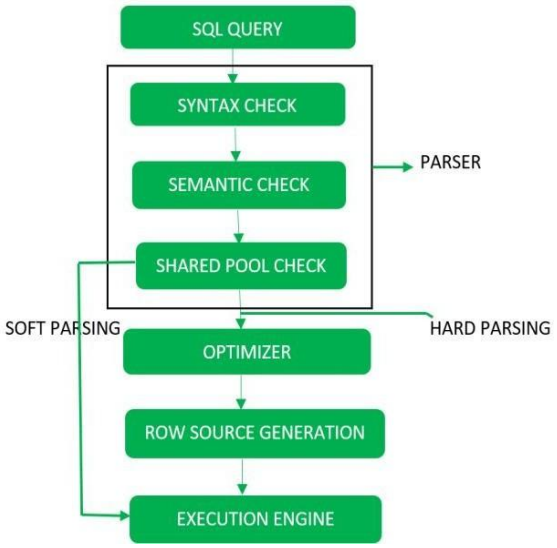
- **Query 3: Get Boat Names with Color Red**

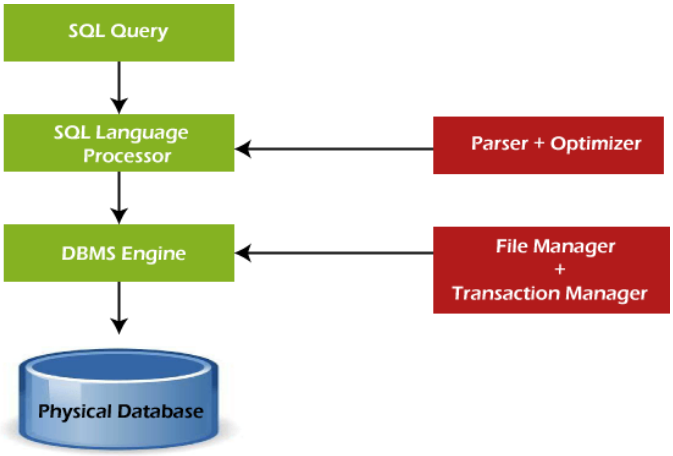
SQL> SELECT bname FROM Boats WHERE bcolor = 'Red'; O/P:

Bname
Interlake

6. Importance of Using WHERE Clause

- Enhances performance by limiting the results returned.
- Helps in retrieving specific data relevant to the user's needs, avoiding unnecessary data retrieval.





1. SQL Query (Input)

- The process begins when a user writes an SQL query (e.g., SELECT * FROM students WHERE age > 18;).
- This query is submitted to the DBMS.

2. SQL Language Processor

- This component handles query parsing and optimization.
- **Parser:** It checks the syntax of the SQL query to ensure it is correctly written.
- **Optimizer:** It analyzes different ways to execute the query efficiently and selects the best execution plan.

3. DBMS Engine

- This is the core part of the DBMS that interacts with the database.
- It coordinates data retrieval and updates based on the optimized query execution plan.

4. File Manager & Transaction Manager

- **File Manager:** Handles the reading and writing of data to and from the physical database.
- **Transaction Manager:** Ensures that transactions (e.g., multiple SQL operations) follow ACID properties (Atomicity, Consistency, Isolation, Durability) to maintain data integrity.

5. Physical Database (Storage)

- The data is stored in the physical database.
- The DBMS retrieves, updates, or modifies data as required by the query.

6. Query Result (Output)

- Once the database processes the query, the required data is retrieved and sent back to the user.

This diagram represents the fundamental workflow of SQL query execution within a DBMS, ensuring efficient data retrieval and management.

----------Arithmetic & logical operations----------

In a Database Management System (DBMS), arithmetic and logical operations are fundamental for performing calculations and making decisions based on data. Below are explanations, examples, and associated SQL queries involving both types of operations with the provided tables: Sailors, Reserves, and Boats.

1. Arithmetic Operations

Arithmetic operations are used to perform mathematical calculations. Common operations include addition, subtraction, multiplication, and division.

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.

Examples and Queries

- **Addition**: Calculate the average age of sailors.

SQL> SELECT AVG(age) AS AverageAge FROM Sailors;

- **Subtraction**: Find the difference between the maximum and minimum ratings of sailors.

SQL> SELECT MAX(rating) - MIN(rating) AS RatingDifference FROM Sailors;

- **Multiplication**: Calculate the total rating of all sailors and multiply it by a factor (e.g., 2).

SQL> SELECT SUM(rating) * 2 AS TotalRatingDouble FROM Sailors;

- **Division**: Calculate the average rating per sailor by dividing total ratings by the number of sailors.

SQL> SELECT SUM(rating) / COUNT(Sid) AS AverageRating FROM Sailors;

2. Logical Operations

Logical operations help in making decisions based on conditions using logical operators like AND, OR, and NOT.

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.

Examples and Queries

- **AND**: Find sailors with a rating greater than 5 and age less than 20.

SQL> SELECT * FROM Sailors WHERE rating > 5 AND age < 20;

- **OR**: Retrieve sailors who are either older than 18 or have a rating of 10.

SQL> SELECT * FROM Sailors WHERE age > 18 OR rating = 10;

- **NOT**: Get sailors who do not have a rating of 3.

SQL> SELECT * FROM Sailors WHERE NOT rating = 3;

Combining Logical Operators: Find sailors with a rating of 7 and not older than 18.

SELECT * FROM Sailors WHERE rating = 7 AND NOT age > 18;

Arithmetic and logical operations in SQL allow for data manipulation and decision-making based on specific criteria. The provided examples illustrate various ways to execute such operations using the Sailors, Reserves, and Boats tables. ----------
Functions in MySQL----------

Functions in MySQL are an essential part of the SQL language, enabling users to perform calculations, manipulate data, and retrieve information efficiently. These functions can edit rows and tables, alter strings, and help manage structured and easy-to-navigate databases.

A function is a predefined command set that performs an operation and returns a value. Functions operate on zero, one, or more values provided as parameters (arguments). MySQL functions are categorized into different types:

- **String Functions**
- **Numeric Functions**
- **Date and Time Functions**

MySQL String Functions

String functions manipulate and transform text strings. Some commonly used string functions include:

S no	Function	Description	Example
1	CHAR()	Returns the character for each integer provided	`SELECT CHAR(70, 65, 67, 69);`
2	CONCAT()	Returns a concatenated string	`SELECT CONCAT(name, aggregate) AS "Name Marks" FROM student WHERE age = 14 OR age = 16;`
3	LOWER() / LCASE()	Converts text to lowercase	`SELECT LOWER('GEEKSFORGEEKS');`
4	SUBSTRING() / SUBSTR()	Extracts a substring from a string	`SELECT SUBSTR('ABCDEFG', 3, 4);`
5	UPPER() / UCASE()	Converts text to uppercase	`SELECT UPPER('Large');`
6	TRIM()	Removes leading and trailing spaces	`SELECT TRIM(' Bar One ');`
7	LENGTH()	Returns the length of a string in bytes	`SELECT LENGTH('CANDIDE');`

MySQL Numeric Functions

Numeric functions perform mathematical operations on numeric values. Some useful numeric functions include:

S.No	Function	Description	Example
1	MOD()	Returns the remainder of division	`SELECT MOD(11, 4);`
2	POWER() / POW()	Returns the value of one expression raised to the power of another	`SELECT POWER(3, 2);`
3	ROUND()	Rounds a numeric expression to an integer or decimal places	`SELECT ROUND(15.193, 1);`
4	SIGN()	Returns the sign of a given number	`SELECT SIGN(-15);`
5	SQRT()	Returns the square root of a number	`SELECT SQRT(26);`
6	TRUNCATE()	Truncates a number to a specified decimal place	`SELECT TRUNCATE(15.79, 1);`

MySQL Date and Time Functions

Date and Time functions manipulate and retrieve date and time values stored in tables. Some useful functions include:

S.No	Function	Description	Example
1	CURDATE() / CURRENT_DATE()	Returns the current date	`SELECT CURDATE();`
2	DATE()	Extracts the date part of a datetime value	`SELECT DATE('2020-12-31 01:02:03');`
3	MONTH()	Returns the month from a given date	`SELECT MONTH('2020-12-31');`
4	YEAR()	Returns the year from a given date	`SELECT YEAR('2020-12-31');`
5	NOW()	Returns the current date and time	`SELECT NOW();`
6	SYSDATE()	Returns the system's current date and time	`SELECT SYSDATE();`

MySQL functions are indispensable tools for database management, providing a range of capabilities from manipulating strings and performing mathematical operations to handling date and time data. By utilizing these functions, users can streamline data retrieval and

transformation processes, ensuring more effective and organized database operations. ---------- **----------Creating Tables with Relationships in DBMS----------**

Introduction to DBMS Relationships

A **Relational Database Management System (RDBMS)** organizes data into tables that are related to each other. Relationships between tables help avoid redundancy and maintain data integrity. The three types of relationships are:

1. **One-to-One (1:1)** – Each row in Table A relates to one row in Table B.
2. **One-to-Many (1:M)** – Each row in Table A can relate to multiple rows in Table B.
3. **Many-to-Many (M:N)** – Multiple rows in Table A relate to multiple rows in Table B through a junction table.

Example of Table Creation with Relationships

Step 1: Creating Tables

We will create three tables: **Sailors, Boats, and Reserves**, where:

- Each **Sailor** can reserve multiple **Boats** (One-to-Many).
- Each **Boat** can be reserved by multiple **Sailors** (Many-to-Many).
- The **Reserves** table acts as a bridge between Sailors and Boats.

Step 2:

INSERT INTO Sailors (Sid, Sname, rating, age) VALUES

(22, 'Dustin', 7, 45),

(29, 'Brutal', 1, 45),

(31, 'Lubber', 8, 15),

(32, 'Andy', 8, 16),

(58, 'Rusty', 10, 17),

(64, 'H', 7, 18),

(71, 'Z', 10, 15),

(74, 'X', 10, 15),

(85, 'Andrew', 9, 18),

(95, 'Bob', 3, 19);

INSERT INTO Boats (bid, bname, bcolor) VALUES

(101, 'Titanic', 'Blue'),

(102, 'Interlake', 'Red'),

(103, 'Clipper', 'Green'),

(104, 'Marine', 'Yellow');

INSERT INTO Reserves (Sid, bid, date) VALUES

(22, 101, '2012-01-01'),

(22, 102, '2012-02-01'),

(31, 103, '2012-03-01'),

(31, 104, '2012-04-01'),

(31, 102, '2012-05-01'),

(64, 101, '2012-06-01'),

(64, 102, '2012-07-01'),

(64, 103, '2012-08-01'),

(74, 103, '2012-09-01'),

(74, 103, '2012-10-01');

Step 3: Sample Queries

1. **Retrieve all sailors who reserved a boat:**

SQL> SELECT DISTINCT Sname FROM Sailors WHERE Sid IN (SELECT Sid FROM Reserves);

Output:

Sname
Dustin
Lubber
H
X

2. **Find the boats reserved by 'Dustin':**

SQL> SELECT bname FROM Boats WHERE bid IN (SELECT bid FROM Reserves WHERE Sid = (SELECT Sid FROM Sailors WHERE Sname = 'Dustin'));

Output:

bname
Titanic
Interlake

3. **List sailors along with the boats they reserved:**

SQL> SELECT S.Sname, B.bname, R.date FROM Sailors S JOIN Reserves R ON S.Sid = R.Sid JOIN Boats B ON R.bid = B.bid;

Output:

Sname	bname	date
Dustin	Titanic	2012-01-01
Dustin	Interlake	2012-02-01
Lubber	Clipper	2012-03-01
Lubber	Marine	2012-04-01
Lubber	Interlake	2012-05-01
H	Titanic	2012-06-01
H	Interlake	2012-07-01
H	Clipper	2012-08-01
X	Clipper	2012-09-01
X	Clipper	2012-10-01

4. **Find sailors who have reserved the boat 'Titanic':**

SQL> SELECT S.Sname FROM Sailors S JOIN Reserves R ON S.Sid = R.Sid JOIN Boats B ON R.bid = B.bid WHERE B.bname = 'Titanic';

Output:

Sname
Dustin
H

5. **Count the number of reservations per sailor:**

SQL> SELECT Sname, COUNT(*) AS total_reservations FROM Sailors S JOIN Reserves R ON S.Sid = R.Sid

GROUP BY Sname;

Output:

Sname	total_reservations
Dustin	2
Lubber	3
H	3
X	2

Relational Database Concepts Used

1. **Primary Key:** Ensures unique identification of each record in Sailors and Boats.
2. **Foreign Key:** Maintains relationships between tables (Reserves references Sailors and Boats).
3. **Joins:**
    - **INNER JOIN:** Combines tables based on a common field.
    - **Subqueries:** Queries within queries to filter specific data.
4. **Aggregation:** COUNT (*) is used to count the number of reservations.
5. **Filtering:** The WHERE clause is used to filter specific conditions.

Normalization

- **1NF (First Normal Form):** No duplicate rows, atomic values.
- **2NF (Second Normal Form):** No partial dependencies (separating boats and sailors).
- **3NF (Third Normal Form):** No transitive dependencies.

Advantages of Table Relationships in DBMS

1. **Data Integrity:** Foreign keys prevent orphaned records.
2. **Avoids Redundancy:** Data is stored efficiently in separate tables.
3. **Scalability:** Multiple relationships can be managed easily.

4. **Efficient Querying:** JOIN operations allow retrieval of related data.
5. **Normalization:** Ensures better database design by eliminating data duplication.

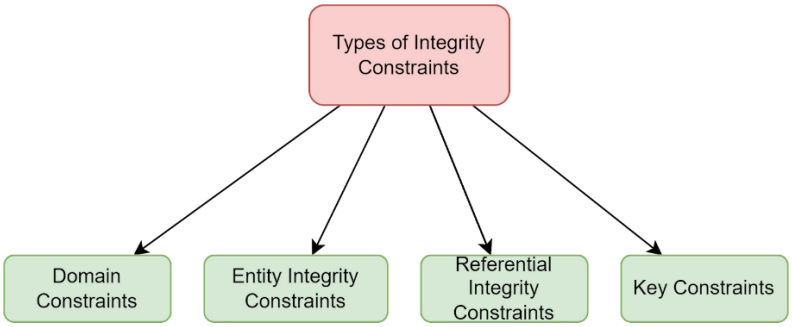
----------**Implementation of key and integrity constraints**----------

In Database Management Systems (DBMS), data integrity constraints are a set of rules applied to table columns or relationships to ensure the overall validity, Integrity, and consistency (i.e., the quality) of the data in the database. we will see an introduction to data integrity Constraints and learn about the types of Integrity Constraints and examples in depth.

Data Integrity

Data integrity means having correct and accurate data in the Database. In simple words, if you are storing the data in the database, you want your data not to be repetitive and incorrect. So data integrity helps you to prevent any broken relationships between tables.

Types of Data Integrity Constraints



1. Domain Integrity Constraints: A domain is a set of values that may assign to a column.

Domain Integrity Constraints are 2 types. They are : i) Not Null ii) Check

i) Not Null: The value which is absent is called NULL. The NOT NULL constraint is used to avoid NULL values into a column of a table. The NOT NULL constraint allows duplicate values.

Eg: SQL> Create table student(Sno number(10) NOT NULL, Sname varchar2(20)); In the above example, Sno column should not allow null values. **ii) Check**: The Check constraint is used to specify a particular condition to a column. It checks the condition satisfied or not.

Eg: SQL> Create table student(Sno number(10), Sname varchar2(20), Sfee number(8,2) CHECK (Sfee>10000)); In the above example, the column 'Sfee' is allows only the values, which are greater than 10000.

Entity Integrity Constraints: The entity integrity constraints are used to avoid invalid data into column of a table. There are 2 types i) Unique ii) Primary Key

i) Unique: It is used to prevent duplicate values to a particular column. i.e., It does not allow , but allows NULL values.

Eg: SQL> Create table emp(E_no number(3) UNIQUE, Ename varchar2(30), Deptno number(5));

ii) Primary Key: A filed (or) combination of fields used to identify a single record will be called as Primary key. It does not allow duplicate values and null values.

Eg: SQL> Create table emp(E_no number(3) PRIMARY KEY, Ename varchar2(30), Deptno number(5));

3. Referential Integrity Constraints: The referential integrity constraint is used to validation of a rule between two tables. A filed which references the primary key of another table will be called as Foreign key.

To implement this type of constraint the Master (parent) table must contain primary key and the same column in the Detailed (child) table as a reference of primary key known as foreign key.

Master Table: Eg: SQL> Create table Dept(Deptno number(5) PRIMARY KEY, Dname varchar2(20), Locaction varchar2(10));

Detailed Table: Eg: SQL> Create table Emp(E_no number(3) PRIMARY KEY, Ename varchar2(20), Deptno number(5) references Dept(Deptno));

The records from the Master table will not be deleted until the corresponding records are deleted from the detail table. Similarly new records cannot be inserted into the detail table, until the corresponding values are inserted in the master table.

SQL has several key constraints that ensure an entity or record is uniquely or differently identifiable in the database. The table may include more than one key, but only one primary key.

----------**Nested queries**----------

A query within the query is called "sub query". The sub queries are used to get the information from one or more tables..

The first query in SQL statement is known as "Outer query".

The query inside the SQL statement is known as "Inner query".

At first, the inner query is executed, then after the outer query is executed.

The inner query must be specified within parenthesis "()".

The output of the inner query is used as the input for the outer query.

The entire SQL statement is sometimes called as nested query.

1. **Single row sub query**: The inner query may return a single value then it is called as "Single row sub query". For example, to display employee details who are doing the same job as 'SMITH'.

Ex: select from emp where job (select job from emp where ename = 'SMITH');

2. **Multiple row sub query:** The inner query may return more than one value then it is called as "Multiple row sub query". In this case, we use IN, ALL, ANY special operators.

Ex (1): select from emp where job IN (select job from emp where deptno = 20);

(or)

select from emp where job ANY (select job from emp where deptno = 20);

It display the employee details whose job is same job in deptno '20'.

3. **Nested sub query:** A sub query is defined in another sub query is called as nested sub query.

Ex: select from emp where sal (select max(sal) from emp where sal < (select max(sal) from emp);

It display the employee details, who drawn second maximum salary.

4. **Correlated sub query:** A Correlated sub query is a sub query, that executes once for each row in the outer query. The inner query is depends on the outer query. This process is similar to nested loops in programming languages.

Ex: for(i=1;i<=2;i++)

for(j=1;j<=3;j++)

printf("i=%d j=%d",i,j);

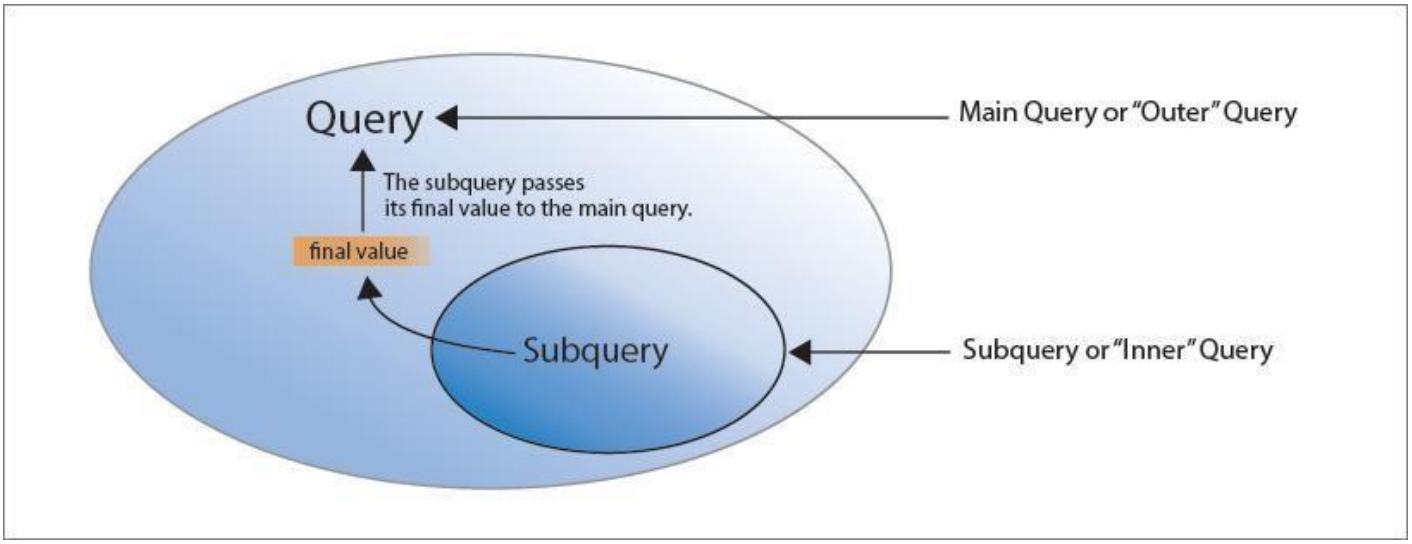
Nested Query: (set membership operations)

- Nested query is a query embedded another query initiate.
- Sub query is always required to write under the 'where' class of main query.

SQL> SELECT <attribute name> from <table name> where (subquery);

- To implement Nested Query, we use 2 different operations: They are: i) In, ii) Not in
- These 2 operations are called as set membership operations.

NOTE: Any query is implemented Right to left evaluation.



QUERIES for nested query:

i. Find the names of sailors who have reserved boat no 103.

SQL> select Ssname from sailors S where Ssid in (select Rsid from Reserve R where Rbid = 103);

Output: Ssname

Ssname
Dustin
Lubber
X

ii. Find the names of sailors, who have reserved Red boat.

SQL> select Ssname from sailors S where Ssid in (select Rsid from reserves R where Rbid in (select Bbid from Boats B where BBcolour=Red));

Output: Ssname

Ssname
Dustin
Lubber
H

iii. Find the colours of boats reserved by Lubber.

SQL> select Bbcolour from Boats B where Bbid in (select Rbid from reserve R where Rsid in (select Ssid from sailors where Ssname=‘Lubber));

Output: Bcolour

Red

Green

Yellow

iv. Find the names of sailors. Who have not reserved Red boat.

SQL> select Ssname from sailors where Ssid Not in(select Rsid from reserves R where Rbid in (select Bbid from boats B where Bbcolour = Red));

Output: Bcolour

X

----------Aggregate Functions (or) Group Functions----------

The Aggregate function is also called Group function or Statistical functions. The group functions takes multiple value as an input and produces a single output. The group functions supported by SQL are

Max ()

Min ()

Sum ()

Avg ()

Count ()

Max: It is used to find the maximum or highest value of the specified column in a table.

Eg:- select max(attribute name) from Relation name or Table name;

Min: It is used to find the minimum lowest value of the specified column in a table.

Eg:- select min(attribute name) from Relation name or Table name;

3) Sum(): It is used to find the total sum of values in a specified column in a table

Eg:- select sum(attribute name) from Relation name or Table name;

- **Avg:-** It is used to find the average of specified column in a table

Eg:- Select avg(attribute name) from Relation name or Table name;

- **Count():** It is used to count the no. of rows or values. It is used in 3 ways.
  i) count (*), ii) count(<column name>), iii) count(distinct<column names>)

i) **The count(*)** function counts the values including nulls and duplicate

Eg:- select count(*) from emp;

ii) **The count(<column name>)** counts the values without including nulls

Eg:- select count(attribute name) from Table name;

iii) **The count(distinct <column name>)** counts the values without including nulls and duplicates.

Eg:- select count(distinct attribute name) from Table name;

SQL> select sum of sno from IVECM select sum(distinct(attribute name)) from relationship;

- Here we used distinct key in order to avoid the duplication among the data. But if does not effect the overall output of the query. But the distinct key is not suitable in finding average of the data.
- In order to find the particular row details or count select count(attribute name) from table name;

Query 1: Find the average marks of all students:

IV ECM

Sno	Sname	Smarks
1	X	20
2	Y	30
3	Z	40

SELECT AVG(Smarks) AS AverageMarks FROM IVECM;

O/P: AverageMarks: 30

Query 2: Find the total marks of all students:

SELECT SUM(Smarks) AS TotalMarks FROM IVECM;

O/P: TotalMarks: 90

Query 3: Find the maximum marks obtained by any student:

SELECT MAX(Smarks) AS MaximumMarks FROM IVECM;

O/P: MaximumMarks: 40

Query 4: Find the minimum marks obtained by any student:

SELECT MIN(Smarks) AS MinimumMarks FROM IVECM;

O/P: MinimumMarks: 20

Query 5: Count the number of students in the table:

SELECT COUNT(*) AS StudentCount FROM IVECM;

O/P: StudentCount: 3

6) Group by clause:

The group by clause is used to organize rows or values in group wise with the help of aggregate functions.

Syntax:- select command group by clause;

Here we need to find the data from one extent to another extent,i.e., Suppose we need to find data>=1200, we denoted as

Eg: select Amount from Bank group by branch having amount >=1200;

Output: Amount

SBI	1800
IB	900
AXIS	1200
ICICI	1400

Entry no	Sno	Amount	Branch
X20	1	500	SBI
X21	2	900	IB
X22	3	1200	AXIS
X23	4	1400	ICICI
X24	5	1300	SBI

7) Oder by clause: In order to arrange any data in ascending or descending order, we use this order by clause.

Syntax: select *from <table name> [where <condition>) order by <column name> [asc/desc]; bydefault the order by clause displays data in ascending order.

Eg: 1) select sm from IVECM order by SM DSC/ASC;

2) select sno from IVECM order by sno<DSC>;

- If we didn't mention any type of indication i.e, as ASC/DSC. It assigns ASC(Ascending) by default.
- If we need to perform ascending or descending from particular instant to particular instant we use like operations. **Order By Queries:**

Query 1: Order by Sno Ascending SELECT * FROM IICS

ORDER BY Sno ASC;

IICS

Sno	Sname	Smarks
1	X	20
2	Y	30
3	Z	40

O/P:

Query 2: Order by Smarks Descending SELECT *

FROM IICS

ORDER BY

Smarks DESC;

Sno	Sname	Smarks
1	X	20
2	Y	30
3	Z	40

O/P:

Sno	Name	Smarks
3	Z	40
2	Y	30
1	X	20

Group By Queries

Query 1: Group by Name and find the average Smarks

SELECT Name, AVG(Smarks) as Average_marks FROM IICS GROUP BY Name;

O/P:

Name	Average marks
X	20
Y	30
Z	40

Query 2: Group by Smarks and find the count of names

SELECT Smarks, COUNT(*) as Name_Count FROM IICS GROUP BY Smarks;

O/P:

Smarks	Name_Count
20	1
30	1
40	1

----------Views----------

A View is a logical window of the physical database i.e, it is a mask of the table. A view takes the output of query. So it is also called as "stored query". A view is one of the database object. A view on which it is created is called a "**base table**" and that view is called **"virtual table".**

Generally View is created based on select query. The query can contain columns computed columns, aliases and aggregate functions from one or more tables. These views are called **"complex (or) NonUpdatable"** views

Creating a View----------

In SQL "Create View" command is used to create a view on one or more base tables.

Let us consider the following EMP table we can create a view

Syn: Create view<view name>as select query;

1. Create a View for Sailors with High Ratings

Create a view named **Sailor_Info** that displays the sailor's name, age, and rating.

CREATE VIEW HighRatedSailors AS SELECT Sid, Sname, rating, age FROM Sailors WHERE rating >= 8;

OUTPUT:

Sid	Sname	Rating	Age
31	Lubber	8	15
32	Andy	8	16
58	Rusty	10	17
71	Z	10	15
74	X	10	15
85	Andrew	9	18

2. Create a View for Reservations on Specific Boats

This view shows the reservations for boats that are blue in color.

CREATE VIEW BlueBoatReservations AS SELECT R.Sid, R.bid, B.bname, R.date FROM Reserves R JOIN Boats B ON R.bid = B.bid WHERE B.bcolor = 'Blue';

Sid	Bid	Bname	Date
22	101	Titanic	1/1/12
64	101	Titanic	8/1/12

Dropping Views----------

1. **Drop the High Rated Sailors View**

This command removes the HighRatedSailors view that was previously created.

DROP VIEW HighRatedSailors;

Drop the BlueBoatReservations View

This command removes the BlueBoatReservations view.

DROP VIEW BlueBoatReservations;

Updating a View ----------

It is a view that can be used to update columns in the base table. i.e., A view that allows DML operators, those views are called "updatable views"

After creating the view we can used the DML command to update the view.

1. **Update Sailors View to Include Age**

This example demonstrates how to update a view to include the age of sailors with ratings of 7 or higher.

CREATE OR REPLACE VIEW SailorsWithAge AS SELECT Sid, Sname, rating, age FROM Sailors WHERE rating >= 7;

Sid	Sname	Rating	Age
22	Dustin	7	45
31	Lubber	8	15
32	Andy	8	16
58	Rusty	10	17
74	Z	10	15
85	Andrew	9	18
95	Bob	9	20

Update Reservations View for Recent Reservations This

view displays reservations made in 2012.

CREATE OR REPLACE VIEW RecentReservations AS SELECT R.Sid, R.bid, R.date FROM Reserves R

WHERE R.date >= '1/1/2012' AND R.date <= '12/31/2012';

Sid	Bid	Date
22	101	1/1/12
22	102	2/1/12
22	104	4/1/12
31	103	5/1/12
31	104	7/1/12
64	101	8/1/12
64	102	9/1/12
74	103	10/1/12

Advantages of Views:

1. Views provide the security in the Data base. Because the view can restrict users to only specified columns and rows in a table

2. Views hides the base table name

3. Views requires less storage space i.e It does not contain any data.

4. View simplifies queries i.e, it avoid the reconstruction of complex queries.

Some Restrictions/Disadvantages of views:

1) Aggregate functions cannot be used or Group by cannot be used

2) Set operators cannot be used i.e union, intersect and minus etc

3) The views must be created only a "single table"

4) The "Primary Key" and "Not Null" columns must be included in views

5) "Sub queries" must not be included

6) The "distinct" and "having" clauses cannot be included.

----------What is JOIN? ----------

In SQL, a JOIN is used to combine rows from two or more tables based on a related column between them. This allows users to retrieve related data across different tables in a single query. Here's a brief overview:

Types of JOINs

- **INNER JOIN**
- Returns records that have matching values in both tables.
- **LEFT JOIN (or LEFT OUTER JOIN)**
- Returns all records from the left table and the matched records from the right table. If no match is found, NULLs are returned for columns from the right table.
- **RIGHT JOIN (or RIGHT OUTER JOIN)**
- Returns all records from the right table and the matched records from the left table. If no match is found, NULLs are returned for columns from the left table.
- **FULL JOIN (or FULL OUTER JOIN)**
- Returns all records when there is a match in either left or right table records. If there's no match, NULLs are returned for columns where there is no match.
- **CROSS JOIN**
- Returns the Cartesian product of the two tables, meaning every row from the first table is combined with every row from the second table.

Using the Example Tables

Given the tables: **Sailors**, **Reserves**, and **Boats**, here's how you might use JOINs with them:

1. Inner Join Query:

SELECT S.Sid, S.Sname, B.bname, R.date FROM Sailors S INNER JOIN Reserves R ON S.Sid = R.Sid

INNER JOIN Boats B ON R.bid = B.bid; **Output:**

Sid	Sname	Bname	Date
22	Dustin	Titanic	1/1/12
22	Dustin	Interlake	2/1/12
31	Lubber	Clipper	3/1/12
31	Lubber	Interlake	4/1/12
64	H	Titanic	6/1/12
64	H	Clipper	9/1/12
74	X	Interlake	7/1/12

2. Left Join

Query:

SELECT S.Sid, S.Sname, R.bid, R.date FROM Sailors S LEFT JOIN Reserves R ON S.Sid = R.Sid;

Output:

Sid	Sname	bid	date
22	Dustin	101	1/1/12
22	Dustin	102	2/1/12
29	Brutal	NULL	NULL
31	Lubber	103	3/1/12
31	Lubber	102	4/1/12
32	Andy	NULL	NULL
58	Rusty	NULL	NULL
64	H	101	6/1/12
71	Z	NULL	NULL
74	X	102	7/1/12
85	Andrew	NULL	NULL
95	Bob	NULL	NULL

3. Right Join

Query:

SELECT R.Sid, S.Sname, R.bid, B.bname FROM Reserves R RIGHT JOIN Sailors S ON R.Sid = S.Sid RIGHT JOIN Boats B ON R.bid = B.bid; **Output:**

Sid	Sname	bid	bname
22	Dustin	101	Titanic
22	Dustin	102	Interlake
NULL	NULL	103	Clipper
31	Lubber	102	Interlake
64	H	101	Titanic
NULL	NULL	104	Marine
74	X	102	Interlake

4. Full Join

Query:

SELECT S.Sid, S.Sname, R.bid, B.bname FROM Sailors S FULL OUTER JOIN Reserves R ON S.Sid = R.Sid FULL OUTER JOIN Boats B ON R.bid = B.bid; **Output:**

Sid	Sname	bid	bname
22	Dustin	101	Titanic
22	Dustin	102	Interlake
29	Brutal	NULL	NULL
31	Lubber	103	Clipper
31	Lubber	102	Interlake
32	Andy	NULL	NULL
58	Rusty	NULL	NULL
64	H	101	Titanic
71	Z	NULL	NULL
74	X	102	Interlake
85	Andrew	NULL	NULL
95	Bob	NULL	NULL
NULL	NULL	104	Marine

SELECT S.Sname, B.bname FROM Sailors S CROSS JOIN Boats B;

Sname	Bname
Dustin	Titanic
Dustin	Interlake
Dustin	Clipper
Dustin	Marine
Brutal	Titanic
Brutal	Interlake
Brutal	Clipper
Brutal	Marine
Lubber	Titanic
Lubber	Interlake
Lubber	Clipper
Lubber	Marine
Andy	Titanic
Andy	Interlake
Andy	Clipper
Andy	Marine
Rusty	Titanic
Rusty	Interlake
Rusty	Clipper
Rusty	Marine
H	Titanic
H	Interlake
H	Clipper
H	Marine
Z	Titanic
Z	Interlake
Z	Clipper
Z	Marine
X	Titanic
X	Interlake
X	Clipper
X	Marine
Andrew	Titanic
Andrew	Interlake
Andrew	Clipper
Andrew	Marine
Bob	Titanic
Bob	Interlake
Bob	Clipper
Bob	Marine

5. Cross Join

Query:

Output:

---------- **Relational Set Operations**----------

Relational set operations are used in databases to combine, filter, or compare results from multiple tables. The common relational set operations include:

1. **Union (∪)** – Combines results from two tables and removes duplicates.
2. **Union All** – Similar to Union but includes duplicates.
3. **Intersection (∩)** – Returns common rows from two tables.
4. **Difference (-)** – Returns rows from the first table that are not in the second.
5. **Cartesian Product (×)** – Combines all rows of one table with all rows of another.
6. **Join Operations** – Combines tables based on a condition.

SQL Queries Based on Given Data 1.

UNION

Find all sailors (`Sname`) and boats (`bname`) together in one list.

SELECT Sname FROM Sailors UNION SELECT bname FROM Boats;

Output:

Sname
Dustin
Brutal
Lubber
Andy
Rusty
H
Z
X
Andrew
Bob
Titanic
Interlake
Clipper
Marine

2. INTERSECTION

Find sailors who have reserved both Titanic and Clipper.

SELECT Sid FROM Reserves WHERE bid = 101 INTERSECT SELECT Sid FROM Reserves WHERE bid = 103;

Output:

Sid
31
64
74

3. DIFFERENCE

Find sailors who have reserved Titanic but not Clipper.

SELECT Sid FROM Reserves WHERE bid = 101 EXCEPT SELECT Sid FROM Reserves WHERE bid = 103;

Output:

Sid
22
29

4. CARTESIAN PRODUCT

Get all possible combinations of sailors and boats.

SELECT * FROM Sailors, Boats;

Output (Partial, as it´s a large table):

Sid	Sname	rating	age	bid	bname	bcolor
22	Dustin	7	45	101	Titanic	Blue
22	Dustin	7	45	102	Interlake	Red
22	Dustin	7	45	103	Clipper	Green
22	Dustin	7	45	104	Marine	Yellow
29	Brutal	1	45	101	Titanic	Blue
...	...	...	...	...	...	...

5. INNER JOIN (Find all sailors who have reserved boats along with boat details)

SELECT S.Sid, S.Sname, B.bname, B.bcolor, R.date FROM Sailors S JOIN Reserves R ON S.Sid = R.Sid JOIN Boats B ON R.bid = B.bid;

Output:

Sid	Sname	bname	bcolor	date
22	Dustin	Titanic	Blue	1/1/12
22	Dustin	Interlake	Red	2/1/12
31	Lubber	Clipper	Green	3/1/12
31	Lubber	Marine	Yellow	4/1/12
64	H	Titanic	Blue	6/1/12
74	Z	Interlake	Red	7/1/12
74	Z	Clipper	Green	8/1/12

6. LEFT JOIN (List all sailors and their reserved boats, if any)

SELECT S.Sid, S.Sname, B.bname, B.bcolor FROM Sailors S LEFT JOIN Reserves R ON S.Sid = R.Sid LEFT JOIN Boats B ON R.bid = B.bid;

Output (showing all sailors, including those who haven´t reserved any boats):

Sid	Sname	bname	bcolor
22	Dustin	Titanic	Blue
22	Dustin	Interlake	Red
31	Lubber	Clipper	Green
31	Lubber	Marine	Yellow
64	H	Titanic	Blue
74	Z	Interlake	Red
74	Z	Clipper	Green
95	Bob	NULL	NULL

GEETHANJALI INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF CYBER SECURITY & DATA SCIENCE

UNIT 4: NORMALIZATION

SYLLABUS----------

Schema Refinement (Normalization): Purpose of Normalization or schema refinement, concept of functional dependency, normal forms based on functional dependency Lossless join and dependency preserving decomposition, (1NF, 2NF and 3 NF), concept of surrogate key, Boyce-Codd normal form (BCNF), MVD, Fourth normal form(4NF), Fifth Normal Form (5NF).

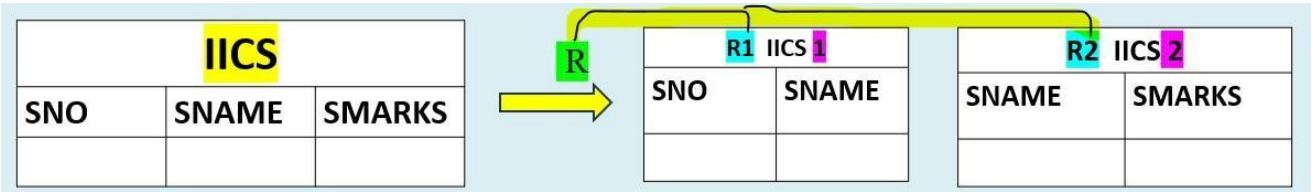
Purpose of Normalization or schema refinement:

Normalization is the process of eliminating the redundancy or duplicate values for given relation or table

To implement the Normalization process we need to know about:

i) Schema refinement, ii) Decomposition process

1. **Schema Refinement**: It is the process of modifying or inserting or alternating or updating of any data of the table to delete particular column of the table then we go for schema refinement.
2. **Decomposition process:** It is the process of dividing any larger volume of data or relation into sub relations.



Decomposition is again **divided** into **2 kinds**

1. Lossy Decomposition
2. Lossless Decomposition

- When decomposition process is done then there will be some loss of values or attributes this type is called as lossy decomposition
- In the ever example R is divided into R1 and R2 , so R1 has a loss of SMARKS and R2 has loss of SNO, So, it is lossy decomposition.
- Any lossy decomposition can be converted to lossless decomposition by combing all the decomposed sub division.

- This combining is done by using

  (i) Join

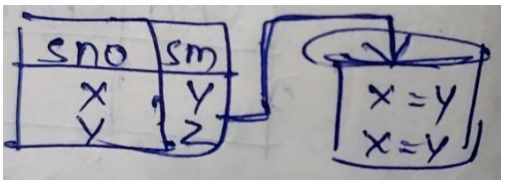
  (ii) Union

- Normalization is the process of removing duplicate values redundant values from the given table. Or
- Normalization is the process of arranging the relation data by using schema refinement and decomposition process.

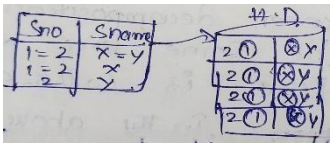
Redundancy: Storing of same information more than one copy into a hard disk is set to be a redundancy hard disk is constructed based on the multiple copies of storage. Is there occurs any hard disk failures, then there won't be any loss of data this work role had done by redundancy only if one copy of data is sent into hard disk it deformedly creates one another copy…

Redundancy problems: It copies large amount of space for more copies of data which gives overloaded this problem is said to be as redundancy. It is a problem to itself only due to sending of same data into two copies, It stores four copies of data. So that redundancy causes overloading of memory of hard disk.

Insertion: Suppose user created information record, That sent to hard disk once created any information in secondary memory it also create another same information.



Update: Want to create update the 1 value to 2 value, Then the

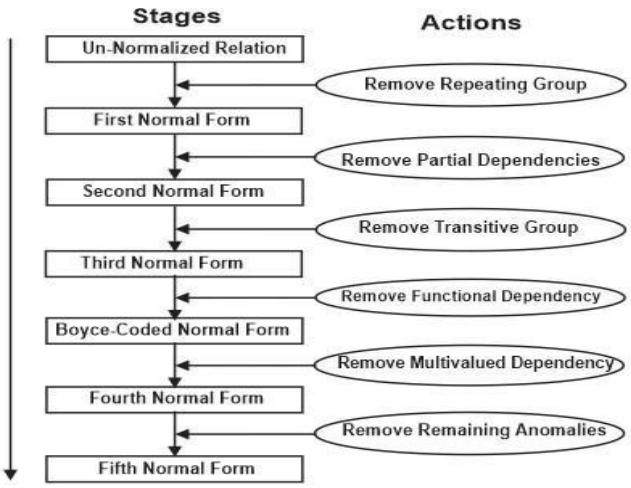


Corresponding is as so also contain two values..

Deletion: The user 1 x removed from the table 1 x and another is removed , similarly 1 x 1 x and from and from second memory also maintain another copy. So, no. of times the process increases.

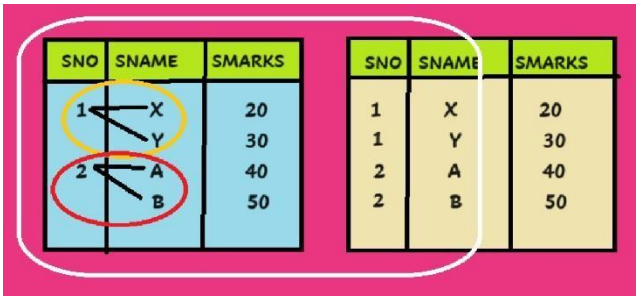
NORMALIZATION:

Normalization is divided into different forms, 1NF, 2NF, BCNF, 4NF, 5NF.



1st Normal Form:

- Consider a unnormalized relation each and every attribute required to maintain atomic i.e., 1st attribute doesn't on 2nd attribute.
- Normalization is the process of eliminating



the un-normalization from tables.

- The main drawback is redundancy.
- To overcome the drawback of 1NF,

we switch to 2NF.

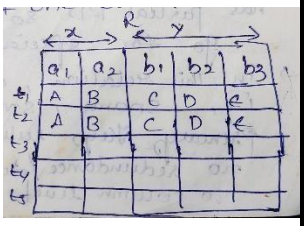
2nd Normal Form:

- It includes 1NF relation.
- 2NF is based on Fully Functional Dependency.
- Suppose there appeared Partial Dependencies, needed to eliminate the Partial dependencies from 1NF .
- Suppose given 1NF maintain the Fully Functional Dependency, at that time we go for the decomposition process.
- The large relation is divided into sub relation is called decomposition process.
- Sometimes it is not applicable these two values.
- Suppose 1, 2, 3, 4 given one as K normal form, then we go for special functional dependency **Functional Dependency (FD):**

- It is a integrity constraint (IC). Hiding any particular information from tables at that time we use integrity constraint. Let us take a relation 'R'. Relation required to maintain two non empty sets. These are X and Y. non empty set means the set maintain at least one element.

X = {a1, a2}

Y = {b1, b2, b3}



- Functional dependency is symbolically represented as **X → Y** or **Y → X**.
- **'X→Y'** means **X** functionally defines Y.
- First normal form relation has maintained special functional dependency i.e., It has primary attributes.
- We have to do decomposition process on first normal form based on primary attributes.
- Now decomposition process is done for SNAME and SMARKS.
-

IICSDS 1	
SNAME	SNO
X	1
Y	1
A	2
B	2

IICSDS 2	
SNAME	SMARKS
X	20
Y	30
A	40
B	50

- Now **IICSDS 1** has removed redundancy by decomposition.
- The main drawback of 2nd Normal Form is., it has lossy decomposition. i.e., some data will be skipped and advantage is, it fully eliminate redundancy values.
- To overcome drawback of 2NF, we move to 3NF.

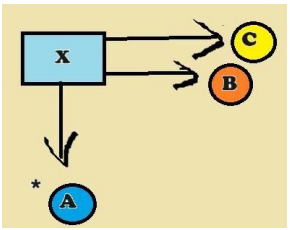
3rd Normal Form:

- Rules of 3NF are, Including first normal form relation.
- 3rd normal form is based on transitive dependency condition for transitive dependency is,

  If A → B and B → C, then A → C.
- X → A here X is super key and A is primary attribute.

  Therefore (X<=A) X set A.
- Super key is a collection of primary attributes and non primary attributes and each and every row is identified as uniquely.

- The entity X has 3 attributes i.e., A, B, C. This is because, 3NF is based on transitive dependency.



Here A❼ primary attribute, B❼ Non primary attribute.

Super Key = Primary Attribute + Non-Primary attribute

X = A + B.C

- Using decomposition process, the available relation is divided into sub relation based on primary attributes.
- Consider a 3NF relation. Check whether

IICSDS = X

B	*A	C
SNO	SNAME	SMARKS
1	X	20
1	Y	30
2	A	40
2	B	50

it have 3 attributes and name the attributes.

- By applying transitive properties $A \rightarrow B, B \rightarrow C, A \rightarrow C$
- When any attribute is given '*'

then it is said to be as Primary

Attribute i.e., Here 'A' is the Primary

Attribute i.e., A*.

A*-->B

SNAME	SNO
X	1
Y	1
A	2
B	2

B--> C

SNO	SMARKS
1	20
1	20
2	40
2	50

*A --> C

SNAME	SMARKS
X	20
Y	30
A	40
B	50

- 'A* → B' says that A* features

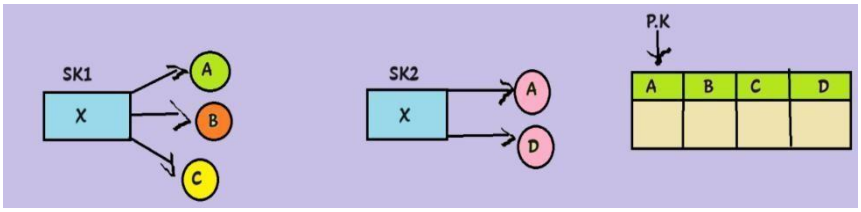
inherited to B. According to Super key, when primary key is transmitted to non- primary key, then the nonprimary key is also connected to primary key.

4) **BCNF (Boyce Code Normal Form):**

- **BCNF** stands for *Boyce code Normal Form.*
- BCNF is based on Trivial Dependency (Trivial means atleast maintain 3 attributes).
- 'X → A' here X is Candidate key and A is primary attribute where, $X \subseteq A$.
- Candidate Key is a collection of minimal super keys, and also each and every row identified as Unique.

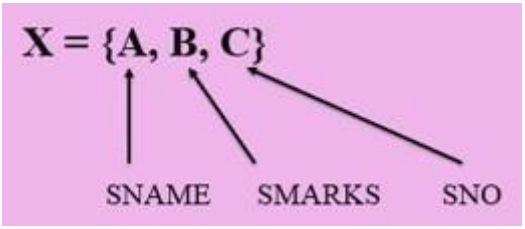
C.K = {SK1 + SK2 + …}

- Candidate Key, Super Key is always based or depends on primary key.

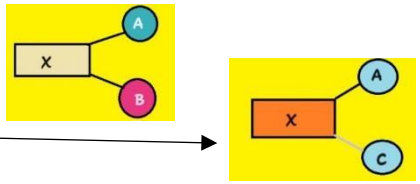


- Consider

SNO	SNAME	SMARKS
1	X	20
1	Y	30
2	A	40
2	B	50



- First normal form is divided into sub parts by using primary attributes by decomposition process.
- Primary attribute A →B
- Primary attribute A→



According to BCNF, each and every attribute left side to maintain primary attribute.

Candidate Key = {SK1 + SK2} = {A,B,C}

- Drawback is missing of some attributes. •

Advantages is automatically the redundancy values are eliminated.

SNO	SMARKS
A	B
X	20
Y	30
A	40
B	50

SNAME	SNO
A	C
X	1
Y	1
A	2
B	2

4th Normal form:

- **Rules:**

Including 1NF relation.

- **4NF** is based on Functional Dependency and Multivalued Dependency.
- Sometimes does not follow and maintain sub functional dependency and multi value dependency at that time we are using another rule, maintained by 4NF, Y ⊆ X (XY = R)

y = {X, A, B,C}. x is always consider as super key, Y={a, b, A, B, C}; x={a, b}

XY =

a	b	A	B	C

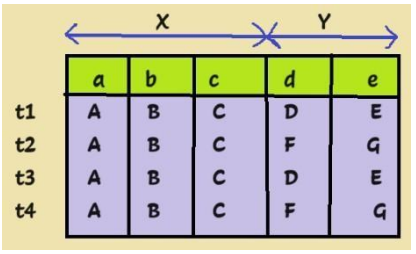
PA NPA

- **4NF** doesn't follow the Join & Union operation.
- **5NF** depend on the Join Dependency.

Comparison between BCNF & 3NF:

BOYCE CODE NORMAL FORM	3rd NORMAL FORM
1. Includes 1st Normal Form relation.	1. Both NF relates to 1NF relation.
2. Based on Transitive dependency.	2. Based on trivial dependency.
3. One Candidate key, which is a collection of minimal super keys.	3. Always depends on super key. (Super key =P.A + N.P.A)
4. Candidate Key is based on the primary attribute each and every row unique.	4. Super Key is based on P.A or Primary Key.
5. Its Functional Dependency X→ A (X ⊆ A), where X→ Candidate Key.	5. It has maintained some functional dependency X→A (X ⊆ A) X super set A.
6. Any given relation maintain above rules go for decomposition (or) Schema refinement process.	6. Any given relation maintain above rules go for decomposition.
7. Missing of some attribute values from decomposition relation (or) for A→C.	7. Missing of some attribute values (or) Decomposition values from relation main functional dependency.
8. Needed to overcome this drawback go for 4NF.	8. To overcome we use BCNF it maintain A→B, A→ C.

MVD (Multivalued Dependency):



- MVD stands for Multivalued Dependency.
- Let us consider as 1 relation 'R' that relation 'R' require to maintain 2 non empty sets, names are X and Y.
- Non-empty set is having minimum 1 element or 1 attribute. X= {a, b, c}, Y= {d, e}.
- It is also one type of integrity constraint, To prevent invalid entry of data into table.
- MVD is syntax notation because of multi value.

X → Y (Double determines Y).

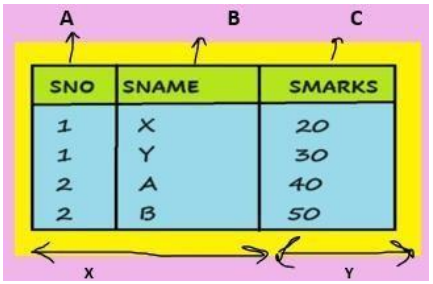
SNO	SNAME	SMARKS
1	X	20
1	Y	30
2	A	40
2	B	50

- Here minimum consider 4 tuples or records.
- MVD has maintained one main rule . If t1.X =t2.X=t3.X=t4.X then t1.Y=t3.Y. Odd tuples are equal t2.Y=t4.Y.
- It may any 4 tuples from the list of tuples available consider IICSDS relation.

Functional Dependency Rules:

- If t1.X=t2.X, t1.Y=t2.Y.
- This relation doesn't maintain FD rules.

Multivalued Dependency Rules:



- t1.X =t2.X=t3.X=t4.X then t1.Y=t3.Y and t2.Y=t4.Y.
- This relation is not suitable for 1NF because not follow MVD rules.
- So go for Y ⊆ X.
- No needed to divide into subparts (XY = R)

Advantages:

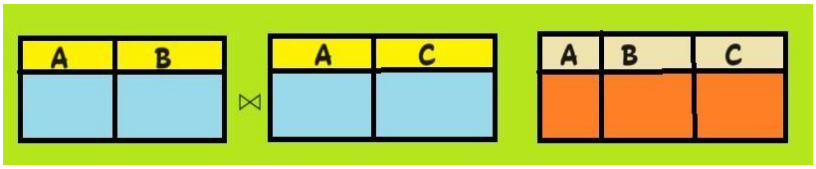
- Not possible to missing of attributes or relation values.
- Removes redundancy by primary key.

Drawback:

- Not specified 'X' Joiny (or) 'X' Union 'Y' so shifted from 4NF to 5NF

5th Normal Form:

- Includes 1NF relation. 5NF is based on Join Dependency. X **U** Y or X⋈ Y
- A→B ⋈ A→C is Join operation must maintain one same column or same attribute.
- One primary attribute is to the non-primary attribute then the NPA to PA and no duplication values.



Closure of FD's:

- All the FD's are given by F(FD's) is called Closure of FD's. It is syntactically denoted as F^+.
- To implement closure of FD's, consider FD'S as follows: A→B, B→C, C→D, D→E.
- The closure of any FD 'F' is represented as F^+.
- Now closure of given FD's as follows.
- A^+ →A B C D E; B^+→BCDE; C^+ → CDE; D^+ →DE; E^+→E.
- Any closure consists of all functional dependency from first attribute to nth attribute, then that closure attribute is called as Primary Attribute.

- In the above example, A is called Primary attribute because it has functional dependencies from A to E.

Armstrong Rules:

- Armstrong rules are used to combine more than one FD's to one FD. The rules are as follows.
- **Reflexibility Rule**: If X ⊇ Y (X subset Y) then X→Y.
- **Augmentation Rule:** If X→Y then XZ→YZ.
- **Transitive Rule:** If A → B, B→C then A→C.
- **Union Rule:** If X→Y, X→Z then X→YZ.
- **Decomposition Rule:** If X→YZ then X→Y, X→Z.
- **Pseudo transitivity:** If X→$\boldsymbol{\alpha}$, $\boldsymbol{\alpha}$ **Y** → $\boldsymbol{\delta}$ then XY → $\boldsymbol{\delta}$.

GEETHANJALI INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF CYBER SECURITY & DATA SCIENCE

UNIT 5 Transaction Concept & Introduction to Indexing Techniques

SYLLABUS----------

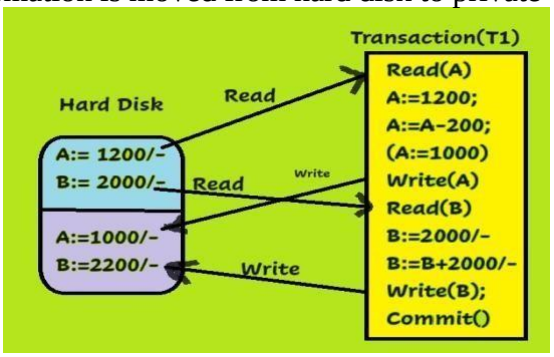
Transaction Concept: Transaction State, ACID properties, Concurrent Executions, Serializability, Recoverability, Implementation of Isolation, Testing for Serializability, lock based, time stamp based, optimistic, concurrency protocols, Deadlocks, Failure Classification, Storage, Recovery and Atomicity, Recovery algorithm.

Introduction to Indexing Techniques: B+ Trees, operations on B+ Trees, Hash Based Indexing.

Transaction:

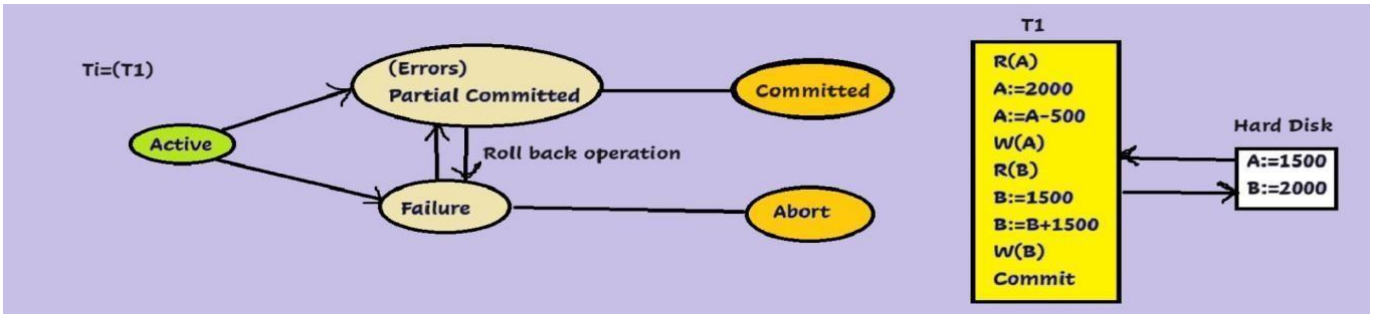
- Transaction is execution of unit of application by using two operations, Namely read and write operations.
- Let us consider two different accounts A and B initially A has an amount of rupees 1200/- and B has an amount rupees 2000/- in their accounts.
- To transfer the amount from A to B then we need to write an transaction application.
- The accounts information will be in hard disk. The information is moved from hard disk to private area because the transaction is done in private area.
- 'Commit' is the keyword used to represent

that the transaction is the successful. If 'Abort' is written at the end of the transaction, then that particular is unsuccessful.



Transaction State:

Transaction is execution of unit of application by using 2 operations namely read and write operations. Let us consider two different accounts A & B. Initially A has an amount of Rs.2000/- in their account. To transfer the amount from A to B then we need to write a transaction application. The accounts will be in hard disk. The into is moved from Hard disk to private area because the transaction is done in private area.



- Each transaction maintains 5 states. If a user wants to create new transaction, must and should enter the transaction to active state.
- Failure state taken case to maintain the failure transaction. The transaction of T1 needed to active need A into does not get properly info from H.D to private area goes to the failure state.
- Once got the 'A' info from the H.D to private area we perform read and write operation. Once completed the transaction successful before the end of the statement send to partial committed. If not successful then goes to failure state.
- Malfunction failures are occurred like power off, failures, mechanical, CPU, memory failures. So we use partial commit corresponding transaction contains failures must send from that p commit to failure state..
- Failure state is taken care to find out the number of errors are available and rectify by the failure state the rectified errors and transaction send back to the partial committed by using Roll back.
- Many times sending it to the failure state must send to the abort state. Once get the error free transaction must send from p commit to committed state.
- If not possible to create a transaction without the transaction states.

ACID PROPERTIES:

ACID stand for Atomicity, Consistency, Isolation and Durability. These are the four properties of transaction.

Atomicity:

It is the process of detecting the mechanical or power failures and presenting them when any transaction is going on, then at the middle of the transaction, if there are any hardware or mechanical or power failure, then after booting the system, the info may or may not be retrieved. So, atomicity is used to detect such failures and prevent them by using specific steps.

Consistency:

To implement the property, consider 2 transactions T1 and T2. Consistency arises when T1, T2 try to access hard disk at the same time or simultaneously concurrency means, executing of more than one transaction same time using same object consistency property is used to detect the concurrency problems and solve them.

Isolation:

Isolation property is used to execute any particular transaction individually, without the involvement of any other transaction. Isolation is the property of serialized transactions. So, after completion of T1, T2 gets executed later T3,…...

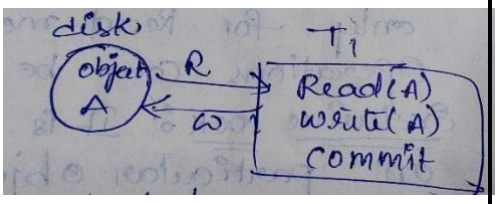
Durability:

Durability property is always applied on successful transactions only whenever there is 'commit' keyword under the transaction, then by durability property that successful transaction is moved from private area to permanent storage device of Hard disks.

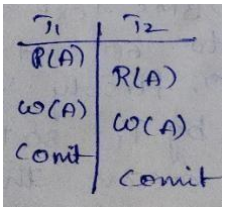
Serializable Transaction:

- It is a process of executing of more than one transaction information in a sequence order is called serializable schedule with in a time.
- If two transaction try to access the same object at the same time. Then there occurs deadlock. So, we go for serializable. So that the transaction can be completed within time avoiding deadlock.

Schedule:



- It is the process of completion of list of actions within a time.
- The actions are, Read, Write, Commit, Abort.
- Serializable schedule means completion of list of actions sequentially.
- Consider a disk has an object A. Then all the actions must be performed on it. So that the process is said to be maintain schedule.
- If all the actions is not performed then the process is said that it does not maintain schedule.
- After completion of T1, if T2 resumes then it is called as serializable schedule.
- The sequential transaction will be as shown.



Testing for Serializability:

1. **Serializability in Concurrency Control:**

To ensure correctness, schedules must be serializable.

Conflict serializability is a stricter form of serializability that can be tested efficiently.

2. Precedence Graph (Serialization Graph):

A directed graph G = (V, E) where:

V (Vertices): Transactions in the schedule.

- E (Edges): Directed edges indicating dependencies between transactions.
- An edge T → T' exists if one of the following conditions holds:

a. **Write-Read (WR Conflict):**

T writes Q before T' reads Q.

b. **Read-Write (RW Conflict):**

T reads Q before T' writes Q.

c. **Write-Write (WW Conflict):**

T writes Q before T' writes Q.

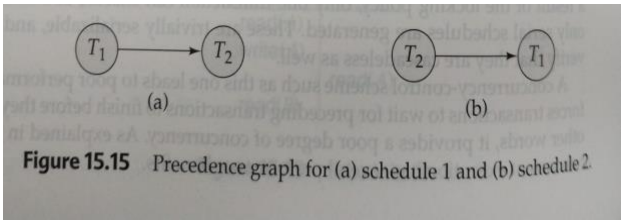
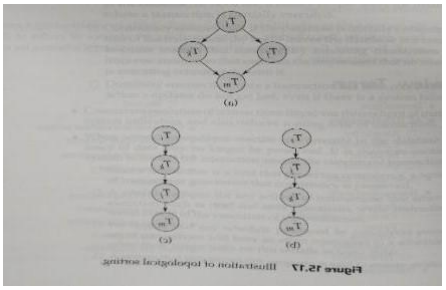


Figure 15.15 Precedence graph for (a) schedule 1 and (b) schedule 2.



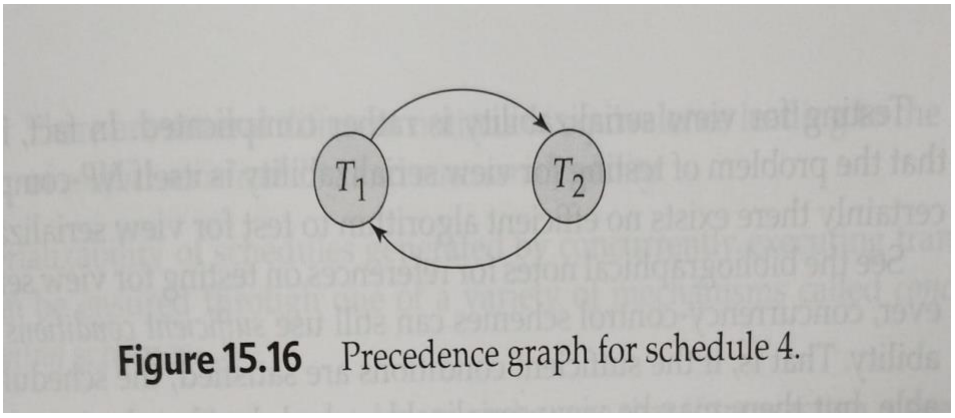
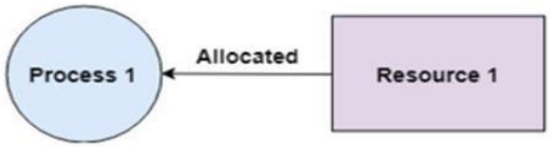


Figure 15.16 Precedence graph for schedule 4.

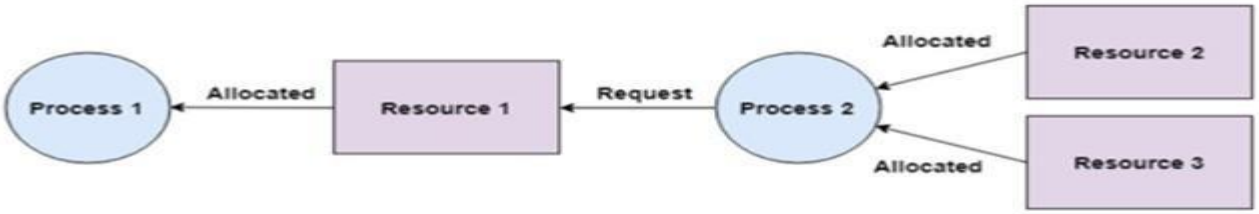
Deadlock:

- A deadlock happens in DBMS when two or more transactions need some resource to complete their execution that is held by the other transactions.
- A deadlock occurs if the four Coffman conditions hold true. But these conditions are not mutually exclusive. They are given as follows −
    1. **Mutual Exclusion**
    2. **Hold and Wait**
    3. **No preemption**
    4. **Circular Wait Mutual Exclusion:**

There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



Hold and Wait: A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.

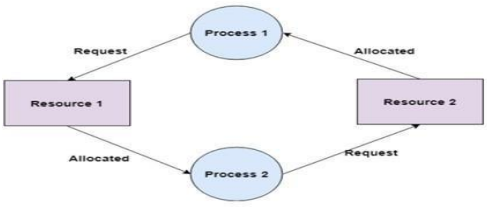


No Preemption:

A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete

Circular Wait:

A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



Methods of handling deadlocks:

- Deadlock detection
- Deadlock prevention
- Deadlock avoidance are the main methods for handling deadlocks.

Deadlock Detection

Deadlock can be detected by the resource scheduler as it keeps track of all the resources that are allocated to different processes. After a deadlock is detected, it can be handed using the given methods

- All the processes that are involved in the deadlock are terminated. This approach is not that useful as all the progress made by the processes is destroyed.
- Resources can be pre-empted from some processes and given to others until the deadlock situation is resolved.

Deadlock Prevention:

It is important to prevent a deadlock before it can occur. So, the system checks each transaction before it is executed to make sure it does not lead to deadlock. If there is even a slight possibility that a transaction may lead to deadlock, it is never allowed to execute.

Some deadlock prevention schemes that use timestamps in order to make sure that a deadlock does not occur are given as follows −

Concurrency:

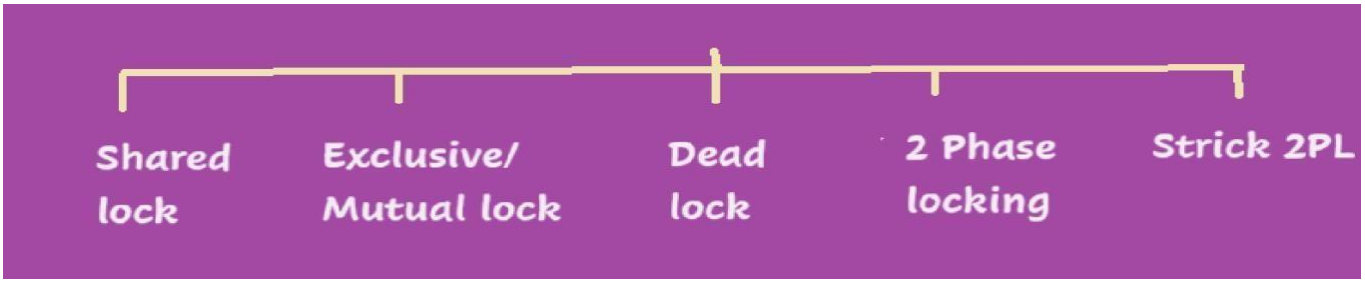
1. Lock is a protocol which maintains a set of rules.
2. Locking techniques is always applied on database objects.
3. When two transactions want to access the same object simultaneously, then conflicts arises, then locking technique is applied in such situations.
4. If T1, T2 wants to access object A from disk, then T1 locks object A. So that after completion of T1, then T2 uses object A. So that the arised conflict will be reduced/removed.
5. Locking techniques are used to attain concurrency.
6. Locking techniques are divided into the two types

1. Concurrency control with lock based techniques

2. Concurrency control without lock based techniques

1. **Concurrency control with locking techniques**:

It is again divided into following types:

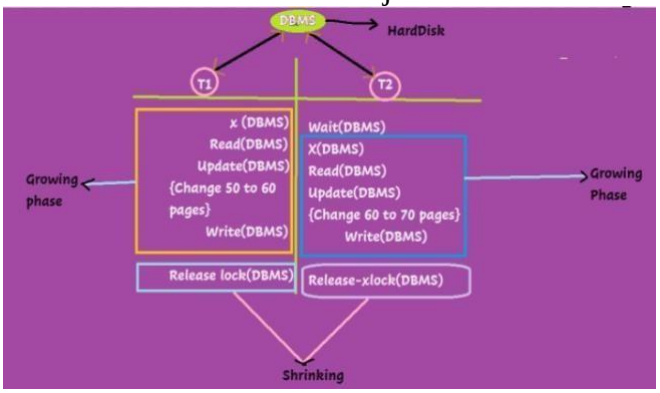


Shared Lock:

When two users are accessing the same object simultaneously, then shared lock will be applied on that particular object so that the object will be used by both of the users. Shared lock is denoted as 'S'. Shared lock applied on the object is used only for Read mode by the users no other operations can be done. So, we shift to exclusive lock.

Exclusive Lock:

It is represented as 'X'. It is applied on particular object of the data base so that the object can under go updation. This lock is applied on more than one object (or) transactions. If T1, T2 wants to update DBMS text of initially 50p to 60 and 70 pages respectively. So, firstly 'X' lock is applied by T1 so that T2 cannot access the object if, DBMS.



After updating the text from 50 to 60 pages, then T1 releases lock. Then T2 can access the object. Now T2 update the text to 70 pages and write it to the hard disk, or secondary memory. Then it releases the lock,
for the object i.e., DBMS. If another transaction exists then it is send to it. Here mutual understanding exists between transaction.

2 Phase Locking:

2PL stands for 2 phase locking. To implement 2 PL we require 2 phases i.e., Shrinking and growing phases.

Growing phase:

According to growing phase, any transaction may obtain locks, but not release any locks.

Shrinking phase:

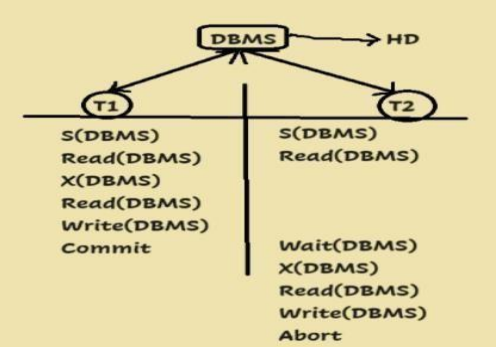
According to shrinking phase, any transaction may releases locks, but not obtain any new locks.

If we take the before example of transactions in exclusive lock, the lock obtaining and updating goes to growing phase and releasing a lock goes to shrinking phase.

The main drawback of 2PL is, it does not specify any transaction as successful or failure.

Strick 2PL:

In stricks2PL, we can identify any transaction is success and failure by using commit and abort respectively. To implement Strick 2PL, one need to have more than one transaction. Strick 2PL applies initially shared lock on object. As shared lock is used only for Read mode, here the read mode appears. But for operations, we use X-lock.. If the transaction is successful, then commit. Now send the object to other. If transaction is not successful, then abort. Strick 2PL maintains S-lock and X-lock. 2PL supports only X-locks.



Deadlock:

When two transactions are trying to access the same object at same time from hard disk, into private area, at that time there arises a conflict called as deadlock. By using X-lock, deadlock prevention and deadlock avoidance techniques, we can overcome the deadlocks.

Binary lock:

It maintains either 0 or 1 i.e., false or true. If 1 is used then info is overloaded into private area/transaction area. If 0 is used. If it is in waiting state, so likewise the binary lock is implemented.

2. Concurrency control without locking techniques :

- Here concurrency can be overcome by not using any lock techniques as shown below :
- The lock manager is responsible for maintaining locks on objects , so that locks can use or not used on the objects .Lock manager looks after whether the locking techniques are applied on the object properly or not .

i) Optimistic concurrency control :

- **Read phase :** All the transactions after completion will be stored in secondary memory. Optimistic concurrency will work on secondary memory. It checks the transactions whether it is useful for

Sec.mem

T1	T2
T3	T4

On concurrency or not. It gathers the particular Required transaction from secondary memory.

To private area by using read phase. That retrieved transaction has read and write modes in it.

ii) Validation phase:

Now the retrieved transaction say T1. T1 is given to the validation phase validation phase checks T1, whether it is conflict or non-conflict transaction. Conflict means more than one transaction trying to access same object at same time. Non-Conflict means a transaction gets completed by individual accessing of the object. If T1 is conflict transaction, then it sends back T1 to read phase.

If T1 is non-conflict transaction, then T1 is send to the write phase. Validation phase maintains some conditions as stated. **iii) Write phase:**

Write phase sends any tested transaction from private area to secondary memory. So, now the tested T1 will be send again to the hard disk for storing purpose.

Time Stamp based:

To implement this technique we use 2 different transactions Ti and Tj.

Each and every transactions needed to enter into execution. For this we initially require a system.

At the time of entering any transaction into the system, then each transaction is assigned some period for execution. This is called time stamp.

Time stamp is represented as TS(Ti) and TS(Tj).

TS is always maintained some mandatory condition, TS(Ti)<TS(Tj). After entering Ti transaction into the system, then Tj will be entered .

Eg: If TS(Ti) = 12.00 then TS(Tj) = 12.01, now the above condition is satisfied.

The transaction enters into the system, only when the given time matches to the time of system clock or logical clock.

After entering the transaction, to execute the transactions properly under cpu, two additional TS are taken i.e., Read TS (RTS) and write TS (WTS).

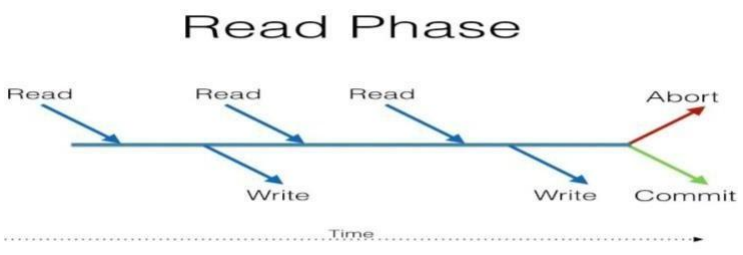
Ti want to execute under CPU needed to follow some issues of read time stamp if it is completely executed under CPU.

1. If TS(Ti)<WTS(X) particular object 'x'. Here only completed read mode but not the write operation the transaction is failure, now need to restart the transaction.
2. If TS(Ti)>WTS(X), here the issues of the write(x) completed the R&W operations the transaction is successful and if R(X), W(X) > W(X) given chance to next transaction **Ti issues write Time stamp:**
    1. If TS(Ti)<RTS(X) then here not completed the R operation successfully and also W operation successfully. Initially any failure transaction send to the failure state.
    2. If TS(Ti) < WTS(X) then completed the R(X) operation but not write operation. So the corresponding transaction send to the abort state.
    3. If TS(Ti)>WTS(X) then completed the Ti transaction successfully.

Note: Tj follows the same rules as Ti.

Optimistic Concurrency Control:

i) **Read phase :**



All the transactions after completion will be stored in secondary memory. Optimistic concurrency will work on secondary memory. It checks the transactions whether it is useful for concurrency or not. It gathers the particular Required transaction from secondary memory to private area by using read phase. That retrieved transaction has read and write modes in it.

ii) Validation phase:

Now the retrieved transaction say T1. T1 is given to the validation phase validation phase checks T1, whether it is conflict or non-conflict transaction. Conflict means more than one transaction trying to access same object at same time.

Non-Conflict means a transaction gets completed by individual accessing of the object. If T1 is conflict transaction, then it sends back T1 to read phase.

If T1 is non-conflict transaction, then T1 is send to the write phase. Validation phase maintains

some conditions as stated.

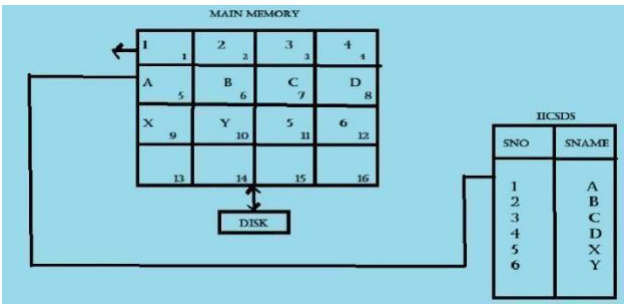
iii) Write phase:

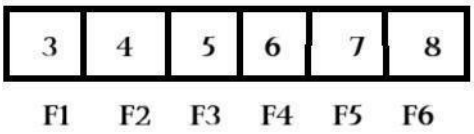
Write phase sends any tested transaction from private area to secondary memory. So, now the tested T1 will be send again to the hard disk for storing purpose .

BUFFER MANAGER (Storage):

- Buffer manager is responsible for swap out.
- It is a software layer (predefined application program).
- It is always appeared on main memory. This software layer responsible for bringing required pages from secondary memory to main memory.
- Let us consider available main memory which is divided into different equal partitions each partition or cell is called a frame or page.
- The process of dividing main memory into equal partitions is called as Buffer pool, each frame has 2 variables. They are 'pin count' and 'dirty'. Each frame has a sequential numbers. Before retreiving assign pincount as zero.
- Dirty variable assigned to boolean values i.e, 0 or 1. If the user wants to retrieve any frame into from main memory by user. So that the pin count will be incremented by 1.
- If user 2 also wants the same frame selected by user1, then again the pin count increments by 1 and so on..

- If any user wants to retrieve the data that is unavailable in main memory. Then main memory sends request to secondary memory for the required data and places in it (main memory).
- If there is no place in main memory for replacement of particular page from secondary memory, then page.



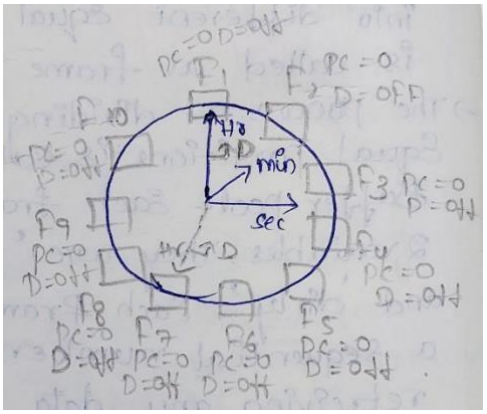


Page replacement techniques are used. They are LRU (Least recent used), MRU (most recently used), clock techniques.

- According to LRU, available page pincounts are arranged in queue order. Here F6 is LRU. So, we remove data from F6 (i.e, p.c = 0).Now the required information of secondary memory can be replaced in f6 of main memory . If any replacement is done , then dirty variable goes to ON state in that particular frame . Here, in F6 the dirty variable will gets actived. Now the frame/page is treated as 'disk page' and blocked.

- According to MRU, F3 is most recently used frame. so,F3 data will be removed and the required into can be filled. Then Dirty variable of F3 goes ON state and that frame is called 'Disk page " and gets blocked.

CLOCK TECHNIQUES:

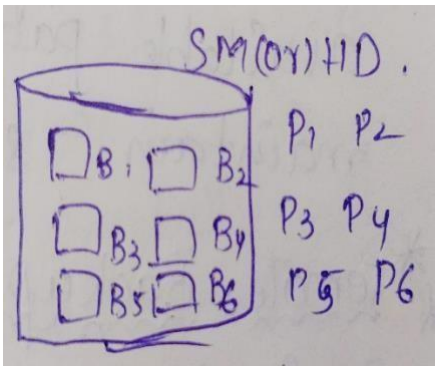


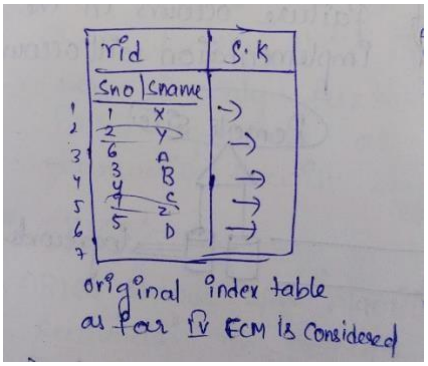
- To implement a clock technique, a circle is needed. That circle may be called as a clock. It has 3 needles .i.e, hours, minutes and seconds needles .All the frames of main memory is sent on o that circle, each and every frame has pincount, which is intially zero and dirty variable ,which is intially OFF and all frames also has an additional dirty variable, which is attached to any one of the 3 clock needles. Now that Dirty variables is rotated in clockwise direction inclock.
- That needle will stop at some frame number the dirty variable will go to ON state that page data will be removed and new data is replaced in the page and that page is treated as disk page.

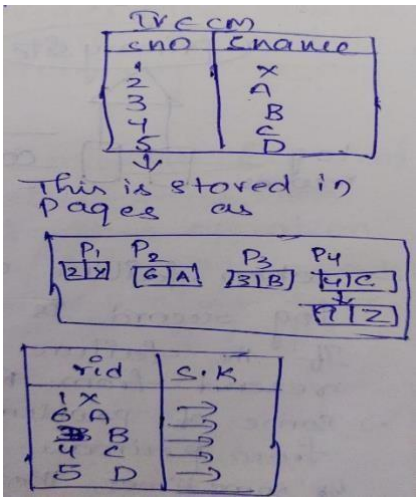
Shadow Paging Technique:

- It is the process of dividing available secondary memory into different block. It is also applied on hard disk.

- Secondary memory is divided into partition called "Block". Here we need pages if Secondary memory don't have pages they are retrieved from main memory through OS as shadow paging technique does not retrieve them so here each and every block in the secondary memory is changed to pages to update any page info of secondary memory we go for shadow paging technique.
- Consider, ex 1x should be updated to 2y, shadow paging technique initially contains maintains "index page table" can "original page table". This index page has <rid, S.K> record identifier, search key.
- This info is obtained from IV ECM table sent from main memory to secondary memory. Shadow paging technique directs OS to original page from main memory to secondary memory.
- Shadow page technique is responsible to take a copy of original 1PT called shadow page.
- Any updations or modifications are done only on original index table but not on shadow page. If changes are done on original table, it is reflected on secondary memory but not on shadow page.
- So the advantage of shadow page is that we can have a copy of original information, so any changes in future can be done.







Failure Classification:

Logical Failures:

These failures cannot be detected by the system i.e., they are syntactically correct but logically incorrect

Eg: 0, 00 etc., such cannot be find by system at compilation.

System Error:

Two transactions are trying to access same object at the same time, then the system goes into

deadlock state (or) system error. **System Crash:**

Every system comprises of os, when the system consists of disk failures, software failures or os failures, then such situations is called system crash.

Disk failures:

To fetch (or) insert any data, Read and write operations are used. Those operations are maintained by hard disk. If any failure (dust, corrosion, etc) exists in Read and write heads then disk failures occurs hard disk info is loaded into a spread sheet, which consists of rows and columns.

Memory Storage:

All four types of failures comes under memory storage. Storage is of three types. They are

- Volatile
- Non-volatile
- Stable

Volatile:

- It is fast accessing storage device. RAM, (cache, main memory) supports the volatile storage. These are temporary storage devices. So, we can modify the data under volatile storage device. It acts as a buffer.

Non-volatile:

- Secondary memory supports non-volatile storage device, large amount of data can be stored.
- It is a permanent storage device. Info will be exchanged from secondary to main memory or vice versa by using swap in and swap out commands.
- Swap In is used to exchange info from secondary, to main memory. Swap out is from main memory to secondary memory.

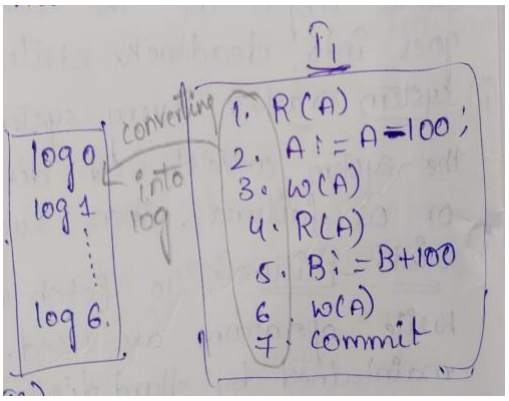
Stable storage:

- Successful transactions will be moved to the stable storage for storing. Durability property maintains stable storage. The stable storage consists of data, which can be modified in future.

Recovery and Atomicity:

- Recovery manager maintains recovery techniques such as following **Log Based Recovery technique:**
- It converts normal transaction elements to log records. It is applied mostly on transactions when normal elements are converted to log records, then storage space under memory will be reduced.

This technique maintains log sequence number order. Log elements are always written in angular brackets (< >)



Eg: <Ti, start>; I= 1 ton.

Requires to maintain 4 different values.

Transaction identifier

Data identifier

Data old value/old data value.

New data value.

The above T1 transaction can be constructed using a single statement as:

<Ti, start>

<Ti, A, V1, V2>

<Ti, A, 500, 400>

<Ti, B, 600, 700>

<Ti, commit>

Where, Ti is Transaction identifier

V1 is old value, V2 is new value

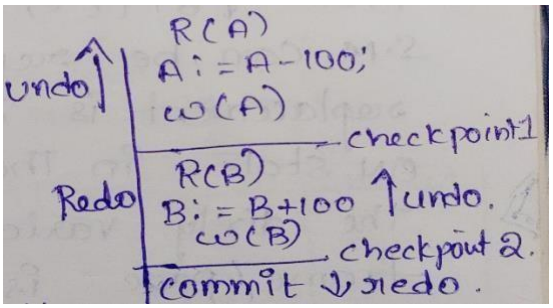
A is data identifier

If the transaction is not successful then, <Ti, abort>

Check Point Recovery Technique:

- It is a disk recovery technique when disk failures or disk crash or power failures or mechanical failure occur at the middle of transaction, then the check point recovery technique is used for recovery purpose. This technique uses undo and redo operations
- Previous statements are assigned to 'undo' operation by check point. So, the previous statements cannot be lost due to any failure or errors.
- Wherever the failure occurs, 'checkpoint' is stated or assigned at that particular statement where failure occurs.

- After restarting the system, if we want to continue the transaction from failure onwards is done by the operation 'Redo'.
- We can assign as many checkpoints as we can to a transaction. The statements before checkpoint is preserved by undo and continuation is done by using Redo.
- Checkpoint detects mechanical and power failures.



INTRODUCTION TO INDEXING TECHNIQUES

B+ Trees:

- B+ trees are a type of self-balancing tree data structure that is widely used in database management systems and file systems. They are designed to optimize disk I/O operations, which are crucial for efficient data retrieval and storage.

Inventor:

- The B-tree, the predecessor of the B+ tree, was invented by Rudolf Bayer and Edward M. McCreight in 1972.

Key Characteristics:

Structure:

- B+ trees consist of root nodes, internal nodes, and leaf nodes.
- Leaf nodes contain the actual data records and are linked together in a sequential manner, forming a linked list.

Balanced Structure: All leaf nodes are at the same level, ensuring consistent access times for all data records

High Fan-out:

- Each node can have a large number of child nodes, reducing the height of the tree and minimizing disk I/O operations.

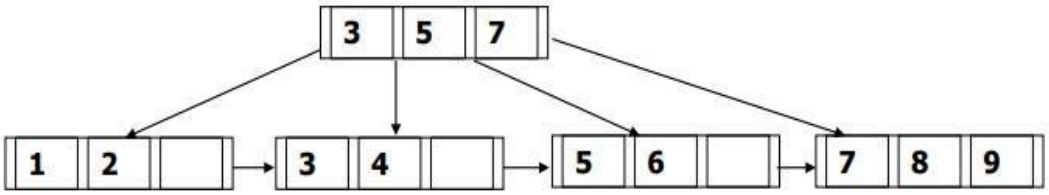
Sequential Access:

- The linked list of leaf nodes enables efficient sequential access and range queries.

Height:

The height of the tree is kept relatively small due to the high fan-out, resulting in logarithmic search times (O(log n)).

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



B+ Trees Example:

A balanced tree

- Each node can have at most m key fields and m+1 pointer fields
- Half-full must be satisfied (except root node):
- m is even and m=2d
  - – Leaf node half full: at least d entries
  - – Non-leaf node half full: at least d entries
  - • m is odd and m = 2d+1
  - – Leaf node half full: at least d+1 entries
  - – Non-leaf node half full: at least d entries (i.e., d+1 pointers)

Show the tree after insertions:

- • Suppose each B+-tree node can hold up to 4 pointers and 3 keys.
- • m=3 (odd), d=1
- • Half-full (for odd m value)
- – Leaf node, at least 2 (d+1) entries
- – Non-leaf nodes, at least 2 (d+1) pointers (1 entry)
- • Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

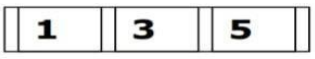
Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

- • Insert 1

- Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

- Insert 3, 5
- Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

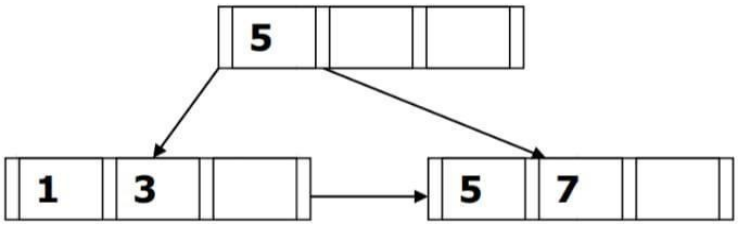
- Insert 3, 5



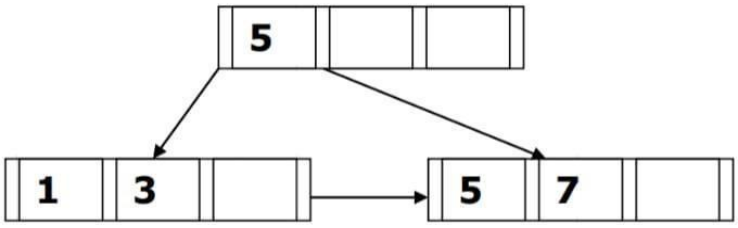
- Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

- Insert 7
- Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

- Insert 7

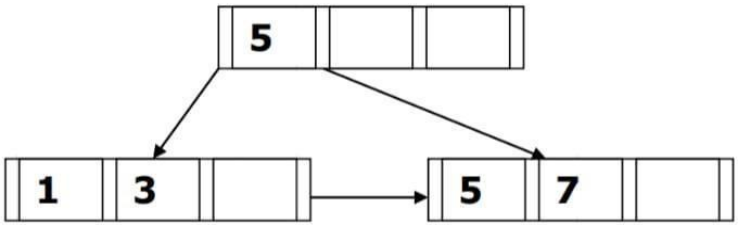


Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

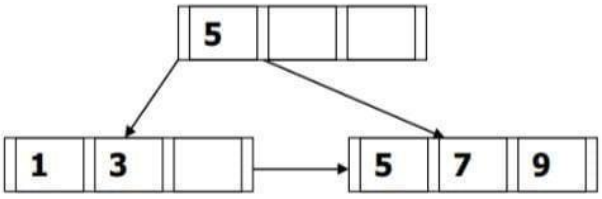


- Insert 9

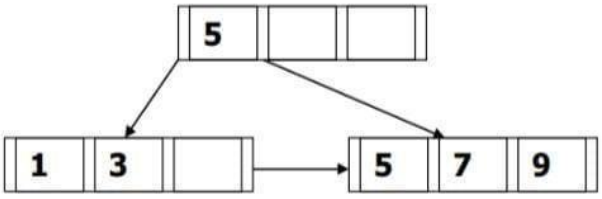
Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10



• Insert 9

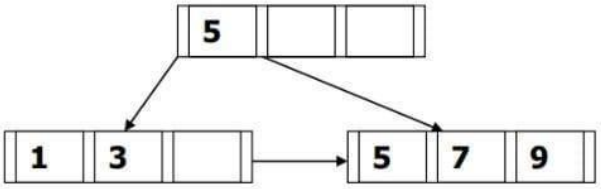


Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

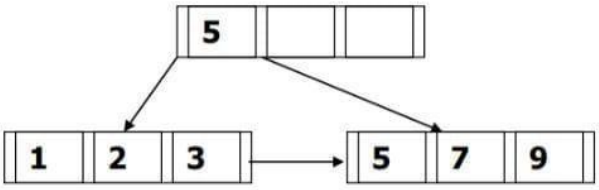


• Insert 2

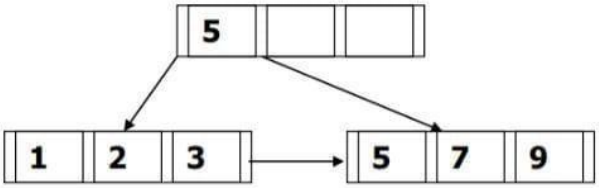
Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10



• Insert 2

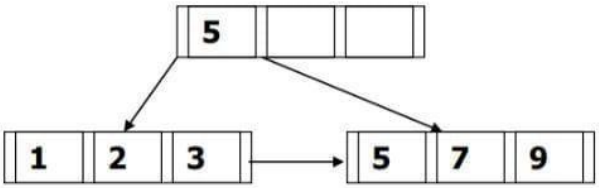


Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

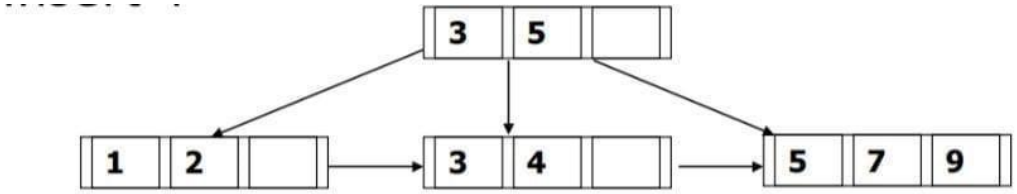


• Insert 4

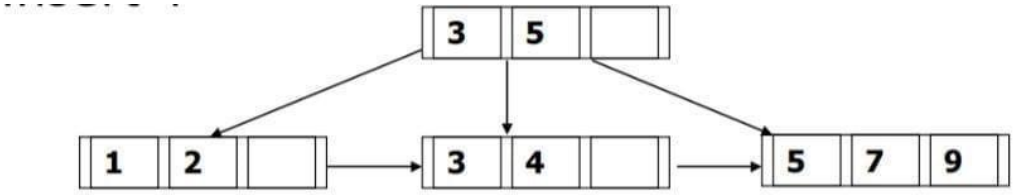
Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10



• Insert 4

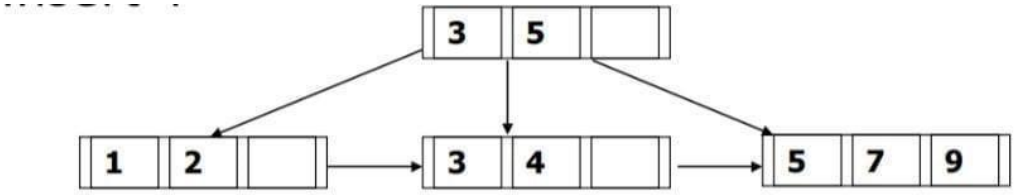


Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

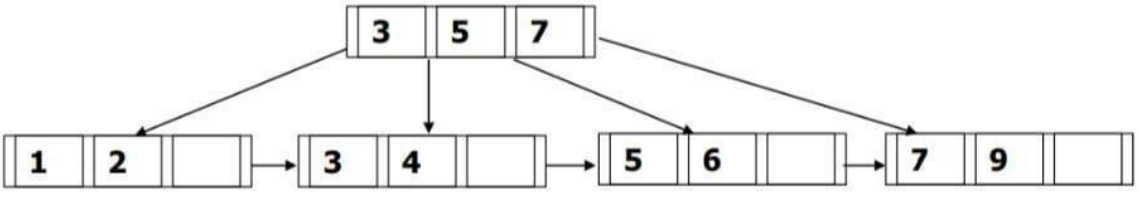


• Insert 6

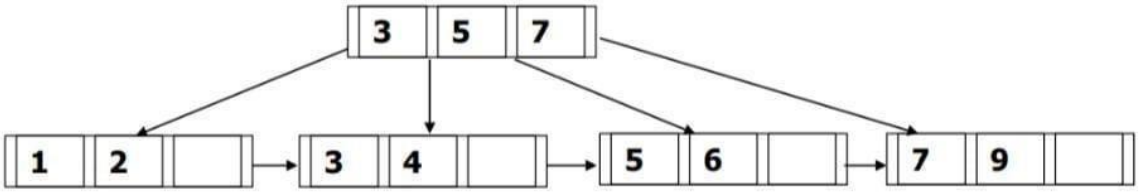
Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10



• Insert 6

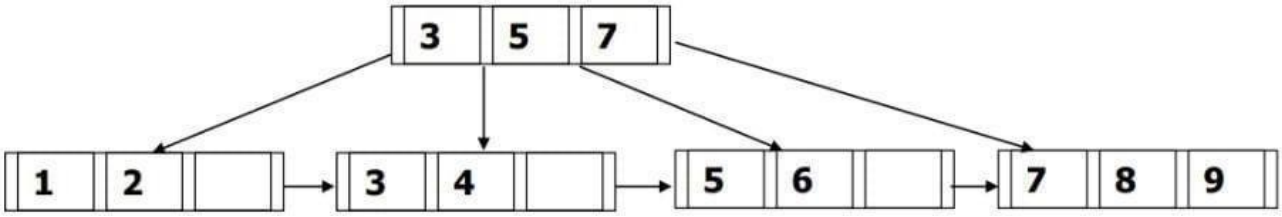


Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

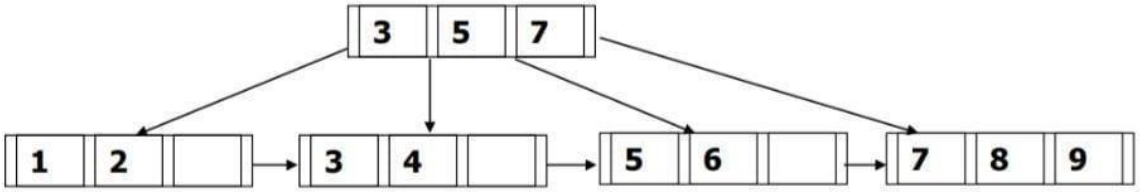


• Insert 8

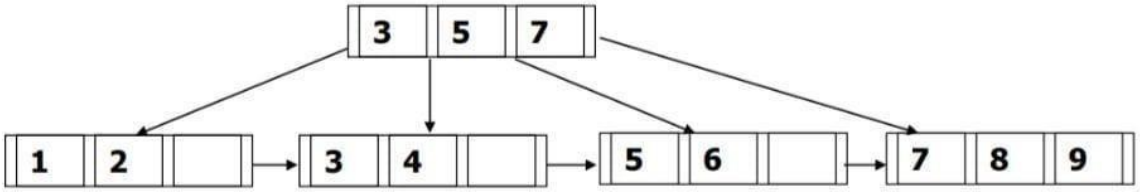
Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10



• Insert 8

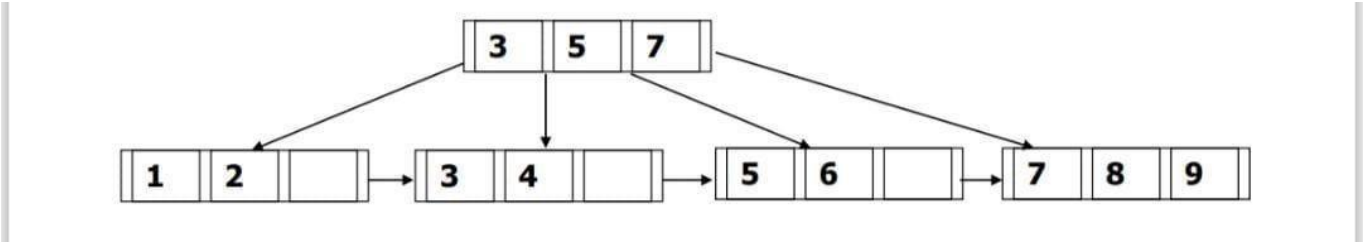


Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

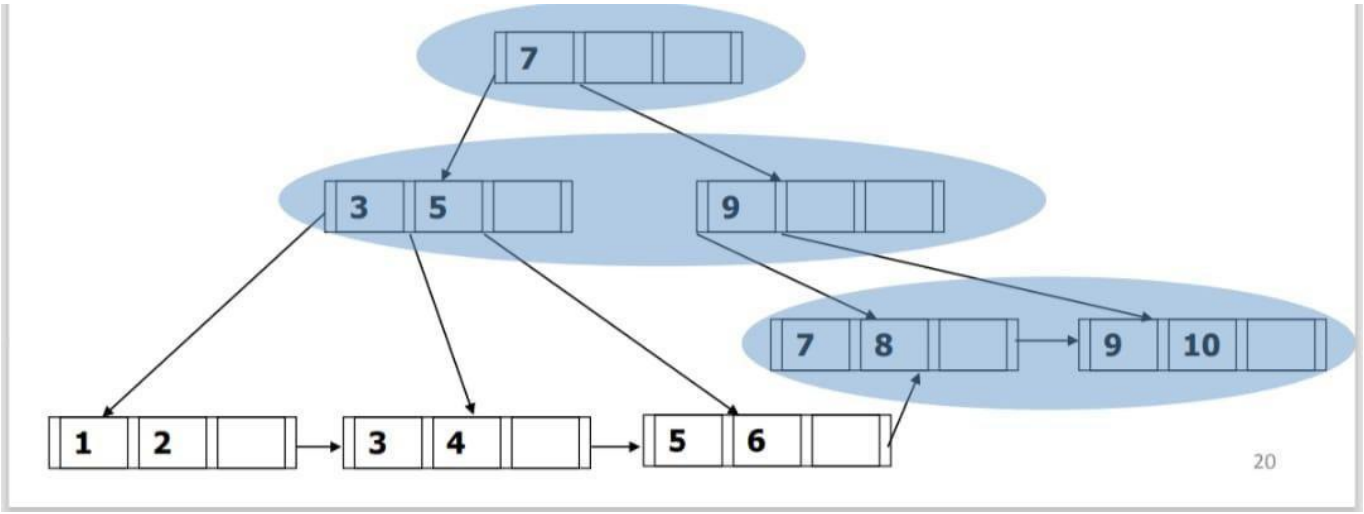


• Insert 10

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

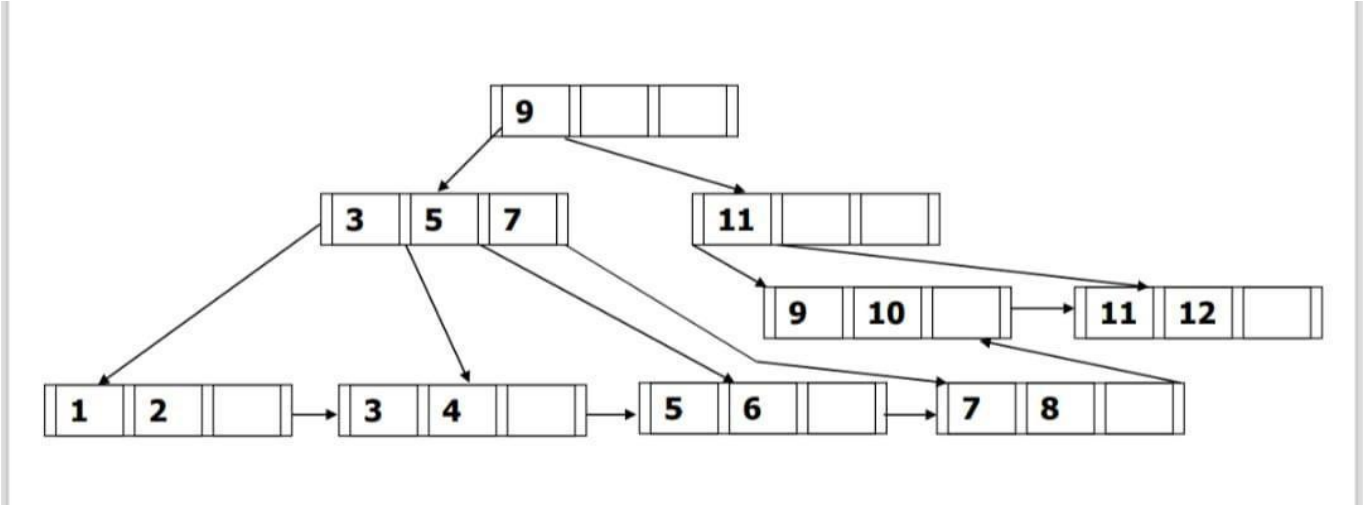


• Insert 10

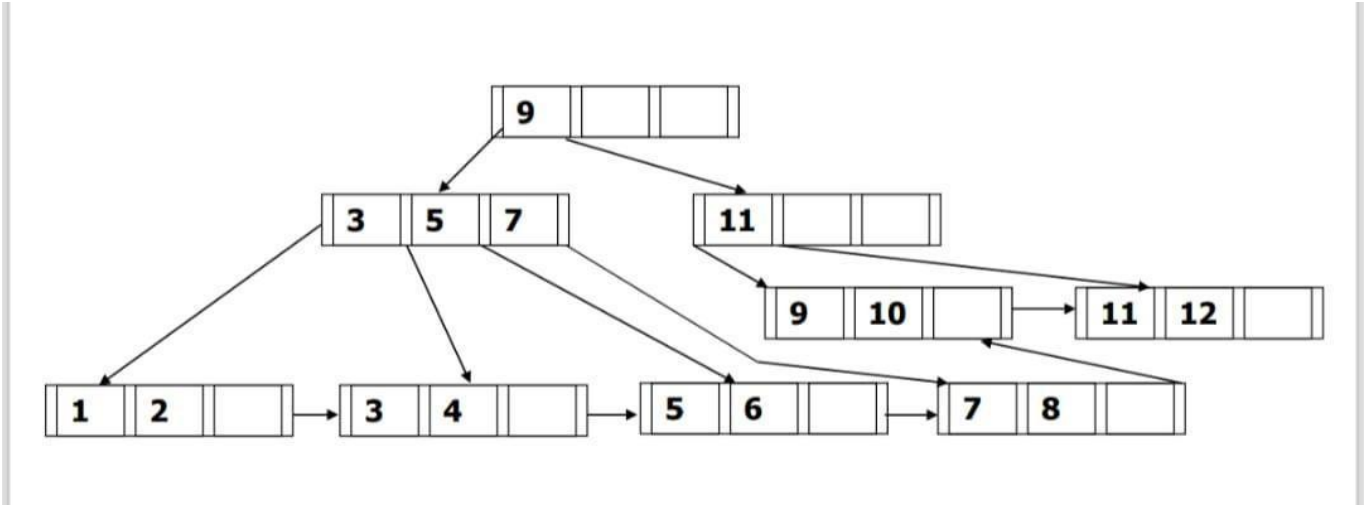


• Deletion

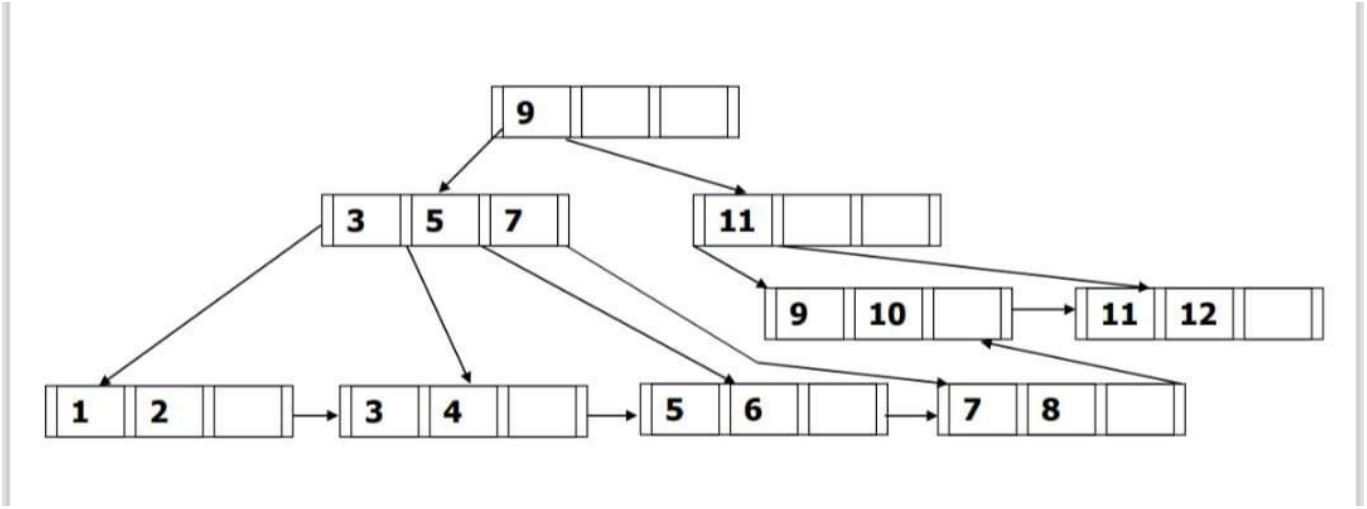
Show the tree after deletions



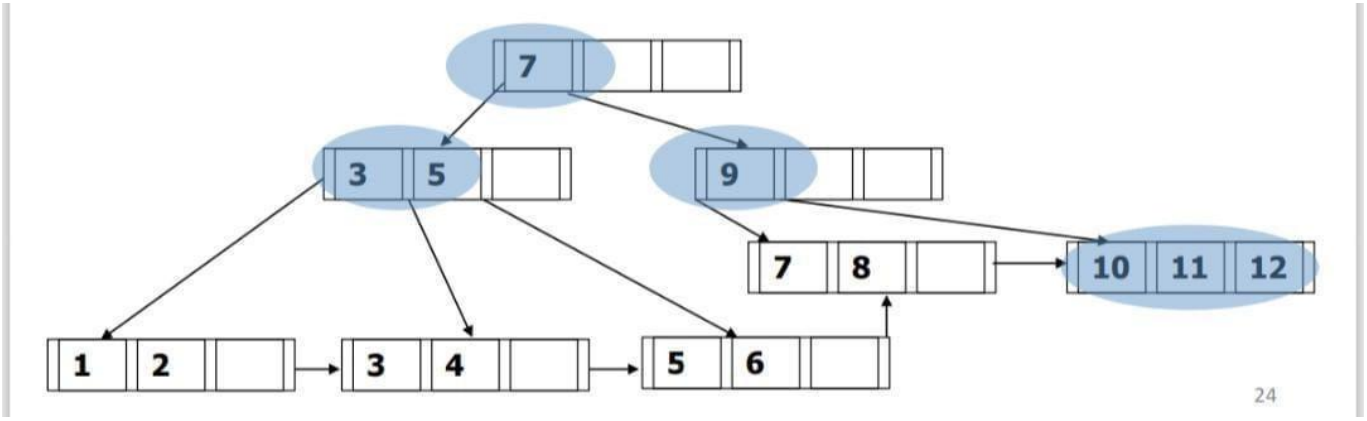
• Remove 9, 7, 8



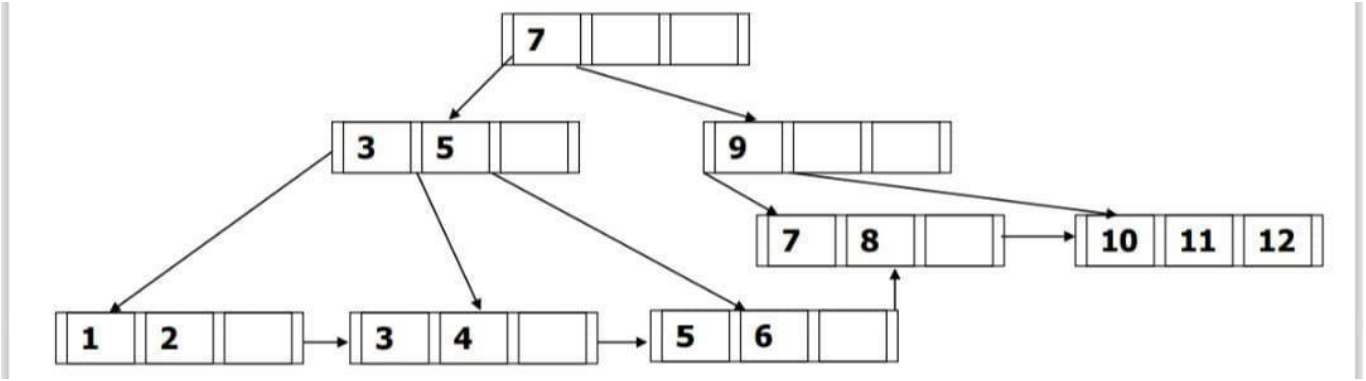
• After removing 9



• After removing 9

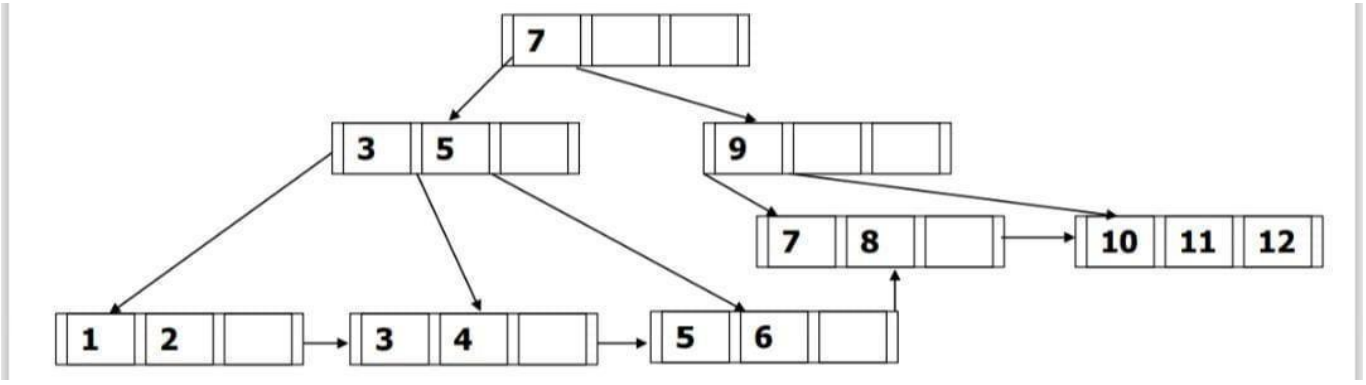


Remove 9, 7, 8

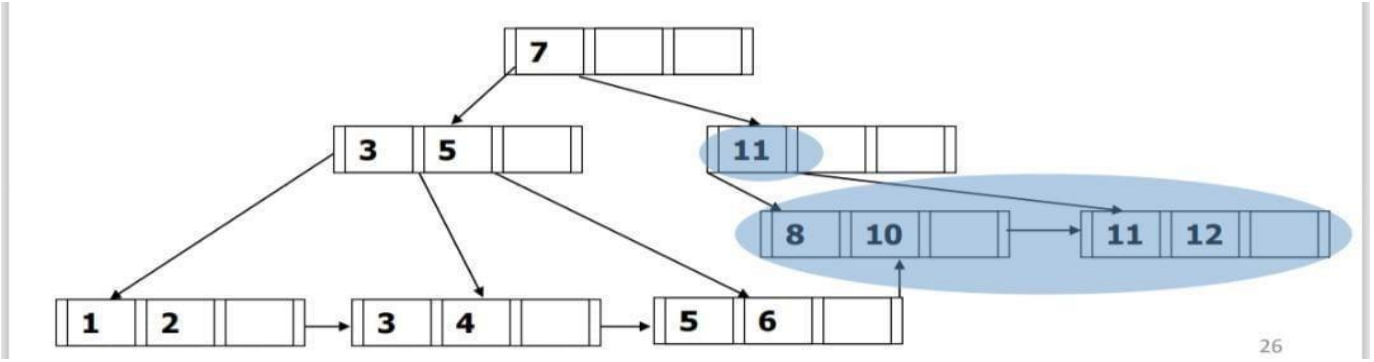


• After removing 7

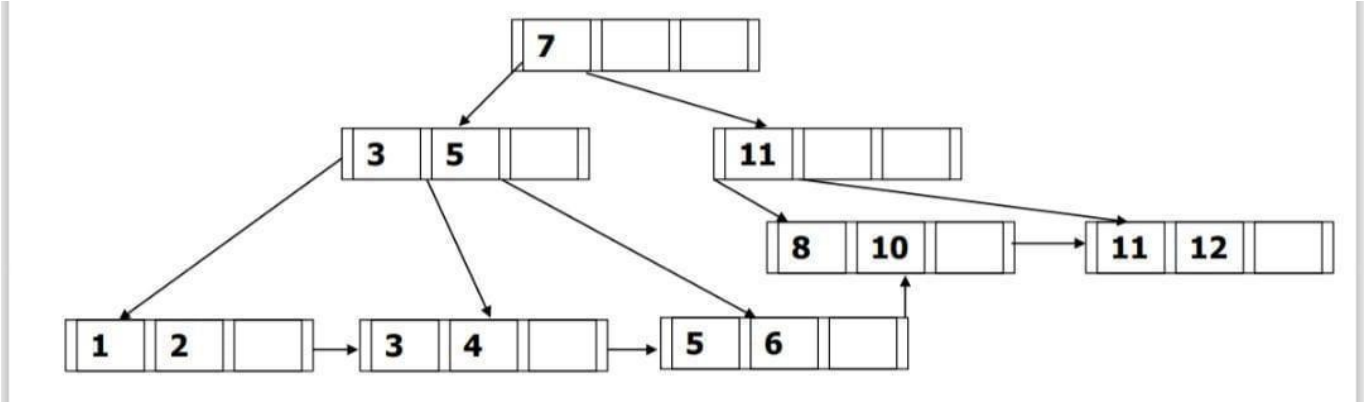
Remove 9, 7, 8



• After removing 7

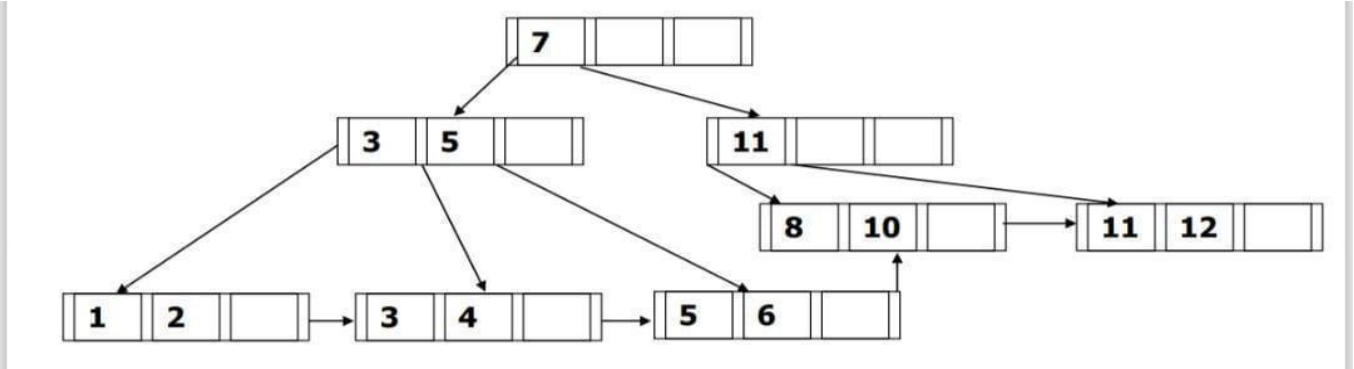


Remove 9, 7, 8

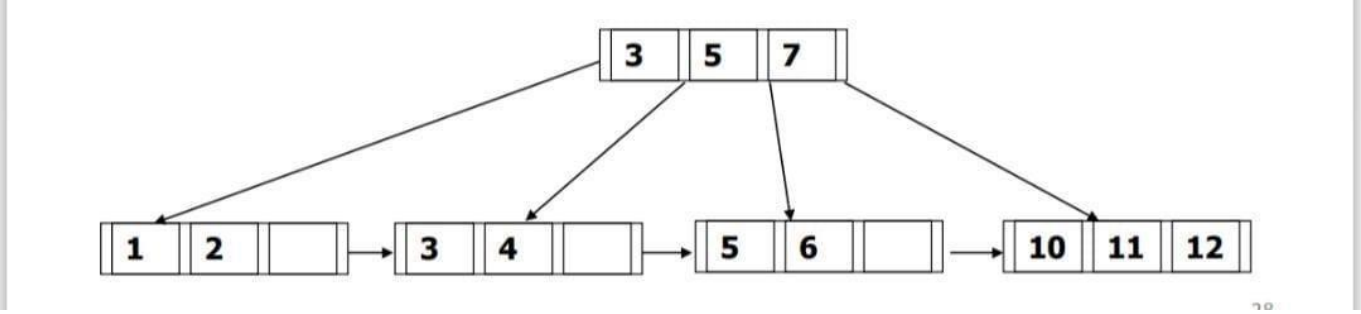


• After removing 8

Remove 9, 7, 8



- After removing 8



🔑 Hash-Based Indexing

Hash-based indexing is a **data retrieval technique** that uses a **hash function** to compute the address (or index) of a record based on its key. It is commonly used in **database management systems (DBMS)** to enable **fast access** to records. **How It Works**

1. **Hash Function:** A function converts a key (e.g., student ID) into a hash value (bucket index).
2. **Buckets:** The hash function assigns records to different "buckets" (memory locations).
3. **Collision Handling:** If two keys hash to the same bucket, techniques like **chaining** (linked lists) or **open addressing** (probing) handle them.

4. **Overflow buckets**

Visualization Example: Hash Indexing

Consider a **student database** where we store student IDs (key) using **hash indexing**.

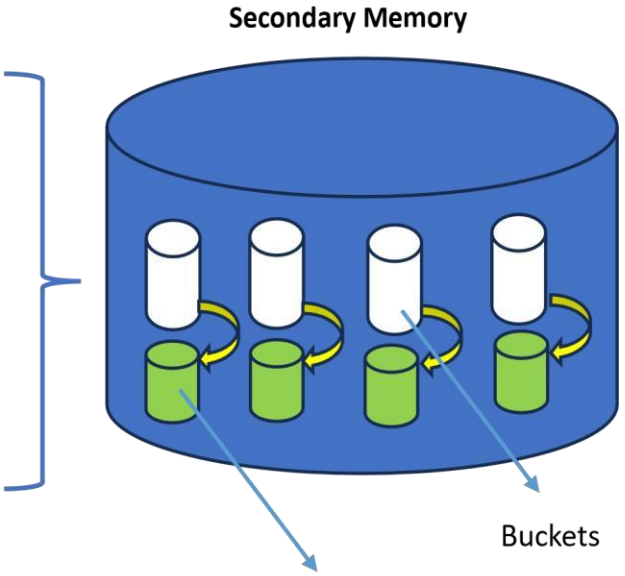
1 Hash Function (Modulo Method)

We use the **modulo hash function**:

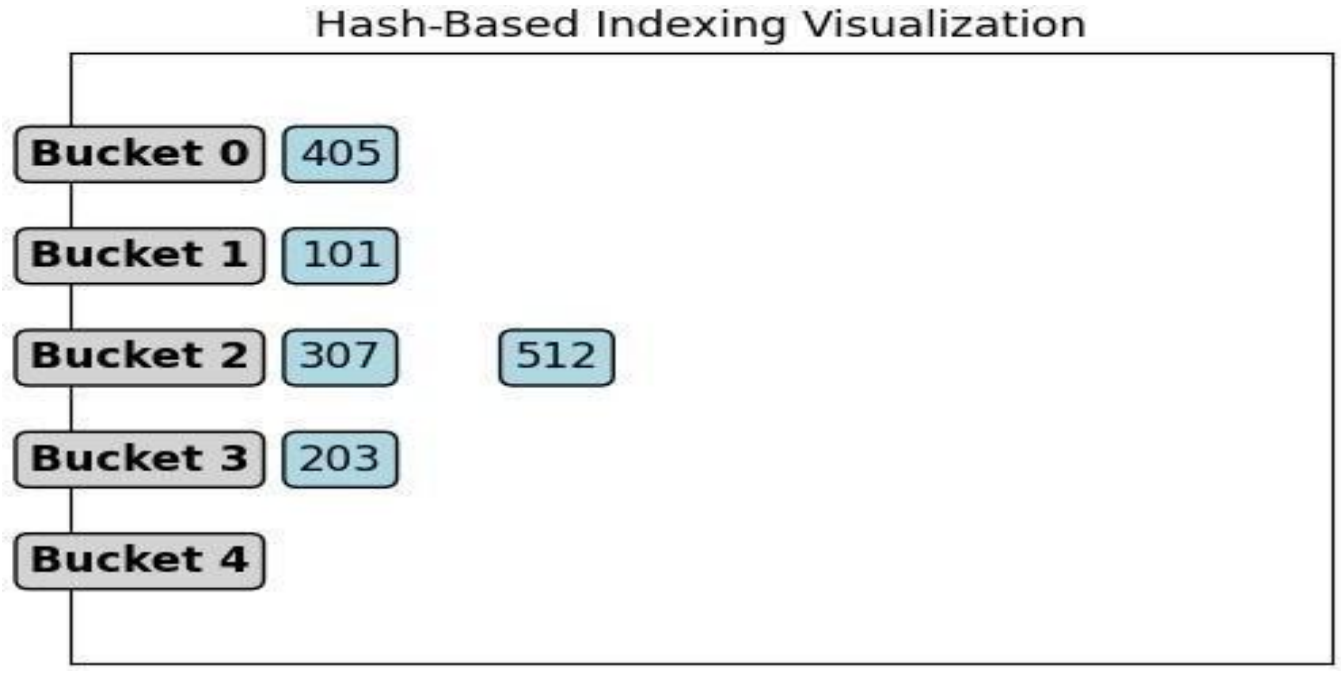
$$\text{Hash(Key)} = \text{Key} \mod \text{Number of Buckets}$$

For example, with **5 buckets**, student IDs are stored as follows:

Student ID	H(K) → (ID%5)	Pointer→	Bucket
101	101%5	1	1
203	203%5	3	3
307	307%5	2	2
405	405%5	0	0
512	512%5	2	2



Here 512 and 307 collide in Bucket 2



Feature	B+ Trees	Hash Based
Structure	Tree-like structure with nodes	Key-value pairs stored in a hash table
Best for	Range queries & sorted data	Exact match queries
Range Queries	Fast and efficient	Not supported
Ordered Data	Maintains sorted order	Does not maintain order
Storage	Uses more disk space due to tree structure	Requires less space but may have collisions
Collisions	No collisions	May occur, requires handling
Use Case	Used for indexes, primary keys, and sorted searches	Used for lookups like caching and hashing based indexes