```
#!/usr/bin/env python3
import psycopg2

#####################################################
##  Database Connection
#####################################################

'''
Connect to the database using the connection string
'''
def openConnection():
    # connection parameters - ENTER YOUR LOGIN AND PASSWORD HERE
    userid = "y21s1c9120_ssri7591"
    passwd = "500299300"
    myHost = "soit-db-pro-2.ucc.usyd.edu.au"

    # Create a connection to the database
    conn = None
    try:
        # Parses the config file and connects using the connect string
        conn = psycopg2.connect(database=userid,
                                    user=userid,
                                    password=passwd,
                                    host=myHost)
    except psycopg2.Error as sqle:
        print("psycopg2.Error : " + sqle.pgerror)

    # return the connection to use
    return conn

'''
Validate a sales agent login request based on username and password
'''
def checkUserCredentials(userName, password):
    # TODO - validate and get user info for a sales agent

    userInfo = ['2', 'novak', 'Novak', 'Djokovic', '222']

    try:
        conn = openConnection()
        curs = conn.cursor()
        curs.execute('''SELECT * FROM Agent WHERE USERNAME = %(user)s
                             AND PASSWORD = %(password)s''', {'user': userName,
'password': password})
        userInfo = curs.fetchone()
        return userInfo
    except psycopg2.Error as e:
        print('problem connecting to database')
    finally:
        conn.close()




'''
List all the associated bookings in the database for a given sales agent Id
'''
def findBookingsBySalesAgent(agentId):
    # TODO - list all the associated bookings in DB for a given sales agent Id
    try:
        conn = openConnection()
        curs = conn.cursor()
        curs.execute('''SELECT BOOKING_NO, CUSTOMER.FIRSTNAME,CUSTOMER.LASTNAME, 
PERFORMANCE,
                            PERFORMANCE_DATE,AGENT.FIRSTNAME, 
AGENT.LASTNAME,INSTRUCTION
                            FROM BOOKING INNER JOIN CUSTOMER ON BOOKING.CUSTOMER = 
CUSTOMER.EMAIL
                            INNER JOIN AGENT ON BOOKING.BOOKED_BY = AGENT.AGENTID
                            WHERE AGENTID = %(agent)s
                            ORDER BY CUSTOMER.FIRSTNAME''', {'agent': agentId})

        # combining all to execute the right format command

        booking_db = curs.fetchall()

        '''booking_db = [
        ['1', 'Bob Smith', 'The Lion King', '2021-06-05', 'Novak Djokovic', 'I\'d 
like to book 3 additional seats'],
        ['5', 'Mia Clark', 'Disney\'s Frozen', '2021-07-18', 'Novak Djokovic', 
'Please upgrade my seats to Box Seats'],
        ['3', 'Ruby Miller', 'Death of a Salesman', '2021-06-27', 'Novak Djokovic', 
'I want to add meals to my booking']'''

        booking_list = [{
            'booking_no': str(row[0]),
            'customer_name': str(row[1]) + ' ' + str(row[2]),
            'performance': row[3],
            'performance_date': row[4],
            'booked_by': str(row[5]) + ' ' + str(row[6]),
            'instruction': row[7]
        } for row in booking_db]

        return booking_list

    except psycopg2.Error:
        print('Problem connecting to database')
    finally:
        conn.close()
'''
Find a list of bookings based on the searchString provided as parameter
See assignment description for search specification
'''
def findBookingsByCustomerAgentPerformance(searchString):
    # TODO - find a list of bookings in DB based on searchString input

    try:
        conn = openConnection()
        curs = conn.cursor()
        curs.execute('''SELECT BOOKING_NO, CUSTOMER.FIRSTNAME,CUSTOMER.LASTNAME, 
PERFORMANCE,
                            PERFORMANCE_DATE,AGENT.FIRSTNAME, 
AGENT.LASTNAME,INSTRUCTION
                            FROM BOOKING INNER JOIN CUSTOMER ON BOOKING.CUSTOMER = 
CUSTOMER.EMAIL
                            INNER JOIN AGENT ON BOOKING.BOOKED_BY = AGENT.AGENTID
                            WHERE LOWER(AGENT.FIRSTNAME) LIKE CONCAT('%%',%
(searchString)s,'%%') OR
```

```
                            LOWER(AGENT.LASTNAME) LIKE CONCAT('%%',%
(searchString)s,'%%') OR
                            LOWER(BOOKING.PERFORMANCE) LIKE CONCAT('%%',%
(searchString)s,'%%') OR
                            LOWER(CUSTOMER.FIRSTNAME) LIKE CONCAT('%%',%
(searchString)s,'%%') OR
                            LOWER(CUSTOMER.LASTNAME) LIKE CONCAT('%%',%
(searchString)s,'%%')
                            ORDER BY CUSTOMER.FIRSTNAME''', {'searchString':
searchString.lower()})
        booking_db = curs.fetchall()
        print(booking_db)

        booking_list = [{
            'booking_no': str(row[0]),
            'customer_name': str(row[1]) + ' ' + str(row[2]),
            'performance': row[3],
            'performance_date': row[4],
            'booked_by': str(row[5]) + ' ' + str(row[6]),
            'instruction': row[7]
        } for row in booking_db]

        return booking_list


    except psycopg2.Error:
        print('Error connecting to database')

    finally:
        conn.close()


####################################################################################
##
##  Booking (customer, performance, performance date, booking agent, instruction)
##
####################################################################################
##
'''
Add a new booking into the database - details for a new booking provided as
parameters
'''
def addBooking(customer, performance, performance_date, booked_by, instruction):
    # TODO - add a booking
    # Insert a new booking into database
    # return False if adding was unsuccessful
    # return True if adding was successful
    try:
        conn = openConnection()
        curs = conn.cursor()

        curs.execute('''SELECT AGENTID FROM AGENT WHERE USERNAME = %(username)s''',
                     {'username': booked_by})
        # if customer or agent isn't valid, return false
        if curs.fetchone() is None:
            return False
        else:
            # USING STORED PROCEDURE
            curs.callproc("USERNAME_TO_ID", [booked_by])
            agent = curs.fetchone()[0]
            curs.execute('''SELECT EMAIL FROM CUSTOMER WHERE EMAIL = %(email)s''',
{'email': customer})
            if curs.fetchone() is None:
                return False
            else:
                pass
            curs.execute('''INSERT INTO BOOKING
(CUSTOMER,PERFORMANCE,PERFORMANCE_DATE,BOOKED_BY,INSTRUCTION)
                VALUES (%(customer)s, %(performance)s, %(performance_date)s,%
(agent)s
                , %(instruction)s); COMMIT''',
                         {'customer': customer, 'performance': performance,
'performance_date': performance_date,
                          'agent': agent, 'instruction': instruction})
            return True

            # (SELECT AGENTID FROM AGENT WHERE LOWER(FIRSTNAME) = % (booked_by)s)

    except psycopg2.Error:
        return False


    finally:
        conn.close()

    return True


'''
Update an existing booking with the booking details provided in the parameters
'''
def updateBooking(booking_no, performance, performance_date, booked_by,
instruction):
    # TODO - update an existing booking in DB
    # return False if updating was unsuccessful
    # return True if updating was successful

    try:
        conn = openConnection()
        curs = conn.cursor()
        curs.execute('''SELECT AGENTID FROM AGENT WHERE USERNAME = %(username)s''',
                     {'username': booked_by})
        # check if entered agent matches
        if curs.fetchone() is None:
            return False
        else:
            curs.execute('''SELECT AGENTID FROM AGENT WHERE USERNAME = %
(username)s''',
                         {'username': booked_by})
            name_to_id = {}
            name_to_id[booked_by] = int(curs.fetchone()[0])
            curs.execute('''UPDATE BOOKING SET PERFORMANCE = %(performance)s,
PERFORMANCE_DATE = %(performance_date)s,
                BOOKED_BY = %(agent)s,INSTRUCTION = %(instruction)s WHERE
BOOKING_NO = %(booking_no)s;  COMMIT''',
                         {'performance': performance, 'performance_date':
performance_date,
                          'agent': name_to_id[booked_by], 'instruction':
```

```
instruction, 'booking_no': booking_no})
            return True

    except psycopg2.Error:
        return False


    finally:
        conn.close()

    return True
```