# Lab 12: SQL Database Connectivity

CS355/CE373 Database Systems
Fall 2024


**Dhanani School of Science and Engineering**
Habib University

# Contents

**1 Instructions** **2**
1.1 Marking scheme . . . . . 2
1.2 Late submission policy . . . . . 2

**2 Objective** **2**

**3 Exercise** **2**

**4 Pyodbc example for CRUD operations** **3**

**5 Skeleton File for Lab Task** **6**

# 1 Instructions

- This lab will contribute 1% towards the final grade.
- The deadline for this project is the end of your lab.
- The lab must be submitted online via CANVAS. You are required to submit a zip file that contains both *.py* and *.ui* files.
- The zip file should be named as *Lab_12_aa1234.zip* where *aa1234* will be replaced with your student id.
- **Files that don't follow the appropriate naming convention will not be graded.**

## 1.1 Marking scheme

This lab will be marked out of 100.

- 50 Marks are for the completion of the lab.
- 50 Marks are for progress and attendance during the lab.

## 1.2 Late submission policy

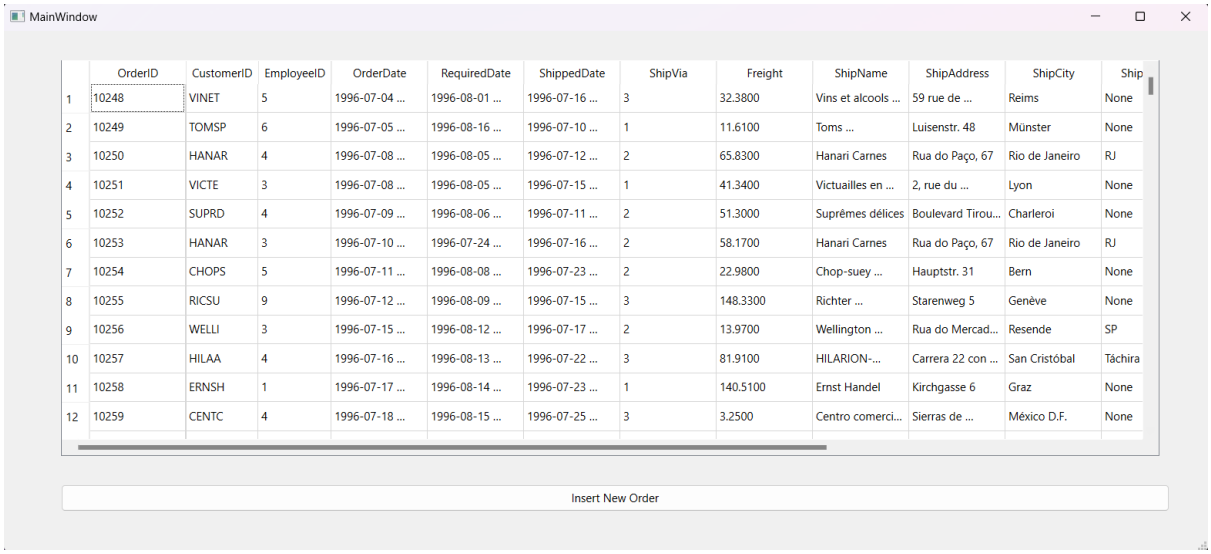
**You can submit late till 11:59 PM on the same day as your lab with a 20% penalty. No submissions will be accepted afterward.**

# 2 Objective

The objective of this lab is to provide hands-on practice on connecting your desktop application in developed in PyQt6 to the Northwind Database using the **pyodbc** library.

# 3 Exercise

In this lab, your task is to create a desktop applications that allows you to insert new orders and view existing orders based on the Northwind Database. You have been skeleton code files and your task to complete the implementations of the functions specified below.




| | OrderID | CustomerID | EmployeeID | OrderDate | RequiredDate | ShippedDate | ShipVia | Freight | ShipName | ShipAddress | ShipCity | Ship |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 10248 | VINET | 5 | 1996-07-04 ... | 1996-08-01 ... | 1996-07-16 ... | 3 | 32.3800 | Vins et alcools ... | 59 rue de ... | Reims | None |
| 2 | 10249 | TOMSP | 6 | 1996-07-05 ... | 1996-08-16 ... | 1996-07-10 ... | 1 | 11.6100 | Toms ... | Luisenstr. 48 | Münster | None |
| 3 | 10250 | HANAR | 4 | 1996-07-08 ... | 1996-08-05 ... | 1996-07-12 ... | 2 | 65.8300 | Hanari Carnes | Rua do Paço, 67 | Rio de Janeiro | RJ |
| 4 | 10251 | VICTE | 3 | 1996-07-08 ... | 1996-08-05 ... | 1996-07-15 ... | 1 | 41.3400 | Victuailles en ... | 2, rue du ... | Lyon | None |
| 5 | 10252 | SUPRD | 4 | 1996-07-09 ... | 1996-08-06 ... | 1996-07-11 ... | 2 | 51.3000 | Suprêmes délices | Boulevard Tirou... | Charleroi | None |
| 6 | 10253 | HANAR | 3 | 1996-07-10 ... | 1996-07-24 ... | 1996-07-16 ... | 2 | 58.1700 | Hanari Carnes | Rua do Paço, 67 | Rio de Janeiro | RJ |
| 7 | 10254 | CHOPS | 5 | 1996-07-11 ... | 1996-08-08 ... | 1996-07-23 ... | 2 | 22.9800 | Chop-suey ... | Hauptstr. 31 | Bern | None |
| 8 | 10255 | RICSU | 9 | 1996-07-12 ... | 1996-08-09 ... | 1996-07-15 ... | 3 | 148.3300 | Richter ... | Starenweg 5 | Genève | None |
| 9 | 10256 | WELLI | 3 | 1996-07-15 ... | 1996-08-12 ... | 1996-07-17 ... | 2 | 13.9700 | Wellington ... | Rua do Mercad... | Resende | SP |
| 10 | 10257 | HILAA | 4 | 1996-07-16 ... | 1996-08-13 ... | 1996-07-22 ... | 3 | 81.9100 | HILARION-... | Carrera 22 con ... | San Cristóbal | Táchira |
| 11 | 10258 | ERNSH | 1 | 1996-07-17 ... | 1996-08-14 ... | 1996-07-23 ... | 1 | 140.5100 | Ernst Handel | Kirchgasse 6 | Graz | None |
| 12 | 10259 | CENTC | 4 | 1996-07-18 ... | 1996-08-15 ... | 1996-07-25 ... | 3 | 3.2500 | Centro comerci... | Sierras de ... | México D.F. | None |

Figure 1: Homepage

The functions that you are required to implement are as follows:

1. `populate_table` which populates the orderTable with data from the Northwind database.
2. `populate_employee_table` which populates the employeesTable with employee data from the Northwind database.
3. `populate_customer_table` which populates the customersTable with customer data from the Northwind database.
4. `populate_ship_table` which populates the shippersTable with shippers data from the Northwind database.
5. `populate_product_table` which populates the productsTable with products data from the Northwind database.
6. `insert_order` which inserts a new order into the Northwind database based on user input.
7. `insert_order_details` which inserts a order details of a new order into the Northwind database based on user input.

In order to implement these functions, you can refer to the code provided in the two sections below.

# 4 Pyodbc example for CRUD operations

```
import pyodbc

# Replace these with your own database connection details
server = 'your_server_name'
database = 'Northwind'  # Name of your Northwind database
use_windows_authentication = True  # Set to True to use Windows Authentication
username = 'your_username'  # Specify a username if not using Windows Authentication
password = 'your_password'  # Specify a password if not using Windows Authentication


# Create the connection string based on the authentication method chosen
```

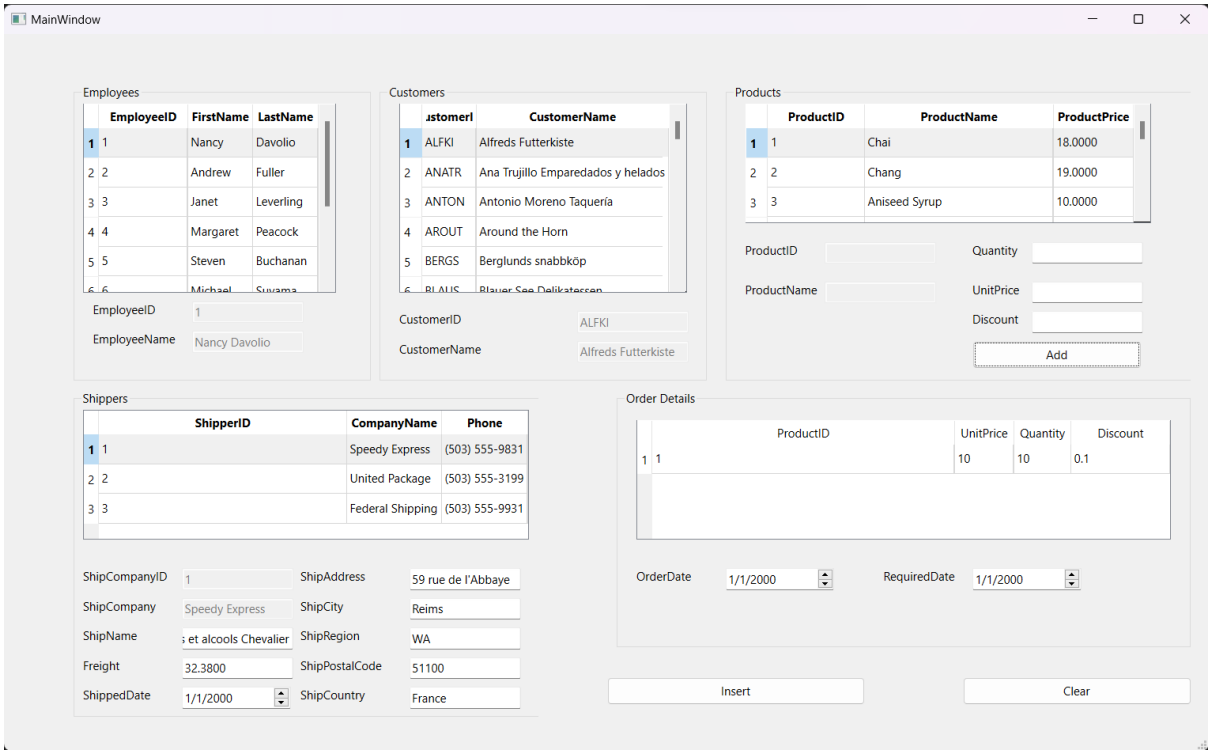



Figure 2: Insert New Order Form

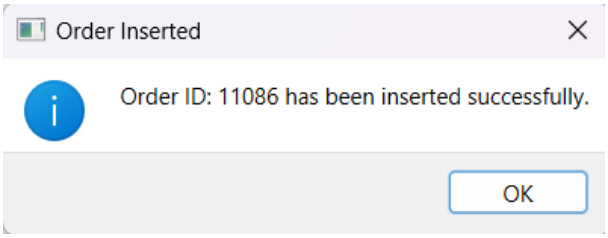



Figure 3: Prompt upon successful creation of a new order.

```
if use_windows_authentication:
    connection_string = f'DRIVER={{ODBC Driver 17 for SQL Server}};SERVER={server};DATABASE={database};Trusted_Connection=yes;'
else:
    connection_string = f'DRIVER={{ODBC Driver 17 for SQL Server}};SERVER={server};DATABASE={database};UID={username};PWD={password}'

# Establish a connection to the database
connection = pyodbc.connect(connection_string)

# Create a cursor to interact with the database
cursor = connection.cursor()

# CREATE - Insert a new employee record
new_employee = (
    'Doe',
    'John',
    'Sales Manager',
    'Mr.',
    '1975-01-15',
    '2023-10-06',
    '123 Main St',
```

```
    'New York',
    'NY',
    '10001',
    'USA',
    '555-123-4567',
    '1234',
    None,  # You can insert the binary image data for the "Photo" field here if needed
    'Notes about John Doe',
    2,  # Replace with the actual ReportsTo value if applicable
    'images/johndoe.jpg'  # Update with the actual file path for the "PhotoPath" field
)
insert_query = """
    INSERT INTO Employees
    ([LastName], [FirstName], [Title], [TitleOfCourtesy], [BirthDate], [HireDate],
    [Address], [City], [Region], [PostalCode], [Country], [HomePhone], [Extension],
    [Photo], [Notes], [ReportsTo], [PhotoPath])
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
"""
cursor.execute(insert_query, new_employee)
connection.commit()  # Commit the transaction

# READ - Fetch and print all employees
select_query = "SELECT FirstName,LastName FROM Employees"
cursor.execute(select_query)
print("All Employees:")
for row in cursor.fetchall():
    print(row)

# UPDATE - Update an employee record
update_query = "UPDATE Employees SET Region = ? WHERE FirstName = ? AND LastName = ?"
updated_region = 'WA'
cursor.execute(update_query, (updated_region, 'John', 'Doe'))
connection.commit()  # Commit the transaction

# READ - Fetch and print the updated employee
cursor.execute("select FirstName,LastName, Region from Employees WHERE FirstName = ? AND LastName = ?",('John', 'Doe'))
print("\nUpdated Employee:")
for row in cursor.fetchall():
    print(row)

# DELETE - Delete an employee record
delete_query = "DELETE FROM Employees WHERE FirstName = ? AND LastName = ?"
cursor.execute(delete_query, ('John', 'Doe'))
connection.commit()  # Commit the transaction

# READ - Fetch and print all employees after deletion
cursor.execute(select_query)
print("\nEmployees After Deletion:")
for row in cursor.fetchall():
    print(row)

# Close the cursor and connection
cursor.close()
connection.close()
```

Listing 1: Skeleton Python file (`example.py`)

# 5 Skeleton File for Lab Task

```
# Importing essential modules
from PyQt6 import QtWidgets, uic
from PyQt6.QtCore import QDate
from PyQt6.QtWidgets import QApplication, QMainWindow, QTableWidget,
    QTableWidgetItem, QVBoxLayout, QWidget, QHeaderView
import sys
import pyodbc


# Main Window Class
class UI(QtWidgets.QMainWindow):
    def __init__(self):
        """
        Initialize the main UI window.

        This constructor is called when an instance of the UI class is created.
        It performs the following tasks:
        1. Calls the constructor of the inherited class.
        2. Loads the user interface (UI) from the 'MainWindow.ui' file.
        3. Populates the 'orderTable' with data.
        4. Connects the "Insert Order" button to the event handler for opening
    the
        master transaction form.

        Note:
        - The 'MainWindow.ui' file should exist and contain the required UI
    elements.

        Returns:
        None
        """
        # Call the inherited classes __init__ method
        super(UI, self).__init__()

        # Load the .ui file
        uic.loadUi('MainWindow.ui', self)

        # Load Orders data
        self.populate_table()

        # Connect Submit Button to Event Handling Code
        self.InsertOrder.clicked.connect(self.open_master_form)

    def populate_table(self):
        """
        Populates the 'orderTable' with data from the Northwind database.

        This function connects to the Northwind database, retrieves orders data
    ,
        and populates the 'orderTable' widget with the fetched data. It also
    adjusts
        the column widths for better content display.

        Note:
        - Ensure that the 'orderTable' widget is set up and available in the UI
    .
        - The database connection parameters (server, database, authentication)
     should
        be correctly configured.

        Returns:
```

```
        None
        """
        # TODO: Provide the  connection string to connect to the Northwind database
        connection = pyodbc.connect(
            ""
        )

        cursor = connection.cursor()

        # TODO: Write SQL query to fetch orders data
        cursor.execute("")

        # Fetch all rows and populate the table
        for row_index, row_data in enumerate(cursor.fetchall()):
            self.orderTable.insertRow(row_index)
            for col_index, cell_data in enumerate(row_data):
                item = QTableWidgetItem(str(cell_data))
                self.orderTable.setItem(row_index, col_index, item)

        # Close the database connection
        connection.close()

        # Adjust content display
        header = self.orderTable.horizontalHeader()
        header.setSectionResizeMode(0, QHeaderView.ResizeMode.Stretch)
        header.setSectionResizeMode(1, QHeaderView.ResizeMode.ResizeToContents)
        header.setSectionResizeMode(2, QHeaderView.ResizeMode.ResizeToContents)

    def open_master_form(self):
        """
        Opens the master transaction form.

        This function is called when the "Insert Order" button is clicked in the main
        UI. It creates and displays the master transaction form (Master class).

        Note:
        - Ensure that the Master class (master_form) is defined and available in the script.

        Returns:
        None
        """
        self.master_form = Master()
        self.master_form.show()


class Master(QtWidgets.QMainWindow):
    def __init__(self):
        """
        Initialize the master transaction form.

        This constructor is called when an instance of the Master class is created.
        It performs the following tasks:
        1. Calls the constructor of the inherited class.
        2. Loads the user interface (UI) from the 'MasterTransactionForm.ui' file.
        3. Populates employee, customer, shipper, and product tables with data.
        4. Sets up event handlers for table selection.
        5. Makes certain input fields read-only or disabled.
        6. Connects buttons like "Add," "Insert," and "Clear" to their
```

```
    respective
113         event handling functions.
114 
115         Note:
116         - The 'MasterTransactionForm.ui' file should exist and contain the
    required UI elements.
117 
118         Returns:
119         None
120         """
121         # Call the inherited classes __init__ method
122         super(Master, self).__init__()
123 
124         # Load the .ui file
125         uic.loadUi('MasterTransactionForm.ui', self)
126 
127         # Load employee table
128         self.populate_employee_table()
129 
130         # Load customer table
131         self.populate_customer_table()
132 
133         # Load shippers table
134         self.populate_ship_table()
135 
136         # Load products table
137         self.populate_product_table()
138 
139         # Employee Table Row selected
140         self.employeesTable.itemSelectionChanged.connect(
141             self.get_selected_employee_data)
142 
143         # Customer Table Selected
144         self.customersTable.itemSelectionChanged.connect(
145             self.get_selected_customer_data)
146 
147         # Product Table Selected
148         self.productsTable.itemSelectionChanged.connect(
149             self.get_selected_product_data)
150 
151         # Shipper Table Selected
152         self.shippersTable.itemSelectionChanged.connect(
153             self.get_selected_shipper_data)
154 
155         # Make EmployeeID and EmployeeName readonly
156         self.EmployeeID.setDisabled(True)
157         self.EmployeeName.setDisabled(True)
158 
159         # Make CustomerID and CustomerName readonly
160         self.CustomerID.setDisabled(True)
161         self.CustomerName.setDisabled(True)
162 
163         # Make ProductID and ProductName readonly
164         self.ProductID.setDisabled(True)
165         self.ProductName.setDisabled(True)
166 
167         # Make ShipCompanyID and ShipCompany readonly
168         self.ShipCompanyID.setDisabled(True)
169         self.ShipCompany.setDisabled(True)
170 
171         # Add Row to Product Table
172         self.Add.clicked.connect(self.add_product)
173 
```

```
        # Insert Product
        self.Insert.clicked.connect(self.insert_order)

        # Clear
        self.Clear.clicked.connect(self.clear)

    def populate_employee_table(self):
        """
        Populates the 'employeesTable' with employee data from the Northwind database.

        This function connects to the Northwind database, retrieves employee data,
        and populates the 'employeesTable' widget with the fetched data. It also adjusts
        the column widths for better content display.

        Note:
        - Ensure that the 'employeesTable' widget is set up and available in the UI.
        - The database connection parameters (server, database, authentication) should
        be correctly configured.

        Returns:
        None
        """
        # TODO: Provide the  connection string to connect to the Northwind database
        connection = pyodbc.connect(
            ""
        )

        cursor = connection.cursor()

        # TODO: Write SQL query to fetch employee data
        cursor.execute("")

        # Fetch all rows and populate the table
        for row_index, row_data in enumerate(cursor.fetchall()):
            self.employeesTable.insertRow(row_index)
            for col_index, cell_data in enumerate(row_data):
                item = QTableWidgetItem(str(cell_data))
                self.employeesTable.setItem(row_index, col_index, item)

        # Close the database connection
        connection.close()

        # Adjust content display
        header = self.employeesTable.horizontalHeader()
        header.setSectionResizeMode(0, QHeaderView.ResizeMode.Stretch)
        header.setSectionResizeMode(1, QHeaderView.ResizeMode.ResizeToContents)
        header.setSectionResizeMode(2, QHeaderView.ResizeMode.ResizeToContents)

    def populate_customer_table(self):
        """
        Populates the 'customersTable' with customer data from the Northwind database.

        This function connects to the Northwind database, retrieves customer data,
        and populates the 'customersTable' widget with the fetched data. It also adjusts
```

```
        the column widths for better content display.

        Note:
        - Ensure that the 'customersTable' widget is set up and available in the UI.
        - The database connection parameters (server, database, authentication) should
        be correctly configured.

        Returns:
        None
        """
        # TODO: Provide the  connection string to connect to the Northwind database
        connection = pyodbc.connect(
            ""
        )

        cursor = connection.cursor()

        # TODO: Write SQL query to fetch customers data
        cursor.execute("")

        # Fetch all rows and populate the table
        for row_index, row_data in enumerate(cursor.fetchall()):
            self.customersTable.insertRow(row_index)
            for col_index, cell_data in enumerate(row_data):
                item = QTableWidgetItem(str(cell_data))
                self.customersTable.setItem(row_index, col_index, item)

        # Close the database connection
        connection.close()

        # Adjust content display
        header = self.customersTable.horizontalHeader()
        header.setSectionResizeMode(0, QHeaderView.ResizeMode.Stretch)
        header.setSectionResizeMode(1, QHeaderView.ResizeMode.ResizeToContents)
        header.setSectionResizeMode(2, QHeaderView.ResizeMode.ResizeToContents)

    def populate_ship_table(self):
        """
        Populate the shippers table with data from the Northwind database.

        This function connects to the Northwind database, executes an SQL query to fetch
        data from the Shippers table, and populates the shippersTable widget with the
        retrieved data. It also adjusts the column widths for better display.

        Returns:
        None
        """
        # TODO: Provide the  connection string to connect to the Northwind database
        connection = pyodbc.connect(
            ""
        )

        cursor = connection.cursor()

        # TODO: Write SQL query to fetch customers data
        cursor.execute("")

```

```
        # Fetch all rows and populate the table
        for row_index, row_data in enumerate(cursor.fetchall()):
            self.shippersTable.insertRow(row_index)
            for col_index, cell_data in enumerate(row_data):
                item = QTableWidgetItem(str(cell_data))
                self.shippersTable.setItem(row_index, col_index, item)

        # Close the database connection
        connection.close()

        # Adjust the column widths for better display
        header = self.shippersTable.horizontalHeader()
        header.setSectionResizeMode(0, QHeaderView.ResizeMode.Stretch)
        header.setSectionResizeMode(1, QHeaderView.ResizeMode.ResizeToContents)
        header.setSectionResizeMode(2, QHeaderView.ResizeMode.ResizeToContents)

    def populate_product_table(self):
        """
        Populate the products table with data from the Northwind database.

        This function connects to the Northwind database, executes an SQL query to fetch
        specific data (ProductID, ProductName, UnitPrice) from the Products table, and
        populates the productsTable widget with the retrieved data. It also adjusts
        the column widths for better display.

        Returns:
        None
        """
        # TODO: Provide the  connection string to connect to the Northwind database
        connection = pyodbc.connect(

        )

        cursor = connection.cursor()

        # TODO: Write SQL query to fetch customers data
        cursor.execute("")

        # Fetch all rows and populate the table
        for row_index, row_data in enumerate(cursor.fetchall()):
            self.productsTable.insertRow(row_index)
            for col_index, cell_data in enumerate(row_data):
                item = QTableWidgetItem(str(cell_data))
                self.productsTable.setItem(row_index, col_index, item)

        # Close the database connection
        connection.close()

        # Adjust the column widths for better display
        header = self.productsTable.horizontalHeader()
        header.setSectionResizeMode(0, QHeaderView.ResizeMode.Stretch)
        header.setSectionResizeMode(1, QHeaderView.ResizeMode.ResizeToContents)
        header.setSectionResizeMode(2, QHeaderView.ResizeMode.ResizeToContents)

    def get_selected_employee_data(self):
        """
        Retrieve and display information of the selected employee.

        This function gets the index of the selected row in the employees table
```

```
    ,
        retrieves the Employee ID, First Name, and Last Name of the selected employee,
        and displays this information in the corresponding input fields.

        Returns:
        None
        """
        # Get the index of the selected row
        selected_row = self.employeesTable.currentRow()
        # Employee ID
        EmployeeID = self.employeesTable.item(selected_row, 0).text()
        # First Name
        FirstName = self.employeesTable.item(selected_row, 1).text()
        # Last Name
        LastName = self.employeesTable.item(selected_row, 2).text()
        # Set Employee ID
        self.EmployeeID.setText(EmployeeID)
        # Set Employee Name
        self.EmployeeName.setText(FirstName + " " + LastName)

    def get_selected_customer_data(self):
        """
        Retrieve and display information of the selected customer.

        This function gets the index of the selected row in the customers table,
        retrieves the Customer ID and Company Name of the selected customer, and
        displays this information in the corresponding input fields.

        Returns:
        None
        """
        # Get the index of the selected row
        selected_row = self.customersTable.currentRow()
        # Customer ID
        customersID = self.customersTable.item(selected_row, 0).text()
        # Company Name
        customersName = self.customersTable.item(selected_row, 1).text()
        # Set Customer ID
        self.CustomerID.setText(customersID)
        # Set Customer Name
        self.CustomerName.setText(customersName)

    def get_selected_product_data(self):
        """
        Retrieve and display information of the selected product.

        This function gets the index of the selected row in the products table,
        retrieves the Product ID and Product Name of the selected product, and
        displays this information in the corresponding input fields.

        Returns:
        None
        """
        # Get the index of the selected row
        selected_row = self.productsTable.currentRow()
        # Product ID
        ProductID = self.productsTable.item(selected_row, 0).text()
        # Product Name
        ProductName = self.productsTable.item(selected_row, 1).text()
        # Set Product ID
```

```
        self.ProductID.setText(ProductID)
        # Set Product Name
        self.ProductName.setText(ProductName)

    def get_selected_shipper_data(self):
        """
        Retrieve and display information of the selected shipper.

        This function gets the index of the selected row in the shippers table,
        retrieves the Ship Company ID and Ship Company Name of the selected shipper,
        and displays this information in the corresponding input fields.

        Returns:
        None
        """
        # Get the index of the selected row
        selected_row = self.shippersTable.currentRow()
        # Ship Company ID
        ShipCompanyID = self.shippersTable.item(selected_row, 0).text()
        # Ship Company Name
        ShipCompanyName = self.shippersTable.item(selected_row, 1).text()
        # Set ShipCompany ID
        self.ShipCompanyID.setText(ShipCompanyID)
        # Set ShipCompany Name
        self.ShipCompany.setText(ShipCompanyName)

    def add_product(self):
        """
        Add a product to the order details table.

        This function adds a new row to the order details table in the user interface
        and populates it with product information entered by the user, such as
        Product ID, Unit Price, Quantity, and Discount. It also adjusts the column
        widths for proper display and clears the input fields for the next entry.

        Note:
        - Ensure that the relevant input fields are correctly configured in the UI.

        Returns:
        None
        """
        row_position = self.orderDetailsTable.rowCount()
        self.orderDetailsTable.insertRow(row_position)

        self.orderDetailsTable.setItem(
            row_position, 0, QTableWidgetItem(self.ProductID.text()))
        self.orderDetailsTable.setItem(
            row_position, 1, QTableWidgetItem(self.UnitPrice.text()))
        self.orderDetailsTable.setItem(
            row_position, 2, QTableWidgetItem(self.Quantity.text()))
        self.orderDetailsTable.setItem(
            row_position, 3, QTableWidgetItem(self.Discount.text()))

        header = self.orderDetailsTable.horizontalHeader()
        header.setSectionResizeMode(0, QHeaderView.ResizeMode.Stretch)
        header.setSectionResizeMode(1, QHeaderView.ResizeMode.ResizeToContents)
        header.setSectionResizeMode(2, QHeaderView.ResizeMode.ResizeToContents)

```

```
        self.ProductID.setText("")
        self.ProductName.setText("")
        self.UnitPrice.setText("")
        self.Quantity.setText("")
        self.Discount.setText("")

    def insert_order(self):
        """
        Insert a new order into the Northwind database based on user input.

        This function retrieves order information from various input fields, such as
        Customer ID, Employee ID, shipping details, and others. It constructs an SQL
        query with parameters to insert a new order into the 'Orders' table of the
        Northwind database. After successfully inserting the order, it retrieves the
        newly assigned Order ID and displays it in a message box.

        Note:
        - Ensure that the relevant input fields are correctly configured in the UI.
        - The function assumes that the database connection is already established.

        Returns:
        None
        """
        # Get order information from input fields
        CustomerID = self.CustomerID.text()
        EmployeeID = self.EmployeeID.text()
        OrderDate = self.OrderDate.date().toString("yyyy-MM-dd")
        RequiredDate = self.RequiredDate.date().toString("yyyy-MM-dd")
        ShippedDate = self.ShippedDate.date().toString("yyyy-MM-dd")
        ShipVia = self.ShipCompanyID.text()
        Freight = self.Freight.text()
        ShipName = self.ShipName.text()
        ShipAddress = self.ShipAddress.text()
        ShipCity = self.ShipCity.text()
        ShipRegion = self.ShipRegion.text()  # Corrected from ShipName.text()
        ShipPostalCode = self.ShipPostalCode.text()
        ShipCountry = self.ShipCountry.text()

        # TODO: Provide the  connection string to connect to the Northwind database
        connection = pyodbc.connect(
            ""
        )

        cursor = connection.cursor()

        # TODO: Write SQL query with parameters to insert order
        sql_query = """

        """

        # Execute the SQL query with parameter values
        cursor.execute(sql_query, (CustomerID, int(EmployeeID), OrderDate, RequiredDate, ShippedDate, int(
            ShipVia), float(Freight), ShipName, ShipAddress, ShipCity, ShipRegion, ShipPostalCode, ShipCountry))
        connection.commit()
```

```
515
516         # Retrieve the newly inserted order ID
517         cursor.execute("SELECT max(orderid) AS OrderID from orders")
518         result = cursor.fetchone()
519         order_id = result[0]
520
521         # Show a message box with the order ID
522         QtWidgets.QMessageBox.information(
523             self, "Order Inserted", f"Order ID: {order_id} has been inserted
    successfully.")
524
525         # Close the database connection
526         connection.close()
527         self.insert_order_details(order_id)
528
529     def insert_order_details(self, order_id):
530         """
531         Inserts order details into the 'Order Details' table for a given order
    ID.
532
533         This function is responsible for inserting the order details, including
     Product ID,
534         Unit Price, Quantity, and Discount, into the 'Order Details' table in
    the Northwind
535         database. The order details are associated with the specified order ID.
536
537         Parameters:
538         - order_id (int): The unique identifier of the order for which details
    are being inserted.
539
540         Note:
541         - Ensure that the 'orderDetailsTable' is correctly populated with the
    order details.
542         - The database connection parameters (server, database, authentication)
     should be
543         correctly configured.
544
545         Returns:
546         None
547         """
548         # TODO: Provide the  connection string to connect to the Northwind
    database
549         connection = pyodbc.connect(
550             ""
551         )
552
553         cursor = connection.cursor()
554         num_rows = self.orderDetailsTable.rowCount()
555
556         # Iterate through the rows of 'orderDetailsTable' and insert order
    details
557         for row in range(num_rows):
558             ProductID = int(self.orderDetailsTable.item(row, 0).text())
559             UnitPrice = float(self.orderDetailsTable.item(row, 1).text())
560             Quantity = int(self.orderDetailsTable.item(row, 2).text())
561             Discount = float(self.orderDetailsTable.item(row, 3).text())
562
563             # TODO: Write SQL query with parameters to insert orders details
564             sql_query = ""
565             # Execute the SQL query with parameter values
566             cursor.execute(sql_query, (int(order_id), ProductID,
567                            UnitPrice, Quantity, Discount))
568             connection.commit()
```

```
569
570         # Close the database connection
571         connection.close()
572
573         # Clear the form after successfully inserting order details
574         self.clear()
575
576     def clear(self):
577         """
578         Clears and resets all input fields and tables in the master transaction
     form.
579
580         This function resets all input fields, including Employee ID, Customer
    ID, Product ID,
581         and other related fields, to their initial state or empty values. It
    also clears
582         the content of the 'orderDetailsTable' and resets date fields to a
    default date.
583         This function is typically used to clear the form after an order has
    been submitted
584         or when the user wants to start with a fresh order entry.
585
586         Note:
587         - Ensure that the relevant input fields and tables are available in the
     UI.
588
589         Returns:
590         None
591         """
592         self.EmployeeID.setText("")
593         self.EmployeeName.setText("")
594         self.CustomerID.setText("")
595         self.CustomerName.setText("")
596         self.ProductID.setText("")
597         self.ProductName.setText("")
598         self.Quantity.setText("")
599         self.UnitPrice.setText("")
600         self.Discount.setText("")
601         self.orderDetailsTable.clearContents()
602         self.ShipCompanyID.setText("")
603         self.ShipCompany.setText("")
604         self.ShipName.setText("")
605         self.Freight.setText("")
606         self.ShippedDate.setDate(QDate(2000, 1, 1))
607         self.OrderDate.setDate(QDate(2000, 1, 1))
608         self.RequiredDate.setDate(QDate(2000, 1, 1))
609         self.ShipAddress.setText("")
610         self.ShipCity.setText("")
611         self.ShipRegion.setText("")
612         self.ShipPostalCode.setText("")
613         self.ShipCountry.setText("")
614
615
616 def main():
617     app = QApplication(sys.argv)
618     window = UI()
619     window.show()
620     sys.exit(app.exec())
621
622
623 if __name__ == "__main__":
```

```
624     main()
```

Listing 2: Skeleton Python file (`main.py`)