Real-Life App Feature Simulation: *Smart Order Dashboard* **[Talabat-style] :)**

Instructions

- ✓ Write one SQL query per widget.
- ✓ Use clear table aliases and add comments.
- ✓ Think carefully before choosing a JOIN type.
- ✓ Document any errors or challenges during testing.
- ✓ Submit your answers in a single `.sql` or `.docx` file.

Scenario:

You are building a dashboard for restaurant partners on a food delivery platform (like Talabat). Your task is to write SQL queries for each widget to show useful order insights, customer activity, and menu performance.

Provided Tables:

- Customers(CustomerID, FullName, Phone, ReferralID)
- Restaurants(RestaurantID, Name, City)
- Orders(OrderID, CustomerID, RestaurantID, OrderDate, Status)
- OrderItems(OrderItemID, OrderID, ItemName, Quantity, Price)
- Menu(MenuID, RestaurantID, ItemName, Price)

Use the provided database schema and sample data. Write one SQL query for each widget below and use JOINs appropriately. Use clear table aliases, and log any challenges or errors you face while testing.

Widget 1: **Active Orders Summary**
Show all active orders (Status = 'Preparing') with customer name, restaurant name, and order date.

Widget 2: **Restaurant Menu Coverage**
List all menu items offered by each restaurant and whether they have ever been ordered. Include items that have never been ordered.

Widget 3: **Customers Without Orders**
Display all customers, including those who have never placed any orders.

Widget 4: **Full Engagement Report**
Display a full list of customer and order combinations, including customers who never ordered and orders that belong to non-existent customers.

Widget 5: **Referral Tree**
List each customer along with the full name of the person who referred them, if any.

Widget 6: **Menu Performance Tracker**
For each restaurant, show item name, number of times it was ordered, and total quantity sold. Include items even if they were never ordered.

Widget 7: **Unused Customers and Items**
Display customers who never placed an order and menu items that were never ordered. Return a unified list with a type column ('Unused Customer' or 'Unused Item').

Widget 8: **Orders with Missing Menu Price Match**
Show all orders where the ordered item does not exist in the current menu of the restaurant.

Widget 9: **Repeat Customers Report**
List customers who have placed more than one order, showing customer name, total number of orders, first and last order dates.

Widget 10: **Item Referral Revenue**
For each referral chain, show the customer, the person who referred them, and the total amount spent by the referred customer.

Data Modification Widgets – JOINs for UPDATE, DELETE, and INSERT

Real dashboards don't just read data — they maintain it.
In real-world platforms like *Talabat*, backend developers regularly perform operations that adjust prices, clean inactive data, or move old records to archives. These operations often rely on **JOINs** to make sure the right records are updated or removed based on conditions from related tables.

In this section, your role shifts from **data analyst** to **data maintainer**. Each widget below represents a realistic task you might encounter as a backend developer or data engineer. Use JOINs smartly to apply changes across related tables.

Widget 11: Update Prices for Bestsellers

Scenario:
The business team wants to automatically increase prices for high-demand items.
Update the price of any menu item that has been ordered **more than 3 times** by **10%**.

Use UPDATE with a JOIN between Menu and aggregated OrderItems.

Widget 12: Delete Inactive Customers

Scenario:
To comply with data privacy regulations, the system must remove customers who:

- Have **never placed an order**,
- And were **not referred by another customer**.

Use DELETE with a LEFT JOIN to Orders and check ReferralID IS NULL.

Widget 13: Adjust Prices for Inactive Restaurants

Scenario:
The pricing team wants to reduce menu prices by **15%** for all items belonging to restaurants that **never received any orders**.
This is a quick way to encourage more orders from underperforming partners.

Your Task:
Update prices in the Menu table using a JOIN with Restaurants and Orders.
Only update menu items for restaurants that do **not appear at all in the Orders table**.

Hint: Use a LEFT JOIN and check for Orders.OrderID IS NULL in the WHERE clause.
Only UPDATE records in the Menu table — do not use subqueries or functions.

Widget 14: Register VIP Customers

Scenario:
The marketing team asks you to manually **insert** VIP customers into a new table VIPCustomers(CustomerID, FullName).
They provide you a shortlist of **Customer IDs** of people who made several purchases.
Your job is to:

Your Task:
Manually use INSERT statements to add at least **two customers** to the VIPCustomers table,
but before inserting them, write a SELECT statement using JOINs between Customers, Orders, and OrderItems
to confirm that those customers appear in both the Orders and OrderItems tables.

Hint: First, use a JOIN to verify which customers placed orders and have items.
Then, manually insert a few of them using INSERT INTO VIPCustomers VALUES (..., ...)

Widget 15: Order Dispatch Overview

Scenario:
Restaurant owners want to see a live summary of current orders including:

- Customer name
- Restaurant name
- Ordered item names
- Order status

Your Task:
Write a SELECT query that uses **multiple JOINs** between at least **4 tables**:
Customers, Orders, Restaurants, and OrderItems.

Ensure the output shows one row per item in the order, with all related names and the order status.

This widget strengthens your ability to:

- Chain multiple joins correctly
- Use aliases to handle column names clearly
- Think about how data is connected across a full transaction