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

#####################################################
## GLOBAL Variables
#####################################################

LOGIN_USER_NAME = []

#####################################################
## Additional Functions
#####################################################

def all_to_str(row: list) -> list:
    row = list(map(lambda x: str(x), row))
    return row


def replace_none(row: list) -> list:
    row = list(map(lambda x: '' if x is None or x is ' ' else x, row))
    return row


def make_dict(col: list, row: list) -> dict:
    res = {}
    for i in range(len(col)):
        res[col[i]] = row[i]
    return res

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

'''
Connect to the database using the connection string
'''
def openConnection():
    # connection parameters - ENTER YOUR LOGIN AND PASSWORD HERE

    myHost = "awsprddbs4836.shared.sydney.edu.au"
    userid = "y25s1c9120_hche0960"
    passwd = "NHvincent"

    # 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
    conn.cursor().execute("SET search_path TO sag_webapp;")
    return conn
```

```
'''
Validate salesperson based on username and password
'''
def checkLogin(login, password):
    global LOGIN_USER_NAME
    try:
        with openConnection() as conn:
            cursor = conn.cursor()
            query = "SELECT Password FROM Salesperson WHERE UserName = %s"
            cursor.execute("SELECT * FROM validate_login(%s, %s)", (login,
password))
            return_info = cursor.fetchone()

            if not return_info:
                return None
            else:
                cursor.execute("SELECT FirstName, LastName FROM Salesperson WHERE
UserName = %s", (login,))
                return_info = cursor.fetchone()
                LOGIN_USER_NAME = [return_info[0], return_info[1]]
            return [login, return_info[0], return_info[1]]
    except:
        return None


"""
    Retrieves the summary of car sales.

    This method fetches the summary of car sales from the database and returns it
    as a collection of summary objects. Each summary contains key information
    about a particular car sale.

    :return: A list of car sale summaries.
"""
def getCarSalesSummary():
    try:
        with openConnection() as conn:
            cursor = conn.cursor()
            cursor.execute("SELECT * FROM get_car_sales_summary()")
            sales = cursor.fetchall()
            sales.sort(key=lambda x: (x[0],x[1]))
            sales = [list(r) for r in sales]
            sales = [replace_none(r) for r in sales]
            sales = [all_to_str(r) for r in sales]
            col = ['make', 'model', 'availableUnits', 'soldUnits',
'soldTotalPrices', 'lastPurchaseAt']
            sales = [make_dict(col, r) for r in sales]
            print("INFO: Summary page")
    except:
        return None
    return sales

"""
    Finds car sales based on the provided search string.

    This method searches the database for car sales that match the provided search
    string. See assignment description for search specification

    :param search_string: The search string to use for finding car sales in the
```

```
database.
    :return: A list of car sales matching the search string.
"""
def findCarSales(searchString):
    try:
        with openConnection() as conn:
            cursor = conn.cursor()
            query = """SELECT sale.CarSaleID, make.MakeName, model.ModelName,
sale.BuiltYear,
            sale.Odometer, sale.Price, sale.IsSold, TO_CHAR(sale.SaleDate, 'DD-MM-
YYYY'),
            CONCAT(buyer.FirstName, ' ', buyer.LastName),
            CONCAT(sp.FirstName, ' ', sp.LastName)
            FROM CarSales sale
            JOIN Model model ON model.ModelCode = sale.ModelCode
            JOIN Make make ON make.MakeCode = sale.MakeCode
            LEFT JOIN Customer buyer ON buyer.CustomerID = sale.BuyerID
            LEFT JOIN Salesperson sp ON sp.UserName = sale.SalespersonID
            WHERE
            (model.ModelName ILIKE %s OR
            make.MakeName ILIKE %s OR
            CONCAT(buyer.FirstName, ' ' ,buyer.LastName) ILIKE %s OR
            CONCAT(sp.FirstName, ' ' ,sp.LastName) ILIKE %s) AND NOT
            (sale.IsSold = TRUE AND sale.SaleDate < CURRENT_DATE - INTERVAL '3
years')
            """
            if searchString == '':
                searchString = ' '.join(LOGIN_USER_NAME)
            print(f"INFO: Search {searchString}")
            searchString = f"%{searchString}%"
            search_content = tuple([searchString]*4)
            cursor.execute(query, search_content)
            res = cursor.fetchall()
            res = [list(r) for r in res]
            res = [replace_none(r) for r in res]
            res = [all_to_str(r) for r in res]
            res.sort(key=lambda x: (x[1],x[2]))
            col = ['carsale_id', 'make', 'model', 'builtYear', 'odometer', 'price',
'isSold', 'sale_date',
                    'buyer', 'salesperson']
            res = [make_dict(col, r) for r in res]
            print(f"INFO: Result:\n{res}")
            return res
    except:
        return None

"""
    Adds a new car sale to the database.

    This method accepts a CarSale object, which contains all the necessary details
    for a new car sale. It inserts the data into the database and returns a
confirmation
    of the operation.

    :param car_sale: The CarSale object to be added to the database.
    :return: A boolean indicating if the operation was successful or not.
"""
def addCarSale(make, model, builtYear, odometer, price):
    try:
        with openConnection() as conn:
            cursor = conn.cursor()

            query = """SELECT MakeCode
            FROM Make
            WHERE
            EXISTS(
            SELECT 1 FROM Make
            WHERE MakeName ILIKE %s) AND
            MakeName ILIKE %s"""
            cursor.execute(query, (make, make))
            make_code = cursor.fetchone()[0]
            if not make_code:
                return False

            query = """SELECT ModelCode
            FROM Model
            WHERE
            EXISTS(
            SELECT 1 FROM Model
            WHERE ModelName ILIKE %s) AND
            ModelName ILIKE %s"""
            cursor.execute(query, (model, model))
            model_code = cursor.fetchone()[0]
            if not model_code:
                return False

            # print(model_code, make_code)

            query = """INSERT INTO CarSales (MakeCode, ModelCode, BuiltYear,
Odometer, Price,
            IsSold, BuyerID, SalespersonID, SaleDate)
            VALUES
            (%s, %s, %s, %s, %s, FALSE, NULL, NULL, NULL)"""
            cursor.execute(query, (make_code, model_code, builtYear, odometer,
price))
            conn.commit()
            return True
    except psycopg2.Error as e:
        print(e)
        return False
    except:
        return False

"""
    Updates an existing car sale in the database.

    This method updates the details of a specific car sale in the database,
ensuring
    that all fields of the CarSale object are modified correctly. It assumes that
    the car sale to be updated already exists.

    :param car_sale: The CarSale object containing updated details for the car
sale.
    :return: A boolean indicating whether the update was successful or not.
"""
def updateCarSale(carsaleid, customer, salesperosn, saledate: str):
    try:
        with openConnection() as conn:
            cursor = conn.cursor()
            print(carsaleid, customer, salesperosn, saledate)

            saledate = '-'.join(saledate.split('-')[::-1])

            query = """UPDATE CarSales
            SET IsSold = TRUE,
            BuyerID = LOWER(%s),
            SalespersonID = LOWER(%s),
            SaleDate = TO_DATE(%s, 'DD-MM-YYYY')
            WHERE CarSaleID = %s AND
            TO_DATE(%s, 'DD-MM-YYYY') <= CURRENT_DATE"""
            cursor.execute(query, (customer, salesperosn, saledate, carsaleid,
saledate))
            conn.commit()
            return True
    except psycopg2.Error as e:
        print(e)
        return False
    except:
        return False
```