The purpose of the Software Development Life Cycle (SDLC) is to generate excellent software that meets all customer expectations and requests. The SDLC is an organized process that makes it possible to produce high-quality, low-cost software in the shortest amount of time.

We have different SLDC models:

Waterfall Model

Prototype Model

Spiral Model

Iterative Increment Model

V model

**Waterfall Model**

it is useful for the small projects and also next stage cant begin with out the completion of intial phase every stage output will be input to next stage. and after completion of all stages the product will be tested

advantages**:**

it is easy to implement this model

it works well for short project

all phases are clearly defined

disadvantages**:**

its not suitable for complex projects

as it’s a sequential approach the it takes more time because each phase should be completed before moving to the next phase

**Example of Waterfall Model is ATM**

**Iterative Model**

For every iteration additional features are getting added.and for every iteration involves coding and testing the software product. software product will get enhanced as we will add new features to the iteration.

Inception phase: Gathering information

Elaboration Phase: adding new features

Construction Phase: In the Construction phase, the architecture is filled in with the code which is ready to be deployed and is created through analysis, designing, implementation, and testing of the functional requirement.

Transition Phase: product will be deployed

Advantages

Adaptability to Changes

Early Risk Identification

Early Defect Detection

Disadvantage

Complete requirments and understanding of products is nesserary to build incrementally

**Example of iterative model is developing mobile phones over the years.**

**Prototype Model**

Prototype is developed prior to the actual software so that uses can inform drawbacks and we can improve the performance of the model.

Advantages

We can add missing features and functionality

Disadvantage

Since customer can chage the requirments it increases the complexity of the Scope.

**Spiral Model**

Each cycle involves the same sequence of steps as the waterfall process model

advantages

Users can be closely tied to all lifecycle steps . Early and frequent feedback from users . Cumulative costs assessed frequently.

Disadvantages

The model is complex. Risk assessment expertise is required.Spiral may continue indefinitely

**V-Shape**

Testing of the product is planned in parallel with a corresponding phase of development

Advantages

Emphasize planning for verification and validation of the product in early stages of product development. Each deliverable must be testable

Disadvantages

Does not handle iterations or phases

There are the various phases of Verification Phase of V-model:

1. **Business requirement analysis**
2. **System Design**
3. **architecture Design**
4. **Module Design**
5. **Coding Phase**

**Agile Model**

The meaning of Agile is versatile. Agile process model refers to a software development approach based on iterative development. Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning.

**Phases of Agile Model:**

Requirements gathering

Design the requirements

Construction/ iteration

Testing/ Quality assurance

Deployment

Feedback

**SCRUM**

scrumis an Agile framework for managing and organizing work on complex projects, primarily used for software development but applicable to various fields.

Product Owner

Scrum Master

Development Team

**Events:**

Sprint**:**

Sprint Planning

Daily Scrum

Sprint Review:

Sprint Retrospective

**Benefits of Scrum Model in SDLC**

Flexibility and Adaptability

Customer Satisfaction

Early and Predictable Delivery

Improved Collaboration

Increased Transparency

—--------------------------------------------------------------------

### 1. Git

Git is a distributed version control system that allows developers to track changes in their code over time. It helps in collaborating with other developers by allowing multiple people to work on the same project simultaneously and keeps a history of every version of a project.

* **Benefits of Git**:
  + Tracks changes and maintains a history.
  + Supports collaboration.
  + Allows for branching and merging, enabling experimentation without affecting the main codebase.

### 2. Git Bash

Git Bash is a command-line interface that lets you use Git commands on Windows. It provides a Unix-like terminal with Git installed, so you can run Git commands as if you were in a Linux environment.

* **Usage**:
  + It’s mainly used on Windows to provide an environment similar to Linux.
  + Supports both Git commands and some Unix/Linux commands.

### 3. GitHub

GitHub is a cloud-based platform for hosting Git repositories. It provides a web interface and additional features, like issue tracking, project management, and collaboration tools, which make it easier for teams to manage their Git-based projects.

* **Features**:
  + Online storage for repositories.
  + Collaboration through pull requests, issues, and wikis.
  + GitHub Actions for automated workflows.

### Basic Git Commands

**Initializing a Git Repository**bash  
Copy code  
git init

1. Initializes a new Git repository in the current directory.

**Cloning a Repository**bash  
Copy code  
git clone <repository-url>

1. Copies a remote repository to your local machine.

**Checking Status**bash  
Copy code  
git status

1. Shows the status of changes, indicating modified, untracked, or staged files.

**Adding Changes**bash  
Copy code  
git add <file>

git add .

1. Stages changes for a specific file or all files to be committed.

**Committing Changes**bash  
Copy code  
git commit -m "Commit message"

1. Records the staged changes to the repository with a message describing the changes.

**Viewing Commit History**bash  
Copy code  
git log

1. Displays the commit history of the repository.

**Branching**bash  
Copy code  
git branch <branch-name>

git checkout -b <branch-name>

1. Creates a new branch or switches to a specific branch.

**Merging Branches**bash  
Copy code  
git merge <branch-name>

1. Merges changes from one branch into the current branch.

**Pushing Changes**bash  
Copy code  
git push origin <branch-name>

1. Sends commits from your local branch to the remote repository.

**Pulling Changes**bash  
Copy code  
git pull origin <branch-name>

1. Updates your local branch with changes from the remote repository.

**Undoing Changes**bash  
Copy code  
git reset

git checkout -- <file>

1. Removes changes from staging or discards local modifications.

These commands will help you with the essential tasks in Git and GitHub, from creating repositories to managing branches and collaborating with others.

—------------------------------------------------------------------

**Java Basics**

a. class helloworld

{

    public void  **printdetails**()

    {

        int a=1;

        a=a+1;

        System.out.print(a+"\n");

    }

    public static void **main**(String *args*[])

    {

        helloworld obj1=new helloworld();

        obj1.printdetails();

        int a=10;

        int b=20;

        int c=a+b;

        System.out.println("the sum of a+b is"+c);

        byte d=127;

        short e=128;

        double f=12.34;

        float g=12.34f;

        char h='A';

        System.out.println(g+" "+d+" "+e+" "+f+" "+h);

        System.out.println("Helloworld");

    }

}

b. public class empdetails {

    String name;

    int age;

    String city;

    public void **printdetails**(){

        System.out.println("Name:"+name);

        System.out.println("age:"+age);

        System.out.println("city:"+city);

    }

    public void  **setdetails**(String *name*,int *age*,String *city*)

    {

        this.name=*name*;

        this.age=*age*;

        this.city=*city*;

    }

    empdetails()

    {

        name="Bhanu";

        age=22;

        city="huzurabad";

*//printdetails();*

    }

    public static void **main**(String *args*[])

    {

        empdetails emp=new empdetails();

       emp.printdetails();

       emp.setdetails("S",23,"kl");

       emp.printdetails();

    }

}

c. public class staticdemo{

    public static int a=0;

    staticdemo()

    {

        a=a+1;

    }

    public void **display**()

    {

        System.out.println("number of objects created"+a);

    }

    public static void **main**(String *args*[])

    {

        staticdemo obj1=new staticdemo();

        obj1.display();

        staticdemo obj2=new staticdemo();

        obj2.display();

    }

}

public class WideningTypeCastingDemo {

    public static void **main**(String *args*[])

    {

        byte b=65;

        short s=b;

        char c=(char)s;

        System.out.println("byte::"+b+" short::"+s+" char::"+c);

        int charval=c;

        System.out.println(charval);

        long l=charval;

        System.out.println("long"+l);

    }

}

d. import java.util.Scanner;

public class Input {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter your name: ");

String name = sc.nextLine();

System.out.println("Enter 1st number: ");

double num1 = sc.nextDouble();

System.out.println("Enter 2nd number: ");

double num2 = sc.nextDouble();

System.out.println("Enter operation: ");

char operation = sc.next().charAt(0);

double result = 0;

switch (operation) {

case '+':

result = num1 + num2;

break;

case '-':

result = num1 - num2;

break;

case '\*':

result = num1 \* num2;

break;

}

System.out.println("Result: " + result);

    }

}

e. public class Operators {

public static void main(String[] args) {

boolean t = true;

boolean f = false;

System.out.println("1. Basic Logical Operations:");

System.out.println(" AND: true && true = " + (t && t));

System.out.println(" AND: true && false = " + (t && f));

System.out.println(" AND: false && true = " + (f && t));

System.out.println(" AND: false && false = " + (f && f));

System.out.println(" OR: true || true = " + (t || t));

System.out.println(" OR: true || false = " + (t || f));

System.out.println(" OR: false || true = " + (f || t));

System.out.println(" OR: false || false = " + (f || f));

System.out.println(" NOT: !true = " + (!t));

System.out.println(" NOT: !false = " + (!f));

System.out.println("\n2. Short-circuit Evaluation:");

System.out.println(" false && (1/0 > 0) = " + (f && (1/0 > 0)));

System.out.println(" true || (1/0 > 0) = " + (t || (1/0 > 0)));

System.out.println("\n3. Operator Precedence:");

System.out.println(" true || false && false = " + (t || f && f));

System.out.println(" (true || false) && false = " + ((t || f) && f));

System.out.println("\n4. Combining with Comparison Operators:");

int x = 5, y = 10;

System.out.println(" (x < y) && (y > 0) = " + ((x < y) && (y > 0)));

System.out.println(" (x > y) || (y < 20) = " + ((x > y) || (y < 20)));

System.out.println("\n5. Complex Conditions:");

boolean a = true, b = false, c = true;

System.out.println(" (a && b) || (a && c) = " + ((a && b) || (a && c)));

System.out.println(" a && (b || c) = " + (a && (b || c)));

System.out.println(" !a || (b && !c) = " + (!a || (b && !c)));

System.out.println("\n6. Bitwise vs. Logical Operators:");

System.out.println(" true & false = " + (t & f)); // Bitwise AND

System.out.println(" true | false = " + (t | f)); // Bitwise OR

System.out.println(" true ^ false = " + (t ^ f)); // Bitwise XOR

System.out.println("\n7. Short-circuit vs. Non-short-circuit:");

int i = 0;

boolean result1 = (f && (++i > 0)); // i is not incremented

boolean result2 = (f & (++i > 0)); // i is incremented

System.out.println(" Short-circuit AND result: " + result1 + ", i = " + i);

System.out.println(" Non-short-circuit AND result: " + result2 + ", i = " + i);

System.out.println("\n8. Logical Operators with Non-boolean Operands:");

System.out.println(" (1 < 2) && (3 < 4) = " + ((1 < 2) && (3 < 4)));

System.out.println(" ('a' < 'b') || ('c' > 'd') = " + (('a' < 'b') || ('c' > 'd')));

System.out.println("\n9. Logical Operators in Control Structures:");

if (t && !f) {

System.out.println(" This will be printed.");

}

}

}

**Banking Application**

import java.util.Scanner;

public class bankingapplication {

    public static void **main**(String *args*[])

    {

        Scanner sc = new Scanner(System.in);

        int customerids[]=new int[4];

        String username[]=new String[4];

        long Balance[]=new long[4];  *//declaring arrays*

        customerids[0]=100;        *//Entering data into the arrays*

        username[0]="Bhanu";

        Balance[0]=5000;

        customerids[1]=101;

        username[1]="Ravi";

        Balance[1]=5000;

        customerids[2]=102;

        username[2]="Raju";

        Balance[2]=5000;

        customerids[3]=103;

        username[3]="sai Koushik";

        Balance[3]=5000;

        int money=0;

                System.out.println("Enter your customerid");

                int a=sc.nextInt();

                sc.nextLine();

                System.out.println("Enter your name"); *//taking inputs such as customerid and name*

                String usern=sc.nextLine();

                int i=0;

                for(i=0;i<username.length;i++)*//if we find customer id and name matches then we will be procedding*

                {

                if(username[i].equals(usern) && customerids[i]==a)

                {

                    System.out.println("Ok Authentication done correct user id and password ");

                    break;

                }

                }

                if(i==customerids.length){

                    System.out.println("not correct");

                }

                while(true)

                {

                    System.out.println("--enter number which operation you what to perform--");*//asking user which task should we perform*

                    System.out.println("--1.add money 2.withdraw 3.balance check 4.exit--");

                    int num=sc.nextInt();

                    if(num==4)

                    {

                        break;

                    }

                    if(num!=3)*//for balence enquire user no need to enter this*

                    {

                        System.out.println("-----how much money do you want to add or withdraw");

                         money=sc.nextInt();

                    }

                    if(num==1)

                    {

                        Balance[i]+=money;

                        System.out.println("Added Sussefully");

                    }

                    else if(num==2)

                    {

                        if(Balance[i]-money>=0)*//we will only give him money if money is present in his account*

                            Balance[i]-=money;

                        else

                        System.out.println("insuffient funds");

                    }

                    else

                        System.out.println("your current Balence is"+Balance[i]);

                }

                System.out.print("Thank you");

    }

}

—-----------------------------------------

ATM Simulation using do while

import java.util.Scanner;

public class ATMSimulator {

static final int PIN = 1234;

static final int MAX\_ATTEMPTS = 3;

static double balance = 2000.00;

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Welcome to the ATM Simulator");

int attempts = 0;

boolean isVerified = false;

while (!isVerified && attempts < MAX\_ATTEMPTS) {

System.out.print("Enter your PIN: ");

if (scanner.nextInt() == PIN) {

isVerified = true;

} else {

attempts++;

System.out.println("Incorrect PIN. Attempts left: " + (MAX\_ATTEMPTS - attempts));

}

}

if (!isVerified) {

System.out.println("Too many incorrect attempts. Your card is blocked.");

return;

}

int choice;

do {

System.out.println("\nMenu:\n1. Check Balance\n2. Exit");

System.out.print("Choose an option: ");

choice = scanner.nextInt();

if (choice == 1) {

System.out.printf("Current balance: $%.2f\n", balance);

} else if (choice != 2) {

System.out.println("Invalid choice. Try again.");

}

} while (choice != 2);

System.out.println("Thank you for using the ATM. Goodbye!");

scanner.close();

}

}

Controlstatments

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

class controlstatments

{

    public static void **main**(String[] *args*) {

        int a = 10;

        int b = 20;

        if (a < b) {

            System.out.println("a is less than b");

        } else {

            System.out.println("a is greater than or equal to b");

        }

        int day = 3;

        switch (day) {

            case 1:

                System.out.println("Monday");

                break;

            case 2:

                System.out.println("Tuesday");

                break;

            case 3:

                System.out.println("Wednesday");

                break;

            default:

                System.out.println("Invalid day");

        }

        for (int i = 0; i < 5; i++) {

            System.out.println("Iteration: " + i);

        }

        int count = 0;

        while (count < 5) {

            System.out.println("Count: " + count);

            count++;

        }

        for(int i=1;i<=10;i++)

        {

            if(i%2==0)

            {

                continue;

            }

            else

            {

                System.out.println(i+"i is the odd number");

            }

        }

        int number = 15;

        if (number > 0) {

            System.out.println(number + " is positive");

            if (number % 2 == 0) {

                System.out.println(number + " is even");

            } else {

                System.out.println(number + " is odd");

            }

        } else if (number < 0) {

            System.out.println(number + " is negative");

        } else {

            System.out.println("The number is zero");

        }

        int n=2;

        do

        {

            System.out.println("Bhanu");

        }while(n<1);

        int arr[]={1,2,3,4,5,6};

        try {

            int[] numbers = {1, 2, 3};

            System.out.println(numbers[3]);

        } catch (Exception *e*) {

            System.out.println("Error: " + e.getMessage());

        }

        for(int i:arr)

        {

            System.out.println(i);

        }

        try{

            BufferedReader br = new BufferedReader(new FileReader("t.txt"));

            String line;

            while ((line = br.readLine()) != null) {

                System.out.println(line);

            }

        } catch (Exception *e*) {

            System.out.println("Error reading file: " + e.getMessage());

        }

    }

}

Oops

class Employee

{

float salary=34\*12;

}

class Executive extends Employee

{

    float bonus=3000\*6;

}

abstract class atm

{

     int balance=0;

    void **checkbalence**()

    {

     System.out.println("balance::"+balance);

    }

    abstract void **addmoney**();

    abstract void **withdrawmoney**();

}

class atmoperations extends atm

{

    void **addmoney**()

    {

        balance=balance+100;

        System.out.println("money added 100 rs"+balance);

    }

    void **withdrawmoney**()

    {

        balance=balance-100;

        System.out.println("money debited 100 rs"+balance);

    }

}

*//overloading*

class operations

{

    int sum=0;

    public void  **add**(int *a*,int *b*)

    {

        sum=*a*+*b*;

        System.out.println(*a*+*b*);

    }

    public void **add**(int *a*,int *b*,int *c*)

    {

        sum=*a*+*b*+*c*;

        System.out.println(*a*+*b*+*c*);

    }

}

class dad

{

    void **property**()

    {

        System.out.println("1000000 accers of land");

    }

}

class son extends dad{

    @Override

    void **property**()

    {

        System.out.println();

    }

    System.out.println("");

    void **wife**()

    {

        System.out.println("xyz ");

    }

}

public class oops

{

public static void **main**(String *args*[])

{

Executive obj=new Executive();

System.out.println("Total salary credited: "+obj.salary);

System.out.println("Bonus of six months: "+obj.bonus);

atmoperations obj1=new atmoperations();

obj1.checkbalence();

obj1.addmoney();

obj1.withdrawmoney();

obj1.checkbalence();

operations obj2=new operations();

obj2.add(1,2);

obj2.add(1,2,3);

}

}

Programe on accessmodifiers

public class AccessModifiersExample {

// Public variable accessible from anywhere

public int publicVar = 10;

// Private variable accessible only within this class

private int privateVar = 20;

// Protected variable accessible within the same package and subclasses

protected int protectedVar = 30;

// Default (package-private) variable accessible within the same package

int defaultVar = 40;

// Public method accessible from anywhere

public void publicMethod() {

System.out.println("Public method called.");

}

// Private method accessible only within this class

private void privateMethod() {

System.out.println("Private method called.");

}

// Protected method accessible within the same package and subclasses

protected void protectedMethod() {

System.out.println("Protected method called.");

}

// Default (package-private) method accessible within the same package

void defaultMethod() {

System.out.println("Default method called.");

}

public static void main(String[] args) {

AccessModifiersExample example = new AccessModifiersExample();

System.out.println("Public variable: " + example.publicVar);

System.out.println("Protected variable: " + example.protectedVar);

System.out.println("Default variable: " + example.defaultVar);

example.publicMethod();

example.protectedMethod();

example.defaultMethod();

}

}

**Project On Billing System**

import java.util.\*;

abstract class ecommers

{

int money=0;

abstract void fooditems(int quantityrice,int quantitydal);

abstract void toys(int numberoftoys);

abstract void Perfume(int numberofperfumes);

}

abstract class food extends ecommers

{

int rice=100,dal=120;

void fooditems(int quantityrice,int quantitydal)

{

//rice,dal;

money+=(rice\*quantityrice)+(dal\*quantitydal);

System.out.println("cost for fooditems is::"+money);

}

}

abstract class toys extends food

{

int toys=10;

void toys(int numberoftoys)

{

System.out.println("cost for only toys is"+(toys\*numberoftoys));

money+=toys\*numberoftoys;

System.out.println("total cost is::"+money);

}

}

class Perfume extends toys

{

int sweethearperfume=250;

void Perfume(int numberofperfumes)

{

System.out.println("cost for only purfumes is"+(numberofperfumes\*sweethearperfume));

money+=numberofperfumes\*sweethearperfume;

System.out.println("total cost::"+money);

}

}

class passwordcheck

{

public boolean authentication(String u,int p)

{

String username="admin";

int password=1234;

if(u.equals(username) && p==password)

{

return true;

}

return false;

}

}

public class GFG {

public static void main (String[] args) {

Scanner scan=new Scanner(System.in);

System.out.println("enter your user name and password for the verification");

String u=scan.nextLine();

int p=scan.nextInt();

passwordcheck obj=new passwordcheck();

if(obj.authentication(u,p))

{

System.out.println("authentication done");

Perfume obj1=new Perfume();

System.out.println("how much quantity of rice and dal u want::");

int rice=scan.nextInt();

int dal=scan.nextInt();

obj1.fooditems(rice,dal);

System.out.println("how many toys and perfumes you need::");

int numberoftoys=scan.nextInt();

obj1.toys(numberoftoys);

int numberofperfumes=scan.nextInt();

obj1.Perfume(numberofperfumes);

}

else{

System.out.println("Authentication failed");

}

}

}

—---------------------------------------------

An **enum** in Java is a special data type that allows you to define a set of named constants. These constants represent a fixed set of values, making your code more readable, maintainable, and robust. Think of enums as a way to create your own custom data types with a predefined set of possible values.

Enum.java

import java.util.Scanner;

public class ShoppingCart {

// Enum definition for products

public enum Product {

APPLE("Apple", 1.0),

BANANA("Banana", 0.8),

ORANGE("Orange", 1.2),

CHOCOLATE("Chocolate", 2.5);

private final String name;

private final double price;

// Constructor for the enum

Product(String name, double price) {

this.name = name;

this.price = price;

}

// Getter methods

public String getName() {

return name;

}

public double getPrice() {

return price;

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

double totalCost = 0.0;

System.out.println("Welcome to the Shopping Cart!");

System.out.println("Available products:");

// Display all available products

for (Product product : Product.values()) {

System.out.printf("%s - $%.2f%n", product.getName(), product.getPrice());

}

// Simulate adding items to the cart

while (true) {

System.out.print("Enter product name (or 'done' to finish): ");

String userInput = scanner.nextLine().toUpperCase();

if (userInput.equals("DONE")) {

break;

}

try {

Product selectedProduct = Product.valueOf(userInput);

System.out.printf("Added %s to your cart. Price: $%.2f%n",

selectedProduct.getName(), selectedProduct.getPrice());

totalCost += selectedProduct.getPrice();

} catch (IllegalArgumentException e) {

System.out.println("Invalid product name. Please try again.");

}

}

System.out.printf("Total cost: $%.2f%n", totalCost);

scanner.close();

}

}

Output:

Welcome to the Shopping Cart!

Available products:

Apple - $1.00

Banana - $0.80

Orange - $1.20

Chocolate - $2.50

Enter product name (or 'done' to finish): orange

Added Orange to your cart. Price: $1.20

Enter product name (or 'done' to finish): done

Total cost: $1.20

=== Code Execution Successful ===

Collections.java

import java.util.\*;

*// This class demonstrates the usage of various collection types in Java*

public class collections {

    public static void main(String *args*[]) {

*// Creating an ArrayList to store Integer values*

        List<Integer> al = new ArrayList<Integer>();

        al.add(2); *// Adding elements to the ArrayList*

        al.add(1);

        Collections.sort(al); *// Sorting the ArrayList*

        System.out.println("ArrayList: " + al); *// Displaying the sorted ArrayList*

        al.remove(1); *// Removing the element at index 1*

        System.out.println("ArrayList after removal: " + al); *// Displaying the ArrayList after removal*

*// Creating a Vector to store Integer values*

        Vector<Integer> v = new Vector<Integer>();

        v.add(1); *// Adding elements to the Vector*

        v.add(2);

        System.out.println("Vector: " + v); *// Displaying the Vector*

*// Creating a LinkedList to store String values*

        List<String> ll = new LinkedList<String>();

        ll.add("apple"); *// Adding an element to the LinkedList*

        System.out.println("LinkedList: " + ll); *// Displaying the LinkedList*

*// Creating a Stack to store String values*

        Stack<String> stack = new Stack<>();

        stack.push("apple"); *// Pushing elements onto the Stack*

        stack.push("banana");

        stack.push("grape");

        System.out.println("Stack elements: " + stack); *// Displaying the Stack elements*

        String str = stack.pop(); *// Popping the top element from the Stack*

        System.out.println("Popped element: " + str); *// Displaying the popped element*

        System.out.println("----------- Set Interface ----------");

*// Creating a HashSet to store String values*

        Set<String> hs = new HashSet<>();

        System.out.println("Initial size of HashSet: " + hs.size()); *// Displaying initial size*

        System.out.println("Set cannot store duplicate elements:");

*// Adding elements to the HashSet*

        hs.add("bhanu");

        hs.add("a");

        hs.add("b");

        hs.add("c");

        System.out.println("HashSet: " + hs + "\nSize of HashSet: " + hs.size()); *// Displaying HashSet and its size*

*// Creating a LinkedHashSet to maintain insertion order*

        LinkedHashSet<String> lhs = new LinkedHashSet<String>();

        System.out.println("This is from LinkedHashSet; it maintains insertion order:\n");

*// Adding elements to the LinkedHashSet*

        lhs.add("bhanu");

        lhs.add("a");

        lhs.add("b");

        lhs.add("c");

        System.out.println(lhs); *// Displaying the LinkedHashSet*

*// Creating a HashMap to store capital cities*

        HashMap<String, String> capitalCities = new HashMap<String, String>();

        capitalCities.put("Telangana", "hyd"); *// Adding key-value pairs to the HashMap*

        capitalCities.put("Maharashtra", "pun");

        capitalCities.put("Karnataka", "ban");

        System.out.println(capitalCities); *// Displaying the HashMap*

        System.out.println("It maintains insertion order");

*// Creating a LinkedHashMap to maintain insertion order*

        LinkedHashMap<String, String> capitalCitie = new LinkedHashMap<String, String>();

        capitalCitie.put("Telangana", "hyd");

        capitalCitie.put("Maharashtra", "pun");

        capitalCitie.put("Karnataka", "ban");

        System.out.println(capitalCitie); *// Displaying the LinkedHashMap*

*// Creating a Queue using LinkedList*

        Queue<String> queue = new LinkedList<>();

        queue.offer("First"); *// Adding elements to the Queue*

        queue.offer("Second");

        queue.offer("Third");

        System.out.println("Queue: " + queue); *// Displaying the Queue*

*// Peeking the first element without removing it*

        System.out.println("Retrieve 1st element but do not remove from Queue: " + queue.peek());

        System.out.println("Queue after peek: " + queue); *// Displaying the Queue after peek*

*// Creating a Deque using ArrayDeque*

        Deque<String> deque = new ArrayDeque<>();

        deque.addFirst("Front"); *// Adding elements to the Deque*

        deque.addLast("Back");

        System.out.println("Deque: " + deque); *// Displaying the Deque*

*// Polling elements from the Deque*

        System.out.println("Poll first from Deque: " + deque.pollFirst());

        System.out.println("Poll last from Deque: " + deque.pollLast());

*// Creating a PriorityQueue*

        PriorityQueue<Integer> pQueue = new PriorityQueue<Integer>();

        pQueue.add(10); *// Adding elements to the PriorityQueue*

        pQueue.add(20);

        pQueue.add(15);

*// Displaying the top element of the PriorityQueue*

        System.out.println("Top element in PriorityQueue: " + pQueue.peek());

        System.out.println("Polled element from PriorityQueue: " + pQueue.poll()); *// Polling the top element*

        System.out.println("Top element after polling: " + pQueue.peek()); *// Displaying the top element after polling*

    }

}

Output::

ArrayList: [1, 2]

ArrayList after removal: [1]

Vector: [1, 2]

LinkedList: [apple]

Stack elements: [apple, banana, grape]

Popped element: grape

----------- Set Interface ----------

Initial size of HashSet: 0

Set cannot store duplicate elements:

HashSet: [a, b, c, bhanu]

Size of HashSet: 4

This is from LinkedHashSet; it maintains insertion order:

[bhanu, a, b, c]

{Telangana=hyd, Karnataka=ban, Maharashtra=pun}

It maintains insertion order

{Telangana=hyd, Maharashtra=pun, Karnataka=ban}

Queue: [First, Second, Third]

Retrieve 1st element but do not remove from Queue: First

Queue after peek: [First, Second, Third]

Deque: [Front, Back]

Poll first from Deque: Front

Poll last from Deque: Back

Top element in PriorityQueue: 10

Polled element from PriorityQueue: 10

Top element after polling: 15

Multithreading

Multithreading allows you to execute multiple threads concurrently within a single process. Each thread represents a lightweight sub-process, and together they share the same memory space. Here are some key concepts related to multithreading:

1. **Threads and Multitasking:**

* A thread is the smallest unit of processing. It runs independently and can perform tasks simultaneously with other threads.
* Multitasking involves executing multiple tasks concurrently. In Java, we achieve multitasking through multithreading.

class mythread extends Thread

{

    public void **run**()

    {

        for(int i=0;i<10;i++)

        {

            System.out.println(Thread.currentThread().getName()+"Bhanu Prakash"+i);

        }

    }

}

class Threadclass

{

    public static void **main**(String[] *args*) {

        System.out.println("1st line");

        mythread obj1=new mythread();

        obj1.start();

        for(int i=0;i<10;i++)

        {

            System.out.println(Thread.currentThread().getName()+"iam from main method"+i);

        }

    }

}

Output::

1st line

mainiam from main method0

mainiam from main method1

mainiam from main method2

Thread-0Bhanu Prakash0

Thread-0Bhanu Prakash1

Thread-0Bhanu Prakash2

Thread-0Bhanu Prakash3

Thread-0Bhanu Prakash4

Thread-0Bhanu Prakash5

Thread-0Bhanu Prakash6

Thread-0Bhanu Prakash7

mainiam from main method3

Thread-0Bhanu Prakash8

Thread-0Bhanu Prakash9

mainiam from main method4

mainiam from main method5

mainiam from main method6

mainiam from main method7

mainiam from main method8

mainiam from main method9

class MyThread extends Thread {

@Override

public void run() {

try {

// Pause for 500 milliseconds (0.5 seconds)

Thread.sleep(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

// Code to be executed in this thread

for (int i = 1; i <= 5; i++) {

System.out.println("Thread using extends: " + i);

}

}

}

class MyRunnable implements Runnable {

@Override

public void run() {

try {

// Pause for 500 milliseconds (0.5 seconds)

Thread.sleep(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

// Code to be executed in this thread

for (int i = 1; i <= 5; i++) {

System.out.println("Thread using Runnable: " + i);

}

}

}

public class Main {

public static void main(String[] args) {

// Using extends

MyThread thread1 = new MyThread();

thread1.start(); // Start the thread

// Using lambda expression

Thread thread3 = new Thread(() -> {

try {

// Pause for 500 milliseconds (0.5 seconds)

Thread.sleep(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

for (int i = 1; i <= 5; i++) {

System.out.println("Thread using lambda: " + i);

}

});

thread3.start(); // Start the thread

// Using Runnable

MyRunnable myRunnable = new MyRunnable();

Thread thread2 = new Thread(myRunnable);

thread2.start(); // Start the thread

// Main thread continues execution

for (int i = 1; i <= 5; i++) {

System.out.println("Main thread: " + i);

}

}

}

—-------------------------------------------------------------------

**Simplecache.java**

import java.util.HashMap;

import java.util.Map;

public class SimpleCache<K,V> {

private final Map<K,V> cache;

public SimpleCache(){

this.cache = new HashMap<>();

}

public void put(K key, V value){

cache.put(key, value);

}

public V get(K key){

return cache.get(key);

}

public void remove(K key){

cache.remove(key);

}

public void clear(){

cache.clear();

}

public int size(){

return cache.size();

}

public static void main(String[] args) {

SimpleCache<String, String> cache = new SimpleCache<>();

cache.put("key1", "value1");

cache.put("key2", "value2");

cache.put("key3", "value3");

System.out.println(cache.get("key1"));

cache.remove("key2");

System.out.println(cache.size());

cache.clear();

System.out.println(cache.size());

}

}

Output::

value1

2

0

**Multithreading.java**

class BankAccount {

    private int balance = 0;

    public synchronized void **deposit**(int *amount*) {

        balance += *amount*;

        System.out.println("Deposited: " + *amount* + " New Balance: " + balance);

    }

    public synchronized void **withdraw**(int *amount*) {

        if (balance >= *amount*) {

            balance -= *amount*;

            System.out.println("Withdrawn: " + *amount* + " New Balance: " + balance);

        } else {

            System.out.println("Insufficient balance");

        }

    }

    public int **getBalance**() {

        return balance;

    }

}

*// Thread class for depositing money*

class DepositThread extends Thread {

    private BankAccount account;

    public DepositThread(BankAccount *account*) {

        this.account = *account*;

    }

    @Override

    public void **run**() {

        for (int i = 0; i < 5; i++) {

            account.deposit(100);

        }

    }

}

*// Thread class for withdrawing money*

class WithdrawThread extends Thread {

    private BankAccount account;

    public WithdrawThread(BankAccount *account*) {

        this.account = *account*;

    }

    @Override

    public void **run**() {

        for (int i = 0; i < 5; i++) {

            account.withdraw(80);

            try {

                Thread.sleep(100);

            } catch (InterruptedException *e*) {

                e.printStackTrace();

            }

        }

    }

}

public class BankDemo {

    public static void **main**(String[] *args*) throws InterruptedException {

        BankAccount account = new BankAccount();

*// Create instances of DepositThread and WithdrawThread*

        DepositThread depositThread = new DepositThread(account);

        WithdrawThread withdrawThread = new WithdrawThread(account);

*// Start the threads*

        depositThread.start();

        withdrawThread.start();

*// Wait for both threads to finish*

        depositThread.join();

        withdrawThread.join();

*// Print the final balance*

        System.out.println("Final Balance: " + account.getBalance());

    }

}

Output:   
Deposited: 100 New Balance: 100

Deposited: 100 New Balance: 200

Withdrawn: 80 New Balance: 120

Deposited: 100 New Balance: 220

Deposited: 100 New Balance: 320

Deposited: 100 New Balance: 420

Withdrawn: 80 New Balance: 340

Withdrawn: 80 New Balance: 260

Withdrawn: 80 New Balance: 180

Withdrawn: 80 New Balance: 100

Final Balance: 100

**DatabaseSimulator.java**

import java.util.HashMap;

import java.util.Map;

import java.util.Random;

class Product {

    private String id;

    private String name;

    private double price;

    public Product(String *id*, String *name*, double *price*) {

        this.id = *id*;

        this.name = *name*;

        this.price = *price*;

    }

*// Getters and setters omitted for brevity*

    @Override

    public String **toString**() {

        return "Product{id='" + id + "', name='" + name + "', price=" + price + "}";

    }

}

class db {

    private Map<String, Product> products = new HashMap<>();

    private Random random = new Random();

    public db(int *numProducts*) {

        for (int i = 0; i < *numProducts*; i++) {

            String id = "PROD" + i;

            products.put(id, new Product(id, "Product " + i, 10 + random.nextDouble() \* 90));

        }

    }

    public Product **getProduct**(String *id*) {

*// Simulate database access delay*

        try {

            Thread.sleep(100); *// 100ms delay to simulate DB access*

        } catch (InterruptedException *e*) {

            e.printStackTrace();

        }

        return products.get(*id*);

    }

}

public class Main {

    public static void **main**(String[] *args*) {

*// Create a DatabaseSimulator with 10 products*

        db db = new db(10);

*// Simulate multiple threads accessing the database*

        for (int i = 0; i < 5; i++) {

            new Thread(() -> {

                String productId = "PROD" + random.nextInt(10);

                Product product = db.getProduct(productId);

                System.out.println(Thread.currentThread().getName() + " retrieved: " + product);

            }).start();

        }

    }

    private static final Random random = new Random();

}

Output:

Thread-4 retrieved: Product{id='PROD5', name='Product 5', price=52.05852138101775}

Thread-2 retrieved: Product{id='PROD2', name='Product 2', price=40.585647657174036}

Thread-3 retrieved: Product{id='PROD5', name='Product 5', price=52.05852138101775}

Thread-1 retrieved: Product{id='PROD1', name='Product 1', price=33.67443221656666}

Thread-0 retrieved: Product{id='PROD3', name='Product 3', price=20.91196024115242}

**ProductCache.java**

import com.google.common.cache.CacheBuilder;

import com.google.common.cache.CacheLoader;

import com.google.common.cache.LoadingCache;

import java.util.HashMap;

import java.util.Map;

import java.util.Random;

import java.util.concurrent.TimeUnit;

class Product {

private String id;

private String name;

private double price;

public Product(String id, String name, double price) {

this.id = id;

this.name = name;

this.price = price;

}

@Override

public String toString() {

return "Product{id='" + id + "', name='" + name + "', price=" + price + "}";

}

}

class DatabaseSimulator {

private Map<String, Product> products = new HashMap<>();

private Random random = new Random();

public DatabaseSimulator(int numProducts) {

for (int i = 0; i < numProducts; i++) {

String id = "PROD" + i;

products.put(id, new Product(id, "Product " + i, 10 + random.nextDouble() \* 90));

}

}

public Product getProduct(String id) {

try {

Thread.sleep(100); // Simulate database access delay

} catch (InterruptedException e) {

e.printStackTrace();

}

return products.get(id);

}

}

class ProductService {

private final DatabaseSimulator database;

private final LoadingCache<String, Product> cache;

public ProductService(int numProducts, int cacheSize) {

this.database = new DatabaseSimulator(numProducts);

this.cache = CacheBuilder.newBuilder()

.maximumSize(cacheSize)

.expireAfterWrite(10, TimeUnit.MINUTES)

.recordStats()

.build(new CacheLoader<String, Product>() {

@Override

public Product load(String id) {

return database.getProduct(id);

}

});

}

public Product getProduct(String id) throws Exception {

// Check if product is in cache

if (cache.asMap().containsKey(id)) {

return cache.get(id); // Cache hit

} else {

// Cache miss: fetch from database and update cache

Product product = database.getProduct(id);

if (product != null) {

cache.put(id, product); // Update cache with the retrieved product

}

return product; // Return the product

}

}

public void printCacheStats() {

System.out.println("Cache stats: " + cache.stats());

}

}

public class CachingPerformanceTest {

public static void main(String[] args) throws Exception {

testPerformance(10\_000, 1\_000); // 10,000 products, 1,000 cache size

testPerformance(1\_000\_000, 100\_000); // 1 million products, 100,000 cache size

}

private static void testPerformance(int numProducts, int cacheSize) throws Exception {

ProductService service = new ProductService(numProducts, cacheSize);

Random random = new Random();

System.out.println("\nTesting with " + numProducts + " products and cache size " + cacheSize);

// Warm up the cache

for (int i = 0; i < 1000; i++) {

service.getProduct("PROD" + random.nextInt(numProducts));

}

// Test performance

long startTime = System.currentTimeMillis();

for (int i = 0; i < 10\_000; i++) {

service.getProduct("PROD" + random.nextInt(numProducts));

}

long endTime = System.currentTimeMillis();

System.out.println("Time taken for 10,000 random product retrievals: " + (endTime - startTime) + "ms");

service.printCacheStats();

}

}

—---------------------------------------------------------------------

**Simplecache.java**

import java.util.HashMap;

import java.util.Map;

public class SimpleCache<K,V> {

private final Map<K,V> cache;

public SimpleCache(){

this.cache = new HashMap<>();

}

public void put(K key, V value){

cache.put(key, value);

}

public V get(K key){

return cache.get(key);

}

public void remove(K key){

cache.remove(key);

}

public void clear(){

cache.clear();

}

public int size(){

return cache.size();

}

public static void main(String[] args) {

SimpleCache<String, String> cache = new SimpleCache<>();

cache.put("key1", "value1");

cache.put("key2", "value2");

cache.put("key3", "value3");

System.out.println(cache.get("key1"));

cache.remove("key2");

System.out.println(cache.size());

cache.clear();

System.out.println(cache.size());

}

}

Output::

value1

2

0

**Multithreading.java**

class BankAccount {

    private int balance = 0;

    public synchronized void **deposit**(int *amount*) {

        balance += *amount*;

        System.out.println("Deposited: " + *amount* + " New Balance: " + balance);

    }

    public synchronized void **withdraw**(int *amount*) {

        if (balance >= *amount*) {

            balance -= *amount*;

            System.out.println("Withdrawn: " + *amount* + " New Balance: " + balance);

        } else {

            System.out.println("Insufficient balance");

        }

    }

    public int **getBalance**() {

        return balance;

    }

}

*// Thread class for depositing money*

class DepositThread extends Thread {

    private BankAccount account;

    public DepositThread(BankAccount *account*) {

        this.account = *account*;

    }

    @Override

    public void **run**() {

        for (int i = 0; i < 5; i++) {

            account.deposit(100);

        }

    }

}

*// Thread class for withdrawing money*

class WithdrawThread extends Thread {

    private BankAccount account;

    public WithdrawThread(BankAccount *account*) {

        this.account = *account*;

    }

    @Override

    public void **run**() {

        for (int i = 0; i < 5; i++) {

            account.withdraw(80);

            try {

                Thread.sleep(100);

            } catch (InterruptedException *e*) {

                e.printStackTrace();

            }

        }

    }

}

public class BankDemo {

    public static void **main**(String[] *args*) throws InterruptedException {

        BankAccount account = new BankAccount();

*// Create instances of DepositThread and WithdrawThread*

        DepositThread depositThread = new DepositThread(account);

        WithdrawThread withdrawThread = new WithdrawThread(account);

*// Start the threads*

        depositThread.start();

        withdrawThread.start();

*// Wait for both threads to finish*

        depositThread.join();

        withdrawThread.join();

*// Print the final balance*

        System.out.println("Final Balance: " + account.getBalance());

    }

}

Output:   
Deposited: 100 New Balance: 100

Deposited: 100 New Balance: 200

Withdrawn: 80 New Balance: 120

Deposited: 100 New Balance: 220

Deposited: 100 New Balance: 320

Deposited: 100 New Balance: 420

Withdrawn: 80 New Balance: 340

Withdrawn: 80 New Balance: 260

Withdrawn: 80 New Balance: 180

Withdrawn: 80 New Balance: 100

Final Balance: 100

**DatabaseSimulator.java**

import java.util.HashMap;

import java.util.Map;

import java.util.Random;

class Product {

    private String id;

    private String name;

    private double price;

    public Product(String *id*, String *name*, double *price*) {

        this.id = *id*;

        this.name = *name*;

        this.price = *price*;

    }

*// Getters and setters omitted for brevity*

    @Override

    public String **toString**() {

        return "Product{id='" + id + "', name='" + name + "', price=" + price + "}";

    }

}

class db {

    private Map<String, Product> products = new HashMap<>();

    private Random random = new Random();

    public db(int *numProducts*) {

        for (int i = 0; i < *numProducts*; i++) {

            String id = "PROD" + i;

            products.put(id, new Product(id, "Product " + i, 10 + random.nextDouble() \* 90));

        }

    }

    public Product **getProduct**(String *id*) {

*// Simulate database access delay*

        try {

            Thread.sleep(100); *// 100ms delay to simulate DB access*

        } catch (InterruptedException *e*) {

            e.printStackTrace();

        }

        return products.get(*id*);

    }

}

public class Main {

    public static void **main**(String[] *args*) {

*// Create a DatabaseSimulator with 10 products*

        db db = new db(10);

*// Simulate multiple threads accessing the database*

        for (int i = 0; i < 5; i++) {

            new Thread(() -> {

                String productId = "PROD" + random.nextInt(10);

                Product product = db.getProduct(productId);

                System.out.println(Thread.currentThread().getName() + " retrieved: " + product);

            }).start();

        }

    }

    private static final Random random = new Random();

}

Output:

Thread-4 retrieved: Product{id='PROD5', name='Product 5', price=52.05852138101775}

Thread-2 retrieved: Product{id='PROD2', name='Product 2', price=40.585647657174036}

Thread-3 retrieved: Product{id='PROD5', name='Product 5', price=52.05852138101775}

Thread-1 retrieved: Product{id='PROD1', name='Product 1', price=33.67443221656666}

Thread-0 retrieved: Product{id='PROD3', name='Product 3', price=20.91196024115242}

**ProductCache.java**

import com.google.common.cache.CacheBuilder;

import com.google.common.cache.CacheLoader;

import com.google.common.cache.LoadingCache;

import java.util.HashMap;

import java.util.Map;

import java.util.Random;

import java.util.concurrent.TimeUnit;

class Product {

private String id;

private String name;

private double price;

public Product(String id, String name, double price) {

this.id = id;

this.name = name;

this.price = price;

}

@Override

public String toString() {

return "Product{id='" + id + "', name='" + name + "', price=" + price + "}";

}

}

class DatabaseSimulator {

private Map<String, Product> products = new HashMap<>();

private Random random = new Random();

public DatabaseSimulator(int numProducts) {

for (int i = 0; i < numProducts; i++) {

String id = "PROD" + i;

products.put(id, new Product(id, "Product " + i, 10 + random.nextDouble() \* 90));

}

}

public Product getProduct(String id) {

try {

Thread.sleep(100); // Simulate database access delay

} catch (InterruptedException e) {

e.printStackTrace();

}

return products.get(id);

}

}

class ProductService {

private final DatabaseSimulator database;

private final LoadingCache<String, Product> cache;

public ProductService(int numProducts, int cacheSize) {

this.database = new DatabaseSimulator(numProducts);

this.cache = CacheBuilder.newBuilder()

.maximumSize(cacheSize)

.expireAfterWrite(10, TimeUnit.MINUTES)

.recordStats()

.build(new CacheLoader<String, Product>() {

@Override

public Product load(String id) {

return database.getProduct(id);

}

});

}

public Product getProduct(String id) throws Exception {

// Check if product is in cache

if (cache.asMap().containsKey(id)) {

return cache.get(id); // Cache hit

} else {

// Cache miss: fetch from database and update cache

Product product = database.getProduct(id);

if (product != null) {

cache.put(id, product); // Update cache with the retrieved product

}

return product; // Return the product

}

}

public void printCacheStats() {

System.out.println("Cache stats: " + cache.stats());

}

}

public class CachingPerformanceTest {

public static void main(String[] args) throws Exception {

testPerformance(10\_000, 1\_000); // 10,000 products, 1,000 cache size

testPerformance(1\_000\_000, 100\_000); // 1 million products, 100,000 cache size

}

private static void testPerformance(int numProducts, int cacheSize) throws Exception {

ProductService service = new ProductService(numProducts, cacheSize);

Random random = new Random();

System.out.println("\nTesting with " + numProducts + " products and cache size " + cacheSize);

// Warm up the cache

for (int i = 0; i < 1000; i++) {

service.getProduct("PROD" + random.nextInt(numProducts));

}

// Test performance

long startTime = System.currentTimeMillis();

for (int i = 0; i < 10\_000; i++) {

service.getProduct("PROD" + random.nextInt(numProducts));

}

long endTime = System.currentTimeMillis();

System.out.println("Time taken for 10,000 random product retrievals: " + (endTime - startTime) + "ms");

service.printCacheStats();

}

}

—---------------------------------------------------

**Selectionsort.java**

Selection sort is a simple sorting algorithm that works by repeatedly finding the minimum element in the unsorted part of the array and swapping it with the first element of the unsorted part. This process is repeated until the entire array is sorted.

Time complexity:O(n^2)

class selectionsort

{

    public static void **main**(String *args*[])

    {

        int arr[]={5,4,1,3,2};

        for(int i=0;i<arr.length;i++)

        {

            int min=i;

            for(int j=i+1;j<arr.length;j++)

            {

                if(arr[j]<arr[min])

                {

                    min=j;

                }

            }

            int temp=arr[i];

            arr[i]=arr[min];

            arr[min]=temp;

        }

        for(int k=0;k<arr.length;k++)

        {

            System.out.println(arr[k]);

        }

    }

}

Output::

1

2

3

4

5

**Bubblesort.java**

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. Bubble Sort in Java is not the best method to sort an array but is one of the most basic implementations for one to learn. In this article, we will learn how to write a program for Bubble Sort in Java.

Time-Complexity:O(n^2)

class bubblesort

{

    public static void **main**(String *args*[])

    {

        int arr[]={5,4,1,3,2};

        for(int i=0;i<arr.length;i++)

        {

            for(int j=0;j<arr.length-1-i;j++)

            {

                if(arr[j]>arr[j+1])

                {

                    int temp=arr[j];

                    arr[j]=arr[j+1];

                    arr[j+1]=temp;

                }

            }

        }

        System.out.println("sorted array is:");

        for(int k=0;k<arr.length;k++)

        {

            System.out.println(arr[k]);

        }

    }

}

Output:

sorted array is:

1

2

3

4

5

**Insertionsort.java**

**Insertion sort**is a simple sorting algorithm that works by building a sorted array one element at a time. It is considered an ” **in-place**” sorting algorithm, meaning it doesn’t require any additional memory space beyond the original array.

public class InsertionSort {

void sort(int arr[])

{

int n = arr.length;

for (int i = 1; i < n; ++i) {

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

/\* A utility function to print array of size n \*/

static void printArray(int arr[])

{

int n = arr.length;

for (int i = 0; i < n; ++i)

System.out.print(arr[i] + " ");

System.out.println();

}

// Driver method

public static void main(String args[])

{

int arr[] = { 1,7,5,8,3,6};

InsertionSort ob = new InsertionSort();

ob.sort(arr);

printArray(arr);

}

}

Output:

1 3 5 6 7 8

**Twosum.java**

import java.util.\*;

public class twosum {

    public static void **main**(String[] *args*) {

*//     int arr[]={4,5,3,2};*

*//     int target=6;*

*//     HashMap<Integer,Integer> hs=new HashMap<>();*

*//     for(int i=0;i<arr.length;i++)*

*//     {*

*//         int s1=arr[i];*

*//         if(hs.containsKey(target-s1))*

*//         {*

*//             System.out.println(hs.get(target-s1)+" "+i);*

*//             break;*

*//         }*

*//         hs.put(s1,i);*

*//     }*

*//optimized soluation using two pointer approach*

    int arr[]={1,2,3,4,5,6};

    int left=0;

    int right=arr.length-1;

    int target=6;

    Arrays.sort(arr);

    while(left<right)

    {

        if(arr[left]+arr[right]==target)

        {

            System.out.println(left+" "+right);

            return ;

        }

        else if(arr[left]+arr[right]<target)

        {

            left++;

        }

        else

        {

            right--;

        }

    }

    }

}

Output:

0 4

**Maximum sub array.java**

public class maximumsubarray {

    public static void main(String[] *args*) {

    int arr[]={-2,-3,4,-1,-2,1,5,-3};

    int sum1=0;

    int max=Integer.MIN\_VALUE;

    for(int i=0;i<arr.length;i++)

    {

        sum1=0;

        for(int j=i;j<arr.length;j++)

        {

            sum1=sum1+arr[j];

*//System.out.println("pair::"+i+"-"+j+"::"+sum1);*

            if(sum1>max)

            {

                max=sum1;

            }

        }

    }

    System.out.println("with out optimization::"+max);

*//hear is the optimum solution using kadanes algorithem*

    int sum=0;

    int min=Integer.MIN\_VALUE;

    for(int i=0;i<arr.length;i++)

    {

        sum=sum+arr[i];

        if(sum<0)

        {

            sum=0;

        }

        else{

            min=Math.max(min,sum);

        }

    }

System.out.println("kadanes::"+min);

    }

}

Output::

with out optimization::7

kadanes::7

**containerwithmostwater.java**

import java.util.\*;

class containerwithmostwater

{

public static void main(String args[])

{

int min=Integer.MIN\_VALUE;

int[] arr={ 1, 5, 4, 3,9 };

int m1,ma=0;

for(int i=0;i<arr.length;i++)

{

for(int z=i+1;z<arr.length;z++)

{

m1=Math.min(arr[i],arr[z]);

ma=Math.max(ma,(z-i)\*m1);

}

}

System.out.println(ma);

}

}

**Output::**

**15**

—-------------------------------------------------

**Largest and secondlargest**

*//find largest and second largest numberin the array*

import java.util.\*;

public class largestnumber {

    public static void main(String[] *args*) {

        Scanner sc=new Scanner(System.in);

        int n=sc.nextInt();

        int arr[]=new int[n];

        for(int i=0;i<n;i++){

            arr[i]=sc.nextInt();

        }

        int largest=arr[0];

        int secondlargest=arr[0];

        for(int i=0;i<n;i++){

            if(arr[i]>largest){

                secondlargest=largest;

                largest=arr[i];

            }

*// else if(arr[i]>secondlargest && arr[i]!=largest){*

*//     secondlargest=arr[i];*

*// }*

        }

        System.out.println("largest number is::"+largest);

        System.out.println("second largest number is::"+secondlargest);

    }

}

Output:

5

1 4 6 7 3

largest number is::7

second largest number is::6

**0s,1s and 2s**

*// Online Java Compiler*

*// Use this editor to write, compile and run your Java code online*

import java.util.\*;

class moving0s1s2s {

    public static void **main**(String[] *args*) {

    HashSet<Integer> hs=new HashSet<Integer>();

    int[] arr={1,1,1,2,2,2,3,3,3};

    for(int i=0;i<arr.length;i++)

    {

        hs.add(arr[i]);

    }

*//System.out.println(hs.size());*

    int i=0;

    for(int z:hs)

    {

        arr[i++]=z;

    }

    for(i=0;i<arr.length;i++)

    {

        System.out.println(arr[i]);

    }

*//optimized solution for this is*

    int[] arr1={1,1,2,2,2,3,3,3,3};

    int i1=0;

    for(int j=1;j<arr1.length;j++)

    {

        if(arr1[i1]!=arr1[j])

        {

            i1++;

            arr1[i1]=arr1[j];

        }

    }

    System.out.println("optimized solution");

    for(int j=0;j<arr1.length;j++)

    {

        System.out.println(arr1[j]);

    }

    }

}

**Output::**

1

2

3

2

2

2

3

3

3

optimized solution

1

2

3

2

2

3

3

3

3

**Sql**

create database employeemanagment;

show databases;

use employeemanagment;

create table emp(id int auto\_increment primary key,first\_name varchar(50)

,last\_name varchar(50),email varchar(50),hire\_date date,salary

decimal(10,2));

select \* from emp;

INSERT INTO emp (first\_name,last\_name,email,hire\_date,salary) VALUES ('a', 'raju', 'asdf@gmail.com', '2019-01-01', 1999.99);

INSERT INTO emp (first\_name,last\_name,email,hire\_date,salary) VALUES ('s', 'raju', 'hgsdj@gmail.com', '2019-01-30', 1999.99);

insert into emp(first\_name,last\_name,email,hire\_date,salary) values('v','b','m@gmail.com','2020-10-10',5000.99);

select \* from emp where last\_name like 'r\_j%';

delete from emp where id=2;

**crudesql.java**

package com.mycompany.app;

import java.sql.\*;

public class MySQLCRUDExample {

    private static final String URL = "jdbc:mysql://localhost:3306/employeemanagment";

    private static final String USERNAME = "root";

    private static final String PASSWORD = "root";

    private static Connection connection;

    private static Statement statement;

    private static ResultSet resultSet;

    private static PreparedStatement preparedStatement;

    public static void main(String[] args) {

        try {

             connection = DriverManager.getConnection(URL, USERNAME, PASSWORD);

             createRecord();

             readRecord();

             updateRecord();

             deleteRecord();

        } catch (SQLException e) {

            e.printStackTrace();

        }finally{

            try {

                if (resultSet != null) {

                    resultSet.close();

                }

                if (statement != null) {

                    statement.close();

                }

                if (connection != null) {

                    connection.close();

                }

                if(preparedStatement != null){

                    preparedStatement.close();

                }

            } catch (SQLException e) {

                e.printStackTrace();

            }

        }

    }

    private static void createRecord(){

        String query = "INSERT INTO emp(first\_name, last\_name, email, hire\_date, salary) VALUES (?, ?, ?, ?, ?)";

        try {

            preparedStatement = connection.prepareStatement(query);

            preparedStatement.setString(1, "Jinesh");

            preparedStatement.setString(2, "Doe");

            preparedStatement.setString(3, "john.doe@example.com");

            preparedStatement.setDate(4, new Date(System.currentTimeMillis()));

            preparedStatement.setDouble(5, 50000.00);

            preparedStatement.executeUpdate();

            System.out.println("Record created successfully.");

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

    private static void readRecord(){

        String query = "SELECT \* FROM emp";

        try {

            statement = connection.createStatement();

            resultSet = statement.executeQuery(query);

            while (resultSet.next()) {

                System.out.println(resultSet.getString("first\_name") + " " + resultSet.getString("last\_name"));

            }

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

    private static void updateRecord(){

        String query = "UPDATE emp SET salary = ? WHERE id = ?";

        try {

            preparedStatement = connection.prepareStatement(query);

            preparedStatement.setDouble(1, 60000.00);

            preparedStatement.setInt(2, 4);

            preparedStatement.executeUpdate();

            System.out.println("Record updated successfully.");

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

    private static void deleteRecord(){

        String query = "DELETE FROM emp WHERE id = ?";

        try {

            preparedStatement = connection.prepareStatement(query);

            preparedStatement.setInt(1, 1);

            preparedStatement.executeUpdate();

            System.out.println("Record deleted successfully.");

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

}

**2sum.java**

import java.util.\*;

public class twosum {

    public static void **main**(String[] *args*) {

*//     int arr[]={4,5,3,2};*

*//     int target=6;*

*//     HashMap<Integer,Integer> hs=new HashMap<>();*

*//     for(int i=0;i<arr.length;i++)*

*//     {*

*//         int s1=arr[i];*

*//         if(hs.containsKey(target-s1))*

*//         {*

*//             System.out.println(hs.get(target-s1)+" "+i);*

*//             break;*

*//         }*

*//         hs.put(s1,i);*

*//     }*

*//optimized soluation using two pointer approach*

    int arr[]={1,2,3,4,5,6};

    int left=0;

    int right=arr.length-1;

    int target=6;

    Arrays.sort(arr);

    while(left<right)

    {

        if(arr[left]+arr[right]==target)

        {

            System.out.println(left+" "+right);

            return ;

        }

        else if(arr[left]+arr[right]<target)

        {

            left++;

        }

        else

        {

            right--;

        }

    }

    }

}

Output::

0 4

**Logestsubsequence.java**

import java.util.\*;

public class longestsubsequence {

    public static void **main**(String[] *args*) {

        int arr[]={11,12,14,15,1,2,3,4,5,6,7,8,101,102,103,104,105,106,107,108,109,110};

        Arrays.sort(arr);

        int count=0;

        int max=0;

*//{1,2,3,4,5,6,7,8,101,102,103,104,105,106,107,108,109,110}*

        for(int i=0;i<arr.length-1;i++)

        {

            if(arr[i+1]-arr[i]==1)

            {

                count++;

            }

            else

            {

                count=0;

            }

            max=Math.max(max,count);

        }

        System.out.println("longest subsequence is::"+(max+1));

    }

}

**Buyingandsellingstock.java**

import java.util.\*;

class buyingandsellingstock {

public static void main(String[] args) {

int arr[]={7,6,4,3,1,9};

int min=arr[0];

int max=Integer.MIN\_VALUE;

for(int i=0;i<arr.length;i++)

{

min=Math.min(arr[i],min);

max=Math.max(max,arr[i]-min);

}

System.out.println("max profit::"+max);

}

}

Output::

max profit::8

**PROJECT**

**sql**

CREATE DATABASE ecommerce;

USE ecommerce;

CREATE TABLE products (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

price DECIMAL(10, 2),

stock INT

);

CREATE TABLE orders (

id INT AUTO\_INCREMENT PRIMARY KEY,

product\_id INT,

quantity INT,

FOREIGN KEY (product\_id) REFERENCES products(id)

);

**Ecommers.java**

package com.cache;

import java.util.\*;

import java.sql.\*;

import java.util.concurrent.\*;

class database{

private static final String url="jdbc:mysql://localhost:3306/ecommerce";

private static final String user="root";

private static final String password="root";

public static Connection createConnection(){

Connection connection=null;

try{

connection= DriverManager.getConnection(url,user,password);

return connection;

}catch(SQLException e){

System.out.println(e.getMessage());

}

return connection;

}

}

class Product{

private int id,stock;

private Double price;

private String name;

public Product(int id, int stock, Double price, String name) {

this.id = id;

this.stock=stock;

this.price=price;

this.name=name;

}

public int getId() {

return id;

}

public String getName() {

return name;

}

public double getPrice() {

return price;

}

public int getStock() {

return stock;

}

public void changeStock(int quantity){

this.stock-=quantity;

}

}

class productCache {

static HashMap<Integer,Product>cache=new HashMap<>();

public static Product getProduct(int id){

return cache.get(id);

}

public static void addProduct(Product product){

cache.put(product.getId(),product);

}

public static boolean containsItem(int id){

return cache.containsKey(id);

}

}

class Ecommerce{

public Product getProductById(int id){

if(productCache.containsItem(id)){

return productCache.getProduct(id);

}

try{

Connection connection=database.createConnection();

PreparedStatement preparedStatement = connection.prepareStatement("SELECT \* FROM products where id = ?");

preparedStatement.setInt(1,id);

ResultSet resultSet = preparedStatement.executeQuery();

Product product = new Product(resultSet.getInt("id"),resultSet.getInt("stock"),resultSet.getDouble("price"),resultSet.getString("name"));

productCache.addProduct(product);

return product;

}catch(SQLException e){

System.out.println(e.getMessage());

}

return null;

}

public void placeOrder(int id, int quantity){

Product product = getProductById(id);

if(product.getStock()<quantity){

System.out.println("insufficient stock");

}

product.changeStock(quantity);

try{

Connection connection = database.createConnection();

PreparedStatement updateStock = connection.prepareStatement("UPDATE products SET stock = stock - ? WHERE id = ?");

PreparedStatement insertOrder = connection.prepareStatement("INSERT INTO orders (product\_id, quantity) VALUES (?, ?)");

updateStock.setInt(1,quantity);

updateStock.setInt(2,id);

updateStock.executeUpdate();

insertOrder.setInt(1,id);

insertOrder.setInt(2,quantity);

insertOrder.executeUpdate();

System.out.println("order placed successfully for product: "+product.getName());

}catch(SQLException e){

e.printStackTrace();

}

}

}

class order extends Thread{

private int id,quantity;

private Ecommerce commerce;

public order(int id, int quantity, Ecommerce commerce){

this.id=id;

this.quantity=quantity;

this.commerce=commerce;

}

public void run(){

try{

commerce.placeOrder(id,quantity);

}catch(Exception e){

e.printStackTrace();

}

}

}

public class eCommerce{

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

Ecommerce commerce = new Ecommerce();

ExecutorService executorService = Executors.newCachedThreadPool();

try{

Connection connection=database.createConnection();

PreparedStatement preparedStatement = connection.prepareStatement("SELECT \* FROM products");

ResultSet resultSet=preparedStatement.executeQuery();

while(resultSet.next()){

System.out.println("id: "+resultSet.getInt("id")+" name: "+resultSet.getString("name")+" price: "+resultSet.getDouble("price")+" stock: "+resultSet.getInt("stock"));

}

}catch(SQLException e){

e.printStackTrace();

}

while (true) {

System.out.println("Enter Product ID to order (0 to exit): ");

int id = scan.nextInt();

if (id == 0)

break;

System.out.println("Enter quantity: ");

int quantity = scan.nextInt();

executorService.execute(new order(id, quantity, commerce));

}

executorService.shutdown();

scan.close();

}

}

**Sql explaination**

**Purpose:**

This code creates a database structure for an e-commerce application, specifically defining tables to store product and order information.

**Breakdown:**

1. **Creating the Database:**

* CREATE DATABASE ecommerce;
* This statement creates a new database named "ecommerce" on your MySQL server. This database will serve as a container for all the tables and data related to your e-commerce application.

1. **Selecting the Database:**

* USE ecommerce;
* This statement switches the current database to "ecommerce." This ensures that any subsequent SQL statements will operate within the context of this database.

1. **Creating the products Table:**

* CREATE TABLE products ( ... );
* This statement defines a table named "products" within the "ecommerce" database.
* The id column is an integer that automatically increments for each new row, serving as the primary key. This means it uniquely identifies each product.
* The name column stores the product name as a text string (maximum 100 characters).
* The price column stores the product price as a decimal number with 10 digits before the decimal point and 2 digits after. This is suitable for representing monetary values.
* The stock column stores the quantity of the product available in inventory.

1. **Creating the orders Table:**

* CREATE TABLE orders ( ... );
* This statement defines a table named "orders" within the "ecommerce" database.
* The id column is an integer that automatically increments for each new row, serving as the primary key.
* The product\_id column stores the ID of the product associated with the order. This establishes a relationship with the products table.
* The quantity column stores the quantity of the product ordered.
* The FOREIGN KEY (product\_id) REFERENCES products(id) constraint defines a foreign key relationship between the orders table and the products table. This ensures that the product\_id in an order must match the id of an existing product, maintaining data integrity.

**Overall Structure:**

This database structure provides a foundation for an e-commerce application by storing information about products and orders. The products table maintains details of available products, while the orders table tracks customer purchases. The foreign key relationship between the two tables ensures that orders are associated with valid products.

This structure can be further enhanced by adding more columns to both tables, such as:

* **products:** Description, image URL, category, etc.
* **orders:** Customer information, order date, shipping address, payment status, etc.

**code explanation**

**Explanation of the E-commerce Java Code**

This code provides a basic framework for an e-commerce application that utilizes caching, database connectivity, and multithreading. Here's a breakdown:

**Packages:**

* com.cache: Likely a custom package to group cache-related classes.
* java.util: Provides essential classes like HashMap, Scanner, and ExecutorService.
* java.sql: Contains classes for interacting with databases using JDBC.

**Classes:**

1. **Database:**

* Static connection details like URL, username, and password.
* createConnection() method establishes a connection to the MySQL database "ecommerce" and returns the Connection object.

1. **Product:**

* Represents a product with id, stock, price, and name.
* Constructor initializes product properties.
* Getters for each product attribute.
* changeStock(int quantity) method updates the product's stock by subtracting the ordered quantity.

1. **productCache:**

* Static HashMap named cache stores Product objects keyed by their ID.
* getProduct(int id) retrieves a product from the cache if it exists.
* addProduct(Product product) adds a product to the cache.
* containsItem(int id) checks if the cache contains a product with the given ID.

1. **Ecommerce:**

* getProductById(int id):
* Checks the cache for the product first.
* If not found, retrieves the product from the database using a prepared statement and ResultSet.
* Creates a Product object from the database results.
* Adds the product to the cache.
* Returns the product.
* placeOrder(int id, int quantity):
* Retrieves the product using getProductById(id).
* Checks if there's sufficient stock.
* If sufficient, updates the product's stock quantity using changeStock(quantity).
* Uses prepared statements for efficient database updates:
* One statement updates the product's stock in the products table.
* Another statement inserts a new order record into the orders table (assuming it exists).
* Prints a success message if the order is placed.

1. **order (extends Thread):**

* Represents an order thread.
* Constructor takes product ID, quantity, and an Ecommerce object as parameters.
* run() method calls commerce.placeOrder(id, quantity) to process the order.

1. **public class eCommerce (main method):**

* Creates a Scanner object for user input.
* Establishes an Ecommerce object.
* Creates a cached thread pool using Executors.newCachedThreadPool().
* Optionally, fetches all products from the database and displays them to the user (commented out).
* Enters a loop where the user can:
* Enter a product ID to order (0 to exit).
* Enter the desired quantity.
* Creates a new order thread and submits it to the thread pool for concurrent execution.
* Shuts down the thread pool and closes the scanner after user exits.

**Key Points:**

* This code demonstrates caching for performance optimization. Product information is retrieved from the database only if it's not already in the cache.
* Prepared statements are used to prevent SQL injection vulnerabilities.
* Multithreading allows concurrent order processing, potentially improving responsiveness under high load. This is particularly beneficial in real-world e-commerce scenarios.
* Error handling is included at various points (commented out in some cases) to catch potential exceptions.

—--------------------------------------------------------------------

**Mergsort.java**

**Merge Sort** is a sorting algorithm that follows the **divide-and-conquer** approach. It works by recursively dividing the array into smaller sub-arrays, sorting each sub-array, and then merging the sorted sub-arrays to produce a sorted array.

**How Merge Sort Works:**

1. **Divide:**

* If the array contains more than one element, it is divided into two approximately equal-sized sub-arrays.

1. **Conquer:**

* Each sub-array is recursively sorted using the same Merge Sort algorithm.

1. **Combine:**

* The sorted sub-arrays are merged together to form a single sorted array.

**Key Steps in Merging:**

1. **Create a temporary array:** This array will store the merged elements.
2. **Initialize pointers:** Two pointers are initialized, one for each sub-array.
3. **Compare and copy:**

* While both pointers are within their respective sub-arrays:
* Compare the elements pointed to by each pointer.
* Copy the smaller element into the temporary array and increment the corresponding pointer.

1. **Copy remaining elements:**

* If one sub-array is empty, copy the remaining elements from the other sub-array into the temporary array.

1. **Copy merged elements back:** Copy the elements from the temporary array back into the original array.

**Time Complexity:**

* **Best case:** O(n log n)
* **Average case:** O(n log n)
* **Worst case:** O(n log n)

**Space Complexity:**

* O(n) due to the temporary array created during merging.

**Advantages of Merge Sort:**

* **Stable:** Preserves the relative order of elements with equal values.
* **Efficient:** Has a time complexity of O(n log n) in all cases.
* **Simple to implement:** The algorithm is relatively straightforward to understand and code.

**Disadvantages of Merge Sort:**

* **Extra space:** Requires additional space for the temporary array.
* **Inefficient for small arrays:** Can be less efficient than simpler algorithms like insertion sort for small inputs.

import java.io.\*;

class mergsort {

// Merges two subarrays of arr[].

// First subarray is arr[l..m]

// Second subarray is arr[m+1..r]

static void merge(int arr[], int l, int m, int r)

{

int n1 = m - l + 1;

int n2 = r - m;

int L[] = new int[n1];

int R[] = new int[n2];

for (int i = 0; i < n1; ++i)

L[i] = arr[l + i];

for (int j = 0; j < n2; ++j)

R[j] = arr[m + 1 + j];

int i = 0, j = 0;

int k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

static void sort(int arr[], int l, int r)

{

if (l < r) {

int m = l + (r - l) / 2;

sort(arr, l, m);

sort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

static void printArray(int arr[])

{

int n = arr.length;

for (int i = 0; i < n; ++i)

System.out.print(arr[i] + " ");

System.out.println();

}

public static void main(String args[])

{

int arr[] = { 12, 11, 13, 5, 6, 7 };

System.out.println("Given array is");

printArray(arr);

sort(arr, 0, arr.length - 1);

System.out.println("\nSorted array is");

printArray(arr);

}

}

Output:

Given array is

12 11 13 5 6 7

Sorted array is

5 6 7 11 12 13

**Quicksort.java**

**Quick Sort** is another efficient sorting algorithm that follows the **divide-and-conquer** approach. It works by partitioning the array into two sub-arrays based on a pivot element, sorting each sub-array recursively, and then combining the sorted sub-arrays.

**How Quick Sort Works:**

1. **Partition:**

* Choose a pivot element from the array (e.g., the first, last, or a random element).
* Rearrange the elements such that all elements smaller than the pivot are on one side, and all elements larger than the pivot are on the other side.
* The pivot's final position is its correct sorted position.

1. **Divide:**

* Recursively sort the sub-array to the left of the pivot and the sub-array to the right of the pivot.

**Key Steps in Partitioning:**

1. **Initialize pointers:** Two pointers, i and j, are initialized. i starts at the leftmost index, and j starts at the rightmost index.
2. **Scan and swap:**

* While i is less than j:
* If the element at i is less than or equal to the pivot, increment i.
* If the element at j is greater than the pivot, decrement j.
* If the element at i is greater than the pivot and the element at j is less than or equal to the pivot, swap the elements at i and j.

1. **Place the pivot:**

* When i and j cross, swap the pivot with the element at i. The pivot is now in its correct sorted position.

**Time Complexity:**

* **Best case:** O(n log n) (when the pivot is always the median)
* **Average case:** O(n log n)
* **Worst case:** O(n^2) (when the pivot is always the smallest or largest element)

**Space Complexity:**

* O(log n) on average due to the recursive function calls. In the worst case, it can be O(n) if the pivot is always the smallest or largest element.

**Advantages of Quick Sort:**

* **Efficient:** Has a time complexity of O(n log n) on average.
* **In-place:** Does not require additional space beyond the original array.
* **Can be optimized:** Various pivot selection strategies and partitioning algorithms can be used to improve performance.

**Disadvantages of Quick Sort:**

* **Worst-case performance:** Can be inefficient in the worst case.
* **Pivot selection:** The choice of pivot can significantly affect performance.

import java.io.\*;

class QuickSort {

// A utility function to swap two elements

static void swap(int[] arr, int i, int j)

{

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

static int partition(int[] arr, int low, int high)

{

// Choosing the pivot

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++) {

if (arr[j] < pivot) {

// Increment index of smaller element

i++;

swap(arr, i, j);

}

}

swap(arr, i + 1, high);

return (i + 1);

}

static void quickSort(int[] arr, int low, int high)

{

if (low < high) {

// pi is partitioning index, arr[p]

// is now at right place

int pi = partition(arr, low, high);

// Separately sort elements before

// partition and after partition

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

public static void printArr(int[] arr)

{

for (int i = 0; i < arr.length; i++) {

System.out.print(arr[i] + " ");

}

}

// Driver Code

public static void main(String[] args)

{

int[] arr = { 10, 7, 8, 9, 1, 5 };

int N = arr.length;

// Function call

quickSort(arr, 0, N - 1);

System.out.println("Sorted array:");

printArr(arr);

}

}

Output::

Sorted Array

1 5 7 8 9 10

—-----------------------------------------

**Reversstring.java**

import java.util.\*;

class main

{

public static void main (String[] args) {

String str="Bhanu Prakash from telangana";

String newstr="";

String arr[]=str.split(" ");

for(int i=arr.length-1;i>=0;i--)

{

System.out.print(arr[i]+" ");

}

//approach 2

Stack<String> st=new Stack<>();

for(int i=0;i<str.length();i++)

{

if(str.charAt(i)==' ')

{

st.push(newstr);

newstr="";

}

else

{

newstr=newstr+str.charAt(i);

}

}

st.push(newstr);

// System.out.println(st.size());

str="";

while(st.size()!=0)

{

str=str+" "+st.pop();

}

System.out.println(str);

}

}

Output:

telangana from Prakash Bhanu telangana from Prakash Bhanu

**coins.java**

import java.io.\*;

public class coin

{

static int minCoins(int coins[], int m, int sum)

{

if (sum == 0) return 0;

int res = Integer.MAX\_VALUE;

for (int i=0; i<m; i++)

{

if (coins[i] <= sum)

{

int sub\_res = minCoins(coins, m, sum-coins[i]);

if (sub\_res != Integer.MAX\_VALUE && sub\_res + 1 < res)

res = sub\_res + 1;

}

}

return res;

}

public static void main(String args[])

{

int coins[] = {9, 6, 5, 1};

int m = coins.length;

int sum = 11;

System.out.println("Minimum coins required is "+ minCoins(coins, m, sum) );

}

}

Output::

Minimum coins required is 2

**anagrams.java**

import java.io.\*;

class anagram {

public static void main (String[] args) {

String s="ABCA";

String s1="CBA";

int arr[]=new int[26];

int c=0;

if(s.length()==s1.length())

{

for(int i=0;i<s.length();i++)

{

arr[s.charAt(i)-'A']++;

}

for(int i=0;i<s.length();i++)

{

arr[s1.charAt(i)-'A']--;

}

}

else

{

System.out.println("not anagram");

return ;

}

for(int i=0;i<26;i++)

{

if(!(arr[i]==0))

{

System.out.println("not anagram");

return ;

}

}

System.out.println("anagram");

}

}

Output:

not anagram

—-----------------------------------------------------------------

DSA

**Display unique elements in array**

import java.util.\*;   
class main   
{   
    public static void main(String args[])   
    {   
        int arr[]={1,2,3,4,5,1,2,3,4,5,99,88,1};   
        HashMap<Integer,Integer> hs=new HashMap<Integer,Integer>();   
        for(int i=0;i<arr.length;i++)   
        {   
            if(!hs.containsKey(arr[i]))   
            {   
                hs.put(arr[i],1);   
            }   
            else   
            {   
                int m=hs.get(arr[i]);   
                m++;   
                hs.put(arr[i],m);   
            }   
        }   
        System.out.println("non duplicates are::");   
        for(int i:hs.keySet())   
        {   
            if(hs.get(i)==1)   
            {   
                System.out.println(i);   
            }   
        }   
    }   
}

output::

non duplicates are::

99

88

**Search in 2d matrix**

import java.util.Arrays;

public class matrix {

public static void main(String[] args)

{

int arr[][] = { { 3, 12, 9 },

{ 5, 2, 89 },

{ 90, 45, 22 } };

int target = 89;

int ans[] = linearSearch(arr, target);

System.out.println("Element found at index: "

+ Arrays.toString(ans));

}

static int[] linearSearch(int[][] arr, int target)

{

for (int i = 0; i < arr.length; i++) {

for (int j = 0; j < arr[i].length; j++) {

if (arr[i][j] == target) {

return new int[] { i, j };

}

}

}

return new int[] { -1, -1 };

}

}

**Longestpaindrome**

class Solution {

public String findLongestPalindrome(String inputString) {

if (inputString == null || inputString.length() < 2) {

return inputString;

}

int startIndex = 0, maxLength = 1;

for (int currentIndex = 0; currentIndex < inputString.length(); currentIndex++) {

int palindromeLength1 = expandPalindrome(inputString, currentIndex, currentIndex);

int palindromeLength2 = expandPalindrome(inputString, currentIndex, currentIndex + 1);

int currentLength = Math.max(palindromeLength1, palindromeLength2);

if (currentLength > maxLength) {

startIndex = currentIndex - (currentLength - 1) / 2;

maxLength = currentLength;

}

}

return inputString.substring(startIndex, startIndex + maxLength);

}

private int expandPalindrome(String inputString, int leftIndex, int rightIndex) {

while (leftIndex >= 0 && rightIndex < inputString.length() && inputString.charAt(leftIndex) == inputString.charAt(rightIndex)) {

leftIndex--;

rightIndex++;

}

return rightIndex - leftIndex - 1;

}

}

**Hashmap implementation**

class SimpleHashMap<K, V> {

private Entry<K, V>[] table;

private int size;

private static final double LOAD\_FACTOR\_THRESHOLD = 0.75;

public SimpleHashMap(int capacity) {

table = new Entry[capacity];

size = 0;

}

public void put(K key, V value) {

if (key == null) {

throw new IllegalArgumentException("Null keys are not allowed");

}

if ((double) size / table.length >= LOAD\_FACTOR\_THRESHOLD) {

resize();

}

int index = hash1(key);

int step = hash2(key);

int i = 0;

while (table[index] != null && !table[index].getKey().equals(key)) {

index = (index + step \* i) % table.length;

i++;

}

table[index] = new Entry<>(key, value);

size++;

}

public V get(K key) {

if (key == null) {

throw new IllegalArgumentException("Null keys are not allowed");

}

int index = hash1(key);

int step = hash2(key);

int i = 0;

while (table[index] != null) {

if (table[index].getKey().equals(key)) {

return table[index].getValue();

}

index = (index + step \* i) % table.length;

i++;

}

return null;

}

private void resize() {

Entry<K, V>[] oldTable = table;

table = new Entry[oldTable.length \* 2];

size = 0;

for (Entry<K, V> entry : oldTable) {

if (entry != null) {

put(entry.getKey(), entry.getValue());

}

}

}

private int hash1(K key) {

return Math.abs(key.hashCode()) % table.length;

}

private int hash2(K key) {

return 1 + (Math.abs(key.hashCode()) % (table.length - 1));

}

private static class Entry<K, V> {

private K key;

private V value;

public Entry(K key, V value) {

this.key = key;

this.value = value;

}

public K getKey() {

return key;

}

public V getValue() {

return value;

}

}

}

—--------------------------------------------------

SQL-MYSQL

**SQL**

SQL constraints are essential rules that govern the data integrity within a database table. They ensure that the data adheres to specific conditions, enhancing both accuracy and reliability. Below is a detailed overview of various SQL constraints, their usage, and examples.

Overview of SQL Constraints

Constraints can be defined at two levels:

- Column Level: Applies to a single column.

- Table Level: Applies to multiple columns.

Constraints can be created during table creation using the `CREATE TABLE` statement or added later with the `ALTER TABLE` statement.

Common SQL Constraints

| Constraint | Description |

|-------------------|---------------------------------------------------------------------------------------------------|

| NOT NULL | Ensures that a column cannot have NULL values. |

| UNIQUE | Ensures all values in a column are distinct. |

| PRIMARY KEY | A combination of `NOT NULL` and `UNIQUE`, uniquely identifies each row in a table. |

| FOREIGN KEY | Establishes a relationship between two tables by referencing a primary key in another table. |

| CHECK | Validates that values in a column satisfy a specific condition. |

| DEFAULT | Sets a default value for a column if no value is provided during insertion. |

| CREATE INDEX | Improves the speed of data retrieval operations on a table. |

Detailed Explanation of Each Constraint

1. NOT NULL

The `NOT NULL` constraint ensures that a column cannot contain NULL values, enforcing that every record must have a value for this column.

Example:

sql

CREATE TABLE Students (

ID INT NOT NULL,

Name VARCHAR(50) NOT NULL

);

2. UNIQUE

The `UNIQUE` constraint guarantees that all values in a column are different from one another, preventing duplicate entries.

Example:

sql

CREATE TABLE Users (

UserID INT UNIQUE,

Email VARCHAR(100) UNIQUE

);

3. PRIMARY KEY

A `PRIMARY KEY` uniquely identifies each record in a table and cannot accept NULL values. It combines the properties of both `NOT NULL` and `UNIQUE`.

Example:

sql

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT NOT NULL

);

4. FOREIGN KEY

The `FOREIGN KEY` constraint links two tables together, ensuring that the value in one table corresponds to an existing value in another table.

Example:

sql

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

5. CHECK

The `CHECK` constraint allows you to specify conditions on the values in a column, ensuring they meet certain criteria before being accepted.

Example:

sql

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Age INT CHECK (Age >= 18)

);

6. DEFAULT

The `DEFAULT` constraint sets a default value for a column when no value is specified during record insertion.

Example:

sql

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

Price DECIMAL(10, 2) DEFAULT 0.00

);

7. CREATE INDEX

The `CREATE INDEX` statement is used to create an index on one or more columns of a table, which improves the speed of data retrieval.

Example:

sql

CREATE INDEX idx\_lastname ON Employees(LastName);

Creating Constraints

You can create constraints using either the `CREATE TABLE` or `ALTER TABLE` statements.

Syntax for CREATE TABLE

sql

CREATE TABLE table\_name (

column1 datatype constraint\_name,

column2 datatype constraint\_name,

...

);

Syntax for ALTER TABLE

sql

ALTER TABLE table\_name

ADD CONSTRAINT constraint\_name

constraint\_type(column\_name);

Viewing Existing Constraints

To view existing constraints on a table, you can query the information schema:

sql

SELECT CONSTRAINT\_NAME, CONSTRAINT\_TYPE

FROM INFORMATION\_SCHEMA.TABLE\_CONSTRAINTS

WHERE TABLE\_NAME = 'your\_table\_name';

Syntax

sql

SELECT column1, column2, ...

FROM table\_name

WHERE column\_name LIKE pattern;

Wildcard Characters

| Wildcard | Description |

|--------------|---------------------------------------------------|

| `%` | Matches any sequence of characters (including none). |

| `\_` | Matches exactly one character. |

Examples of Using LIKE

1. Basic Usage with %

To find all records where the name starts with "A":

sql

SELECT \*

FROM Customers

WHERE Name LIKE 'A%';

2. Usage with \_

To find all records where the name has "a" as the second character:

sql

SELECT \*

FROM Customers

WHERE Name LIKE '\_a%';

3. Combining Wildcards

To find names that contain "an" anywhere within:

sql

SELECT \*

FROM Customers

WHERE Name LIKE '%an%';

5. Escaping Wildcards

If you need to search for a literal `%` or `\_`, you can escape these characters using a backslash (`\`) or another escape character defined by your database.

Example:

sql

SELECT \*

FROM Products

WHERE ProductName LIKE '100\%' ESCAPE '\';

Practical Applications

1. Searching for Email Patterns

To find all users with Gmail addresses:

sql

SELECT \*

FROM Users

WHERE Email LIKE '%@gmail.com';

2. Finding Partial Matches

To search for products that include "Pro" in their names:

sql

SELECT \*

FROM Products

WHERE ProductName LIKE '%Pro%';

3. Filtering Data in Reports

When generating reports, you can filter records based on specific patterns, such as finding all orders placed in January:

sql

SELECT \*

FROM Orders

WHERE OrderDate LIKE '2024-01%';

Here are the SQL queries to address each of your questions based on the provided sample data setup.

1. Top 3 Highest Paid Employees in Each Department

sql

WITH RankedEmployees AS (

SELECT

employee\_id,

first\_name,

last\_name,

department,

salary,

ROW\_NUMBER() OVER (PARTITION BY department ORDER BY salary DESC) AS rank

FROM employees

)

SELECT employee\_id, first\_name, last\_name, department, salary

FROM RankedEmployees

WHERE rank <= 3;

2. Running Total of Salaries in Each Department Ordered by Hire Date

sql

SELECT

employee\_id,

first\_name,

last\_name,

department,

salary,

SUM(salary) OVER (PARTITION BY department ORDER BY hire\_date) AS running\_total

FROM employees

ORDER BY department, hire\_date;

3. Employees Working on More Than One Project

sql

SELECT

e.employee\_id,

e.first\_name,

e.last\_name,

COUNT(ep.project\_id) AS project\_count

FROM employees e

JOIN employee\_projects ep ON e.employee\_id = ep.employee\_id

GROUP BY e.employee\_id, e.first\_name, e.last\_name

HAVING COUNT(ep.project\_id) > 1;

4. Projects with Team Members from All Departments

sql

SELECT p.project\_id, p.project\_name

FROM projects p

JOIN employee\_projects ep ON p.project\_id = ep.project\_id

JOIN employees e ON ep.employee\_id = e.employee\_id

GROUP BY p.project\_id, p.project\_name

HAVING COUNT(DISTINCT e.department) = (SELECT COUNT(DISTINCT department) FROM employees);

5. Average Salary for Each Department (Last 3 Years)

sql

SELECT

department,

AVG(salary) AS average\_salary

FROM employees

WHERE hire\_date >= DATEADD(YEAR, -3, GETDATE())

GROUP BY department;

6. Pivot Table for Employee Count by Salary Ranges

sql

SELECT

department,

SUM(CASE WHEN salary < 65000 THEN 1 ELSE 0 END) AS "<65000",

SUM(CASE WHEN salary BETWEEN 65000 AND 75000 THEN 1 ELSE 0 END) AS "65000-75000",

SUM(CASE WHEN salary > 75000 THEN 1 ELSE 0 END) AS ">75000"

FROM employees

GROUP BY department;

7. Employee(s) with Highest Salary in Their Departments on Longest-Running Project

sql

WITH LongestRunningProject AS (

SELECT

project\_id,

DATEDIFF(DAY, start\_date, end\_date) AS duration

FROM projects

ORDER BY duration DESC

LIMIT 1 -- Assuming SQL dialect supports LIMIT; use TOP or FETCH FIRST as needed.

),

HighestPaidInDepartments AS (

SELECT

e.department,

MAX(e.salary) AS max\_salary

FROM employees e

JOIN employee\_projects ep ON e.employee\_id = ep.employee\_id

WHERE ep.project\_id IN (SELECT project\_id FROM LongestRunningProject)

GROUP BY e.department

)

SELECT

e.\*

FROM employees e

JOIN HighestPaidInDepartments hpd ON e.department = hpd.department AND e.salary = hpd.max\_salary;

8. Percentage of Each Department's Salary Compared to Total Salary

sql

WITH TotalSalaries AS (

SELECT SUM(salary) AS total\_salary FROM employees

)

SELECT

department,

SUM(salary) AS department\_salary,

(SUM(salary) \* 100.0 / (SELECT total\_salary FROM TotalSalaries)) AS percentage\_of\_total\_salary

FROM employees

GROUP BY department;

9. Employees with Salary Higher than Department Average

sql

WITH DepartmentAverages AS (

SELECT

department,

AVG(salary) AS avg\_salary

FROM employees

GROUP BY department

)

SELECT

e.employee\_id,

e.first\_name,

e.last\_name,

e.salary,

da.avg\_salary,

((e.salary - da.avg\_salary) / da.avg\_salary \* 100) AS percentage\_exceeding\_average

FROM employees e

JOIN DepartmentAverages da ON e.department = da.department

WHERE e.salary > da.avg\_salary;

—----------------------------------------------------

**Primary Key**

The PRIMARY KEY constraint uniquely identifies each record in a table.

**FOREIGN KEY**

The FOREIGN KEY constraint is a key used to link two tables together.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

CREATE TABLE Orders (   
    OrderID int NOT NULL,   
    OrderNumber int NOT NULL,   
    PersonID int,   
    PRIMARY KEY (OrderID),   
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)   
);

**Generalization**

Generalization is the process of extracting common properties from a set of entities and creating a generalized entity from it. It is a bottom-up approach in which two or more entities can be generalized to a higher-level entity if they have some attributes in common.

Generalization is also called as Bottom-up approach.

**Advanced SQL Practice Questions**

create database testdb1;

use testdb1;

CREATE TABLE customers (

customer\_id INT PRIMARY KEY AUTO\_INCREMENT,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

email VARCHAR(100) UNIQUE,

registration\_date DATE

);

CREATE TABLE products (

product\_id INT PRIMARY KEY AUTO\_INCREMENT,

product\_name VARCHAR(100),

category VARCHAR(50),

price DECIMAL(10, 2),

stock\_quantity INT

);

CREATE TABLE orders (

order\_id INT PRIMARY KEY AUTO\_INCREMENT,

customer\_id INT,

order\_date DATETIME,

total\_amount DECIMAL(10, 2),

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

CREATE TABLE order\_items (

order\_item\_id INT PRIMARY KEY AUTO\_INCREMENT,

order\_id INT,

product\_id INT,

quantity INT,

price\_per\_unit DECIMAL(10, 2),

FOREIGN KEY (order\_id) REFERENCES orders(order\_id),

FOREIGN KEY (product\_id) REFERENCES products(product\_id)

);

CREATE TABLE product\_reviews (

review\_id INT PRIMARY KEY AUTO\_INCREMENT,

product\_id INT,

customer\_id INT,

rating INT CHECK (rating BETWEEN 1 AND 5),

review\_text TEXT,

review\_date DATE,

FOREIGN KEY (product\_id) REFERENCES products(product\_id),

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

INSERT INTO customers (first\_name, last\_name, email, registration\_date) VALUES

('John', 'Doe', 'john.doe@example.com', '2023-01-15'),

('Jane', 'Smith', 'jane.smith@example.com', '2023-02-20'),

('Alice', 'Johnson', 'alice.johnson@example.com', '2023-03-05'),

('Bob', 'Williams', 'bob.williams@example.com', '2023-04-10'),

('Charlie', 'Brown', 'charlie.brown@example.com', '2023-05-12'),

('David', 'Jones', 'david.jones@example.com', '2023-06-25'),

('Eva', 'Garcia', 'eva.garcia@example.com', '2023-07-30'),

('Frank', 'Martinez', 'frank.martinez@example.com', '2023-08-15'),

('Grace', 'Davis', 'grace.davis@example.com', '2023-09-01'),

('Henry', 'Miller', 'henry.miller@example.com', '2023-09-10'),

('Isabella', 'Hernandez', 'isabella.hernandez@example.com', '2023-09-15'),

('Jack', 'Moore', 'jack.moore@example.com', '2023-09-20'),

('Kathy', 'Taylor', 'kathy.taylor@example.com', '2023-09-25'),

('Liam', 'Wilson', 'liam.wilson@example.com', '2023-09-30'),

('Mia', 'Anderson', 'mia.anderson@example.com', '2023-10-01'),

('Noah', 'Thomas', 'noah.thomas@example.com', '2023-10-02'),

('Olivia', 'Jackson', 'olivia.jackson@example.com', '2023-10-05'),

('Paul', 'White', 'paul.white@example.com', '2023-10-07'),

('Quinn', 'Harris', 'quinn.harris@example.com', '2023-10-08'),

('Rita', 'Clark', 'rita.clark@example.com', '2023-10-09');

INSERT INTO products (product\_name, category, price, stock\_quantity) VALUES

('Laptop', 'Electronics', 999.99, 50),

('Smartphone', 'Electronics', 599.99, 150),

('Tablet', 'Electronics', 399.99, 60),

('Headphones', 'Electronics', 149.99, 75),

('Monitor', 'Electronics', 299.99, 40),

('Printer', 'Electronics', 199.99, 25),

('Coffee Maker', 'Home Appliances', 89.99, 30),

('Blender', 'Home Appliances', 59.99, 20),

('Desk', 'Furniture', 249.99, 20),

('Chair', 'Furniture', 149.99, 45),

('Bookshelf', 'Furniture', 199.99, 15),

('Notebook', 'Stationery', 4.99, 200),

('Pen', 'Stationery', 1.99, 300),

('Eraser', 'Stationery', 0.99, 150),

('Backpack', 'Accessories', 39.99, 100),

('Camera', 'Electronics', 799.99, 10),

('Smartwatch', 'Electronics', 249.99, 70),

('Wireless Charger', 'Accessories', 29.99, 200),

('Bluetooth Speaker', 'Electronics', 59.99, 100),

('Gaming Console', 'Electronics', 499.99, 30);

INSERT INTO orders (customer\_id, order\_date, total\_amount) VALUES

(1, '2023-01-16 10:00:00', 999.99),

(2, '2023-02-21 11:30:00', 599.99),

(3, '2023-03-06 12:15:00', 399.99),

(4, '2023-04-11 14:00:00', 149.99),

(5, '2023-05-13 09:45:00', 89.99),

(6, '2023-06-26 10:15:00', 59.99),

(7, '2023-07-31 15:30:00', 299.99),

(8, '2023-08-16 10:30:00', 199.99),

(9, '2023-09-02 14:20:00', 249.99),

(10, '2023-09-11 16:45:00', 149.99),

(11, '2023-09-16 09:00:00', 399.99),

(12, '2023-09-21 13:30:00', 249.99),

(13, '2023-09-26 12:05:00', 59.99),

(14, '2023-09-27 14:15:00', 89.99),

(15, '2023-09-29 10:00:00', 19.99),

(16, '2023-10-01 11:20:00', 299.99),

(17, '2023-10-03 12:30:00', 499.99),

(18, '2023-10-04 13:10:00', 149.99),

(19, '2023-10-06 14:00:00', 59.99),

(20, '2023-10-08 15:00:00', 499.99);

INSERT INTO order\_items (order\_id, product\_id, quantity, price\_per\_unit) VALUES

(1, 1, 1, 999.99),

(2, 2, 1, 599.99),

(3, 3, 1, 399.99),

(4, 4, 2, 74.99),

(5, 5, 1, 89.99),

(6, 6, 1, 59.99),

(7, 7, 3, 199.99),

(8, 8, 2, 99.99),

(9, 9, 1, 249.99),

(10, 10, 1, 149.99),

(11, 11, 1, 399.99),

(12, 12, 2, 124.99),

(13, 13, 1, 59.99),

(14, 14, 1, 89.99),

(15, 15, 1, 19.99),

(16, 16, 1, 299.99),

(17, 17, 1, 499.99),

(18, 18, 2, 119.99),

(19, 19, 3, 59.99),

(20, 20, 2, 99.99);

INSERT INTO product\_reviews (product\_id, customer\_id, rating, review\_text, review\_date) VALUES

(1, 1, 5, 'Excellent laptop, very powerful.', '2023-01-17'),

(2, 2, 4, 'Great smartphone, very user-friendly.', '2023-02-22'),

(3, 3, 3, 'Good tablet but a bit pricey.', '2023-03-07'),

(4, 4, 4, 'Coffee maker works well, easy to use.', '2023-04-12'),

(5, 5, 5, 'Headphones have great sound quality.', '2023-05-14'),

(6, 6, 5, 'Printer is fast and reliable.', '2023-06-27'),

(7, 7, 3, 'Desk is sturdy, but hard to assemble.', '2023-07-31'),

(8, 8, 4, 'Chair is comfortable for long hours.', '2023-08-17'),

(9, 9, 2, 'Monitor is bright and clear.', '2023-09-03'),

(10, 10, 4, 'Nice quality Bluetooth speaker.', '2023-09-12'),

(11, 11, 5, 'Smartwatch is very helpful.', '2023-09-17'),

(12, 12, 3, 'Gaming console has great graphics.', '2023-09-22'),

(13, 13, 4, 'Backpack is spacious and stylish.', '2023-09-27'),

(14, 14, 2, 'Camera quality is average.', '2023-09-28'),

(15, 15, 5, 'E-reader is fantastic for books.', '2023-10-02'),

(16, 16, 4, 'Wireless charger is convenient.', '2023-10-04'),

(17, 17, 5, 'Blender works like a charm.', '2023-10-05'),

(18, 18, 4, 'Great quality skincare set.', '2023-10-06'),

(19, 19, 3, 'Good but not great dumbbells.', '2023-10-07'),

(20, 20, 4, 'Nice ergonomic chair.', '2023-10-09');

SELECT \* FROM customers;

SELECT \* FROM products;

SELECT \* FROM orders;

SELECT \* FROM order\_items;

SELECT \* FROM product\_reviews;

Question 1: Top 3 Customers by Total Spend

sql

SELECT

c.customer\_id,

CONCAT(c.first\_name, ' ', c.last\_name) AS customer\_name,

SUM(o.total\_amount) AS total\_spend,

COUNT(o.order\_id) AS order\_count

FROM

customers c

JOIN

orders o ON c.customer\_id = o.customer\_id

GROUP BY

c.customer\_id

ORDER BY

total\_spend DESC

LIMIT 3;

Question 2: Average Rating by Product Category (with at least 2 reviews)

sql

SELECT

p.category,

AVG(pr.rating) AS average\_rating

FROM

products p

JOIN

product\_reviews pr ON p.product\_id = pr.product\_id

GROUP BY

p.category

HAVING

COUNT(pr.review\_id) >= 2;

Question 3: Products Never Ordered

sql

SELECT

p.product\_id,

p.product\_name

FROM

products p

LEFT JOIN

order\_items oi ON p.product\_id = oi.product\_id

WHERE

oi.order\_item\_id IS NULL;

Question 4: Time Difference Between Registration and First Order Date for Each Customer

sql

SELECT

c.customer\_id,

CONCAT(c.first\_name, ' ', c.last\_name) AS customer\_name,

DATEDIFF(MIN(o.order\_date), c.registration\_date) AS days\_to\_first\_order

FROM

customers c

LEFT JOIN

orders o ON c.customer\_id = o.customer\_id

GROUP BY

c.customer\_id;

Question 5: Total Revenue for Each Month with Running Total

sql

SELECT

DATE\_FORMAT(order\_date, '%Y-%m') AS month,

SUM(total\_amount) AS monthly\_revenue,

SUM(SUM(total\_amount)) OVER (ORDER BY DATE\_FORMAT(order\_date, '%Y-%m')) AS running\_total

FROM

orders

GROUP BY

month;

Question 6: Customers Who Made a Purchase but Never Left a Review

sql

SELECT

DISTINCT c.customer\_id,

CONCAT(c.first\_name, ' ', c.last\_name) AS customer\_name

FROM

customers c

JOIN

orders o ON c.customer\_id = o.customer\_id

LEFT JOIN

product\_reviews pr ON c.customer\_id = pr.customer\_id

WHERE

pr.review\_id IS NULL;

Question 7: Most Ordered Product by Quantity

sql

SELECT

p.product\_id,

p.product\_name,

SUM(oi.quantity) AS total\_quantity\_ordered

FROM

products p

JOIN

order\_items oi ON p.product\_id = oi.product\_id

GROUP BY

p.product\_id

ORDER BY

total\_quantity\_ordered DESC

LIMIT 1;

Question 8: Percentage of Total Revenue by Product Category

sql

SELECT

p.category,

SUM(o.total\_amount) AS category\_revenue,

(SUM(o.total\_amount) / (SELECT SUM(total\_amount) FROM orders) \* 100) AS percentage\_of\_total\_revenue

FROM

products p

JOIN

order\_items oi ON p.product\_id = oi.product\_id

JOIN

orders o ON oi.order\_id = o.order\_id

GROUP BY

p.category;

Question 9: Most Frequently Purchased Product Category for Each Customer

sql

SELECT

customer\_id,

category,

COUNT(\*) AS purchase\_count,

RANK() OVER (PARTITION BY customer\_id ORDER BY COUNT(\*) DESC) as rank\_order

FROM (

SELECT

o.customer\_id,

p.category

FROM

orders o

JOIN

order\_items oi ON o.order\_id = oi.order\_id

JOIN

products p ON oi.product\_id = p.product\_id

) as subquery

GROUP BY customer\_id, category

HAVING rank\_order = 1;

Question 10: Distribution of Ratings for Each Product

sql

SELECT

product\_id,

rating,

COUNT(\*) AS rating\_count

FROM

product\_reviews

GROUP BY

product\_id, rating

ORDER BY

product\_id, rating;

MySQL Force Index Definition and Goal

* MySQL has a hint called the Force Index whereby developers can indicate clearly what index it should be using when issuing a query. Although MySQL's query optimizer usually makes an intelligent decision regarding which index it should use, there are instances where forcing a particular index advances its utility.
* The main purpose of the Force Index is to manage the query execution plan and ensure that a certain index has been used. This can occur in cases where the optimizer may make suboptimal decisions because of stale statistics or overly complicated conditions with queries.

create database test\_db\_poc;

use test\_db\_poc;

create table if not exists large\_table(

id int auto\_increment primary key,

name varchar(50),

value INT

);

DELIMITER //

CREATE procedure INSERT\_MILLION\_RECORDS()

BEGIN

declare i int default 0;

while i <100000 do

insert into large\_table(name,value) values(concat('Name',i),floor(1 + RAND()\*1000000));

set i=i+1;

end while;

END //

SHOW procedure status WHERE DB ='test\_db\_poc';

select count(\*) from large\_table;

CALL INSERT\_MILLION\_RECORDS();

select sql\_no\_cache sum(value) from large\_table

select sum(value) from large\_table;

create index idx\_value on large\_table(value)

alter table large\_table drop index idx\_value

select \* from large\_table

select a.value

from large\_table a

cross join large\_table b

on a.value=b.value\*100

**TRIGGER**

DELIMITER //

CREATE TRIGGER log\_phone\_changes

BEFORE UPDATE ON customers

FOR EACH ROW

BEGIN

IF old.phone != new.phone THEN

INSERT INTO phone\_changes\_log(customer\_id, old\_phone, new\_phone)

VALUES(old.id, old.phone, new.phone);

END IF;

END//

UPDATE customers

SET phone = '321-654-0987'

WHERE id = 1;

**Overview of Snowflake Schema**

The snowflake schema is a sophisticated data modeling technique used primarily in data warehousing. It is characterized by its multi-dimensional structure, where dimension tables are normalized into multiple related tables, creating a more intricate design than the simpler star schema.

Key Features

- Normalization: Unlike star schemas, where dimension tables are denormalized, snowflake schemas utilize a high degree of normalization. This means that dimension tables are broken down into sub-dimensions to eliminate redundancy and enhance data integrity.

- Hierarchical Structure: The schema resembles a snowflake due to its hierarchical organization. Each parent table can connect to multiple child tables, forming a complex network of relationships.

- Efficient Query Performance: While the normalization leads to more joins when querying data, it also allows for efficient retrieval of data through structured relationships. This can be particularly beneficial in environments with large datasets.

- Granular Data Representation: By breaking down dimension tables into smaller, attribute-focused tables, snowflake schemas provide detailed data analysis capabilities.

Use Cases

Snowflake schemas are particularly useful in industries that require complex data analysis, such as:

- Retail: For analyzing sales across various dimensions like time, product categories, and customer demographics.

- Healthcare: To manage and analyze patient data across various attributes and relationships.

- Manufacturing: For tracking production metrics against multiple factors like supply chain and inventory levels.

Advantages and Disadvantages

Advantages:

- Improved data integrity and quality due to normalization.

- Efficient storage usage by minimizing duplication.

- Enhanced analytical capabilities through detailed relationships.

Disadvantages:

- Increased complexity in database design.

- Potentially slower query performance due to the need for multiple joins.

- Higher maintenance costs associated with managing numerous tables[4][6].

—-----------------------------------------------------------

SPRING

**Primary Key**

The PRIMARY KEY constraint uniquely identifies each record in a table.

**FOREIGN KEY**

The FOREIGN KEY constraint is a key used to link two tables together.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

CREATE TABLE Orders (   
    OrderID int NOT NULL,   
    OrderNumber int NOT NULL,   
    PersonID int,   
    PRIMARY KEY (OrderID),   
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)   
);

**Generalization**

Generalization is the process of extracting common properties from a set of entities and creating a generalized entity from it. It is a bottom-up approach in which two or more entities can be generalized to a higher-level entity if they have some attributes in common.

Generalization is also called as Bottom-up approach.

**Advanced SQL Practice Questions**

create database testdb1;

use testdb1;

CREATE TABLE customers (

customer\_id INT PRIMARY KEY AUTO\_INCREMENT,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

email VARCHAR(100) UNIQUE,

registration\_date DATE

);

CREATE TABLE products (

product\_id INT PRIMARY KEY AUTO\_INCREMENT,

product\_name VARCHAR(100),

category VARCHAR(50),

price DECIMAL(10, 2),

stock\_quantity INT

);

CREATE TABLE orders (

order\_id INT PRIMARY KEY AUTO\_INCREMENT,

customer\_id INT,

order\_date DATETIME,

total\_amount DECIMAL(10, 2),

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

CREATE TABLE order\_items (

order\_item\_id INT PRIMARY KEY AUTO\_INCREMENT,

order\_id INT,

product\_id INT,

quantity INT,

price\_per\_unit DECIMAL(10, 2),

FOREIGN KEY (order\_id) REFERENCES orders(order\_id),

FOREIGN KEY (product\_id) REFERENCES products(product\_id)

);

CREATE TABLE product\_reviews (

review\_id INT PRIMARY KEY AUTO\_INCREMENT,

product\_id INT,

customer\_id INT,

rating INT CHECK (rating BETWEEN 1 AND 5),

review\_text TEXT,

review\_date DATE,

FOREIGN KEY (product\_id) REFERENCES products(product\_id),

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

INSERT INTO customers (first\_name, last\_name, email, registration\_date) VALUES

('John', 'Doe', 'john.doe@example.com', '2023-01-15'),

('Jane', 'Smith', 'jane.smith@example.com', '2023-02-20'),

('Alice', 'Johnson', 'alice.johnson@example.com', '2023-03-05'),

('Bob', 'Williams', 'bob.williams@example.com', '2023-04-10'),

('Charlie', 'Brown', 'charlie.brown@example.com', '2023-05-12'),

('David', 'Jones', 'david.jones@example.com', '2023-06-25'),

('Eva', 'Garcia', 'eva.garcia@example.com', '2023-07-30'),

('Frank', 'Martinez', 'frank.martinez@example.com', '2023-08-15'),

('Grace', 'Davis', 'grace.davis@example.com', '2023-09-01'),

('Henry', 'Miller', 'henry.miller@example.com', '2023-09-10'),

('Isabella', 'Hernandez', 'isabella.hernandez@example.com', '2023-09-15'),

('Jack', 'Moore', 'jack.moore@example.com', '2023-09-20'),

('Kathy', 'Taylor', 'kathy.taylor@example.com', '2023-09-25'),

('Liam', 'Wilson', 'liam.wilson@example.com', '2023-09-30'),

('Mia', 'Anderson', 'mia.anderson@example.com', '2023-10-01'),

('Noah', 'Thomas', 'noah.thomas@example.com', '2023-10-02'),

('Olivia', 'Jackson', 'olivia.jackson@example.com', '2023-10-05'),

('Paul', 'White', 'paul.white@example.com', '2023-10-07'),

('Quinn', 'Harris', 'quinn.harris@example.com', '2023-10-08'),

('Rita', 'Clark', 'rita.clark@example.com', '2023-10-09');

INSERT INTO products (product\_name, category, price, stock\_quantity) VALUES

('Laptop', 'Electronics', 999.99, 50),

('Smartphone', 'Electronics', 599.99, 150),

('Tablet', 'Electronics', 399.99, 60),

('Headphones', 'Electronics', 149.99, 75),

('Monitor', 'Electronics', 299.99, 40),

('Printer', 'Electronics', 199.99, 25),

('Coffee Maker', 'Home Appliances', 89.99, 30),

('Blender', 'Home Appliances', 59.99, 20),

('Desk', 'Furniture', 249.99, 20),

('Chair', 'Furniture', 149.99, 45),

('Bookshelf', 'Furniture', 199.99, 15),

('Notebook', 'Stationery', 4.99, 200),

('Pen', 'Stationery', 1.99, 300),

('Eraser', 'Stationery', 0.99, 150),

('Backpack', 'Accessories', 39.99, 100),

('Camera', 'Electronics', 799.99, 10),

('Smartwatch', 'Electronics', 249.99, 70),

('Wireless Charger', 'Accessories', 29.99, 200),

('Bluetooth Speaker', 'Electronics', 59.99, 100),

('Gaming Console', 'Electronics', 499.99, 30);

INSERT INTO orders (customer\_id, order\_date, total\_amount) VALUES

(1, '2023-01-16 10:00:00', 999.99),

(2, '2023-02-21 11:30:00', 599.99),

(3, '2023-03-06 12:15:00', 399.99),

(4, '2023-04-11 14:00:00', 149.99),

(5, '2023-05-13 09:45:00', 89.99),

(6, '2023-06-26 10:15:00', 59.99),

(7, '2023-07-31 15:30:00', 299.99),

(8, '2023-08-16 10:30:00', 199.99),

(9, '2023-09-02 14:20:00', 249.99),

(10, '2023-09-11 16:45:00', 149.99),

(11, '2023-09-16 09:00:00', 399.99),

(12, '2023-09-21 13:30:00', 249.99),

(13, '2023-09-26 12:05:00', 59.99),

(14, '2023-09-27 14:15:00', 89.99),

(15, '2023-09-29 10:00:00', 19.99),

(16, '2023-10-01 11:20:00', 299.99),

(17, '2023-10-03 12:30:00', 499.99),

(18, '2023-10-04 13:10:00', 149.99),

(19, '2023-10-06 14:00:00', 59.99),

(20, '2023-10-08 15:00:00', 499.99);

INSERT INTO order\_items (order\_id, product\_id, quantity, price\_per\_unit) VALUES

(1, 1, 1, 999.99),

(2, 2, 1, 599.99),

(3, 3, 1, 399.99),

(4, 4, 2, 74.99),

(5, 5, 1, 89.99),

(6, 6, 1, 59.99),

(7, 7, 3, 199.99),

(8, 8, 2, 99.99),

(9, 9, 1, 249.99),

(10, 10, 1, 149.99),

(11, 11, 1, 399.99),

(12, 12, 2, 124.99),

(13, 13, 1, 59.99),

(14, 14, 1, 89.99),

(15, 15, 1, 19.99),

(16, 16, 1, 299.99),

(17, 17, 1, 499.99),

(18, 18, 2, 119.99),

(19, 19, 3, 59.99),

(20, 20, 2, 99.99);

INSERT INTO product\_reviews (product\_id, customer\_id, rating, review\_text, review\_date) VALUES

(1, 1, 5, 'Excellent laptop, very powerful.', '2023-01-17'),

(2, 2, 4, 'Great smartphone, very user-friendly.', '2023-02-22'),

(3, 3, 3, 'Good tablet but a bit pricey.', '2023-03-07'),

(4, 4, 4, 'Coffee maker works well, easy to use.', '2023-04-12'),

(5, 5, 5, 'Headphones have great sound quality.', '2023-05-14'),

(6, 6, 5, 'Printer is fast and reliable.', '2023-06-27'),

(7, 7, 3, 'Desk is sturdy, but hard to assemble.', '2023-07-31'),

(8, 8, 4, 'Chair is comfortable for long hours.', '2023-08-17'),

(9, 9, 2, 'Monitor is bright and clear.', '2023-09-03'),

(10, 10, 4, 'Nice quality Bluetooth speaker.', '2023-09-12'),

(11, 11, 5, 'Smartwatch is very helpful.', '2023-09-17'),

(12, 12, 3, 'Gaming console has great graphics.', '2023-09-22'),

(13, 13, 4, 'Backpack is spacious and stylish.', '2023-09-27'),

(14, 14, 2, 'Camera quality is average.', '2023-09-28'),

(15, 15, 5, 'E-reader is fantastic for books.', '2023-10-02'),

(16, 16, 4, 'Wireless charger is convenient.', '2023-10-04'),

(17, 17, 5, 'Blender works like a charm.', '2023-10-05'),

(18, 18, 4, 'Great quality skincare set.', '2023-10-06'),

(19, 19, 3, 'Good but not great dumbbells.', '2023-10-07'),

(20, 20, 4, 'Nice ergonomic chair.', '2023-10-09');

SELECT \* FROM customers;

SELECT \* FROM products;

SELECT \* FROM orders;

SELECT \* FROM order\_items;

SELECT \* FROM product\_reviews;

Question 1: Top 3 Customers by Total Spend

sql

SELECT

c.customer\_id,

CONCAT(c.first\_name, ' ', c.last\_name) AS customer\_name,

SUM(o.total\_amount) AS total\_spend,

COUNT(o.order\_id) AS order\_count

FROM

customers c

JOIN

orders o ON c.customer\_id = o.customer\_id

GROUP BY

c.customer\_id

ORDER BY

total\_spend DESC

LIMIT 3;

Question 2: Average Rating by Product Category (with at least 2 reviews)

sql

SELECT

p.category,

AVG(pr.rating) AS average\_rating

FROM

products p

JOIN

product\_reviews pr ON p.product\_id = pr.product\_id

GROUP BY

p.category

HAVING

COUNT(pr.review\_id) >= 2;

Question 3: Products Never Ordered

sql

SELECT

p.product\_id,

p.product\_name

FROM

products p

LEFT JOIN

order\_items oi ON p.product\_id = oi.product\_id

WHERE

oi.order\_item\_id IS NULL;

Question 4: Time Difference Between Registration and First Order Date for Each Customer

sql

SELECT

c.customer\_id,

CONCAT(c.first\_name, ' ', c.last\_name) AS customer\_name,

DATEDIFF(MIN(o.order\_date), c.registration\_date) AS days\_to\_first\_order

FROM

customers c

LEFT JOIN

orders o ON c.customer\_id = o.customer\_id

GROUP BY

c.customer\_id;

Question 5: Total Revenue for Each Month with Running Total

sql

SELECT

DATE\_FORMAT(order\_date, '%Y-%m') AS month,

SUM(total\_amount) AS monthly\_revenue,

SUM(SUM(total\_amount)) OVER (ORDER BY DATE\_FORMAT(order\_date, '%Y-%m')) AS running\_total

FROM

orders

GROUP BY

month;

Question 6: Customers Who Made a Purchase but Never Left a Review

sql

SELECT

DISTINCT c.customer\_id,

CONCAT(c.first\_name, ' ', c.last\_name) AS customer\_name

FROM

customers c

JOIN

orders o ON c.customer\_id = o.customer\_id

LEFT JOIN

product\_reviews pr ON c.customer\_id = pr.customer\_id

WHERE

pr.review\_id IS NULL;

Question 7: Most Ordered Product by Quantity

sql

SELECT

p.product\_id,

p.product\_name,

SUM(oi.quantity) AS total\_quantity\_ordered

FROM

products p

JOIN

order\_items oi ON p.product\_id = oi.product\_id

GROUP BY

p.product\_id

ORDER BY

total\_quantity\_ordered DESC

LIMIT 1;

Question 8: Percentage of Total Revenue by Product Category

sql

SELECT

p.category,

SUM(o.total\_amount) AS category\_revenue,

(SUM(o.total\_amount) / (SELECT SUM(total\_amount) FROM orders) \* 100) AS percentage\_of\_total\_revenue

FROM

products p

JOIN

order\_items oi ON p.product\_id = oi.product\_id

JOIN

orders o ON oi.order\_id = o.order\_id

GROUP BY

p.category;

Question 9: Most Frequently Purchased Product Category for Each Customer

sql

SELECT

customer\_id,

category,

COUNT(\*) AS purchase\_count,

RANK() OVER (PARTITION BY customer\_id ORDER BY COUNT(\*) DESC) as rank\_order

FROM (

SELECT

o.customer\_id,

p.category

FROM

orders o

JOIN

order\_items oi ON o.order\_id = oi.order\_id

JOIN

products p ON oi.product\_id = p.product\_id

) as subquery

GROUP BY customer\_id, category

HAVING rank\_order = 1;

Question 10: Distribution of Ratings for Each Product

sql

SELECT

product\_id,

rating,

COUNT(\*) AS rating\_count

FROM

product\_reviews

GROUP BY

product\_id, rating

ORDER BY

product\_id, rating;

MySQL Force Index Definition and Goal

* MySQL has a hint called the Force Index whereby developers can indicate clearly what index it should be using when issuing a query. Although MySQL's query optimizer usually makes an intelligent decision regarding which index it should use, there are instances where forcing a particular index advances its utility.
* The main purpose of the Force Index is to manage the query execution plan and ensure that a certain index has been used. This can occur in cases where the optimizer may make suboptimal decisions because of stale statistics or overly complicated conditions with queries.

create database test\_db\_poc;

use test\_db\_poc;

create table if not exists large\_table(

id int auto\_increment primary key,

name varchar(50),

value INT

);

DELIMITER //

CREATE procedure INSERT\_MILLION\_RECORDS()

BEGIN

declare i int default 0;

while i <100000 do

insert into large\_table(name,value) values(concat('Name',i),floor(1 + RAND()\*1000000));

set i=i+1;

end while;

END //

SHOW procedure status WHERE DB ='test\_db\_poc';

select count(\*) from large\_table;

CALL INSERT\_MILLION\_RECORDS();

select sql\_no\_cache sum(value) from large\_table

select sum(value) from large\_table;

create index idx\_value on large\_table(value)

alter table large\_table drop index idx\_value

select \* from large\_table

select a.value

from large\_table a

cross join large\_table b

on a.value=b.value\*100

**TRIGGER**

DELIMITER //

CREATE TRIGGER log\_phone\_changes

BEFORE UPDATE ON customers

FOR EACH ROW

BEGIN

IF old.phone != new.phone THEN

INSERT INTO phone\_changes\_log(customer\_id, old\_phone, new\_phone)

VALUES(old.id, old.phone, new.phone);

END IF;

END//

UPDATE customers

SET phone = '321-654-0987'

WHERE id = 1;

**Overview of Snowflake Schema**

The snowflake schema is a sophisticated data modeling technique used primarily in data warehousing. It is characterized by its multi-dimensional structure, where dimension tables are normalized into multiple related tables, creating a more intricate design than the simpler star schema.

Key Features

- Normalization: Unlike star schemas, where dimension tables are denormalized, snowflake schemas utilize a high degree of normalization. This means that dimension tables are broken down into sub-dimensions to eliminate redundancy and enhance data integrity.

- Hierarchical Structure: The schema resembles a snowflake due to its hierarchical organization. Each parent table can connect to multiple child tables, forming a complex network of relationships.

- Efficient Query Performance: While the normalization leads to more joins when querying data, it also allows for efficient retrieval of data through structured relationships. This can be particularly beneficial in environments with large datasets.

- Granular Data Representation: By breaking down dimension tables into smaller, attribute-focused tables, snowflake schemas provide detailed data analysis capabilities.

Use Cases

Snowflake schemas are particularly useful in industries that require complex data analysis, such as:

- Retail: For analyzing sales across various dimensions like time, product categories, and customer demographics.

- Healthcare: To manage and analyze patient data across various attributes and relationships.

- Manufacturing: For tracking production metrics against multiple factors like supply chain and inventory levels.

Advantages and Disadvantages

Advantages:

- Improved data integrity and quality due to normalization.

- Efficient storage usage by minimizing duplication.

- Enhanced analytical capabilities through detailed relationships.

Disadvantages:

- Increased complexity in database design.

- Potentially slower query performance due to the need for multiple joins.

- Higher maintenance costs associated with managing numerous tables[4][6].

—---------------------------------------------------

SPRING-BOOT

**Primary Key**

The PRIMARY KEY constraint uniquely identifies each record in a table.

**FOREIGN KEY**

The FOREIGN KEY constraint is a key used to link two tables together.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

CREATE TABLE Orders (   
    OrderID int NOT NULL,   
    OrderNumber int NOT NULL,   
    PersonID int,   
    PRIMARY KEY (OrderID),   
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)   
);

**Generalization**

Generalization is the process of extracting common properties from a set of entities and creating a generalized entity from it. It is a bottom-up approach in which two or more entities can be generalized to a higher-level entity if they have some attributes in common.

Generalization is also called as Bottom-up approach.

**Advanced SQL Practice Questions**

create database testdb1;

use testdb1;

CREATE TABLE customers (

customer\_id INT PRIMARY KEY AUTO\_INCREMENT,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

email VARCHAR(100) UNIQUE,

registration\_date DATE

);

CREATE TABLE products (

product\_id INT PRIMARY KEY AUTO\_INCREMENT,

product\_name VARCHAR(100),

category VARCHAR(50),

price DECIMAL(10, 2),

stock\_quantity INT

);

CREATE TABLE orders (

order\_id INT PRIMARY KEY AUTO\_INCREMENT,

customer\_id INT,

order\_date DATETIME,

total\_amount DECIMAL(10, 2),

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

CREATE TABLE order\_items (

order\_item\_id INT PRIMARY KEY AUTO\_INCREMENT,

order\_id INT,

product\_id INT,

quantity INT,

price\_per\_unit DECIMAL(10, 2),

FOREIGN KEY (order\_id) REFERENCES orders(order\_id),

FOREIGN KEY (product\_id) REFERENCES products(product\_id)

);

CREATE TABLE product\_reviews (

review\_id INT PRIMARY KEY AUTO\_INCREMENT,

product\_id INT,

customer\_id INT,

rating INT CHECK (rating BETWEEN 1 AND 5),

review\_text TEXT,

review\_date DATE,

FOREIGN KEY (product\_id) REFERENCES products(product\_id),

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

INSERT INTO customers (first\_name, last\_name, email, registration\_date) VALUES

('John', 'Doe', 'john.doe@example.com', '2023-01-15'),

('Jane', 'Smith', 'jane.smith@example.com', '2023-02-20'),

('Alice', 'Johnson', 'alice.johnson@example.com', '2023-03-05'),

('Bob', 'Williams', 'bob.williams@example.com', '2023-04-10'),

('Charlie', 'Brown', 'charlie.brown@example.com', '2023-05-12'),

('David', 'Jones', 'david.jones@example.com', '2023-06-25'),

('Eva', 'Garcia', 'eva.garcia@example.com', '2023-07-30'),

('Frank', 'Martinez', 'frank.martinez@example.com', '2023-08-15'),

('Grace', 'Davis', 'grace.davis@example.com', '2023-09-01'),

('Henry', 'Miller', 'henry.miller@example.com', '2023-09-10'),

('Isabella', 'Hernandez', 'isabella.hernandez@example.com', '2023-09-15'),

('Jack', 'Moore', 'jack.moore@example.com', '2023-09-20'),

('Kathy', 'Taylor', 'kathy.taylor@example.com', '2023-09-25'),

('Liam', 'Wilson', 'liam.wilson@example.com', '2023-09-30'),

('Mia', 'Anderson', 'mia.anderson@example.com', '2023-10-01'),

('Noah', 'Thomas', 'noah.thomas@example.com', '2023-10-02'),

('Olivia', 'Jackson', 'olivia.jackson@example.com', '2023-10-05'),

('Paul', 'White', 'paul.white@example.com', '2023-10-07'),

('Quinn', 'Harris', 'quinn.harris@example.com', '2023-10-08'),

('Rita', 'Clark', 'rita.clark@example.com', '2023-10-09');

INSERT INTO products (product\_name, category, price, stock\_quantity) VALUES

('Laptop', 'Electronics', 999.99, 50),

('Smartphone', 'Electronics', 599.99, 150),

('Tablet', 'Electronics', 399.99, 60),

('Headphones', 'Electronics', 149.99, 75),

('Monitor', 'Electronics', 299.99, 40),

('Printer', 'Electronics', 199.99, 25),

('Coffee Maker', 'Home Appliances', 89.99, 30),

('Blender', 'Home Appliances', 59.99, 20),

('Desk', 'Furniture', 249.99, 20),

('Chair', 'Furniture', 149.99, 45),

('Bookshelf', 'Furniture', 199.99, 15),

('Notebook', 'Stationery', 4.99, 200),

('Pen', 'Stationery', 1.99, 300),

('Eraser', 'Stationery', 0.99, 150),

('Backpack', 'Accessories', 39.99, 100),

('Camera', 'Electronics', 799.99, 10),

('Smartwatch', 'Electronics', 249.99, 70),

('Wireless Charger', 'Accessories', 29.99, 200),

('Bluetooth Speaker', 'Electronics', 59.99, 100),

('Gaming Console', 'Electronics', 499.99, 30);

INSERT INTO orders (customer\_id, order\_date, total\_amount) VALUES

(1, '2023-01-16 10:00:00', 999.99),

(2, '2023-02-21 11:30:00', 599.99),

(3, '2023-03-06 12:15:00', 399.99),

(4, '2023-04-11 14:00:00', 149.99),

(5, '2023-05-13 09:45:00', 89.99),

(6, '2023-06-26 10:15:00', 59.99),

(7, '2023-07-31 15:30:00', 299.99),

(8, '2023-08-16 10:30:00', 199.99),

(9, '2023-09-02 14:20:00', 249.99),

(10, '2023-09-11 16:45:00', 149.99),

(11, '2023-09-16 09:00:00', 399.99),

(12, '2023-09-21 13:30:00', 249.99),

(13, '2023-09-26 12:05:00', 59.99),

(14, '2023-09-27 14:15:00', 89.99),

(15, '2023-09-29 10:00:00', 19.99),

(16, '2023-10-01 11:20:00', 299.99),

(17, '2023-10-03 12:30:00', 499.99),

(18, '2023-10-04 13:10:00', 149.99),

(19, '2023-10-06 14:00:00', 59.99),

(20, '2023-10-08 15:00:00', 499.99);

INSERT INTO order\_items (order\_id, product\_id, quantity, price\_per\_unit) VALUES

(1, 1, 1, 999.99),

(2, 2, 1, 599.99),

(3, 3, 1, 399.99),

(4, 4, 2, 74.99),

(5, 5, 1, 89.99),

(6, 6, 1, 59.99),

(7, 7, 3, 199.99),

(8, 8, 2, 99.99),

(9, 9, 1, 249.99),

(10, 10, 1, 149.99),

(11, 11, 1, 399.99),

(12, 12, 2, 124.99),

(13, 13, 1, 59.99),

(14, 14, 1, 89.99),

(15, 15, 1, 19.99),

(16, 16, 1, 299.99),

(17, 17, 1, 499.99),

(18, 18, 2, 119.99),

(19, 19, 3, 59.99),

(20, 20, 2, 99.99);

INSERT INTO product\_reviews (product\_id, customer\_id, rating, review\_text, review\_date) VALUES

(1, 1, 5, 'Excellent laptop, very powerful.', '2023-01-17'),

(2, 2, 4, 'Great smartphone, very user-friendly.', '2023-02-22'),

(3, 3, 3, 'Good tablet but a bit pricey.', '2023-03-07'),

(4, 4, 4, 'Coffee maker works well, easy to use.', '2023-04-12'),

(5, 5, 5, 'Headphones have great sound quality.', '2023-05-14'),

(6, 6, 5, 'Printer is fast and reliable.', '2023-06-27'),

(7, 7, 3, 'Desk is sturdy, but hard to assemble.', '2023-07-31'),

(8, 8, 4, 'Chair is comfortable for long hours.', '2023-08-17'),

(9, 9, 2, 'Monitor is bright and clear.', '2023-09-03'),

(10, 10, 4, 'Nice quality Bluetooth speaker.', '2023-09-12'),

(11, 11, 5, 'Smartwatch is very helpful.', '2023-09-17'),

(12, 12, 3, 'Gaming console has great graphics.', '2023-09-22'),

(13, 13, 4, 'Backpack is spacious and stylish.', '2023-09-27'),

(14, 14, 2, 'Camera quality is average.', '2023-09-28'),

(15, 15, 5, 'E-reader is fantastic for books.', '2023-10-02'),

(16, 16, 4, 'Wireless charger is convenient.', '2023-10-04'),

(17, 17, 5, 'Blender works like a charm.', '2023-10-05'),

(18, 18, 4, 'Great quality skincare set.', '2023-10-06'),

(19, 19, 3, 'Good but not great dumbbells.', '2023-10-07'),

(20, 20, 4, 'Nice ergonomic chair.', '2023-10-09');

SELECT \* FROM customers;

SELECT \* FROM products;

SELECT \* FROM orders;

SELECT \* FROM order\_items;

SELECT \* FROM product\_reviews;

Question 1: Top 3 Customers by Total Spend

sql

SELECT

c.customer\_id,

CONCAT(c.first\_name, ' ', c.last\_name) AS customer\_name,

SUM(o.total\_amount) AS total\_spend,

COUNT(o.order\_id) AS order\_count

FROM

customers c

JOIN

orders o ON c.customer\_id = o.customer\_id

GROUP BY

c.customer\_id

ORDER BY

total\_spend DESC

LIMIT 3;

Question 2: Average Rating by Product Category (with at least 2 reviews)

sql

SELECT

p.category,

AVG(pr.rating) AS average\_rating

FROM

products p

JOIN

product\_reviews pr ON p.product\_id = pr.product\_id

GROUP BY

p.category

HAVING

COUNT(pr.review\_id) >= 2;

Question 3: Products Never Ordered

sql

SELECT

p.product\_id,

p.product\_name

FROM

products p

LEFT JOIN

order\_items oi ON p.product\_id = oi.product\_id

WHERE

oi.order\_item\_id IS NULL;

Question 4: Time Difference Between Registration and First Order Date for Each Customer

sql

SELECT

c.customer\_id,

CONCAT(c.first\_name, ' ', c.last\_name) AS customer\_name,

DATEDIFF(MIN(o.order\_date), c.registration\_date) AS days\_to\_first\_order

FROM

customers c

LEFT JOIN

orders o ON c.customer\_id = o.customer\_id

GROUP BY

c.customer\_id;

Question 5: Total Revenue for Each Month with Running Total

sql

SELECT

DATE\_FORMAT(order\_date, '%Y-%m') AS month,

SUM(total\_amount) AS monthly\_revenue,

SUM(SUM(total\_amount)) OVER (ORDER BY DATE\_FORMAT(order\_date, '%Y-%m')) AS running\_total

FROM

orders

GROUP BY

month;

Question 6: Customers Who Made a Purchase but Never Left a Review

sql

SELECT

DISTINCT c.customer\_id,

CONCAT(c.first\_name, ' ', c.last\_name) AS customer\_name

FROM

customers c

JOIN

orders o ON c.customer\_id = o.customer\_id

LEFT JOIN

product\_reviews pr ON c.customer\_id = pr.customer\_id

WHERE

pr.review\_id IS NULL;

Question 7: Most Ordered Product by Quantity

sql

SELECT

p.product\_id,

p.product\_name,

SUM(oi.quantity) AS total\_quantity\_ordered

FROM

products p

JOIN

order\_items oi ON p.product\_id = oi.product\_id

GROUP BY

p.product\_id

ORDER BY

total\_quantity\_ordered DESC

LIMIT 1;

Question 8: Percentage of Total Revenue by Product Category

sql

SELECT

p.category,

SUM(o.total\_amount) AS category\_revenue,

(SUM(o.total\_amount) / (SELECT SUM(total\_amount) FROM orders) \* 100) AS percentage\_of\_total\_revenue

FROM

products p

JOIN

order\_items oi ON p.product\_id = oi.product\_id

JOIN

orders o ON oi.order\_id = o.order\_id

GROUP BY

p.category;

Question 9: Most Frequently Purchased Product Category for Each Customer

sql

SELECT

customer\_id,

category,

COUNT(\*) AS purchase\_count,

RANK() OVER (PARTITION BY customer\_id ORDER BY COUNT(\*) DESC) as rank\_order

FROM (

SELECT

o.customer\_id,

p.category

FROM

orders o

JOIN

order\_items oi ON o.order\_id = oi.order\_id

JOIN

products p ON oi.product\_id = p.product\_id

) as subquery

GROUP BY customer\_id, category

HAVING rank\_order = 1;

Question 10: Distribution of Ratings for Each Product

sql

SELECT

product\_id,

rating,

COUNT(\*) AS rating\_count

FROM

product\_reviews

GROUP BY

product\_id, rating

ORDER BY

product\_id, rating;

MySQL Force Index Definition and Goal

* MySQL has a hint called the Force Index whereby developers can indicate clearly what index it should be using when issuing a query. Although MySQL's query optimizer usually makes an intelligent decision regarding which index it should use, there are instances where forcing a particular index advances its utility.
* The main purpose of the Force Index is to manage the query execution plan and ensure that a certain index has been used. This can occur in cases where the optimizer may make suboptimal decisions because of stale statistics or overly complicated conditions with queries.

create database test\_db\_poc;

use test\_db\_poc;

create table if not exists large\_table(

id int auto\_increment primary key,

name varchar(50),

value INT

);

DELIMITER //

CREATE procedure INSERT\_MILLION\_RECORDS()

BEGIN

declare i int default 0;

while i <100000 do

insert into large\_table(name,value) values(concat('Name',i),floor(1 + RAND()\*1000000));

set i=i+1;

end while;

END //

SHOW procedure status WHERE DB ='test\_db\_poc';

select count(\*) from large\_table;

CALL INSERT\_MILLION\_RECORDS();

select sql\_no\_cache sum(value) from large\_table

select sum(value) from large\_table;

create index idx\_value on large\_table(value)

alter table large\_table drop index idx\_value

select \* from large\_table

select a.value

from large\_table a

cross join large\_table b

on a.value=b.value\*100

**TRIGGER**

DELIMITER //

CREATE TRIGGER log\_phone\_changes

BEFORE UPDATE ON customers

FOR EACH ROW

BEGIN

IF old.phone != new.phone THEN

INSERT INTO phone\_changes\_log(customer\_id, old\_phone, new\_phone)

VALUES(old.id, old.phone, new.phone);

END IF;

END//

UPDATE customers

SET phone = '321-654-0987'

WHERE id = 1;

**Overview of Snowflake Schema**

The snowflake schema is a sophisticated data modeling technique used primarily in data warehousing. It is characterized by its multi-dimensional structure, where dimension tables are normalized into multiple related tables, creating a more intricate design than the simpler star schema.

Key Features

- Normalization: Unlike star schemas, where dimension tables are denormalized, snowflake schemas utilize a high degree of normalization. This means that dimension tables are broken down into sub-dimensions to eliminate redundancy and enhance data integrity.

- Hierarchical Structure: The schema resembles a snowflake due to its hierarchical organization. Each parent table can connect to multiple child tables, forming a complex network of relationships.

- Efficient Query Performance: While the normalization leads to more joins when querying data, it also allows for efficient retrieval of data through structured relationships. This can be particularly beneficial in environments with large datasets.

- Granular Data Representation: By breaking down dimension tables into smaller, attribute-focused tables, snowflake schemas provide detailed data analysis capabilities.

Use Cases

Snowflake schemas are particularly useful in industries that require complex data analysis, such as:

- Retail: For analyzing sales across various dimensions like time, product categories, and customer demographics.

- Healthcare: To manage and analyze patient data across various attributes and relationships.

- Manufacturing: For tracking production metrics against multiple factors like supply chain and inventory levels.

Advantages and Disadvantages

Advantages:

- Improved data integrity and quality due to normalization.

- Efficient storage usage by minimizing duplication.

- Enhanced analytical capabilities through detailed relationships.

Disadvantages:

- Increased complexity in database design.

- Potentially slower query performance due to the need for multiple joins.

- Higher maintenance costs associated with managing numerous tables[4][6].

—----------------------------------------------------

ANGULAR

**Primary Key**

The PRIMARY KEY constraint uniquely identifies each record in a table.

**FOREIGN KEY**

The FOREIGN KEY constraint is a key used to link two tables together.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

CREATE TABLE Orders (   
    OrderID int NOT NULL,   
    OrderNumber int NOT NULL,   
    PersonID int,   
    PRIMARY KEY (OrderID),   
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)   
);

**Generalization**

Generalization is the process of extracting common properties from a set of entities and creating a generalized entity from it. It is a bottom-up approach in which two or more entities can be generalized to a higher-level entity if they have some attributes in common.

Generalization is also called as Bottom-up approach.

**Advanced SQL Practice Questions**

create database testdb1;

use testdb1;

CREATE TABLE customers (

customer\_id INT PRIMARY KEY AUTO\_INCREMENT,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

email VARCHAR(100) UNIQUE,

registration\_date DATE

);

CREATE TABLE products (

product\_id INT PRIMARY KEY AUTO\_INCREMENT,

product\_name VARCHAR(100),

category VARCHAR(50),

price DECIMAL(10, 2),

stock\_quantity INT

);

CREATE TABLE orders (

order\_id INT PRIMARY KEY AUTO\_INCREMENT,

customer\_id INT,

order\_date DATETIME,

total\_amount DECIMAL(10, 2),

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

CREATE TABLE order\_items (

order\_item\_id INT PRIMARY KEY AUTO\_INCREMENT,

order\_id INT,

product\_id INT,

quantity INT,

price\_per\_unit DECIMAL(10, 2),

FOREIGN KEY (order\_id) REFERENCES orders(order\_id),

FOREIGN KEY (product\_id) REFERENCES products(product\_id)

);

CREATE TABLE product\_reviews (

review\_id INT PRIMARY KEY AUTO\_INCREMENT,

product\_id INT,

customer\_id INT,

rating INT CHECK (rating BETWEEN 1 AND 5),

review\_text TEXT,

review\_date DATE,

FOREIGN KEY (product\_id) REFERENCES products(product\_id),

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

INSERT INTO customers (first\_name, last\_name, email, registration\_date) VALUES

('John', 'Doe', 'john.doe@example.com', '2023-01-15'),

('Jane', 'Smith', 'jane.smith@example.com', '2023-02-20'),

('Alice', 'Johnson', 'alice.johnson@example.com', '2023-03-05'),

('Bob', 'Williams', 'bob.williams@example.com', '2023-04-10'),

('Charlie', 'Brown', 'charlie.brown@example.com', '2023-05-12'),

('David', 'Jones', 'david.jones@example.com', '2023-06-25'),

('Eva', 'Garcia', 'eva.garcia@example.com', '2023-07-30'),

('Frank', 'Martinez', 'frank.martinez@example.com', '2023-08-15'),

('Grace', 'Davis', 'grace.davis@example.com', '2023-09-01'),

('Henry', 'Miller', 'henry.miller@example.com', '2023-09-10'),

('Isabella', 'Hernandez', 'isabella.hernandez@example.com', '2023-09-15'),

('Jack', 'Moore', 'jack.moore@example.com', '2023-09-20'),

('Kathy', 'Taylor', 'kathy.taylor@example.com', '2023-09-25'),

('Liam', 'Wilson', 'liam.wilson@example.com', '2023-09-30'),

('Mia', 'Anderson', 'mia.anderson@example.com', '2023-10-01'),

('Noah', 'Thomas', 'noah.thomas@example.com', '2023-10-02'),

('Olivia', 'Jackson', 'olivia.jackson@example.com', '2023-10-05'),

('Paul', 'White', 'paul.white@example.com', '2023-10-07'),

('Quinn', 'Harris', 'quinn.harris@example.com', '2023-10-08'),

('Rita', 'Clark', 'rita.clark@example.com', '2023-10-09');

INSERT INTO products (product\_name, category, price, stock\_quantity) VALUES

('Laptop', 'Electronics', 999.99, 50),

('Smartphone', 'Electronics', 599.99, 150),

('Tablet', 'Electronics', 399.99, 60),

('Headphones', 'Electronics', 149.99, 75),

('Monitor', 'Electronics', 299.99, 40),

('Printer', 'Electronics', 199.99, 25),

('Coffee Maker', 'Home Appliances', 89.99, 30),

('Blender', 'Home Appliances', 59.99, 20),

('Desk', 'Furniture', 249.99, 20),

('Chair', 'Furniture', 149.99, 45),

('Bookshelf', 'Furniture', 199.99, 15),

('Notebook', 'Stationery', 4.99, 200),

('Pen', 'Stationery', 1.99, 300),

('Eraser', 'Stationery', 0.99, 150),

('Backpack', 'Accessories', 39.99, 100),

('Camera', 'Electronics', 799.99, 10),

('Smartwatch', 'Electronics', 249.99, 70),

('Wireless Charger', 'Accessories', 29.99, 200),

('Bluetooth Speaker', 'Electronics', 59.99, 100),

('Gaming Console', 'Electronics', 499.99, 30);

INSERT INTO orders (customer\_id, order\_date, total\_amount) VALUES

(1, '2023-01-16 10:00:00', 999.99),

(2, '2023-02-21 11:30:00', 599.99),

(3, '2023-03-06 12:15:00', 399.99),

(4, '2023-04-11 14:00:00', 149.99),

(5, '2023-05-13 09:45:00', 89.99),

(6, '2023-06-26 10:15:00', 59.99),

(7, '2023-07-31 15:30:00', 299.99),

(8, '2023-08-16 10:30:00', 199.99),

(9, '2023-09-02 14:20:00', 249.99),

(10, '2023-09-11 16:45:00', 149.99),

(11, '2023-09-16 09:00:00', 399.99),

(12, '2023-09-21 13:30:00', 249.99),

(13, '2023-09-26 12:05:00', 59.99),

(14, '2023-09-27 14:15:00', 89.99),

(15, '2023-09-29 10:00:00', 19.99),

(16, '2023-10-01 11:20:00', 299.99),

(17, '2023-10-03 12:30:00', 499.99),

(18, '2023-10-04 13:10:00', 149.99),

(19, '2023-10-06 14:00:00', 59.99),

(20, '2023-10-08 15:00:00', 499.99);

INSERT INTO order\_items (order\_id, product\_id, quantity, price\_per\_unit) VALUES

(1, 1, 1, 999.99),

(2, 2, 1, 599.99),

(3, 3, 1, 399.99),

(4, 4, 2, 74.99),

(5, 5, 1, 89.99),

(6, 6, 1, 59.99),

(7, 7, 3, 199.99),

(8, 8, 2, 99.99),

(9, 9, 1, 249.99),

(10, 10, 1, 149.99),

(11, 11, 1, 399.99),

(12, 12, 2, 124.99),

(13, 13, 1, 59.99),

(14, 14, 1, 89.99),

(15, 15, 1, 19.99),

(16, 16, 1, 299.99),

(17, 17, 1, 499.99),

(18, 18, 2, 119.99),

(19, 19, 3, 59.99),

(20, 20, 2, 99.99);

INSERT INTO product\_reviews (product\_id, customer\_id, rating, review\_text, review\_date) VALUES

(1, 1, 5, 'Excellent laptop, very powerful.', '2023-01-17'),

(2, 2, 4, 'Great smartphone, very user-friendly.', '2023-02-22'),

(3, 3, 3, 'Good tablet but a bit pricey.', '2023-03-07'),

(4, 4, 4, 'Coffee maker works well, easy to use.', '2023-04-12'),

(5, 5, 5, 'Headphones have great sound quality.', '2023-05-14'),

(6, 6, 5, 'Printer is fast and reliable.', '2023-06-27'),

(7, 7, 3, 'Desk is sturdy, but hard to assemble.', '2023-07-31'),

(8, 8, 4, 'Chair is comfortable for long hours.', '2023-08-17'),

(9, 9, 2, 'Monitor is bright and clear.', '2023-09-03'),

(10, 10, 4, 'Nice quality Bluetooth speaker.', '2023-09-12'),

(11, 11, 5, 'Smartwatch is very helpful.', '2023-09-17'),

(12, 12, 3, 'Gaming console has great graphics.', '2023-09-22'),

(13, 13, 4, 'Backpack is spacious and stylish.', '2023-09-27'),

(14, 14, 2, 'Camera quality is average.', '2023-09-28'),

(15, 15, 5, 'E-reader is fantastic for books.', '2023-10-02'),

(16, 16, 4, 'Wireless charger is convenient.', '2023-10-04'),

(17, 17, 5, 'Blender works like a charm.', '2023-10-05'),

(18, 18, 4, 'Great quality skincare set.', '2023-10-06'),

(19, 19, 3, 'Good but not great dumbbells.', '2023-10-07'),

(20, 20, 4, 'Nice ergonomic chair.', '2023-10-09');

SELECT \* FROM customers;

SELECT \* FROM products;

SELECT \* FROM orders;

SELECT \* FROM order\_items;

SELECT \* FROM product\_reviews;

Question 1: Top 3 Customers by Total Spend

sql

SELECT

c.customer\_id,

CONCAT(c.first\_name, ' ', c.last\_name) AS customer\_name,

SUM(o.total\_amount) AS total\_spend,

COUNT(o.order\_id) AS order\_count

FROM

customers c

JOIN

orders o ON c.customer\_id = o.customer\_id

GROUP BY

c.customer\_id

ORDER BY

total\_spend DESC

LIMIT 3;

Question 2: Average Rating by Product Category (with at least 2 reviews)

sql

SELECT

p.category,

AVG(pr.rating) AS average\_rating

FROM

products p

JOIN

product\_reviews pr ON p.product\_id = pr.product\_id

GROUP BY

p.category

HAVING

COUNT(pr.review\_id) >= 2;

Question 3: Products Never Ordered

sql

SELECT

p.product\_id,

p.product\_name

FROM

products p

LEFT JOIN

order\_items oi ON p.product\_id = oi.product\_id

WHERE

oi.order\_item\_id IS NULL;

Question 4: Time Difference Between Registration and First Order Date for Each Customer

sql

SELECT

c.customer\_id,

CONCAT(c.first\_name, ' ', c.last\_name) AS customer\_name,

DATEDIFF(MIN(o.order\_date), c.registration\_date) AS days\_to\_first\_order

FROM

customers c

LEFT JOIN

orders o ON c.customer\_id = o.customer\_id

GROUP BY

c.customer\_id;

Question 5: Total Revenue for Each Month with Running Total

sql

SELECT

DATE\_FORMAT(order\_date, '%Y-%m') AS month,

SUM(total\_amount) AS monthly\_revenue,

SUM(SUM(total\_amount)) OVER (ORDER BY DATE\_FORMAT(order\_date, '%Y-%m')) AS running\_total

FROM

orders

GROUP BY

month;

Question 6: Customers Who Made a Purchase but Never Left a Review

sql

SELECT

DISTINCT c.customer\_id,

CONCAT(c.first\_name, ' ', c.last\_name) AS customer\_name

FROM

customers c

JOIN

orders o ON c.customer\_id = o.customer\_id

LEFT JOIN

product\_reviews pr ON c.customer\_id = pr.customer\_id

WHERE

pr.review\_id IS NULL;

Question 7: Most Ordered Product by Quantity

sql

SELECT

p.product\_id,

p.product\_name,

SUM(oi.quantity) AS total\_quantity\_ordered

FROM

products p

JOIN

order\_items oi ON p.product\_id = oi.product\_id

GROUP BY

p.product\_id

ORDER BY

total\_quantity\_ordered DESC

LIMIT 1;

Question 8: Percentage of Total Revenue by Product Category

sql

SELECT

p.category,

SUM(o.total\_amount) AS category\_revenue,

(SUM(o.total\_amount) / (SELECT SUM(total\_amount) FROM orders) \* 100) AS percentage\_of\_total\_revenue

FROM

products p

JOIN

order\_items oi ON p.product\_id = oi.product\_id

JOIN

orders o ON oi.order\_id = o.order\_id

GROUP BY

p.category;

Question 9: Most Frequently Purchased Product Category for Each Customer

sql

SELECT

customer\_id,

category,

COUNT(\*) AS purchase\_count,

RANK() OVER (PARTITION BY customer\_id ORDER BY COUNT(\*) DESC) as rank\_order

FROM (

SELECT

o.customer\_id,

p.category

FROM

orders o

JOIN

order\_items oi ON o.order\_id = oi.order\_id

JOIN

products p ON oi.product\_id = p.product\_id

) as subquery

GROUP BY customer\_id, category

HAVING rank\_order = 1;

Question 10: Distribution of Ratings for Each Product

sql

SELECT

product\_id,

rating,

COUNT(\*) AS rating\_count

FROM

product\_reviews

GROUP BY

product\_id, rating

ORDER BY

product\_id, rating;

MySQL Force Index Definition and Goal

* MySQL has a hint called the Force Index whereby developers can indicate clearly what index it should be using when issuing a query. Although MySQL's query optimizer usually makes an intelligent decision regarding which index it should use, there are instances where forcing a particular index advances its utility.
* The main purpose of the Force Index is to manage the query execution plan and ensure that a certain index has been used. This can occur in cases where the optimizer may make suboptimal decisions because of stale statistics or overly complicated conditions with queries.

create database test\_db\_poc;

use test\_db\_poc;

create table if not exists large\_table(

id int auto\_increment primary key,

name varchar(50),

value INT

);

DELIMITER //

CREATE procedure INSERT\_MILLION\_RECORDS()

BEGIN

declare i int default 0;

while i <100000 do

insert into large\_table(name,value) values(concat('Name',i),floor(1 + RAND()\*1000000));

set i=i+1;

end while;

END //

SHOW procedure status WHERE DB ='test\_db\_poc';

select count(\*) from large\_table;

CALL INSERT\_MILLION\_RECORDS();

select sql\_no\_cache sum(value) from large\_table

select sum(value) from large\_table;

create index idx\_value on large\_table(value)

alter table large\_table drop index idx\_value

select \* from large\_table

select a.value

from large\_table a

cross join large\_table b

on a.value=b.value\*100

**TRIGGER**

DELIMITER //

CREATE TRIGGER log\_phone\_changes

BEFORE UPDATE ON customers

FOR EACH ROW

BEGIN

IF old.phone != new.phone THEN

INSERT INTO phone\_changes\_log(customer\_id, old\_phone, new\_phone)

VALUES(old.id, old.phone, new.phone);

END IF;

END//

UPDATE customers

SET phone = '321-654-0987'

WHERE id = 1;

**Overview of Snowflake Schema**

The snowflake schema is a sophisticated data modeling technique used primarily in data warehousing. It is characterized by its multi-dimensional structure, where dimension tables are normalized into multiple related tables, creating a more intricate design than the simpler star schema.

Key Features

- Normalization: Unlike star schemas, where dimension tables are denormalized, snowflake schemas utilize a high degree of normalization. This means that dimension tables are broken down into sub-dimensions to eliminate redundancy and enhance data integrity.

- Hierarchical Structure: The schema resembles a snowflake due to its hierarchical organization. Each parent table can connect to multiple child tables, forming a complex network of relationships.

- Efficient Query Performance: While the normalization leads to more joins when querying data, it also allows for efficient retrieval of data through structured relationships. This can be particularly beneficial in environments with large datasets.

- Granular Data Representation: By breaking down dimension tables into smaller, attribute-focused tables, snowflake schemas provide detailed data analysis capabilities.

Use Cases

Snowflake schemas are particularly useful in industries that require complex data analysis, such as:

- Retail: For analyzing sales across various dimensions like time, product categories, and customer demographics.

- Healthcare: To manage and analyze patient data across various attributes and relationships.

- Manufacturing: For tracking production metrics against multiple factors like supply chain and inventory levels.

Advantages and Disadvantages

Advantages:

- Improved data integrity and quality due to normalization.

- Efficient storage usage by minimizing duplication.

- Enhanced analytical capabilities through detailed relationships.

Disadvantages:

- Increased complexity in database design.

- Potentially slower query performance due to the need for multiple joins.

- Higher maintenance costs associated with managing numerous tables[4][6].

—-----------------------------------------------------------

PROJECT-CAR TIRE SERVICE(SPRING,ANGULAR,SQL)

The project is designed to allow users to book car service appointments and view services offered, while admins manage services, bookings, and customer data. This setup includes:

1. **Front End (Angular)** for the user interface.
2. **Back End (Spring Boot)** for handling business logic and database interactions.
3. **Database (MySQL)** to store service, booking, and user data.

### Key Components

#### 1. Home Page (User Interface)

* **Purpose**: The home page provides an overview of services, offers a navigation bar, and may include sections like service categories, special offers, and customer testimonials.
* **Features**:
  + **Navigation Bar**: Links to the main sections, such as "Our Tire Services," "Book a Service," "Contact Us," etc.
  + **Service Overview**: Displays car services offered (e.g., tire change, engine oil check, alignment).
  + **Booking Section**: An easy way for users to book a service or check availability.
  + **Contact Information**: Footer with contact details and social media links.

#### 2. Admin Dashboard

* **Purpose**: The admin dashboard is where the admin can manage services, view bookings, and handle customer inquiries.
* **Features**:
  + **Manage Services**: Allows adding, updating, or deleting services. An admin can enable or disable services based on availability.
  + **Manage Bookings**: Shows current and past bookings with options to confirm or cancel bookings.
  + **Customer Management**: A list of registered customers, with basic information and contact details.
  + **Analytics Section**: For example, a dashboard with charts displaying stats like the number of bookings or service popularity.

### Database Tables

To structure the application, the following tables are suggested in MySQL:

1. **Users Table**Stores user (customer and admin) details.
   * **Columns**:
     + user\_id (Primary Key)
     + username
     + password
     + email
     + role (e.g., Admin, Customer)
     + created\_at
2. **Services Table**Contains information about the various car services.
   * **Columns**:
     + service\_id (Primary Key)
     + name (e.g., Tire Replacement)
     + description
     + price
     + status (e.g., Available/Unavailable)
3. **Bookings Table**Manages booking records for each service.
   * **Columns**:
     + booking\_id (Primary Key)
     + user\_id (Foreign Key linking to Users)
     + service\_id (Foreign Key linking to Services)
     + booking\_date
     + status (e.g., Confirmed, Pending, Completed)
     + total\_cost
4. **Payments Table**Records payment information for completed bookings.
   * **Columns**:
     + payment\_id (Primary Key)
     + booking\_id (Foreign Key linking to Bookings)
     + amount
     + payment\_date
     + status (e.g., Paid, Unpaid)
5. **Reviews Table** (optional)  
   Allows users to leave feedback about services.
   * **Columns**:
     + review\_id (Primary Key)
     + user\_id (Foreign Key linking to Users)
     + service\_id (Foreign Key linking to Services)
     + rating (e.g., out of 5 stars)
     + comment

### Backend (Spring Boot) Setup

In the backend:

* **Controllers**: Handle incoming requests for services and bookings, e.g., ServiceController, BookingController.
* **Services**: Business logic to process requests, calculate totals, and manage status updates.
* **Repositories**: Interfaces that allow CRUD operations on the database tables.

Security can be added using **JWT (JSON Web Token)** for secure login and role-based access.

### Frontend (Angular) Setup

In Angular:

* **Components**:
  + Home Component (to show the main page)
  + Service Component (to list all services)
  + Booking Component (to manage bookings)
  + Admin Components (e.g., AdminDashboard, ManageServices)
* **Services**: Make HTTP calls to the Spring Boot backend for fetching data or performing actions (like creating a booking or managing a service).
* **Routing**: Define routes for each component (e.g., /home, /bookings, /admin/dashboard).