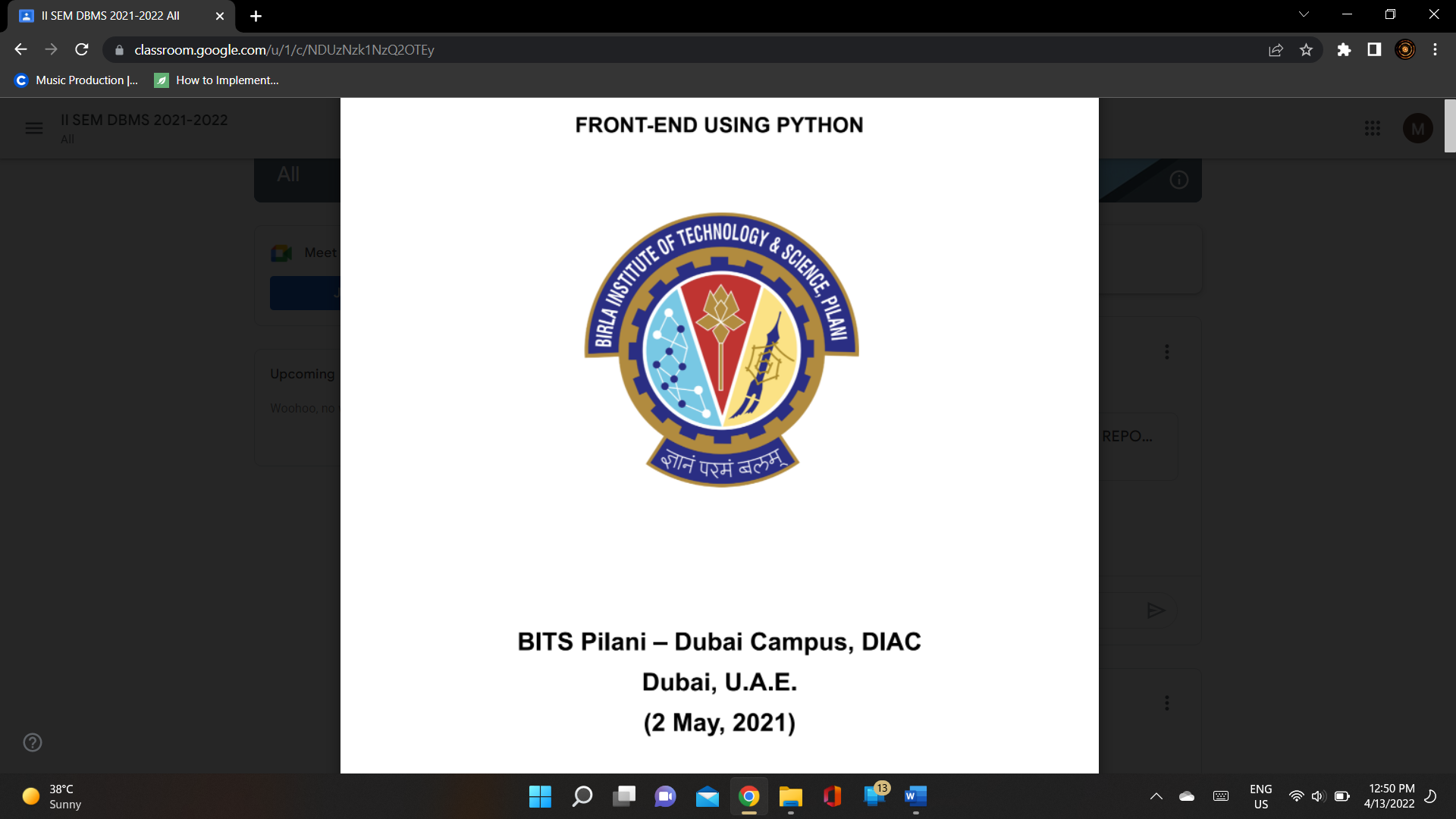
**A PROJECT**

**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 Sapna Sadhwani**

**Instructor in charge**

**By**

**Megha Manoj**

**Approved by**

**Dr Sapna Sadhwani**

**Instructor in charge**

**April 2022**

**Acknowledgement**

I would like to express my gratitude towards Dr. Sapna Sadhwani, 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.** |
| 1. **Acknowledgement** | **2** |
| 1. **Specification and purpose**    1. **Purpose**    2. **Specifications** | **4**  **4**  **4** |
| 1. **ER Diagram** | **5** |
| 1. **3NF Decomposition and Table creations**    1. **3RD Normal Form**    2. **Creating tables** | **6**  **6**  **7** |
| 1. **Populating tables** | **11** |
| 1. **Queries** | **13** |
| 1. **Constraints, procedures and triggers used**    1. **Constraints**    2. **Procedure**    3. **Trigger** | **15**  **15**  **16**  **17** |
| 1. **Front end development using Python**    1. **MySQL Connector**    2. **Pandas**    3. **Tkinter** | **18**  **19**  **20**  **20** |
| 1. **Source code** | **23** |
| 1. **Program output** | **30** |

1. **Specification and purpose**
   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. **Specifications**

The specifications included in the database are the following:

1. **Use of a front-end software**

For displaying and interacting with the user, the Python software has been used as a front-end.

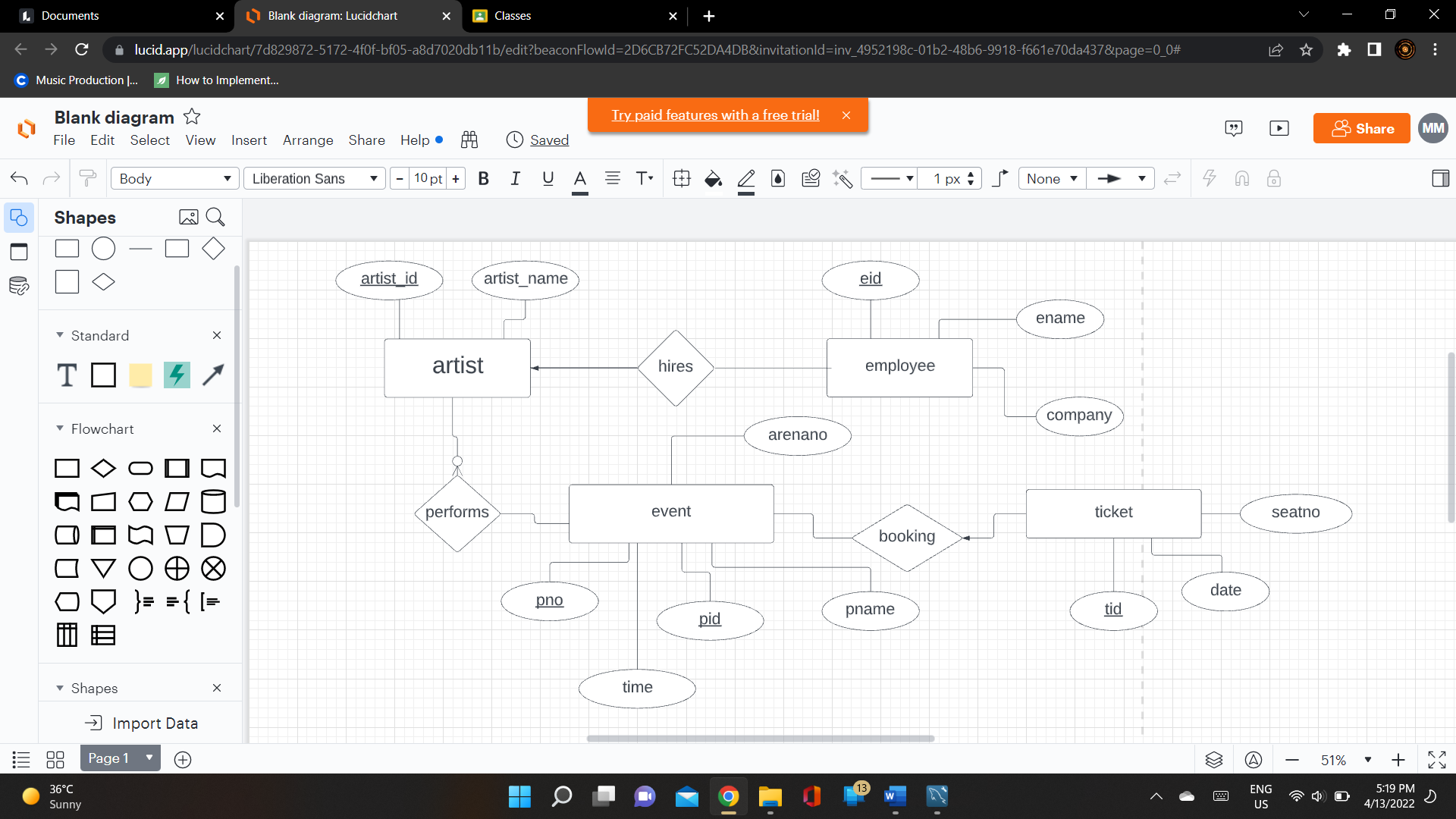
The Tkinter package in Python has been used for Graphical User Interface along with SQL Connect for connecting the front-end with the back-end

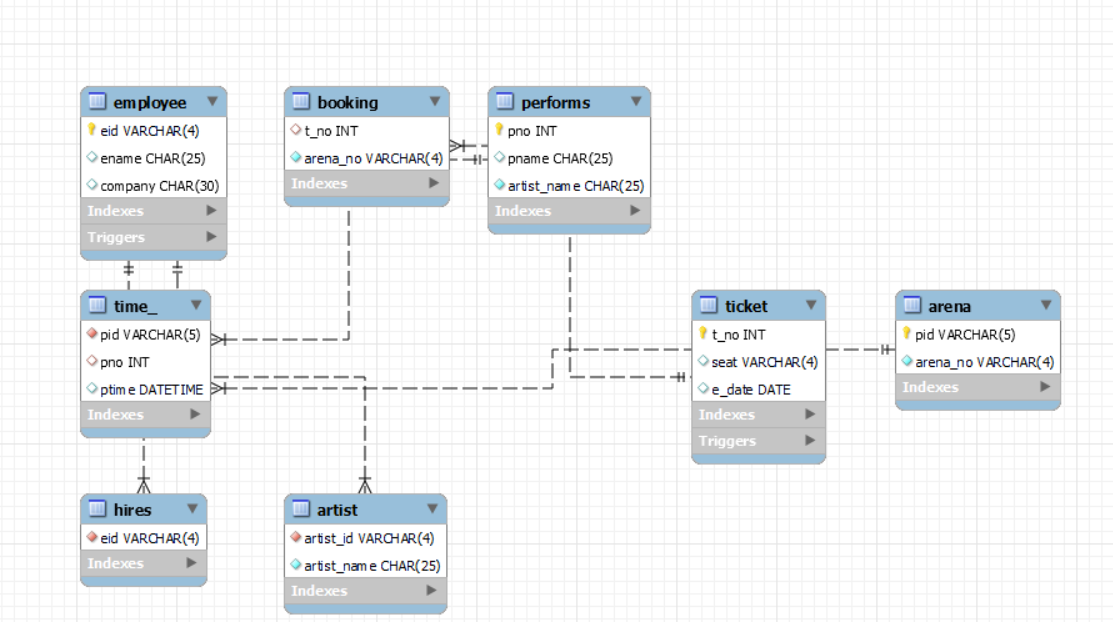
1. **Use of a back-end software**

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

1. **Normalization using 3NF**
2. **Basic SQL queries**
3. **Constraints and procedure**
4. **2 triggers**
5. **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**.**

1. **3NF decomposition and Table creations**
   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 –

1. No non-primary attribute is transitively dependent.
2. It contains an atomic value, there are no multivalued attributes
3. There is no partial dependency where absence of one attribute can still determine the required results
   1. **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:

1. **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)**

**);**

1. **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%') );**

1. **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));**

1. **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:

**event**

|  |  |  |  |
| --- | --- | --- | --- |
| **pid** | **pno** | **pname** | **ptime** |

**performs**

|  |  |  |
| --- | --- | --- |
| **pno** | **pname** | **artist\_name** |

**arena**

|  |  |
| --- | --- |
| **pid** | **arena\_no** |

**time\_**

|  |  |  |
| --- | --- | --- |
| **pid** | **pno** | **time** |

* 1. **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));**

* 1. **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));**

* 1. **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));**

1. **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));**

1. **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));**

1. **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.

* 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');**

* 1. **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');**

* 1. **Table arena**

**insert into arena values('S9213', 'EG13'),**

**('X902','DF21'),**

**('J872','EG13'),**

**('D452','BA89'),**

**('G481','DF21');**

* 1. **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');**

* 1. **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');**

* 1. **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');**

1. **Queries**

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

* 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;**

* 1. 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;**

* 1. 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;**

* 1. 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;**

* 1. 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.

1. **Constraints, procedures and triggers used**
   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:

1. **Not null**

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

1. **Unique**

To distinguish the records inserted.

1. **Foreign key**

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

1. **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.

1. **Check**

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

* 1. **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’.

* 1. **Triggers**

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

1. **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 ;**

1. **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 ;**

1. **Front end development using Python**

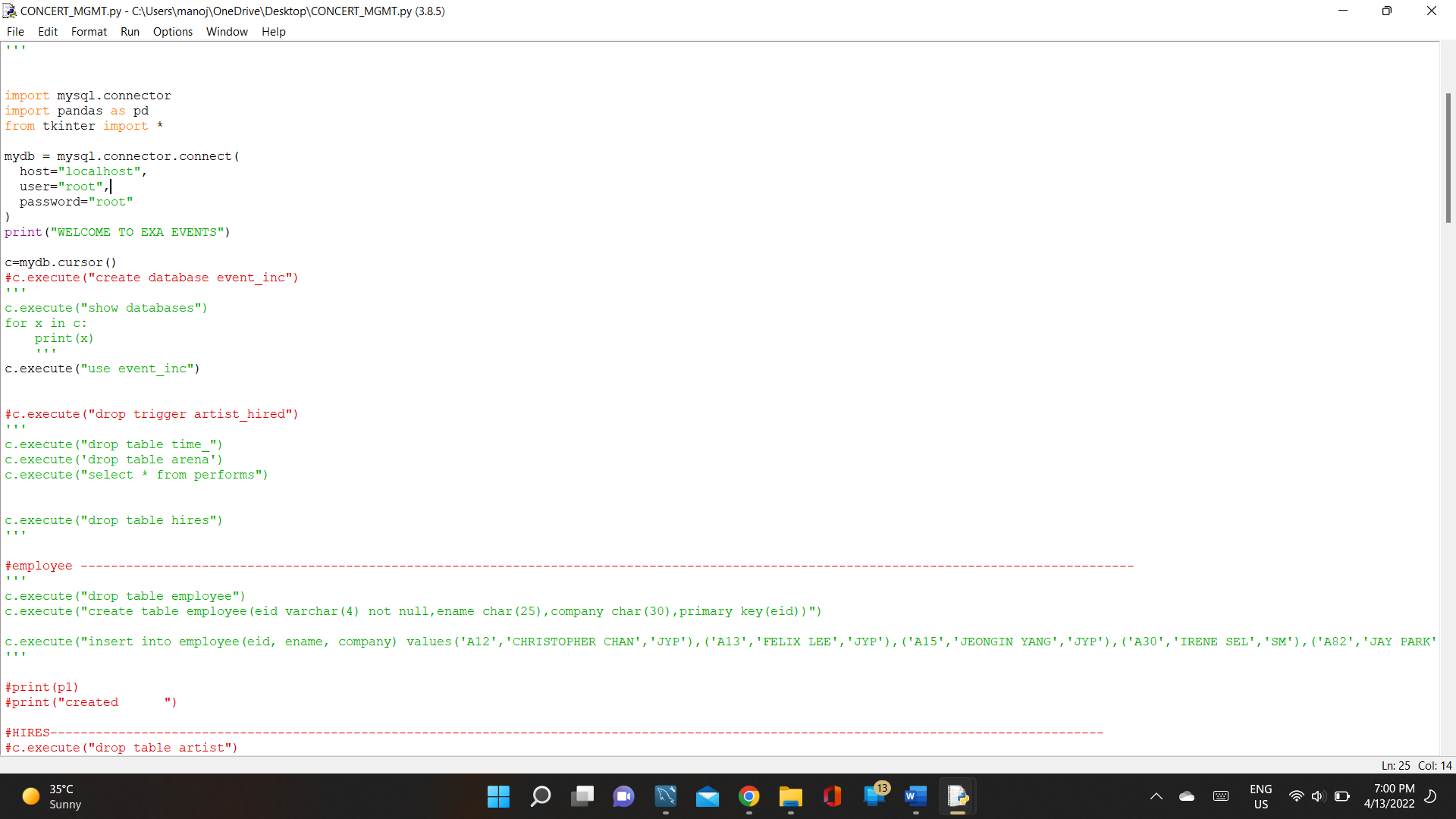
The front-end of this concert management system has been applied using the Python software for ease of access and application. The features of the developed front-end are as follows:

1. The queries are executed using functions from the imported packages
2. 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.
3. 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:

* 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: ****

1. **Mysql.connector.connect()**

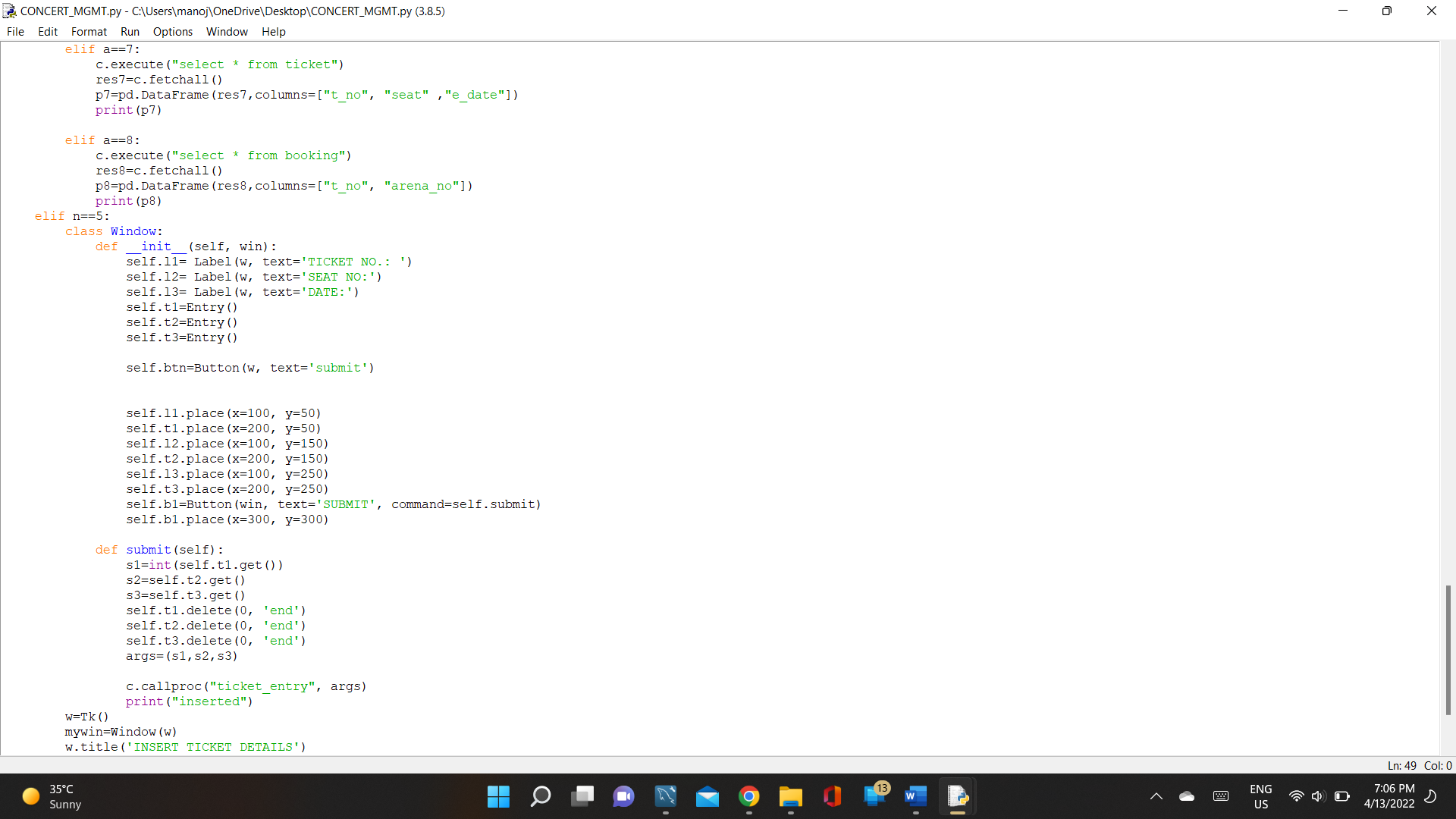
To create a connection to the MySQL server and receive a MySQLConnection object to work with.

1. **.cursor()**

An object which fetches data and executes queries.

1. **.execute()**

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



1. **.fetchall()**

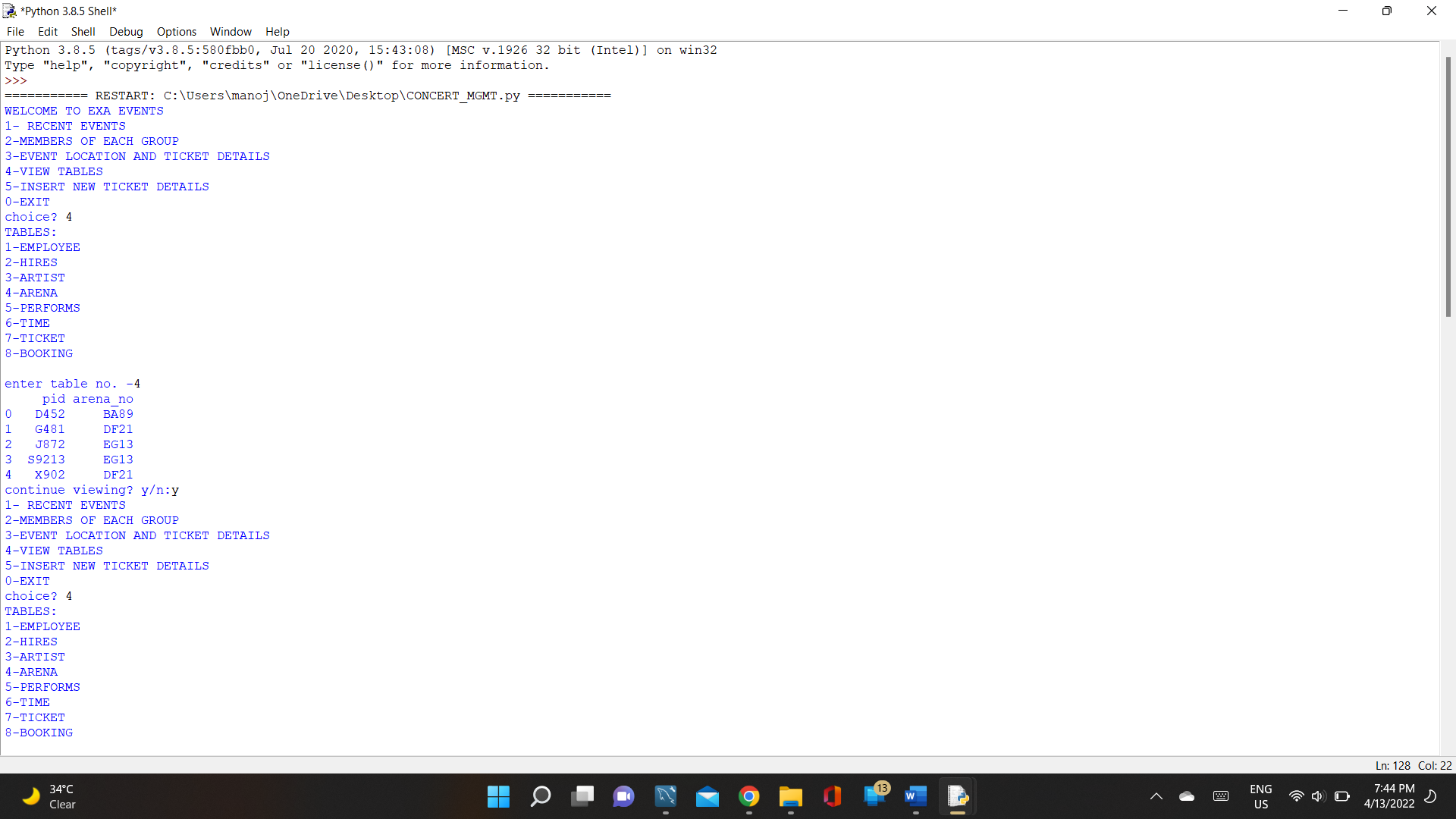
.fetchall() fetches all the rows of the executed queries

1. **.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.

* 1. **Pandas**

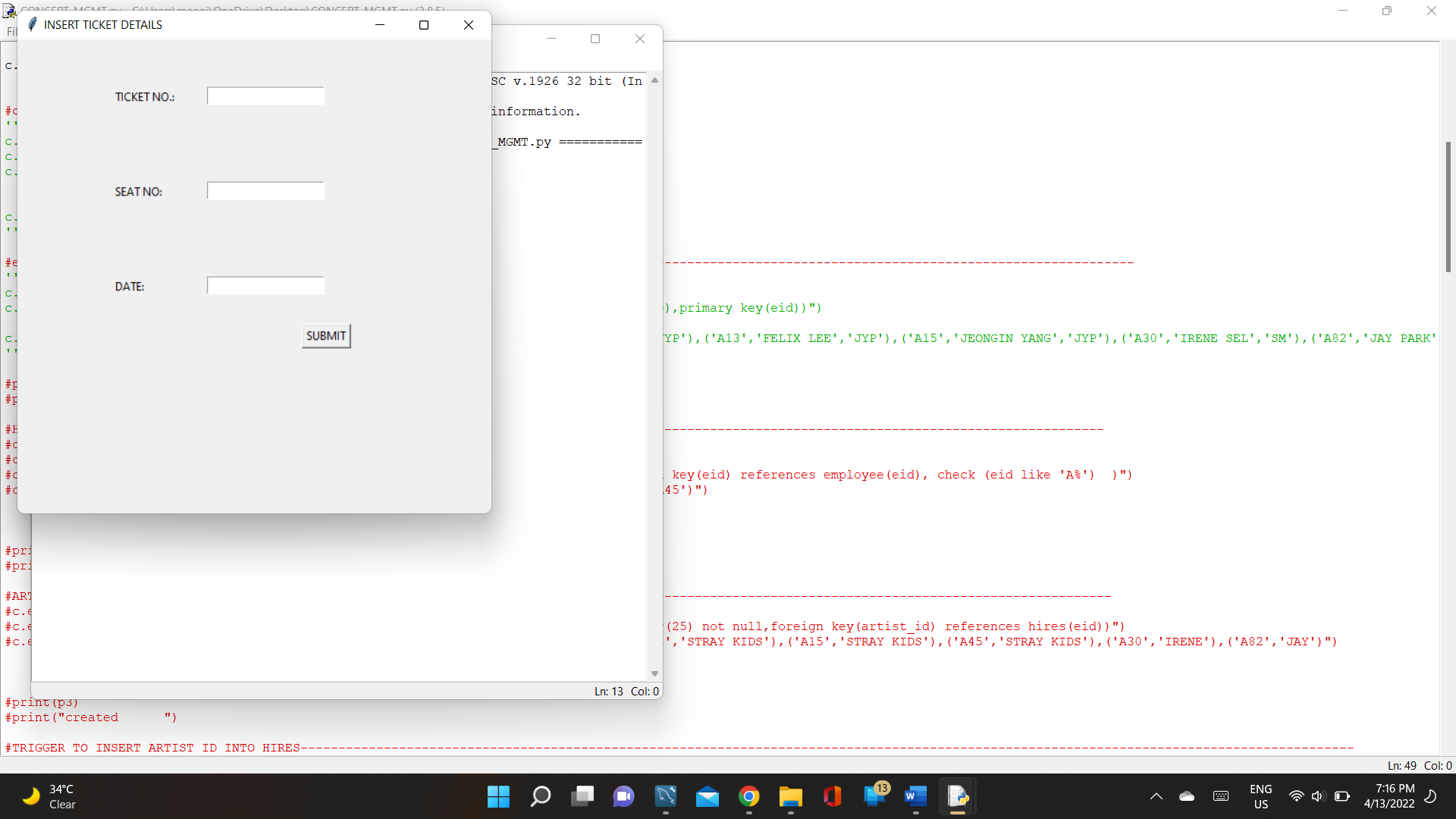
Pandas is a software library which is used in data manipulation and analysis. Here the function. DataFrame() has been used from the package to display the data in a tabular formate. The parameters of this function are- the fetched results and an array of the column titles.



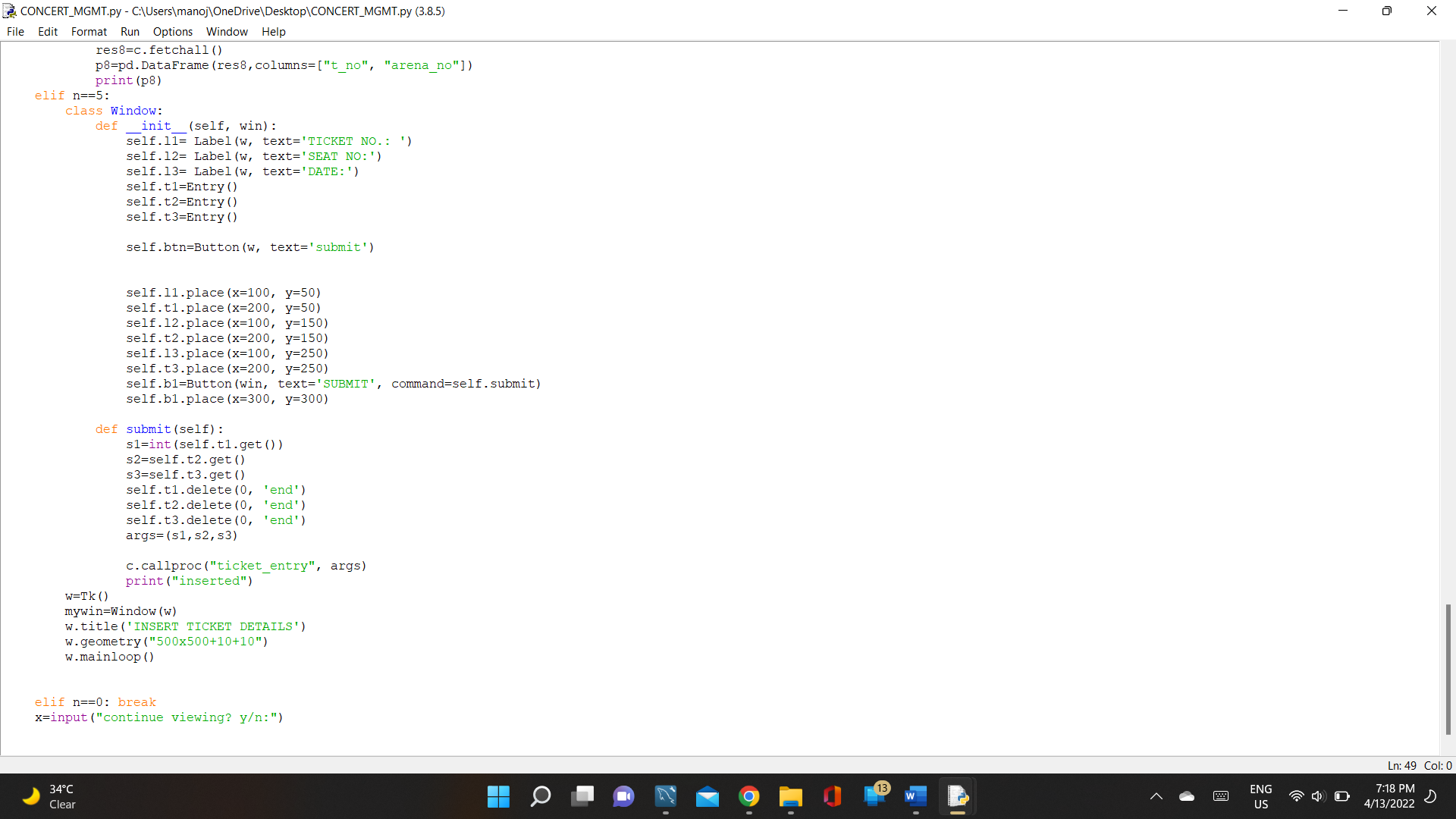
* 1. **Tkinter**

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:



1. **Label()**

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

1. **Button()**

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

1. **Entry()**

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

1. **.place()**

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

1. **.get()**

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

1. **.delete()**

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

1. **.title()**

It gives the created window a title at the header.

1. **.geometry()**

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

1. **Tk()**

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

1. **Source code**

****

****

****

****

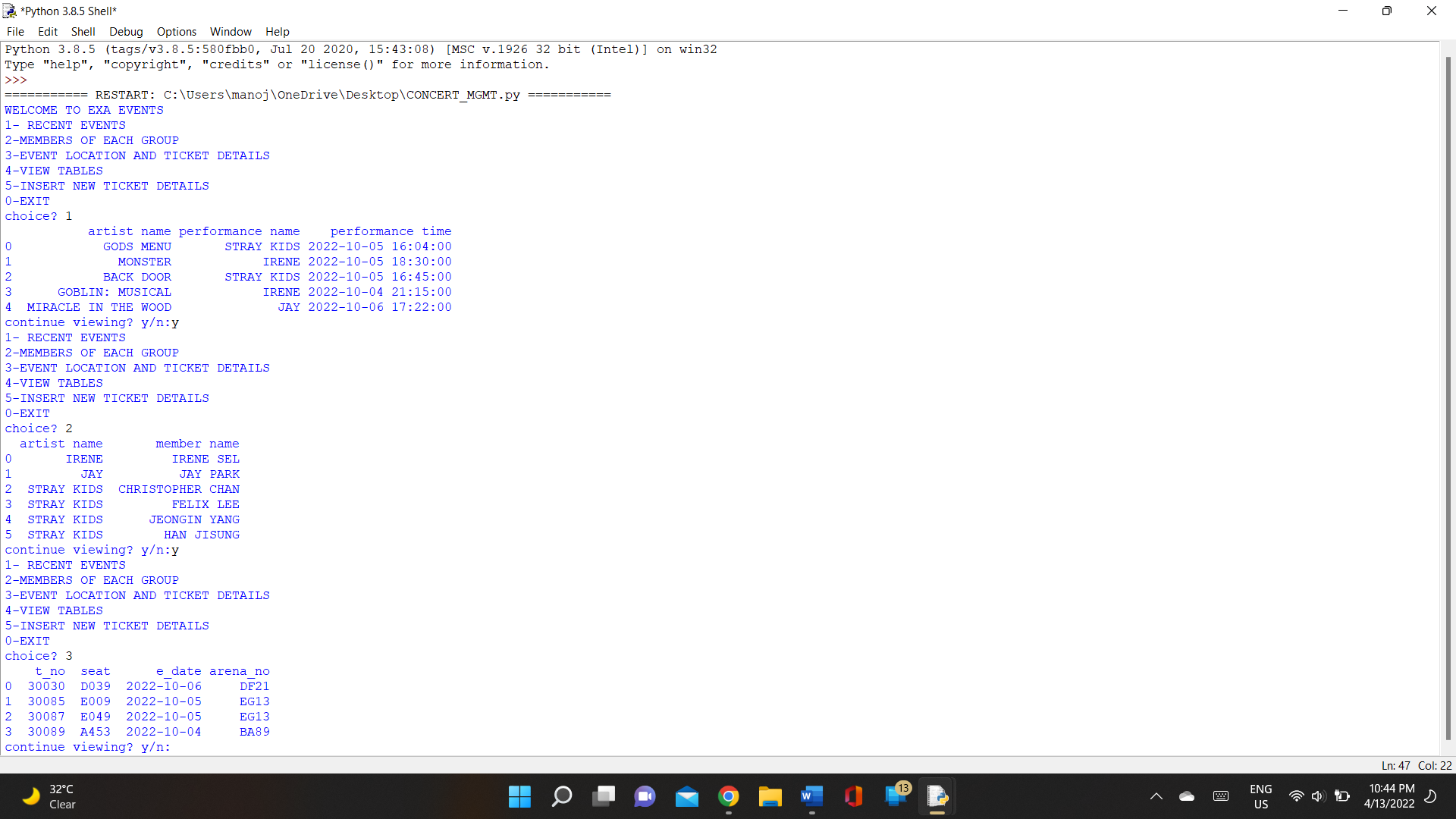
****

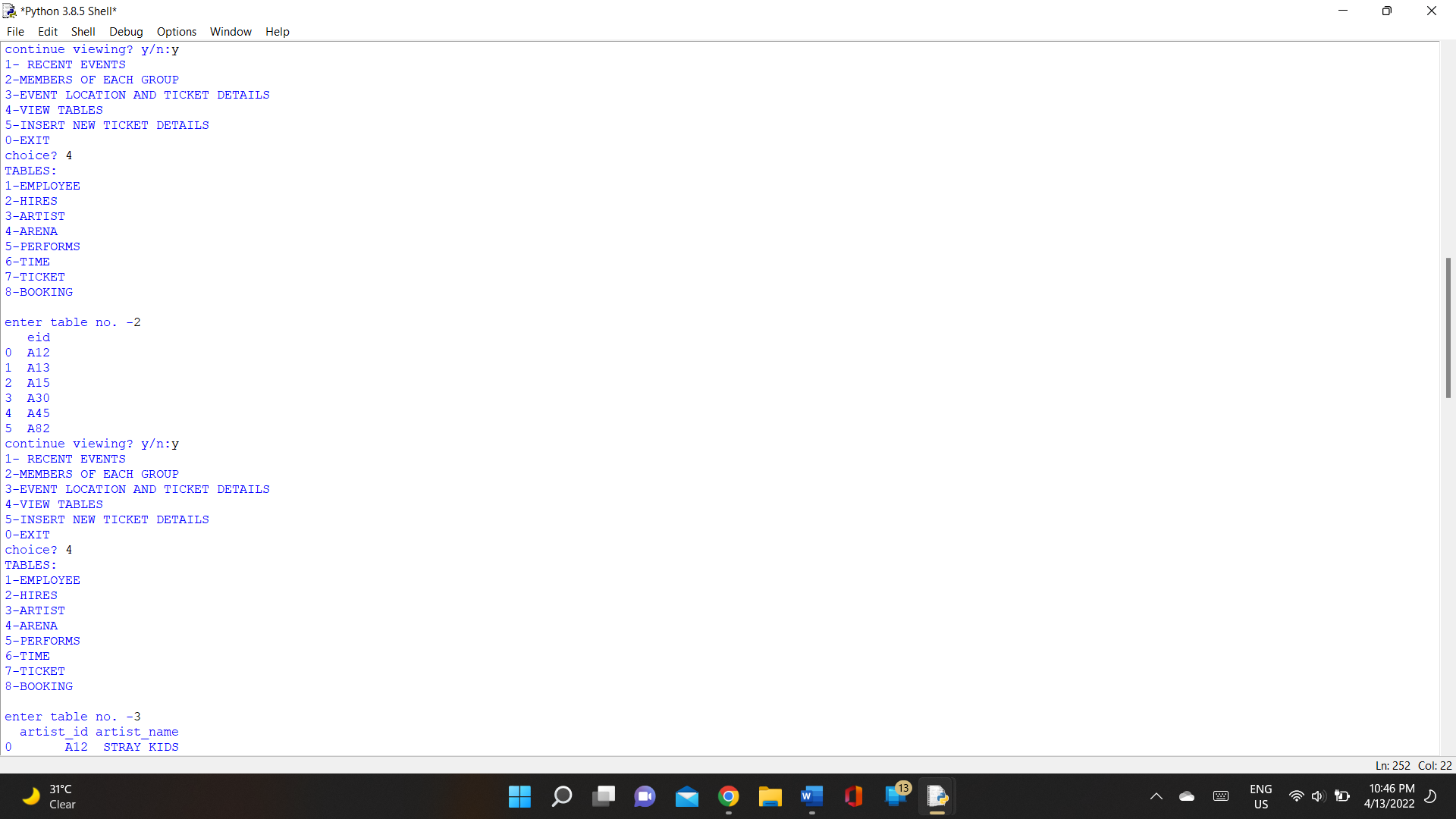
****

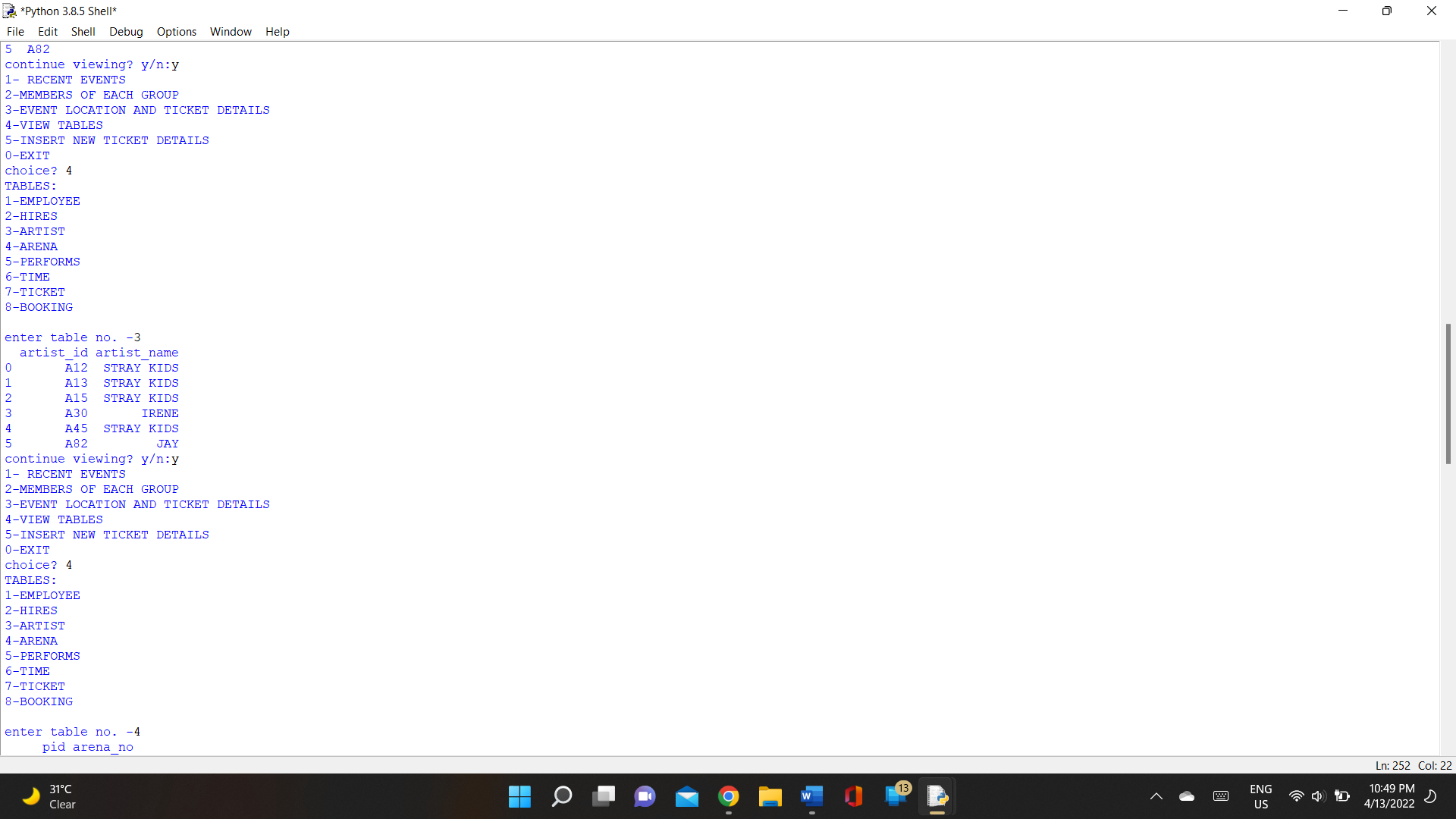
****

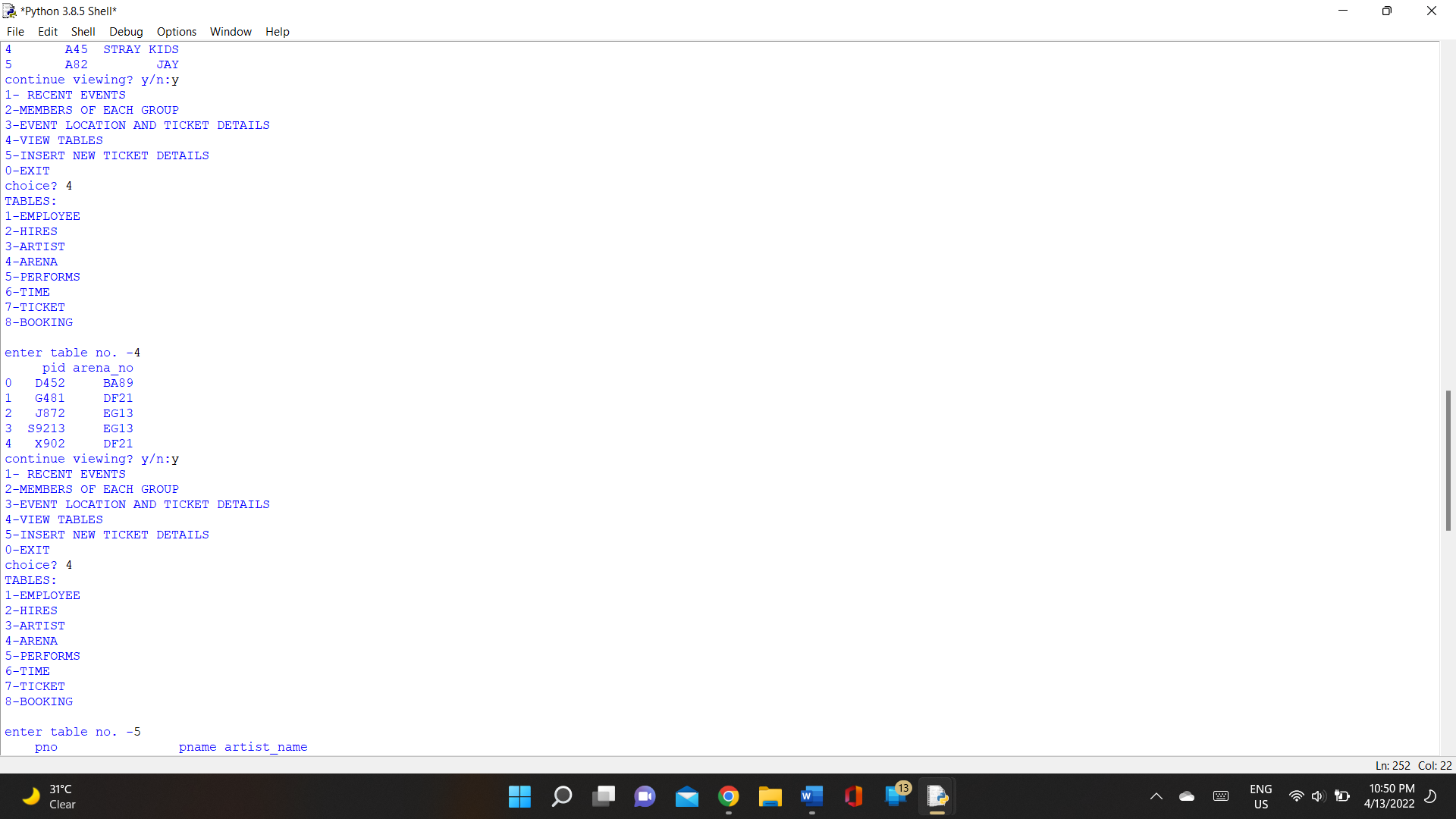
1. **Program output**

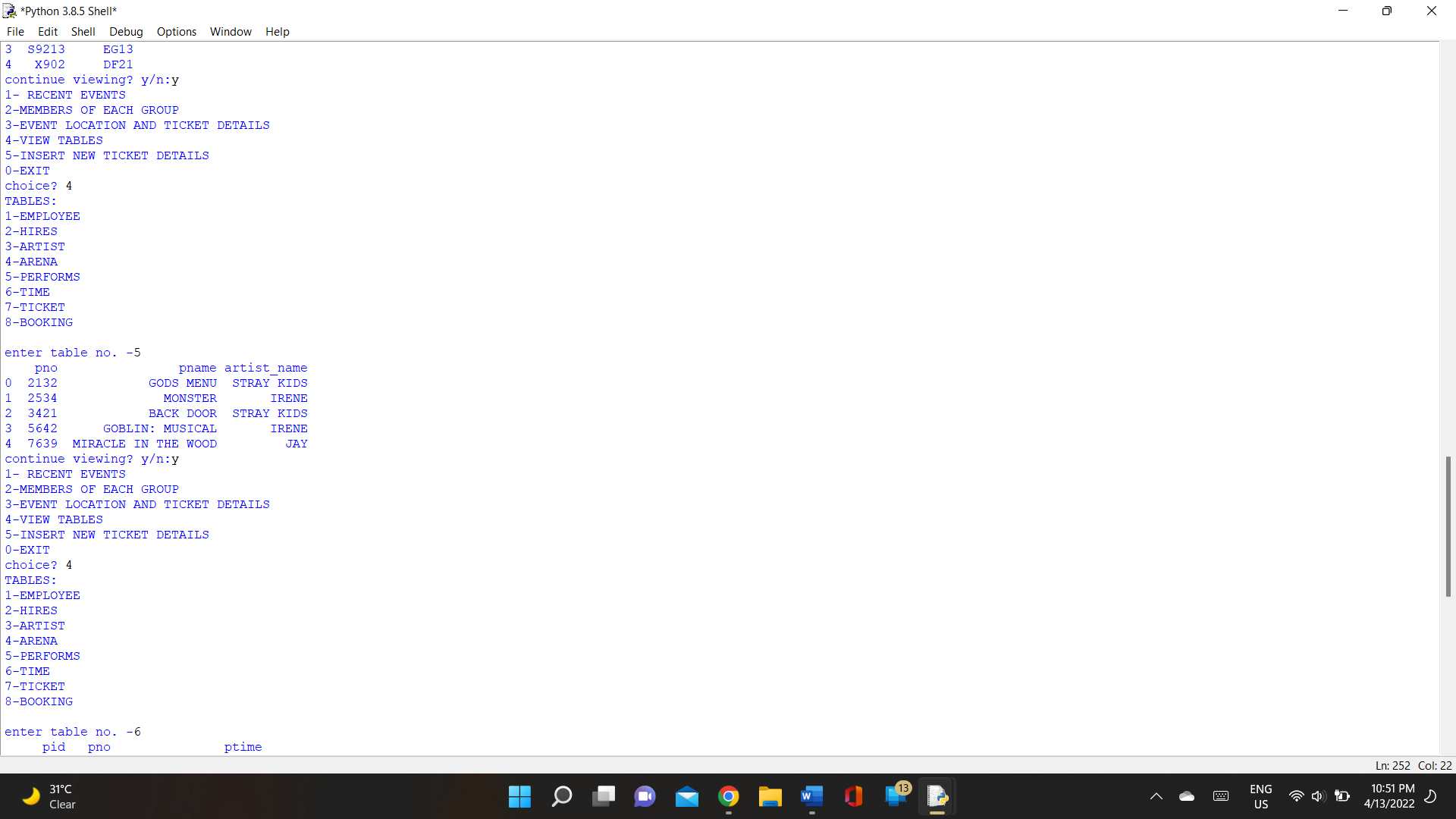
The screenshots below show the output of the program upon execution.

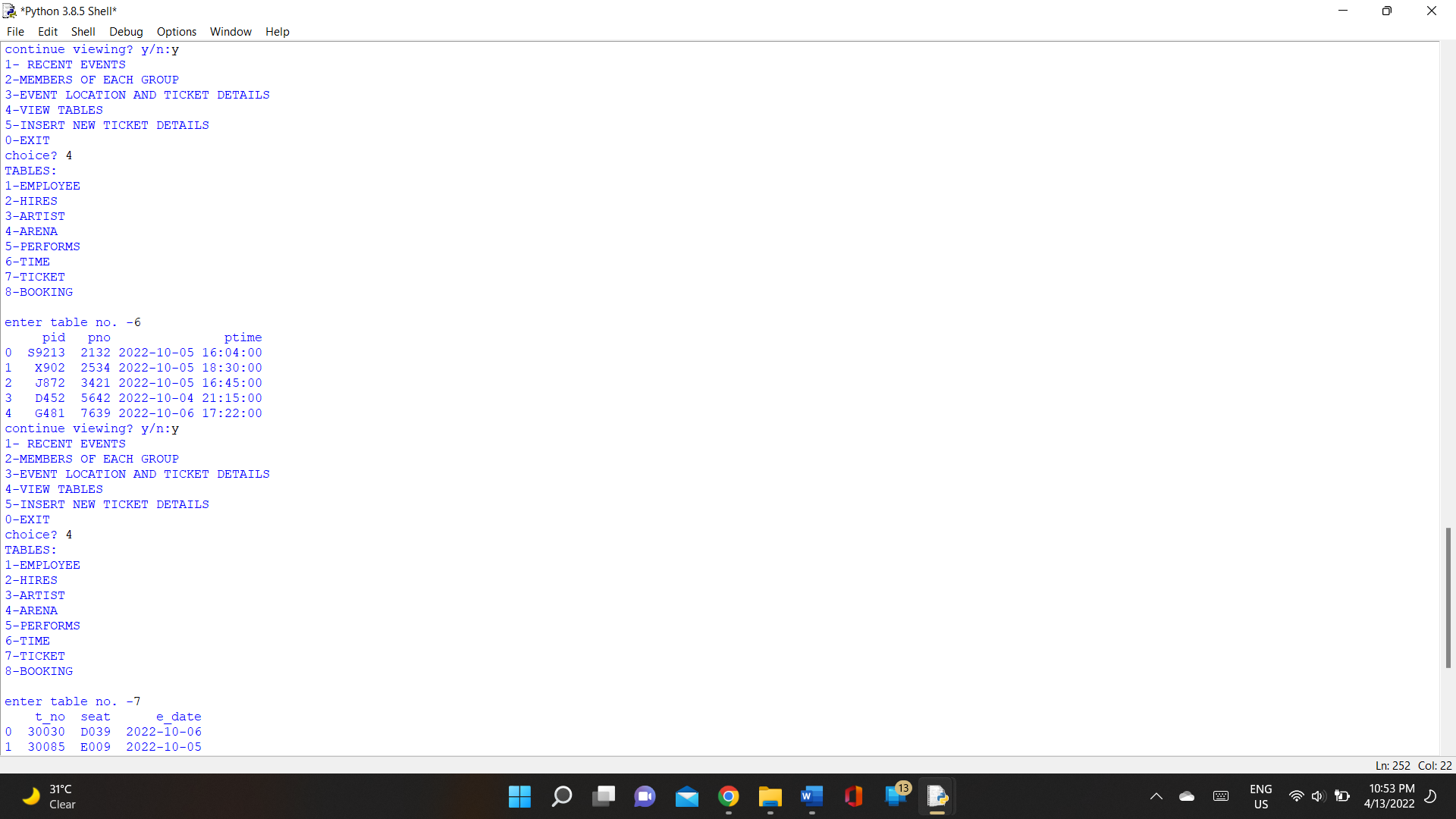


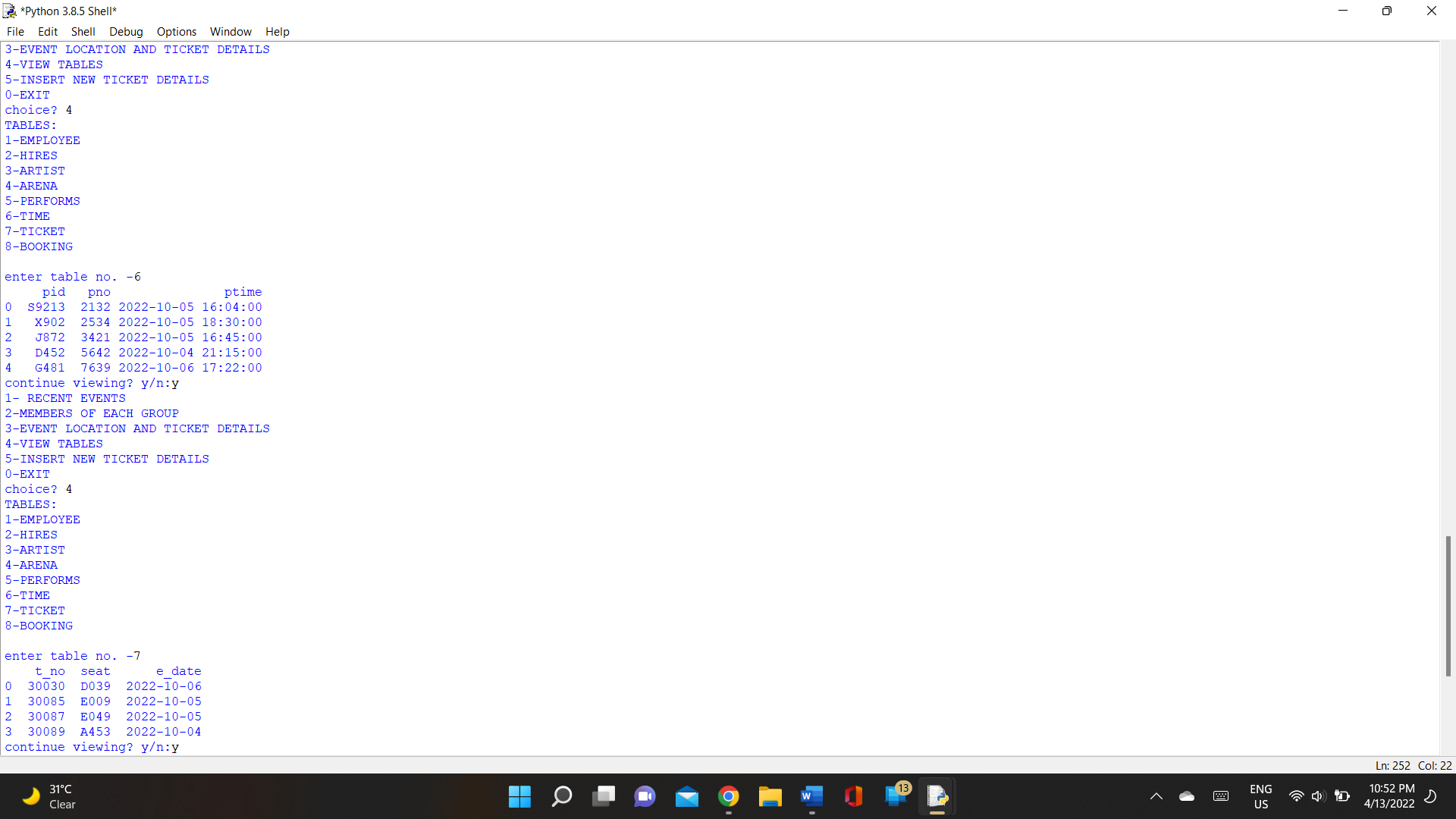


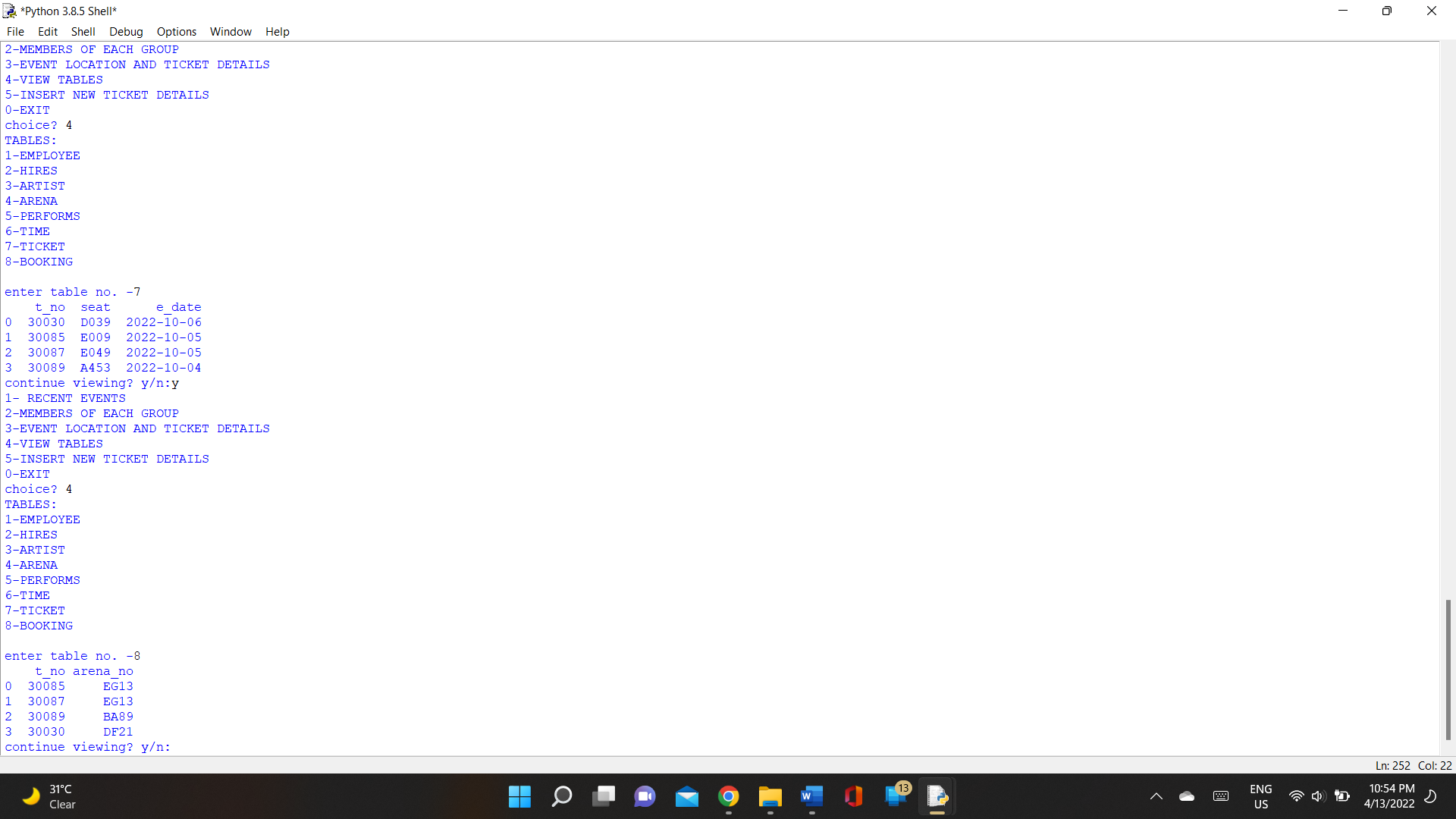




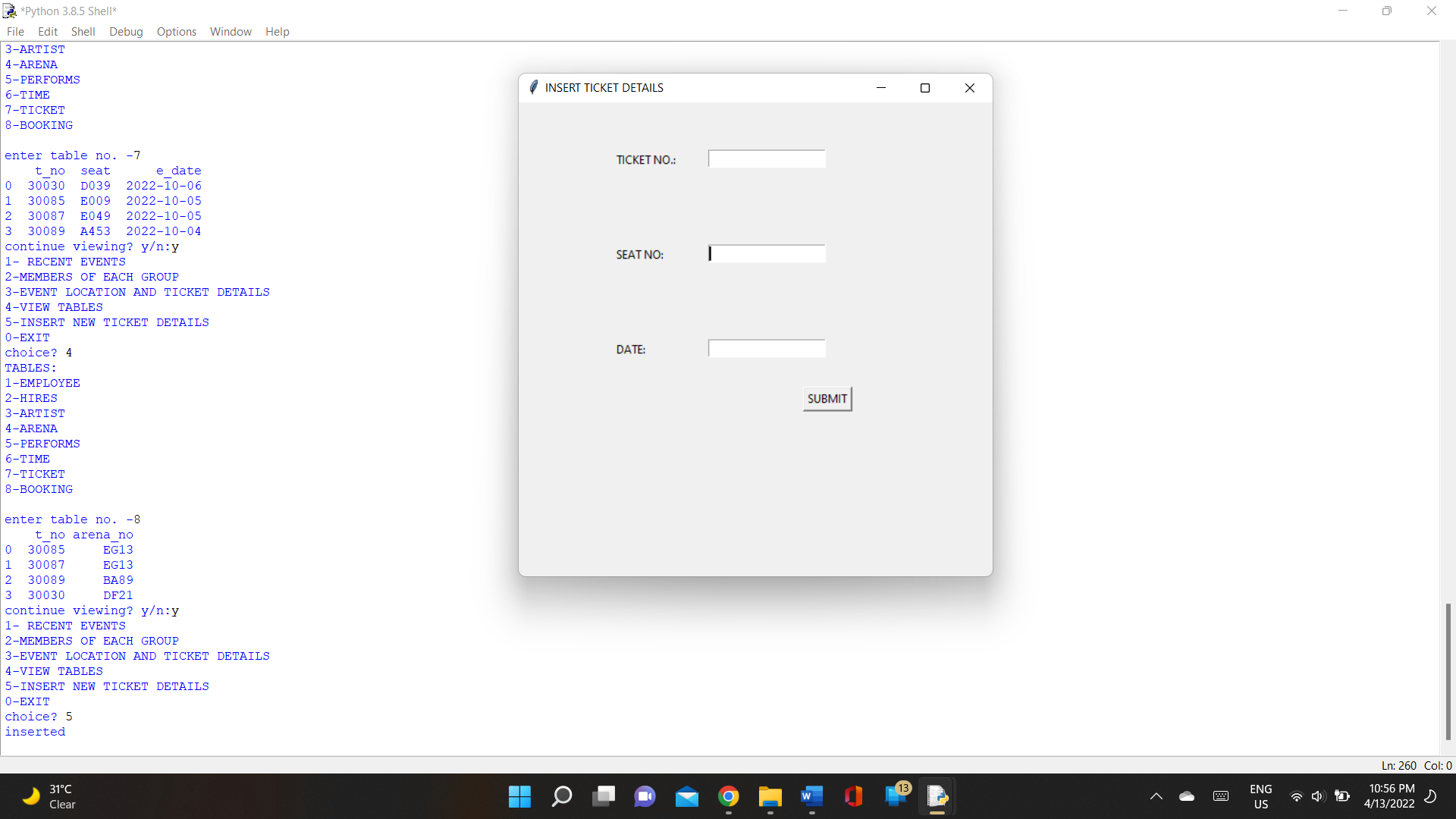








Output after entering a record



Data is recorded into ‘ticket’ and ‘booking’

