Python Assignment – Ticket Booking System

Submitted By-

Subrat Shukla, Python Batch 1

Control structure

Task : Conditional Statements

In a BookingSystem, you have been given the task is to create a program to book tickets. if available tickets more than noOfTicket to book then display the remaining tickets or ticket unavailable:

Tasks:

1. Write a program that takes the availableTicket and noOfBookingTicket as input.

2. Use conditional statements (if-else) to determine if the ticket is available or not.

3. Display an appropriate message based on ticket availability.

```python
def checkBookingTicket(availableTicket, noOfBookingTicket):  1 usage
    if availableTicket >= noOfBookingTicket:
        remainingticket = availableTicket - noOfBookingTicket
        print(f"Remaining Tikets are {remainingticket}")
    else:
        print("No  available tickets ")

availableTicket = int(input("Enter the available tickets: "))
noOfBookingTicket = int(input("Enter the number of booking tickets: "))
checkBookingTicket(availableTicket, noOfBookingTicket)
```

```
Run    task1
C:\Users\Shubh\AppData\Local\Programs\Pyt
Enter the available tickets: 20
Enter the number of booking tickets: 10
Remaining Tikets are 10

Process finished with exit code 0
```

Task : Nested Conditional Statements

Create a program that simulates a Ticket booking and calculating cost of tickets. Display tickets options such as "Silver", "Gold", "Dimond". Based on ticket category fix the base ticket price and get the user input for ticket type and no of tickets need and calculate the total cost of tickets booked.

Task 3: Looping

From the above task book the tickets for repeatedly until user type "Exit".

```python
#task - 2

def calculateCostOfBooking(category, noOfTickets):  2 usages
    Clist = {"silver":100, "gold":200, "diamond":3000}
    if category in Clist:
        basePrice = Clist[category]
        totalPrice = basePrice*noOfTickets
        print(f"Your total price is: {totalPrice}")

    else:
        print("Invalid ticket category")

category = input("Enter the category (gold, silver, diamond): ")
noOfTickets = int(input("Enter the number of tickets: "))
calculateCostOfBooking(category, noOfTickets)

#task - 3
while(True):
    category = input("Enter the category (gold, silver, diamond): ")
    if category == "Exit":
        break
    noOfTickets = int(input("Enter the number of tickets: "))

    calculateCostOfBooking(category, noOfTickets)
```

```
C:\Users\Shubh\AppData\Local\Programs\Python\Python312\pyth
Enter the category (gold, silver, diamond): gold
Enter the number of tickets: 20
Your total price is: 4000
Enter the category (gold, silver, diamond): diamond
Enter the number of tickets: 10
Your total price is: 30000
Enter the category (gold, silver, diamond): Exit

Process finished with exit code 0
```

Implement oops

Task : Class & Object

Create Following classes with the given attributes and methods:

1. Event Class

```python
import decimal
import enum

class event:  6 usages
    def __init__(self,event_name,event_date,event_time,venue_name,
                 total_seats,available_seats,
                 ticket_price: decimal,event_type: enum):
        self.event_name=event_name
        self.event_date=event_date
        self.event_time=event_time
        self.venue_name=venue_name
        self.total_seats=total_seats
        self.available_seats=available_seats
        self.ticket_price=ticket_price
        self.event_type=event_type

    # getters
    @property  1 usage
    def get_event_name(self):
        return self.event_name

    @property  2 usages
    def get_event_date(self):
        return self.event_date
    @property  1 usage
    def get_event_time(self):
        return self.event_time
```

```python
    @property  1 usage
    def get_venue_name(self):
        return self.venue_name

    @property  1 usage
    def get_total_seats(self):
        return self.total_seats

    @property  2 usages
    def get_available_seats(self):
        return self.available_seats

    @property  1 usage
    def get_ticket_price(self):
        return self.ticket_price

    @property  1 usage
    def get_event_type(self):
        return self.event_type
```

```python
    #setters
    @get_event_date.setter
    def set_event_name(self,event_name):
         self.event_name=event_name
    @ get_event_date.setter
    def set_event_date(self,event_date):
        self.event_date=event_date
    @get_event_time.setter
    def set_event_time(self,event_time):
        self.event_time=event_time
    @get_venue_name.setter
    def set_venue_name(self,venue_name):
        self.venue_name=venue_name
    @get_total_seats.setter
    def set_total_seats(self,total_seats):
        self.total_seats=total_seats
    @get_available_seats.setter
    def set_available_seats(self,available_seats):
        self.available_seats=available_seats
    @get_ticket_price.setter
    def set_ticket_price(self,ticket_price):
        self.ticket_price=ticket_price
    @get_event_type.setter
    def set_event_type(self,event_type):
        self.event_type=event_type
```

```python
    def print_event_info(self):  3 usages
        print("Event name: ",self.event_name)
        print("Event_date: ", self.event_date)
        print("Event_time: ", self.event_time)
        print("Venue_name: ", self.venue_name)
        print("Total_seats: ", self.total_seats)
        print("Available_seats: ", self.available_seats)
        print("Ticket_price: ", self.ticket_price)
        print("Event_type: ", self.event_type)

    def totalrevenue(self):
        tickets_sold=self.total_seats-self.available_Seats
        return tickets_sold*self.ticket_price

    def totaltickets_sold(self):
        return self.total_seats-self.available_Seats

    def book_tickets(self,num_tickets):  1 usage
        if num_tickets <=self.available_seats:
            self.available_seats-=num_tickets
            print(f"Booked{num_tickets}tickets. Available seats: {self.available_seats}")
        else:
            print("Not enough available seats for the requested number of tickets.")

    def cancel_tickets(self,num_tickets):  1 usage
        self.available_seats+=num_tickets
        print(f"After canceling{num_tickets} available tickets are {self.available_seats}")
```

2. Venue Class

```python
class venue:
    def __init__(self,venue_name,address):
        self.venue_name=venue_name
        self.address=address

    def print_venue_details(self):
        print("venue name: ",self.venue_name)
        print("address: ",self.address)

    # getters
    @property  1 usage
    def get_venue_name(self):
        return self.venue_name
    @property  1 usage
    def get_address(self):
        return self.address

    # setters
    @get_venue_name.setter
    def set_venue_name(self, venue_name):
        self.venue_name = venue_name
    @get_address.setter
    def set_address(self, address):
        self.address = address
```

3. Customer Class

```python
class customer:
    def __init__(self,firstname,lastname, email,phone_number,address):
        self.firstname=firstname
        self.lastname = lastname
        self.email=email
        self.phone_number=phone_number
        self.address=address

    #getters
    @property  1 usage
    def getfirstname(self):
        return self.customer_name

    @property  1 usage
    def getlastname(self):
        return self.lastname

    @property  1 usage
    def getemail(self):
        return self.email

    @property  1 usage
    def getphone_number(self):
        return self.phone_number

    @property  1 usage
    def getaddress(self):
        return self.address
```

```python
    #setters
    @getfirstname.setter
    def set_customer_name(self,firstname):
        self.firstname=firstname

    @getlastname.setter
    def setlastname(self, lastname):
        self.lastname = lastname
    @getemail.setter
    def set_email(self,email):
        self.email=email
    @getphone_number.setter
    def set_phone_number(self,phone_number):
        self.phone_number=phone_number

    @getaddress.setter
    def setaddress(self, address):
        self.address = address
```

4. Booking Class

```python
class bookings:
    def __int__(self,event_id,num_tickets,total_cost,booking_date):
        self.event_id=event_id
        self.num_tickets=num_tickets
        self.total_cost=total_cost
        self.booking_date=booking_date

    @property  1 usage
    def getevent_id(self):
        return self.event_id
    @property  1 usage
    def getnum_tickets(self):
        return self.num_tickets
    @property  2 usages
    def total_cost(self):
        return self.total_cost
    @property  2 usages
    def booking_date(self):
        return self.booking_date
```

```python
    @getevent_id.setter
    def setevent_id(self,event_id):
        self.event_id=event_id
    @getnum_tickets.setter
    def setnum_tickets(self,num_tickets):
        self.num_tickets=num_tickets

    @total_cost.setter
    def settotal_cost(self,total_cost):
        self.total_cost=total_cost

    @booking_date.setter
    def setbooking_date(self,booking_date):
        self.booking_date=booking_date
```

Task : Inheritance and polymorphism

- Create a subclass **Movie** that inherits from Event. Add the given attributes and methods:

```python
# task-5
from events import event
class movie(event):  1 usage
    def __init__(self):
        self.genre=" "
        self.actorname=" "
        self.actressname=" "

    @property  2 usages
    def getgenre(self):
        return self.genre
    @property  2 usages
    def getactorname(self):
        return self.actorname
    @property  2 usages
    def getactressname(self):
        return self.actressname
```

```python

    @getgenre.setter  1 usage
    def setgenre(self,genre):
        self.genre=genre
    @getactorname.setter  1 usage
    def setactorname(self,actorname):
        self.actorname=actorname
    @getactressname.setter  1 usage
    def setactressname(self,actressname):
        self.actressname=actressname

    def display_event_details(self):  2 usages (1 dynamic)
        event1.print_event_info()
        print("genre: ",self.genre)

```

- Create another subclass **Concert** that inherits from Event. Add the given attributes and methods:

```python
class concert(event):
    def __int__(self):
        self.artist=" "
        self.type=" "

    @property  2 usages
    def getartist(self):
        return self.artist
    @getartist.setter
    def setartist(self,artist):
        self.artist=artist

    @property
    def gettype(self):
        return self.type

    @getartist.setter
    def settype(self, type):
        self.type = type

    def display_concert_details(self):
        event1.print_event_info()
        print("artist: ", self.artist)
```

- Create another subclass **Sports** that inherits from Event. Add the given attributes and methods:

```python
class sports(event):
    def __init__(self):
        self.sportname=" "
        self.teams=" "

    @property  1 usage
    def getsportname(self):
        return self.sportname
    @property  1 usage
    def getteams(self):
        return self.teams

    @getsportname.setter
    def setsportname(self,sportname):
        self.sportname=sportname
    @getteams.setter
    def setactorname(self,teams):
        self.teams=teams

    def display_sport_details(self):
        event1.print_event_info()
        print("genre: ",self.sportname)
```

- Create a class **TicketBookingSystem** with the given methods:

```python
class TicketBookingSystem:  1 usage
    def __init__(self):
        self.events = []
    def create_event(self,event_name: str, date:str, time:str, total_seats: int,  1 usage
                     ticket_price: float, event_type: str, venue_name:str):
        new_event = event(event_name, date, time, total_seats, ticket_price,
                          event_type, venue_name)
        self.events.append(new_event)
        return new_event

    def display_event_details(self, event):  2 usages (1 dynamic)
        event.display_event_details()

    def book_tickets(self, event, num_tickets):  1 usage
        if num_tickets <= event.availableSeats:
            event.availableSeats -= num_tickets
            total_cost = num_tickets * event.ticketPrice
            return total_cost
        else:
            print("Sorry, the event is sold out. Not enough available seats.")
            return 0
```

```python
    def cancel_tickets(self, event, num_tickets):  1 usage
        event.availableSeats += num_tickets
        print(f"{num_tickets} tickets canceled for the event.")

    def main(self):  1 usage
        while True:
            print("\n1. Create Event\n2. Display Event Details\n"
                  "3. Book Tickets\n4. Cancel Tickets\n5. Exit")
            choice = input("Enter your choice (1-5): ")

            if choice == '1':
                event_name = input("Enter event name: ")
                date = input("Enter date: ")
                time = input("Enter time: ")
                total_seats = int(input("Enter total seats: "))
                ticket_price = float(input("Enter ticket price: "))
                event_type = input("Enter event type (movie, sport, concert): ")
                venue_name = input("Enter venue name: ")

                new_event = self.create_event(event_name, date, time, total_seats,
                                              ticket_price, event_type, venue_name)
                print(f"Event '{new_event.eventName}' created successfully!")
```

```python
            elif choice == '2':
                event_index = int(input("Enter the index of the event to display details: "))
                if 0 <= event_index < len(self.events):
                    self.display_event_details(self.events[event_index])
                else:
                    print("Invalid event index.")

            elif choice == '3':
                event_index = int(input("Enter the index of the event to book tickets: "))
                if 0 <= event_index < len(self.events):
                    num_tickets = int(input("Enter the number of tickets to book: "))
                    total_cost = self.book_tickets(self.events[event_index], num_tickets)
                    if total_cost > 0:
                        print(f"Tickets booked successfully! Total Cost: ${total_cost}")
                else:
                    print("Invalid event index.")
```

```python
            elif choice == '4':
                event_index = int(input("Enter the index of the event to cancel tickets: "))
                if 0 <= event_index < len(self.events):
                    num_tickets = int(input("Enter the number of tickets to cancel: "))
                    self.cancel_tickets(self.events[event_index], num_tickets)
                else:
                    print("Invalid event index.")

            elif choice == '5':
                print("Exiting the Ticket Booking System.")
                break

            else:
                print("Invalid choice. Please enter a number between 1 and 5.")

t1=TicketBookingSystem
t1.main(self=2)
```

Task : Abstraction

Requirements

1. Event Abstraction:

```python
from abc import ABC, abstractmethod

class Event(ABC):
    def __init__(self, event_name, date, time, venue_name, total_seats, available_seats, ticket_price, event_type):
        self.event_name = event_name
        self.date = date
        self.time = time
        self.venue_name = venue_name
        self.total_seats = total_seats
        self.available_seats = available_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    @abstractmethod
    def display_event_details(self):
        pass
```

2. Concrete Event Classes:

```python
class Movie(Event):
    def display_event_details(self):
        print(f"Event Type: Movie")
        print(f"Event Name: {self.event_name}")
        print(f"Date: {self.date}")
        print(f"Time: {self.time}")
        print(f"Venue: {self.venue_name}")
        print(f"Total Seats: {self.total_seats}")
        print(f"Available Seats: {self.available_seats}")
        print(f"Ticket Price: {self.ticket_price}")
```

```python
class Concert(Event):
    def display_event_details(self):
        print(f"Event Type: Concert")
        print(f"Event Name: {self.event_name}")
        print(f"Date: {self.date}")
        print(f"Time: {self.time}")
        print(f"Venue: {self.venue_name}")
        print(f"Total Seats: {self.total_seats}")
        print(f"Available Seats: {self.available_seats}")
        print(f"Ticket Price: {self.ticket_price}")
```

```python
class Sport(Event):
    def display_event_details(self):
        print(f"Event Type: Sport")
        print(f"Event Name: {self.event_name}")
        print(f"Date: {self.date}")
        print(f"Time: {self.time}")
        print(f"Venue: {self.venue_name}")
        print(f"Total Seats: {self.total_seats}")
        print(f"Available Seats: {self.available_seats}")
        print(f"Ticket Price: {self.ticket_price}")
```

3. BookingSystem Abstraction:

```python
class BookingSystem(ABC):
    @abstractmethod
    def create_event(self, event_name, date, time, total_seats, ticket_price, event_type, venue_name):
        pass

    @abstractmethod
    def display_event_details(self, event):
        pass

    @abstractmethod
    def book_tickets(self, event, num_tickets):
        pass

    @abstractmethod
    def cancel_tickets(self, event, num_tickets):
        pass
```

Task : Interface/abstract class, and Single Inheritance, static variable

```python
class Venue:
    def __init__(self, venue_name, address):
        self.venue_name = venue_name
        self.address = address

    def display_venue_details(self):
        print("Venue Name:", self.venue_name)
        print("Address:", self.address)
```

```python
class IEventServiceProvider(ABC):
    @abstractmethod
    def create_event(self, event_name: str, date: str, time: str, total_seats: int, ticket_price: float,
                     event_type: str, venue: Venue):
        pass

    @abstractmethod
    def getEventDetails(self):
        pass

    @abstractmethod
    def getAvailableNoOfTickets(self):
        pass
```

```python
class IBookingSystemServiceProvider(ABC):
    @abstractmethod
    def calculate_booking_cost(self, num_tickets):
        pass

    @abstractmethod
    def book_tickets(self, eventname: str, num_tickets: int, arrayOfCustomer):
        pass

    @abstractmethod
    def cancel_booking(self, booking_id: int):
        pass

    @abstractmethod
    def get_booking_details(self, booking_id: int):
        pass
```

```python
class EventServiceProviderImpl(IEventServiceProvider):
    def __init__(self):
        self.events = []

    def create_event(self, event_name: str, date: str, time: str, total_seats: int, ticket_price: float,
                     event_type: str, venue: Venue):
        event = None
        if event_type == EventType.MOVIE.value:
            event = Movie(event_name, date, time, venue, total_seats, ticket_price, EventType.MOVIE)
        elif event_type == EventType.CONCERT.value:
            event = Concert(event_name, date, time, venue, total_seats, ticket_price, EventType.CONCERT)
        elif event_type == EventType.SPORTS.value:
            event = Sport(event_name, date, time, venue, total_seats, ticket_price, EventType.SPORTS)
        if event:
            self.events.append(event)
            return event
        else:
            print("Invalid event type.")
            return None

    def getEventDetails(self):
        return [event.display_event_details() for event in self.events]

    def getAvailableNoOfTickets(self):
        total_available_tickets = 0
        for event in self.events:
            total_available_tickets += event.available_seats
        return total_available_tickets
```

```python
class BookingSystemServiceProviderImpl(IBookingSystemServiceProvider, EventServiceProviderImpl):
    def calculate_booking_cost(self, num_tickets):
        pass

    def book_tickets(self, eventname, num_tickets, arrayOfCustomer):
        pass

    def cancel_booking(self, booking_id):
        pass

    def get_booking_details(self, booking_id):
        pass
```

```python
from abc import ABC, abstractmethod

class IBookingSystemRepository:  1 usage
    def create_event(self):
        pass
    def get_Event_Details(self):
        pass
    def get_available_tickets(self):
        pass
    def book_tickets(self,num_tickets):
        pass
    def cancel_tickets(self):
        pass
```

```python
class BookingSystemRepositoryImpl(IBookingSystemRepository):  2 usages
    def create_event(self):
        return True
    def get_Event_Details(self):
        return True
    def get_available_tickets(self):
        return True
    def book_tickets(self,num_tickets):
        return True
    def cancel_tickets(self):
        return True
```

Task : Exception Handling

throw the exception whenever needed and Handle in main method,

1. EventNotFoundException throw this exception when user try to book the tickets for Event not listed in the menu.
2. InvalidBookingIDException throw this exception when user entered the invalid bookingId when he tries to view the booking or cancel the booking.
3. NullPointerException handle in main method.

Throw these exceptions from the methods in TicketBookingSystem class. Make necessary changes to accommodate exception in the source code. Handle all these exceptions from the main program.

```python
class EventNotFoundException(Exception):  2 usages
    pass
class InvalidBookingIDException(Exception):  2 usages
    pass


class TicketBookingSystem1():  1 usage

    def book_tickets_menu(self):  1 usage
        try:
            eventname = input("Enter the event name: ")
            # Check if the event exists
            query1="select * from event where event_name=%s"
            cur.execute(query1,(eventname,))
            event=cur.fetchone()

            if not event:
                raise EventNotFoundException(f"Event '{eventname}' not found in the menu.")

        except EventNotFoundException as e:
            print(f"Error: {e}")
```

```python
    def booking_details_menu(self):  1 usage
        try:
            booking_id = input("Enter the booking ID: ")
            query1 = "select * from booking where booking_id=%s"
            cur.execute(query1,(booking_id,))
            booking = cur.fetchone()

            if not booking:
                raise InvalidBookingIDException(f"Invalid booking ID: {booking_id}")

        except InvalidBookingIDException as e:
            print(f"Error: {e}")

    def event_exists(self, event_name):
        pass

    def is_valid_booking_id(self, booking_id):
        pass

if __name__ == "__main__":
    ticket = TicketBookingSystem1()
    ticket.book_tickets_menu()
    ticket.booking_details_menu()
```

Task 11: Database Connectivity.

```python
import pyodbc

class DBConnection:  7 usages
    con = None

    @staticmethod  2 usages
    def getConnection():
        if DBConnection.con is None:
            try:
                DBConnection.con = pyodbc.connect(
                    'Driver={SQL Server};'
                    'Server=DESKTOP-MKS6P3G;'
                    'Database=TicketBookingSystem;'
                )
                print("Database Connected Successfully!!")
            except pyodbc.Error as err:
                print(f"Error connecting DB: {err}")

        return DBConnection.con
```

Ticket Booking System App:

```python
import functools
from datetime import *
import pyodbc
from dbutil import DBConnection
from abstract_methods import BookingSystemRepositoryImpl

class EventNotFoundException(Exception):
    pass

class InvalidBookingIDException(Exception):
    pass

con = DBConnection.getConnection()
cur = con.cursor()
```

```python
class TicketBookingSystem:
    while True:
        print("")
        print("1. Create Event")
        print("2. Book tickets")
        print("3. Cancel tickets")
        print("4. Get available tickets")
        print("5. Get event details")
        print("6. Exit")

        choice = input("Select from above options: ")
        if choice == "1":
            b.create_event()
        elif choice == "2":
            num_tickets = int(input("Enter the number of tickets: "))
            b.book_tickets(num_tickets)
        elif choice == "3":
            b.cancel_tickets()
        elif choice == "4":
            b.get_available_tickets()
        elif choice == "5":
            b.get_event_details()
        elif choice == "6":
            print("Exiting the system. Thank you!")
            break
        else:
            print("Invalid option, choose from the options above.")
```

```python
class BookingSystemRepository(BookingSystemRepositoryImpl):  1 usage

    def create_event(self):  1 usage
        event_id = input("Enter the event id: ")
        event_name = input("Enter event name: ")

        # Convert date and time to strings in the required format
        event_date = self.get_current_date().strftime('%Y-%m-%d')
        event_time = self.get_current_time().strftime('%H:%M:%S')

        total_seats = int(input("Enter total seats: "))
        ticket_price = float(input("Enter ticket price: "))
        event_type = input("Enter event type (movie, sport, concert): ")

        # Prepare query and parameterized values
        query = """
        INSERT INTO event(event_id, event_name, event_date, event_time, total_seats,
        ticket_price, event_type)
        VALUES (?, ?, ?, ?, ?, ?, ?)
        """
        values=(event_id,event_name,event_date,event_time,total_seats,ticket_price,event_type)
```

```python
        # Execute the query and commit
        try:
            cur.execute(query, values)
            con.commit()
            print("Event created successfully.")
        except pyodbc.Error as e:
            print(f"Error: {e}")
        finally:
            cur.execute("SELECT * FROM event")
            events = cur.fetchall()
            for event in events:
                print(event)

    def get_current_date(self):  2 usages
        return date.today()

    def get_current_time(self):  1 usage
        return datetime.now().time()

    def get_event_details(self):  1 usage
        return self.all_events()
```

```python
def all_events(self):  2 usages
    query = "SELECT * FROM event"
    cur.execute(query)
    events = cur.fetchall()
    for event in events:
        print(event)

def get_available_tickets(self):  1 usage
    query = "SELECT available_seats, event_name FROM event"
    cur.execute(query)
    events = cur.fetchall()
    for event in events:
        print(event)

def create_customer(self):  1 usage
    customer_id = self.unique_customer_id()
    customer_name = input("Enter your name: ")
    email = input("Enter your email: ")
    phone = input("Enter your phone number: ")
    booking_id = self.unique_booking_id()

    query = "INSERT INTO customer(customer_id, customer_name, email, phone_number, booking_id) VALUES(?,?,?,?,?)"
    values = (customer_id, customer_name, email, phone, booking_id)

    cur.execute(query, values)
    con.commit()
    print("Customer created successfully.")
```

```python
    def get_all_customers(self):
        query = "SELECT * FROM customer"
        cur.execute(query)
        return cur.fetchall()

    # def unique_customer_id(self):
    #     return len(self.get_all_customers()) + 1

    def unique_customer_id(self):  1 usage
        query = "SELECT ISNULL(MAX(customer_id), 0) FROM customer"
        cur.execute(query)
        max_id = cur.fetchone()[0]
        return max_id + 1  # Increment to get a new unique ID

    def book_tickets(self, num_tickets: int):  1 usage
        self.all_events()
        event_name = input("Enter the event name: ")

        # Fetch event details
        query = "SELECT event_id, total_seats, ticket_price FROM event WHERE event_name = ?"
        cur.execute(query, (event_name,))
        event = cur.fetchone()
        if not event:
            print(f"Error: Event '{event_name}' not found.")
            return None
```

```python
        event_id, total_seats, ticket_price = event

        # Check if enough seats are available
        if num_tickets > total_seats:
            print("Not enough available seats for the requested number of tickets.")
            return None

        customer_id = self.create_customer()  # Get newly created customer ID
        total_cost = float(num_tickets) * float(ticket_price)  # Ensure float conversion
        booking_date = self.get_current_date().strftime('%Y-%m-%d')  # Convert to correct format
        booking_id = self.unique_booking_id()

        # Ensure all values have correct types
        values = (
            int(booking_id),  # booking_id as integer
            int(customer_id),  # customer_id as integer
            int(event_id),  # event_id as integer
            int(num_tickets),  # num_tickets as integer
            float(total_cost),  # total_cost as float
            booking_date  # booking_date as string (YYYY-MM-DD)
        )
```

```python
        # Insert booking details
        query2 = """
        INSERT INTO booking(booking_id, customer_id, event_id, num_tickets, total_cost, booking_date)
        VALUES (?, ?, ?, ?, ?, ?)
        """
        cur.execute(query2, values)

        # Update available seats
        new_seat_count = total_seats - num_tickets
        update_query = "UPDATE event SET total_seats = ? WHERE event_id = ?"
        cur.execute(update_query, (new_seat_count, event_id))

        con.commit()
        print(f"Successfully booked {num_tickets} tickets for {event_name}. Remaining seats: {new_seat_count}")

    def get_all_bookings(self):  1 usage
        query = "SELECT * FROM booking"
        cur.execute(query)
        return cur.fetchall()

    def unique_booking_id(self):  2 usages
        return len(self.get_all_bookings()) + 1
```

```python
    def cancel_tickets(self):  1 usage
        booking_id = int(input("Enter the booking_id: "))

        query = "SELECT booking_id FROM booking WHERE booking_id = ?"
        cur.execute(query, (booking_id,))
        booking = cur.fetchone()
        if not booking:
            print(f"Error: {booking_id} not found")
            return

        query = "SELECT num_tickets FROM booking WHERE booking_id = ?"
        cur.execute(query, (booking_id,))
        num_tickets = cur.fetchone()[0]

        query = "DELETE FROM booking WHERE booking_id = ?"
        cur.execute(query, (booking_id,))
        con.commit()

        print(f"Successfully canceled {num_tickets} tickets.")

# Main interaction loop for the TicketBookingSystem
b = BookingSystemRepository()
```

Output:

```
C:\Users\Shubh\AppData\Local\Programs\Python\Python312\
Database Connected Successfully!!

1. Create Event
2. Book tickets
3. Cancel tickets
4. Get available tickets
5. Get event details
6. Exit
Select from above options:
```

1. Creating an Event:

```
Select from above options: 1
Enter the event id: 15
Enter event name: Winter Carnival
Enter total seats: 200
Enter ticket price: 50
Enter event type (movie, sport, concert): sport
Event created successfully.
(1, 'Jaipur Film Festival', '2024-09-15', '18:00:00.0000000', 1, 500, 450, Decimal('15.00'), 'Movie', 1)
(2, 'Jaipur Cricket Match', '2024-10-10', '15:30:00.0000000', 2, 1000, 800, Decimal('25.00'), 'Sports', 2)
(3, 'Udaipur Music Concert', '2024-11-05', '20:00:00.0000000', 3, 800, 700, Decimal('20.00'), 'Concert', 3)
(4, 'Trishul Dance Show', '2024-11-20', '19:30:00.0000000', 4, 600, 550, Decimal('18.00'), 'Concert', 4)
(5, 'High Act', '2024-10-15', '18:00:00.0000000', 5, 1200, 1000, Decimal('10.00'), 'Drama', 5)
(6, 'Vellore Movie Night', '2024-09-08', '21:00:00.0000000', 6, 300, 280, Decimal('12.00'), 'Movie', 6)
(7, 'Cultural Festival', '2024-10-17', '17:00:00.0000000', 7, 700, 650, Decimal('22.00'), 'Concert', 7)
(8, 'Erode Football Championship', '2024-12-12', '16:45:00.0000000', 8, 1500, 1300, Decimal('30.00'), 'Sports', 8)
(9, 'Rajmandir Art Exhibition', '2024-11-30', '10:30:00.0000000', 9, 400, 380, Decimal('8.00'), 'Concert', 9)
(10, 'JC Comedy Show', '2024-11-18', '19:00:00.0000000', 10, 250, 230, Decimal('15.00'), 'Concert', 10)
(11, 'World Cup Auction', '2024-10-18', '20:00:00.0000000', 7, 1500, 1450, Decimal('1500.00'), 'Sports', 11)
(12, 'Thane Art Exhibition', '2024-12-28', '20:00:00.0000000', 7, 18500, 1450, Decimal('1500.00'), 'Sports', 11)
(13, 'Chennai Music Concert', '2024-11-18', '20:00:00.0000000', 10, 16000, 1450, Decimal('1500.00'), 'Sports', 11)
(14, 'Navratri Garba Night', '2024-10-09', '17:48:37.0000000', None, 595, None, Decimal('100.00'), 'concert', None)
(15, 'Winter Carnival', '2024-10-12', '22:21:00.0000000', None, 200, None, Decimal('50.00'), 'sport', None)
```

2. Booking Event Tickets:

```
Select from above options: 2
Enter the number of tickets: 5
(1, 'Jaipur Film Festival', '2024-09-15', '18:00:00.0000000', 1, 500, 450, Decimal('15.00'), 'Movie', 1)
(2, 'Jaipur Cricket Match', '2024-10-10', '15:30:00.0000000', 2, 1000, 800, Decimal('25.00'), 'Sports', 2)
(3, 'Udaipur Music Concert', '2024-11-05', '20:00:00.0000000', 3, 800, 700, Decimal('20.00'), 'Concert', 3)
(4, 'Trishul Dance Show', '2024-11-20', '19:30:00.0000000', 4, 600, 550, Decimal('18.00'), 'Concert', 4)
(5, 'High Act', '2024-10-15', '18:00:00.0000000', 5, 1200, 1000, Decimal('10.00'), 'Drama', 5)
(6, 'Vellore Movie Night', '2024-09-08', '21:00:00.0000000', 6, 300, 280, Decimal('12.00'), 'Movie', 6)
(7, 'Cultural Festival', '2024-10-17', '17:00:00.0000000', 7, 700, 650, Decimal('22.00'), 'Concert', 7)
(8, 'Erode Football Championship', '2024-12-12', '16:45:00.0000000', 8, 1500, 1300, Decimal('30.00'), 'Sports', 8)
(9, 'Rajmandir Art Exhibition', '2024-11-30', '10:30:00.0000000', 9, 400, 380, Decimal('8.00'), 'Concert', 9)
(10, 'JC Comedy Show', '2024-11-18', '19:00:00.0000000', 10, 250, 230, Decimal('15.00'), 'Concert', 10)
(11, 'World Cup Auction', '2024-10-18', '20:00:00.0000000', 7, 1500, 1450, Decimal('1500.00'), 'Sports', 11)
(12, 'Thane Art Exhibition', '2024-12-28', '20:00:00.0000000', 7, 18500, 1450, Decimal('1500.00'), 'Sports', 11)
(13, 'Chennai Music Concert', '2024-11-18', '20:00:00.0000000', 10, 16000, 1450, Decimal('1500.00'), 'Sports', 11)
(14, 'Navratri Garba Night', '2024-10-09', '17:48:37.0000000', None, 595, None, Decimal('100.00'), 'concert', None)
(15, 'Winter Carnival', '2024-10-12', '22:21:00.0000000', None, 200, None, Decimal('50.00'), 'sport', None)
Enter the event name: Winter Carnival
Enter your name: Subrat Shukla
Enter your email: subrat@email.com
Enter your phone number: 9928211389
Customer created successfully.
Successfully booked 5 tickets for Winter Carnival. Remaining seats: 195
```

3. Cancelling Tickets:

```
1. Create Event
2. Book tickets
3. Cancel tickets
4. Get available tickets
5. Get event details
6. Exit
Select from above options: 3
Enter the booking_id: 12
Successfully canceled 5 tickets.
```

4. Get Available Tickets:

```
Select from above options: 4
(450, 'Jaipur Film Festival')
(800, 'Jaipur Cricket Match')
(700, 'Udaipur Music Concert')
(550, 'Trishul Dance Show')
(1000, 'High Act')
(280, 'Vellore Movie Night')
(650, 'Cultural Festival')
(1300, 'Erode Football Championship')
(380, 'Rajmandir Art Exhibition')
(230, 'JC Comedy Show')
(1450, 'World Cup Auction')
(1450, 'Thane Art Exhibition')
(1450, 'Chennai Music Concert')
(None, 'Navratri Garba Night')
(None, 'Winter Carnival')
```

5. Get Event Details:

```
Select from above options: 5
(1, 'Jaipur Film Festival', '2024-09-15', '18:00:00.0000000', 1, 500, 450, Decimal('15.00'), 'Movie', 1)
(2, 'Jaipur Cricket Match', '2024-10-10', '15:30:00.0000000', 2, 1000, 800, Decimal('25.00'), 'Sports', 2)
(3, 'Udaipur Music Concert', '2024-11-05', '20:00:00.0000000', 3, 800, 700, Decimal('20.00'), 'Concert', 3)
(4, 'Trishul Dance Show', '2024-11-20', '19:30:00.0000000', 4, 600, 550, Decimal('18.00'), 'Concert', 4)
(5, 'High Act', '2024-10-15', '18:00:00.0000000', 5, 1200, 1000, Decimal('10.00'), 'Drama', 5)
(6, 'Vellore Movie Night', '2024-09-08', '21:00:00.0000000', 6, 300, 280, Decimal('12.00'), 'Movie', 6)
(7, 'Cultural Festival', '2024-10-17', '17:00:00.0000000', 7, 700, 650, Decimal('22.00'), 'Concert', 7)
(8, 'Erode Football Championship', '2024-12-12', '16:45:00.0000000', 8, 1500, 1300, Decimal('30.00'), 'Sports', 8)
(9, 'Rajmandir Art Exhibition', '2024-11-30', '10:30:00.0000000', 9, 400, 380, Decimal('8.00'), 'Concert', 9)
(10, 'JC Comedy Show', '2024-11-18', '19:00:00.0000000', 10, 250, 230, Decimal('15.00'), 'Concert', 10)
(11, 'World Cup Auction', '2024-10-18', '20:00:00.0000000', 7, 1500, 1450, Decimal('1500.00'), 'Sports', 11)
(12, 'Thane Art Exhibition', '2024-12-28', '20:00:00.0000000', 7, 18500, 1450, Decimal('1500.00'), 'Sports', 11)
(13, 'Chennai Music Concert', '2024-11-18', '20:00:00.0000000', 10, 16000, 1450, Decimal('1500.00'), 'Sports', 11)
(14, 'Navratri Garba Night', '2024-10-09', '17:48:37.0000000', None, 595, None, Decimal('100.00'), 'concert', None)
(15, 'Winter Carnival', '2024-10-12', '22:21:00.0000000', None, 195, None, Decimal('50.00'), 'sport', None)
```

6. Exiting from system:

```
1. Create Event
2. Book tickets
3. Cancel tickets
4. Get available tickets
5. Get event details
6. Exit
Select from above options: 6
Exiting the system. Thank you!

Process finished with exit code 0
```