ASSIGNMENT

By

Diya Bhat (2022a1r017)

Shoieb Ali (2022a1r014)

Ishika Razdan (2022a1r016)

Prachi Khajuria (2022a1r015)

4th Semester

CSE Department

Model Institute of Engineering & Technology (Autonomous)

(Permanently Affiliated to the University of Jammu, Accredited by NAAC with "A" Grade)

Jammu, India 2023

ASSIGNMENT

GROUP - 4

Subject Code: Relational Database Management System

Due Date: 24-05-24

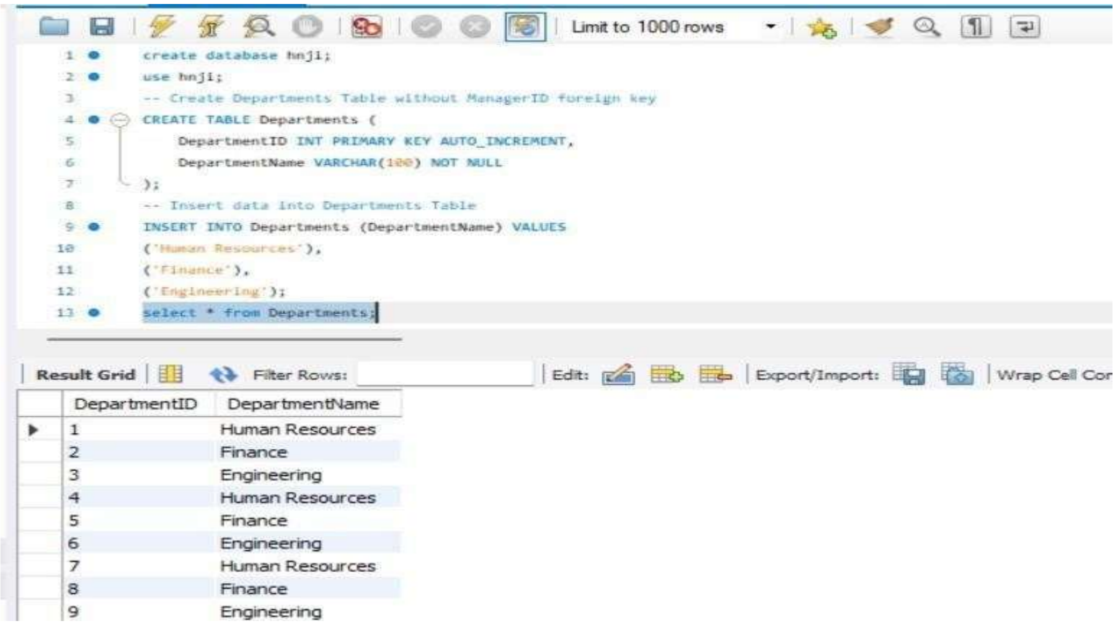
Question Number	Course Outcomes	Blooms' Level	Maximum Marks	Marks Obtain
Q1, Q2, Q3	CO 3, CO 4, CO 5	3-6	20	
Total Marks			20	
Faculty Signature: Mr. Navin M Upadhyay Email: navin.cse@mietjammu.in				

Assignment 1 -- SQL as a Data Manipulation Language Using MySQL:

1. Create the tables for the Company database in your text, and populate them with data.

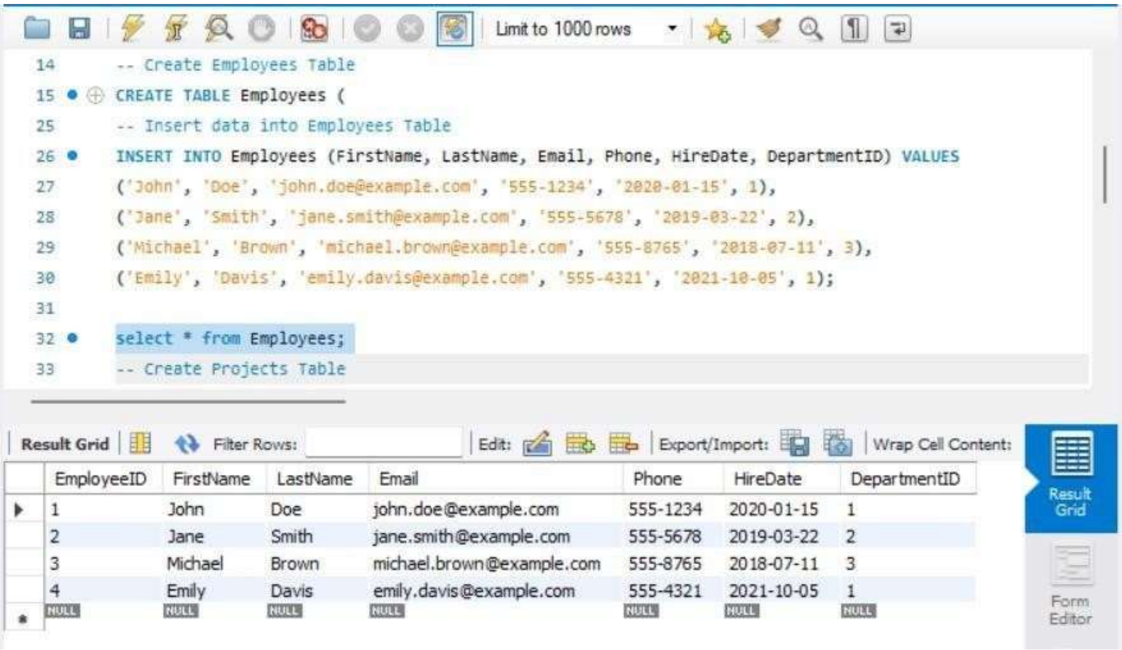
ANS 1: CODE :-

```
create database hnji; use hnji;
-- Create Departments Table without ManagerID foreign key CREATE TABLE Departments (
DepartmentID INT PRIMARY KEY AUTO_INCREMENT,
DepartmentName VARCHAR(100) NOT NULL
);
-- Insert data into Departments Table
INSERT INTO Departments (DepartmentName) VALUES ('Human Resources'),
('Finance'),
('Engineering');
select * from Departments;

-- Create Employees Table CREATE TABLE Employees (
EmployeeID INT PRIMARY KEY AUTO_INCREMENT, FirstName VARCHAR(50) NOT NULL,
LastName VARCHAR(50) NOT NULL,
Email VARCHAR(100) NOT NULL UNIQUE, Phone VARCHAR(15),
HireDate DATE, DepartmentID INT,
FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);
-- Insert data into Employees Table
INSERT INTO Employees (FirstName, LastName, Email, Phone, HireDate, DepartmentID) VALUES
('John', 'Doe', 'john.doe@example.com', '555-1234', '2020-01-15', 1),
('Jane', 'Smith', 'jane.smith@example.com', '555-5678', '2019-03-22', 2),
('Michael', 'Brown', 'michael.brown@example.com', '555-8765', '2018-07-11', 3),
('Emily', 'Davis', 'emily.davis@example.com', '555-4321', '2021-10-05', 1); select * from Employess;

-- Create Projects Table CREATE TABLE Projects (
ProjectID INT PRIMARY KEY AUTO_INCREMENT, ProjectName VARCHAR(100) NOT NULL,
StartDate DATE,
EndDate DATE
);
-- Insert data into Projects Table
INSERT INTO Projects (ProjectName, StartDate, EndDate) VALUES ('Project Alpha', '2023-01-01',
'2023-12-31'),
('Project Beta', '2023-02-01', '2023-11-30'),
('Project Gamma', '2023-03-01', '2023-10-31');
select * from projects;

-- Create Salaries Table CREATE TABLE Salaries (
EmployeeID INT PRIMARY KEY,
SalaryAmount DECIMAL(10, 2) NOT NULL, EffectiveDate DATE NOT NULL,
FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)
);
-- Insert data into Salaries Table
INSERT INTO Salaries (EmployeeID, SalaryAmount, EffectiveDate) VALUES (1, 60000.00, '2023-
01-01'),
(2, 70000.00, '2023-01-01'),
(3, 80000.00, '2023-01-01'),
(4, 50000.00, '2023-01-01');
select * from salaries;
```



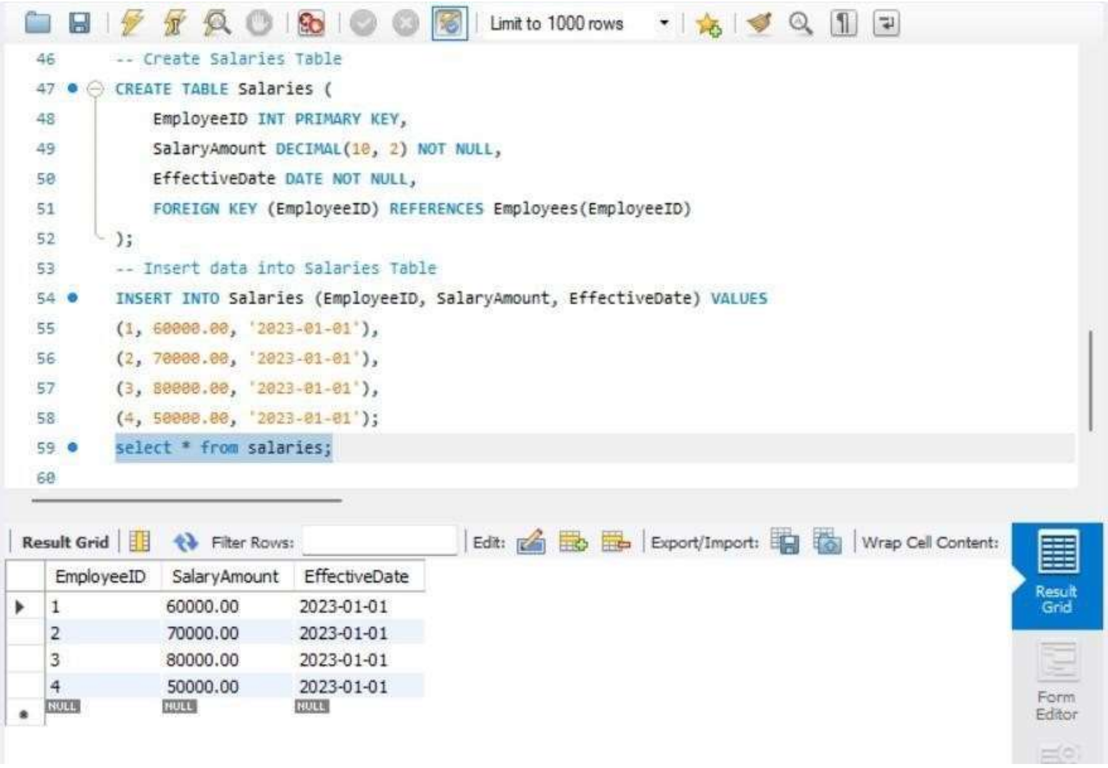
```
create database hnji;
use hnji;
-- Create Departments Table without ManagerID foreign key
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY AUTO_INCREMENT,
    DepartmentName VARCHAR(100) NOT NULL
);
-- Insert data into Departments Table
INSERT INTO Departments (DepartmentName) VALUES
('Human Resources'),
('Finance'),
('Engineering');
select * from Departments;
```

DepartmentID	DepartmentName
1	Human Resources
2	Finance
3	Engineering
4	Human Resources
5	Finance
6	Engineering
7	Human Resources
8	Finance
9	Engineering



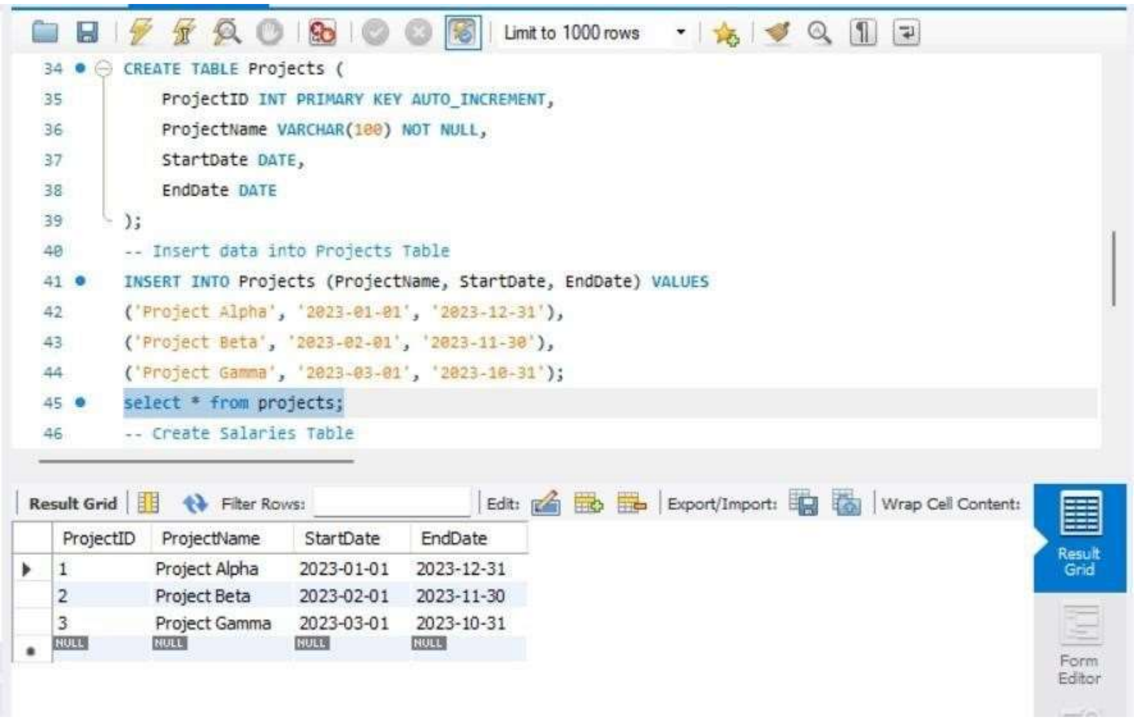
```sql
-- Create Employees Table
CREATE TABLE Employees (
-- Insert data into Employees Table
INSERT INTO Employees (FirstName, LastName, Email, Phone, HireDate, DepartmentID) VALUES
('John', 'Doe', 'john.doe@example.com', '555-1234', '2020-01-15', 1),
('Jane', 'Smith', 'jane.smith@example.com', '555-5678', '2019-03-22', 2),
('Michael', 'Brown', 'michael.brown@example.com', '555-8765', '2018-07-11', 3),
('Emily', 'Davis', 'emily.davis@example.com', '555-4321', '2021-10-05', 1);

select * from Employees;
-- Create Projects Table
```

EmployeeID	FirstName	LastName	Email	Phone	HireDate	DepartmentID
1	John	Doe	john.doe@example.com	555-1234	2020-01-15	1
2	Jane	Smith	jane.smith@example.com	555-5678	2019-03-22	2
3	Michael	Brown	michael.brown@example.com	555-8765	2018-07-11	3
4	Emily	Davis	emily.davis@example.com	555-4321	2021-10-05	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL



```sql
-- Create Salaries Table
CREATE TABLE Salaries (
    EmployeeID INT PRIMARY KEY,
    SalaryAmount DECIMAL(10, 2) NOT NULL,
    EffectiveDate DATE NOT NULL,
    FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)
);
-- Insert data into Salaries Table
INSERT INTO Salaries (EmployeeID, SalaryAmount, EffectiveDate) VALUES
(1, 60000.00, '2023-01-01'),
(2, 70000.00, '2023-01-01'),
(3, 80000.00, '2023-01-01'),
(4, 50000.00, '2023-01-01');
select * from salaries;
```

EmployeeID	SalaryAmount	EffectiveDate
1	60000.00	2023-01-01
2	70000.00	2023-01-01
3	80000.00	2023-01-01
4	50000.00	2023-01-01
NULL	NULL	NULL



```
CREATE TABLE Projects (
    ProjectID INT PRIMARY KEY AUTO_INCREMENT,
    ProjectName VARCHAR(100) NOT NULL,
    StartDate DATE,
    EndDate DATE
);
-- Insert data into Projects Table
INSERT INTO Projects (ProjectName, StartDate, EndDate) VALUES
('Project Alpha', '2023-01-01', '2023-12-31'),
('Project Beta', '2023-02-01', '2023-11-30'),
('Project Gamma', '2023-03-01', '2023-10-31');
select * from projects;
-- Create Salaries Table
```

ProjectID	ProjectName	StartDate	EndDate
1	Project Alpha	2023-01-01	2023-12-31
2	Project Beta	2023-02-01	2023-11-30
3	Project Gamma	2023-03-01	2023-10-31
NULL	NULL	NULL	NULL

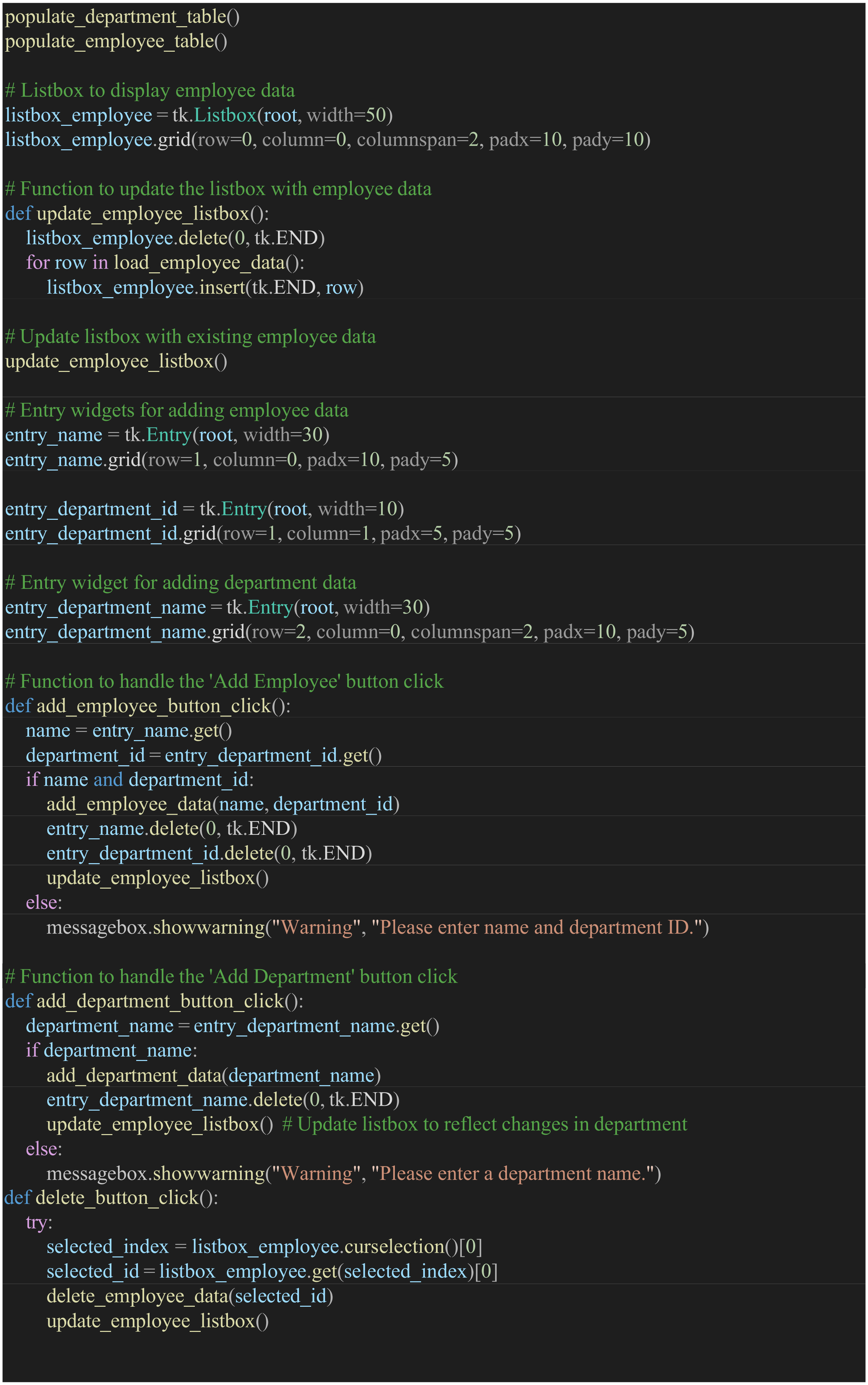
2. **Create a simple desktop app to load, add, and delete the data from the database. [Use any language Python tk, c#, .net, etc.]**

ANS 2: -

<u>Code for creating GUI Application :</u>

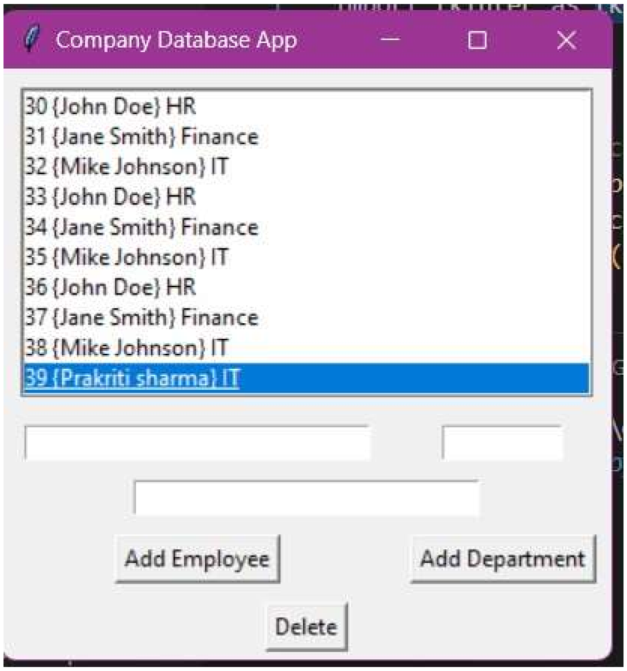
```
import tkinter as tk
from tkinter import messagebox
import sqlite3

# Function to connect to the Company database
def connect_to_database():
    conn = sqlite3.connect('company.db')
    c = conn.cursor()
    return conn, c

# Function to create the Department table if not exists
def create_department_table():
    conn, c = connect_to_database()
    c.execute('''CREATE TABLE IF NOT EXISTS Department (
                id INTEGER PRIMARY KEY,
                name TEXT)''')
    conn.commit()
    conn.close()

# Function to create the Employee table if not exists
def create_employee_table():
    conn, c = connect_to_database()
    c.execute('''CREATE TABLE IF NOT EXISTS Employee (
                id INTEGER PRIMARY KEY,
                name TEXT,
                department_id INTEGER,
                FOREIGN KEY (department_id) REFERENCES Department(id))''')
    conn.commit()
    conn.close()

# Function to populate the Department table with sample data
def populate_department_table():
```

```python
    conn, c = connect_to_database()
    c.execute("INSERT INTO Department (name) VALUES ('HR')")
    c.execute("INSERT INTO Department (name) VALUES ('Finance')")
    c.execute("INSERT INTO Department (name) VALUES ('IT')")
    conn.commit()
    conn.close()

# Function to populate the Employee table with sample data
def populate_employee_table():
    conn, c = connect_to_database()
    c.execute("INSERT INTO Employee (name, department_id) VALUES ('John Doe', 1)")
    c.execute("INSERT INTO Employee (name, department_id) VALUES ('Jane Smith', 2)")
    c.execute("INSERT INTO Employee (name, department_id) VALUES ('Mike Johnson', 3)")
    conn.commit()
    conn.close()

# Function to load all data from the Employee table
def load_employee_data():
    conn, c = connect_to_database()
    c.execute("SELECT Employee.id, Employee.name, Department.name FROM Employee INNER JOIN Department ON Employee.department_id = Department.id")
    rows = c.fetchall()
    conn.close()
    return rows

# Function to add data to the Employee table
def add_employee_data(name, department_id):
    conn, c = connect_to_database()
    c.execute("INSERT INTO Employee (name, department_id) VALUES (?, ?)", (name, department_id))
    conn.commit()
    conn.close()

# Function to add department to the Department table
def add_department_data(department_name):
    conn, c = connect_to_database()
    c.execute("INSERT INTO Department (name) VALUES (?)", (department_name,))
    conn.commit()
    conn.close()

# Function to delete data from the Employee table
def delete_employee_data(id):
    conn, c = connect_to_database()
    c.execute("DELETE FROM Employee WHERE id=?", (id,))
    conn.commit()
    conn.close()

# Main tkinter window
root = tk.Tk()
root.title("Company Database App")

# Create Department and Employee tables if not exists
create_department_table()
create_employee_table()
```



```
populate_department_table()
populate_employee_table()

# Listbox to display employee data
listbox_employee = tk.Listbox(root, width=50)
listbox_employee.grid(row=0, column=0, columnspan=2, padx=10, pady=10)

# Function to update the listbox with employee data
def update_employee_listbox():
    listbox_employee.delete(0, tk.END)
    for row in load_employee_data():
        listbox_employee.insert(tk.END, row)

# Update listbox with existing employee data
update_employee_listbox()

# Entry widgets for adding employee data
entry_name = tk.Entry(root, width=30)
entry_name.grid(row=1, column=0, padx=10, pady=5)

entry_department_id = tk.Entry(root, width=10)
entry_department_id.grid(row=1, column=1, padx=5, pady=5)

# Entry widget for adding department data
entry_department_name = tk.Entry(root, width=30)
entry_department_name.grid(row=2, column=0, columnspan=2, padx=10, pady=5)

# Function to handle the 'Add Employee' button click
def add_employee_button_click():
    name = entry_name.get()
    department_id = entry_department_id.get()
    if name and department_id:
        add_employee_data(name, department_id)
        entry_name.delete(0, tk.END)
        entry_department_id.delete(0, tk.END)
        update_employee_listbox()
    else:
        messagebox.showwarning("Warning", "Please enter name and department ID.")

# Function to handle the 'Add Department' button click
def add_department_button_click():
    department_name = entry_department_name.get()
    if department_name:
        add_department_data(department_name)
        entry_department_name.delete(0, tk.END)
        update_employee_listbox()  # Update listbox to reflect changes in department
    else:
        messagebox.showwarning("Warning", "Please enter a department name.")
def delete_button_click():
    try:
        selected_index = listbox_employee.curselection()[0]
        selected_id = listbox_employee.get(selected_index)[0]
        delete_employee_data(selected_id)
        update_employee_listbox()
    except IndexError:
        messagebox.showwarning("Warning", "Please select an item to delete.")

# 'Add Employee' button
add_employee_button = tk.Button(root, text="Add Employee",
command=add_employee_button_click)
add_employee_button.grid(row=3, column=0, padx=5, pady=5)

# 'Add Department' button
add_department_button = tk.Button(root, text="Add Department",
command=add_department_button_click)
add_department_button.grid(row=3, column=1, padx=5, pady=5)

# 'Delete' button
delete_button = tk.Button(root, text="Delete", command=delete_button_click)
delete_button.grid(row=4, column=0, columnspan=2, padx=5, pady=5)

root.mainloop()
```

OUTPUT:



```
30 {John Doe} HR
31 {Jane Smith} Finance
32 {Mike Johnson} IT
33 {John Doe} HR
34 {Jane Smith} Finance
35 {Mike Johnson} IT
36 {John Doe} HR
37 {Jane Smith} Finance
38 {Mike Johnson} IT
39 {Prakriti sharma} IT
```

EXPLANATION: -

- **Functionality:** The application allows users to manage employee data within a company database. It provides features to add new employees, delete existing employees, and displays a list of employees along with their respective departments.

- **GUI Structure:** The GUI consists of a main window where users can view employee data in a listbox, add new employees using entry fields, and delete selected employees using buttons.

- **Database Interaction:** It utilizes SQLite to create and manage two tables: "Employee" and "Department". The Employee table stores employee details such as name and department ID, while the Department table stores department names.

Code Structure:

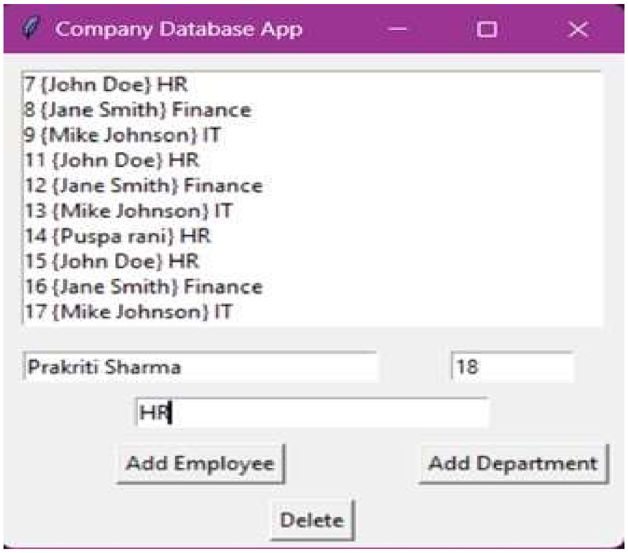
- **Database Operations Functions:** There are functions for creating tables (create_department_table() and create_employee_table()), populating tables with sample data (populate_department_table() and populate_employee_table()), loading employee data (load_employee_data()), adding new employees (add_employee_data()), and deleting employees (delete_employee_data()).

- **GUI Setup:** The Tkinter window is created (root = tk.Tk()) along with the title "Company Database App". It defines the listbox (listbox_employee) for displaying employee data and entry fields (entry_name and entry_department_id) for adding new.
- **Button Functions:** The add_button_click() and delete_button_click() functions handle the "Add" and "Delete" button clicks respectively. They retrieve input data from entry fields, perform necessary database operations, and update the listbox accordingly. Error messages are displayed if required fields are empty or if invalid data is entered.

- **Main Loop:** The root.mainloop() function initiates the Tkinter event loop, allowing the GUI to respond to user interactions.

OPERATIONS :

1. **TO ADD DATA :**

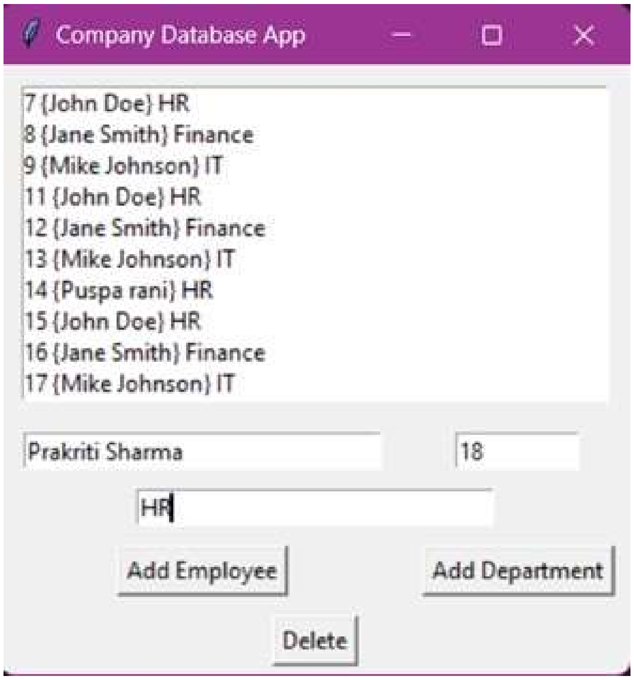


```
7 {John Doe} HR
8 {Jane Smith} Finance
9 {Mike Johnson} IT
11 {John Doe} HR
12 {Jane Smith} Finance
13 {Mike Johnson} IT
14 {Puspa rani} HR
15 {John Doe} HR
16 {Jane Smith} Finance
17 {Mike Johnson} IT
```

Prakriti Sharma

18

HR

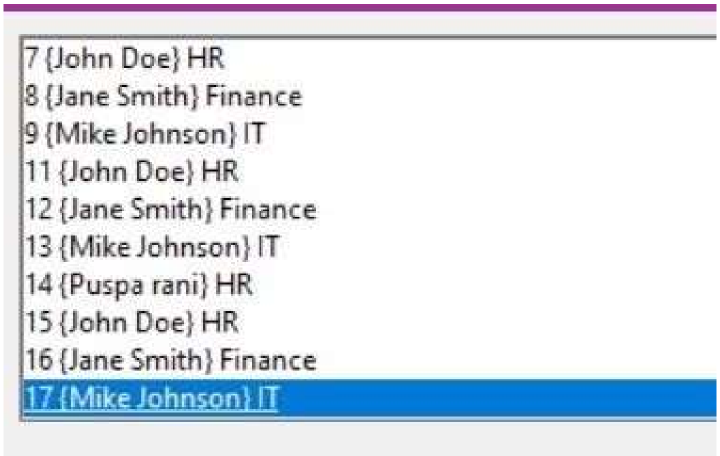


7 {John Doe} HR
8 {Jane Smith} Finance
9 {Mike Johnson} IT
11 {John Doe} HR
12 {Jane Smith} Finance
13 {Mike Johnson} IT
14 {Puspa rani} HR
15 {John Doe} HR
16 {Jane Smith} Finance
17 {Mike Johnson} IT

Prakriti Sharma
18
HR

2. TO DELETE DATA:-

-



7 {John Doe} HR
8 {Jane Smith} Finance
9 {Mike Johnson} IT
11 {John Doe} HR
12 {Jane Smith} Finance
13 {Mike Johnson} IT
14 {Puspa rani} HR
15 {John Doe} HR
16 {Jane Smith} Finance
17 {Mike Johnson} IT

3. **Create a Mini Project report for the application you have created.**

Mini Project Report: Company Database Application

Introduction

The Company Database Application is a graphical user interface (GUI) application developed using Python's Tkinter library and SQLite database. The primary objective of this application is to provide a user-friendly interface for managing employee data within a company. It allows users to add new employees, delete existing employees, and view a list of employees along with their respective departments.

Features

- Add New Employee: Users can enter the name of a new employee and select the corresponding department ID from a drop-down menu to add a new employee to the database.
- Delete Employee: Users can select an employee from the list and click the "Delete" button to remove the employee's record from the database.
- View Employee List: The application displays a list of employees, including their names and respective departments, in a listbox.
- Add New Department: Users can enter the name of a new department, which will be added to the Department table in the database.

Technologies Used

- Programming Language: Python
- GUI Library: Tkinter

Code Structure

The application is structured into several functions, each serving a specific purpose:

a. Database Connection: The `connect_to_database()` function establishes a connection to the SQLite database file (`company.db`).

b. Table Creation: The `create_department_table()` and `create_employee_table()` functions create the necessary tables (`Department` and `Employee`) if they do not already exist.

c. Data Manipulation: The `populate_department_table()` and `populate_employee_table()` functions populate the respective tables with sample data. The `load_employee_data()` function retrieves employee data from the database, while `add_employee_data()`, `add_department_data()`, and `delete_employee_data()` functions handle adding and deleting data.

d. GUI Setup: The main Tkinter window is created, along with a listbox for displaying employee data and entry fields for user input.

e. Event Handling: The `add_employee_button_click()`, `add_department_button_click()`, and `delete_button_click()` functions handle button click events and perform the corresponding database operations.

Installation and Usage

To run the Company Database Application, follow these steps:

1. Ensure that you have Python installed on your system.
2. Clone or download the project files to your local machine.
3. Open a terminal or command prompt and navigate to the project directory.
4. Run the following command to execute the application:

```
python main.py
```

5. The application window will open, and you can start interacting with the application.

Conclusion

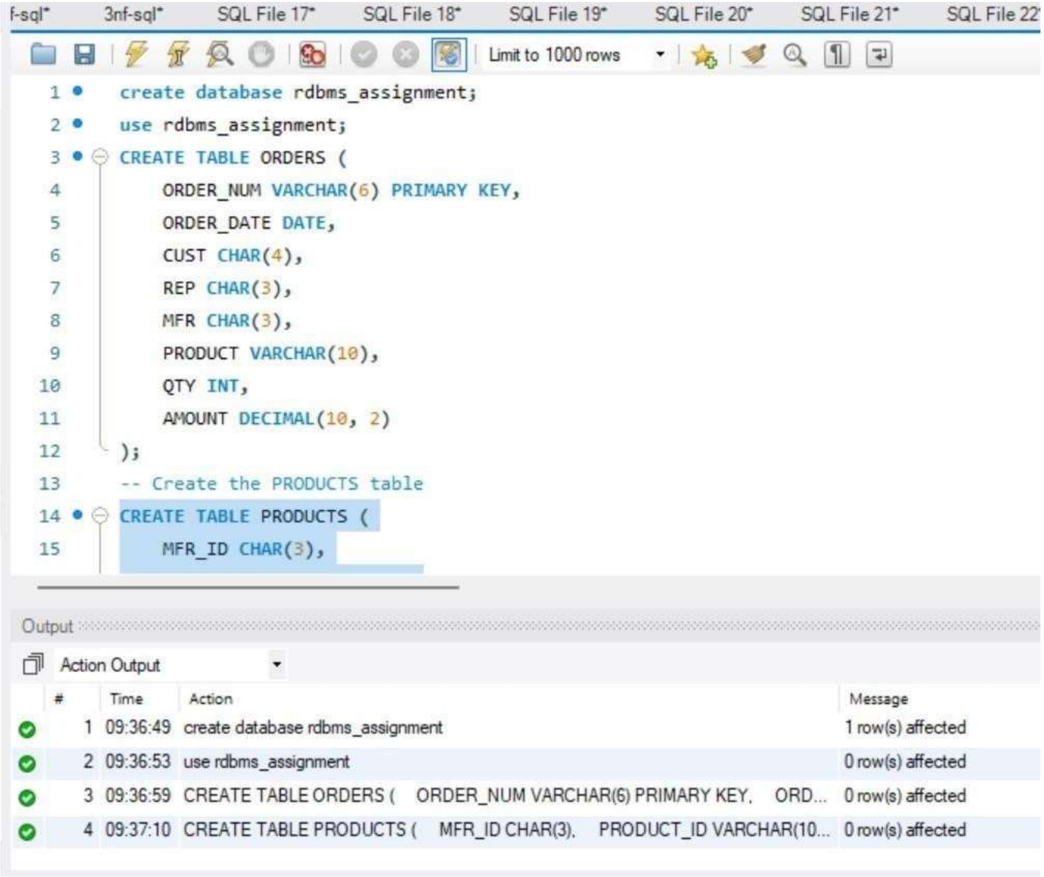
The Company Database Application provides a simple yet functional solution for managing employee data within a company. It demonstrates the integration of Python's Tkinter library with SQLite database for creating a user-friendly GUI application. Further enhancements can be made to include additional features, such as editing employee data, generating reports, or implementing user authentication.

Assignment 2 -- Practice writing Queries

The database for this assignment contains data that supports a simple order processing application for a small distribution company. It consists of five tables:

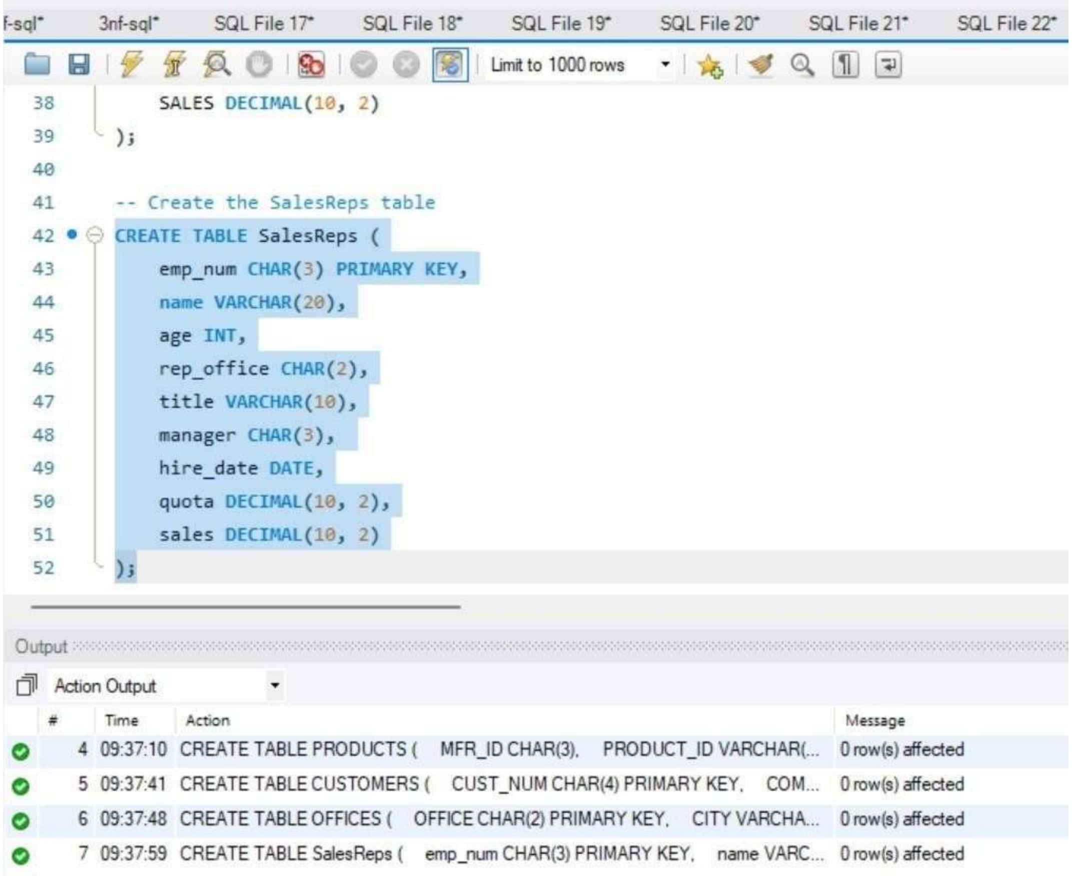
1. The CUSTOMERS table stores data about each customer, such as the company name, credit limit, and the salesperson who calls on the customer.
2. The SALESREP table stores the employee number, name, age, year-to-date sales and other data about each salesperson.
3. The OFFICES table stores data about each of the five sales offices including the city where the office is located, the sales region to which it belongs, an so on.
4. The ORDERS table keeps track of every order placed by a customer, identifying the salesperson who took the order (not necessarily the salesperson who calls on the customer), the product ordered, the quantity and amount of the order, and so on. For simplicity, each order is for only one product.
5. The PRODUCTS table stores data about each product available for sale, such as the manufacturer, product number, description, and price.

Creation of database:



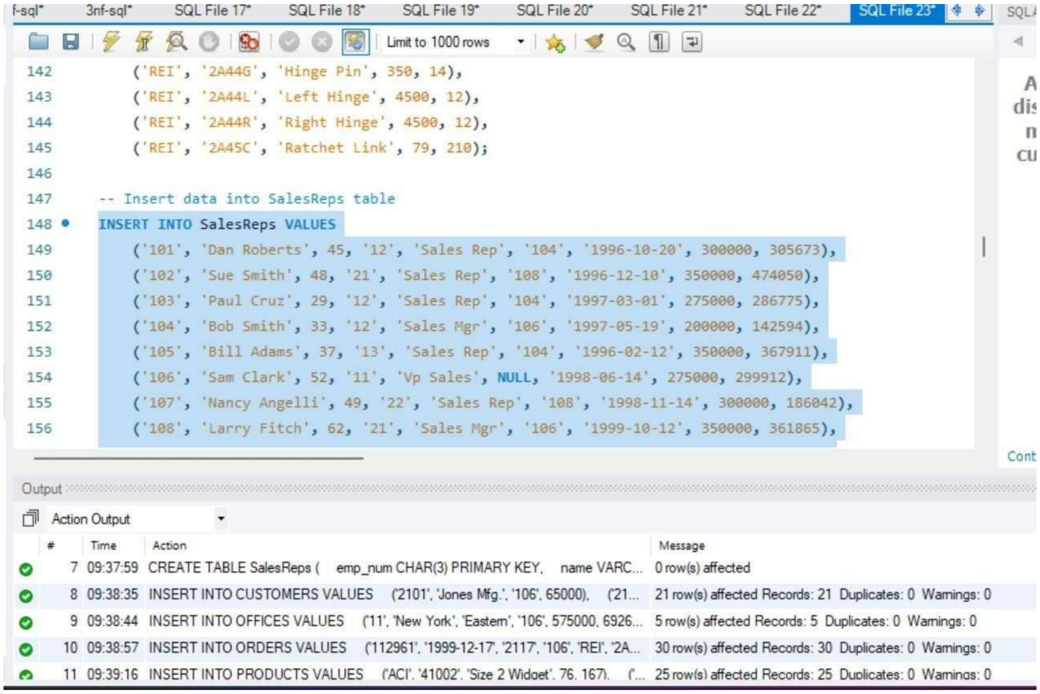
```
create database rdbms_assignment;
use rdbms_assignment;
CREATE TABLE ORDERS (
    ORDER_NUM VARCHAR(6) PRIMARY KEY,
    ORDER_DATE DATE,
    CUST CHAR(4),
    REP CHAR(3),
    MFR CHAR(3),
    PRODUCT VARCHAR(10),
    QTY INT,
    AMOUNT DECIMAL(10, 2)
);
-- Create the PRODUCTS table
CREATE TABLE PRODUCTS (
    MFR_ID CHAR(3),
```

```
#  Time      Action                                                                          Message
1  09:36:49  create database rdbms_assignment                                                1 row(s) affected
2  09:36:53  use rdbms_assignment                                                            0 row(s) affected
3  09:36:59  CREATE TABLE ORDERS (    ORDER_NUM VARCHAR(6) PRIMARY KEY,    ORD...            0 row(s) affected
4  09:37:10  CREATE TABLE PRODUCTS (    MFR_ID CHAR(3),    PRODUCT_ID VARCHAR(10...           0 row(s) affected
```



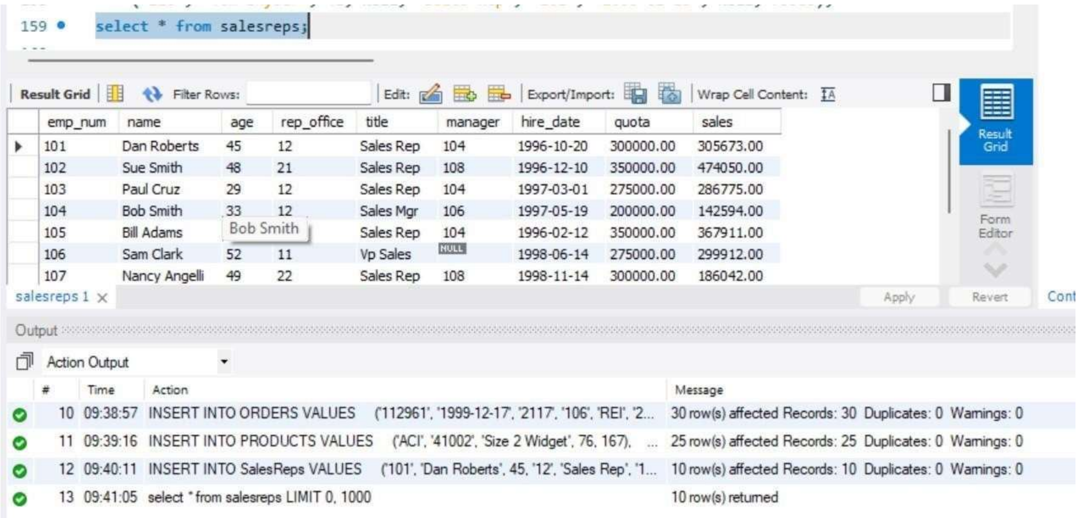
```sql
    SALES DECIMAL(10, 2)
);

-- Create the SalesReps table
CREATE TABLE SalesReps (
    emp_num CHAR(3) PRIMARY KEY,
    name VARCHAR(20),
    age INT,
    rep_office CHAR(2),
    title VARCHAR(10),
    manager CHAR(3),
    hire_date DATE,
    quota DECIMAL(10, 2),
    sales DECIMAL(10, 2)
);
```

#	Time	Action	Message
4	09:37:10	CREATE TABLE PRODUCTS (MFR_ID CHAR(3), PRODUCT_ID VARCHAR(...	0 row(s) affected
5	09:37:41	CREATE TABLE CUSTOMERS (CUST_NUM CHAR(4) PRIMARY KEY, COM...	0 row(s) affected
6	09:37:48	CREATE TABLE OFFICES (OFFICE CHAR(2) PRIMARY KEY, CITY VARCHA...	0 row(s) affected
7	09:37:59	CREATE TABLE SalesReps (emp_num CHAR(3) PRIMARY KEY, name VARC...	0 row(s) affected



```sql
    ('REI', '2A44G', 'Hinge Pin', 350, 14),
    ('REI', '2A44L', 'Left Hinge', 4500, 12),
    ('REI', '2A44R', 'Right Hinge', 4500, 12),
    ('REI', '2A45C', 'Ratchet Link', 79, 210);

-- Insert data into SalesReps table
INSERT INTO SalesReps VALUES
    ('101', 'Dan Roberts', 45, '12', 'Sales Rep', '104', '1996-10-20', 300000, 305673),
    ('102', 'Sue Smith', 48, '21', 'Sales Rep', '108', '1996-12-10', 350000, 474050),
    ('103', 'Paul Cruz', 29, '12', 'Sales Rep', '104', '1997-03-01', 275000, 286775),
    ('104', 'Bob Smith', 33, '12', 'Sales Mgr', '106', '1997-05-19', 200000, 142594),
    ('105', 'Bill Adams', 37, '13', 'Sales Rep', '104', '1996-02-12', 350000, 367911),
    ('106', 'Sam Clark', 52, '11', 'Vp Sales', NULL, '1998-06-14', 275000, 299912),
    ('107', 'Nancy Angelli', 49, '22', 'Sales Rep', '108', '1998-11-14', 300000, 186042),
    ('108', 'Larry Fitch', 62, '21', 'Sales Mgr', '106', '1999-10-12', 350000, 361865),
```

#	Time	Action	Message
7	09:37:59	CREATE TABLE SalesReps (emp_num CHAR(3) PRIMARY KEY, name VARC...	0 row(s) affected
8	09:38:35	INSERT INTO CUSTOMERS VALUES ('2101', 'Jones Mfg.', '106', 65000), ('21...	21 row(s) affected Records: 21 Duplicates: 0 Warnings: 0
9	09:38:44	INSERT INTO OFFICES VALUES ('11', 'New York', 'Eastern', '106', 575000, 6926...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0
10	09:38:57	INSERT INTO ORDERS VALUES ('112961', '1999-12-17', '2117', '106', 'REI', '2A...	30 row(s) affected Records: 30 Duplicates: 0 Warnings: 0
11	09:39:16	INSERT INTO PRODUCTS VALUES ('ACI', '41002', 'Size 2 Widget', 76, 167), ('...	25 row(s) affected Records: 25 Duplicates: 0 Warnings: 0



```
select * from salesreps;
```

emp_num	name	age	rep_office	title	manager	hire_date	quota	sales
101	Dan Roberts	45	12	Sales Rep	104	1996-10-20	300000.00	305673.00
102	Sue Smith	48	21	Sales Rep	108	1996-12-10	350000.00	474050.00
103	Paul Cruz	29	12	Sales Rep	104	1997-03-01	275000.00	286775.00
104	Bob Smith	33	12	Sales Mgr	106	1997-05-19	200000.00	142594.00
105	Bill Adams	Bob Smith		Sales Rep	104	1996-02-12	350000.00	367911.00
106	Sam Clark	52	11	Vp Sales	NULL	1998-06-14	275000.00	299912.00
107	Nancy Angelli	49	22	Sales Rep	108	1998-11-14	300000.00	186042.00

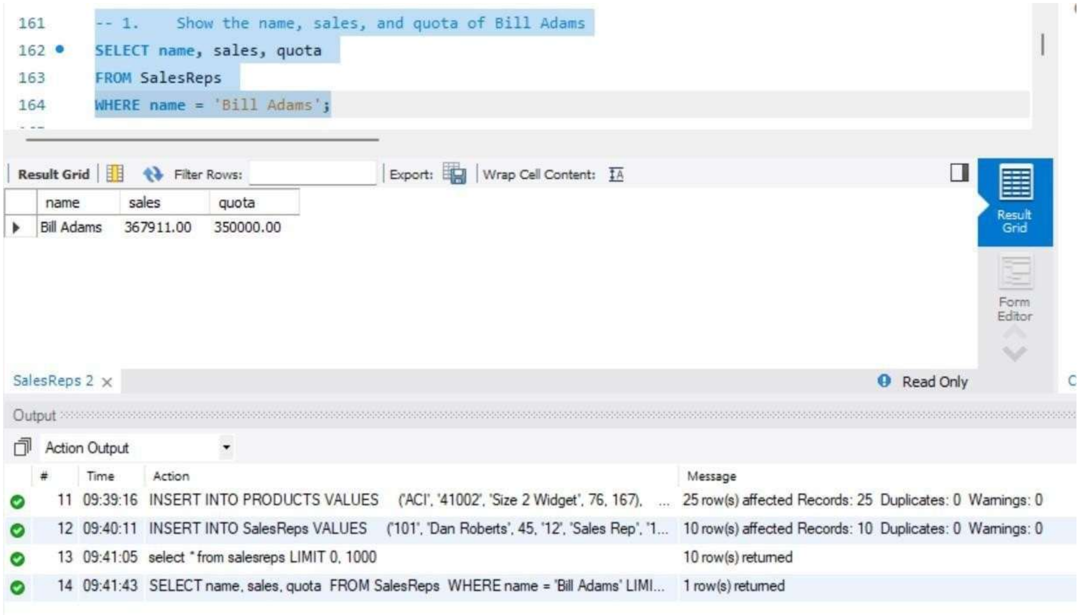
```
#   Time      Action                                                                         Message
10  09:38:57  INSERT INTO ORDERS VALUES    ('112961', '1999-12-17', '2117', '106', 'REI', '2...  30 row(s) affected Records: 30  Duplicates: 0  Wamings: 0
11  09:39:16  INSERT INTO PRODUCTS VALUES   ('ACI', '41002', 'Size 2 Widget', 76, 167),    ...  25 row(s) affected Records: 25  Duplicates: 0  Wamings: 0
12  09:40:11  INSERT INTO SalesReps VALUES   ('101', 'Dan Roberts', 45, '12', 'Sales Rep', '1...  10 row(s) affected Records: 10  Duplicates: 0  Wamings: 0
13  09:41:05  select * from salesreps LIMIT 0, 1000                                          10 row(s) returned
```

Queries:

(Unless otherwise instructed, you should assume the query is asking for names, not id numbers of customers, people, product or city offices; rename attributes if the meaning of the resultant table is not clear.)

1. **Show the name, sales, and quota of Bill Adams.**

ANS:



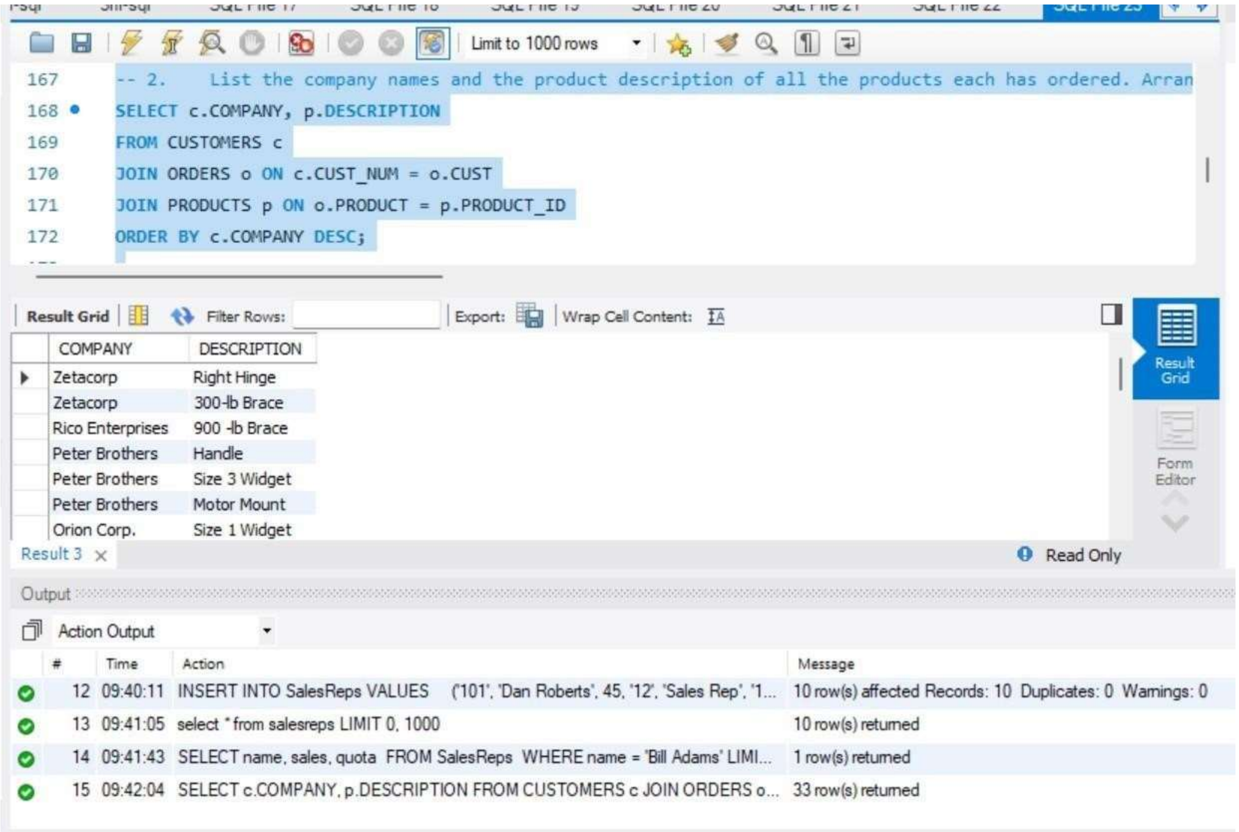
```
-- 1.    Show the name, sales, and quota of Bill Adams
SELECT name, sales, quota
FROM SalesReps
WHERE name = 'Bill Adams';
```

name	sales	quota
Bill Adams	367911.00	350000.00

```
#   Time      Action                                                                          Message
11  09:39:16  INSERT INTO PRODUCTS VALUES   ('ACI', '41002', 'Size 2 Widget', 76, 167),    ...   25 row(s) affected Records: 25  Duplicates: 0  Wamings: 0
12  09:40:11  INSERT INTO SalesReps VALUES   ('101', 'Dan Roberts', 45, '12', 'Sales Rep', '1...  10 row(s) affected Records: 10  Duplicates: 0  Wamings: 0
13  09:41:05  select * from salesreps LIMIT 0, 1000                                           10 row(s) returned
14  09:41:43  SELECT name, sales, quota  FROM SalesReps  WHERE name = 'Bill Adams' LIMI...     1 row(s) returned
```

2. **List the company names and the product description of all the products each has ordered. Arrange descending by company.**

ANS:



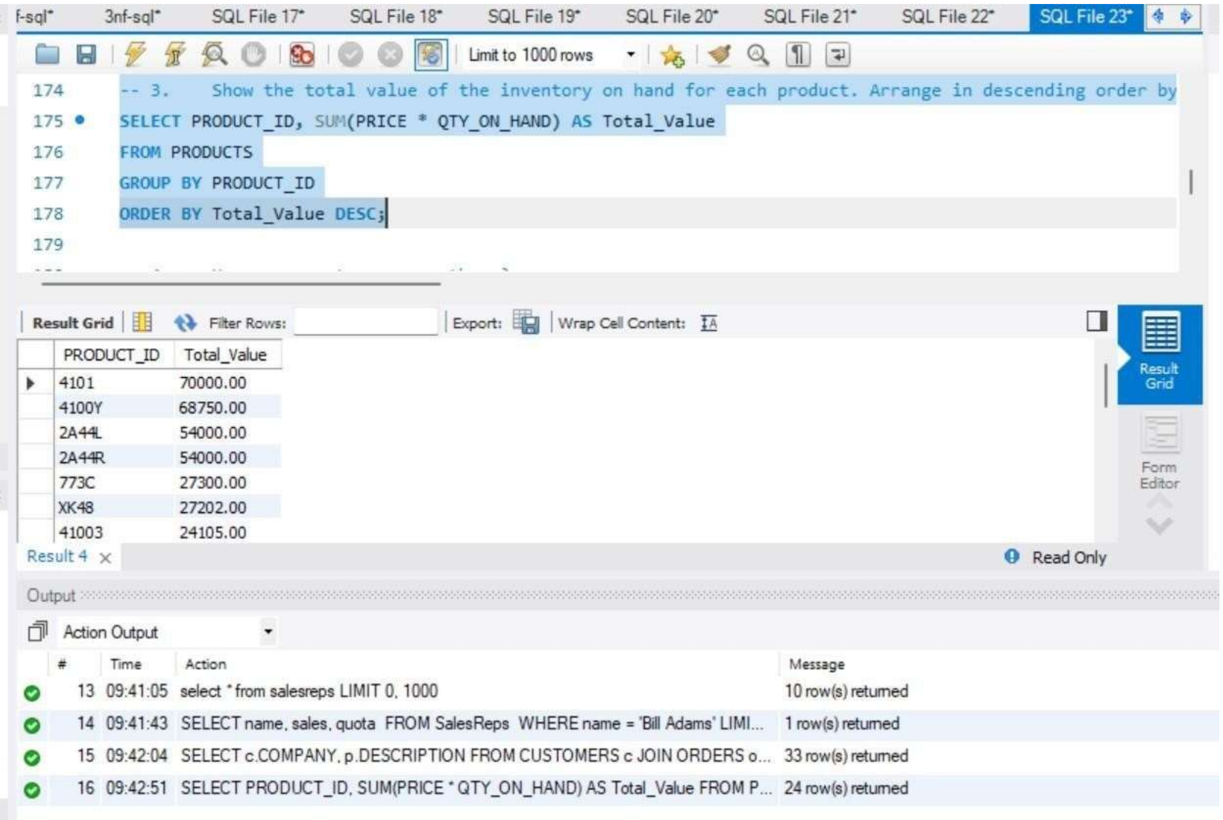
```sql
-- 2.   List the company names and the product description of all the products each has ordered. Arran
SELECT c.COMPANY, p.DESCRIPTION
FROM CUSTOMERS c
JOIN ORDERS o ON c.CUST_NUM = o.CUST
JOIN PRODUCTS p ON o.PRODUCT = p.PRODUCT_ID
ORDER BY c.COMPANY DESC;
```

COMPANY	DESCRIPTION
Zetacorp	Right Hinge
Zetacorp	300-lb Brace
Rico Enterprises	900 -lb Brace
Peter Brothers	Handle
Peter Brothers	Size 3 Widget
Peter Brothers	Motor Mount
Orion Corp.	Size 1 Widget

#	Time	Action	Message
12	09:40:11	INSERT INTO SalesReps VALUES ('101', 'Dan Roberts', 45, '12', 'Sales Rep', '1...	10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0
13	09:41:05	select * from salesreps LIMIT 0, 1000	10 row(s) returned
14	09:41:43	SELECT name, sales, quota FROM SalesReps WHERE name = 'Bill Adams' LIMI...	1 row(s) returned
15	09:42:04	SELECT c.COMPANY, p.DESCRIPTION FROM CUSTOMERS c JOIN ORDERS o...	33 row(s) returned

3. Show the total value of the inventory on hand for each product. Arrange in descending order by total value.

ANS:



```sql
-- 3.    Show the total value of the inventory on hand for each product. Arrange in descending order by
SELECT PRODUCT_ID, SUM(PRICE * QTY_ON_HAND) AS Total_Value
FROM PRODUCTS
GROUP BY PRODUCT_ID
ORDER BY Total_Value DESC;
```

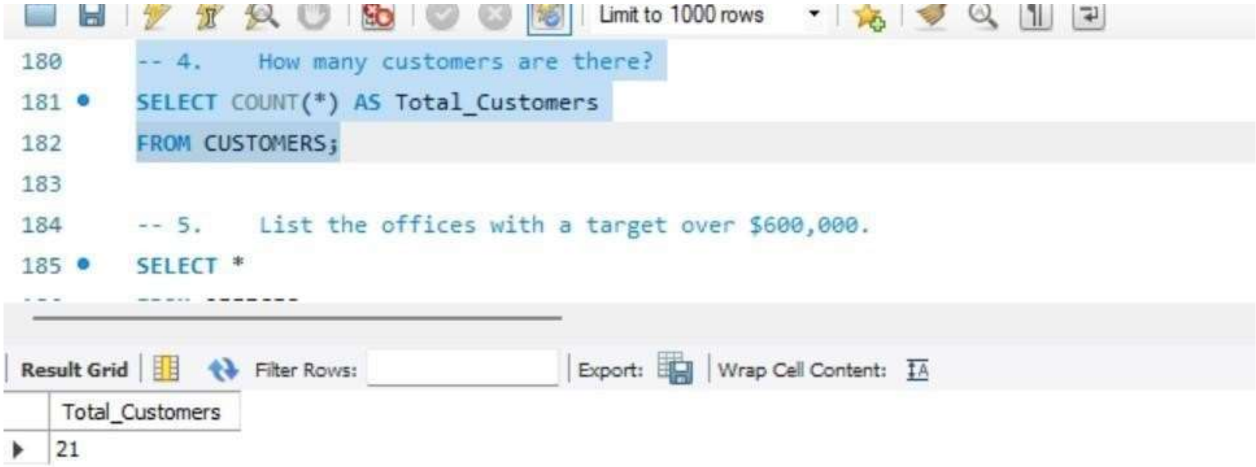
PRODUCT_ID	Total_Value
4101	70000.00
4100Y	68750.00
2A44L	54000.00
2A44R	54000.00
773C	27300.00
XK48	27202.00
41003	24105.00

Action Output

#	Time	Action	Message
13	09:41:05	select * from salesreps LIMIT 0, 1000	10 row(s) returned
14	09:41:43	SELECT name, sales, quota FROM SalesReps WHERE name = 'Bill Adams' LIMI...	1 row(s) returned
15	09:42:04	SELECT c.COMPANY, p.DESCRIPTION FROM CUSTOMERS c JOIN ORDERS o...	33 row(s) returned
16	09:42:51	SELECT PRODUCT_ID, SUM(PRICE * QTY_ON_HAND) AS Total_Value FROM P...	24 row(s) returned

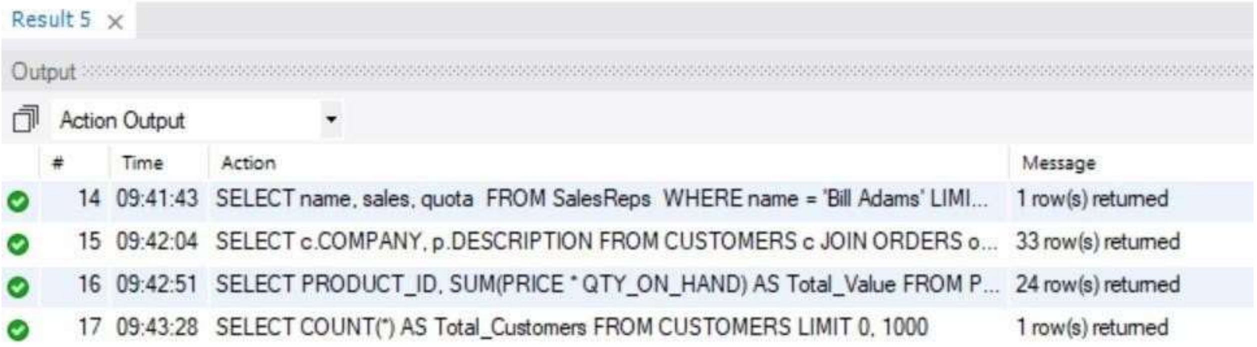
4. How many customers are there?

ANS:



```sql
-- 4.    How many customers are there?
SELECT COUNT(*) AS Total_Customers
FROM CUSTOMERS;

-- 5.    List the offices with a target over $600,000.
SELECT *
```

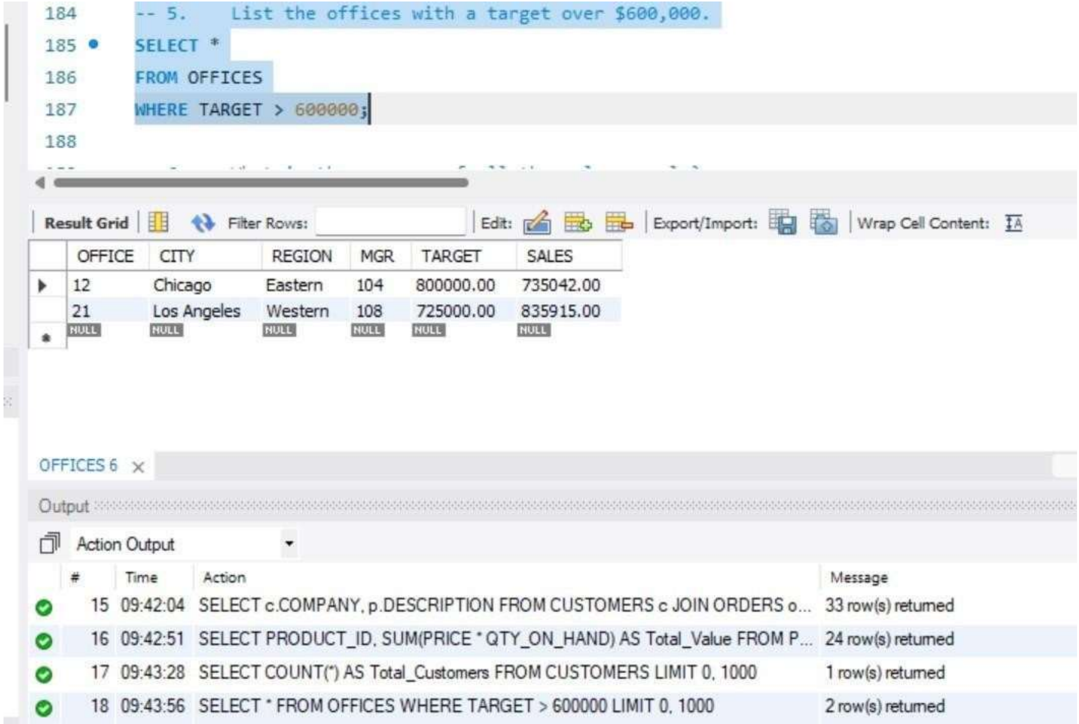
Total_Customers
21



```
#   Time      Action                                                                   Message
14  09:41:43  SELECT name, sales, quota  FROM SalesReps  WHERE name = 'Bill Adams' LIMI...  1 row(s) returned
15  09:42:04  SELECT c.COMPANY, p.DESCRIPTION FROM CUSTOMERS c JOIN ORDERS o...          33 row(s) returned
16  09:42:51  SELECT PRODUCT_ID, SUM(PRICE * QTY_ON_HAND) AS Total_Value FROM P...       24 row(s) returned
17  09:43:28  SELECT COUNT(*) AS Total_Customers FROM CUSTOMERS LIMIT 0, 1000           1 row(s) returned
```

5. List the offices with a target over $600,000.

ANS:



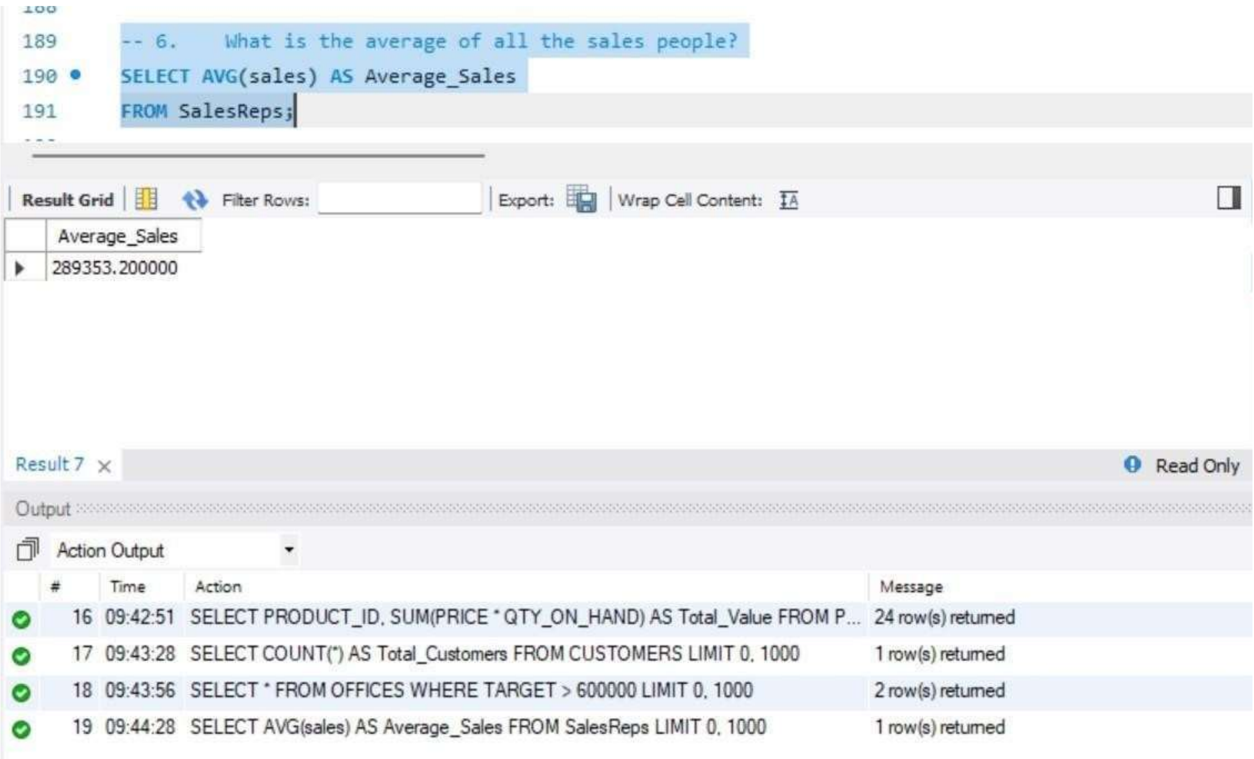
```
-- 5.   List the offices with a target over $600,000.
SELECT *
FROM OFFICES
WHERE TARGET > 600000;
```

OFFICE	CITY	REGION	MGR	TARGET	SALES
12	Chicago	Eastern	104	800000.00	735042.00
21	Los Angeles	Western	108	725000.00	835915.00
NULL	NULL	NULL	NULL	NULL	NULL

```
#   Time      Action                                                                    Message
15  09:42:04  SELECT c.COMPANY, p.DESCRIPTION FROM CUSTOMERS c JOIN ORDERS o...        33 row(s) returned
16  09:42:51  SELECT PRODUCT_ID, SUM(PRICE * QTY_ON_HAND) AS Total_Value FROM P...      24 row(s) returned
17  09:43:28  SELECT COUNT(*) AS Total_Customers FROM CUSTOMERS LIMIT 0, 1000           1 row(s) returned
18  09:43:56  SELECT * FROM OFFICES WHERE TARGET > 600000 LIMIT 0, 1000                2 row(s) returned
```

6. **What is the average of all the sales people?**

ANS:



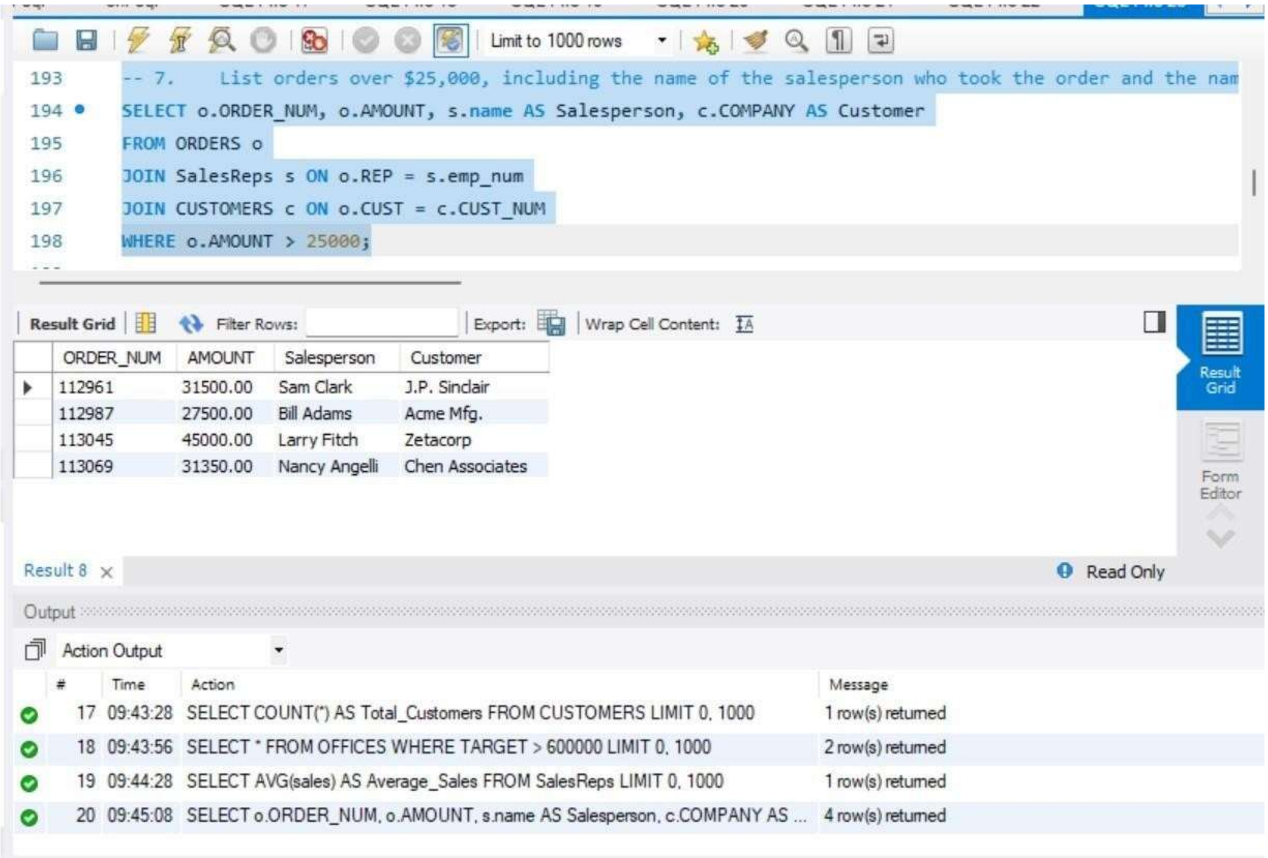
```
-- 6.   What is the average of all the sales people?
SELECT AVG(sales) AS Average_Sales
FROM SalesReps;
```

Average_Sales
289353.200000

```
#   Time      Action                                                                    Message
16  09:42:51  SELECT PRODUCT_ID, SUM(PRICE * QTY_ON_HAND) AS Total_Value FROM P...      24 row(s) returned
17  09:43:28  SELECT COUNT(*) AS Total_Customers FROM CUSTOMERS LIMIT 0, 1000           1 row(s) returned
18  09:43:56  SELECT * FROM OFFICES WHERE TARGET > 600000 LIMIT 0, 1000                2 row(s) returned
19  09:44:28  SELECT AVG(sales) AS Average_Sales FROM SalesReps LIMIT 0, 1000           1 row(s) returned
```

7. **List orders over $25,000, including the name of the salesperson who took the order and the name of the customer who placed it.**

ANS:



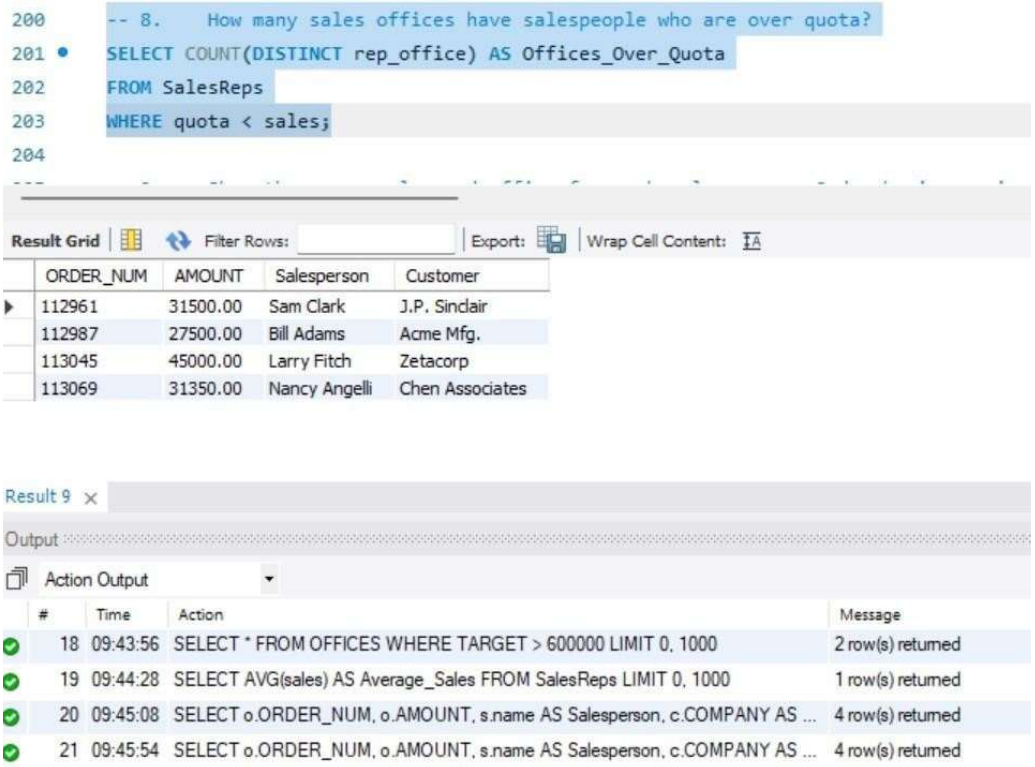
```sql
-- 7.    List orders over $25,000, including the name of the salesperson who took the order and the nam
SELECT o.ORDER_NUM, o.AMOUNT, s.name AS Salesperson, c.COMPANY AS Customer
FROM ORDERS o
JOIN SalesReps s ON o.REP = s.emp_num
JOIN CUSTOMERS c ON o.CUST = c.CUST_NUM
WHERE o.AMOUNT > 25000;
```

ORDER_NUM	AMOUNT	Salesperson	Customer
112961	31500.00	Sam Clark	J.P. Sinclair
112987	27500.00	Bill Adams	Acme Mfg.
113045	45000.00	Larry Fitch	Zetacorp
113069	31350.00	Nancy Angelli	Chen Associates

#	Time	Action	Message
17	09:43:28	SELECT COUNT(*) AS Total_Customers FROM CUSTOMERS LIMIT 0, 1000	1 row(s) returned
18	09:43:56	SELECT * FROM OFFICES WHERE TARGET > 600000 LIMIT 0, 1000	2 row(s) returned
19	09:44:28	SELECT AVG(sales) AS Average_Sales FROM SalesReps LIMIT 0, 1000	1 row(s) returned
20	09:45:08	SELECT o.ORDER_NUM, o.AMOUNT, s.name AS Salesperson, c.COMPANY AS ...	4 row(s) returned

8. **How many sales offices have salespeople who are over quota?**

ANS:



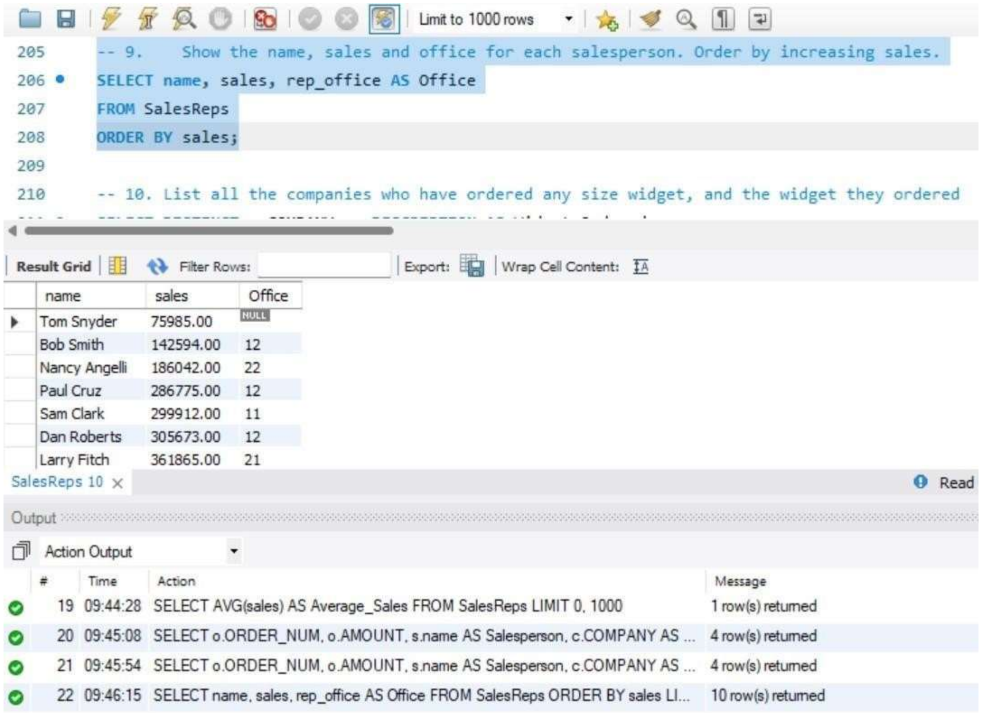
```sql
-- 8.    How many sales offices have salespeople who are over quota?
SELECT COUNT(DISTINCT rep_office) AS Offices_Over_Quota
FROM SalesReps
WHERE quota < sales;
```

ORDER_NUM	AMOUNT	Salesperson	Customer
112961	31500.00	Sam Clark	J.P. Sinclair
112987	27500.00	Bill Adams	Acme Mfg.
113045	45000.00	Larry Fitch	Zetacorp
113069	31350.00	Nancy Angelli	Chen Associates

#	Time	Action	Message
18	09:43:56	SELECT * FROM OFFICES WHERE TARGET > 600000 LIMIT 0, 1000	2 row(s) returned
19	09:44:28	SELECT AVG(sales) AS Average_Sales FROM SalesReps LIMIT 0, 1000	1 row(s) returned
20	09:45:08	SELECT o.ORDER_NUM, o.AMOUNT, s.name AS Salesperson, c.COMPANY AS ...	4 row(s) returned
21	09:45:54	SELECT o.ORDER_NUM, o.AMOUNT, s.name AS Salesperson, c.COMPANY AS ...	4 row(s) returned

9. Show the name, sales and office for each salesperson. Order by increasing sales.

ANS:



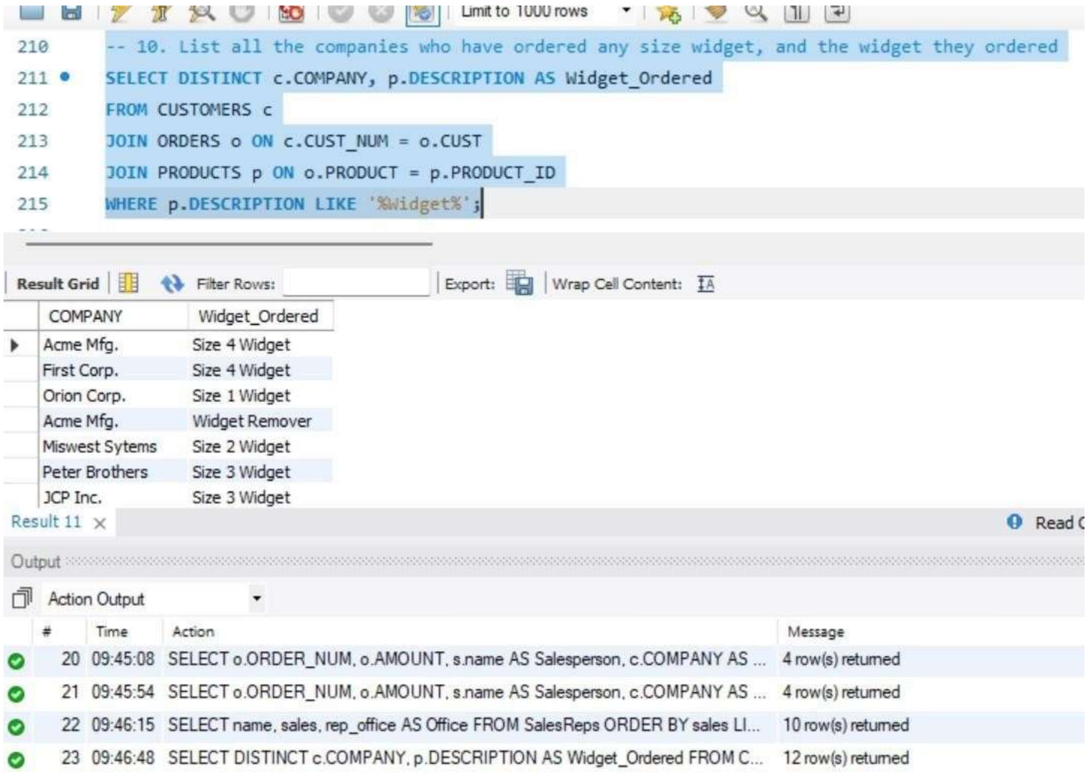
```
-- 9.   Show the name, sales and office for each salesperson. Order by increasing sales.
SELECT name, sales, rep_office AS Office
FROM SalesReps
ORDER BY sales;

-- 10. List all the companies who have ordered any size widget, and the widget they ordered
```

name	sales	Office
Tom Snyder	75985.00	NULL
Bob Smith	142594.00	12
Nancy Angelli	186042.00	22
Paul Cruz	286775.00	12
Sam Clark	299912.00	11
Dan Roberts	305673.00	12
Larry Fitch	361865.00	21

#	Time	Action	Message
19	09:44:28	SELECT AVG(sales) AS Average_Sales FROM SalesReps LIMIT 0, 1000	1 row(s) returned
20	09:45:08	SELECT o.ORDER_NUM, o.AMOUNT, s.name AS Salesperson, c.COMPANY AS ...	4 row(s) returned
21	09:45:54	SELECT o.ORDER_NUM, o.AMOUNT, s.name AS Salesperson, c.COMPANY AS ...	4 row(s) returned
22	09:46:15	SELECT name, sales, rep_office AS Office FROM SalesReps ORDER BY sales LI...	10 row(s) returned

10. List all the companies who have ordered any size widget, and the widget they ordered.

ANS:



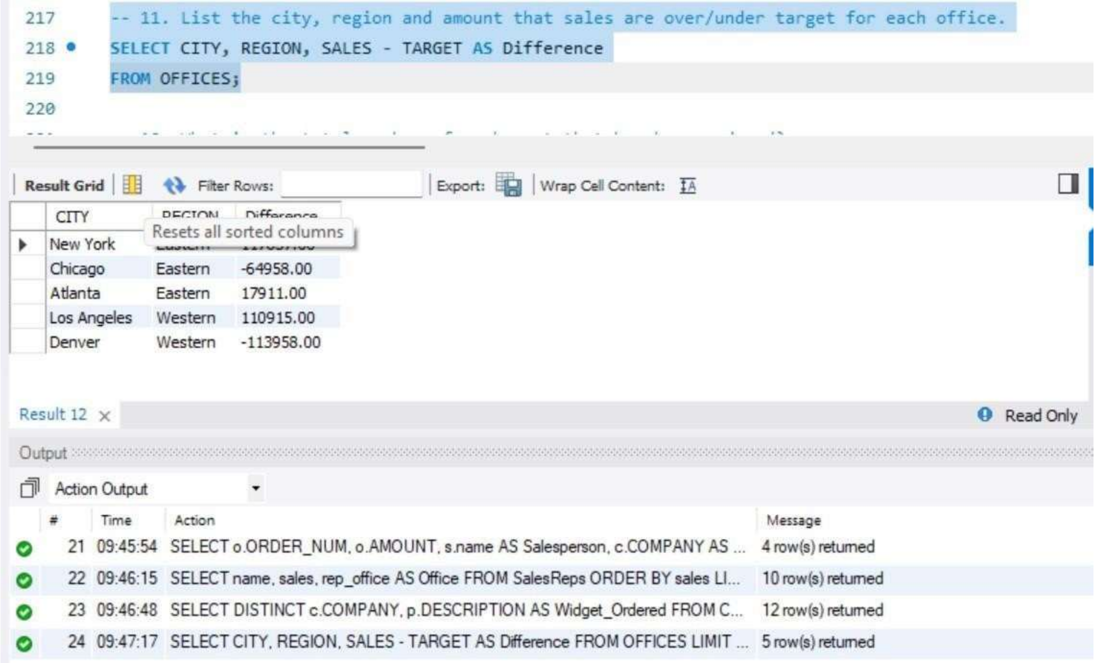
```
-- 10. List all the companies who have ordered any size widget, and the widget they ordered
SELECT DISTINCT c.COMPANY, p.DESCRIPTION AS Widget_Ordered
FROM CUSTOMERS c
JOIN ORDERS o ON c.CUST_NUM = o.CUST
JOIN PRODUCTS p ON o.PRODUCT = p.PRODUCT_ID
WHERE p.DESCRIPTION LIKE '%Widget%';
```

COMPANY	Widget_Ordered
Acme Mfg.	Size 4 Widget
First Corp.	Size 4 Widget
Orion Corp.	Size 1 Widget
Acme Mfg.	Widget Remover
Miswest Sytems	Size 2 Widget
Peter Brothers	Size 3 Widget
JCP Inc.	Size 3 Widget

```
Output
Action Output
    #   Time      Action                                                                Message
    20  09:45:08  SELECT o.ORDER_NUM, o.AMOUNT, s.name AS Salesperson, c.COMPANY AS ...   4 row(s) returned
    21  09:45:54  SELECT o.ORDER_NUM, o.AMOUNT, s.name AS Salesperson, c.COMPANY AS ...   4 row(s) returned
    22  09:46:15  SELECT name, sales, rep_office AS Office FROM SalesReps ORDER BY sales LI...  10 row(s) returned
    23  09:46:48  SELECT DISTINCT c.COMPANY, p.DESCRIPTION AS Widget_Ordered FROM C...  12 row(s) returned
```

11. List the city, region and amount that sales are over/under target for each office.

ANS:



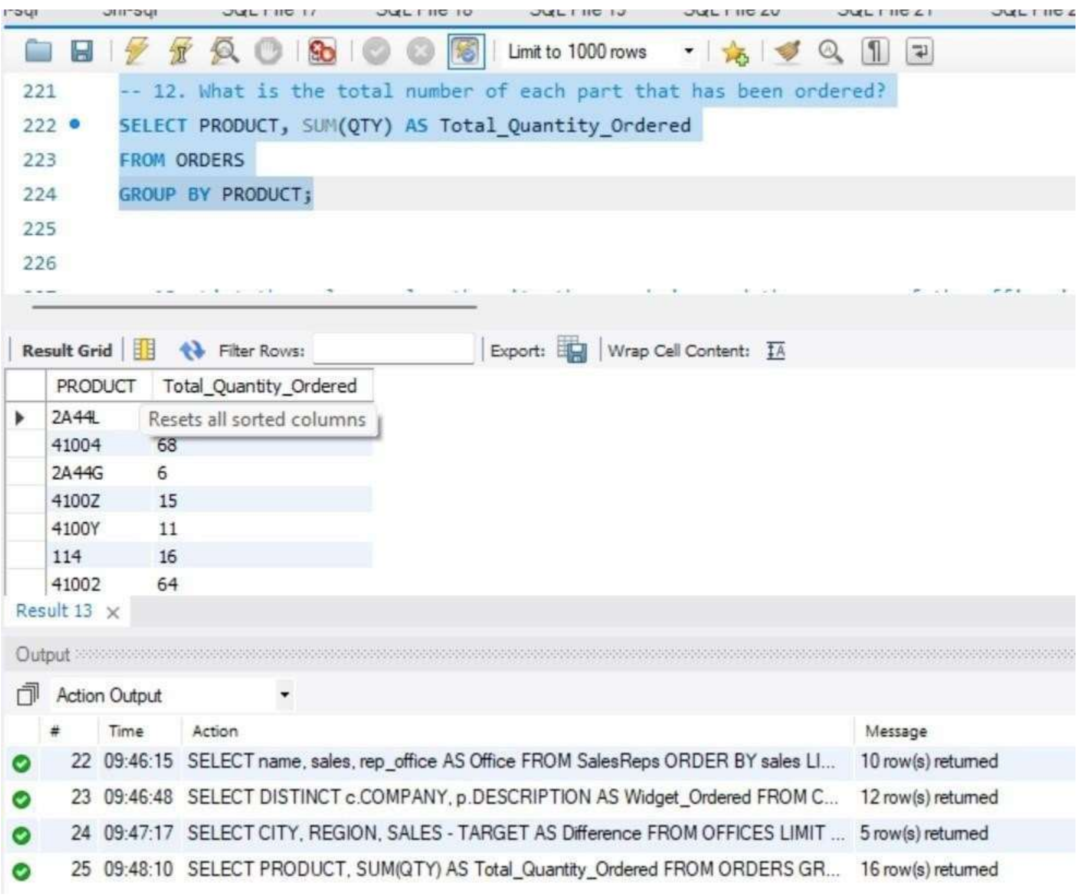
```sql
-- 11. List the city, region and amount that sales are over/under target for each office.
SELECT CITY, REGION, SALES - TARGET AS Difference
FROM OFFICES;
```

CITY	REGION	Difference
New York	[illegible]	[illegible]
Chicago	Eastern	-64958.00
Atlanta	Eastern	17911.00
Los Angeles	Western	110915.00
Denver	Western	-113958.00

#	Time	Action	Message
21	09:45:54	SELECT o.ORDER_NUM, o.AMOUNT, s.name AS Salesperson, c.COMPANY AS ...	4 row(s) returned
22	09:46:15	SELECT name, sales, rep_office AS Office FROM SalesReps ORDER BY sales LI...	10 row(s) returned
23	09:46:48	SELECT DISTINCT c.COMPANY, p.DESCRIPTION AS Widget_Ordered FROM C...	12 row(s) returned
24	09:47:17	SELECT CITY, REGION, SALES - TARGET AS Difference FROM OFFICES LIMIT ...	5 row(s) returned

12. **What is the total number of each part that has been ordered?**

ANS:



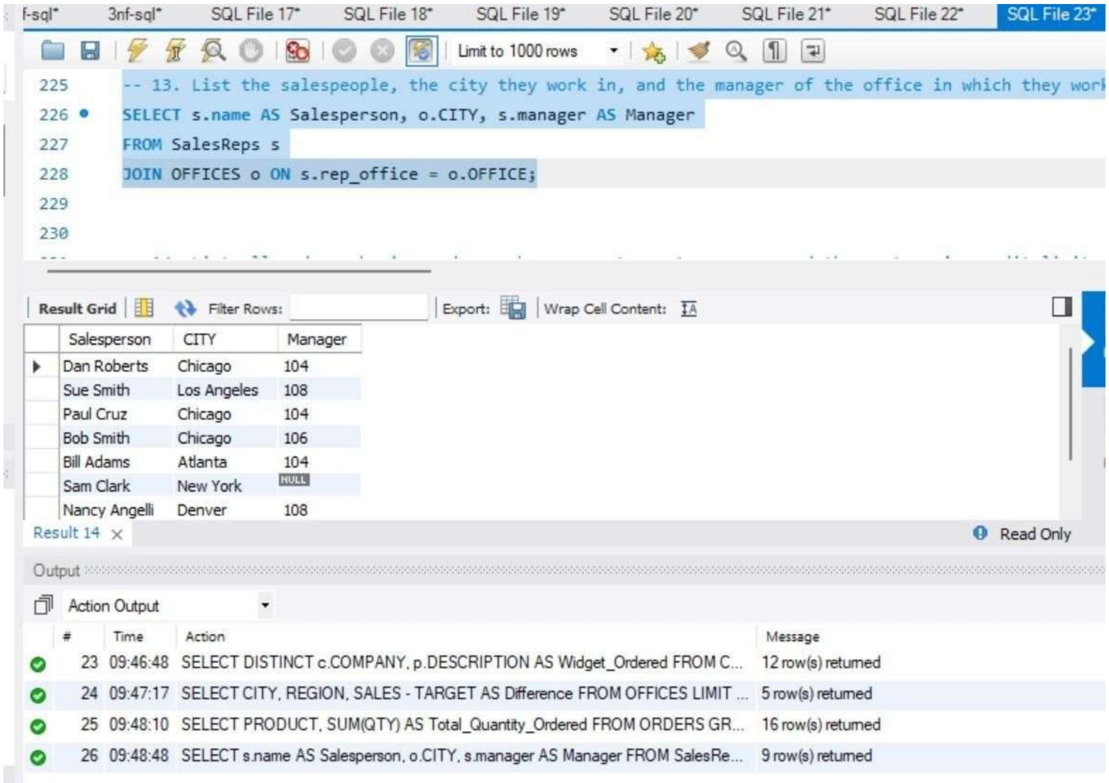
```sql
-- 12. What is the total number of each part that has been ordered?
SELECT PRODUCT, SUM(QTY) AS Total_Quantity_Ordered
FROM ORDERS
GROUP BY PRODUCT;
```

PRODUCT	Total_Quantity_Ordered
2A44L	[illegible]
41004	68
2A44G	6
4100Z	15
4100Y	11
114	16
41002	64

#	Time	Action	Message
22	09:46:15	SELECT name, sales, rep_office AS Office FROM SalesReps ORDER BY sales LI...	10 row(s) returned
23	09:46:48	SELECT DISTINCT c.COMPANY, p.DESCRIPTION AS Widget_Ordered FROM C...	12 row(s) returned
24	09:47:17	SELECT CITY, REGION, SALES - TARGET AS Difference FROM OFFICES LIMIT ...	5 row(s) returned
25	09:48:10	SELECT PRODUCT, SUM(QTY) AS Total_Quantity_Ordered FROM ORDERS GR...	16 row(s) returned

13. **List the salespeople, the city they work in, and the manager of the office in which they work.**

ANS:



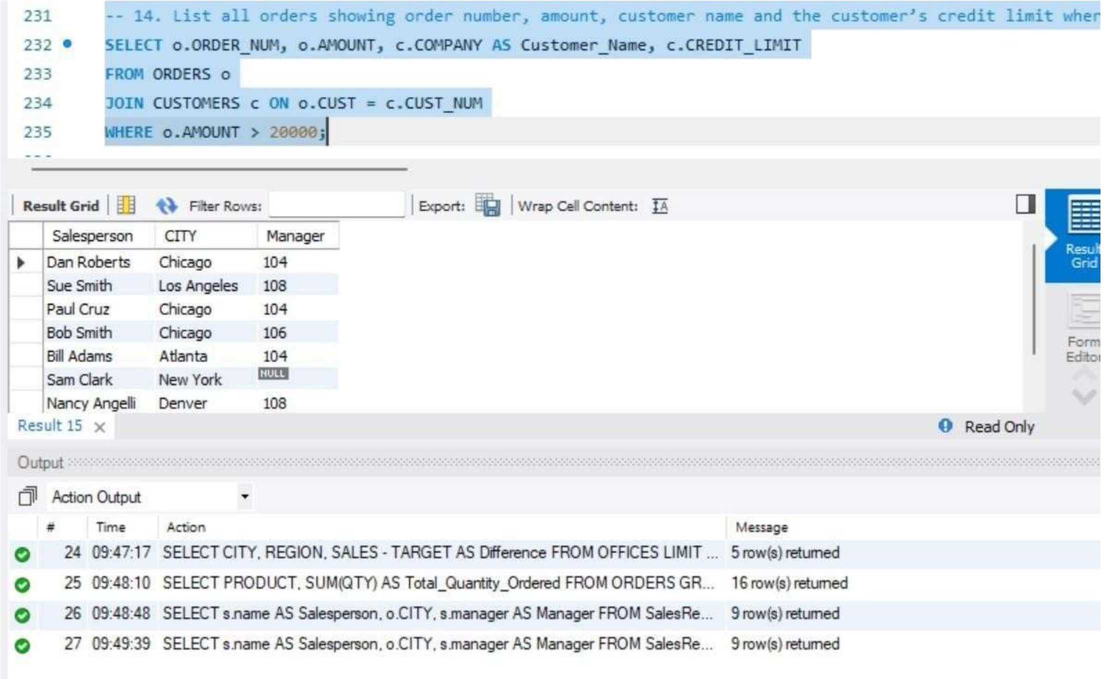
```sql
-- 13. List the salespeople, the city they work in, and the manager of the office in which they work
SELECT s.name AS Salesperson, o.CITY, s.manager AS Manager
FROM SalesReps s
JOIN OFFICES o ON s.rep_office = o.OFFICE;
```

Salesperson	CITY	Manager
Dan Roberts	Chicago	104
Sue Smith	Los Angeles	108
Paul Cruz	Chicago	104
Bob Smith	Chicago	106
Bill Adams	Atlanta	104
Sam Clark	New York	NULL
Nancy Angelli	Denver	108

```
#   Time      Action                                                                 Message
23  09:46:48  SELECT DISTINCT c.COMPANY, p.DESCRIPTION AS Widget_Ordered FROM C...     12 row(s) returned
24  09:47:17  SELECT CITY, REGION, SALES - TARGET AS Difference FROM OFFICES LIMIT ... 5 row(s) returned
25  09:48:10  SELECT PRODUCT, SUM(QTY) AS Total_Quantity_Ordered FROM ORDERS GR...    16 row(s) returned
26  09:48:48  SELECT s.name AS Salesperson, o.CITY, s.manager AS Manager FROM SalesRe... 9 row(s) returned
```

14. **List all orders showing order number, amount, customer name and the customer's credit limit where the order was greater than $20,000.**

ANS:



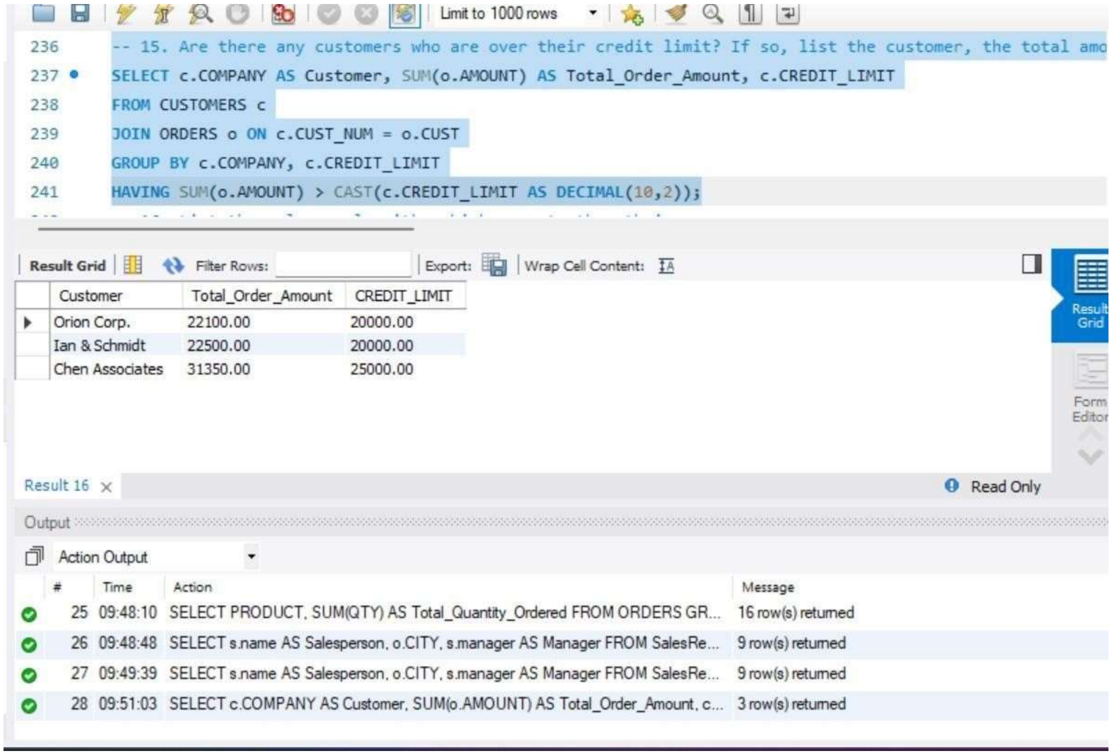
```sql
-- 14. List all orders showing order number, amount, customer name and the customer's credit limit wher
SELECT o.ORDER_NUM, o.AMOUNT, c.COMPANY AS Customer_Name, c.CREDIT_LIMIT
FROM ORDERS o
JOIN CUSTOMERS c ON o.CUST = c.CUST_NUM
WHERE o.AMOUNT > 20000;
```

Salesperson	CITY	Manager
Dan Roberts	Chicago	104
Sue Smith	Los Angeles	108
Paul Cruz	Chicago	104
Bob Smith	Chicago	106
Bill Adams	Atlanta	104
Sam Clark	New York	NULL
Nancy Angelli	Denver	108

```
#   Time      Action                                                                 Message
24  09:47:17  SELECT CITY, REGION, SALES - TARGET AS Difference FROM OFFICES LIMIT ... 5 row(s) returned
25  09:48:10  SELECT PRODUCT, SUM(QTY) AS Total_Quantity_Ordered FROM ORDERS GR...    16 row(s) returned
26  09:48:48  SELECT s.name AS Salesperson, o.CITY, s.manager AS Manager FROM SalesRe... 9 row(s) returned
27  09:49:39  SELECT s.name AS Salesperson, o.CITY, s.manager AS Manager FROM SalesRe... 9 row(s) returned
```

15. **Are there any customers who are over their credit limit? If so, list the customer, the total amount the customer has on order, and the credit limit.**

ANS:



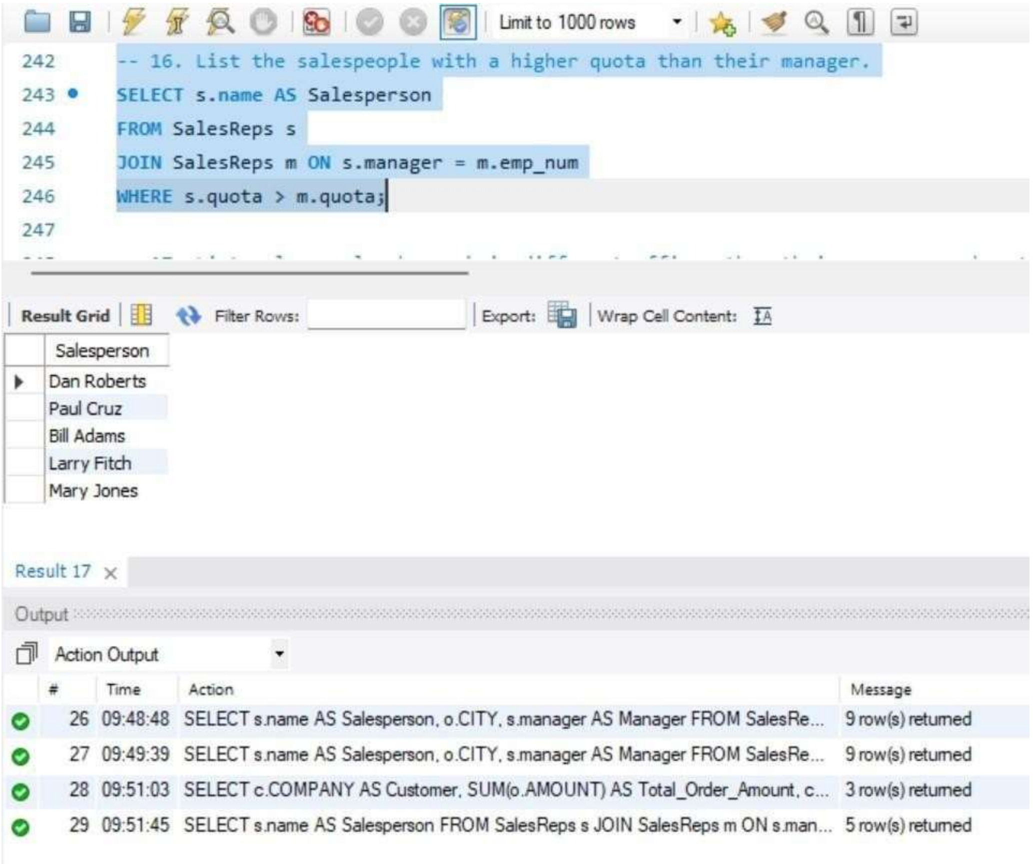
```sql
-- 15. Are there any customers who are over their credit limit? If so, list the customer, the total amo
SELECT c.COMPANY AS Customer, SUM(o.AMOUNT) AS Total_Order_Amount, c.CREDIT_LIMIT
FROM CUSTOMERS c
JOIN ORDERS o ON c.CUST_NUM = o.CUST
GROUP BY c.COMPANY, c.CREDIT_LIMIT
HAVING SUM(o.AMOUNT) > CAST(c.CREDIT_LIMIT AS DECIMAL(10,2));
```

Customer	Total_Order_Amount	CREDIT_LIMIT
Orion Corp.	22100.00	20000.00
Ian & Schmidt	22500.00	20000.00
Chen Associates	31350.00	25000.00

#	Time	Action	Message
25	09:48:10	SELECT PRODUCT, SUM(QTY) AS Total_Quantity_Ordered FROM ORDERS GR...	16 row(s) returned
26	09:48:48	SELECT s.name AS Salesperson, o.CITY, s.manager AS Manager FROM SalesRe...	9 row(s) returned
27	09:49:39	SELECT s.name AS Salesperson, o.CITY, s.manager AS Manager FROM SalesRe...	9 row(s) returned
28	09:51:03	SELECT c.COMPANY AS Customer, SUM(o.AMOUNT) AS Total_Order_Amount, c...	3 row(s) returned

16. List the salespeople with a higher quota than their manager.

ANS:



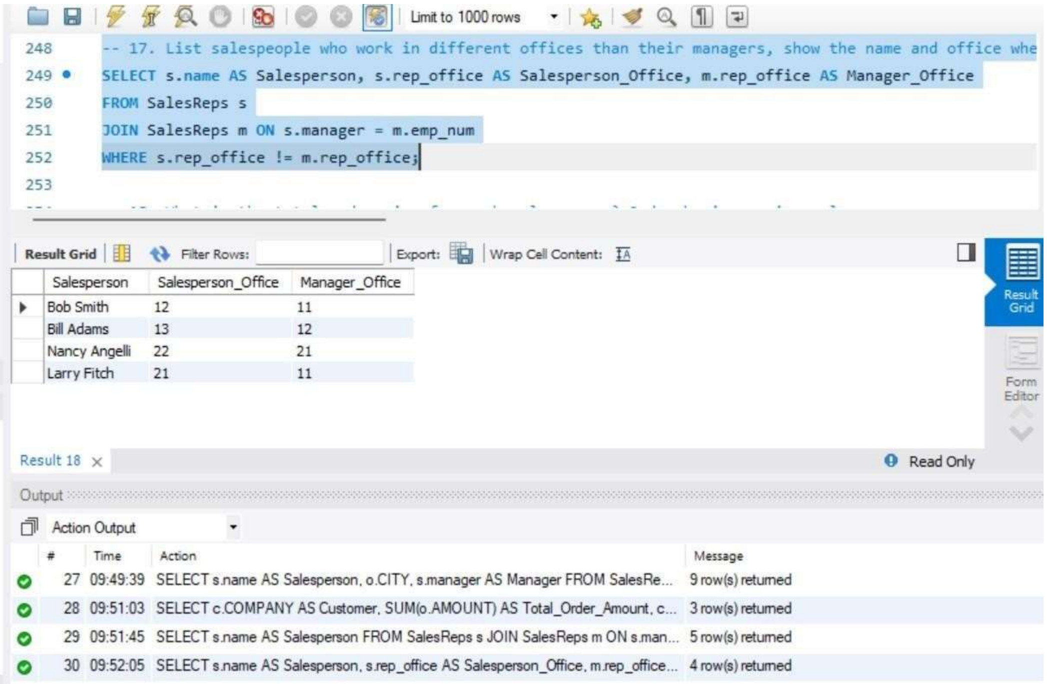
```sql
-- 16. List the salespeople with a higher quota than their manager.
SELECT s.name AS Salesperson
FROM SalesReps s
JOIN SalesReps m ON s.manager = m.emp_num
WHERE s.quota > m.quota;
```

Salesperson
Dan Roberts
Paul Cruz
Bill Adams
Larry Fitch
Mary Jones

#	Time	Action	Message
26	09:48:48	SELECT s.name AS Salesperson, o.CITY, s.manager AS Manager FROM SalesRe...	9 row(s) returned
27	09:49:39	SELECT s.name AS Salesperson, o.CITY, s.manager AS Manager FROM SalesRe...	9 row(s) returned
28	09:51:03	SELECT c.COMPANY AS Customer, SUM(o.AMOUNT) AS Total_Order_Amount, c...	3 row(s) returned
29	09:51:45	SELECT s.name AS Salesperson FROM SalesReps s JOIN SalesReps m ON s.man...	5 row(s) returned

17. List salespeople who work in different offices than their managers, show the name and office where each work.

ANS:



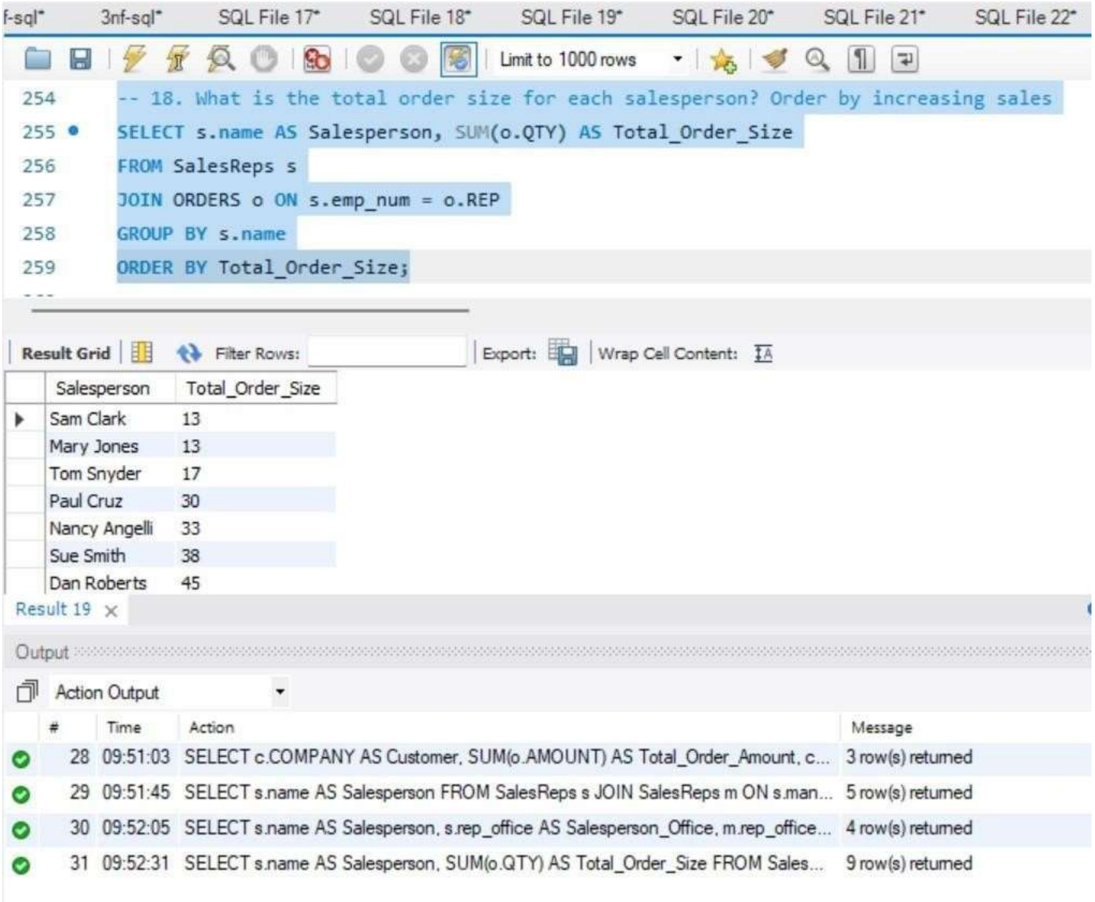
```
-- 17. List salespeople who work in different offices than their managers, show the name and office whe
SELECT s.name AS Salesperson, s.rep_office AS Salesperson_Office, m.rep_office AS Manager_Office
FROM SalesReps s
JOIN SalesReps m ON s.manager = m.emp_num
WHERE s.rep_office != m.rep_office;
```

Salesperson	Salesperson_Office	Manager_Office
Bob Smith	12	11
Bill Adams	13	12
Nancy Angelli	22	21
Larry Fitch	21	11

#	Time	Action	Message
27	09:49:39	SELECT s.name AS Salesperson, o.CITY, s.manager AS Manager FROM SalesRe...	9 row(s) returned
28	09:51:03	SELECT c.COMPANY AS Customer, SUM(o.AMOUNT) AS Total_Order_Amount, c...	3 row(s) returned
29	09:51:45	SELECT s.name AS Salesperson FROM SalesReps s JOIN SalesReps m ON s.man...	5 row(s) returned
30	09:52:05	SELECT s.name AS Salesperson, s.rep_office AS Salesperson_Office, m.rep_office...	4 row(s) returned

18. **What is the total order size for each salesperson? Order by increasing sales.**

ANS:



Limit to 1000 rows

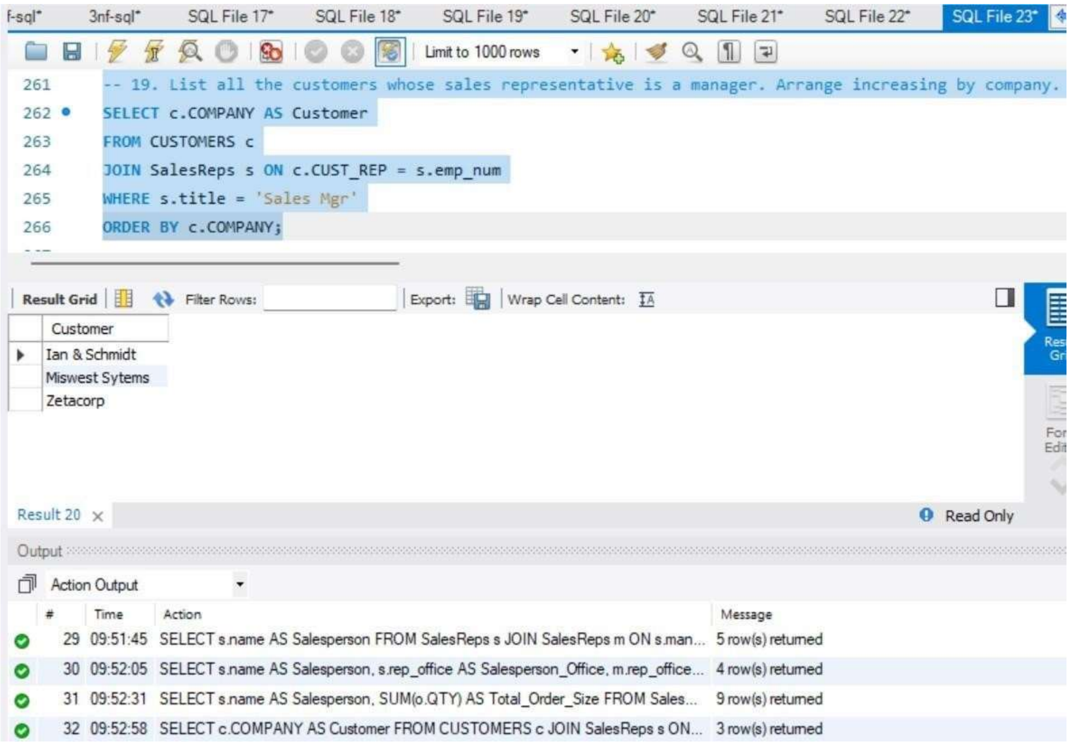
```sql
-- 18. What is the total order size for each salesperson? Order by increasing sales
SELECT s.name AS Salesperson, SUM(o.QTY) AS Total_Order_Size
FROM SalesReps s
JOIN ORDERS o ON s.emp_num = o.REP
GROUP BY s.name
ORDER BY Total_Order_Size;
```

Salesperson	Total_Order_Size
Sam Clark	13
Mary Jones	13
Tom Snyder	17
Paul Cruz	30
Nancy Angelli	33
Sue Smith	38
Dan Roberts	45

```
#   Time      Action                                                                          Message
28  09:51:03  SELECT c.COMPANY AS Customer, SUM(o.AMOUNT) AS Total_Order_Amount, c...        3 row(s) returned
29  09:51:45  SELECT s.name AS Salesperson FROM SalesReps s JOIN SalesReps m ON s.man...     5 row(s) returned
30  09:52:05  SELECT s.name AS Salesperson, s.rep_office AS Salesperson_Office, m.rep_office... 4 row(s) returned
31  09:52:31  SELECT s.name AS Salesperson, SUM(o.QTY) AS Total_Order_Size FROM Sales...      9 row(s) returned
```

19. **List all the customers whose sales representative is a manager. Arrange increasing by company.**

ANS:



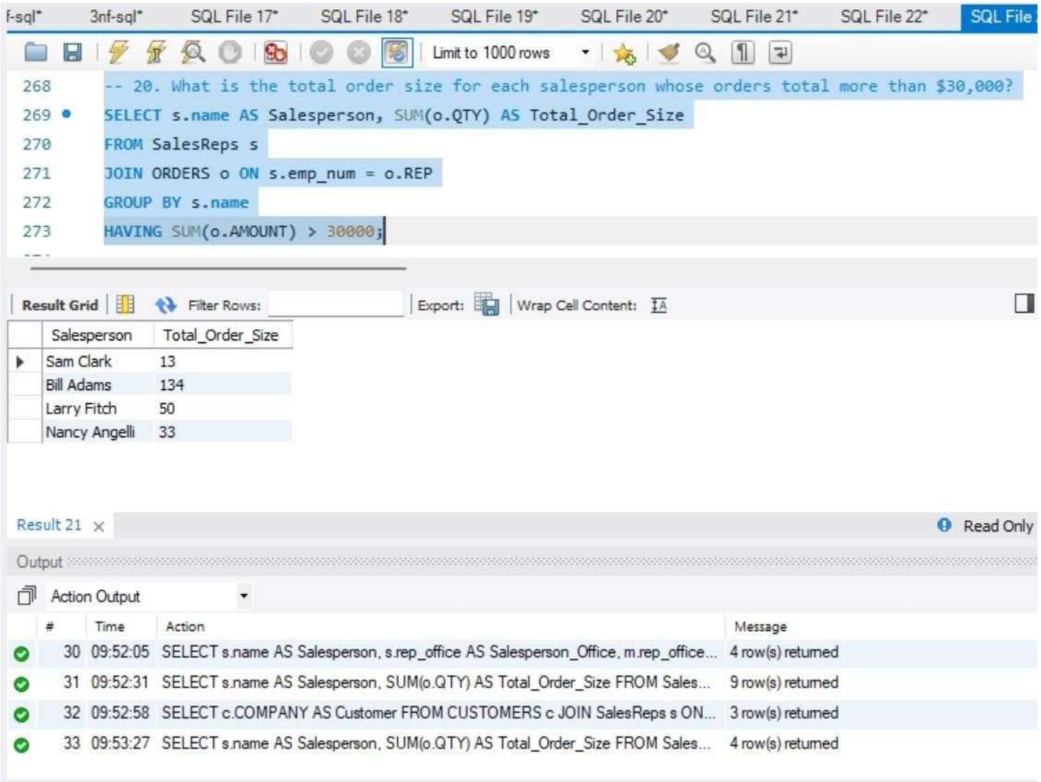
```sql
-- 19. List all the customers whose sales representative is a manager. Arrange increasing by company.
SELECT c.COMPANY AS Customer
FROM CUSTOMERS c
JOIN SalesReps s ON c.CUST_REP = s.emp_num
WHERE s.title = 'Sales Mgr'
ORDER BY c.COMPANY;
```

```
Customer
Ian & Schmidt
Miswest Sytems
Zetacorp
```

```
#   Time      Action                                                                      Message
29  09:51:45  SELECT s.name AS Salesperson FROM SalesReps s JOIN SalesReps m ON s.man...   5 row(s) returned
30  09:52:05  SELECT s.name AS Salesperson, s.rep_office AS Salesperson_Office, m.rep_office...  4 row(s) returned
31  09:52:31  SELECT s.name AS Salesperson, SUM(o.QTY) AS Total_Order_Size FROM Sales...   9 row(s) returned
32  09:52:58  SELECT c.COMPANY AS Customer FROM CUSTOMERS c JOIN SalesReps s ON...         3 row(s) returned
```

20. **What is the total order size for each salesperson whose orders total more than $30,000?**

ANS:



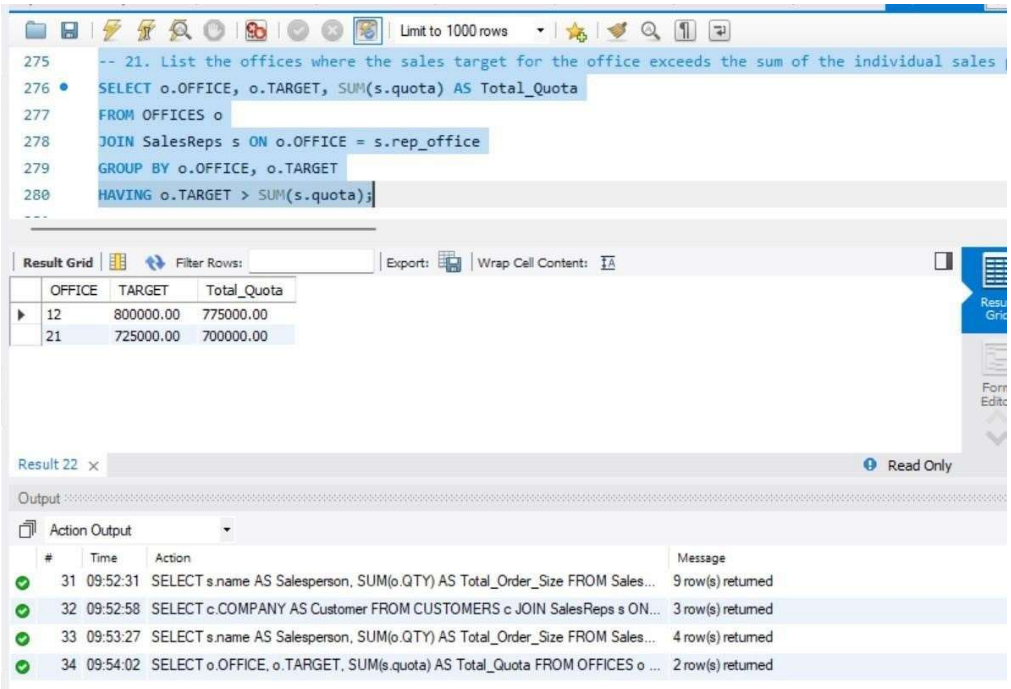
```sql
-- 20. What is the total order size for each salesperson whose orders total more than $30,000?
SELECT s.name AS Salesperson, SUM(o.QTY) AS Total_Order_Size
FROM SalesReps s
JOIN ORDERS o ON s.emp_num = o.REP
GROUP BY s.name
HAVING SUM(o.AMOUNT) > 30000;
```

```
Salesperson     Total_Order_Size
Sam Clark       13
Bill Adams      134
Larry Fitch     50
Nancy Angelli   33
```

```
#   Time      Action                                                                      Message
30  09:52:05  SELECT s.name AS Salesperson, s.rep_office AS Salesperson_Office, m.rep_office...  4 row(s) returned
31  09:52:31  SELECT s.name AS Salesperson, SUM(o.QTY) AS Total_Order_Size FROM Sales...   9 row(s) returned
32  09:52:58  SELECT c.COMPANY AS Customer FROM CUSTOMERS c JOIN SalesReps s ON...         3 row(s) returned
33  09:53:27  SELECT s.name AS Salesperson, SUM(o.QTY) AS Total_Order_Size FROM Sales...   4 row(s) returned
```

21. **List the offices where the sales target for the office exceeds the sum of the individual sales people's quotas.**

ANS:



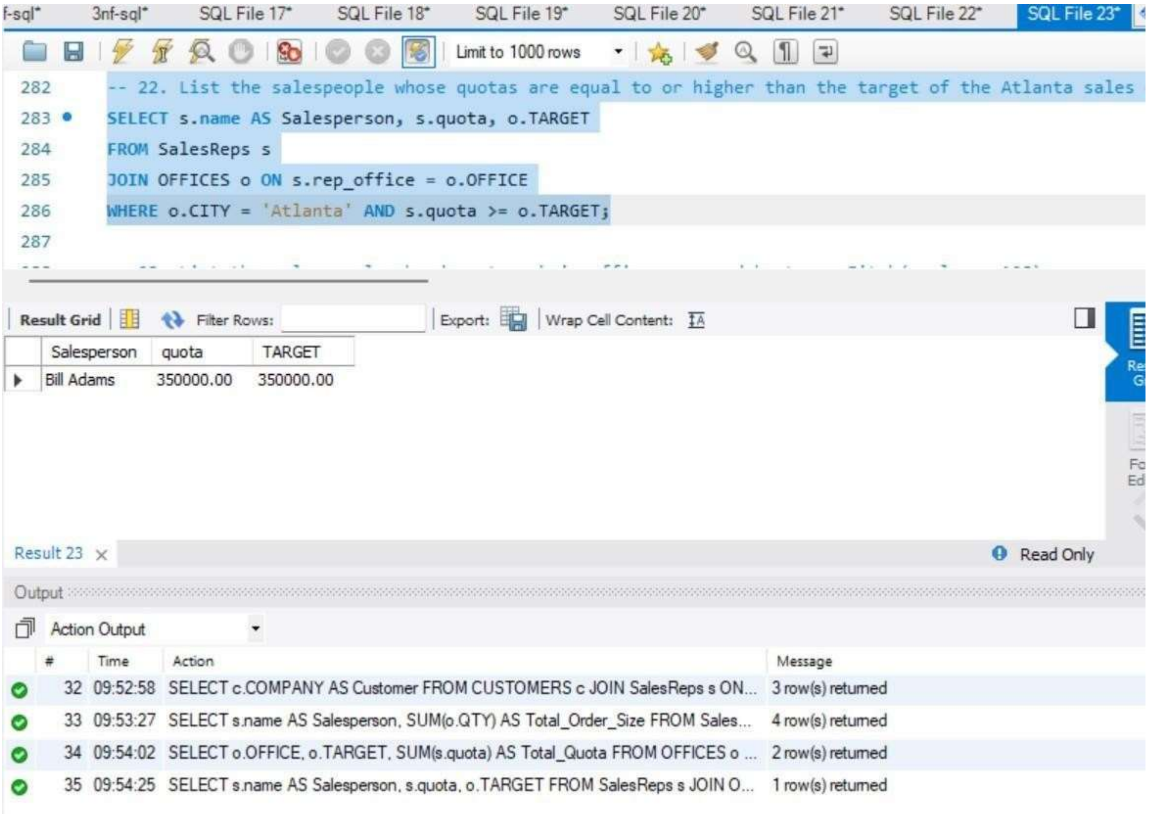
```sql
-- 21. List the offices where the sales target for the office exceeds the sum of the individual sales
SELECT o.OFFICE, o.TARGET, SUM(s.quota) AS Total_Quota
FROM OFFICES o
JOIN SalesReps s ON o.OFFICE = s.rep_office
GROUP BY o.OFFICE, o.TARGET
HAVING o.TARGET > SUM(s.quota);
```

OFFICE	TARGET	Total_Quota
12	800000.00	775000.00
21	725000.00	700000.00

#	Time	Action	Message
31	09:52:31	SELECT s.name AS Salesperson, SUM(o.QTY) AS Total_Order_Size FROM Sales...	9 row(s) returned
32	09:52:58	SELECT c.COMPANY AS Customer FROM CUSTOMERS c JOIN SalesReps s ON...	3 row(s) returned
33	09:53:27	SELECT s.name AS Salesperson, SUM(o.QTY) AS Total_Order_Size FROM Sales...	4 row(s) returned
34	09:54:02	SELECT o.OFFICE, o.TARGET, SUM(s.quota) AS Total_Quota FROM OFFICES o ...	2 row(s) returned

22. List the salespeople whose quotas are equal to or higher than the target of the Atlanta sales office.

ANS:



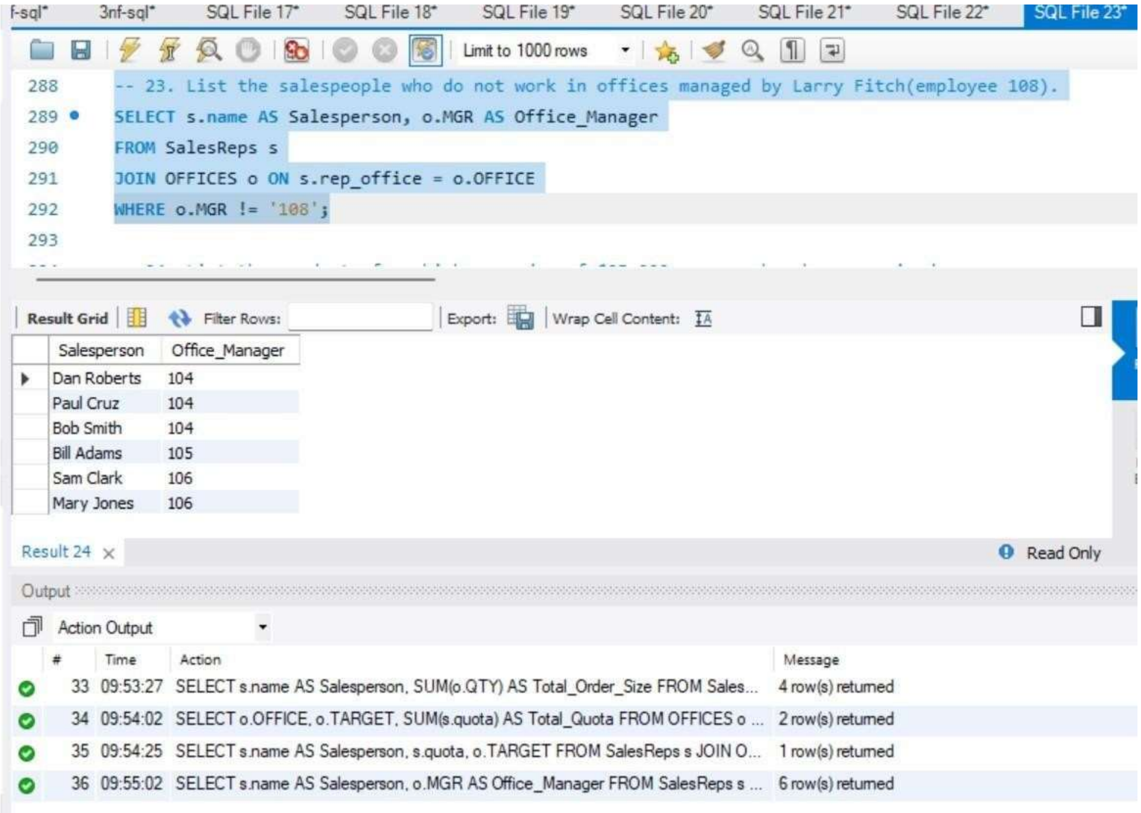
```sql
-- 22. List the salespeople whose quotas are equal to or higher than the target of the Atlanta sales
SELECT s.name AS Salesperson, s.quota, o.TARGET
FROM SalesReps s
JOIN OFFICES o ON s.rep_office = o.OFFICE
WHERE o.CITY = 'Atlanta' AND s.quota >= o.TARGET;
```

Salesperson	quota	TARGET
Bill Adams	350000.00	350000.00

#	Time	Action	Message
32	09:52:58	SELECT c.COMPANY AS Customer FROM CUSTOMERS c JOIN SalesReps s ON...	3 row(s) returned
33	09:53:27	SELECT s.name AS Salesperson, SUM(o.QTY) AS Total_Order_Size FROM Sales...	4 row(s) returned
34	09:54:02	SELECT o.OFFICE, o.TARGET, SUM(s.quota) AS Total_Quota FROM OFFICES o ...	2 row(s) returned
35	09:54:25	SELECT s.name AS Salesperson, s.quota, o.TARGET FROM SalesReps s JOIN O...	1 row(s) returned

23. List the salespeople who do not work in offices managed by Larry Fitch(employee 108).

ANS:


f-sql* 3nf-sql* SQL File 17* SQL File 18* SQL File 19* SQL File 20* SQL File 21* SQL File 22* SQL File 23*
Limit to 1000 rows
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
Result 24
Read Only
Output
Action Output

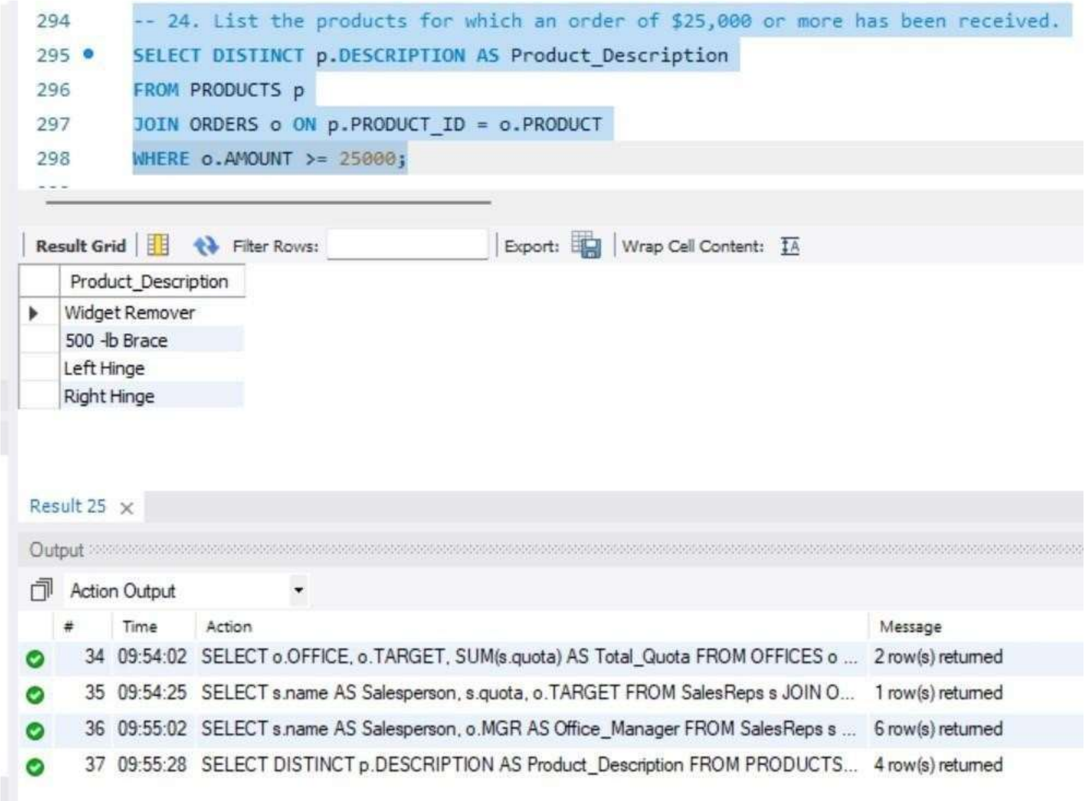
```sql
-- 23. List the salespeople who do not work in offices managed by Larry Fitch(employee 108).
SELECT s.name AS Salesperson, o.MGR AS Office_Manager
FROM SalesReps s
JOIN OFFICES o ON s.rep_office = o.OFFICE
WHERE o.MGR != '108';
```

Salesperson	Office_Manager
Dan Roberts	104
Paul Cruz	104
Bob Smith	104
Bill Adams	105
Sam Clark	106
Mary Jones	106

```
#   Time      Action                                                                   Message
33  09:53:27  SELECT s.name AS Salesperson, SUM(o.QTY) AS Total_Order_Size FROM Sales...  4 row(s) returned
34  09:54:02  SELECT o.OFFICE, o.TARGET, SUM(s.quota) AS Total_Quota FROM OFFICES o ...   2 row(s) returned
35  09:54:25  SELECT s.name AS Salesperson, s.quota, o.TARGET FROM SalesReps s JOIN O...   1 row(s) returned
36  09:55:02  SELECT s.name AS Salesperson, o.MGR AS Office_Manager FROM SalesReps s ...   6 row(s) returned
```

24. **List the products for which an order of $25,000 or more has been received.**

ANS:



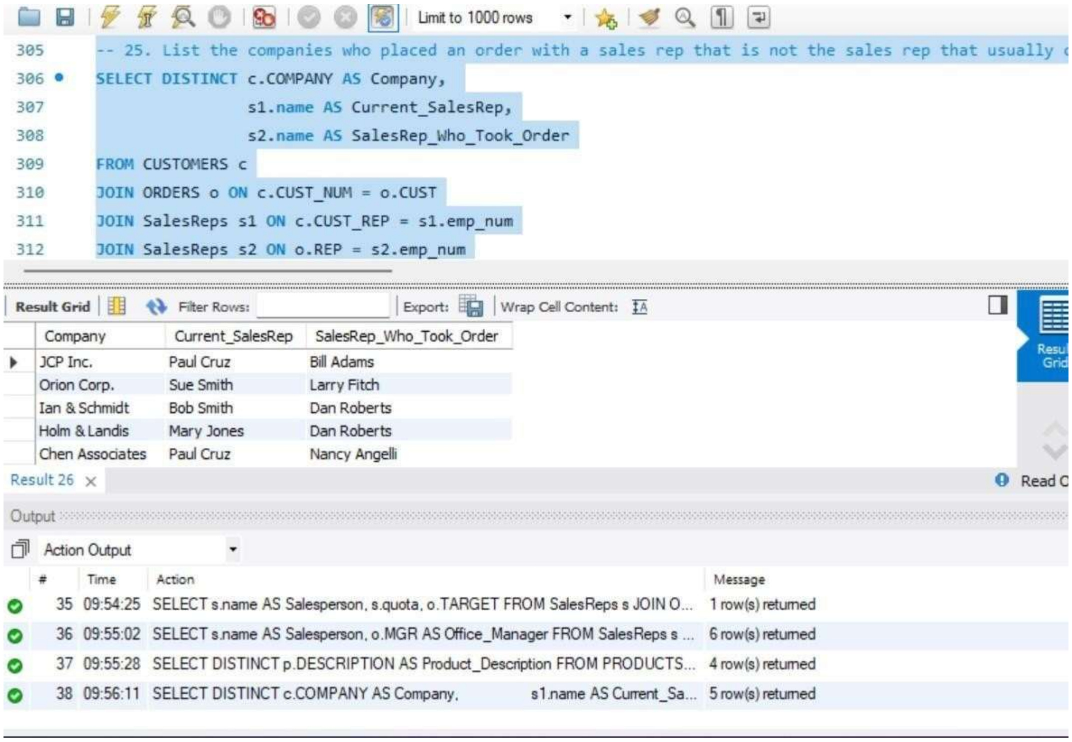
```sql
-- 24. List the products for which an order of $25,000 or more has been received.
SELECT DISTINCT p.DESCRIPTION AS Product_Description
FROM PRODUCTS p
JOIN ORDERS o ON p.PRODUCT_ID = o.PRODUCT
WHERE o.AMOUNT >= 25000;
```

Product_Description
Widget Remover
500 -lb Brace
Left Hinge
Right Hinge

```
#   Time      Action                                                                   Message
34  09:54:02  SELECT o.OFFICE, o.TARGET, SUM(s.quota) AS Total_Quota FROM OFFICES o ...   2 row(s) returned
35  09:54:25  SELECT s.name AS Salesperson, s.quota, o.TARGET FROM SalesReps s JOIN O...   1 row(s) returned
36  09:55:02  SELECT s.name AS Salesperson, o.MGR AS Office_Manager FROM SalesReps s ...   6 row(s) returned
37  09:55:28  SELECT DISTINCT p.DESCRIPTION AS Product_Description FROM PRODUCTS...        4 row(s) returned
```

25. **List the companies who placed an order with a sales rep that is not the sales rep that usually calls on them. Include the names of the salesreps, indicating by attribute name who took the order.**

ANS:



```sql
-- 25. List the companies who placed an order with a sales rep that is not the sales rep that usually c
SELECT DISTINCT c.COMPANY AS Company,
                s1.name AS Current_SalesRep,
                s2.name AS SalesRep_Who_Took_Order
FROM CUSTOMERS c
JOIN ORDERS o ON c.CUST_NUM = o.CUST
JOIN SalesReps s1 ON c.CUST_REP = s1.emp_num
JOIN SalesReps s2 ON o.REP = s2.emp_num
```

Company	Current_SalesRep	SalesRep_Who_Took_Order
JCP Inc.	Paul Cruz	Bill Adams
Orion Corp.	Sue Smith	Larry Fitch
Ian & Schmidt	Bob Smith	Dan Roberts
Holm & Landis	Mary Jones	Dan Roberts
Chen Associates	Paul Cruz	Nancy Angelli

#	Time	Action	Message
35	09:54:25	SELECT s.name AS Salesperson, s.quota, o.TARGET FROM SalesReps s JOIN O...	1 row(s) returned
36	09:55:02	SELECT s.name AS Salesperson, o.MGR AS Office_Manager FROM SalesReps s ...	6 row(s) returned
37	09:55:28	SELECT DISTINCT p.DESCRIPTION AS Product_Description FROM PRODUCTS...	4 row(s) returned
38	09:56:11	SELECT DISTINCT c.COMPANY AS Company, s1.name AS Current_Sa...	5 row(s) returned

Assignment 3 Database Design Assignment

Part I -- the ER Diagram

1. Draw the ER diagram.

   - Remember your first pass at designing this way is to include all the attributes of each entity, then find the relationships by attributes that refer to other entities.

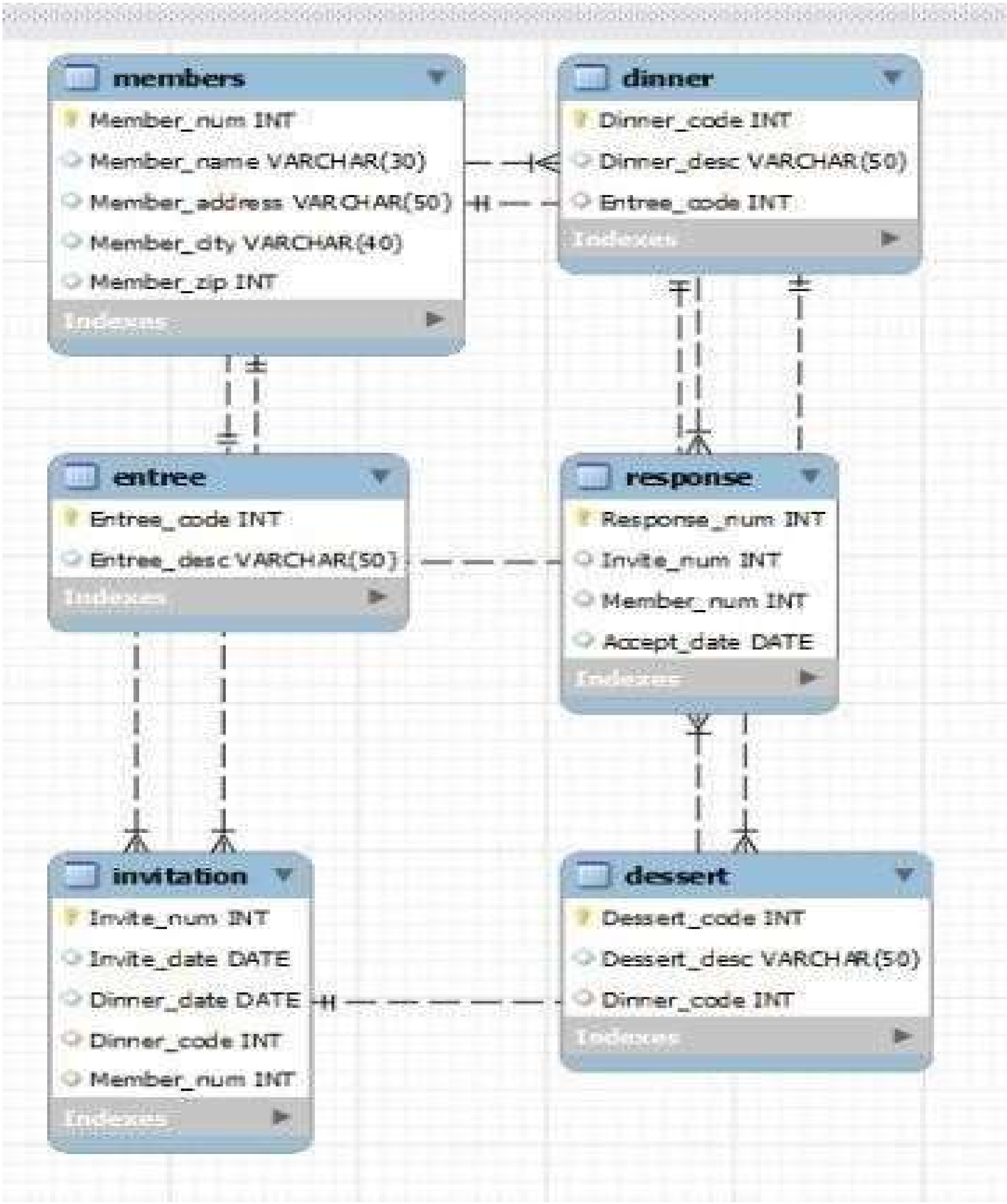
   - Be sure to include whether the relationships are partial or total, and the cardinality ratio.

2. Map your ER diagram to a relational schema (tables).

ANS:

In the below ER diagram:

- The **Member** entity has a one-to-many relationship with **Invitation**, as each member can receive multiple invitations.
- The **Invitation** entity has a one-to-many relationship with **Response**, as each invitation can have multiple responses (accept or decline).
- The **Response** entity has a one-to-many relationship with **Dinner**, as each response can be associated with multiple dinners (if the member attends multiple dinners).
- The **Dinner** entity has a one-to-one relationship with **Entree**, as each dinner is based on a single entree.
- The **Dinner** entity has a one-to-many relationship with **Dessert**, as each dinner can have multiple desserts.



members

Member_num INT

Member_name VARCHAR(30)

Member_address VARCHAR(50)

Member_city VARCHAR(40)

Member_zip INT

Indexes

dinner

Dinner_code INT

Dinner_desc VARCHAR(50)

Entree_code INT

Indexes

entree

Entree_code INT

Entree_desc VARCHAR(50)

Indexes

response

Response_num INT

Invite_num INT

Member_num INT

Accept_date DATE

Indexes

invitation

Invite_num INT

Invite_date DATE

Dinner_date DATE

Dinner_code INT

Member_num INT

Indexes

dessert

Dessert_code INT

Dessert_desc VARCHAR(50)

Dinner_code INT

Indexes

2. Relational Schema (Tables): Member

- Member_num (PK)
- Member_name
- Member_address

- Member_city
- Member_zip

Invitation

- Invite_num (PK)
- Invite_date
- Dinner_date
- Dinner_code (FK references Dinner.Dinner_code)
- Member_num (FK references Member.Member_num)

Response

- Invite_num (PK, FK references Invitation.Invite_num)
- Member_num (PK, FK references Member.Member_num)
- Accept_date

Dinner

- Dinner_code (PK)
- Dinner_desc
- Entree_code (FK references Entree.Entree_code)

Entree

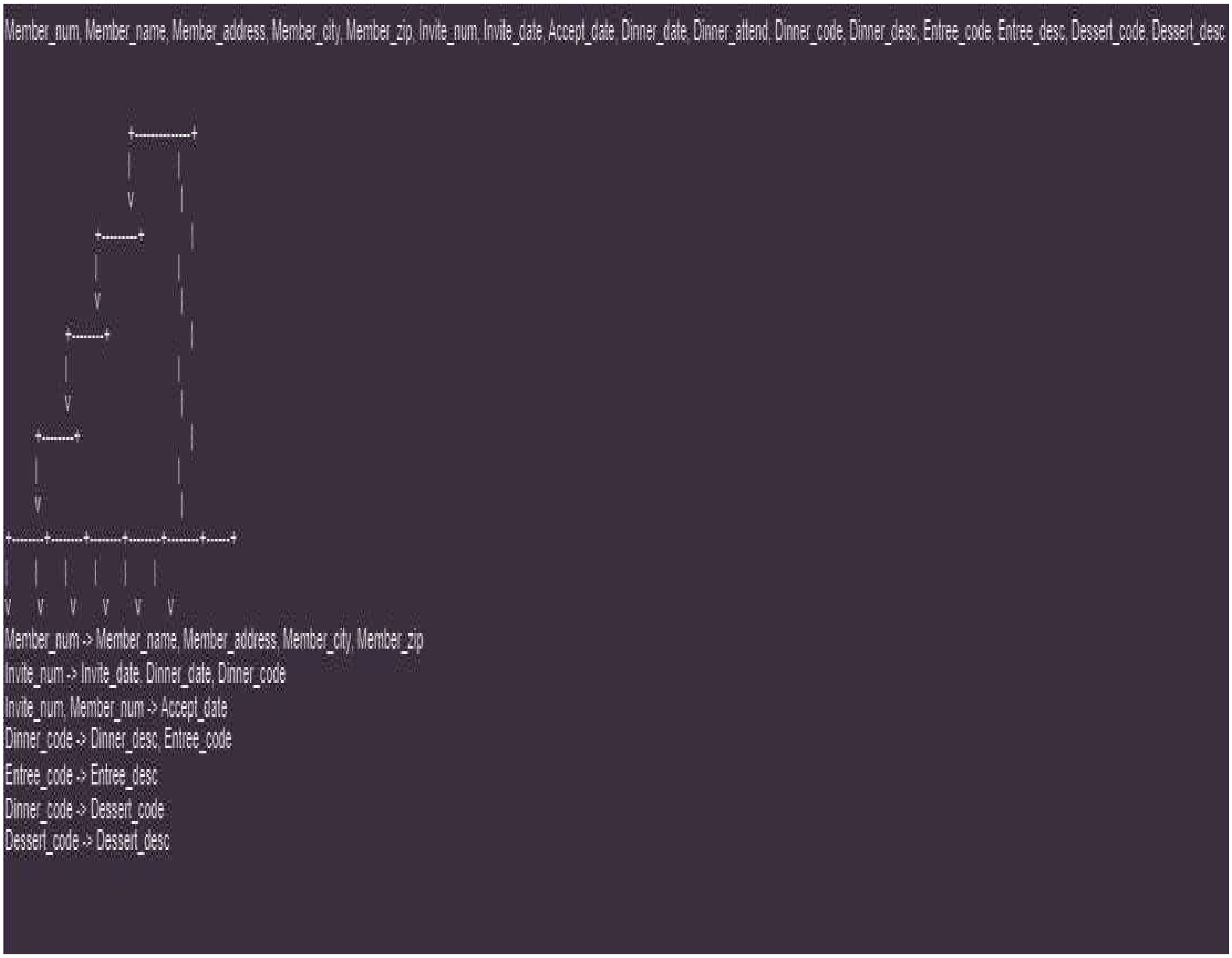
- Entree_code (PK)
- Entree_desc

Dessert

- Dessert_code (PK)
- Dessert_desc
- Dinner_code (FK references Dinner.Dinner_code)

Part II -- Normalization of the Universal Relation

1. **Given the above structure, draw its dependency diagram. Label all transitive and/or partial dependencies.**



```
Member_num, Member_name, Member_address, Member_city, Member_zip, Invite_num, Invite_date, Accept_date, Dinner_date, Dinner_attend, Dinner_code, Dinner_desc, Entree_code, Entree_desc, Dessert_code, Dessert_desc



               +-----------+
               |           |
               v           |
           +-------+        |
           |               |
           v               |
       +------+             |
       |                   |
       v                   |
   +-------+               |
   |                       |
   v                       |
+-------+------+------+------+------+------+-----+
|    |    |    |    |    |
v    v    v    v    v    v
Member_num -> Member_name, Member_address, Member_city, Member_zip
Invite_num -> Invite_date, Dinner_date, Dinner_code
Invite_num, Member_num -> Accept_date
Dinner_code -> Dinner_desc, Entree_code
Entree_code -> Entree_desc
Dinner_code -> Dessert_code
Dessert_code -> Dessert_desc
```

In the above dependency diagram, the transitive dependencies are:

- Member_num -> Member_name, Member_address, Member_city, Member_zip
- Dinner_code -> Dinner_desc, Entree_code
- Entree_code -> Entree_desc
- Dinner_code -> Dessert_code
- Dessert_code -> Dessert_desc

2. Normalize the diagram above to produce dependency diagrams in are in 3NF.

To normalize the universal relation to the Third Normal Form (3NF), we need to remove any transitive dependencies and partial dependencies.

Step 1: Remove transitive dependencies to achieve 3NF

Member(Member_num, Member_name, Member_address, Member_city, Member_zip) Invitation(Invite_num, Invite_date, Dinner_date, Accept_date) Dinner(Dinner_code, Dinner_desc, Entree_code, Dessert_code) Entree(Entree_code, Entree_desc) Dessert(Dessert_code, Dessert_desc)

The resulting tables after normalization to 3NF are the same as the tables obtained from the ER diagram approach. Both methods lead to the same final database design, ensuring data integrity and minimizing redundancy.

Dependency Diagram:

Member_num -> Member_name
Member_address
Member_city
Member_zip

Invite_num -> Invite_date
Accept_date
Member_num
Dinner_date
Dinner_attend
Dinner_code

Dinner_date,
Dinner_code -> Dinner_desc

Entree_code -> Entree_desc

Dessert_code -> Dessert_desc

Member_num, -> Dinner_date
Dinner_attend Dinner_code

Transitive Dependencies:

- Invite_num -> Dinner_date -> Dinner_desc
- Invite_num -> Dinner_code -> Dinner_desc
- Invite_num -> Dinner_code -> Entree_desc

- Invite_num -> Dinner_code -> Dessert_desc
- Member_num, Dinner_attend -> Dinner_date -> Dinner_desc
- Member_num, Dinner_attend -> Dinner_code -> Dinner_desc
- Member_num, Dinner_attend -> Dinner_code -> Entree_desc
- Member_num, Dinner_attend -> Dinner_code -> Dessert_desc

Partial dependencies:

- Invite_num -> Member_num (Member_num is part of a composite key in the Invitation table)
- Invite_num -> Dinner_date (Dinner_date is part of a composite key in the Dinner table)
- Invite_num -> Dinner_code (Dinner_code is part of a composite key in the Dinner table)
- Member_num, Dinner_attend -> Member_num (Member_num is not part of the dependency but is included in the composite key).

Normalization to 3NF:

To normalize the universal connection with 3NF, we must remove the transitive and partial dependencies.

Step 1: Make separate tables for each entity (Member, Invitation, Dinner, Entree, and Dessert) and their attributes.

Code:

```
CREATE TABLE Member (
  Member_num INT NOT NULL,
  Member_name VARCHAR(255) NOT NULL,
  Member_address VARCHAR(255) NOT NULL,
  Member_city VARCHAR(255) NOT NULL,
  Member_zip VARCHAR(10) NOT NULL,
  PRIMARY KEY (Member_num)
);

CREATE TABLE Invitation (
  Invite_num INT NOT NULL,
  Invite_date DATE NOT NULL,
  Accept_date DATE,
  Member_num INT NOT NULL,
  Dinner_date DATE NOT NULL,
  Dinner_attend ENUM('Y', 'N') NOT NULL,
  PRIMARY KEY (Invite_num),
  FOREIGN KEY (Member_num) REFERENCES Member(Member_num)
);

CREATE TABLE Dinner (
  Dinner_date DATE NOT NULL,
  Dinner_code INT NOT NULL,
  Dinner_desc VARCHAR(255) NOT NULL,
  Entree_code INT NOT NULL,
  Dessert_code INT NOT NULL,
  PRIMARY KEY (Dinner_date, Dinner_code),
  FOREIGN KEY (Entree_code) REFERENCES Entree(Entree_code),
  FOREIGN KEY (Dessert_code) REFERENCES Dessert(Dessert_code)
);

CREATE TABLE Entree (
  Entree_code INT NOT NULL,
  Entree_desc VARCHAR(255) NOT NULL,
  PRIMARY KEY (Entree_code)
);
```

```
CREATE TABLE Dessert (
  Dessert_code INT NOT NULL,
  Dessert_desc VARCHAR(255) NOT NULL,
  PRIMARY KEY (Dessert_code)
);
```

Step 2: Remove partial dependencies by creating a new table for the Invitation-Dinner relationship:

Code:

```
CREATE TABLE Invitation_Dinner (
  Invite_num INT NOT NULL,
  Dinner_date DATE NOT NULL,
  Dinner_code INT NOT NULL,
  PRIMARY KEY (Invite_num),
  FOREIGN KEY (Invite_num) REFERENCES Invitation(Invite_num),
  FOREIGN KEY (Dinner_date, Dinner_code) REFERENCES Dinner(Dinner_date, Dinner_code)
);
```

Step 3: Remove transitive dependencies by creating a new table for the Dinner-Entree and Dinner-Dessert relationships:

```
CREATE TABLE Dinner_Entree (
  Dinner_date DATE NOT NULL,
  Dinner_code INT NOT NULL,
  Entree_code INT NOT NULL,
  PRIMARY KEY (Dinner_date, Dinner_code),
  FOREIGN KEY (Dinner_date, Dinner_code) REFERENCES Dinner(Dinner_date, Dinner_code),
  FOREIGN KEY (Entree_code) REFERENCES Entree(Entree_code)
);

CREATE TABLE Dinner_Dessert (
  Dinner_date DATE NOT NULL,
  Dinner_code INT NOT NULL,
  Dessert_code INT NOT NULL,
  PRIMARY KEY (Dinner_date, Dinner_code),
  FOREIGN KEY (Dinner_date, Dinner_code) REFERENCES Dinner(Dinner_date, Dinner_code),
  FOREIGN KEY (Dessert_code) REFERENCES Dessert(Dessert_code)
);
```

Hence the problem solved.