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

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

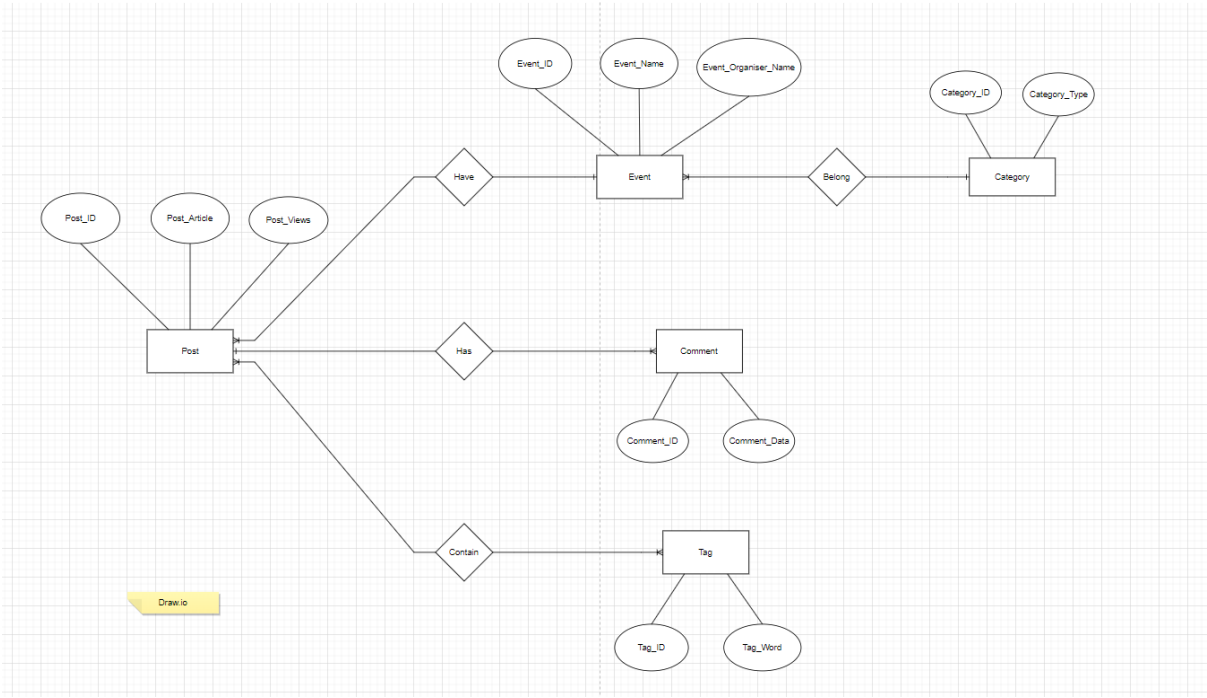
4.List of Relationships

- 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

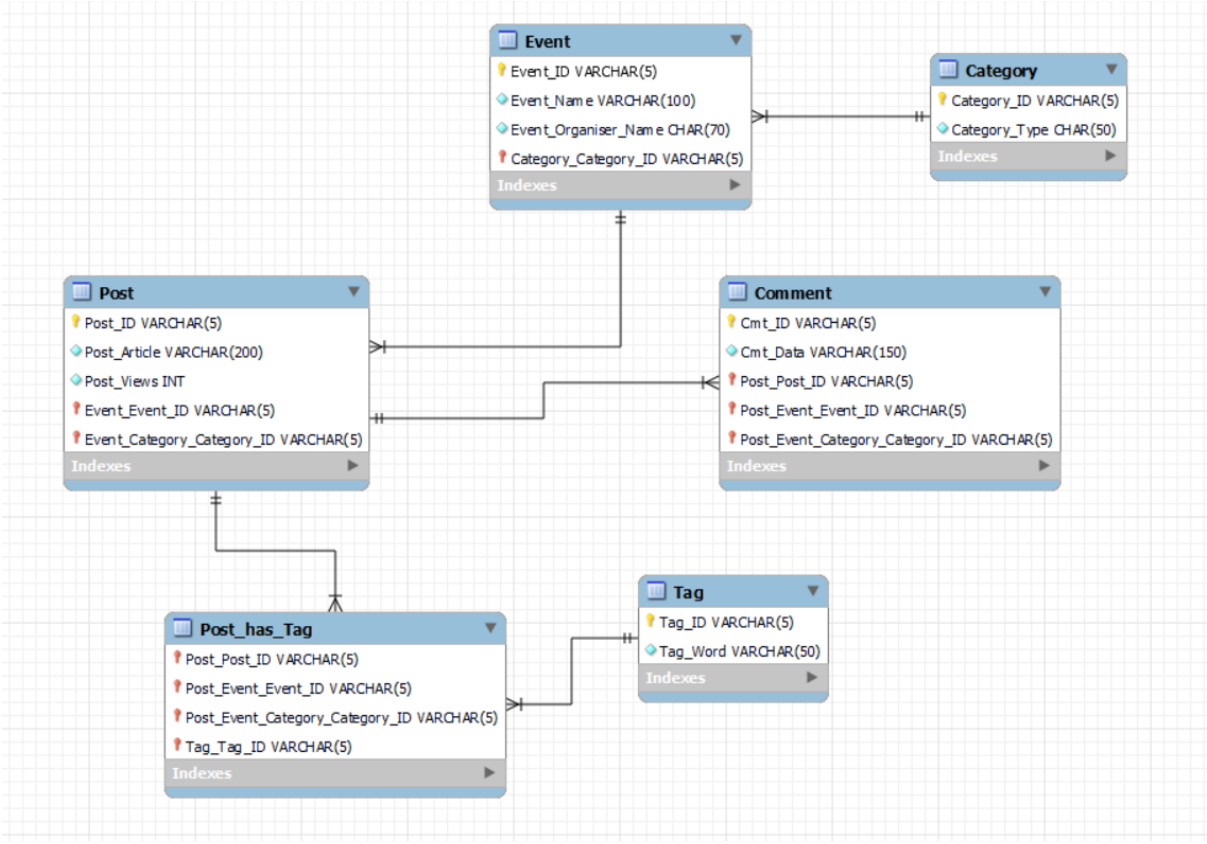
5.1.Design the logical view using ER Diagrams with tools

<u>ER Diagram</u> :



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_Z
ERO_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;
```

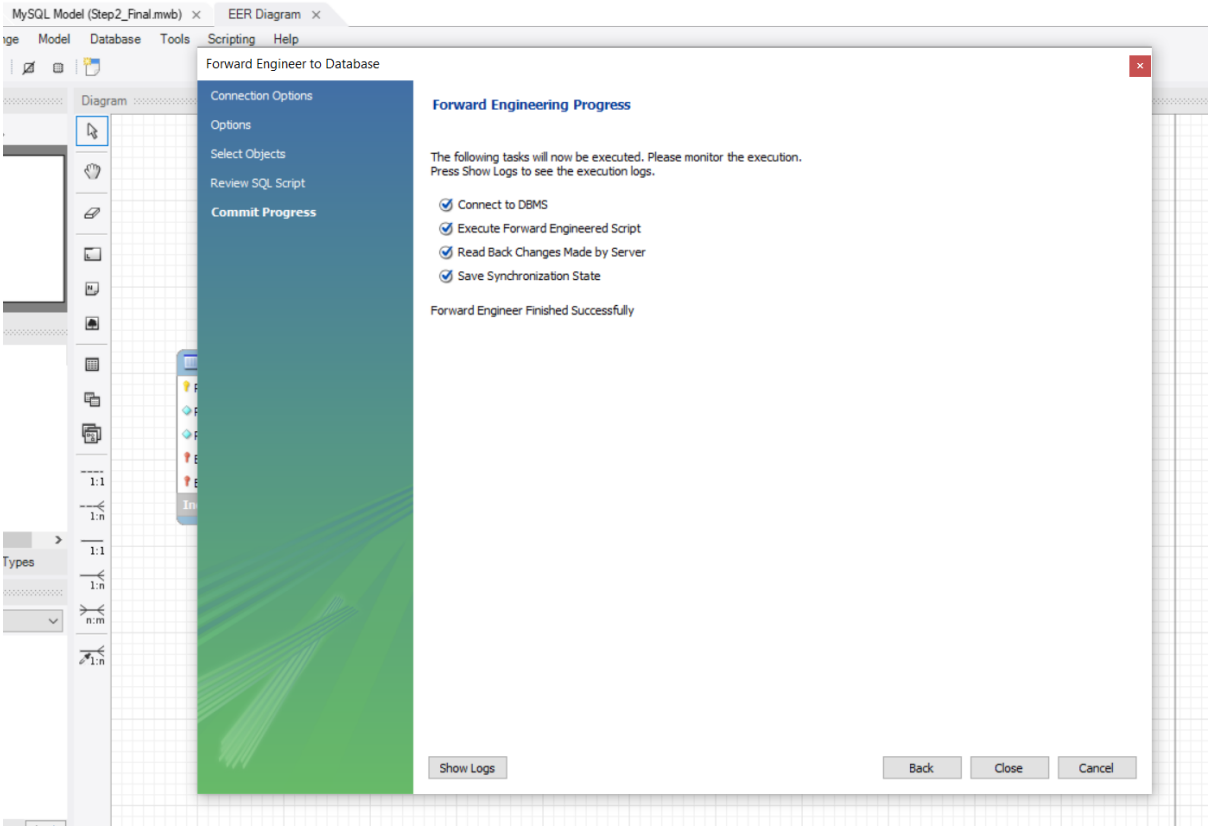
```
-- -----------------------------------------------------
-- Table `mydb`.`Tag`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Tag` (
  `Tag_ID` VARCHAR(5) NOT NULL,
  `Tag_Word` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`Tag_ID`),
  UNIQUE INDEX `Tag_ID_UNIQUE` (`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]

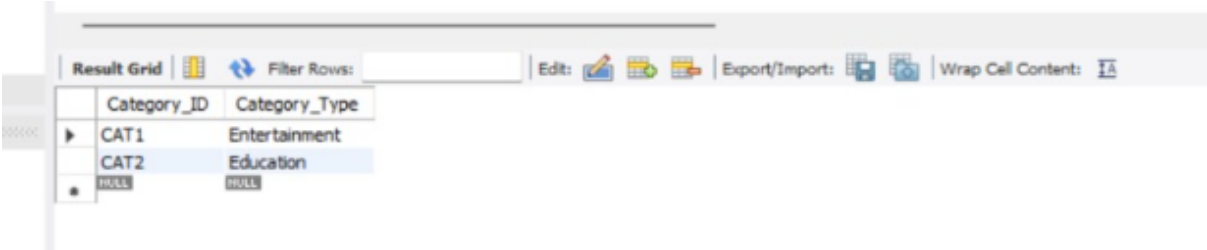
5.3.2.Tables in Database

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 :



Category_ID	Category_Type
CAT1	Entertainment
CAT2	Education
NULL	NULL

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 :

Event_ID	Event_Name	Event_Organiser_Name	Category_Category_ID
EV1	Silver Jublee	Anusha	CAT1
EV2	Anual Day	Hemanth	CAT1
EV3	Tech Fest	Yashwanth	CAT2
NULL	NULL	NULL	NULL

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 :

Post_ID	Post_Article	Post_Views	Event_Event_ID	Event_Category_Category_ID
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
NULL	NULL	NULL	NULL	NULL

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 :

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

Comment_ID	Comment_Data	Post_Post_ID	Post_Event_Event_ID	Post_Event_Category_Category_ID
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
NULL	NULL	NULL	NULL	NULL

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 :

Tag_ID	Tag_Word
TAG1	#25Years
TAG2	#AnualDay
TAG3	#Freedom
TAG4	#Memories
TAG5	#Satisfaction
NULL	NULL

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 :

Post_Post_ID	Post_Event_Event_ID	Post_Event_Category_Category_ID	Tag_Tag_ID
PT6	EV1	CAT1	TAG1
PT7	EV1	CAT1	TAG1
PT4	EV2	CAT1	TAG2
PT5	EV2	CAT1	TAG2
PT1	EV3	CAT2	TAG3
PT3	EV1	CAT1	TAG4
PT1	EV3	CAT2	TAG5
PT2	EV3	CAT2	TAG5
NULL	NULL	NULL	NULL

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 :

Result Grid | Filter Rows:

Category_ID	Category_Type
CAT1	Entertainment
CAT2	Education
CAT3	Spiritual
NULL	NULL

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 :

```
PS C:\ProgramData\MySQL\connect> & C:/ProgramData/anaconda3/python.exe c:/ProgramData/MySQL/connect/hi.py
Query: SELECT * FROM Category
```

Category_ID	Category_Type
CAT1	Entertainment
CAT2	Education
CAT3	Spiritual

5.4.1.2.Insert Query

Original table :

Category Table :

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

Category_ID	Category_Type
CAT1	Entertainment
CAT2	Education
NULL	NULL

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()
```

<u>Output</u> :

```
PS C:\ProgramData\MySQL\connect> & C:/ProgramData/anaconda3/python.exe c:/ProgramData/MySQL/connect/Insert_Display_Category.py
```

Category_ID	Category_Type
CAT1	Entertainment
CAT2	Education

Record added successfully:

Category_ID	Category_Type
CAT1	Entertainment
CAT2	Education
CAT3	Spiritual

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()
```

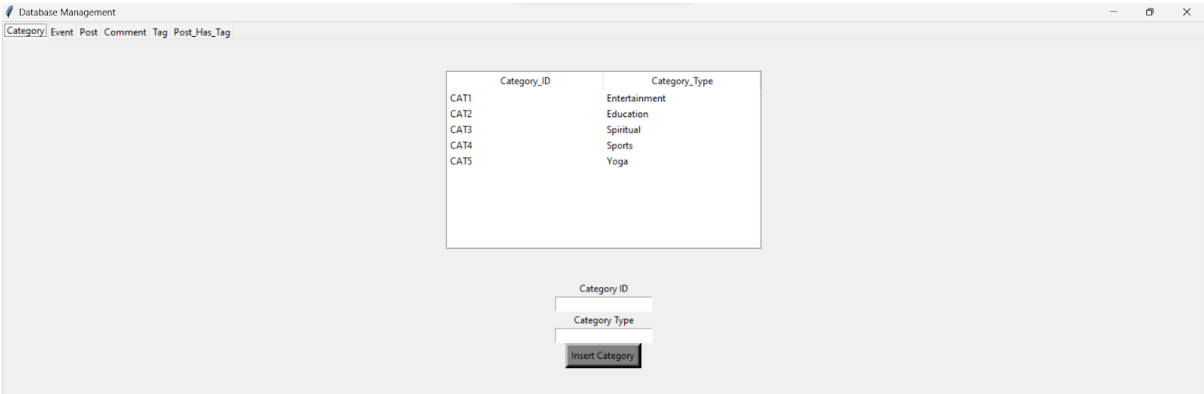
5.4.2.2.Code Repository Link

Git Hub - Code link

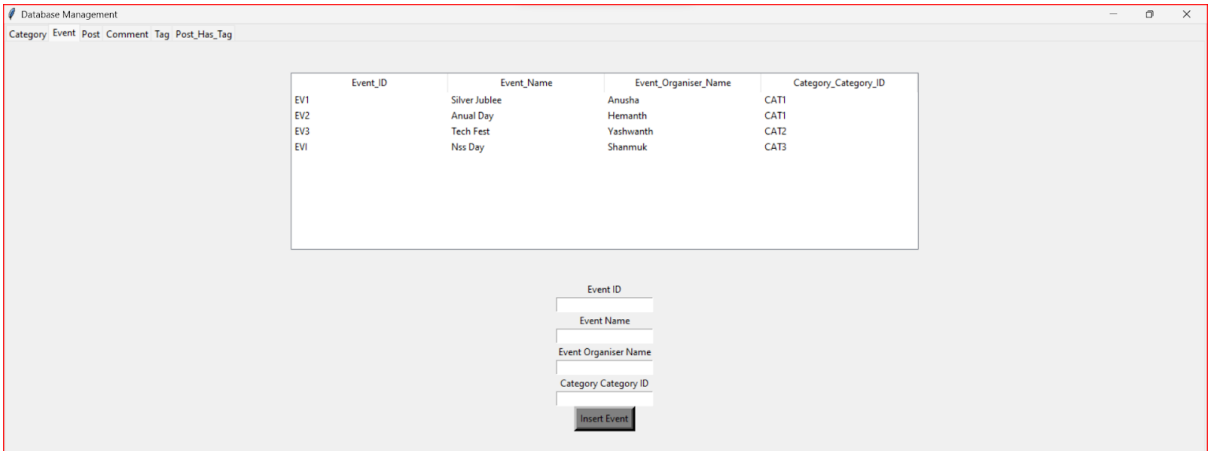
5.4.2.3.Output

5.4.2.3.1.Display Tables

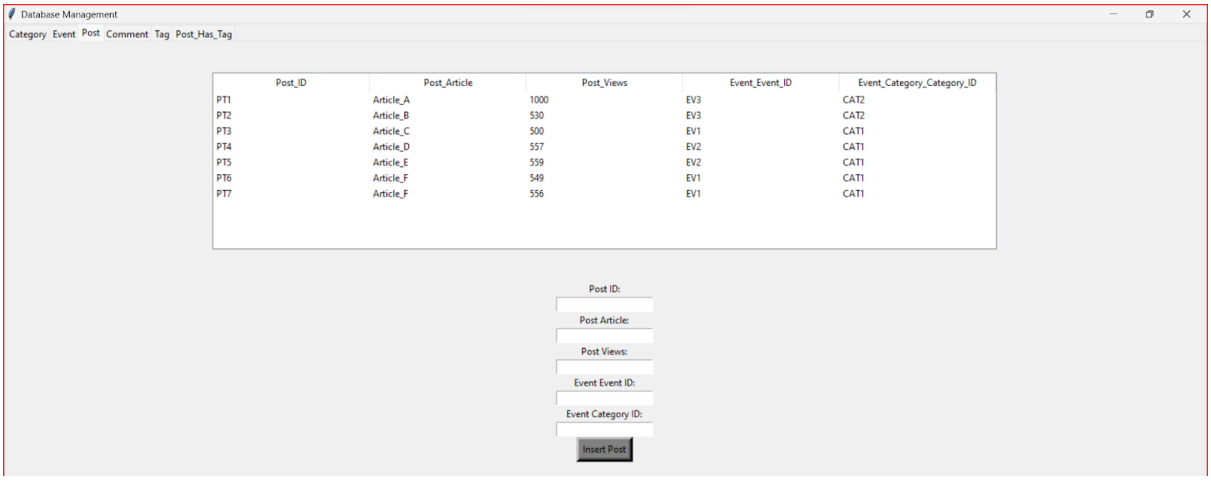
Category :



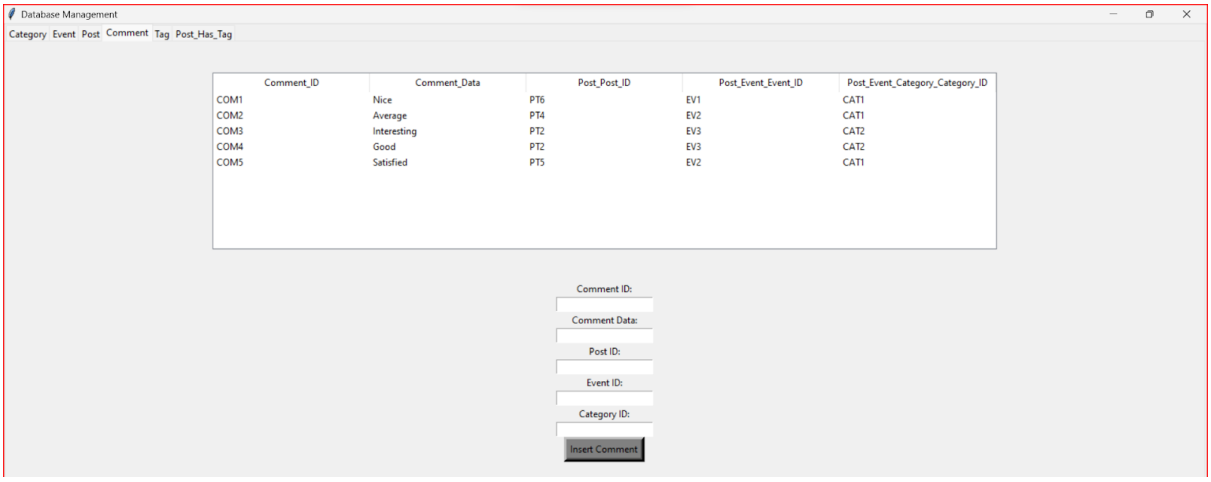
Event :



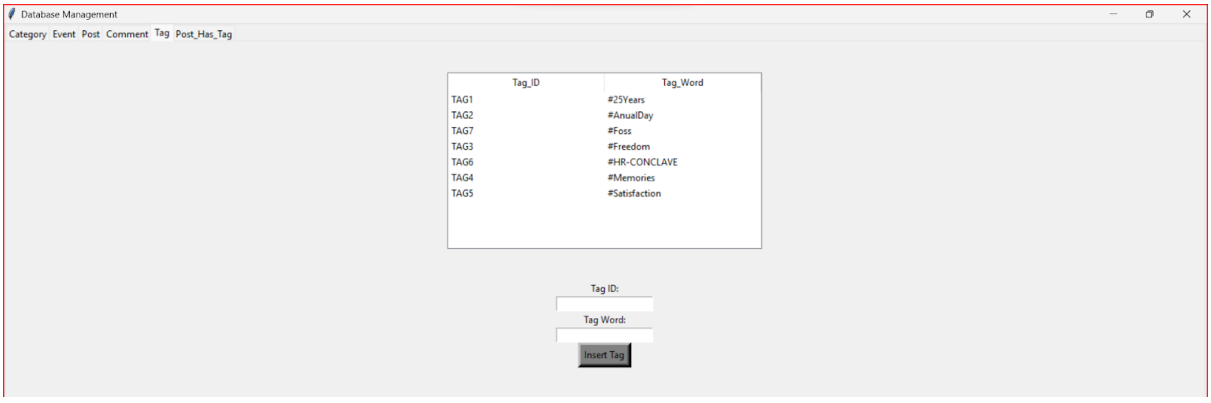
Post :



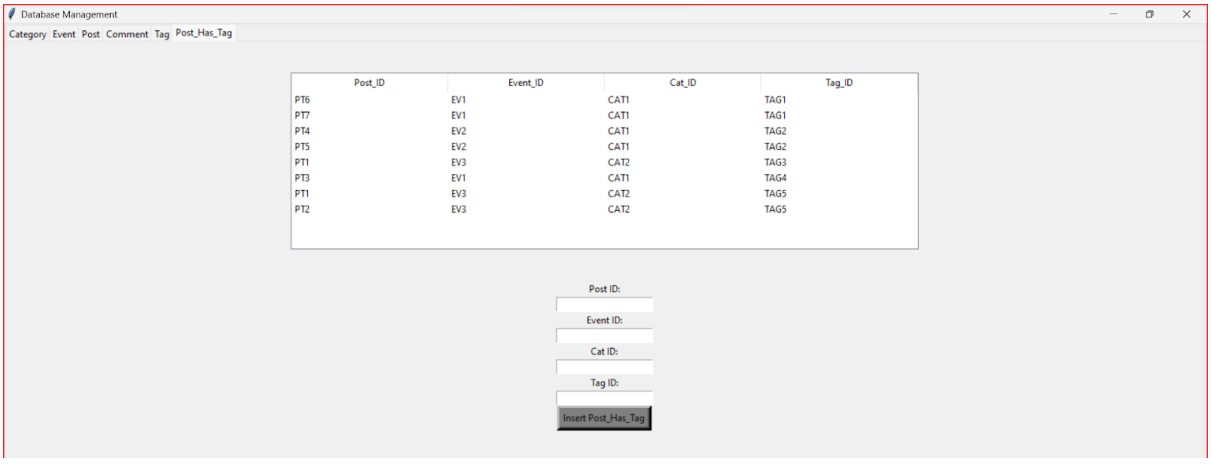
Comment :



Tag :

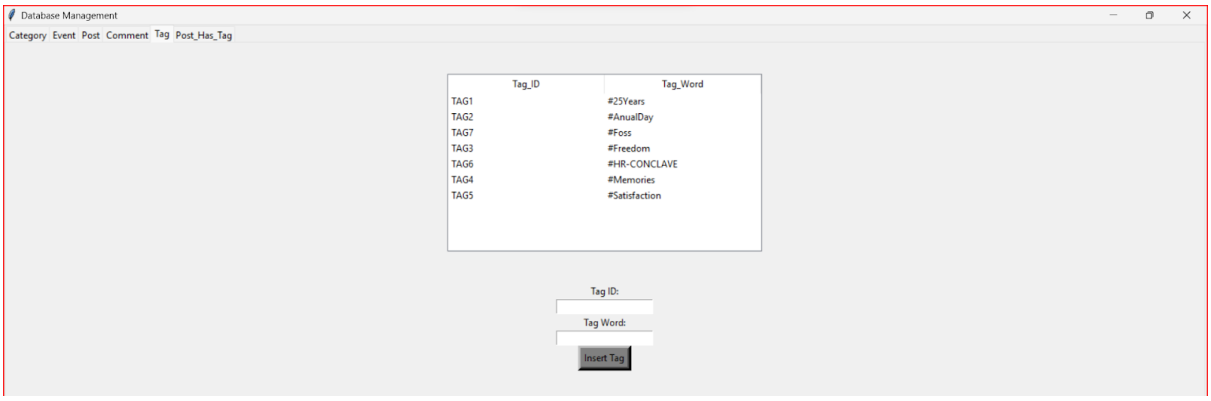


Post_Has_Tag :

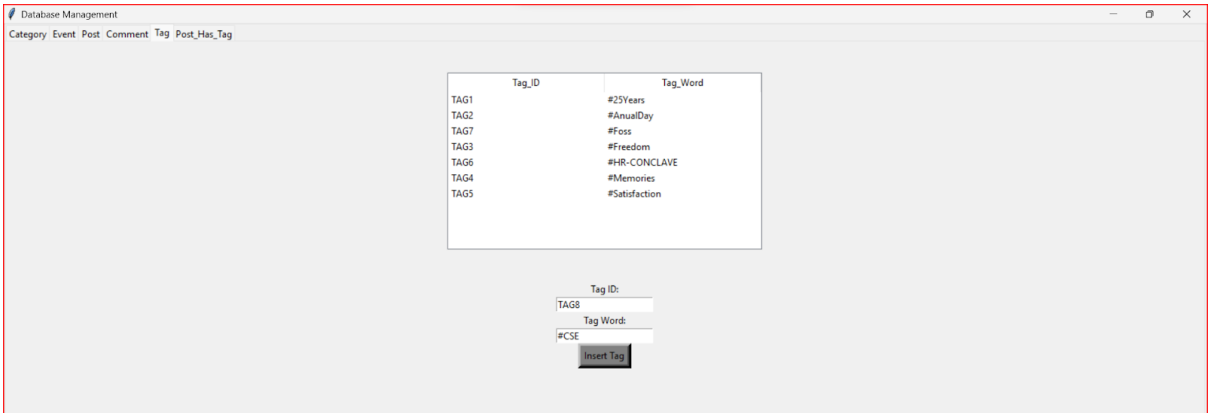


5.4.2.3.2.Insert values

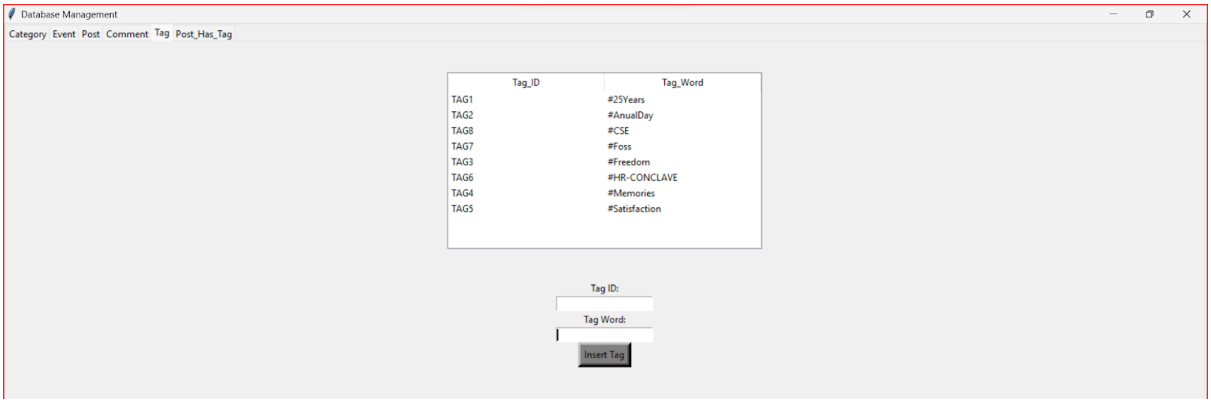
Before Insertion :



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 :

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

concat("Popular Blog Posts"," are :")	Post_Views	Post_ID
Popular Blog Posts are :	1000	PT1

6.1.2.Manage comments

Query : select * from Comment;

Output :

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

Comment_ID	Comment_Data	Post_Post_ID	Post_Event_Event_ID	Post_Event_Category_Category_ID
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
NULL	NULL	NULL	NULL	NULL

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 :

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Post_ID	Event_Category_Category_ID	Category_Type
PT1	CAT2	Education
PT2	CAT2	Education
PT3	CAT1	Entertainment
PT4	CAT1	Entertainment
PT5	CAT1	Entertainment
PT6	CAT1	Entertainment
PT7	CAT1	Entertainment

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

Query: SELECT concat("Popular Blog Posts", " are :"), Post_Views, Post_ID FROM Post WHERE Post_Views = (SELECT MAX(Post_Views) FROM Post);

concat("Popular Blog Posts", " are :")	Post_Views	Post_ID
Popular Blog Posts are :	1000	PT1

Query: 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;

Post_ID	Event_Category_Category_ID	Category_Type
PT1	CAT2	Education
PT2	CAT2	Education
PT3	CAT1	Entertainment
PT4	CAT1	Entertainment
PT5	CAT1	Entertainment
PT6	CAT1	Entertainment
PT7	CAT1	Entertainment

Query: SELECT * FROM Comment;

Comment_ID	Comment_Data	Post_Post_ID	Post_Event_Event_ID	Post_Event_Category_Category_ID
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

7.Remarks