

**PROJECT BASED LEARNING**

**PROJECT REPORT SUBMITTED IN A**

**SEMESTER 5 OF BACHELOR OF TECHNOLOGY**

**IN COMPUTER SCIENCE AND ENGINEERING BY**

**21331A0530 – Ch.Yashwanth**

**21331A0549 – G.Anusha**

**21331A0556 – G.Varshit Varma**

**21331A0557– G.Sai Hemanth Kumar**

**21331A0559 – G.Amani**

Under the esteemed guidance of

**Mr. M.Vamsi Krishna**

**Department of Computer Science and Engineering**

**MVGR College of Engineering**

**Table of Contents :**

[**1.Introduction 3**](#_dhde49cjtub8)

[**2.Project Overviews 3**](#_2w1ujjbvq7jx)

[**3.List of Entities & Attributes 3**](#_6n7y9n1a0aml)

[**4.List of Relationships 4**](#_2gda5il3q1q7)

[**5.Basics To Do 4**](#_rf1rpz8lzf8e)

[5.1.Design the logical view using ER Diagrams with tools 4](#_jaf434todd12)

[5.2.Design Enhanced ER diagram using Workbench 5](#_3gv660qdl78l)

[5.3.Forward Engineer your EER diagram in Workbench 6](#_27msyfyumbr5)

[5.3.1.Schema 6](#_lgcbgf832yrk)

[5.3.1.1.Query 6](#_tsj381ucb4b0)

[5.3.1.2.Successful Completion of Forward Engineering 10](#_yfpxkfon3g64)

[5.3.1.3.Code Repository Link 10](#_puwdc5crezyg)

[5.3.2.Tables in Database 11](#_nteakbukungq)

[5.3.2.1.Category 11](#_cvyqplhr4l3z)

[5.3.2.2.Event 11](#_9o0v7y4v8o94)

[5.3.2.3.Post 12](#_1n5xxhpo7obm)

[5.4.2.4.Comment 13](#_6v7r2nsks4nw)

[5.4.2.5.Tag 13](#_k1jitgucy2ca)

[5.4.2.6.Post\_has\_Tag 14](#_35iek4uzpvpq)

[5.4.SQL Queries to demonstrate the working 15](#_aavri0cktrx3)

[5.4.1.Test Cases in Python 15](#_palwf2ejddj0)

[5.4.1.1.Select Query 15](#_5zy1hzhq39be)

[5.4.1.2.Insert Query 16](#_ds3p6dtaem5f)

[5.4.2.GUI 18](#_m3rueqap1nfl)

[5.4.2.1.GUI Code in Python 18](#_yottvlb4r71s)

[5.4.2.2.Code Repository Link 32](#_tbb1dupbu497)

[5.4.2.3.Output 33](#_65y61t8rzlp7)

[5.4.2.3.1.Display Tables 33](#_k0h7x8ch5v8e)

[5.4.2.3.2.Insert values 35](#_pp7eagxvutwz)

[**6.Implement SQL Queries to display in 36**](#_cgdhkgysba2s)

[6.1.Mysql Workbench(using Mysql) 36](#_7xy3a7riju9n)

[6.1.1.Popular blog posts 36](#_6ekpvg1hl838)

[6.1.2.Manage comments 36](#_j8jkn03mbh9i)

[6.1.3.Categorize posts 37](#_utpcwo4dtvwr)

[6.2.Visual Studio Code(using Python) 37](#_s1qe6j3u6jum)

[6.2.1.Query:(Display Popular blog posts, manage comments, Categorize posts) 37](#_8pqzb8lsxozj)

[6.2.2.Output 39](#_tqdeg0fpbbx3)

[**7.Remarks 39**](#_11kvuurvi3xc)

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# **1.Introduction**

This is a project on a “Blogging Platform” where we have Blog Posts, Blog Categories, Tags of a Blog, Comments on the posts of a Blog. We need to manage all the data of the blogs in an efficient way where we can store the data in an efficient way with reduction of duplicate values in the tables and easy to handle the data. We need to organise the data and provide users more access and data control over their data.

# **2.Project Overviews**

The Blogging Platform Project in DBMS revolves around the creation of a robust and user-friendly platform where individuals and entities can create and manage blog posts, categorise them, receive feedback through comments, enhance discoverability with tags, and organise or participate in events. The primary focus of this project is the design and implementation of a sophisticated database system that efficiently handles the diverse data associated with these entities.

# **3.List of Entities & Attributes**

| **Entity** | **Attribute** |
| --- | --- |
| Category | Category\_ID, Category\_Type |
| Event | Event\_ID, Event\_Name, Event\_Organiser\_Name |
| Post | Post\_ID, Post\_Article, Post\_Views |
| Comment | Cmt\_ID, Cmt\_Data |
| Tag | Tag\_ID, Tag\_Word |

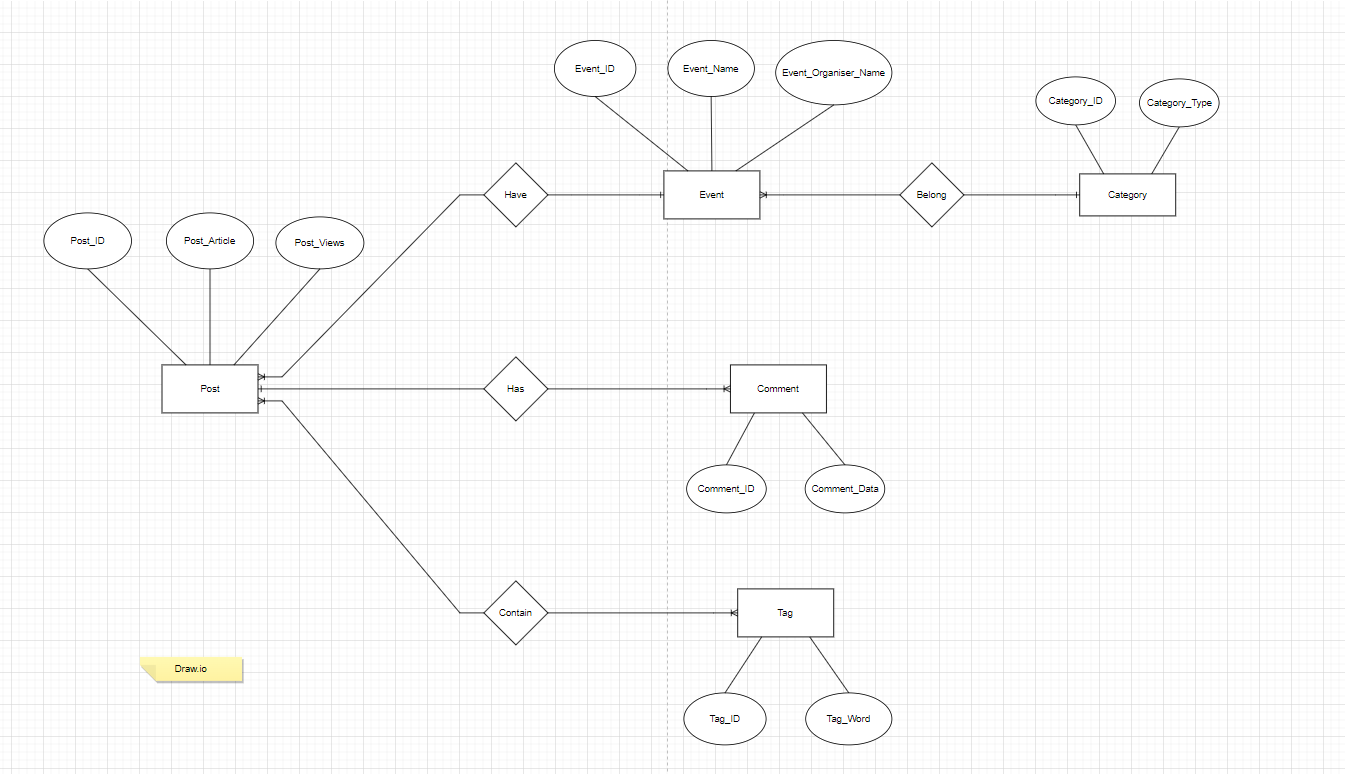
# **4.List of Relationships**

* One or many Events can belong to one Category.
* One Event can have many Posts.
* One or many Posts can contain one or many Tags.
* One Post can have many Comments.

# **5.Basics To Do**

## **5.1.Design the logical view using ER Diagrams with tools**

ER Diagram :

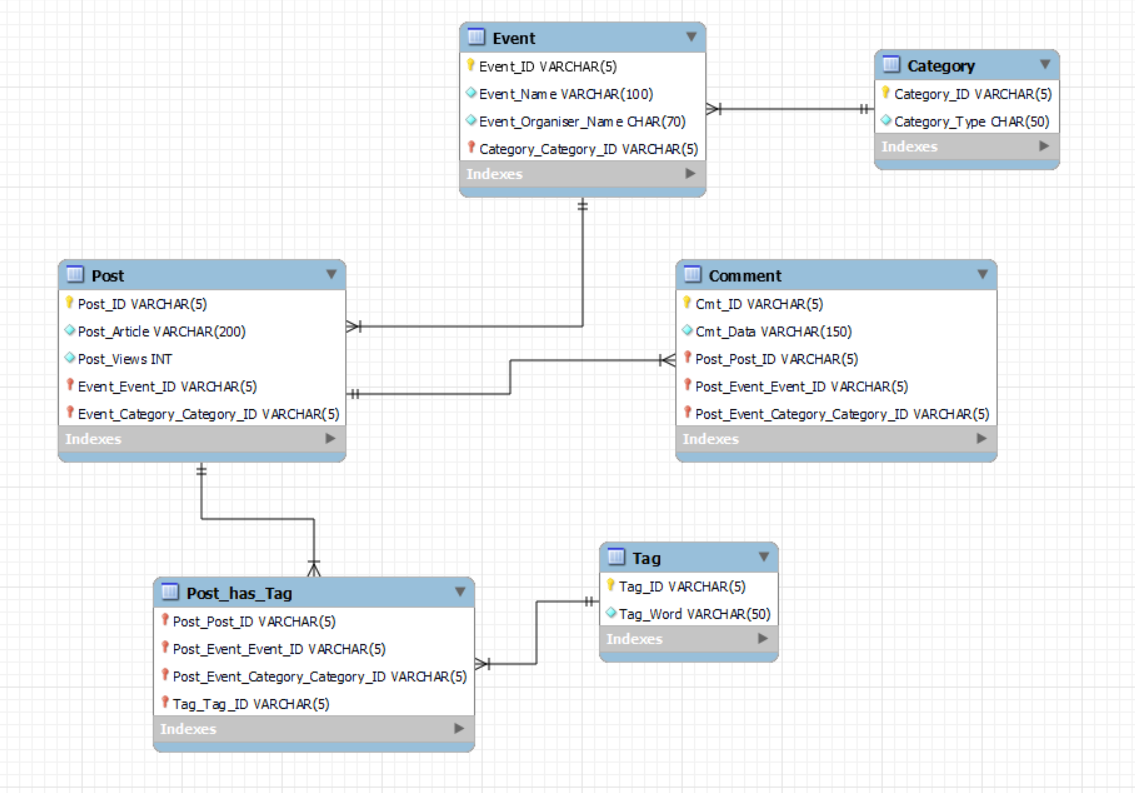


## 

## 

## **5.2.Design Enhanced ER diagram using Workbench**

EER Diagram :

****

## **5.3.Forward Engineer your EER diagram in Workbench**

### **5.3.1.Schema**

#### **5.3.1.1.Query**

-- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;

USE `mydb` ;

-- -----------------------------------------------------

-- Table `mydb`.`Category`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Category` (

`Category\_ID` VARCHAR(5) NOT NULL,

`Category\_Type` CHAR(50) NOT NULL,

PRIMARY KEY (`Category\_ID`),

UNIQUE INDEX `Category\_ID\_UNIQUE` (`Category\_ID` ASC) VISIBLE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Event`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Event` (

`Event\_ID` VARCHAR(5) NOT NULL,

`Event\_Name` VARCHAR(100) NOT NULL,

`Event\_Organiser\_Name` CHAR(70) NOT NULL,

`Category\_Category\_ID` VARCHAR(5) NOT NULL,

PRIMARY KEY (`Event\_ID`, `Category\_Category\_ID`),

UNIQUE INDEX `Event\_ID\_UNIQUE` (`Event\_ID` ASC) VISIBLE,

INDEX `fk\_Event\_Category\_idx` (`Category\_Category\_ID` ASC) VISIBLE,

CONSTRAINT `fk\_Event\_Category`

FOREIGN KEY (`Category\_Category\_ID`)

REFERENCES `mydb`.`Category` (`Category\_ID`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Post`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Post` (

`Post\_ID` VARCHAR(5) NOT NULL,

`Post\_Article` VARCHAR(600) NOT NULL,

`Post\_Views` INT NOT NULL,

`Event\_Event\_ID` VARCHAR(5) NOT NULL,

`Event\_Category\_Category\_ID` VARCHAR(5) NOT NULL,

PRIMARY KEY (`Post\_ID`, `Event\_Event\_ID`, `Event\_Category\_Category\_ID`),

UNIQUE INDEX `Post\_ID\_UNIQUE` (`Post\_ID` ASC) VISIBLE,

INDEX `fk\_Post\_Event1\_idx` (`Event\_Event\_ID` ASC, `Event\_Category\_Category\_ID` ASC) VISIBLE,

CONSTRAINT `fk\_Post\_Event1`

FOREIGN KEY (`Event\_Event\_ID` , `Event\_Category\_Category\_ID`)

REFERENCES `mydb`.`Event` (`Event\_ID` , `Category\_Category\_ID`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Comment`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Comment` (

`Comment\_ID` VARCHAR(5) NOT NULL,

`Comment\_Data` VARCHAR(500) NOT NULL,

`Post\_Post\_ID` VARCHAR(5) NOT NULL,

`Post\_Event\_Event\_ID` VARCHAR(5) NOT NULL,

`Post\_Event\_Category\_Category\_ID` VARCHAR(5) NOT NULL,

PRIMARY KEY (`Comment\_ID`, `Post\_Post\_ID`, `Post\_Event\_Event\_ID`, `Post\_Event\_Category\_Category\_ID`),

UNIQUE INDEX `Comment\_ID\_UNIQUE` (`Comment\_ID` ASC) VISIBLE,

INDEX `fk\_Comment\_Post1\_idx` (`Post\_Post\_ID` ASC, `Post\_Event\_Event\_ID` ASC, `Post\_Event\_Category\_Category\_ID` ASC) VISIBLE,

CONSTRAINT `fk\_Comment\_Post1`

FOREIGN KEY (`Post\_Post\_ID` , `Post\_Event\_Event\_ID` , `Post\_Event\_Category\_Category\_ID`)

REFERENCES `mydb`.`Post` (`Post\_ID` , `Event\_Event\_ID` , `Event\_Category\_Category\_ID`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Tag`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Tag` (

`Tag\_ID` VARCHAR(5) NOT NULL,

`Tag\_Word` VARCHAR(50) NOT NULL,

PRIMARY KEY (`Tag\_ID`),

UNIQUE INDEX `Tag\_ID\_UNIQUE` (`Tag\_ID` ASC) VISIBLE,

UNIQUE INDEX `Tag\_Word\_UNIQUE` (`Tag\_Word` ASC) VISIBLE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Post\_has\_Tag`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Post\_has\_Tag` (

`Post\_Post\_ID` VARCHAR(5) NOT NULL,

`Post\_Event\_Event\_ID` VARCHAR(5) NOT NULL,

`Post\_Event\_Category\_Category\_ID` VARCHAR(5) NOT NULL,

`Tag\_Tag\_ID` VARCHAR(5) NOT NULL,

PRIMARY KEY (`Post\_Post\_ID`, `Post\_Event\_Event\_ID`, `Post\_Event\_Category\_Category\_ID`, `Tag\_Tag\_ID`),

INDEX `fk\_Post\_has\_Tag\_Tag1\_idx` (`Tag\_Tag\_ID` ASC) VISIBLE,

INDEX `fk\_Post\_has\_Tag\_Post1\_idx` (`Post\_Post\_ID` ASC, `Post\_Event\_Event\_ID` ASC, `Post\_Event\_Category\_Category\_ID` ASC) VISIBLE,

CONSTRAINT `fk\_Post\_has\_Tag\_Post1`

FOREIGN KEY (`Post\_Post\_ID` , `Post\_Event\_Event\_ID` , `Post\_Event\_Category\_Category\_ID`)

REFERENCES `mydb`.`Post` (`Post\_ID` , `Event\_Event\_ID` , `Event\_Category\_Category\_ID`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Post\_has\_Tag\_Tag1`

FOREIGN KEY (`Tag\_Tag\_ID`)

REFERENCES `mydb`.`Tag` (`Tag\_ID`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

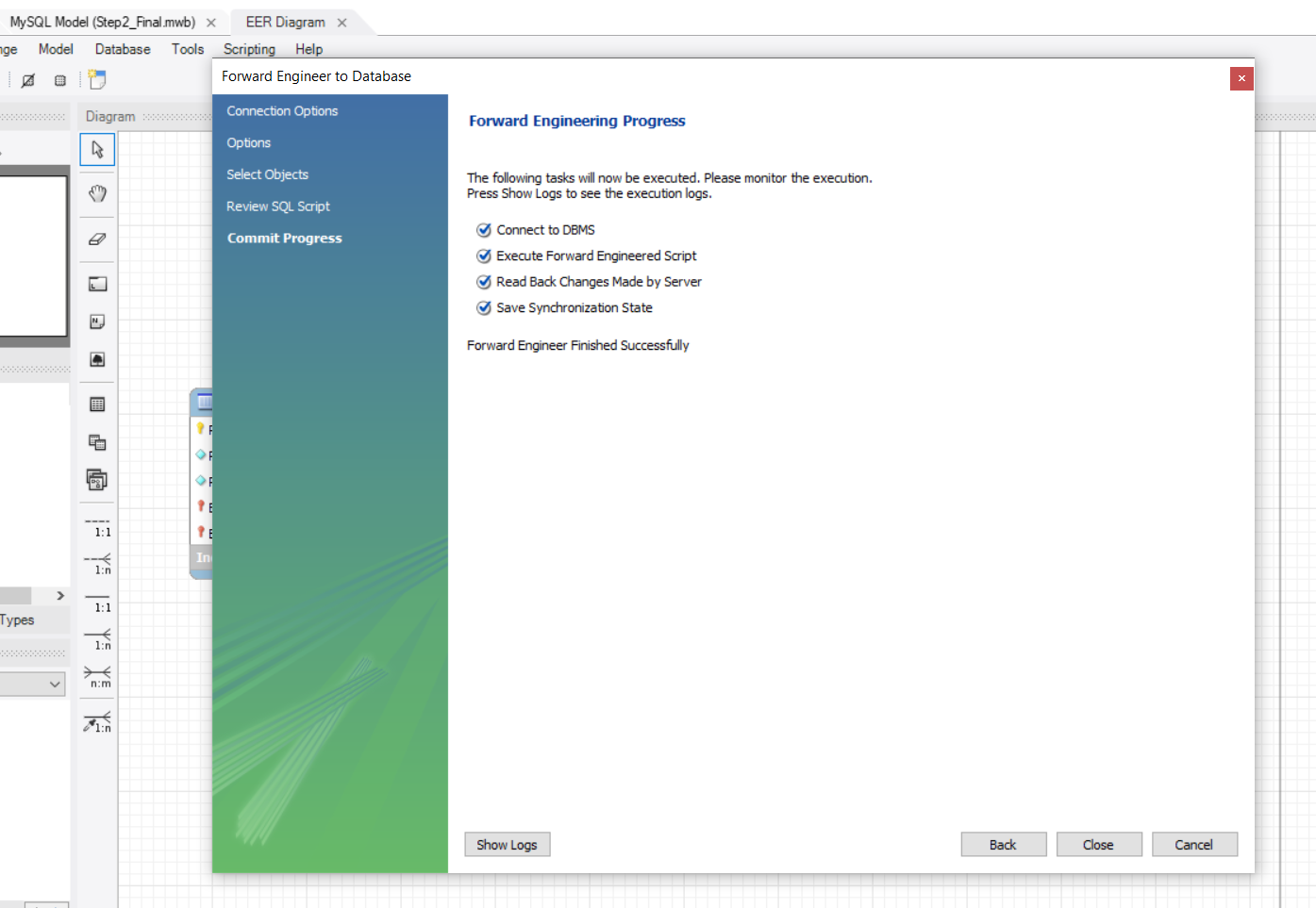
ENGINE = InnoDB;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

#### **5.3.1.2.Successful Completion of Forward Engineering**



#### 

#### **5.3.1.3.Code Repository Link**

[Final Schema](https://github.com/Yash-530/DBMS_PBL/blob/main/Final_Schema.sql)

### **5.3.2.Tables in Database**

#### **5.3.2.1.Category**

Query :

-- category Values--

insert into Category (Category\_ID, Category\_Type)

values

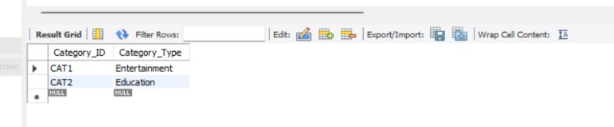
('CAT1','Entertainment'),

('CAT2', 'Education');

select \* from Category;

drop table Category;

Output :



#### **5.3.2.2.Event**

Query :

-- Events Values--

INSERT INTO events (Event\_ID, Event\_Name, Event\_Organiser\_Name, Category\_Category\_ID)

VALUES

('EV1', 'Silver Jublee', 'Anusha', 'CAT1'),

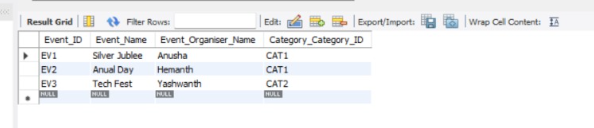
('EV2', 'Annual Day', 'Hemanth', 'CAT1'),

('EV3', 'Tech Fest', 'Yashwanth', 'CAT2');

select \* from Event;

drop table Event;

Output :



#### **5.3.2.3.Post**

Query :

-- values in Post --

insert into post values

("PT1","Article\_A",1000,"EV3","CAT2"),

("PT2","Article\_B",530,"EV3","CAT2"),

("PT3","Article\_C",500,"EV1","CAT1"),

("PT4","Article\_D",557,"EV2","CAT1"),

("PT5","Article\_E",559,"EV2","CAT1"),

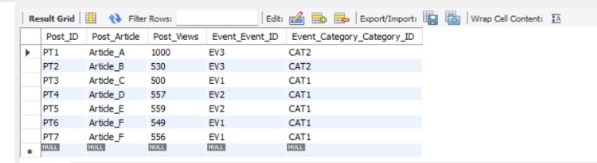
("PT6","Article\_F",549,"EV1","CAT1"),

("PT7","Article\_F",556,"EV1","CAT1");

select \* from Post;

drop table post;

Output :



#### **5.4.2.4.Comment**

Query :

-- values in Comment --

insert into Comment values

("COM1","Nice","PT6","EV1","CAT1"),

("COM2","Average","PT4","EV2","CAT1"),

("COM3","Interesting","PT2","EV3","CAT2"),

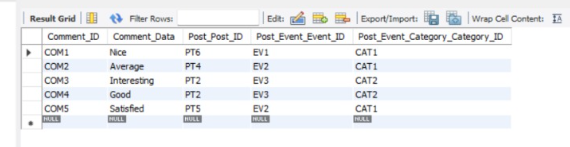
("COM4","Good","PT2","EV3","CAT2"),

("COM5","Satisfied","PT5","EV2","CAT1");

select \* from Comment;

drop table Comment;

Output :



#### **5.4.2.5.Tag**

Query :

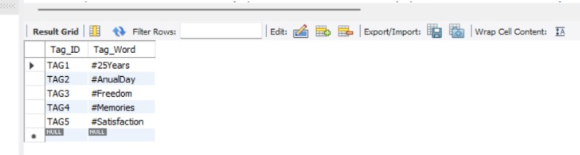
-- values in Tag --

insert into Tag values ("TAG1","#25Years"),("TAG2","#AnualDay"),("TAG3","#Freedom"),

("TAG4","#Memories"),("TAG5","#Satisfaction");

select \* from Tag;

Output :



#### **5.4.2.6.Post\_has\_Tag**

Query :

-- values in Post\_Has\_Tag --

insert into Post\_Has\_Tag

values ('PT1','EV3','CAT2','TAG3'),

('PT2','EV3','CAT2','TAG5'),

('PT3','EV1','CAT1','TAG4'),

('PT4','EV2','CAT1','TAG2'),

('PT5','EV2','CAT1','TAG2'),

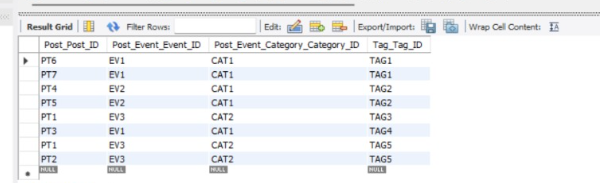
('PT6','EV1','CAT1','TAG1'),

('PT7','EV1','CAT1','TAG1'),

('PT1','EV3','CAT2','TAG5');

select \* from Post\_Has\_Tag;

Output :



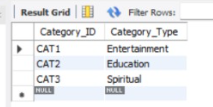
## **5.4.SQL Queries to demonstrate the working**

### **5.4.1.Test Cases in Python**

#### **5.4.1.1.Select Query**

Original table :

Category Table :



Query :

import mysql.connector as c

from tabulate import tabulate

from termcolor import colored

conn = c.connect(user='Pbl', password='Pbl@123', host='localhost', database='mydb')

cursor = conn.cursor()

sql1 = 'SELECT \* FROM Category'

cursor.execute(sql1)

result1 = cursor.fetchall()

def print\_results\_as\_table(query, results):

print(colored("Query:", "blue"), colored(query, "cyan"))

headers = [colored(i[0], "green") for i in cursor.description]

colored\_results = [[colored(str(cell), "yellow") for cell in row] for row in results]

print(tabulate(colored\_results, headers=headers, tablefmt="fancy\_grid"))

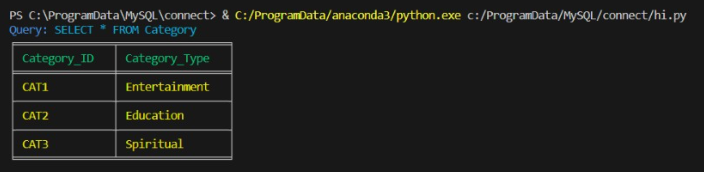
print()

print\_results\_as\_table(sql1, result1)

cursor.close()

conn.close()

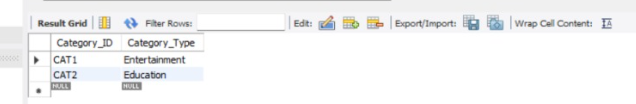
Output :



#### **5.4.1.2.Insert Query**

Original table :

Category Table :



Query **:**

import mysql.connector as c

from tabulate import tabulate

from termcolor import colored

conn = c.connect(user='Pbl', password='Pbl@123', host='localhost', database='mydb')

cursor = conn.cursor()

# Display the Category table before the update

cursor.execute("SELECT \* FROM Category;")

category\_result, headers = cursor.fetchall(), [colored(i[0], "blue") for i in cursor.description]

table\_data = [[colored(cell, 'red') if cell in ['CAT3', 'Spiritual'] else colored(cell, 'blue') for cell in row] for row in category\_result]

print(tabulate(table\_data, headers=headers, tablefmt="fancy\_grid"))

# Insert a new record into the Category table

insert\_query = "INSERT INTO Category (Category\_ID, Category\_Type) VALUES (%s, %s)"

values = ('CAT3', 'Spiritual')

try:

cursor.execute(insert\_query, values)

conn.commit()

print(colored("Record added successfully:", "green"))

except c.Error as err:

print(colored(f"Error: {err}", "red"))

# Display the Category table after the update

cursor.execute("SELECT \* FROM Category;")

category\_result, headers = cursor.fetchall(), [colored(i[0], "blue") for i in cursor.description]

table\_data = [[colored(cell, 'red') if cell in ['CAT3', 'Spiritual'] else colored(cell, 'blue') for cell in row] for row in category\_result]

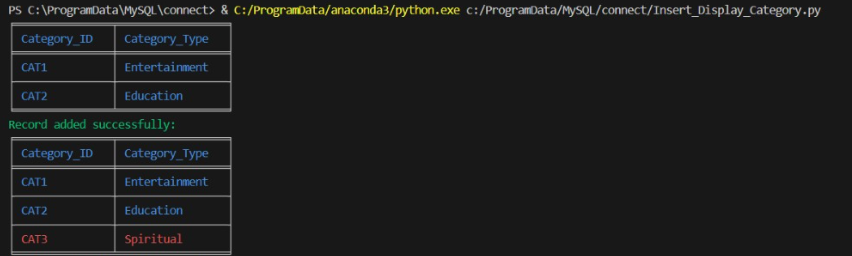
print(tabulate(table\_data, headers=headers, tablefmt="fancy\_grid"))

# Close the cursor and connection

cursor.close()

conn.close()

Output :



### 

### **5.4.2.GUI**

### **5.4.2.1.GUI Code in Python**

import tkinter as tk

from tkinter import ttk

import mysql.connector as c # or pymysql if you prefer

# Create a database connection

db = c.connect(

host="localhost",

user="Pbl",

password="Pbl@123",

db="mydb"

)

# Create a Tkinter application window

app = tk.Tk()

app.title("Database Management")

app["bg"]= "black"

# Function to load and display data for the Category table

def load\_category\_data():

cursor = db.cursor()

cursor.execute("SELECT \* FROM Category")

data = cursor.fetchall()

cursor.close()

# Clear existing data in the treeview

for item in category\_tree.get\_children():

category\_tree.delete(item)

# Insert data into the treeview

for row in data:

category\_tree.insert("", "end", values=row)

# Function to load and display data for the Event table

def load\_event\_data():

cursor = db.cursor()

cursor.execute("SELECT \* FROM Event")

data = cursor.fetchall()

cursor.close()

# Clear existing data in the treeview

for item in event\_tree.get\_children():

event\_tree.delete(item)

# Insert data into the treeview

for row in data:

event\_tree.insert("", "end", values=row)

# Function to load and display data for the Post table

def load\_post\_data():

cursor = db.cursor()

cursor.execute("SELECT \* FROM Post")

data = cursor.fetchall()

cursor.close()

# Clear existing data in the treeview

for item in post\_tree.get\_children():

post\_tree.delete(item)

# Insert data into the treeview

for row in data:

post\_tree.insert("", "end", values=row)

# Function to load and display data for the Comment table

def load\_comment\_data():

cursor = db.cursor()

cursor.execute("SELECT \* FROM Comment")

data = cursor.fetchall()

cursor.close()

# Clear existing data in the treeview

for item in comment\_tree.get\_children():

comment\_tree.delete(item)

# Insert data into the treeview

for row in data:

comment\_tree.insert("", "end", values=row)

# Function to load and display data for the Tag table

def load\_tag\_data():

cursor = db.cursor()

cursor.execute("SELECT \* FROM Tag")

data = cursor.fetchall()

cursor.close()

# Clear existing data in the treeview

for item in tag\_tree.get\_children():

tag\_tree.delete(item)

# Insert data into the treeview

for row in data:

tag\_tree.insert("", "end", values=row)

# Function to load and display data for the Post\_Has\_Tag table

def load\_post\_has\_tag\_data():

cursor = db.cursor()

cursor.execute("SELECT \* FROM Post\_Has\_Tag")

data = cursor.fetchall()

cursor.close()

# Clear existing data in the treeview

for item in post\_has\_tag\_tree.get\_children():

post\_has\_tag\_tree.delete(item)

# Insert data into the treeview

for row in data:

post\_has\_tag\_tree.insert("", "end", values=row)

# Create tabs for Post, Comment, Tag, and Post\_Has\_Tag tables

tab\_control = ttk.Notebook(app)

category\_tab = ttk.Frame(tab\_control)

event\_tab = ttk.Frame(tab\_control)

post\_tab = ttk.Frame(tab\_control)

comment\_tab = ttk.Frame(tab\_control)

tag\_tab = ttk.Frame(tab\_control)

post\_has\_tag\_tab = ttk.Frame(tab\_control)

tab\_control.add(category\_tab, text='Category')

tab\_control.add(event\_tab, text='Event')

tab\_control.add(post\_tab, text='Post')

tab\_control.add(comment\_tab, text='Comment')

tab\_control.add(tag\_tab, text='Tag')

tab\_control.add(post\_has\_tag\_tab, text='Post\_Has\_Tag')

tab\_control.pack(expand=1, fill="both")

# Create a Treeview widget for displaying Category data

category\_columns = ["Category\_ID", "Category\_Type"]

category\_tree = ttk.Treeview(category\_tab, columns=category\_columns, show="headings")

for col in category\_columns:

category\_tree.heading(col, text=col)

category\_tree.pack(pady=40)

load\_category\_data()

# Create a Treeview widget for displaying Event data

event\_columns = ["Event\_ID", "Event\_Name", "Event\_Organiser\_Name", "Category\_Category\_ID"]

event\_tree = ttk.Treeview(event\_tab, columns=event\_columns, show="headings")

for col in event\_columns:

event\_tree.heading(col, text=col)

event\_tree.pack(pady=40)

load\_event\_data()

# Create a Treeview widget for displaying Post data

post\_columns = ["Post\_ID", "Post\_Article", "Post\_Views", "Event\_Event\_ID", "Event\_Category\_Category\_ID"]

post\_tree = ttk.Treeview(post\_tab, columns=post\_columns, show="headings")

for col in post\_columns:

post\_tree.heading(col, text=col)

post\_tree.pack(pady=40)

load\_post\_data()

# Create a Treeview widget for displaying Comment data

comment\_columns = ["Comment\_ID", "Comment\_Data", "Post\_Post\_ID", "Post\_Event\_Event\_ID", "Post\_Event\_Category\_Category\_ID"]

comment\_tree = ttk.Treeview(comment\_tab, columns=comment\_columns, show="headings")

for col in comment\_columns:

comment\_tree.heading(col, text=col)

comment\_tree.pack(pady=40)

load\_comment\_data()

# Create a Treeview widget for displaying Tag data

tag\_columns = ["Tag\_ID", "Tag\_Word"]

tag\_tree = ttk.Treeview(tag\_tab, columns=tag\_columns, show="headings")

for col in tag\_columns:

tag\_tree.heading(col, text=col)

tag\_tree.pack(pady=40)

load\_tag\_data()

# Create a Treeview widget for displaying Post\_Has\_Tag data

post\_has\_tag\_columns = ["Post\_ID", "Event\_ID", "Cat\_ID", "Tag\_ID"]

post\_has\_tag\_tree = ttk.Treeview(post\_has\_tag\_tab, columns=post\_has\_tag\_columns, show="headings")

for col in post\_has\_tag\_columns:

post\_has\_tag\_tree.heading(col, text=col)

post\_has\_tag\_tree.pack(pady=40)

load\_post\_has\_tag\_data()

# Create entry fields and labels for inserting data into Category table

category\_id\_label = tk.Label(category\_tab, text="Category ID")

category\_id\_label.pack()

category\_id\_entry = tk.Entry(category\_tab)

category\_id\_entry.pack()

category\_type\_label = tk.Label(category\_tab, text="Category Type")

category\_type\_label.pack()

category\_type\_entry = tk.Entry(category\_tab)

category\_type\_entry.pack()

# Create entry fields and labels for inserting data into Event table

event\_id\_label = tk.Label(event\_tab, text="Event ID")

event\_id\_label.pack()

event\_id\_entry = tk.Entry(event\_tab)

event\_id\_entry.pack()

event\_name\_label = tk.Label(event\_tab, text="Event Name")

event\_name\_label.pack()

event\_name\_entry = tk.Entry(event\_tab)

event\_name\_entry.pack()

event\_organiser\_name\_label = tk.Label(event\_tab, text="Event Organiser Name")

event\_organiser\_name\_label.pack()

event\_organiser\_name\_entry = tk.Entry(event\_tab)

event\_organiser\_name\_entry.pack()

category\_category\_id\_label = tk.Label(event\_tab, text="Category Category ID")

category\_category\_id\_label.pack()

category\_category\_id\_entry = tk.Entry(event\_tab)

category\_category\_id\_entry.pack()

# Create entry fields for the Post table

post\_id\_label = tk.Label(post\_tab, text="Post ID:")

post\_id\_label.pack()

post\_id\_entry = tk.Entry(post\_tab)

post\_id\_entry.pack()

post\_article\_label = tk.Label(post\_tab, text="Post Article:")

post\_article\_label.pack()

post\_article\_entry = tk.Entry(post\_tab)

post\_article\_entry.pack()

post\_views\_label = tk.Label(post\_tab, text="Post Views:")

post\_views\_label.pack()

post\_views\_entry = tk.Entry(post\_tab)

post\_views\_entry.pack()

event\_event\_id\_label = tk.Label(post\_tab, text="Event Event ID:")

event\_event\_id\_label.pack()

event\_event\_id\_entry = tk.Entry(post\_tab)

event\_event\_id\_entry.pack()

event\_category\_id\_label = tk.Label(post\_tab, text="Event Category ID:")

event\_category\_id\_label.pack()

event\_category\_id\_entry = tk.Entry(post\_tab)

event\_category\_id\_entry.pack()

# Create entry fields for the Comment table

comment\_id\_label = tk.Label(comment\_tab, text="Comment ID:")

comment\_id\_label.pack()

comment\_id\_entry = tk.Entry(comment\_tab)

comment\_id\_entry.pack()

comment\_data\_label = tk.Label(comment\_tab, text="Comment Data:")

comment\_data\_label.pack()

comment\_data\_entry = tk.Entry(comment\_tab)

comment\_data\_entry.pack()

comment\_post\_id\_label = tk.Label(comment\_tab, text="Post ID:")

comment\_post\_id\_label.pack()

comment\_post\_id\_entry = tk.Entry(comment\_tab)

comment\_post\_id\_entry.pack()

comment\_event\_id\_label = tk.Label(comment\_tab, text="Event ID:")

comment\_event\_id\_label.pack()

comment\_event\_id\_entry = tk.Entry(comment\_tab)

comment\_event\_id\_entry.pack()

comment\_category\_id\_label = tk.Label(comment\_tab, text="Category ID:")

comment\_category\_id\_label.pack()

comment\_category\_id\_entry = tk.Entry(comment\_tab)

comment\_category\_id\_entry.pack()

# Create entry fields for the Tag table

tag\_id\_label = tk.Label(tag\_tab, text="Tag ID:")

tag\_id\_label.pack()

tag\_id\_entry = tk.Entry(tag\_tab)

tag\_id\_entry.pack()

tag\_word\_label = tk.Label(tag\_tab, text="Tag Word:")

tag\_word\_label.pack()

tag\_word\_entry = tk.Entry(tag\_tab)

tag\_word\_entry.pack()

# Create entry fields for the Post\_Has\_Tag table

post\_has\_tag\_post\_id\_label = tk.Label(post\_has\_tag\_tab, text="Post ID:")

post\_has\_tag\_post\_id\_label.pack()

post\_has\_tag\_post\_id\_entry = tk.Entry(post\_has\_tag\_tab)

post\_has\_tag\_post\_id\_entry.pack()

post\_has\_tag\_event\_id\_label = tk.Label(post\_has\_tag\_tab, text="Event ID:")

post\_has\_tag\_event\_id\_label.pack()

post\_has\_tag\_event\_id\_entry = tk.Entry(post\_has\_tag\_tab)

post\_has\_tag\_event\_id\_entry.pack()

post\_has\_tag\_cat\_id\_label = tk.Label(post\_has\_tag\_tab, text="Cat ID:")

post\_has\_tag\_cat\_id\_label.pack()

post\_has\_tag\_cat\_id\_entry = tk.Entry(post\_has\_tag\_tab)

post\_has\_tag\_cat\_id\_entry.pack()

post\_has\_tag\_tag\_id\_label = tk.Label(post\_has\_tag\_tab, text="Tag ID:")

post\_has\_tag\_tag\_id\_label.pack()

post\_has\_tag\_tag\_id\_entry = tk.Entry(post\_has\_tag\_tab)

post\_has\_tag\_tag\_id\_entry.pack()

# Function to insert data into Category table

def insert\_category\_data():

category\_id = category\_id\_entry.get()

category\_type = category\_type\_entry.get()

cursor = db.cursor()

cursor.execute("INSERT INTO Category (Category\_ID, Category\_Type) VALUES (%s, %s)", (category\_id, category\_type))

db.commit()

cursor.close()

category\_id\_entry.delete(0, tk.END)

category\_type\_entry.delete(0, tk.END)

load\_category\_data()

# Button to insert data into Category table

insert\_category\_button = tk.Button(category\_tab, text="Insert Category",fg='black',bg='gray',borderwidth=5, command=insert\_category\_data)

insert\_category\_button.pack()

# Function to insert data into Event table

def insert\_event\_data():

Event\_ID = event\_id\_entry.get()

Event\_Name = event\_name\_entry.get()

Event\_Organiser\_Name = event\_organiser\_name\_entry.get()

Category\_Category\_ID = category\_category\_id\_entry.get()

cursor = db.cursor()

cursor.execute("INSERT INTO Event (Event\_ID, Event\_Name, Event\_Organiser\_Name, Category\_Category\_ID) VALUES (%s, %s, %s, %s)", (Event\_ID, Event\_Name, Event\_Organiser\_Name, Category\_Category\_ID))

db.commit()

cursor.close()

event\_id\_entry.delete(0, tk.END)

event\_name\_entry.delete(0, tk.END)

event\_organiser\_name\_entry.delete(0, tk.END)

category\_category\_id\_entry.delete(0, tk.END)

load\_event\_data()

# Button to insert data into Event table

insert\_event\_button = tk.Button(event\_tab, text="Insert Event",fg='black',bg='gray',borderwidth=5, command=insert\_event\_data)

insert\_event\_button.pack()

# Function to insert data into Post table

def insert\_post\_data():

# Get data from entry fields

post\_id = post\_id\_entry.get()

post\_article = post\_article\_entry.get()

post\_views = post\_views\_entry.get()

event\_event\_id = event\_event\_id\_entry.get()

event\_category\_id = event\_category\_id\_entry.get()

# Insert data into the Post table

cursor = db.cursor()

cursor.execute("INSERT INTO Post (Post\_ID, Post\_Article, Post\_Views, Event\_Event\_ID, Event\_Category\_Category\_ID) VALUES (%s, %s, %s, %s, %s)",

(post\_id, post\_article, post\_views, event\_event\_id, event\_category\_id))

db.commit()

cursor.close()

# Clear entry fields

post\_id\_entry.delete(0, tk.END)

post\_article\_entry.delete(0, tk.END)

post\_views\_entry.delete(0, tk.END)

event\_event\_id\_entry.delete(0, tk.END)

event\_category\_id\_entry.delete(0, tk.END)

# Reload data in the treeview

load\_post\_data()

# Create an insertion button for Post table

insert\_post\_button = tk.Button(post\_tab, text="Insert Post", command=insert\_post\_data,fg='black',bg='gray',borderwidth=5)

insert\_post\_button.pack()

# Function to insert data into Comment table

def insert\_comment\_data():

# Get data from entry fields

comment\_id = comment\_id\_entry.get()

comment\_data = comment\_data\_entry.get()

post\_post\_id = comment\_post\_id\_entry.get()

post\_event\_id = comment\_event\_id\_entry.get()

post\_category\_id = comment\_category\_id\_entry.get()

# Insert data into the Comment table

cursor = db.cursor()

cursor.execute("INSERT INTO Comment (Comment\_ID, Comment\_Data, Post\_Post\_ID, Post\_Event\_Event\_ID, Post\_Event\_Category\_Category\_ID) VALUES (%s, %s, %s, %s, %s)",

(comment\_id, comment\_data, post\_post\_id, post\_event\_id, post\_category\_id))

db.commit()

cursor.close()

# Clear entry fields

comment\_id\_entry.delete(0, tk.END)

comment\_data\_entry.delete(0, tk.END)

comment\_post\_id\_entry.delete(0, tk.END)

comment\_event\_id\_entry.delete(0, tk.END)

comment\_category\_id\_entry.delete(0, tk.END)

# Reload data in the treeview

load\_comment\_data()

# Create an insertion button for Comment table

insert\_comment\_button = tk.Button(comment\_tab, text="Insert Comment",fg='black',bg='gray',borderwidth=5, command=insert\_comment\_data)

insert\_comment\_button.pack()

# Function to insert data into Tag table

def insert\_tag\_data():

# Get data from entry fields

tag\_id = tag\_id\_entry.get()

tag\_word = tag\_word\_entry.get()

# Insert data into the Tag table

cursor = db.cursor()

cursor.execute("INSERT INTO Tag (Tag\_ID, Tag\_Word) VALUES (%s, %s)",

(tag\_id, tag\_word))

db.commit()

cursor.close()

# Clear entry fields

tag\_id\_entry.delete(0, tk.END)

tag\_word\_entry.delete(0, tk.END)

# Reload data in the treeview

load\_tag\_data()

# Create an insertion button for Tag table

insert\_tag\_button = tk.Button(tag\_tab, text="Insert Tag", fg='black',bg='gray',borderwidth=5,command=insert\_tag\_data,)

insert\_tag\_button.pack()

# Function to insert data into Post\_Has\_Tag table (similar to Post)

def insert\_post\_has\_tag\_data():

# Get data from entry fields

post\_id = post\_has\_tag\_post\_id\_entry.get()

event\_id = post\_has\_tag\_event\_id\_entry.get()

cat\_id = post\_has\_tag\_cat\_id\_entry.get()

tag\_id = post\_has\_tag\_tag\_id\_entry.get()

# Insert data into the Post\_Has\_Tag table

cursor = db.cursor()

cursor.execute("INSERT INTO Post\_Has\_Tag (Post\_ID, Event\_ID, Cat\_ID, Tag\_ID) VALUES (%s, %s, %s, %s)",

(post\_id, event\_id, cat\_id, tag\_id))

db.commit()

cursor.close()

# Clear entry fields

post\_has\_tag\_post\_id\_entry.delete(0, tk.END)

post\_has\_tag\_event\_id\_entry.delete(0, tk.END)

post\_has\_tag\_cat\_id\_entry.delete(0, tk.END)

post\_has\_tag\_tag\_id\_entry.delete(0, tk.END)

# Reload data in the treeview

load\_post\_has\_tag\_data()

# Create an insertion button for Post\_Has\_Tag table

insert\_post\_has\_tag\_button = tk.Button(post\_has\_tag\_tab, text="Insert Post\_Has\_Tag",fg='black',bg='gray',borderwidth=5, command=insert\_post\_has\_tag\_data)

insert\_post\_has\_tag\_button.pack()

# ... (previous code)

# Start the Tkinter event loop

app.mainloop()

# Close the database connection when the application is closed

db.close()

# Start the Tkinter event loop

app.mainloop()

# Close the database connection when the application is closed

db.close()

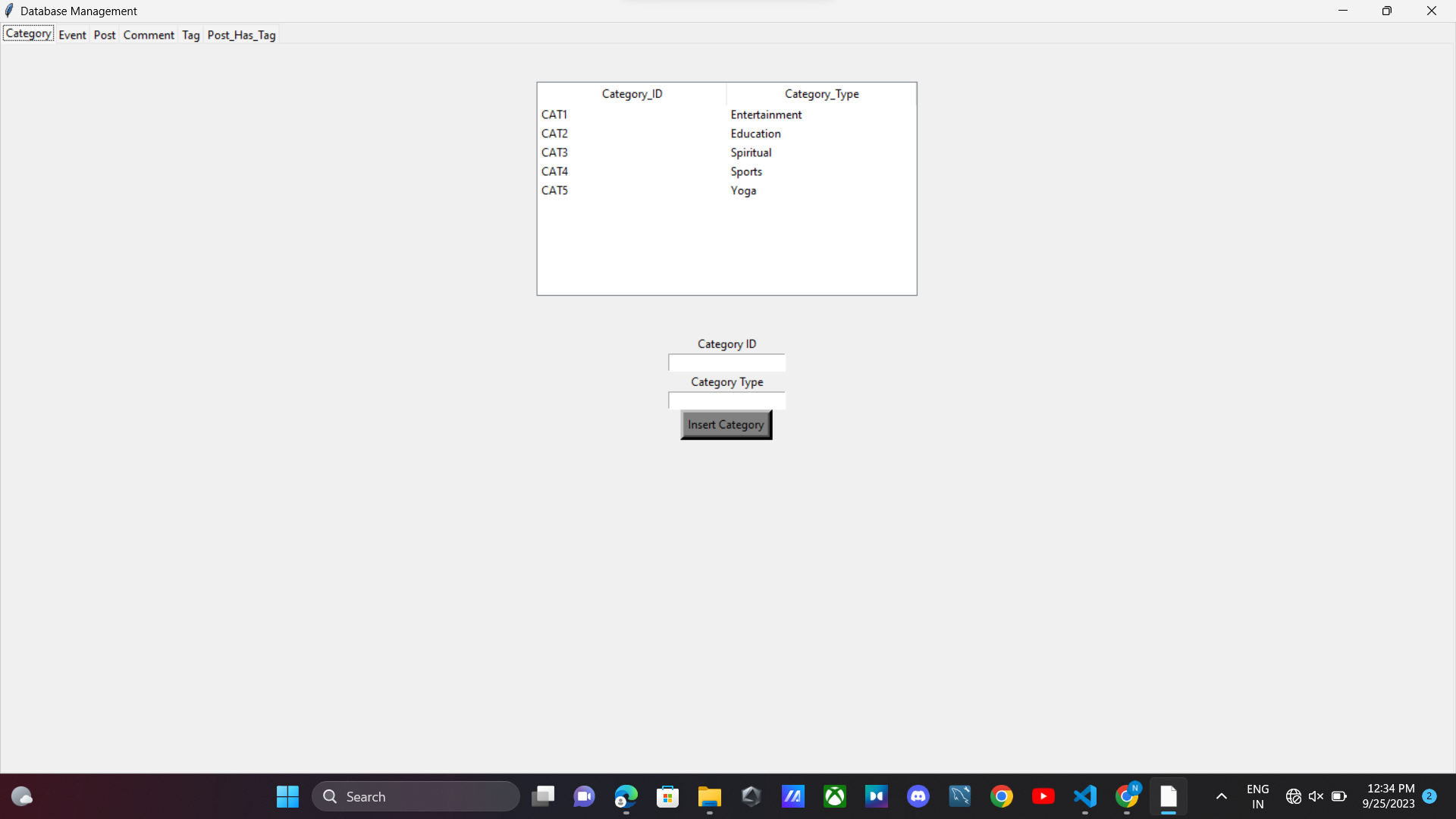
### **5.4.2.2.Code Repository Link**

[Git Hub - Code link](https://github.com/Yash-530/DBMS_PBL/blob/main/GUI_Final.py)

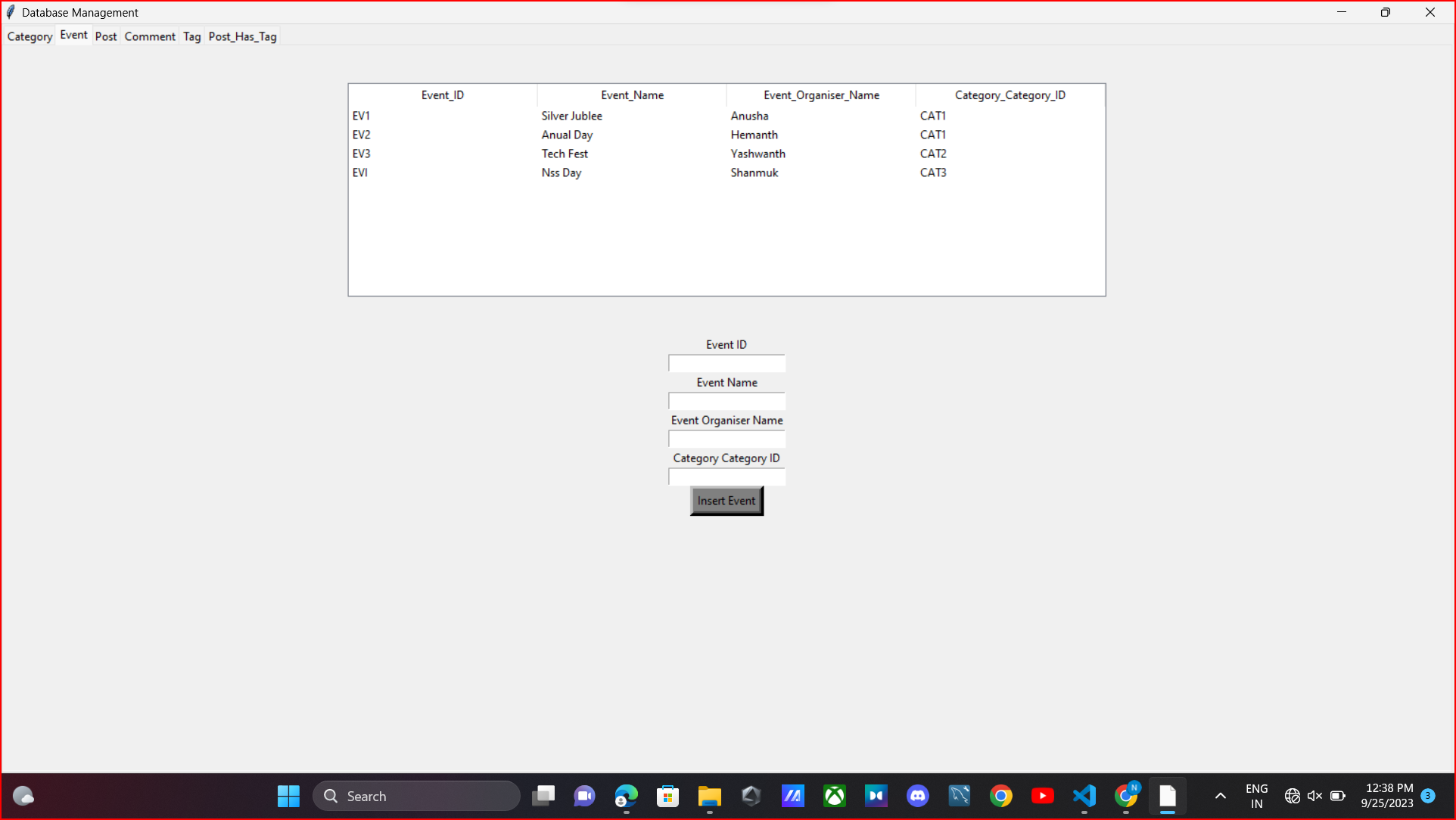
### **5.4.2.3.Output**

##### **5.4.2.3.1.Display Tables**

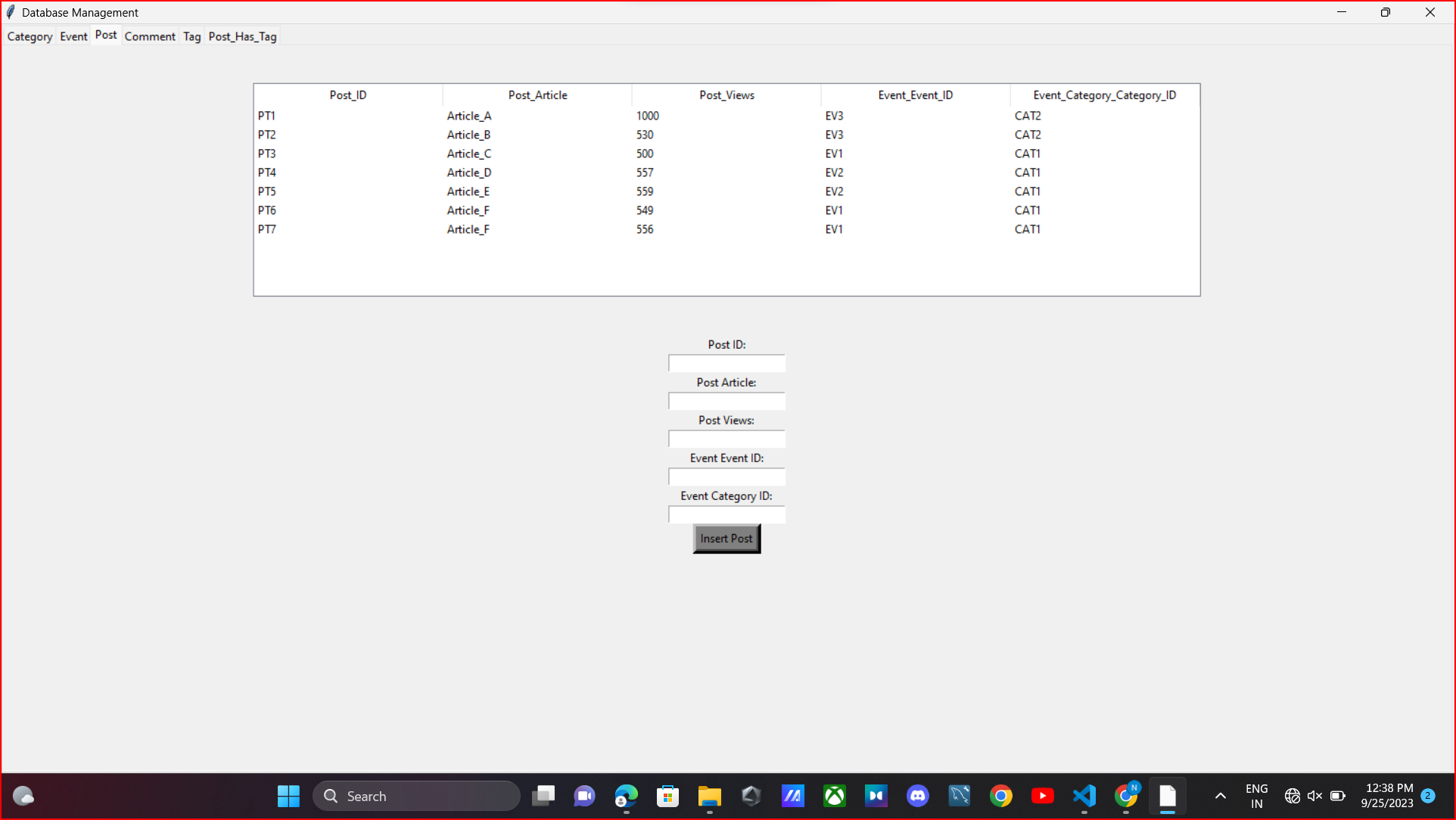
Category :



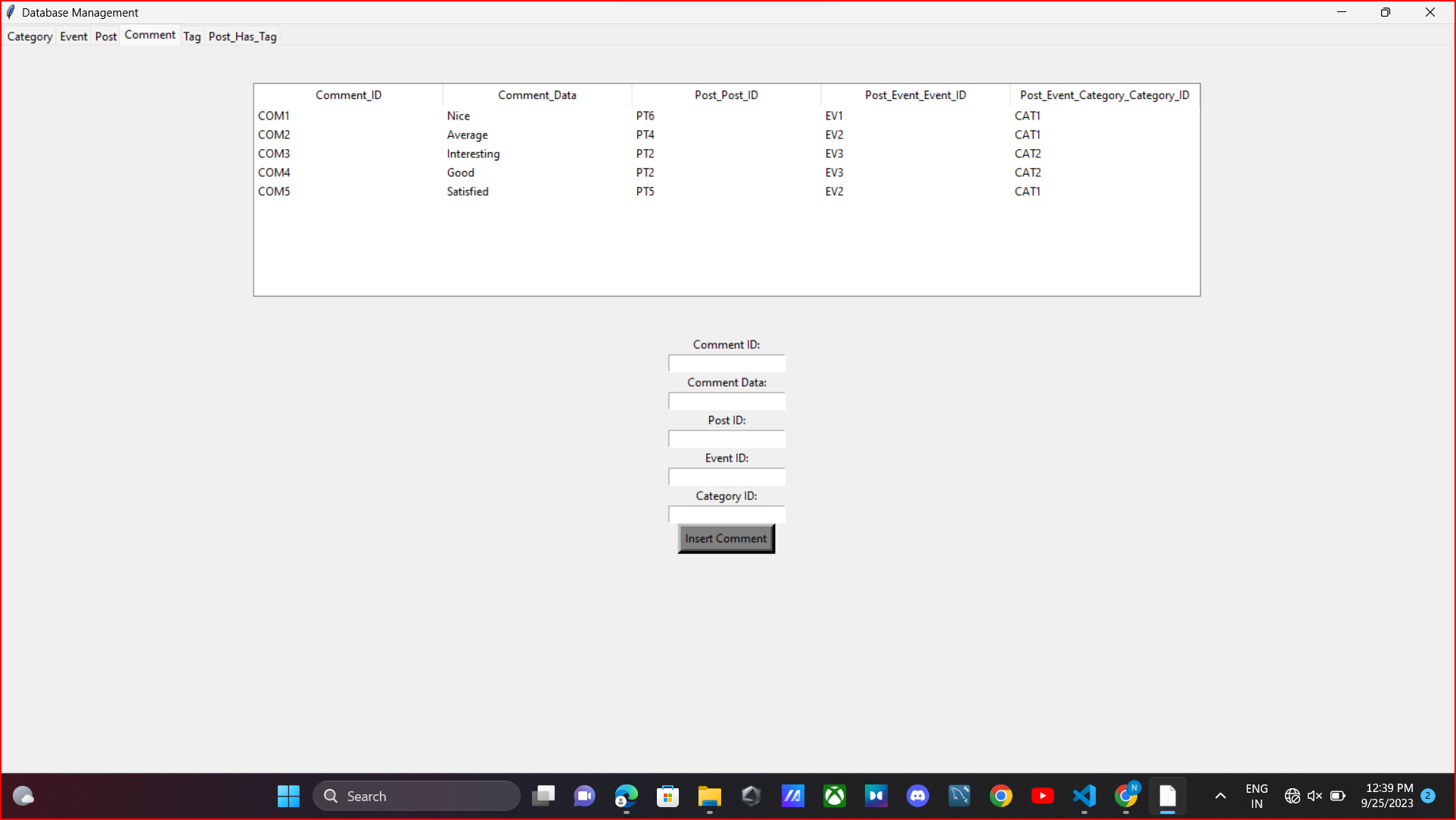
Event :



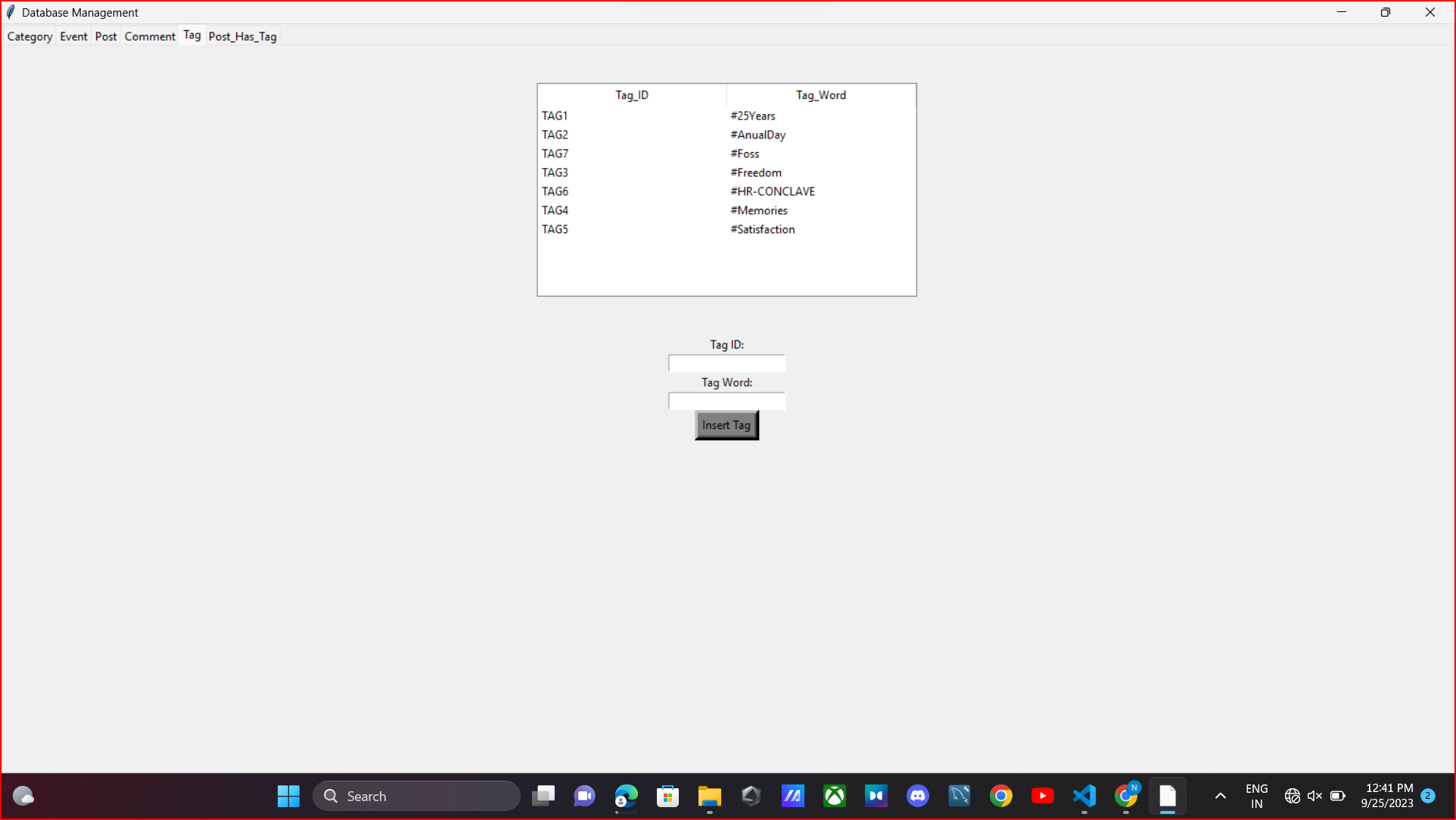
Post :



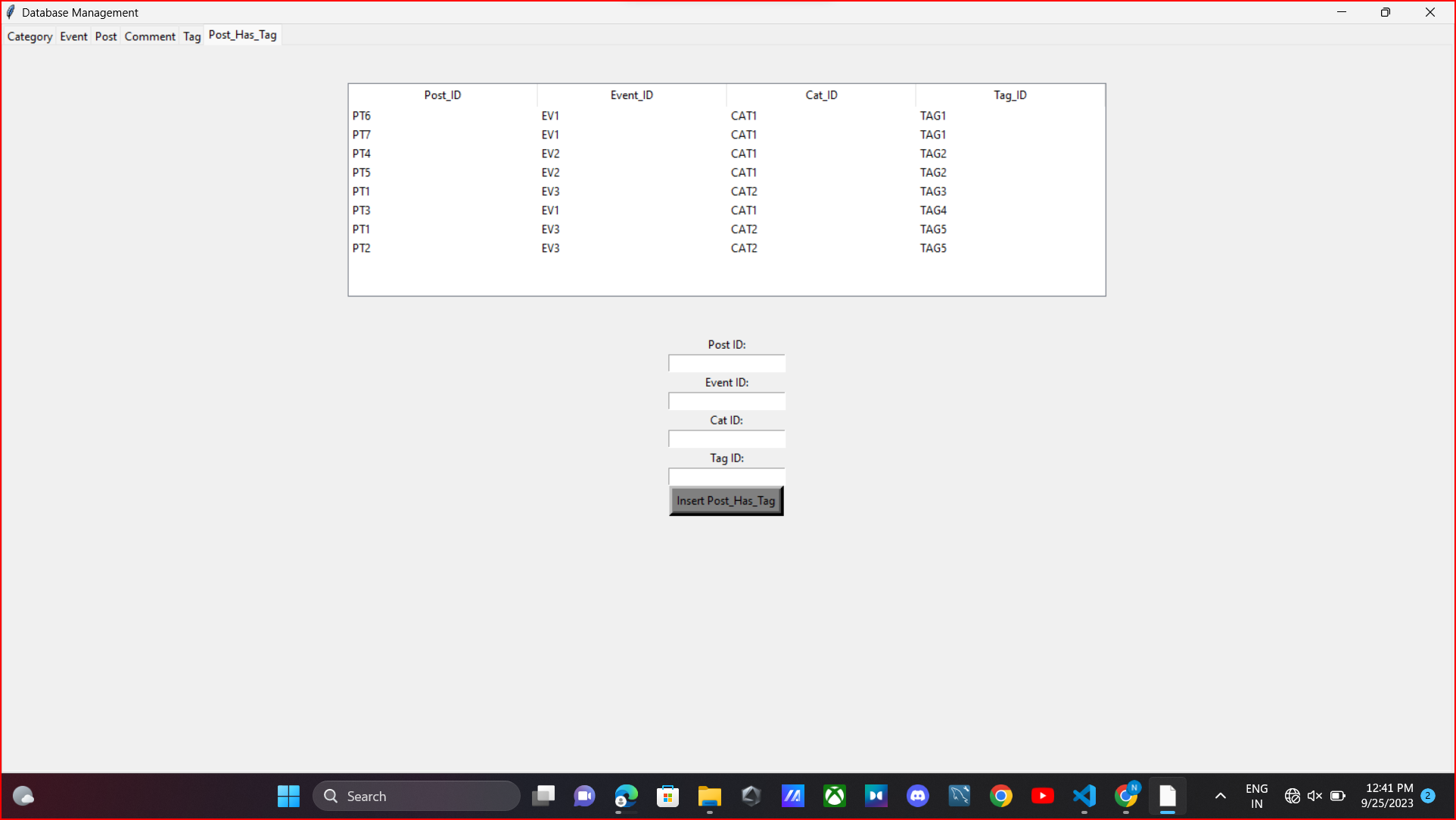
Comment :



Tag :

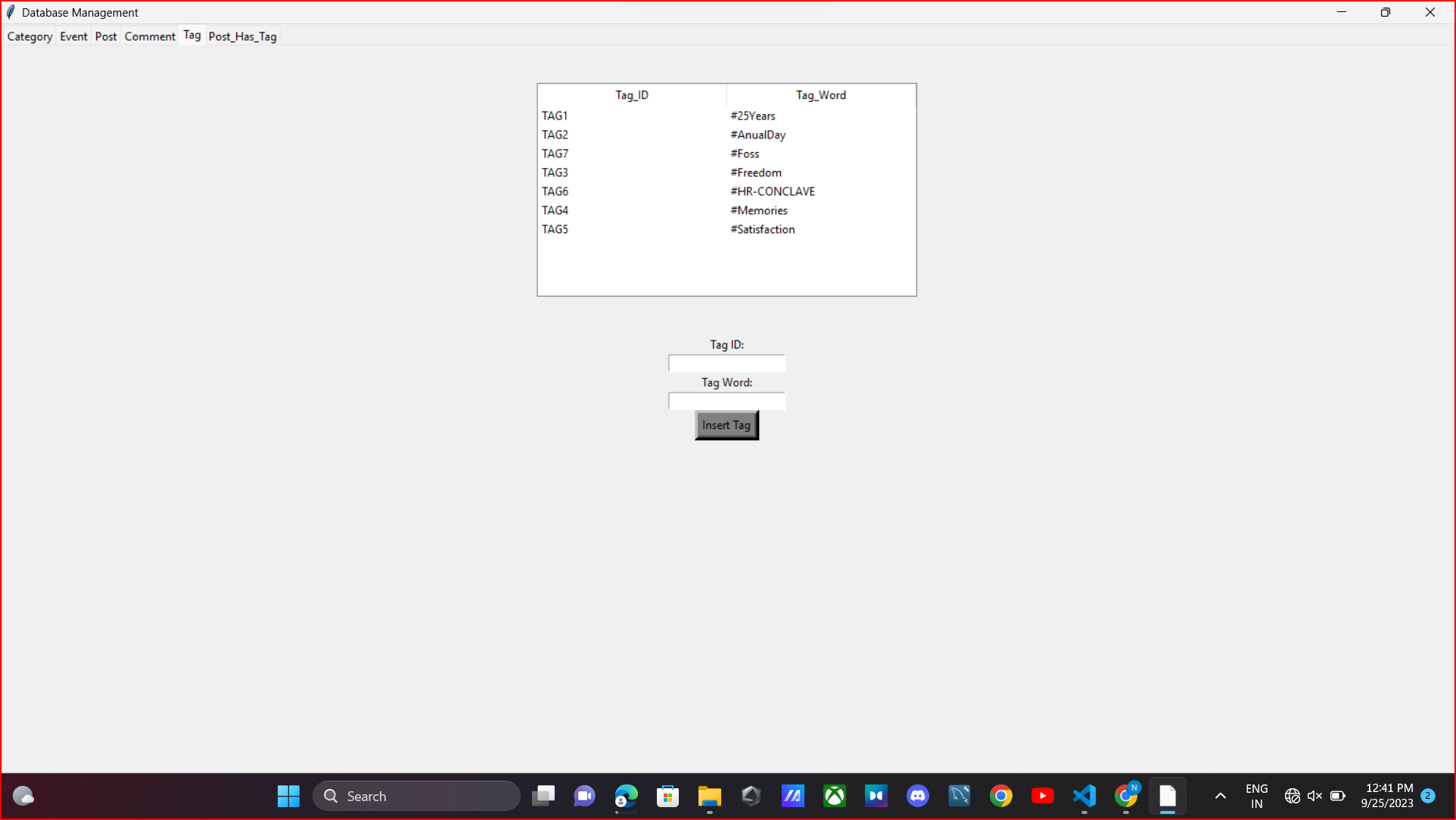


Post\_Has\_Tag :

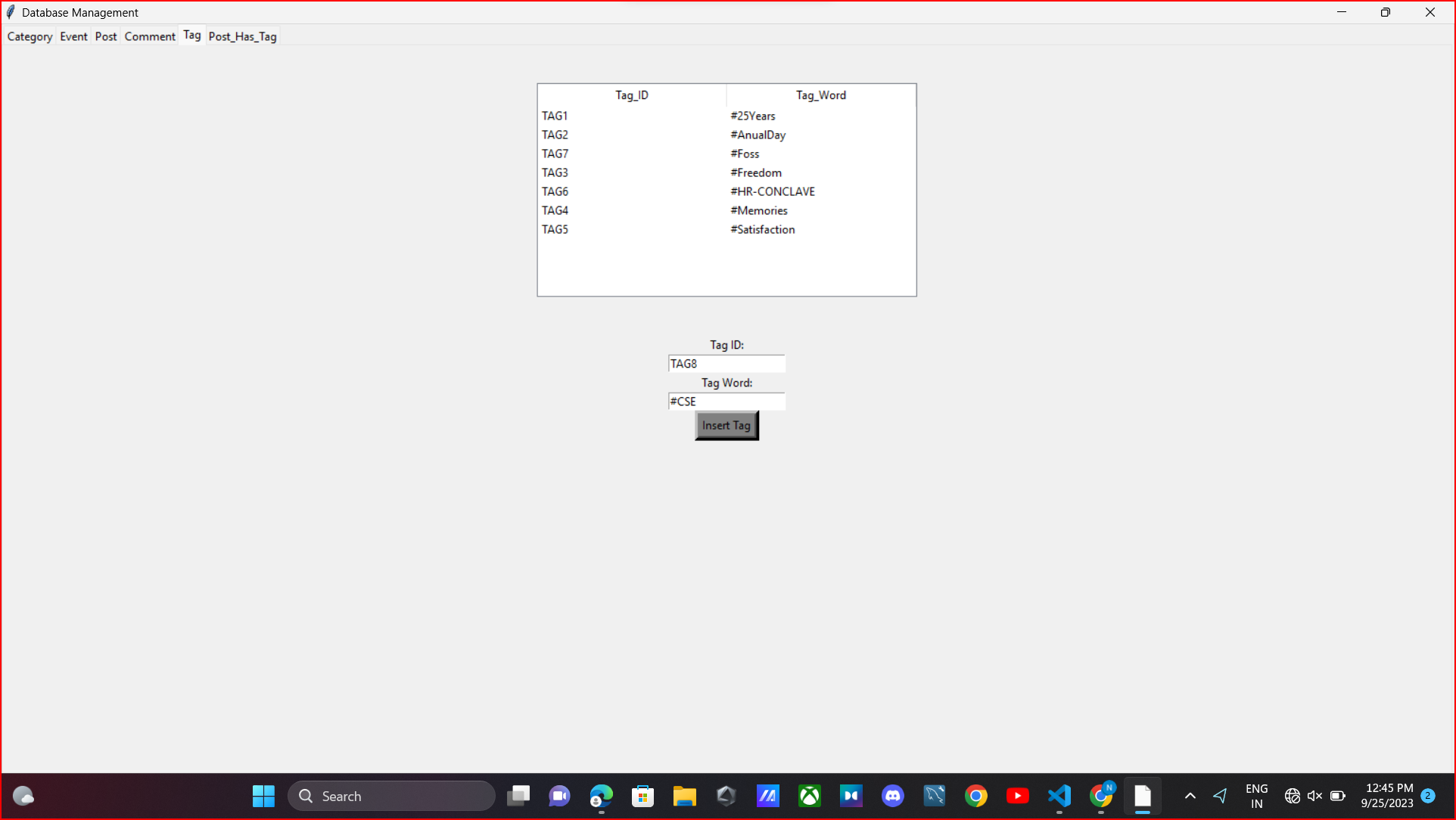


##### **5.4.2.3.2.Insert values**

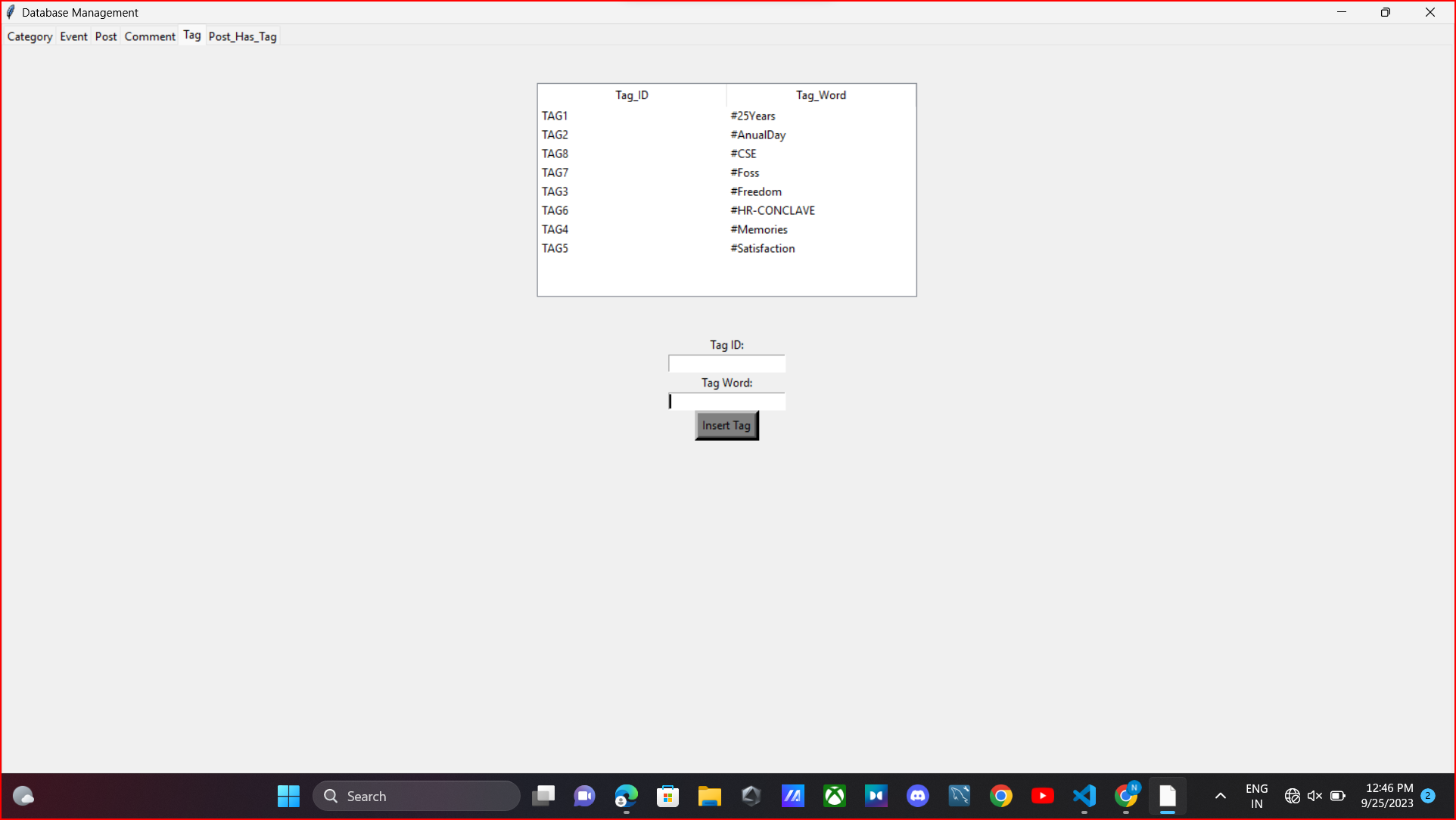
Before Insertion :



During Insertion :



After Insertion :



# **6.Implement SQL Queries to display in**

## **6.1.Mysql Workbench(using Mysql)**

### **6.1.1.Popular blog posts**

Query :

select concat("Popular Blog Posts"," are :"), Post\_Views, Post\_ID from Post where Post\_Views = (select max(Post\_Views) from Post);

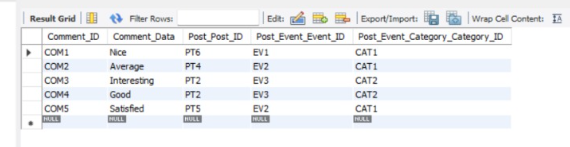
Output :



### **6.1.2.Manage comments**

Query : select \* from Comment;

Output :



### **6.1.3.Categorize posts**

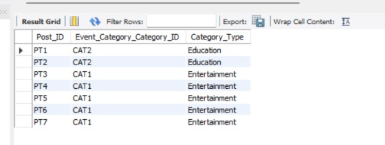
Query :

SELECT p.Post\_ID, p.Event\_Cat\_Cat\_ID, c.Category\_Type

FROM post AS p

JOIN Category AS c ON p.Event\_CatCat\_ID = c.Cat\_ID;

Output :



## **6.2.Visual Studio Code(using Python)**

### **6.2.1.Query:(Display Popular blog posts, manage comments, Categorize posts)**

import mysql.connector as c

from tabulate import tabulate

from termcolor import colored

conn = c.connect(

user='Pbl',

password='Pbl@123',

host='localhost',

database='mydb'

)

cursor = conn.cursor()

# Define your SQL queries

sql1 = 'SELECT concat("Popular Blog Posts", " are :"), Post\_Views, Post\_ID FROM Post WHERE Post\_Views = (SELECT MAX(Post\_Views) FROM Post);'

sql2 = 'SELECT p.Post\_ID, p.Event\_Category\_Category\_ID, c.Category\_Type FROM Post AS p LEFT JOIN Category AS c ON p.Event\_Category\_Category\_ID = c.Category\_ID;'

sql3 = 'SELECT \* FROM Comment;'

# Define a function to execute and print query results as a colorful table

def execute\_and\_print\_query(cursor, query):

cursor.execute(query)

result = cursor.fetchall()

print(colored("Query: ", "blue"), colored(query, "blue"))

headers = [colored(i[0], "green") for i in cursor.description]

colored\_results = [[colored(str(cell), "yellow") for cell in row] for row in result]

print(tabulate(colored\_results, headers=headers, tablefmt="fancy\_grid"))

print()

# Execute and print results for each query

execute\_and\_print\_query(cursor, sql1)

execute\_and\_print\_query(cursor, sql2)

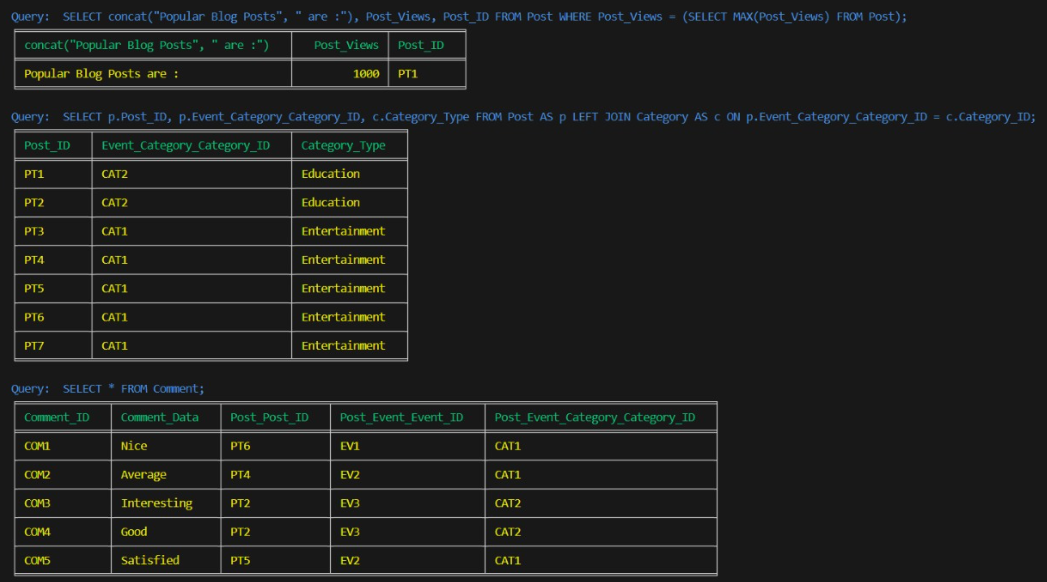
execute\_and\_print\_query(cursor, sql3)

# Close the cursor and connection

cursor.close()

conn.close()

### **6.2.2.Output**



# **7.Remarks**