**WALCHAND COLLEGE OF ENGINEERING, SANGLI**



Department of Information Technology

IT PRACTICES LAB1 (6IT372)

Academic Year: 2023-24

Term: Semester-2

Class: T.Y. I.T. Batch : T4

Name: PRN.No:

Penshanwar Omkar Ram 21610018

**Certificate**



This is to certify that

Mr. Penshanwar Omkar Ram (21610018)

Of

T.Y. B.Tech (IT) class has completed experiments satisfactorily in IT PRACTICES LAB1 (6IT372)during the

Year 2023-24

Dr. R. R. Rathod Dr. R. R. Rathod

COURSE TEACHER HOD

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. no.** | **Experiment no.** | **Experiment Name** | **Page No.** |
| 1. | 2 | Implementation of Bloom join |  |
| 2. | 4 | implementation of Cube operator in OLAP queries in data Warehousing |  |
| 3. | 5 | Implementation of ADT concept |  |
| 4. | 7 | Implementation of Path expression for XML data |  |
| 5. | 8 | Implementation of Quad tree indexing |  |
| 6. | 10 | Implementation of data log queries for deductive databases |  |

**INDEX**

**Code :**

#include <iostream>

#include <vector>

#include <sstream>

#include <bitset>

#include <unordered\_map>

using namespace std;

vector<string> bloomJoin(vector<string> &studentTable, vector<string> &courseTable)

{

    // Create bitsets for student and course IDs

    bitset<10> studentBits;

    bitset<10> courseBits;

    // Set bits for student IDs in the student bitset

    for (auto &row : studentTable)

    {

        stringstream ss(row);

        string studentId;

        getline(ss, studentId, ',');

        int id = stoi(studentId);

        studentBits.set(id);

    }

    // Set bits for course IDs in the course bitset

    for (auto &row : courseTable)

    {

        stringstream ss(row);

        string studentId;

        getline(ss, studentId, ',');

        int id = stoi(studentId);

        courseBits.set(id);

    }

    // Perform join

    vector<string> result;

    for (auto &row : studentTable)

    {

        stringstream ss(row);

        string studentId, name, age;

        getline(ss, studentId, ',');

        getline(ss, name, ',');

        getline(ss, age, ',');

        int id = stoi(studentId);

        if (studentBits.test(id) && courseBits.test(id))

        {

            for (auto &courseRow : courseTable)

            {

                stringstream css(courseRow);

                string courseId, courseName;

                getline(css, courseId, ',');

                if (id == stoi(courseId))

                {

                    getline(css, courseName, ',');

                    result.push\_back(studentId + "," + name + "," + age + "," + courseName);

                    break;

                }

            }

        }

    }

    return result;

}

int main()

{

    vector<string> studentTable = {

        "1, Vijay, 20",

        "2, Ajay, 21",

        "3, Aman, 22",

        "4, Dilip, 23"};

    vector<string> courseTable = {

        "1, Math",

        "2, Science",

        "3, History",

        "4, English"};

    // Perform Bloom join

    vector<string> result = bloomJoin(studentTable, courseTable);

    // Print result

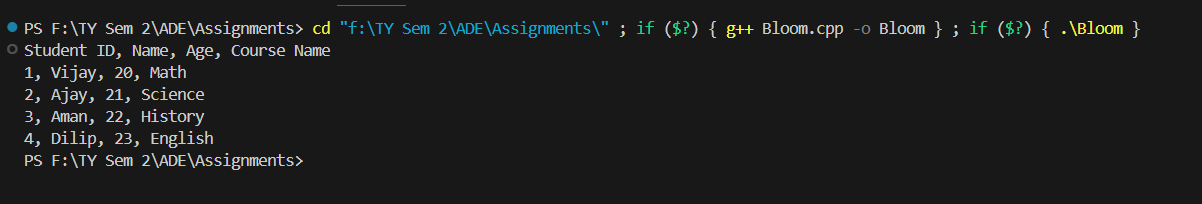
    cout << "Student ID, Name, Age, Course Name" << endl;

    for (auto &row : result)

        cout << row << endl;

    return 0;

}

**Output :**

**Code :**

CREATE DATABASE assign4;

SELECT \* FROM information\_schema.tables WHERE table\_type = 'BASE TABLE';

USE assign4;

CREATE TABLE Employees (

id INT PRIMARY KEY IDENTITY(1,1),

name VARCHAR(50) NOT NULL,

gender CHAR(1) NOT NULL,

salary DECIMAL(10,2) NOT NULL,

department VARCHAR(20) NOT NULL

);

INSERT INTO Employees (name, gender, salary, department) VALUES

('John Doe', 'M', 85000.00, 'Sales'),

('Jane Smith', 'F', 72000.50, 'Marketing'),

('Alice Jones', 'F', 68500.75, 'Engineering'),

('David Lee', 'M', 90000.00, 'Management'),

('Michael Brown', 'M', 75000.00, 'Sales'),

('Emily Davis', 'F', 68000.25, 'Marketing'),

('Robert Johnson', 'M', 80000.00, 'Engineering'),

('Sarah Wilson', 'F', 71000.75, 'Management'),

('Daniel Martinez', 'M', 82000.00, 'Sales'),

('Olivia Thomas', 'F', 70000.50, 'Marketing'),

('Matthew White', 'M', 77000.25, 'Engineering'),

('Sophia Anderson', 'F', 69000.75, 'Management'),

('William Harris', 'M', 84000.00, 'Sales'),

('Isabella Lewis', 'F', 73000.50, 'Marketing'),

('James Robinson', 'M', 78000.25, 'Engineering'),

('Amelia Garcia', 'F', 72000.75, 'Management'),

('Ethan Jackson', 'M', 86000.00, 'Sales'),

('Ava Martinez', 'F', 74000.50, 'Marketing'),

('Benjamin Thompson', 'M', 79000.25, 'Engineering'),

('Mia Davis', 'F', 73000.75, 'Management'),

('Alexander Wilson', 'M', 88000.00, 'Sales'),

('Charlotte Thomas', 'F', 75000.50, 'Marketing'),

('Jacob Brown', 'M', 80000.25, 'Engineering'),

('Harper Johnson', 'F', 74000.75, 'Management'),

('Liam Martinez', 'M', 90000.00, 'Sales'),

('Grace Anderson', 'F', 76000.50, 'Marketing'),

('Daniel Harris', 'M', 81000.25, 'Engineering'),

('Ella Thompson', 'F', 75000.75, 'Management'),

('Logan Garcia', 'M', 87000.00, 'Sales'),

('Avery Wilson', 'F', 77000.50, 'Marketing'),

('Jackson Thomas', 'M', 82000.25, 'Engineering'),

('Sofia Lee', 'F', 76000.75, 'Management'),

('Mason Martinez', 'M', 89000.00, 'Sales'),

('Chloe Brown', 'F', 78000.50, 'Marketing'),

('Lucas Johnson', 'M', 83000.25, 'Engineering'),

('Zoe Wilson', 'F', 78000.75, 'Management'),

('Liam Harris', 'M', 91000.00, 'Sales'),

('Emma Thompson', 'F', 79000.50, 'Marketing'),

('Noah Garcia', 'M', 84000.25, 'Engineering'),

('Aria Davis', 'F', 79000.75, 'Management'),

('James Wilson', 'M', 92000.00, 'Sales'),

('Ava Johnson', 'F', 80000.50, 'Marketing'),

('Oliver Martinez', 'M', 85000.25, 'Engineering'),

('Ella Brown', 'F', 80000.75, 'Management'),

('William Lee', 'M', 93000.00, 'Sales'),

('Isabella Thompson', 'F', 81000.50, 'Marketing');

SELECT coalesce(department,'Total Sum'), SUM(salary) as Total FROM Employees

GROUP BY department WITH ROLLUP;

SELECT coalesce(department,'Total Sum'),coalesce(gender,'Sum'), SUM(salary) as Total

FROM Employees

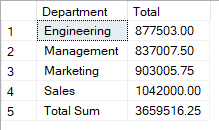
GROUP BY department,gender WITH ROLLUP;

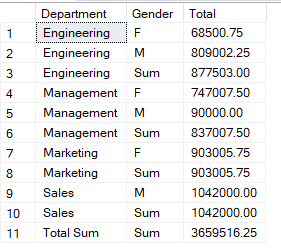
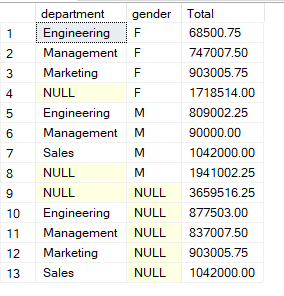
SELECT department, gender, SUM(salary) AS Total

FROM Employees

GROUP BY CUBE(department, gender);

**Output :**

****

****

**ROLL-UP ON 1 COL**

**CUBE- operator**

**ROLL-UP ON 2 COLS**

**Code :**

SELECT datname FROM pg\_database;

SELECT current\_database();

SELECT table\_name

FROM information\_schema.tables

WHERE table\_schema = 'public'; -- Replace 'public' with your schema if needed

CREATE TYPE pan\_number AS (

first\_five\_chars CHAR(5),

next\_four\_digits INTEGER,

last\_char CHAR(1)

);

CREATE TABLE employees (

empid SERIAL PRIMARY KEY,

name VARCHAR(255),

age INTEGER,

photo BYTEA,

pan\_num\_data pan\_number

);

INSERT INTO employees (name, age, photo, pan\_num\_data)

VALUES

('Rajesh Khanna', 30, E'\\xDEADBEEF', ROW('ABCDE', 1234, 'F')),

('Amitabh Bacchan', 32, E'\\xDEACBEEE', ROW('FYXTA', 6542, 'C')),

('Dilip Kumar', 35, E'\\xDEAFBEEE', ROW('QWERT', 9876, 'D')),

('Mohanlal', 40, E'\\xDEAEBAEE', ROW('ZXCVB', 4567, 'G'));

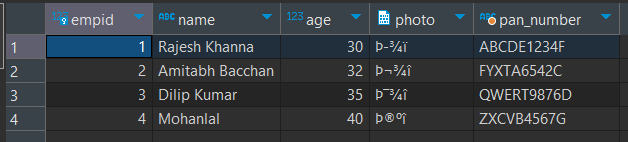
SELECT

empid, name, age, photo,

(pan\_num\_data).first\_five\_chars || (pan\_num\_data).next\_four\_digits || (pan\_num\_data).last\_char AS pan\_number

FROM employees;

**Output :**

****

**Code :**

from lxml import etree

tree = etree.parse('f:/TY Sem 2/ADE/Assignments/POSTMSE/Path Expression Queries/books.xml')

root = tree.getroot()

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Books Titles \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

titles = root.xpath('//book/title/text()')

print("Titles:")

for title in titles:

    print(title)

print("\n\n")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* All Authors \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

authors = root.xpath('//book/author/text()')

for author in authors:

    print(author)

print("\n\n")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Books by Genre with author \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

for book in root.xpath('//book'):

    title = book.xpath('title/text()')[0]

    author = book.xpath('author/text()')[0]

    genre = book.xpath('genre/text()')[0]

    print(f"Book: {title} - {author} - {genre}")

print("\n\n")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Books by Genre with title \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

for genre in set(root.xpath('//book/genre/text()')):

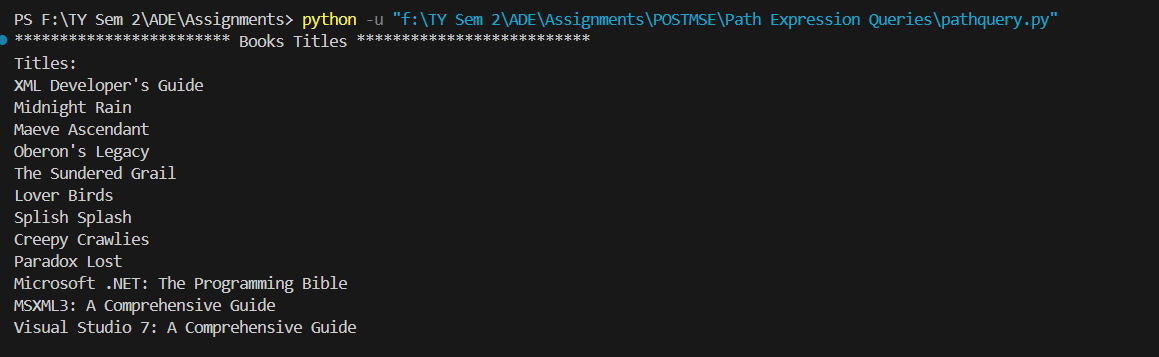
    books\_in\_genre = root.xpath(f"//book[genre='{genre}']")

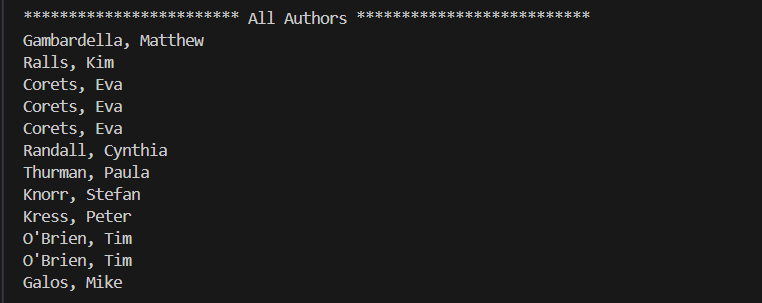
    print(f"Books in {genre}:")

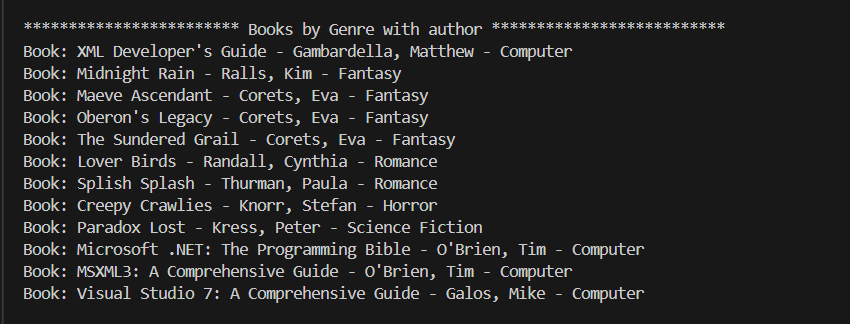
    for book in books\_in\_genre:

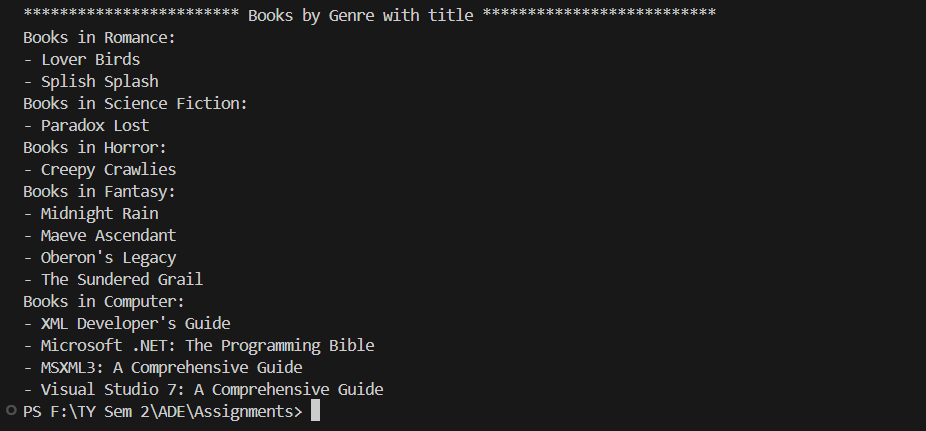
        print(f"- {book.find('title').text}")

**Output :**









**Code :**

import random

import matplotlib.pyplot as plt

import matplotlib.patches as patches

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

class Node:

def \_\_init\_\_(self, x0, y0, w, h, points):

self.x0 = x0

self.y0 = y0

self.width = w

self.height = h

self.points = points

self.children = []

def get\_width(self):

return self.width

def get\_height(self):

return self.height

def get\_points(self):

return self.points

class QTree:

def \_\_init\_\_(self, k, n):

self.threshold = k

self.points = [Point(random.uniform(0, 10), random.uniform(0, 10)) for \_ in range(n)]

self.root = Node(0, 0, 10, 10, self.points)

def add\_point(self, x, y):

self.points.append(Point(x, y))

def get\_points(self):

return self.points

def subdivide(self):

self.\_recursive\_subdivide(self.root, self.threshold)

def visualize(self):

self.\_visualize\_recursive(self.root, 0)

def \_recursive\_subdivide(self, node, k):

if len(node.points) <= k:

return

w\_ = float(node.width / 2)

h\_ = float(node.height / 2)

p = self.\_contains(node.x0, node.y0, w\_, h\_, node.points)

x1 = Node(node.x0, node.y0, w\_, h\_, p)

self.\_recursive\_subdivide(x1, k)

p = self.\_contains(node.x0, node.y0 + h\_, w\_, h\_, node.points)

x2 = Node(node.x0, node.y0 + h\_, w\_, h\_, p)

self.\_recursive\_subdivide(x2, k)

p = self.\_contains(node.x0 + w\_, node.y0, w\_, h\_, node.points)

x3 = Node(node.x0 + w\_, node.y0, w\_, h\_, p)

self.\_recursive\_subdivide(x3, k)

p = self.\_contains(node.x0 + w\_, node.y0 + h\_, w\_, h\_, node.points)

x4 = Node(node.x0 + w\_, node.y0 + h\_, w\_, h\_, p)

self.\_recursive\_subdivide(x4, k)

node.children = [x1, x2, x3, x4]

def \_contains(self, x, y, w, h, points):

pts = []

for point in points:

if x <= point.x <= x + w and y <= point.y <= y + h:

pts.append(point)

return pts

def \_visualize\_recursive(self, node, depth):

if not node.children:

plt.gca().add\_patch(patches.Rectangle((node.x0, node.y0), node.width, node.height, fill=False, edgecolor='black'))

return

for child in node.children:

self.\_visualize\_recursive(child, depth + 1)

# Example usage:

qtree = QTree(4, 50)

qtree.subdivide()

qtree.visualize()

# Plot points with random colors

colors = [(random.random(), random.random(), random.random()) for \_ in range(len(qtree.get\_points()))]

plt.scatter([p.x for p in qtree.get\_points()], [p.y for p in qtree.get\_points()], color=colors)

plt.gca().set\_aspect('equal', adjustable='box')

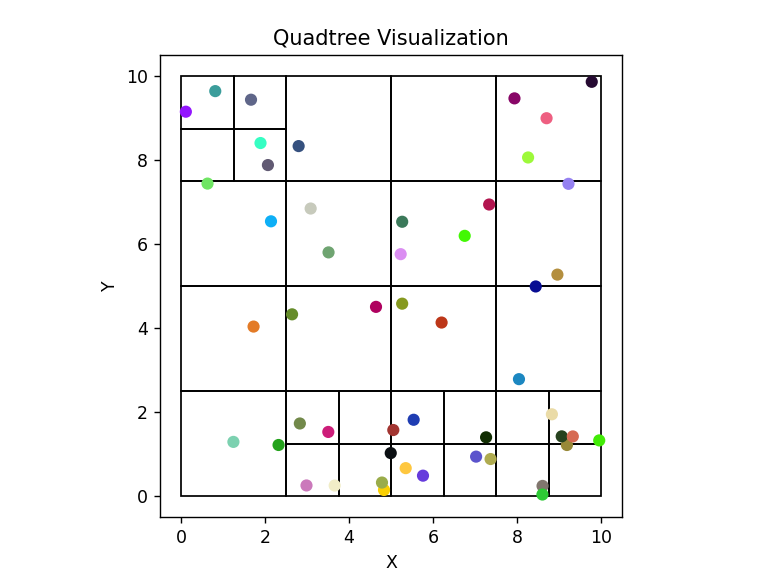
plt.title("Quadtree Visualization")

plt.xlabel("X")

plt.ylabel("Y")

plt.show()

**Output :**

****

**Code :**

assembly(computer, processor, 1).

assembly(computer, memory, 2).

assembly(computer, storage, 1).

assembly(computer, peripherals, 1).

assembly(processor, cpu, 1).

assembly(processor, cooler, 1).

assembly(processor, thermal\_paste, 1).

assembly(memory, ram, 4).

assembly(memory, cache, 1).

assembly(memory, controller, 1).

assembly(storage, hdd, 1).

assembly(storage, ssd, 1).

assembly(storage, optical\_drive, 1).

assembly(peripherals, monitor, 1).

assembly(peripherals, keyboard, 1).

assembly(peripherals, mouse, 1).

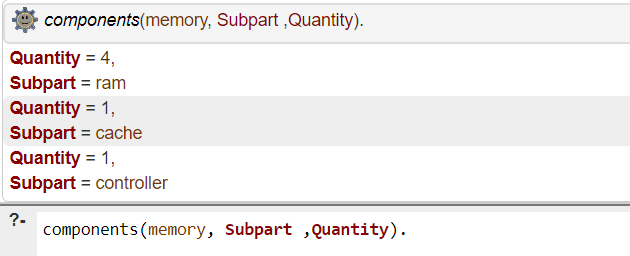
% Components relationship

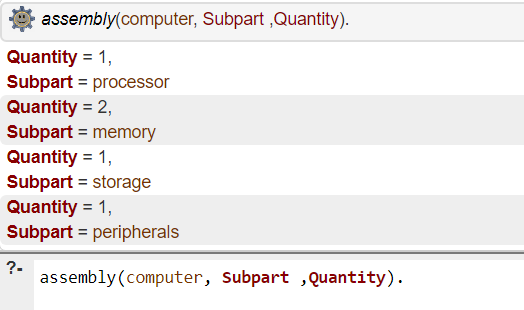
components(Part, Subpart, Quantity) :- assembly(Part, Subpart, Quantity).

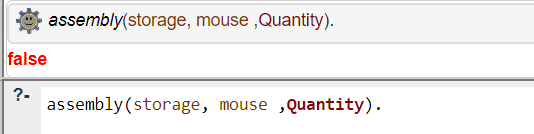
components(Part, Subpart, Quantity) :- assembly(Part, Intermediate, Qty1),

components(Intermediate, Subpart, Qty2),

Quantity is Qty1 \* Qty2.

**Output :**

****

****