

**A PROJECT
ON
CONCERT DATABASE MANAGEMENT
USING PYTHON**



**BITS Pilani-Dubai Campus, DIAC
(April 13, 2022)**

By-

Megha Manoj

2020A7PS0016U

Birla Institute of Technology and Science

D.I.A.C, Dubai, UAE

A Report

on

Concert Database

Management

Prepared for

Dr Pramod Gaur

Instructor in charge

By

Megha Manoj

Approved by

Dr Pramod Gaur

Instructor in charge

April 2022

Acknowledgement

I would like to express my gratitude towards Dr. Pramod Gaur, instructor in charge of the course 'Database systems' for giving this opportunity to design a database system using front-end software. Working on this project has been particularly useful in understanding the application of SQL in databases.

I would also like to thank Dr. Tamizhasan Periyasamy for clearing queries regarding various concepts applied in databases and teaching efficient ways to improve conceptual understanding of the topics applied.

TABLE OF CONTENTS

CONTENTS	PAGE NO.
i. Acknowledgement	2
1. Specification and purpose	4
1.1. Purpose	4
1.2. Specifications	4
2. ER Diagram	5
3. 3NF Decomposition and Table creations	6
3.1. 3 RD Normal Form	6
3.2. Creating tables	7
4. Populating tables	11
5. Queries	13
6. Constraints, procedures and triggers used	15
6.1. Constraints	15
6.2. Procedure	16
6.3. Trigger	17
7. Front end development using Python	18
7.1. MySQL Connector	19
7.2. Pandas	20
7.3. Tkinter	20
8. Source code	23
9. Program output	30

1.Specification and purpose

1.1. Purpose

Event management refers to the process of planning an event. This process involves division of work and a large amount of scheduling to ensure that the planned event goes smoothly.

In terms of concerts, this concept may apply from hiring artists for performance to selling tickets to the attending audience.

The designed database is a small model to simulate how data can be recorded to document all the required details for smooth management of events.

1.2. Specifications

The specifications included in the database are the following:

i. Use of a front-end software

For displaying and interacting with the user, the Python software has been used as a front-end. The Tkinter package in Python has been used for Graphical User Interface along with SQL Connect for connecting the front-end with the back-end

ii. Use of a back-end software

SQL Workbench has been used as a back-end in this project for storing the data in tabular format and preventing redundancy in the data

iii. Normalization using 3NF

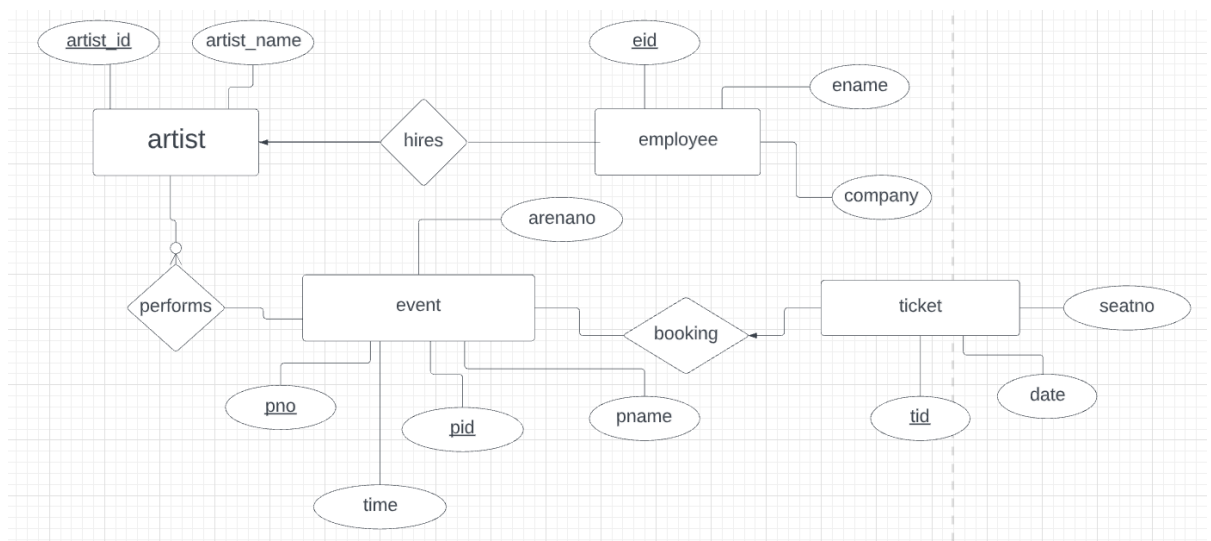
iv. Basic SQL queries

v. Constraints and procedure

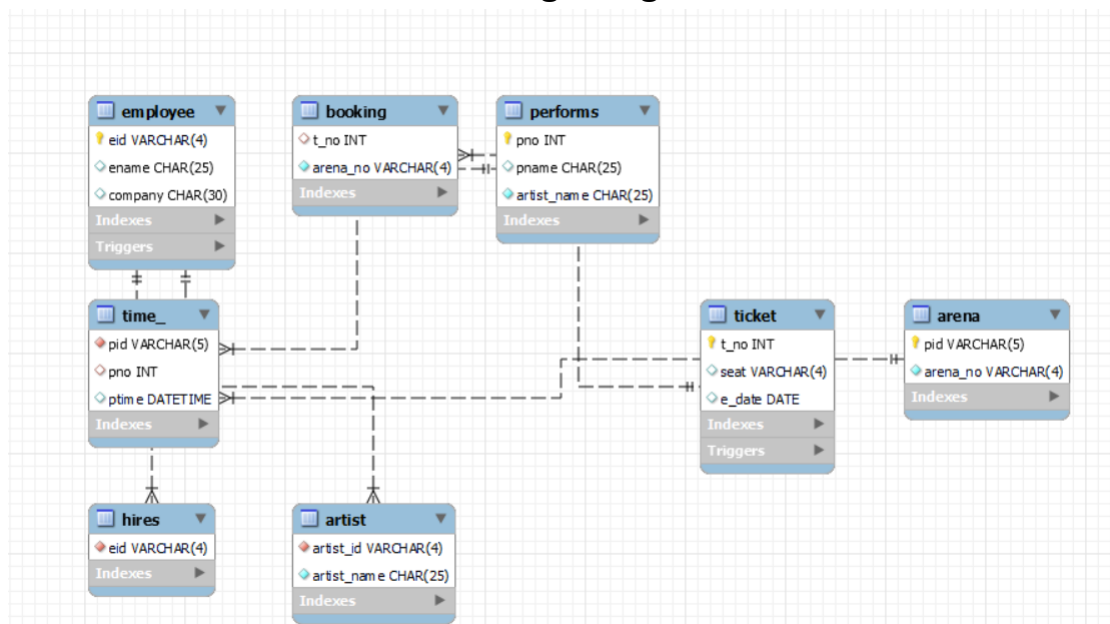
vi. 2 triggers

2.ER Diagram

The diagram given below is the ER diagram used for building the tables



And the following diagram shows how the tables have been linked to one another via the SQL ER diagram generated in workbench.



3.3NF decomposition and Table creations

3.1. 3rd normal form(3NF)

Normalization refers to the decomposing of tables to ensure that there is no data redundancy and all the data is stored in one place.

3NF refers to a relation in which –

- No non-primary attribute is transitively dependent.
- It contains an atomic value, there are no multivalued attributes
- There is no partial dependency where absence of one attribute can still determine the required results

3.2. Creating tables

The database, 'event_inc', created consists of 8 tables, each having certain constraints present on them to avoid clash of data. The 8 tables and the queries used in creating them are as follows:

i. employee

Table employee holds details regarding all the employees working on the event and which company they belong to. A primary key has been used to uniquely identify each employee.

SQL query-

```
create table employee(  
eid varchar(4) not null,  
ename char(25),  
company char(30),  
primary key(eid)  
);
```

ii. hires

Table hires has been created for the ease of accessing data regarding the hired artists. The table only has one attribute 'eid' present in it. To ensure that the data inserted is correct, a trigger 'artist_hired' has been created to automatically update the table .

SQL query-

```
create table hires(  
eid varchar(4) not null unique,  
primary key(eid),  
foreign key(eid) references employee(eid),
```


check (eid like 'A%'));

iii. artist

This table consists of the stage names of the performing artists and their employee ids. The purpose of this table is to avoid confusion regarding performances where multiple artists participate.

SQL Query-

**create table artist(
artist_id varchar(4) not null unique ,
artist_name char(25) not null,
foreign key(artist_id) references hires(eid));**

iv. event

Table event holds information regarding the performances scheduled, the name, location and time at which the concert will take place. The table has been decomposed using 3NF to create a neater visualization of the data such that no repetition occurs when queries regarding the events are executed. The tables formed from decomposition are as follows:



a. arena

Table arena consists of a unique performance id('pid') and the arena number at which the performance takes place.

SQL Query-

```
CREATE TABLE arena(
pid varchar(5) not null,
arena_no varchar(4) not null,
primary key(pid));
```

b. performs

Table performs has been used to show the relationship between the artist and the event, namely for understanding which artist will perform at which concert.

SQL Query-

```
CREATE TABLE performs(
pno int(4),
pname char(25),
artist_name CHAR(25) not null,
```

primary key(pno));

c. time_

This particular table uses 2 foreign key attributes- pid and pno, to establish a relation to the other 2 tables. The purpose of this table is to store the date and time at which each concert will occur.

SQL Query-

```
create table time_(  
pid varchar(5) not null,  
pno int(4),  
ptime datetime,  
foreign key(pno) references performs(pno),  
foreign key(pid) references arena(pid));
```

v. booking

booking is a relationship table to indicate that a customer has purchased tickets for a particular event. It stores the ticket number and corresponding location where the concert will take place.

A trigger has been created to automatically update this table each time a customer purchases a ticket.

SQL Query-

```
create table booking(  
arena_no varchar(4) not null,  
t_no int(5),  
foreign key(t_no) references ticket(t_no));
```

vi. ticket

The ticket table contains records on the purchased tickets. Each customer is represented by a unique ticket number and assigned a seat number for the particular arena they will be attending the concert in on the corresponding day.

SQL Query-

```
create table ticket(  
t_no int(5),  
seat varchar(4) unique ,  
e_date date,  
primary key(t_no));
```

4. Populating tables

Data has been inserted into the tables using the query **'insert into table_name values();'**.

The insertion applies to all the tables created except 'hires' and 'booking' where insertion takes place via trigger.

Furthermore, to make the front end interactive, customers can insert their ticket information into the table 'ticket' by entering the details into a pop-up box created using GUI in python.

4.1. Table employee

```
insert into employee(eid, ename, company)  
values('A12','CHRISTOPHER CHAN','JYP'),  
( 'A13','FELIX LEE','JYP'),  
( 'A15','JEONGIN YANG','JYP'),  
( 'A30','IRENE SEL','SM'),
```

```
('A82','JAY PARK','BIGHIT LAB'),  
('M23','SILVIA KELP','MG SERVICES'),  
('T82','JAKE KIM','MGTECH'),  
('T21','KELLY HAN','Y TECHNO'),  
('A45','HAN JISUNG','JYP'),  
('M67','HARRY POTTS','K MGMT');
```

4.2. Table artist

```
insert into artist(artist_id, artist_name)  
values('A12','STRAY KIDS'),  
('A13','STRAY KIDS'),  
('A15','STRAY KIDS'),  
('A45','STRAY KIDS'),  
('A30','IRENE'),  
('A82','JAY');
```

4.3. Table arena

```
insert into arena values('S9213', 'EG13'),  
('X902','DF21'),  
('J872','EG13'),  
('D452','BA89'),  
('G481','DF21');
```

4.4. Table performs

```
insert into performs(pno, pname, artist_name)  
values(2132,'GODS MENU','STRAY KIDS'),  
(2534,'MONSTER','IRENE'),  
(3421,'BACK DOOR', 'STRAY KIDS'),  
(5642,'GOBLIN: MUSICAL', 'IRENE'),  
(7639,'MIRACLE IN THE WOOD','JAY');
```

4.5. Table time_

```
insert into time_(pid,pno, ptime)
values('S9213', 2132, '22-10-05 16:04' ),
('X902', 2534 , '22-10-05 18:30'),
('J872',3421,'22-10-05 16:45'),
('D452', 5642,'22-10-04 21:15' ),
('G481', 7639, '22-10-06 17:22');
```

4.6. Table ticket

```
insert into ticket(t_no, seat ,e_date)
values(30085, 'E009','22-10-05' ),
(30087, 'E049','22-10-05'),
(30089, 'A453', '22-10-04'),
(30030, 'D039','22-10-06');
```

5.Queries

The following queries have been created for interacting with the database:

- 5.1. Display all the artists stage names and their actual names by using subquery

Query:

```
select a.artist_name, e.ename
from employee e , artist a
where a.artist_id=e.eid
and e.eid in (select * from hires)
order by a.artist_name;
```

- 5.2. Display all the recent events scheduled to take place
Query:

```
create view even_ as  
  select p.pname, p.artist_name, t.ptime  
  from performs p, time_ t  
  where p.pno=t.pno  
  group by p.pname;
```

- 5.3. Display the event location and corresponding ticket details associated with the venue for all customers using inner join
Query:

```
select t.t_no, t.seat ,t.e_date, b.arena_no  
from ticket t  
  inner join booking b  
on t.t_no=b.t_no group by t.t_no;
```

- 5.4. Display details of all tables
Queries:

```
select * from employee;
```

```
select * from hires;
```

```
select * from artist;
```

```
select * from arena;
```

```
select * from performs;
```

```
select * from time_;
```

```
select * from ticket;
```

```
select * from booking;
```

5.5. Insert records into ticket table

Query:

```
Insert into ticket values(tno, seatno, date);
```

Note: tno, seatno, date are inputs from the user which is inserted into the table using a procedure.

6. Constraints, procedures and triggers used

6.1. Constraints

SQL constraints have been used in the project to ensure that the data does not overlap. It has been helpful in maintaining a limit on the type of data inserted and keeping records distinctive.

The constraints used are as follows:

a. Not null

To ensure the values in the records are not null at important columns.

b. Unique

To distinguish the records inserted.

c. Foreign key

To relate records in one table to another and ensure that the data is in sync amongst these tables.

d. Primary key

It has the same purpose as 'unique'. Since there can only be one primary key in each table, 'unique' has been used wherever an extra unique attribute is required.

e. Check

Check has been used to ensure that the data inserted matches certain conditions required in the table.

6.2. Procedures

The concert database uses procedures for inserting data into the back-end for table 'ticket' from the front-end interaction with the user. It has been helpful in avoiding errors in the front-end code. The procedure used is as follows:

```
DELIMITER $$  
create procedure ticket_entry(in tno int(4), seat  
varchar(4), edate date)  
begin  
    insert into ticket values(tno,seat, edate);  
end$$  
DELIMITER ;
```

Note: Once the data is inserted via the procedure, a trigger inserts the required details into the table 'booking'.

6.3. Triggers

The database uses 2 triggers for automatically inserting data into 2 tables. The triggers created are as follows:

a. Trigger artist_hired

The trigger updates table 'hires' with the artists employee ids when a new artist is hired by the company.

Query:

```
DELIMITER $$  
CREATE TRIGGER artist_hired AFTER INSERT ON  
employee  
FOR EACH ROW  
BEGIN  
    if (new.eid like 'A%') then  
        insert into hires values(new.eid);  
    end if;  
END $$  
DELIMITER ;
```

b. Trigger BOOKING_DETAILS

This particular trigger has been created for inserting the ticket number and arena number into table 'booking' based on the seat number from the table 'ticket'

Query:

```

DELIMITER $$
CREATE TRIGGER BOOKING_DETAILS AFTER INSERT
ON ticket FOR EACH ROW BEGIN declare ar varchar(4)
; if (new.seat like 'E%') then set ar='EG13'; ELSE IF
(new.seat like 'A%') then set ar='BA89'; ELSE IF
(new.seat like 'D%') then set ar='DF21'; END IF; END
IF; END IF; insert into booking values(new.t_no, ar);
END $$
DELIMITER ;

```

7. Front end development using Python

The front-end of this concert management system has been applied using the Python software for ease of access and application. The features of the developed front-end are as follows:

- i. The queries are executed using functions from the imported packages
- ii. A menu is displayed when the program is run using a while loop with asks the user to enter an index to carry out the query associated with the menu option.
- iii. A function created in one of the menu options (option 5) displays a pop-up window to enter details. When this window is closed, the program returns to the while loop menu to continue serving the user.

The program creates the tables required with using functions from various packages. The packages used to build the front-end as follows:

7.1. MySQL Connector-

MySQL Connector is a package which allows Python to connect to the MySQL database and interact with it by using Application Programming interface.

The commands used for this purpose are given below:

```
import mysql.connector
import pandas as pd
from tkinter import *

mydb = mysql.connector.connect(
    host="localhost",
    user="root",|
    password="root"
)
print("WELCOME TO EXA EVENTS")

c=mydb.cursor()
#c.execute("create database event_inc")
'''
c.execute("show databases")
for x in c:
    print(x)
'''
c.execute("use event_inc")
```

a. Mysql.connector.connect()

To create a connection to the MySQL server and receive a MySQLConnection object to work with.

b. .cursor()

An object which fetches data and executes queries.

c. .execute()

A function to execute SQL queries and return the corresponding results.

```
res8=c.fetchall()
p8=pd.DataFrame(res8,columns=["t_no", "arena_no"])
```

d. .fetchall()

.fetchall() fetches all the rows of the executed queries

e. .callproc()

The function calls the particular procedure to be used. The parameters of this function are- name of procedure and a tuple of the arguments to be passed in the procedure.

7.2. Pandas

Pandas is a software library which is used in data manipulation and analysis. Here the function. DataFrame() has been used from the package to display the data in a tabular formate. The parameters of this function are- the fetched results and an array of the column titles.

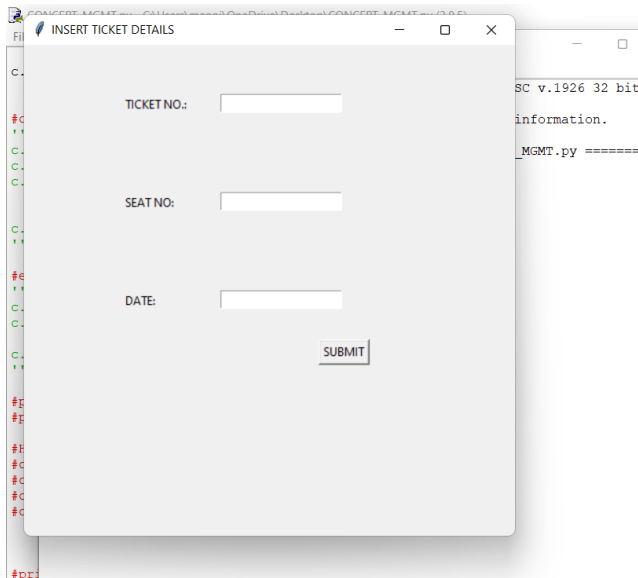
```
WELCOME TO EXA EVENTS
1- RECENT EVENTS
2-MEMBERS OF EACH GROUP
3-EVENT LOCATION AND TICKET DETAILS
4-VIEW TABLES
5-INSERT NEW TICKET DETAILS
0-EXIT
choice? 4
TABLES:
1-EMPLOYEE
2-HIRES
3-ARTIST
4-ARENA
5-PERFORMS
6-TIME
7-TICKET
8-BOOKING

enter table no. -4
      pid arena_no
0   D452    BA89
1   G481    DF21
2   J872    EG13
3   S9213   EG13
4   X902    DF21
continue viewing? y/n:y
1- RECENT EVENTS
.
```

7.3. Tkinter

It is a standard Graphical User Interface (GUI) for python which has various widgets which are highly applicable for making the code more interactive. For this project,

The package has been used to create a pop-up window where the user can enter ticket details into the text fields.



The data entered will be inserted into the 'ticket' table upon clicking submit. The functions used from this package are as follows:

```
class Window:
    def __init__(self, win):
        self.l1= Label(w, text='TICKET NO.: ')
        self.l2= Label(w, text='SEAT NO: ')
        self.l3= Label(w, text='DATE: ')
        self.t1=Entry()
        self.t2=Entry()
        self.t3=Entry()

        self.btn=Button(w, text='submit')

        self.l1.place(x=100, y=50)
        self.t1.place(x=200, y=50)
        self.l2.place(x=100, y=150)
        self.t2.place(x=200, y=150)
        self.l3.place(x=100, y=250)
        self.t3.place(x=200, y=250)
        self.b1=Button(win, text='SUBMIT', command=self.submit)
        self.b1.place(x=300, y=300)

    def submit(self):
        s1=int(self.t1.get())
        s2=self.t2.get()
        s3=self.t3.get()
        self.t1.delete(0, 'end')
        self.t2.delete(0, 'end')
        self.t3.delete(0, 'end')
        args=(s1,s2,s3)

        c.callproc("ticket_entry", args)
        print("inserted")

w=Tk()
mywin=Window(w)
w.title('INSERT TICKET DETAILS')
w.geometry("500x500+10+10")
w.mainloop()
```

a. Label()

It is a widget to place text and images in the window

b. Button()

It is used to create a button in the pop-up. The button can have text displayed on it.

c. Entry()

It is used to enter single line text into the text fields created by the same function.

d. .place()

To place a widget at a certain position in the window by mentioned coordinates.

e. .get()

It is used to retrieve the data entered into the text fields.

f. .delete()

The function has been implemented to clear the text fields after the data gets submitted.

g. .title()

It gives the created window a title at the header.

h. .geometry()

.geometry() sets the dimensions of the window created.

i. Tk()

It is the root for creating a window using Tkinter. All the other functions depend on Tk().

8.Source code

```
import mysql.connector
import pandas as pd
from tkinter import *

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="root"
)

print("WELCOME TO EXA EVENTS")

c=mydb.cursor()

c.execute("create database event_inc")

c.execute("use event_inc")

#employee -----
c.execute("create table employee(eid varchar(4) not null,ename
char(25),company char(30),primary key(eid))")

c.execute("insert into employee(eid, ename, company)
values('A12','CHRISTOPHER CHAN','JYP'),('A13','FELIX
LEE','JYP'),('A15','JEONGIN YANG','JYP'),('A30','IRENE
SEL','SM'),('A82','JAY PARK','BIGHIT LAB'),('M23','SILVIA KELP','MG
SERVICES'),('T82','JAKE KIM','MGTECH'),('T21','KELLY HAN','Y
TECHNO'),('A45','HAN JISUNG','JYP'),('M67','HARRY POTTS','K MGMT')")

#HIRES-----

c.execute("create table hires( eid varchar(4) not null
unique,primary key(eid),foreign key(eid) references employee(eid),
check (eid like 'A%')  )")

c.execute("insert into hires(eid) values ('A12'),('A13'),
('A15'),('A30'), ('A82'),('A45')")
```



```

#ARTIST-----
c.execute("create table artist(artist_id varchar(4) not null unique
, artist_name char(25) not null,foreign key(artist_id) references
hires(eid))")

c.execute("insert into artist(artist_id, artist_name)
values('A12','STRAY KIDS'),('A13','STRAY KIDS'),('A15','STRAY
KIDS'),('A45','STRAY KIDS'),('A30','IRENE'),('A82','JAY')")

#TRIGGER TO INSERT ARTIST ID INTO HIRES-----
c.execute("CREATE TRIGGER artist_hired AFTER INSERT ON employee FOR
EACH ROW BEGIN if (new.eid like'A%') then insert into hires
values(new.eid); end if;END ")

#ARENA-----
c.execute("CREATE TABLE arena(  pid varchar(5) not null,arena_no
varchar(4) not null,primary key(pid))")

c.execute("insert into arena values('S9213',
'EG13'),('X902','DF21'),('J872','EG13'),('D452','BA89'),('G481','DF2
1')")

#performs-----
c.execute("CREATE TABLE performs(pno int(4),pname
char(25),artist_name CHAR(25) not null,primary key(pno))")

c.execute("insert into performs(pno, pname, artist_name)
values(2132,'GODS MENU','STRAY
KIDS'),(2534,'MONSTER','IRENE'),(3421,'BACK DOOR', 'STRAY
KIDS'),(5642,'GOBLIN: MUSICAL', 'IRENE'),(7639,'MIRACLE IN THE
WOOD','JAY')")

#time_-----
c.execute("create table time_(pid varchar(5) not null,pno
int(4),ptime datetime,foreign key(pno) references
performs(pno),foreign key(pid) references arena(pid))")

c.execute("insert into time_(pid,pno, ptime) values('S9213', 2132,
'22-10-05 16:04' ),('X902', 2534 ,'22-10-05
18:30'),('J872',3421,'22-10-05 16:45'),('D452', 5642,'22-10-04
21:15' ),('G481', 7639, '22-10-06 17:22')")

```

```

#ticket-----
c.execute("create table ticket(t_no int(5),seat varchar(4) unique
,e_date date, primary key(t_no))")

c.execute("insert into ticket(t_no, seat ,e_date) values(30085,
'E009','22-10-05' ),(30087, 'E049','22-10-05'),(30089, 'A453', '22-
10-04'),(30030, 'D039','22-10-06')")

#booking-----

c.execute("create table booking(t_no int(5),arena_no varchar(4) not
null, foreign key(t_no) references ticket(t_no) )")

c.execute(" insert into booking values(30085, 'EG13'),(30087, 'EG13'
),(30089, 'BA89'),(30030, 'DF21')")

#booking_Trigger-----

c.execute("CREATE TRIGGER BOOKING_DETAILS AFTER INSERT ON ticket FOR
EACH ROW BEGIN declare ar varchar(4) ; if (new.seat like 'E%') then
set ar='EG13'; ELSE IF (new.seat like 'A%') then set ar='BA89'; ELSE
IF (new.seat like 'D%') then set ar='DF21'; END IF; END IF; END IF;
insert into booking values(new.t_no, ar); END")

#procedure-----

c.execute("create procedure ticket_entry(in tno int(5), seat
varchar(4), edate date) begin insert into ticket values(tno,seat,
edate); end")

#MENU-----

x="y"

while(x=="y"):

    print("1- RECENT EVENTS")

    print("2-MEMBERS OF EACH GROUP")

    print("3-EVENT LOCATION AND TICKET DETAILS")

    print("4-VIEW TABLES")

    print("5-INSERT NEW TICKET DETAILS")

    print("0-EXIT")

    n=int(input("choice? "))

    if n==2:

        #subquery

```

```

c.execute("select a.artist_name, e.ename from employee e , artist a
where a.artist_id=e.eid and e.eid in (select * from hires) order by
a.artist_name")

e=c.fetchall()

print(pd.DataFrame(e,columns=["artist name", "member
name"]))

elif n==1:

    #select query as view

    c.execute("create view even_ as select p.pname,
p.artist_name, t.ptime from performs p, time_ t where p.pno=t.pno
group by p.pname")

    c.execute("select * from even_")

    e=c.fetchall()

    print(pd.DataFrame(e,columns=["artist name", "performance
name","performance time"]))

elif n==3:

    #inner join

    c.execute(" select t.t_no, t.seat ,t.e_date, b.arena_no
from ticket t inner join booking b on t.t_no=b.t_no group by t.t_no
")

    e=c.fetchall()

    print(pd.DataFrame(e,columns=["t_no", "seat" ,"e_date",
"arena_no" ]))

elif n==4:

    #display tables

    print("TABLES:\n1-EMPLOYEE\n2-HIRES\n3-ARTIST\n4-ARENA\n5-
PERFORMS\n6-TIME\n7-TICKET\n8-BOOKING\n")

    a=int(input("enter table no. -"))

    if a==1:

        c.execute("select * from employee")

        res=c.fetchall()

        p1=pd.DataFrame(res,columns=["eid","ename","company"])

        print(p1)

```

```

elif a==2:
    c.execute("select * from hires")
    res2=c.fetchall()
    p2=pd.DataFrame(res2,columns=["eid"])
    print(p2)

elif a==3:
    c.execute("select * from artist")
    res3=c.fetchall()
    p3=pd.DataFrame(res3,columns=["artist_id",
"artist_name"])
    print(p3)

elif a==4:
    c.execute("select * from arena")
    res4=c.fetchall()
    p4=pd.DataFrame(res4,columns=["pid", "arena_no"])
    print(p4)

elif a==5:
    c.execute("select * from performs")
    res5=c.fetchall()
    p5=pd.DataFrame(res5,columns=["pno", "pname",
"artist_name"])
    print(p5)

elif a==6:
    c.execute("select * from time_")
    res6=c.fetchall()
    p6=pd.DataFrame(res6,columns=["pid","pno", "ptime"])
    print(p6)

```

```

elif a==7:

    c.execute("select * from ticket")
    res7=c.fetchall()

    p7=pd.DataFrame(res7,columns=["t_no", "seat" ,"e_date"])
    print(p7)


elif a==8:

    c.execute("select * from booking")
    res8=c.fetchall()

    p8=pd.DataFrame(res8,columns=["t_no", "arena_no"])
    print(p8)

elif n==5:

    class Window:

        def __init__(self, win):

            self.l1= Label(w, text='TICKET NO.: ')
            self.l2= Label(w, text='SEAT NO:')
            self.l3= Label(w, text='DATE:')
            self.t1=Entry()
            self.t2=Entry()
            self.t3=Entry()


            self.btn=Button(w, text='submit')


            self.l1.place(x=100, y=50)
            self.t1.place(x=200, y=50)
            self.l2.place(x=100, y=150)
            self.t2.place(x=200, y=150)
            self.l3.place(x=100, y=250)
            self.t3.place(x=200, y=250)

            self.b1=Button(win, text='SUBMIT',
command=self.submit)

            self.b1.place(x=300, y=300)

```

```

        def submit(self):
            s1=int(self.t1.get())
            s2=self.t2.get()
            s3=self.t3.get()
            self.t1.delete(0, 'end')
            self.t2.delete(0, 'end')
            self.t3.delete(0, 'end')
            args=(s1,s2,s3)

            c.callproc("ticket_entry", args)
            print("inserted")

w=Tk()

mywin=Window(w)

w.title('INSERT TICKET DETAILS')

w.geometry("500x500+10+10")

w.mainloop()


elif n==0: break
x=input("continue viewing? y/n:")

```

9. Program output

The screenshots below show the output of the program upon execution.

```
===== RESTART: C:\Users\manoj\OneDrive\Desktop\CONCERT_MGMT.py =====
WELCOME TO EXA EVENTS
1- RECENT EVENTS
2-MEMBERS OF EACH GROUP
3-EVENT LOCATION AND TICKET DETAILS
4-VIEW TABLES
5-INSERT NEW TICKET DETAILS
0-EXIT
choice? 1
      artist name performance name    performance time
0      GODS MENU      STRAY KIDS 2022-10-05 16:04:00
1      MONSTER        IRENE 2022-10-05 18:30:00
2      BACK DOOR      STRAY KIDS 2022-10-05 16:45:00
3      GOBLIN: MUSICAL      IRENE 2022-10-04 21:15:00
4      MIRACLE IN THE WOOD      JAY 2022-10-06 17:22:00
continue viewing? y/n:y
1- RECENT EVENTS
2-MEMBERS OF EACH GROUP
3-EVENT LOCATION AND TICKET DETAILS
4-VIEW TABLES
5-INSERT NEW TICKET DETAILS
0-EXIT
choice? 2
      artist name      member name
0      IRENE           IRENE SEL
1      JAY             JAY PARK
2      STRAY KIDS      CHRISTOPHER CHAN
3      STRAY KIDS      FELIX LEE
4      STRAY KIDS      JEONGIN YANG
5      STRAY KIDS      HAN JISUNG
continue viewing? y/n:y
1- RECENT EVENTS
2-MEMBERS OF EACH GROUP
3-EVENT LOCATION AND TICKET DETAILS
4-VIEW TABLES
5-INSERT NEW TICKET DETAILS
0-EXIT
choice? 3
      t_no  seat      e_date arena_no
0  30030  D039  2022-10-06    DF21
1  30085  E009  2022-10-05    EG13
2  30087  E049  2022-10-05    EG13
3  30089  A453  2022-10-04    BA89
continue viewing? y/n:
```

```

continue viewing? y/n:y
1- RECENT EVENTS
2-MEMBERS OF EACH GROUP
3-EVENT LOCATION AND TICKET DETAILS
4-VIEW TABLES
5-INSERT NEW TICKET DETAILS
0-EXIT
choice? 4
TABLES:
1-EMPLOYEE
2-HIRES
3-ARTIST
4-ARENA
5-PERFORMS
6-TIME
7-TICKET
8-BOOKING

enter table no. -2
    eid
0  A12
1  A13
2  A15
3  A30
4  A45
5  A82

```

```

continue viewing? y/n:y
1- RECENT EVENTS
2-MEMBERS OF EACH GROUP
3-EVENT LOCATION AND TICKET DETAILS
4-VIEW TABLES
5-INSERT NEW TICKET DETAILS
0-EXIT
choice? 4
TABLES:
1-EMPLOYEE
2-HIRES
3-ARTIST
4-ARENA
5-PERFORMS
6-TIME
7-TICKET
8-BOOKING

enter table no. -3
    artist_id artist_name
0          A12  STRAY KIDS
1          A13  STRAY KIDS
2          A15  STRAY KIDS
3          A30      IRENE
4          A45  STRAY KIDS
5          A82      JAY
continue viewing? y/n:y
1- RECENT EVENTS

```



```
continue viewing? y/n:y
1- RECENT EVENTS
2-MEMBERS OF EACH GROUP
3-EVENT LOCATION AND TICKET DETAILS
4-VIEW TABLES
5-INSERT NEW TICKET DETAILS
0-EXIT
```

choice? 4

TABLES:

```
1-EMPLOYEE
2-HIRES
3-ARTIST
4-ARENA
5-PERFORMS
6-TIME
7-TICKET
8-BOOKING
```

enter table no. -4

	pid	arena_no
0	D452	BA89
1	G481	DF21
2	J872	EG13
3	S9213	EG13
4	X902	DF21

continue viewing? y/n:y

```
continue viewing? y/n:y
1- RECENT EVENTS
2-MEMBERS OF EACH GROUP
3-EVENT LOCATION AND TICKET DETAILS
4-VIEW TABLES
5-INSERT NEW TICKET DETAILS
0-EXIT
```

choice? 4

TABLES:

```
1-EMPLOYEE
2-HIRES
3-ARTIST
4-ARENA
5-PERFORMS
6-TIME
7-TICKET
8-BOOKING
```

enter table no. -5

	pno	pname	artist_name
0	2132	GODS MENU	STRAY KIDS
1	2534	MONSTER	IRENE
2	3421	BACK DOOR	STRAY KIDS
3	5642	GOBLIN: MUSICAL	IRENE
4	7639	MIRACLE IN THE WOOD	JAY

continue viewing? y/n:y

```
1- RECENT EVENTS
2-MEMBERS OF EACH GROUP
```

3-EVENT LOCATION AND TICKET DETAILS

4-VIEW TABLES

5-INSERT NEW TICKET DETAILS

0-EXIT

choice? 4

TABLES:

1-EMPLOYEE

2-HIRES

3-ARTIST

4-ARENA

5-PERFORMS

6-TIME

7-TICKET

8-BOOKING

enter table no. -6

	pid	pno	ptime	
0	S9213	2132	2022-10-05	16:04:00
1	X902	2534	2022-10-05	18:30:00
2	J872	3421	2022-10-05	16:45:00
3	D452	5642	2022-10-04	21:15:00
4	G481	7639	2022-10-06	17:22:00

continue viewing? y/n:y

1- RECENT EVENTS

2-MEMBERS OF EACH GROUP

3-EVENT LOCATION AND TICKET DETAILS

4-VIEW TABLES

5-INSERT NEW TICKET DETAILS

0-EXIT

choice? 4

TABLES:

1-EMPLOYEE

2-HIRES

3-ARTIST

4-ARENA

5-PERFORMS

6-TIME

7-TICKET

8-BOOKING

enter table no. -7

	t_no	seat	e_date
0	30030	D039	2022-10-06
1	30085	E009	2022-10-05
2	30087	E049	2022-10-05
3	30089	A453	2022-10-04

continue viewing? y/n:y

```

continue viewing? y/n:y
1- RECENT EVENTS
2-MEMBERS OF EACH GROUP
3-EVENT LOCATION AND TICKET DETAILS
4-VIEW TABLES
5-INSERT NEW TICKET DETAILS
0-EXIT
choice? 4
TABLES:
1-EMPLOYEE
2-HIRES
3-ARTIST
4-ARENA
5-PERFORMS
6-TIME
7-TICKET
8-BOOKING

enter table no. -8
      t_no arena_no
0  30085      EG13
1  30087      EG13
2  30089      BA89
3  30030      DF21
continue viewing? y/n:

```

Output after entering a record

```

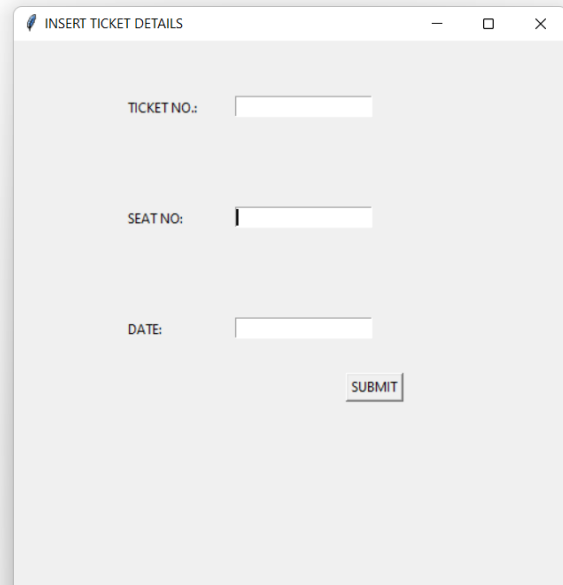
Python 3.8.5 Shell*
File Edit Shell Debug Options Window Help

3-ARTIST
4-ARENA
5-PERFORMS
6-TIME
7-TICKET
8-BOOKING

enter table no. -7
      t_no seat      e_date
0  30030 D039  2022-10-06
1  30085 E009  2022-10-05
2  30087 E049  2022-10-05
3  30089 A453  2022-10-04
continue viewing? y/n:y
1- RECENT EVENTS
2-MEMBERS OF EACH GROUP
3-EVENT LOCATION AND TICKET DETAILS
4-VIEW TABLES
5-INSERT NEW TICKET DETAILS
0-EXIT
choice? 4
TABLES:
1-EMPLOYEE
2-HIRES
3-ARTIST
4-ARENA
5-PERFORMS
6-TIME
7-TICKET
8-BOOKING

enter table no. -8
      t_no arena_no
0  30085      EG13
1  30087      EG13
2  30089      BA89
3  30030      DF21
continue viewing? y/n:y
1- RECENT EVENTS
2-MEMBERS OF EACH GROUP
3-EVENT LOCATION AND TICKET DETAILS
4-VIEW TABLES
5-INSERT NEW TICKET DETAILS
0-EXIT
choice? 5
inserted

```



INSERT TICKET DETAILS

TICKET NO:

SEAT NO:

DATE:

Data is recorded into 'ticket' and 'booking'

```
*Python 3.8.5 Shell*
File Edit Shell Debug Options Window Help
4-VIEW TABLES
5-INSERT NEW TICKET DETAILS
0-EXIT
choice? 4
TABLES:
1-EMPLOYEE
2-HIRES
3-ARTIST
4-ARENA
5-PERFORMS
6-TIME
7-TICKET
8-BOOKING

enter table no. -7
      t_no  seat    e_date
0  30030  D039   2022-10-06
1  30085  E009   2022-10-05
2  30087  E049   2022-10-05
3  30089  A453   2022-10-04
4  30313  D495   2002-01-22
continue viewing? y/n:y
1- RECENT EVENTS
2-MEMBERS OF EACH GROUP
3-EVENT LOCATION AND TICKET DETAILS
4-VIEW TABLES
5-INSERT NEW TICKET DETAILS
0-EXIT
choice? 4
TABLES:
1-EMPLOYEE
2-HIRES
3-ARTIST
4-ARENA
5-PERFORMS
6-TIME
7-TICKET
8-BOOKING

enter table no. -8
      t_no arena_no
0  30085      EG13
1  30087      EG13
2  30089      BA89
3  30030      DF21
4  30313      DF21
continue viewing? y/n:|
```