Clothing Retail Store Database Management System

Saloni Patel 501246930, Abisa Rajkumar 501250245, Immanuel Gnanaseelan 501155878

Section 12

Team 7

Table of Contents:

Introduction

This comprehensive database management system (DBMS) presents a multi-store clothing retail chain. Customer management, staff assignment, store operations, inventory tracking, order processing, and discount management are all supported by the system. The database was created, constructed, and normalized to BCNF, populated with test data, and queries were performed using both SQL and Unix Shell-based menu scripts during assignments A1-A10. The finished product combines all phases of development into a polished, fully standardized design, guaranteeing scalability, consistency, and dependability for real-world retail operations.

Application Description

The Clothing Retail Store Database Management System (DBMS) is intended to handle several essential tasks, such as customer tracking, employee assignments, store operations, sales activity, inventory control, and discount programs. To provide an integrated platform that enables precise record-keeping and efficient business operations, the database stores information on customers, employees, orders, stores, products, and store-level inventory. Store managers as well as executives are the system's primary users, as they can use it to track product availability, monitor employee responsibilities, offer loyalty discounts, and keep an eye on sales performance. Accurate recording of daily transactions is ensured by the DBMS, which also produces insightful data such as identifying best-selling items, tracking premium customer behaviour, and sending out low-stock warnings.

The system is composed of several core entities that are linked by important relationships.Customer (Customer ID, Email, VIP Tier) helps keep track of membership status and consumer data for monitoring and loyalty rewards. Customer ID and Email uniquely identify each customer, while VIP Tier (Silver, Gold, Platinum) determines the discount eligibility. A customer may place many orders, resulting in a one-to-many relationship with the Orders table.

Employee (Employee ID, Role, Email, Store ID, Sales Per Week) keeps track of staff information and associates each employee with a particular store. Weekly sales performance is also monitored. Each employee belongs to exactly one store, forming a many-to-one relationship. Employees may also be connected to orders to track who processed each transaction.

Store (Store ID, Address, Opened Date, Status) represents the operational details of each retail location. Stores employ multiple employees, manage their own inventory, and may complete many customer orders. This results in one-to-many relationships with both Employee and Inventory.

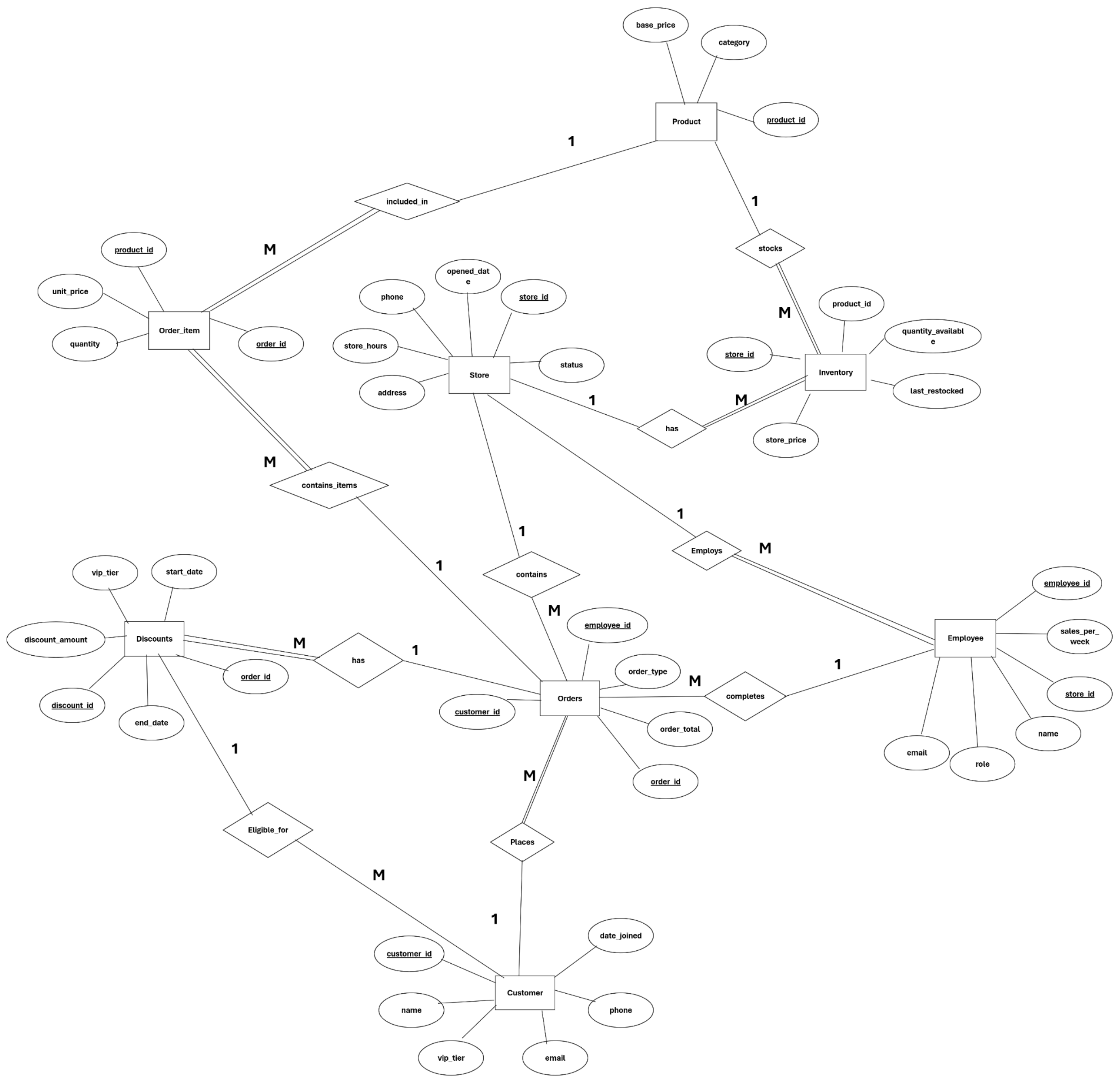
Product (Product ID, Category, Base Price) represents the items sold by the retail chain. Products appear in both Inventory and Order Item records. Each product has one base price, and many stores may carry the same product.

Orders (Order ID, Customer ID, Employee ID, Order Type, Order Total) record sales transactions, establishing connections between customers and employees. The Order Type identifies whether the purchase was online, in-store, or pickup. Since an order may contain multiple products, the system uses a separate Order Item (Order ID, Product ID, Unit Price, Quantity) table to accurately record all purchased items.

To provide precise stock control, Inventory (Store ID, Product ID, Quantity Available, Store Price) tracks how many units of each product are available at each store location. Each row uniquely identifies the quantity of a specific product in a specific store. Inventory is updated whenever an order is placed, and low-stock conditions can trigger alerts.

Discounts (Discount ID, VIP Tier, Discount Amount, Order ID) define discount rules that can be applied to orders based on promotional periods or customer VIP level. An order may receive multiple discounts, creating a one-to-many relationship. The customer's VIP Tier determines which discount amounts may be applied.

ER Model and Final Schema



Key Relationships

STORE 1:M EMPLOYEE *(Each store can have multiple employees)*

STORE 1:M INVENTORY *(Each store can have many inventory rows)*

STORE 1:M ORDERS *(Each store can have many orders)*

CUSTOMER 1:M ORDERS *(Each customer can place many orders)*

EMPLOYEE 1:M ORDERS *(Each employee can complete many orders)*

PRODUCT 1:M INVENTORY *(Each product can appear in many store inventories)*

PRODUCT 1:M ORDER_ITEM *(Each product can appear in many order items)*

PRODUCT 1:M INVENTORY *(Each product can appear in many store inventories*

ORDERS 1:M DISCOUNTS *(Each order can have zero or many discounts)*

ORDERS 1:M ORDER_ITEM *(Each order can contain many line items)*

Relational Schema

STORE(store_id, address, opened_date, status, phone, store_hours)

CUSTOMER(customer_id, name, email, phone, date_joined, vip_tier)

EMPLOYEE(employee_id, store_id, name, email, role, sales_per_week)

PRODUCT(product_id, category, base_price)

INVENTORY(store_id, product_id, quantity_available, last_restocked, store_price)

ORDERS(order_id, customer_id, employee_id, order_type, order_total)

ORDER_ITEM(order_id, product_id, unit_price, quantity)

DISCOUNTS(discount_id, vip_tier, start_date, end_date, discount_amount, order_id)

Relational Schema & SQL Implementation

```
-- 1. STORE
CREATE TABLE Store (
   store_id       NUMBER                 PRIMARY KEY,
   address        VARCHAR2(200)          NOT NULL,
   opened_date          DATE                        DEFAULT SYSDATE NOT NULL,
   status               VARCHAR2(20)                DEFAULT 'OPEN' NOT NULL,
   phone          VARCHAR2(20),          NOT NULL
   store_hours          VARCHAR2(100),
   CONSTRAINT uq_store_phone UNIQUE (phone),
   CONSTRAINT chk_store_status CHECK (status IN ('OPEN', 'CLOSED', 'RENOVATION'))
);


-- 2. CUSTOMER
CREATE TABLE Customer (
   customer_id          NUMBER(10)                  PRIMARY KEY,
   name           VARCHAR2(100)                NOT NULL,
   email                VARCHAR2(200)               NOT NULL,
   phone          VARCHAR2(20),
   date_joined          DATE                        DEFAULT SYSDATE NOT NULL,
   vip_tier             VARCHAR2(20)                      DEFAULT 'NONE' NOT NULL,
   CONSTRAINT uq_customer_email UNIQUE (email),
   CONSTRAINT chk_customer_viptier CHECK (vip_tier IN ('NONE', 'SILVER', 'GOLD',
'PLATINUM'))
);


-- 3. PRODUCT
CREATE TABLE Product (
   product_id           NUMBER(10)         PRIMARY KEY,
   category             VARCHAR2(50)       NOT NULL,
   base_price           NUMBER(10,2)       NOT NULL,
   CONSTRAINT chk_product_price_nonneg CHECK (base_price >= 0)
);
```

```
-- 4. EMPLOYEE
CREATE TABLE Employee (
   employee_id          NUMBER                PRIMARY KEY,
   store_id      NUMBER(10)          NOT NULL,
   name          VARCHAR2(100)       NOT NULL,
   email                VARCHAR2(100)       UNIQUE,
   role                 VARCHAR2(50),
   sales_per_week NUMBER(10,2),
   CONSTRAINT fk_employee_store
      FOREIGN KEY (store_id)
      REFERENCES Store(store_id)
      ON DELETE CASCADE
);


-- 5. ORDERS
CREATE TABLE Orders (
   order_id             NUMBER(12)          PRIMARY KEY,
   customer_id          NUMBER(10)          NOT NULL,
   employee_id          NUMBER              NOT NULL,
   order_type           VARCHAR2(20)        DEFAULT 'ONLINE' NOT NULL,
   order_total          NUMBER(10,2)        NOT NULL,
   order_date           DATE                DEFAULT SYSDATE NOT NULL,
   CONSTRAINT chk_orders_type CHECK (order_type IN ('ONLINE', 'IN_STORE',
'PICKUP')),
   CONSTRAINT chk_orders_total_nonneg CHECK (order_total >= 0),
   CONSTRAINT fk_orders_customer
      FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),
   CONSTRAINT fk_orders_employee
      FOREIGN KEY (employee_id) REFERENCES Employee(employee_id)
);


-- 6. INVENTORY
CREATE TABLE Inventory (
   store_id             NUMBER(10)     NOT NULL,
   product_id           NUMBER(10)     NOT NULL,
   quantity_available   NUMBER(10)     DEFAULT 0 NOT NULL,
   last_restocked       DATE,
   store_price          NUMBER(10,2)    NOT NULL,
   CONSTRAINT pk_inventory PRIMARY KEY (store_id, product_id),
   CONSTRAINT chk_inventory_qty_nonneg CHECK (quantity_available >= 0),
   CONSTRAINT chk_inventory_price_nonneg CHECK (store_price >= 0),
   CONSTRAINT fk_inventory_store
      FOREIGN KEY (store_id) REFERENCES Store(store_id),
   CONSTRAINT fk_inventory_product
      FOREIGN KEY (product_id) REFERENCES Product(product_id)
);


-- 7. ORDER_ITEM
CREATE TABLE Order_Item (
   order_id            NUMBER(12)          NOT NULL,
   product_id          NUMBER(10)          NOT NULL,
   quantity            NUMBER(10)          DEFAULT 1 NOT NULL,
   unit_price          NUMBER(10,2)                NOT NULL,
   CONSTRAINT pk_order_item PRIMARY KEY (order_id, product_id),
   CONSTRAINT chk_order_item_qty_pos CHECK (quantity > 0),
   CONSTRAINT chk_order_item_price_nonneg CHECK (unit_price >= 0),
   CONSTRAINT fk_order_item_order
      FOREIGN KEY (order_id) REFERENCES Orders(order_id)
      ON DELETE CASCADE,
   CONSTRAINT fk_order_item_product
      FOREIGN KEY (product_id) REFERENCES Product(product_id)
);



-- 8. DISCOUNTS
CREATE TABLE Discounts (
   discount_id             NUMBER(12)          PRIMARY KEY,
   vip_tier                VARCHAR2(20)        NOT NULL,
   start_date              DATE                NOT NULL,
   end_date                DATE                NOT NULL,
   discount_amount         NUMBER(5,2)         NOT NULL,
   order_id                NUMBER(12)          NOT NULL,
   CONSTRAINT chk_discounts_viptier CHECK (vip_tier IN
('NONE','SILVER','GOLD','PLATINUM')),
   CONSTRAINT chk_discounts_pct CHECK (discount_amount >= 0 AND discount_amount
<= 100),
   CONSTRAINT chk_discounts_dates CHECK (end_date >= start_date),
   CONSTRAINT fk_discounts_order
      FOREIGN KEY (order_id) REFERENCES Orders(order_id)
);
```

Data Population & Sample Records

```
-- 1. STORE
INSERT INTO Store (store_id, address, opened_date, status, phone, store_hours)
VALUES (1, '100 King St W, Toronto, ON', DATE '2022-03-01', 'OPEN', '416-555-1000', 'Mon-Fri
10:00-21:00');

INSERT INTO Store (store_id, address, opened_date, status, phone, store_hours)
VALUES (2, '200 Queen St W, Toronto, ON', DATE '2023-05-15', 'OPEN', '416-555-2000', 'Mon-Sun
11:00-19:00');


-- 2. CUSTOMER
INSERT INTO Customer (customer_id, name, email, phone, date_joined, vip_tier)
VALUES (1001, 'Saloni Patel',  'saloni.patel@hotmail.com',  '647-555-0101', DATE '2024-01-10',
'SILVER');

INSERT INTO Customer (customer_id, name, email, phone, date_joined, vip_tier)
VALUES (1002, 'Abisa Rajkumar', 'abisa.rajkumar@gmail.com', '647-555-0202', DATE '2024-02-05',
'GOLD');

INSERT INTO Customer (customer_id, name, email, phone, date_joined, vip_tier)
VALUES (1003, 'Nehal Goel', 'nehal.goel@yahoo.ca', NULL, DATE '2024-03-01', 'NONE');


-- 3. PRODUCT
INSERT INTO Product (product_id, category, base_price)
VALUES (100, 'Tops',        30.00);

INSERT INTO Product (product_id, category, base_price)
VALUES (101, 'Jeans',       60.00);

INSERT INTO Product (product_id, category, base_price)
VALUES (102, 'Accessories', 15.00);
```

```
-- 4. EMPLOYEE
INSERT INTO Employee (employee_id, store_id, name, email, role, sales_per_week)
VALUES (5001, 1, 'Mike Brown',  'mike.brown@corp.com',  'Manager',        15000.00);

INSERT INTO Employee (employee_id, store_id, name, email, role, sales_per_week)
VALUES (5002, 1, 'Immanuel Gnanaseelan',   'immanuel.gnanaseelan@corp.com',   'Sales Associate',
8000.00);

INSERT INTO Employee (employee_id, store_id, name, email, role, sales_per_week)
VALUES (5003, 2, 'Mia Johnson',   'mia.johnson@corp.com',   'Sales Associate',  9000.00);


-- 5. ORDERS
-- Order 90001: SILVER customer, handled by employee 5002
INSERT INTO Orders (order_id, customer_id, employee_id, order_type, order_total, order_date)
VALUES (90001, 1001, 5002, 'ONLINE', 70.00, DATE '2024-11-01');

-- Order 90002: GOLD customer, in-store purchase
INSERT INTO Orders (order_id, customer_id, employee_id, order_type, order_total, order_date)
VALUES (90002, 1002, 5003, 'IN_STORE', 60.00, DATE '2024-11-02');

-- Order 90003: repeat customer, in-store
INSERT INTO Orders (order_id, customer_id, employee_id, order_type, order_total, order_date)
VALUES (90003, 1001, 5002, 'IN_STORE', 44.00, DATE '2024-11-03');


-- 6. INVENTORY
-- Store 1
INSERT INTO Inventory (store_id, product_id, quantity_available, last_restocked, store_price)
VALUES (1, 100, 50, DATE '2024-10-01', 28.00);

INSERT INTO Inventory (store_id, product_id, quantity_available, last_restocked, store_price)
VALUES (1, 101, 35, DATE '2024-10-05', 60.00);

-- Store 2
INSERT INTO Inventory (store_id, product_id, quantity_available, last_restocked, store_price)
VALUES (2, 100, 20, DATE '2024-10-03', 30.00);

INSERT INTO Inventory (store_id, product_id, quantity_available, last_restocked, store_price)
VALUES (2, 102, 60, DATE '2024-10-06', 14.00);


-- 7. ORDER_ITEM
-- Order 90001: 2 Tops + 1 Accessory
INSERT INTO Order_Item (order_id, product_id, quantity, unit_price)
VALUES (90001, 100, 2, 28.00);

INSERT INTO Order_Item (order_id, product_id, quantity, unit_price)
VALUES (90001, 102, 1, 14.00);

-- Order 90002: 1 Jeans
INSERT INTO Order_Item (order_id, product_id, quantity, unit_price)
VALUES (90002, 101, 1, 60.00);

-- Order 90003: 1 Top + 1 Accessory
INSERT INTO Order_Item (order_id, product_id, quantity, unit_price)
VALUES (90003, 100, 1, 30.00);

INSERT INTO Order_Item (order_id, product_id, quantity, unit_price)
VALUES (90003, 102, 1, 14.00);


-- 8. DISCOUNTS
INSERT INTO Discounts (discount_id, vip_tier, start_date, end_date, discount_amount, order_id)
VALUES (1, 'SILVER', DATE '2024-11-01', DATE '2024-11-30', 10.00, 90001);

INSERT INTO Discounts (discount_id, vip_tier, start_date, end_date, discount_amount, order_id)
VALUES (2, 'GOLD',   DATE '2024-11-01', DATE '2024-11-15', 15.00, 90002);

INSERT INTO Discounts (discount_id, vip_tier, start_date, end_date, discount_amount, order_id)
VALUES (3, 'NONE',   DATE '2024-11-03', DATE '2024-11-30',  0.00, 90003);
```

Simple Queries & Views

QUERIES

Q: Outputs Employees by Weekly Sales in ascending order

```
SELECT
  employee_id     AS "Employee ID",
  name            AS "Employee Name",
  role            AS "Role",
  sales_per_week  AS "Sales/Week ($)",
  store_id        AS "Store ID"
FROM Employee
ORDER BY sales_per_week DESC, name ASC;
```

Q: Outputs each type of Employee Role that exists in every store, without repeating duplicates

```
SELECT DISTINCT store_id AS "Store #",
                role AS "Employee Role"
FROM Employee
ORDER BY "Store #", "Employee Role";
```

Q: Outputs Store that has low stock in their inventory that is less than or equal to 10, in ascending order

```
SELECT store_id AS "Store",
       product_id AS "Product #",
       category AS "Category",
       quantity_available AS "Quantity",
       price AS "Price",
       last_restocked AS "Last Restock"
FROM inventory
WHERE quantity_available <= 10
ORDER BY quantity_available ASC, store_id, product_id;
```

Q: Outputs all the discounts that are currently active today in descending order, showing the largest discounts first

```
SELECT discount_id AS "Discount #",
    vip_tier AS "Tier",
    discount_amount AS "Amount",
    order_id AS "Order #",
    start_date AS "Start",
    end_date AS "End"
FROM discounts
WHERE SYSDATE BETWEEN start_date AND end_date
ORDER BY discount_amount DESC, end_date DESC;
```

Q: Outputs the largest amount of money spent on Orders for each Customer, in descending order

```
SELECT
  customer_id            AS "Customer ID",
  COUNT(*)               AS "Orders",
  ROUND(SUM(price),2)    AS "Total Spend ($)",
  ROUND(AVG(price),2)    AS "Avg Order ($)"
FROM orders
GROUP BY customer_id
ORDER BY "Total Spend ($)" DESC;
```

Q: Outputs the VIP Customers based on their joined date in descending order

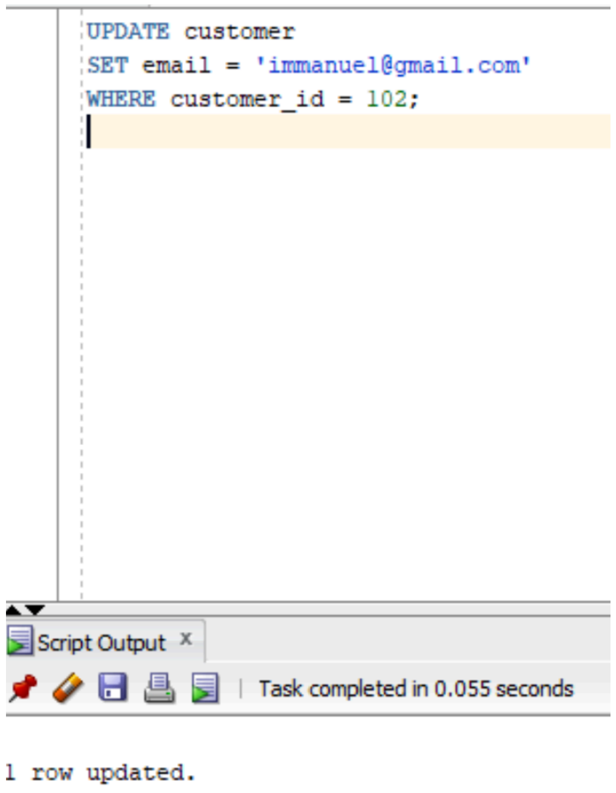
```
SELECT customer_id AS "Customer #",
    name AS "Name",
    email AS "Email",
    date_joined AS "Joined Date",
    vip_tier AS "Tier"
FROM Customer
WHERE vip_tier <> 'NONE'
ORDER BY date_joined DESC;
```

Q: Output the total number of Stores by status in descending order

```
SELECT
  status          AS "Status",
  COUNT(*)        AS "Total Stores"
FROM Store
GROUP BY status
ORDER BY "Total Stores" DESC, status ASC;
```

Q: Update customer email where id is '102'

```
UPDATE customer
SET email = 'immanuel@gmail.com'
WHERE customer_id = 102;
```



```
UPDATE customer
SET email = 'immanuel@gmail.com'
WHERE customer_id = 102;
```

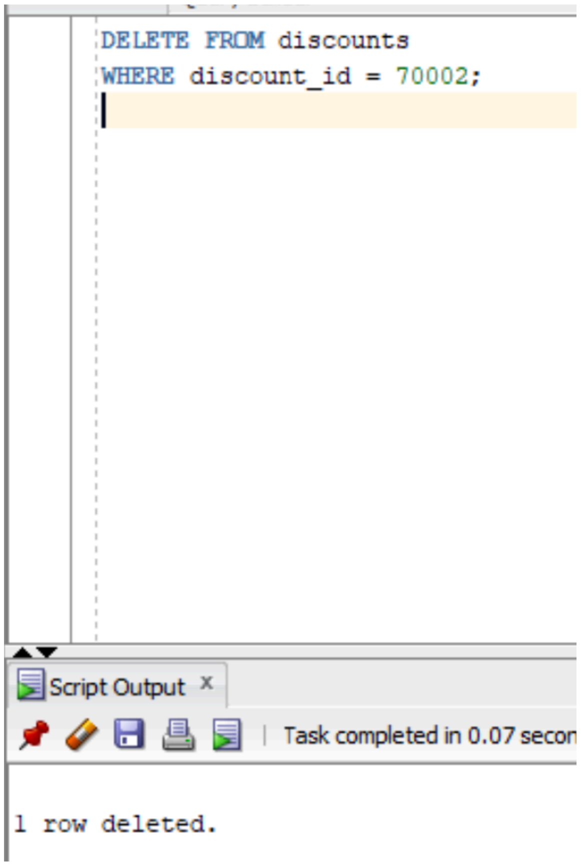
```
Task completed in 0.055 seconds
```

```
1 row updated.
```

Q: Update VIP tier for customer

```
UPDATE customers
SET vip_tier = 'SILVER'
WHERE customer_id = 102;
```

Q: Delete a discount record



```
DELETE FROM discounts
WHERE discount_id = 70002;
```

```
1 row deleted.
```

View, Join, and Advanced Queries:

Q: View v_customer_spend, shows one row per customer showing how many orders they've placed, their total spend, their average order value, and their name and VIP tier.

```
CREATE OR REPLACE VIEW v_customer_spend AS
SELECT
  c.customer_id,
  c.name          AS customer_name,
  c.vip_tier      AS tier,
  COUNT(o.order_id)               AS orders_count,
  ROUND(SUM(o.price), 2)          AS total_spend,
  ROUND(AVG(o.price), 2)          AS avg_order
FROM customer c
LEFT JOIN orders o
  ON o.customer_id = c.customer_id
GROUP BY c.customer_id, c.name, c.vip_tier;

-- Example:
SELECT customer_id AS "Customer ID", customer_name AS "Customer", tier AS "Tier",
       orders_count AS "Orders", total_spend AS "Total Spend ($)", avg_order AS "Avg Order ($)"
```

```
FROM v_customer_spend
ORDER BY "Total Spend ($)" DESC, "Avg Order ($)" DESC, "Customer";
```

Q: View v_open_stores, only stores currently OPEN

```
CREATE OR REPLACE VIEW v_open_stores AS
SELECT store_id, address, opened_date, phone, store_hours
FROM Store
WHERE status = 'OPEN';

-- Example:
SELECT * FROM v_open_stores ORDER BY opened_date DESC, store_id;
```

Q: View v_recent_customers_2025, Customers who joined in the year 2025.

```
CREATE OR REPLACE VIEW v_recent_customers_2025 AS
SELECT customer_id, name, email, phone, date_joined, vip_tier
FROM Customer
WHERE date_joined >= DATE '2025-01-01'
  AND date_joined <  DATE '2026-01-01';

-- Example:
SELECT * FROM v_recent_customers_2025 ORDER BY date_joined DESC, name;
```

Q: Top Customers (With Tiers) shows customers ranked by total spend, then average order.

```
SELECT
  c.customer_id      AS "Customer ID",
  c.name             AS "Customer",
  c.vip_tier         AS "Tier",
  COUNT(*)           AS "Orders",
  ROUND(SUM(o.price), 2) AS "Total Spend ($)",
  ROUND(AVG(o.price), 2) AS "Average Order ($)"
FROM orders o
JOIN customer c
  ON c.customer_id = o.customer_id
GROUP BY c.customer_id, c.name, c.vip_tier
ORDER BY "Total Spend ($)" DESC,
         "Average Order ($)" DESC,
         "Customer";
```

Q: In-store/Pickup Orders Handled with Employee & Store

```
SELECT
  o.order_id         AS "Order #",
  s.address          AS "Store Address",
  e.name             AS "Employee Name",
  o.order_type       AS "Type",
  ROUND(o.price, 2)  AS "Price ($)"
FROM orders o
JOIN store s     ON s.store_id     = o.store_id
JOIN employee e  ON e.employee_id  = o.employee_id
WHERE o.order_type IN ('IN_STORE','PICKUP')
ORDER BY s.store_id, e.name, o.order_id;
```

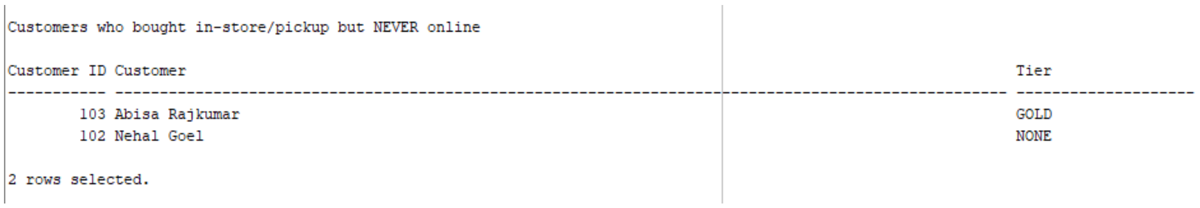
Q: Show Employee Sales by Store through the revenue handled by each employee at each store.
Joins: orders + employee + store

```
SELECT
  s.store_id                     AS "Store",
  e.employee_id                  AS "Emp ID",
  e.name                   AS "Employee",
  COUNT(*)                       AS "Orders",
  ROUND(SUM(o.price), 2)    AS "Revenue ($)"
FROM orders o
JOIN employee e ON e.employee_id = o.employee_id
JOIN store    s ON s.store_id    = o.store_id
WHERE o.order_type IN ('IN_STORE','PICKUP')
GROUP BY s.store_id, e.employee_id, e.name
ORDER BY "Revenue ($)" DESC, "Store", "Employee";
```

Advanced Queries

Q1: Outputs customers who have placed in-store or pickup orders but have never ordered online

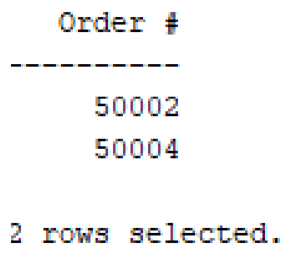
```
SELECT c.customer_id AS "Customer ID",
       c.name        AS "Customer",
       c.vip_tier    AS "Tier"
FROM   customer c
WHERE  EXISTS (
         SELECT 1 FROM orders o
         WHERE  o.customer_id = c.customer_id
         AND    o.order_type IN ('IN_STORE','PICKUP')
       )
AND    NOT EXISTS (
         SELECT 1 FROM orders o2
         WHERE  o2.customer_id = c.customer_id
         AND    o2.order_type = 'ONLINE'
       )
ORDER BY "Customer";
```



```
Customers who bought in-store/pickup but NEVER online

Customer ID Customer                                                                                                   Tier
----------- ---------------------------------------------------------------------------------------------------- --------------------
        103 Abisa Rajkumar                                                                                         GOLD
        102 Nehal Goel                                                                                             NONE

2 rows selected.
```

Q2: Display orders that do not have any discount

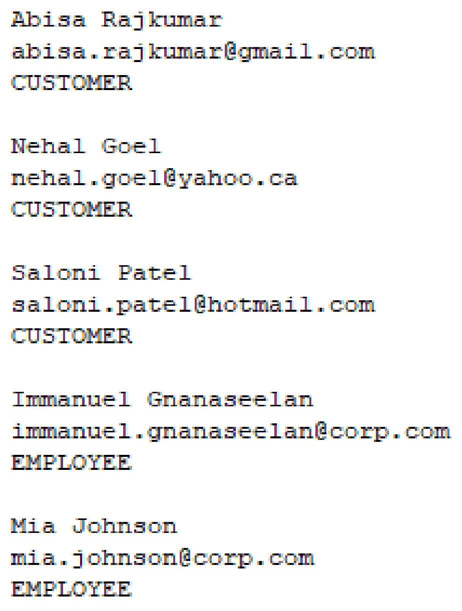
```
SELECT
   o.order_id AS "Order #"
FROM
   orders o
MINUS
SELECT
   d.order_id
FROM
   discounts d
ORDER BY
   1;
```



```
   Order #
----------
     50002
     50004

2 rows selected.
```

Q3: Output combined directory of all people — both customers and employees — with their names, emails, and type (Customer or Employee).

```
SELECT
   RTRIM(name)  AS "Name",
   RTRIM(email) AS "Email",
   'CUSTOMER'   AS "Type"
FROM customer
UNION ALL
SELECT
   RTRIM(name)  AS "Name",
   RTRIM(email) AS "Email",
   'EMPLOYEE'   AS "Type"
FROM employee
ORDER BY
   "Type", "Name";
```



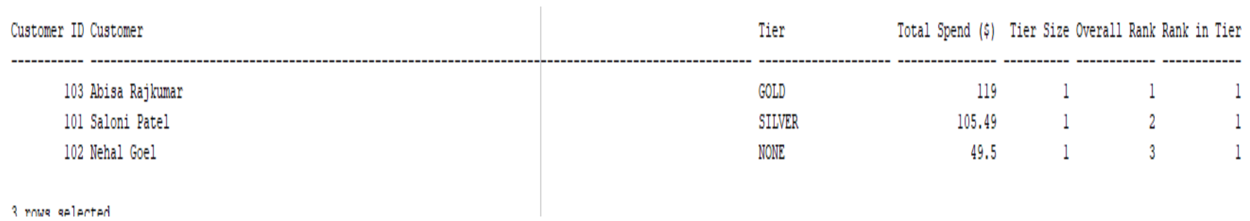
```
Abisa Rajkumar
abisa.rajkumar@gmail.com
CUSTOMER

Nehal Goel
nehal.goel@yahoo.ca
CUSTOMER

Saloni Patel
saloni.patel@hotmail.com
CUSTOMER

Immanuel Gnanaseelan
immanuel.gnanaseelan@corp.com
EMPLOYEE

Mia Johnson
mia.johnson@corp.com
EMPLOYEE
```

Q4: Calculate and display store-level revenue statistics (total, average, and standard deviation of order prices) for in-store and pickup orders

```
SELECT s.store_id                AS "Store",
       COUNT(DISTINCT e.employee_id) AS "Employees",
       ROUND(SUM(o.price),2)        AS "Revenue ($)",
       ROUND(AVG(o.price),2)        AS "Avg Order ($)",
       ROUND(STDDEV(o.price),2)     AS "StdDev Order ($)"
FROM   orders   o
JOIN   employee e ON e.employee_id = o.employee_id
JOIN   store    s ON s.store_id    = o.store_id
WHERE  o.order_type IN ('IN_STORE','PICKUP')
GROUP  BY s.store_id
HAVING SUM(o.price) > 0
ORDER  BY "Revenue ($)" DESC, "Store";
```

Q5: Ranks customers within their VIP tier based on total spending

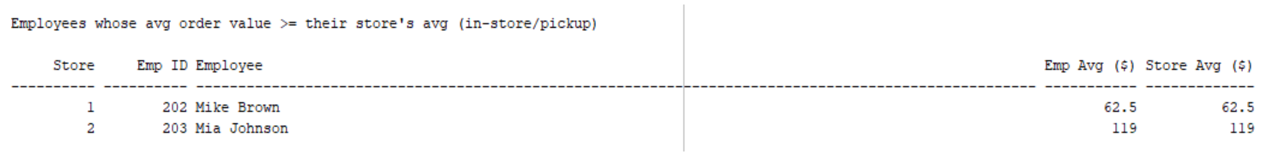
```
WITH spends AS (
  SELECT
     c.customer_id,
     c.name           AS customer_name,
     NVL(c.vip_tier,'NONE') AS vip_tier,
     NVL(SUM(o.price),0)    AS total_spend
  FROM   customer c
  LEFT JOIN orders o
       ON o.customer_id = c.customer_id
  GROUP  BY c.customer_id, c.name, c.vip_tier
)
SELECT
    customer_id                    AS "Customer ID",
    customer_name                  AS "Customer",
    vip_tier                       AS "Tier",
    ROUND(total_spend,2)           AS "Total Spend ($)",
    COUNT(*) OVER (PARTITION BY vip_tier)               AS "Tier Size",
    DENSE_RANK() OVER (ORDER BY total_spend DESC)            AS "Overall Rank",
    DENSE_RANK() OVER (PARTITION BY vip_tier
                 ORDER BY total_spend DESC)            AS "Rank in Tier"
FROM spends
ORDER BY "Overall Rank", "Tier", "Rank in Tier", "Customer";
```



```
Customer ID Customer                                                   Tier                Total Spend ($)  Tier Size Overall Rank Rank in Tier
----------- ------------------------------------------------------- ------------------- -------------- ---------- ------------ ------------
        103 Abisa Rajkumar                                          GOLD                           119          1            1            1
        101 Saloni Patel                                            SILVER                      105.49          1            2            1
        102 Nehal Goel                                              NONE                          49.5          1            3            1

3 rows selected
```

Q6: Finds employees whose average order value is greater than or equal to their store's average

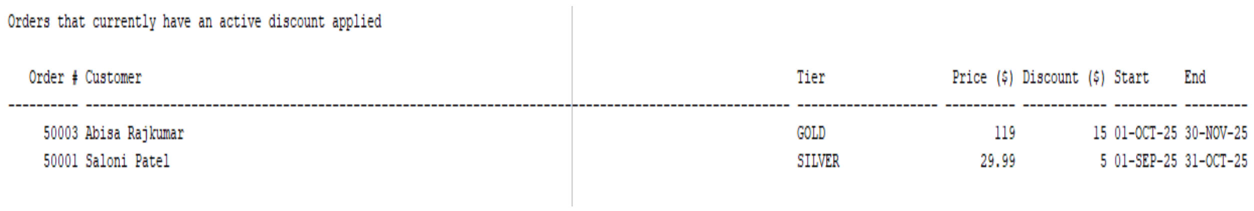
```
SELECT s.store_id AS "Store",
       e.employee_id AS "Emp ID",
       e.name AS "Employee",
       ROUND(AVG(o.price),2) AS "Emp Avg ($)",
       ROUND( (SELECT AVG(o2.price)
              FROM orders o2
              WHERE o2.store_id = s.store_id
                AND o2.order_type IN ('IN_STORE','PICKUP')), 2) AS "Store Avg ($)"
FROM   orders   o
JOIN   employee e ON e.employee_id = o.employee_id
JOIN   store    s ON s.store_id    = o.store_id
WHERE  o.order_type IN ('IN_STORE','PICKUP')
GROUP  BY s.store_id, e.employee_id, e.name
HAVING AVG(o.price) >= (SELECT AVG(o2.price)
                      FROM orders o2
                      WHERE o2.store_id = s.store_id
                        AND o2.order_type IN ('IN_STORE','PICKUP'))
ORDER  BY "Store","Emp Avg ($)" DESC,"Employee";
```



```
Employees whose avg order value >= their store's avg (in-store/pickup)

     Store     Emp ID Employee                                                                  Emp Avg ($) Store Avg ($)
---------- ---------- ------------------------------------------------------------------------- ----------- -------------
         1        202 Mike Brown                                                                       62.5          62.5
         2        203 Mia Johnson                                                                       119           119
```

Q7: Lists orders that currently have active discounts

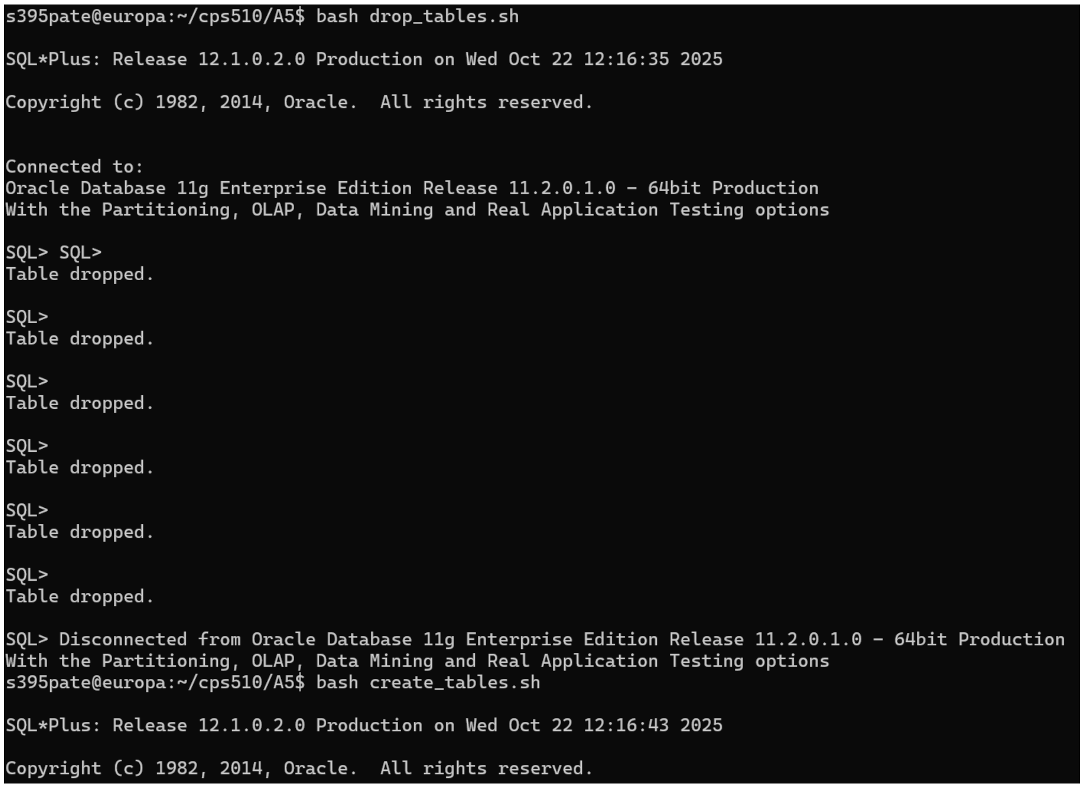
```
SELECT o.order_id   AS "Order #",
       c.name       AS "Customer",
       c.vip_tier   AS "Tier",
       ROUND(o.price,2) AS "Price ($)",
       d.discount_amount AS "Discount ($)",
       d.start_date AS "Start",
       d.end_date   AS "End"
FROM   orders o
JOIN   customer c  ON c.customer_id = o.customer_id
JOIN   discounts d ON d.order_id    = o.order_id
WHERE  SYSDATE BETWEEN d.start_date AND d.end_date
ORDER  BY d.discount_amount DESC, o.order_id;
```



```
Orders that currently have an active discount applied

   Order # Customer                                                                                              Tier                  Price ($) Discount ($) Start     End
---------- ---------------------------------------------------------------------------------------------------- -------------------- ---------- ------------ --------- ---------
     50003 Abisa Rajkumar                                                                                        GOLD                        119           15 01-OCT-25 30-NOV-25
     50001 Saloni Patel                                                                                          SILVER                    29.99            5 01-SEP-25 31-OCT-25
```

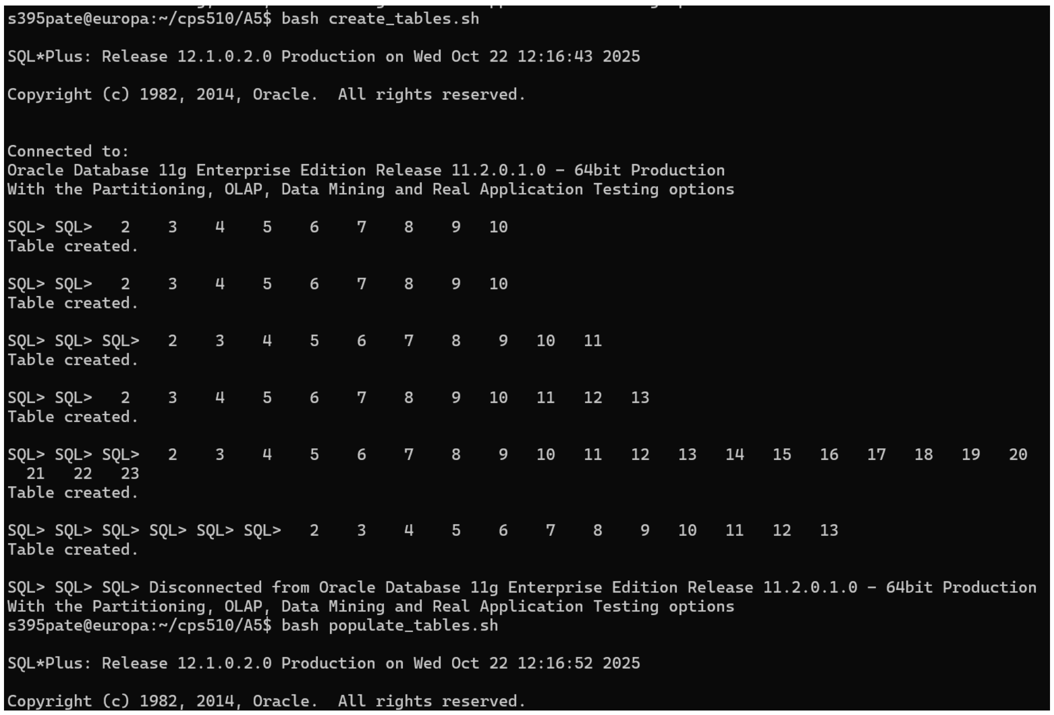
Screenshots from Shell:

bash drop_tables.sh



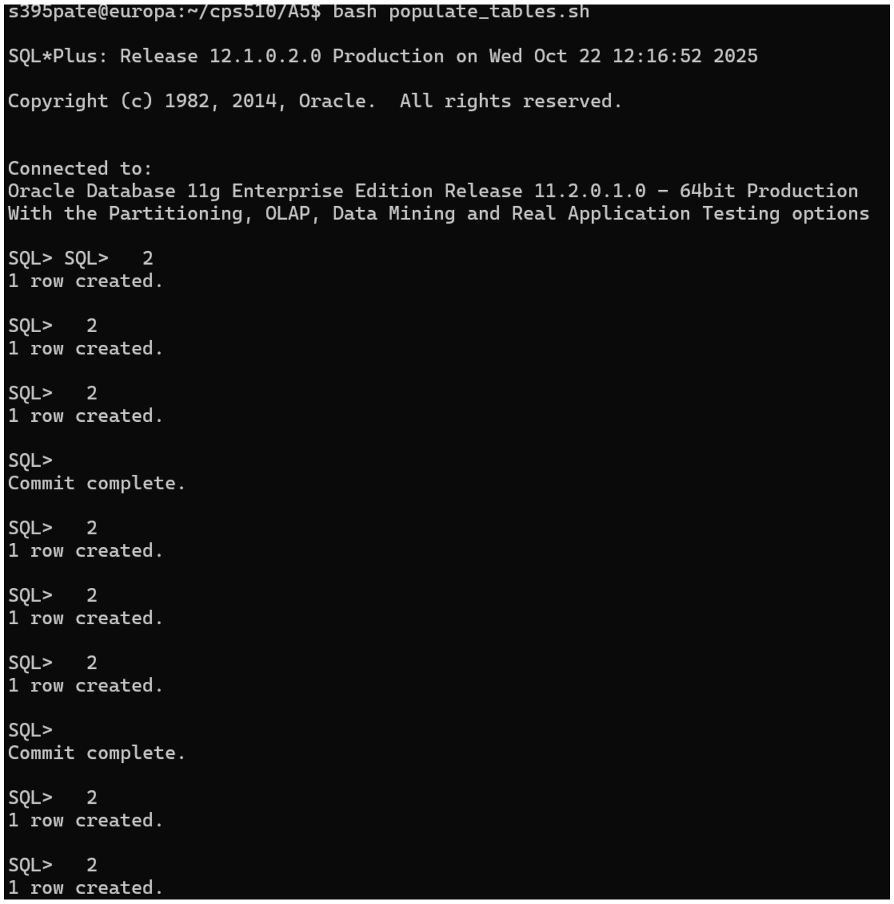
```
s395pate@europa:~/cps510/A5$ bash drop_tables.sh

SQL*Plus: Release 12.1.0.2.0 Production on Wed Oct 22 12:16:35 2025

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
s395pate@europa:~/cps510/A5$ bash create_tables.sh

SQL*Plus: Release 12.1.0.2.0 Production on Wed Oct 22 12:16:43 2025

Copyright (c) 1982, 2014, Oracle.  All rights reserved.
```

bash create_tables.sh

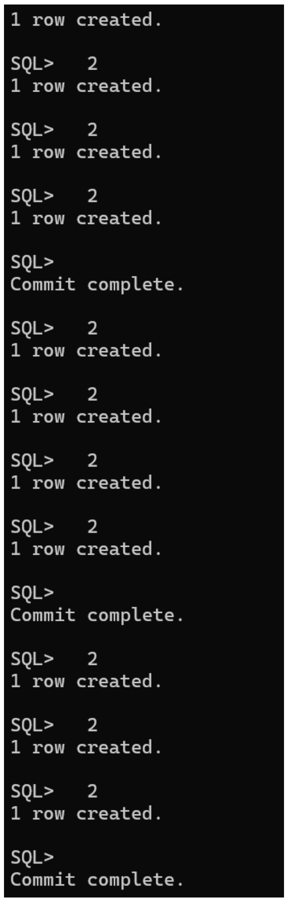


```
s395pate@europa:~/cps510/A5$ bash create_tables.sh

SQL*Plus: Release 12.1.0.2.0 Production on Wed Oct 22 12:16:43 2025

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL>   2    3    4    5    6    7    8    9   10
Table created.

SQL> SQL>   2    3    4    5    6    7    8    9   10
Table created.

SQL> SQL> SQL>   2    3    4    5    6    7    8    9   10   11
Table created.

SQL> SQL>   2    3    4    5    6    7    8    9   10   11   12   13
Table created.

SQL> SQL> SQL>   2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20
  21   22   23
Table created.

SQL> SQL> SQL> SQL> SQL> SQL>   2    3    4    5    6    7    8    9   10   11   12   13
Table created.

SQL> SQL> SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
s395pate@europa:~/cps510/A5$ bash populate_tables.sh

SQL*Plus: Release 12.1.0.2.0 Production on Wed Oct 22 12:16:52 2025

Copyright (c) 1982, 2014, Oracle.  All rights reserved.
```

bash populate_tables

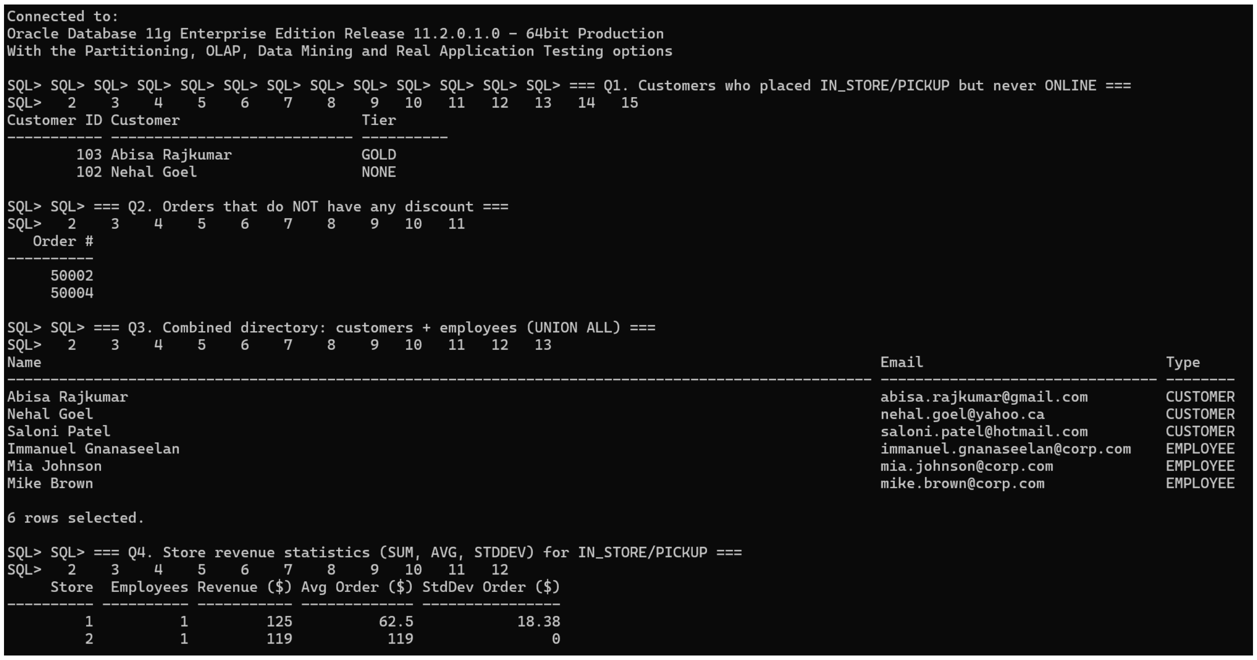


```
s395pate@europa:~/cps510/A5$ bash populate_tables.sh

SQL*Plus: Release 12.1.0.2.0 Production on Wed Oct 22 12:16:52 2025

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL>   2
1 row created.

SQL>   2
1 row created.

SQL>   2
1 row created.

SQL>
Commit complete.

SQL>   2
1 row created.

SQL>   2
1 row created.

SQL>   2
1 row created.

SQL>
Commit complete.

SQL>   2
1 row created.

SQL>   2
1 row created.
```

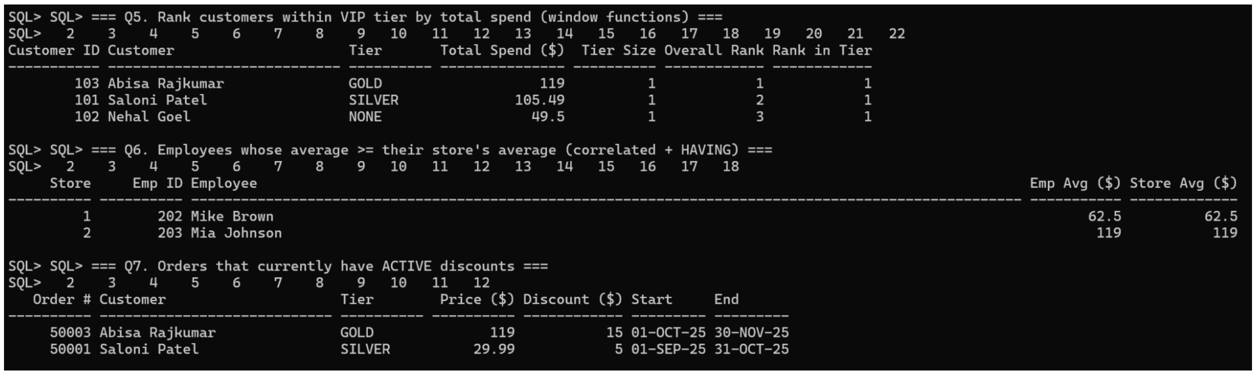


```
1 row created.

SQL>   2
1 row created.

SQL>   2
1 row created.

SQL>   2
1 row created.

SQL>
Commit complete.

SQL>   2
1 row created.

SQL>   2
1 row created.

SQL>   2
1 row created.

SQL>   2
1 row created.

SQL>
Commit complete.

SQL>   2
1 row created.

SQL>   2
1 row created.

SQL>   2
1 row created.

SQL>
Commit complete.
```

bash queries.sh



```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL> === Q1. Customers who placed IN_STORE/PICKUP but never ONLINE ===
SQL>   2    3    4    5    6    7    8    9   10   11   12   13   14   15
Customer ID Customer                     Tier
----------- ---------------------------- ----------
        103 Abisa Rajkumar               GOLD
        102 Nehal Goel                   NONE

SQL> SQL> === Q2. Orders that do NOT have any discount ===
SQL>   2    3    4    5    6    7    8    9   10   11
   Order #
----------
     50002
     50004

SQL> SQL> === Q3. Combined directory: customers + employees (UNION ALL) ===
SQL>   2    3    4    5    6    7    8    9   10   11   12   13
Name                                                                                                 Email                            Type
---------------------------------------------------------------------------------------------------- -------------------------------- --------
Abisa Rajkumar                                                                                       abisa.rajkumar@gmail.com         CUSTOMER
Nehal Goel                                                                                           nehal.goel@yahoo.ca              CUSTOMER
Saloni Patel                                                                                         saloni.patel@hotmail.com         CUSTOMER
Immanuel Gnanaseelan                                                                                 immanuel.gnanaseelan@corp.com    EMPLOYEE
Mia Johnson                                                                                          mia.johnson@corp.com             EMPLOYEE
Mike Brown                                                                                           mike.brown@corp.com              EMPLOYEE

6 rows selected.

SQL> SQL> === Q4. Store revenue statistics (SUM, AVG, STDDEV) for IN_STORE/PICKUP ===
SQL>   2    3    4    5    6    7    8    9   10   11   12
     Store  Employees Revenue ($) Avg Order ($) StdDev Order ($)
---------- ---------- ----------- ------------- ----------------
         1          1         125          62.5            18.38
         2          1         119           119                0
```



```
SQL> SQL> === Q5. Rank customers within VIP tier by total spend (window functions) ===
SQL>   2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20   21   22
Customer ID Customer                     Tier       Total Spend ($)  Tier Size Overall Rank Rank in Tier
----------- ---------------------------- ---------- --------------- ---------- ------------ ------------
        103 Abisa Rajkumar               GOLD                   119          1            1            1
        101 Saloni Patel                 SILVER              105.49          1            2            1
        102 Nehal Goel                   NONE                  49.5          1            3            1

SQL> SQL> === Q6. Employees whose average >= their store's average (correlated + HAVING) ===
SQL>   2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18
     Store     Emp ID Employee                                                                                              Emp Avg ($) Store Avg ($)
---------- ---------- ---------------------------------------------------------------------------------------------------- ----------- -------------
         1        202 Mike Brown                                                                                                  62.5          62.5
         2        203 Mia Johnson                                                                                                  119           119

SQL> SQL> === Q7. Orders that currently have ACTIVE discounts ===
SQL>   2    3    4    5    6    7    8    9   10   11   12
   Order # Customer                     Tier        Price ($) Discount ($) Start     End
---------- ---------------------------- ---------- ---------- ------------ --------- ---------
     50003 Abisa Rajkumar               GOLD              119           15 01-OCT-25 30-NOV-25
     50001 Saloni Patel                 SILVER          29.99            5 01-SEP-25 31-OCT-25
```

Functional Dependencies

Table: STORE
Primary Key: store_id
Candidate Keys: store_id, phone

Minimal Functional Dependencies

- store_id → address
- store_id → opened_date
- store_id → status
- store_id → phone
- store_id → store_hours
- phone → store_id

Set Format:
{store_id} → {address, opened_date, status, phone, store_hours}
{phone} → {store_id}

Each store_id uniquely identifies one of its attributes (one-to-many FD). Since phone is unique, we also have a 1-to-1 FD. These FDs show that store_id is the primary determinant for all the other attributes, and phone is an alternate key.

Table: CUSTOMER
Primary Key: customer_id
Candidate Keys: customer_id, email

Minimal Functional Dependencies
- customer_id → name
- customer_id → email
- customer_id → phone
- customer_id → date_joined
- customer_id → vip_tier
- email → customer_id

Set Format:
{customer_id} → {name, email, phone, date_joined, vip_tier}
{email} → {customer_id}

Each customer ID determines all descriptive attributes. Email → customer_id forms a 1-to-1 FD since email is unique. Both can uniquely identify the same record, but customer_id is the main key used throughout the database.

Table: EMPLOYEE
Primary Key: employee_id
Candidate Keys: employee_id, email

Minimal Functional Dependencies

- employee_id → store_id
- employee_id → name
- employee_id → email
- employee_id → role
- employee_id → sales_per_week
- email → employee_id

Set Format:
{employee_id} → {store_id, name, email, role, sales_per_week}
{email} → {employee_id}

Each employee works at one store and is identified by employee_id. Since each email is unique, we also have email → employee_id. This shows that if we know the employee's email, we can find exactly one record for them. Closure of {employee_id} includes all attributes, which is why it's a verified key.

Table: INVENTORY
Primary Keys: store_id, product_id

Minimal Functional Dependencies

- (store_id, product_id) → quantity_available
- (store_id, product_id) → last_restocked
- (store_id, product_id) → category
- (store_id, product_id) → price

Set Format:
{store_id, product_id} → {quantity_available, last_restocked, category, price}

No single attribute by itself can uniquely identify a row, because one store can carry many products, and one product can appear in many stores. The closure of {store_id, product_id} determines all other attributes of the inventory entry, which is why it is a verified composite key. Every non-key attribute must depend on the entire key, not just part of it. That means the table is in 2NF, since there are no partial dependencies on store_id or product_id.

Table: ORDERS
Primary Key: order_id

Minimal Functional Dependencies

- order_id → customer_id
- order_id → store_id
- order_id → employee_id
- order_id → product_id
- order_id → order_type
- order_id → price

Set Format:
{order_id} → {customer_id, store_id, employee_id, product_id, order_type, price}

Each order is identified by its order_id, which determines all related details such as the customer, store, employee, and price. Closure of {order_id} includes all non-key attributes, making it the verified primary key.

Table: DISCOUNTS
Primary Key: discount_id

Minimal Functional Dependencies

- discount_id → vip_tier
- discount_id → start_date
- discount_id → end_date
- discount_id → discount_amount
- discount_id → order_id

Set Format:
{discount_id} → {vip_tier, start_date, end_date, discount_amount, order_id}

Each discount record is identified by discount_id, which determines all other attributes in the table. Because multiple discounts can apply to the same order_id, there is no FD from order_id to other attributes; only the primary key determines them.

Normalization to 3NF

STORE

Primary Key: store_id
Candidate Key: phone

Functional Dependencies:
store_id → address, opened_date, status, phone, store_hours

phone → store_id

1NF: Yes
2NF: Yes
3NF: Yes
STORE is 3NF because all non-key attributes depend directly on store_id.

CUSTOMER

Primary Key: customer_id
Candidate Key: email

Functional Dependencies:
customer_id → name, email, phone, date_joined, vip_tier

email → customer_id

1NF: Yes
2NF: Yes
3NF: Yes
CUSTOMER is in 3NF because each attribute depends only on customer_id; there are no transitive dependencies.

EMPLOYEE

Primary Key: employee_id
Candidate Key: email

Functional Dependencies:
employee_id → store_id, name, email, role, sales_per_week

email → employee_id

1NF: Yes
2NF: Yes
3NF: Yes
EMPLOYEE is in 3NF because all columns depend directly on employee_id; no partial or transitive dependencies.

INVENTORY

Primary Key: (store_id, product_id)

Functional Dependencies:
(store_id, product_id) → quantity_available, last_restocked, category, price

1NF: Yes
2NF: No — category and price depend only on product_id (product_id → category, price)
3NF: No — partial dependency violates 2NF/3NF
INVENTORY is not in 2NF because category and price describe the product, not the store; they depend only on product_id, when quantity_available and last_restocked depend on both store_id and product_id.

Decomposition:
PRODUCT(product_id, category, base_price)

INVENTORY(store_id, product_id, quantity_available, last_restocked, store_price)

ORDERS

Primary Key: order_id

Functional Dependencies:
order_id → customer_id, store_id, employee_id, product_id, order_type, price

1NF: Yes
2NF: Yes
3NF: No — transitive dependency (employee_id → store_id)
ORDERS is not in 3NF because employee_id determines store_id, as each employee belongs to one store. This makes store_id transitively dependent on the key, which is order_id, violating 3NF. Also, order_id includes only one product_id, limiting flexibility, which violates normalization.

Decomposition:
ORDERS(order_id, customer_id, employee_id, order_type, order_total)
ORDER_ITEM(order_id, product_id, unit_price, quantity, PRIMARY KEY(order_id, product_id))

DISCOUNTS

Primary Key: discount_id

Functional Dependencies:
discount_id → vip_tier, start_date, end_date, discount_amount, order_id

1NF: Yes
2NF: Yes
3NF: Yes

DISCOUNTS is 3NF because all columns depend on the primary key on discount_id, allowing discount_amount to be set manually per discount_id. There are no transitive dependencies, as every non-key attribute depends directly on the key, satisfying 3NF.

Normalization to BCNF

STORE
Keys: PK store_id; CK phone
FDs used: store_id → address, opened_date, status, phone, store_hours; phone → store_id.
BCNF check: Both determinants (store_id, phone) uniquely determine all attributes. This means all the determinants on the left side of every FD are superkeys. There are no transitive or partial dependencies. Since each non-trivial FD has a key on its left, STORE → BCNF.

CUSTOMER
Keys: PK customer_id; CK email
FDs used: customer_id → name, email, phone, date_joined, vip_tier; email → customer_id.
BCNF check: Determinants are keys (customer_id, email). In other words, all FDs have a superkey on the left, satisfying the BCNF conditions. CUSTOMER → BCNF.

EMPLOYEE
Keys: PK employee_id; CK email
FDs used: employee_id → store_id, name, email, role, sales_per_week; email → employee_id.
BCNF check: Both (employee_id, email) function as unique identifiers. The only FDs are from these keys to dependent attributes. All non-trivial FDs have a key determinant, which means EMPLOYEE → BCNF.

INVENTORY(Before Normalization):
(store_id, product_id) → quantity_available, last_restocked, category, price

- The Left-hand side is a primary key, that is non-trivial which doesn't violate BCNF.

(product_id) → category, price

- The left-hand side is not a key which violates BCNF. This means category and price depend only on the product itself, not on the store, creating redundancy.

<u>BCNF Algorithm:</u>

R1 = (X U Y) = (product_id, category, base_price)
R2 = (R - Y) = (store_id, product_id, quantity_available, last_restocked, store_price)

X = (product_id)
Y = (category, price)

We have to now create two tables:

PRODUCT(product_id,category,base_price)

- FD is product_id → category, base_price
- product_id is a key ⇒ BCNF

INVENTORY(store_id, product_id, quantity_available, last_restocked, store_price)

- FD is (store_id, product_id) → quantity_available, last_restocked, store_price
- Composite key ⇒ BCNF

INVENTORY (After Normalization)
Keys: PK (store_id, product_id)
FDs used: (store_id, product_id) → quantity_available, last_restocked, store_price.
BCNF check: The composite key (store_id, product_id) uniquely identifies each inventory record. All other attributes depend on this pair, not on a subset. Since this composite determinant is a key, all FDs meet the conditions, which means INVENTORY → BCNF

PRODUCT
Keys: PK product_id
FDs used: product_id → category, base_price.
BCNF check: product_id is the sole determinant and also the primary key that every non-key attribute depends on fully. No other FDs exist, therefore PRODUCT → BCNF.

ORDERS Before Normalization:
ORDERS(order_id, customer_id, store_id, employee_id, product_id, order_type, price)

order_id → customer_id, employee_id, order_type, price (Each order uniquely determines the customer, employee, type, and total price).
employee_id → store_id (Each employee works at exactly one store).
product_id → base_price (Each product has one base price (not per order)).

<u>**Bernstein Algorithm:**</u>

First, determine the FD for the relation R.
F = {
order_id → customer_id
order_id → employee_id
order_id → order_type
order_id → price
employee_id → store_id
product_id → base_price }
All left hand sides are single attributes, so F is already minimal. We break down the right-hand side and remove redundancies. Then we can create one relation per FD.

FD Group: order_id → {customer_id, employee_id, order_type, price}
Relation Created: R1(order_id, customer_id, employee_id, order_type, price)

FD Group: employee_id → store_id
Relation Created: R2(employee_id, store_id)

FD Group: product_id → base_price
Relation Created: R3(product_id, base_price)

Now, add the key relation. Since the primary key for orders is order_id, which already appears as a key in R1, so no extra key relation is required. Therefore, after applying the Bernstein algorithm, we get 3 relations:

ORDERS(order_id, customer_id, employee_id, order_type, price)
EMPLOYEE(employee_id, store_id)
PRODUCT(product_id, base_price)

ORDERS (After Normalization)
Keys: PK order_id
FDs used: order_id → customer_id, employee_id, order_type, (optional) order_total.
BCNF check: Determinant is the key → BCNF

ORDER_ITEM
Keys: PK (order_id, product_id)
FDs used: (order_id, product_id) → quantity, unit_price.
BCNF check: Determinant is the composite key → BCNF (unit_price is the sale snapshot, not a global product price).

DISCOUNTS
Keys: PK discount_id
FDs used: discount_id → vip_tier, start_date, end_date, discount_amount, order_id
BCNF check: Determinant is the key → BCNF.

Final Updated Schema Summary

STORE

STORE(store_id PK, address, opened_date, status, phone UNIQUE, store_hours)

CUSTOMER

CUSTOMER(customer_id PK, name, email UNIQUE, phone, date_joined, vip_tier)

EMPLOYEE

EMPLOYEE(employee_id PK, store_id FK → STORE.store_id, name, email UNIQUE, role, sales_per_week)

PRODUCT

PRODUCT(product_id PK, category, base_price)

INVENTORY

INVENTORY(store_id FK → STORE.store_id, product_id FK → PRODUCT.product_id, quantity_available, last_restocked, store_price, PRIMARY KEY (store_id, product_id))

ORDERS

ORDERS(order_id PK, customer_id FK → CUSTOMER.customer_id, employee_id FK → EMPLOYEE.employee_id, order_type, order_total)

ORDER_ITEM

ORDER_ITEM(order_id FK → ORDERS.order_id, product_id FK → PRODUCT.product_id, unit_price, quantity, PRIMARY KEY (order_id, product_id))

DISCOUNTS

DISCOUNTS(discount_id PK, vip_tier, start_date, end_date, discount_amount, order_id FK → ORDERS.order_id)

Relational Algebra for Query Descriptions

Q: Customers who placed IN_STORE/PICKUP but never ONLINE

C = customer

O = order

A = InStoreCustomerIDs = π_customer_id(σ_{order_type∈{IN_STORE, PICKUP}}(O))

B = OnlineCustomerIDs = π_customer_id(σ_{order_type='ONLINE'}(O))

πcustomer_id,name,vip_tier(Customer⋈(A−B))

Q: Combined directory: customers + employees

$C_{directory}=\pi_{name}$,email,type←′CUSTOMER′(Customer)

$E_{directory}=\pi_{name}$, email, type←′EMPLOYEE′(Employee)

$C_{directory} \uplus E_{directory}$

Q: Shows store revenue statistics

O′ = σorder_type∈{IN_STORE,PICKUP}(Orders)

J=(O′⋈O′.employee_id=E.employee_idEmployeeE)⋈E.store_id=S.store_id StoreS

R=γS.store_id;COUNT(DISTINCTE.employee_id)→Employees,SUM(O′.order_total)→Revenu

e,AVG(O′.order_total)→AvgOrder,STDDEV(O′.order_total)

σRevenue>0(R)

Q: Rank customers within VIP tier by total spend

S = γ_{customer_id, name, vip_tier ; SUM(order_total)→total_spend}
(Customer ⋈_{Customer.customer_id = Orders.customer_id} Orders)
Compare = S AS A ⋈_{A.vip_tier = B.vip_tier ∧ A.total_spend ≤ B.total_spend} S AS B
Ranked = γ_{A.customer_id, A.name, A.vip_tier, A.total_spend ;
COUNT(B.customer_id)→RankInTier}
(Compare)

TierSize = γ_{vip_tier ; COUNT(customer_id)→TierSize} (S)
Final = Ranked ⋈_{Ranked.vip_tier = TierSize.vip_tier} TierSize
Result = π_{customer_id, name, vip_tier, total_spend, TierSize, RankInTier}(Final)

Q: Employees whose average is greater than their store's average

O′=σorder_type∈{IN_STORE,PICKUP}(Orders)
J=O′⋈O′.employee_id=E.employee_idEmployeeE
EmpAvg=γE.store_id,E.employee_id,E.name;AVG(O′.order_total)→emp_avg(J)
StoreAvg=γE.store_id;AVG(O′.order_total)→store_avg(J)
πstore_id,employee_id,name,emp_avg,store_avg(σemp_avg≥store_avg(EmpAvg⋈EmpAvg.store_id=StoreAvg.store_idStoreAvg))

GUI Application

For the demonstration of the application component of the project, we created a Java-based GUI application. It connects to the TMU Oracle database using JDBC. This application allows users to manage a fully BCNF normalized store database. Through the GUI, users are able to drop and create the BCNF tables, populate tables with sample data, view all tables, search records using the primary key for a specific table, update any column value, delete any row, and run 5 advanced queries. The application is written in Java Swing and uses the ojdbc8 JDBC driver to connect to the Oracle hosted database. The application will be accessible through the attached zip file.

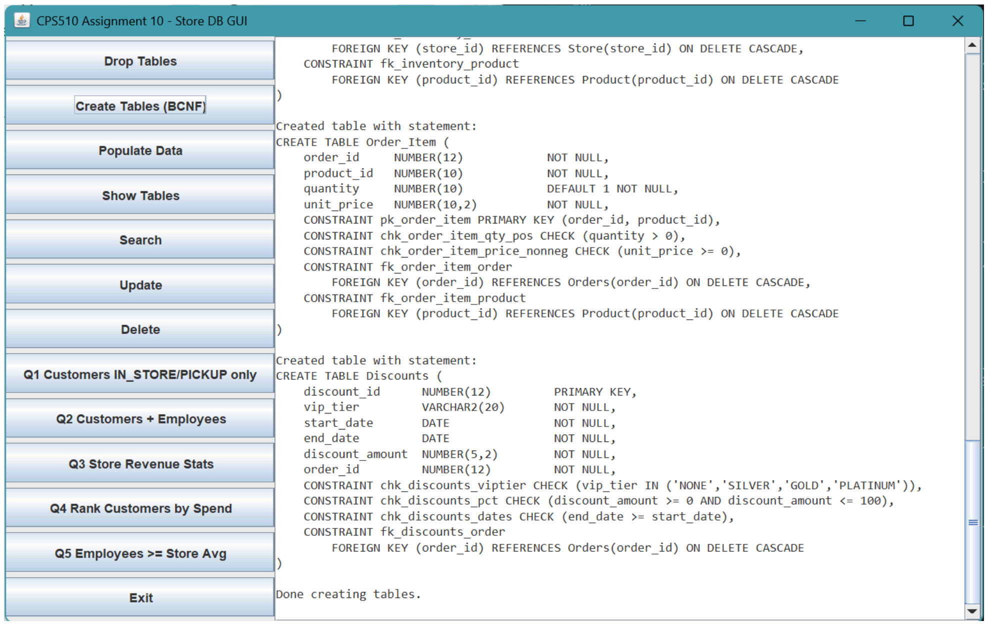
How to Open and Run the GUI:

1. Download Java, you must have JDK 21 or higher installed
2. Place the A9GUI.java and ojdbc8.jar files together in the same folder
3. Compile the program in powershell using this command:

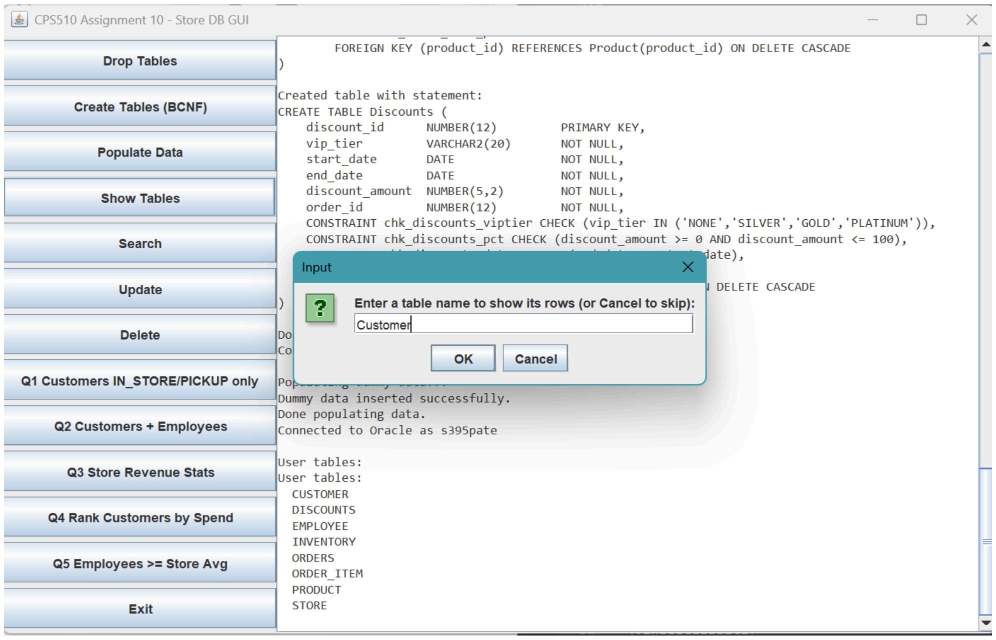
   ```
   javac -cp ".;ojdbc8.jar" A9GUI.java
   ```
4. Run the GUI: `java -cp ".;ojdbc8.jar" A9GUI`

Note: Before you compile and run, make sure you change the Oracle username and password to your own credentials

Screenshots of Application:



```
        FOREIGN KEY (store_id) REFERENCES Store(store_id) ON DELETE CASCADE,
    CONSTRAINT fk_inventory_product
        FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE CASCADE
)

Created table with statement:
CREATE TABLE Order_Item (
    order_id      NUMBER(12)          NOT NULL,
    product_id    NUMBER(10)          NOT NULL,
    quantity      NUMBER(10)          DEFAULT 1 NOT NULL,
    unit_price    NUMBER(10,2)        NOT NULL,
    CONSTRAINT pk_order_item PRIMARY KEY (order_id, product_id),
    CONSTRAINT chk_order_item_qty_pos CHECK (quantity > 0),
    CONSTRAINT chk_order_item_price_nonneg CHECK (unit_price >= 0),
    CONSTRAINT fk_order_item_order
        FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE,
    CONSTRAINT fk_order_item_product
        FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE CASCADE
)

Created table with statement:
CREATE TABLE Discounts (
    discount_id       NUMBER(12)         PRIMARY KEY,
    vip_tier          VARCHAR2(20)       NOT NULL,
    start_date        DATE               NOT NULL,
    end_date          DATE               NOT NULL,
    discount_amount   NUMBER(5,2)        NOT NULL,
    order_id          NUMBER(12)         NOT NULL,
    CONSTRAINT chk_discounts_viptier CHECK (vip_tier IN ('NONE','SILVER','GOLD','PLATINUM')),
    CONSTRAINT chk_discounts_pct CHECK (discount_amount >= 0 AND discount_amount <= 100),
    CONSTRAINT chk_discounts_dates CHECK (end_date >= start_date),
    CONSTRAINT fk_discounts_order
        FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE
)

Done creating tables.
```

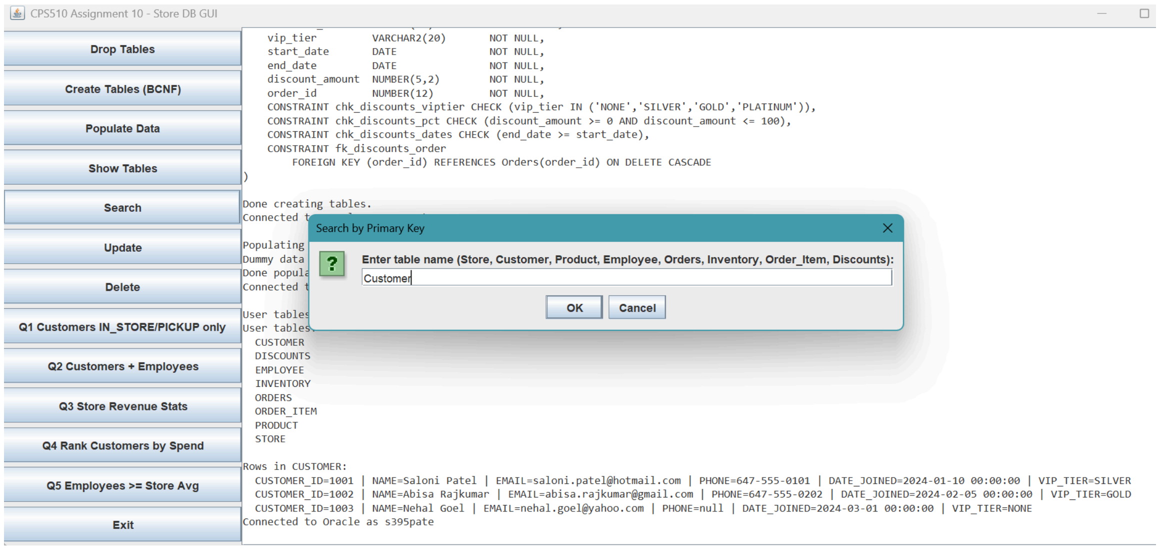


```
        FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE CASCADE
)

Created table with statement:
CREATE TABLE Discounts (
    discount_id       NUMBER(12)         PRIMARY KEY,
    vip_tier          VARCHAR2(20)       NOT NULL,
    start_date        DATE               NOT NULL,
    end_date          DATE               NOT NULL,
    discount_amount   NUMBER(5,2)        NOT NULL,
    order_id          NUMBER(12)         NOT NULL,
    CONSTRAINT chk_discounts_viptier CHECK (vip_tier IN ('NONE','SILVER','GOLD','PLATINUM')),
    CONSTRAINT chk_discounts_pct CHECK (discount_amount >= 0 AND discount_amount <= 100),
                                                              date),
                                                              DELETE CASCADE
)

Do
Co
Po
Dummy data inserted successfully.
Done populating data.
Connected to Oracle as s395pate

User tables:
User tables:
  CUSTOMER
  DISCOUNTS
  EMPLOYEE
  INVENTORY
  ORDERS
  ORDER_ITEM
  PRODUCT
  STORE
```

Enter a table name to show its rows (or Cancel to skip):

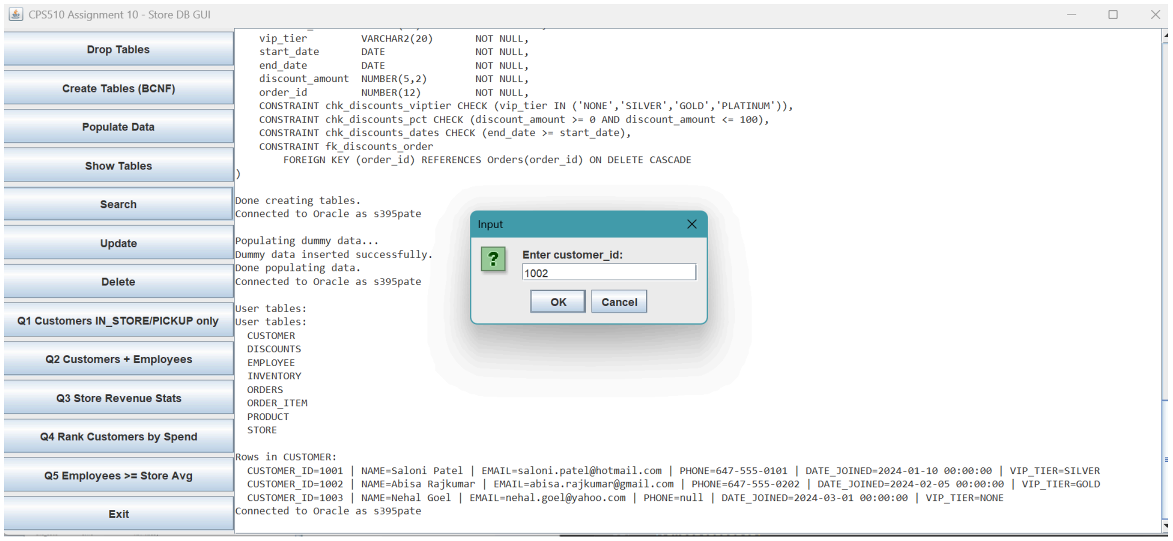
Customer

```
Connected to Oracle as s395pate

User tables:
User tables:
  CUSTOMER
  DISCOUNTS
  EMPLOYEE
  INVENTORY
  ORDERS
  ORDER_ITEM
  PRODUCT
  STORE

Rows in CUSTOMER:
  CUSTOMER_ID=1001 | NAME=Saloni Patel | EMAIL=saloni.patel@hotmail.com | PHONE=647-555-0101 | DATE_JOINED=2024-01-10 00:00:00 | VIP_TIER=SILVER
  CUSTOMER_ID=1002 | NAME=Abisa Rajkumar | EMAIL=abisa.rajkumar@gmail.com | PHONE=647-555-0202 | DATE_JOINED=2024-02-05 00:00:00 | VIP_TIER=GOLD
  CUSTOMER_ID=1003 | NAME=Nehal Goel | EMAIL=nehal.goel@yahoo.com | PHONE=null | DATE_JOINED=2024-03-01 00:00:00 | VIP_TIER=NONE
```



```
    vip_tier        VARCHAR2(20)       NOT NULL,
    start_date      DATE               NOT NULL,
    end_date        DATE               NOT NULL,
    discount_amount NUMBER(5,2)        NOT NULL,
    order_id        NUMBER(12)         NOT NULL,
    CONSTRAINT chk_discounts_viptier CHECK (vip_tier IN ('NONE','SILVER','GOLD','PLATINUM')),
    CONSTRAINT chk_discounts_pct CHECK (discount_amount >= 0 AND discount_amount <= 100),
    CONSTRAINT chk_discounts_dates CHECK (end_date >= start_date),
    CONSTRAINT fk_discounts_order
        FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE
)
Done creating tables.
Connected t
Populating
Dummy data
Done popula
Connected t
User tables
User tables
  CUSTOMER
  DISCOUNTS
  EMPLOYEE
  INVENTORY
  ORDERS
  ORDER_ITEM
  PRODUCT
  STORE

Rows in CUSTOMER:
  CUSTOMER_ID=1001 | NAME=Saloni Patel | EMAIL=saloni.patel@hotmail.com | PHONE=647-555-0101 | DATE_JOINED=2024-01-10 00:00:00 | VIP_TIER=SILVER
  CUSTOMER_ID=1002 | NAME=Abisa Rajkumar | EMAIL=abisa.rajkumar@gmail.com | PHONE=647-555-0202 | DATE_JOINED=2024-02-05 00:00:00 | VIP_TIER=GOLD
  CUSTOMER_ID=1003 | NAME=Nehal Goel | EMAIL=nehal.goel@yahoo.com | PHONE=null | DATE_JOINED=2024-03-01 00:00:00 | VIP_TIER=NONE
Connected to Oracle as s395pate
```

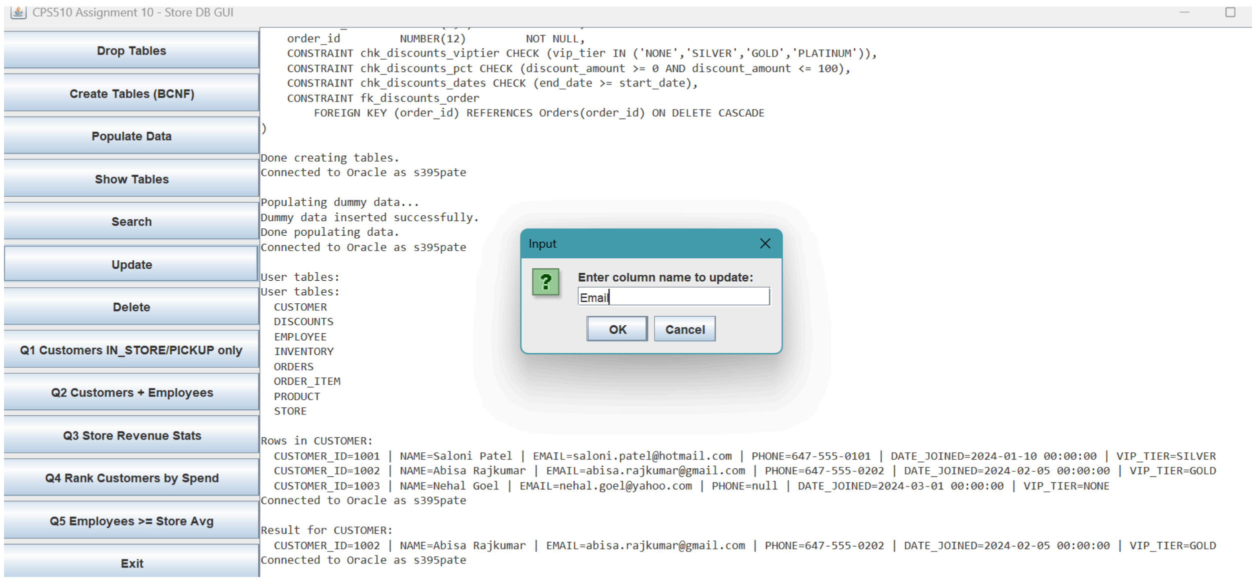
Enter table name (Store, Customer, Product, Employee, Orders, Inventory, Order_Item, Discounts): Customer



```
    vip_tier        VARCHAR2(20)       NOT NULL,
    start_date      DATE               NOT NULL,
    end_date        DATE               NOT NULL,
    discount_amount NUMBER(5,2)        NOT NULL,
    order_id        NUMBER(12)         NOT NULL,
    CONSTRAINT chk_discounts_viptier CHECK (vip_tier IN ('NONE','SILVER','GOLD','PLATINUM')),
    CONSTRAINT chk_discounts_pct CHECK (discount_amount >= 0 AND discount_amount <= 100),
    CONSTRAINT chk_discounts_dates CHECK (end_date >= start_date),
    CONSTRAINT fk_discounts_order
        FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE
)
Done creating tables.
Connected to Oracle as s395pate

Populating dummy data...
Dummy data inserted successfully.
Done populating data.
Connected to Oracle as s395pate

User tables:
User tables:
  CUSTOMER
  DISCOUNTS
  EMPLOYEE
  INVENTORY
  ORDERS
  ORDER_ITEM
  PRODUCT
  STORE

Rows in CUSTOMER:
  CUSTOMER_ID=1001 | NAME=Saloni Patel | EMAIL=saloni.patel@hotmail.com | PHONE=647-555-0101 | DATE_JOINED=2024-01-10 00:00:00 | VIP_TIER=SILVER
  CUSTOMER_ID=1002 | NAME=Abisa Rajkumar | EMAIL=abisa.rajkumar@gmail.com | PHONE=647-555-0202 | DATE_JOINED=2024-02-05 00:00:00 | VIP_TIER=GOLD
  CUSTOMER_ID=1003 | NAME=Nehal Goel | EMAIL=nehal.goel@yahoo.com | PHONE=null | DATE_JOINED=2024-03-01 00:00:00 | VIP_TIER=NONE
Connected to Oracle as s395pate
```

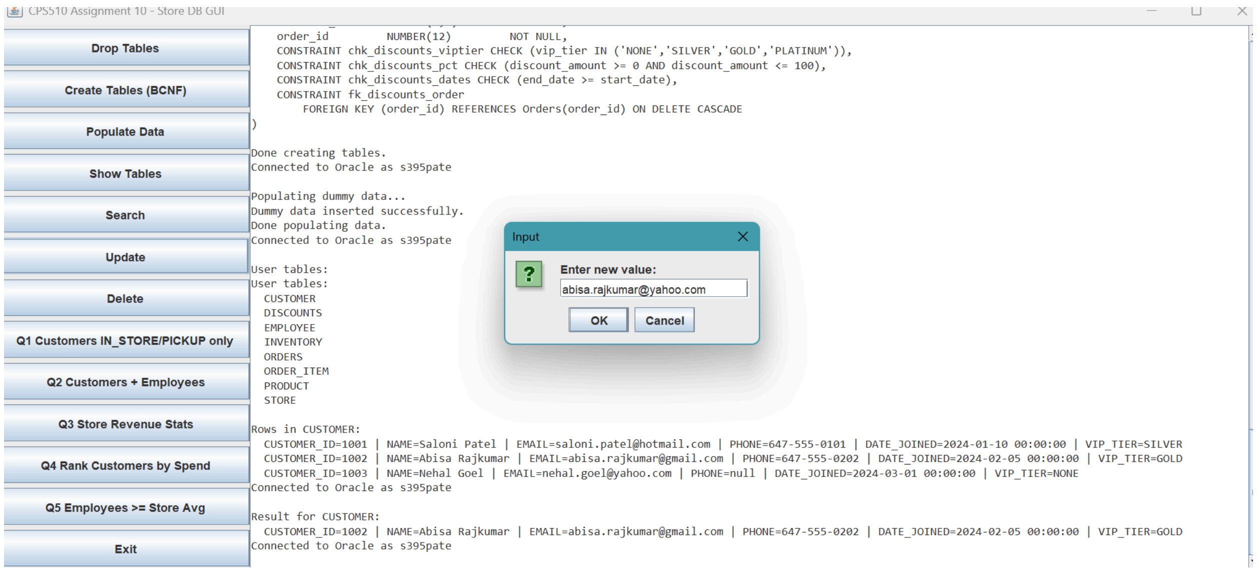
Enter customer_id: 1002

```
Result for CUSTOMER:
  CUSTOMER_ID=1002 | NAME=Abisa Rajkumar | EMAIL=abisa.rajkumar@gmail.com | PHONE=647-555-0202 | DATE_JOINED=2024-02-05 00:00:00 | VIP_TIER=GOLD
```



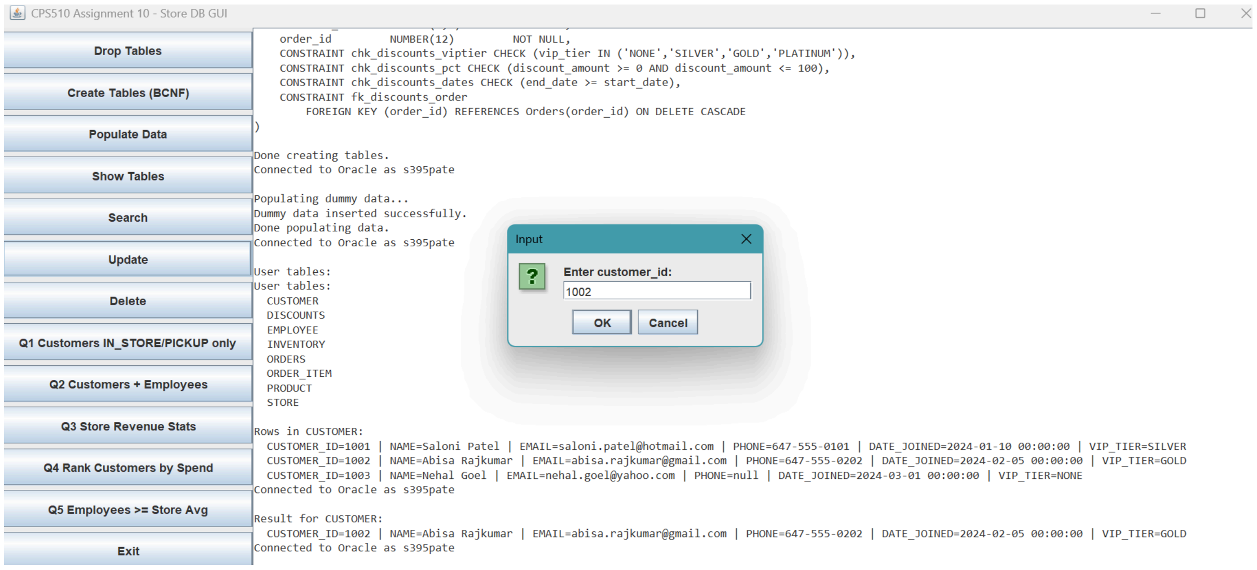
```
    order_id         NUMBER(12)         NOT NULL,
    CONSTRAINT chk_discounts_viptier CHECK (vip_tier IN ('NONE','SILVER','GOLD','PLATINUM')),
    CONSTRAINT chk_discounts_pct CHECK (discount_amount >= 0 AND discount_amount <= 100),
    CONSTRAINT chk_discounts_dates CHECK (end_date >= start_date),
    CONSTRAINT fk_discounts_order
        FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE
)

Done creating tables.
Connected to Oracle as s395pate

Populating dummy data...
Dummy data inserted successfully.
Done populating data.
Connected to Oracle as s395pate

User tables:
User tables:
  CUSTOMER
  DISCOUNTS
  EMPLOYEE
  INVENTORY
  ORDERS
  ORDER_ITEM
  PRODUCT
  STORE

Rows in CUSTOMER:
  CUSTOMER_ID=1001 | NAME=Saloni Patel | EMAIL=saloni.patel@hotmail.com | PHONE=647-555-0101 | DATE_JOINED=2024-01-10 00:00:00 | VIP_TIER=SILVER
  CUSTOMER_ID=1002 | NAME=Abisa Rajkumar | EMAIL=abisa.rajkumar@gmail.com | PHONE=647-555-0202 | DATE_JOINED=2024-02-05 00:00:00 | VIP_TIER=GOLD
  CUSTOMER_ID=1003 | NAME=Nehal Goel | EMAIL=nehal.goel@yahoo.com | PHONE=null | DATE_JOINED=2024-03-01 00:00:00 | VIP_TIER=NONE
Connected to Oracle as s395pate

Result for CUSTOMER:
  CUSTOMER_ID=1002 | NAME=Abisa Rajkumar | EMAIL=abisa.rajkumar@gmail.com | PHONE=647-555-0202 | DATE_JOINED=2024-02-05 00:00:00 | VIP_TIER=GOLD
Connected to Oracle as s395pate
```

Enter column name to update: Email



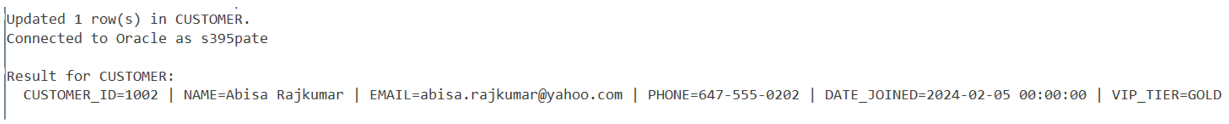
```
    order_id         NUMBER(12)         NOT NULL,
    CONSTRAINT chk_discounts_viptier CHECK (vip_tier IN ('NONE','SILVER','GOLD','PLATINUM')),
    CONSTRAINT chk_discounts_pct CHECK (discount_amount >= 0 AND discount_amount <= 100),
    CONSTRAINT chk_discounts_dates CHECK (end_date >= start_date),
    CONSTRAINT fk_discounts_order
        FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE
)

Done creating tables.
Connected to Oracle as s395pate

Populating dummy data...
Dummy data inserted successfully.
Done populating data.
Connected to Oracle as s395pate

User tables:
User tables:
  CUSTOMER
  DISCOUNTS
  EMPLOYEE
  INVENTORY
  ORDERS
  ORDER_ITEM
  PRODUCT
  STORE

Rows in CUSTOMER:
  CUSTOMER_ID=1001 | NAME=Saloni Patel | EMAIL=saloni.patel@hotmail.com | PHONE=647-555-0101 | DATE_JOINED=2024-01-10 00:00:00 | VIP_TIER=SILVER
  CUSTOMER_ID=1002 | NAME=Abisa Rajkumar | EMAIL=abisa.rajkumar@gmail.com | PHONE=647-555-0202 | DATE_JOINED=2024-02-05 00:00:00 | VIP_TIER=GOLD
  CUSTOMER_ID=1003 | NAME=Nehal Goel | EMAIL=nehal.goel@yahoo.com | PHONE=null | DATE_JOINED=2024-03-01 00:00:00 | VIP_TIER=NONE
Connected to Oracle as s395pate

Result for CUSTOMER:
  CUSTOMER_ID=1002 | NAME=Abisa Rajkumar | EMAIL=abisa.rajkumar@gmail.com | PHONE=647-555-0202 | DATE_JOINED=2024-02-05 00:00:00 | VIP_TIER=GOLD
Connected to Oracle as s395pate
```

Enter new value: abisa.rajkumar@yahoo.com

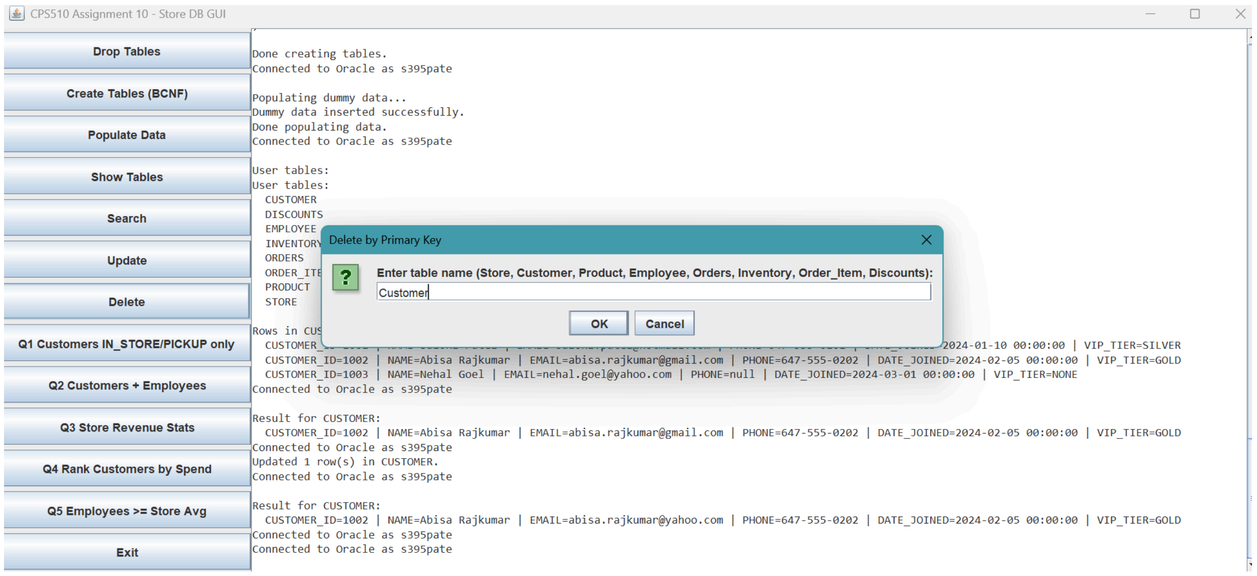


```
    order_id          NUMBER(12)          NOT NULL,
    CONSTRAINT chk_discounts_viptier CHECK (vip_tier IN ('NONE','SILVER','GOLD','PLATINUM')),
    CONSTRAINT chk_discounts_pct CHECK (discount_amount >= 0 AND discount_amount <= 100),
    CONSTRAINT chk_discounts_dates CHECK (end_date >= start_date),
    CONSTRAINT fk_discounts_order
        FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE
)

Done creating tables.
Connected to Oracle as s395pate

Populating dummy data...
Dummy data inserted successfully.
Done populating data.
Connected to Oracle as s395pate

User tables:
User tables:
  CUSTOMER
  DISCOUNTS
  EMPLOYEE
  INVENTORY
  ORDERS
  ORDER_ITEM
  PRODUCT
  STORE

Rows in CUSTOMER:
  CUSTOMER_ID=1001 | NAME=Saloni Patel | EMAIL=saloni.patel@hotmail.com | PHONE=647-555-0101 | DATE_JOINED=2024-01-10 00:00:00 | VIP_TIER=SILVER
  CUSTOMER_ID=1002 | NAME=Abisa Rajkumar | EMAIL=abisa.rajkumar@gmail.com | PHONE=647-555-0202 | DATE_JOINED=2024-02-05 00:00:00 | VIP_TIER=GOLD
  CUSTOMER_ID=1003 | NAME=Nehal Goel | EMAIL=nehal.goel@yahoo.com | PHONE=null | DATE_JOINED=2024-03-01 00:00:00 | VIP_TIER=NONE
Connected to Oracle as s395pate

Result for CUSTOMER:
  CUSTOMER_ID=1002 | NAME=Abisa Rajkumar | EMAIL=abisa.rajkumar@gmail.com | PHONE=647-555-0202 | DATE_JOINED=2024-02-05 00:00:00 | VIP_TIER=GOLD
Connected to Oracle as s395pate
```

Enter customer_id: 1002

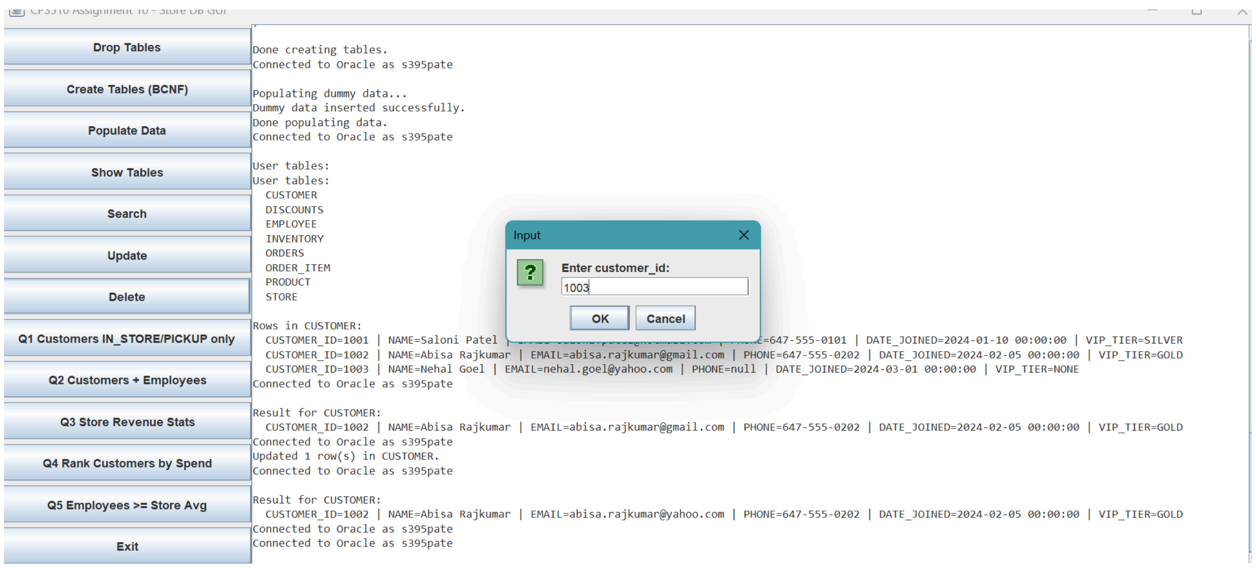


```
Updated 1 row(s) in CUSTOMER.
Connected to Oracle as s395pate

Result for CUSTOMER:
  CUSTOMER_ID=1002 | NAME=Abisa Rajkumar | EMAIL=abisa.rajkumar@yahoo.com | PHONE=647-555-0202 | DATE_JOINED=2024-02-05 00:00:00 | VIP_TIER=GOLD
```

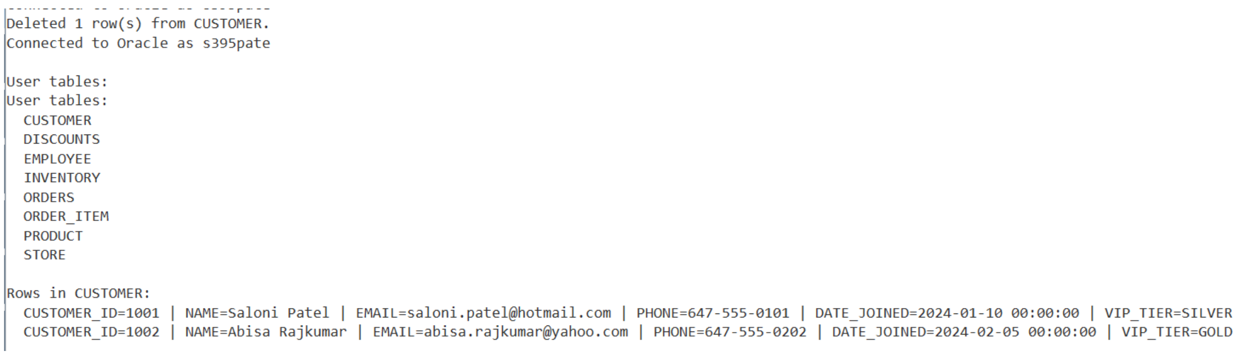


```
Done creating tables.
Connected to Oracle as s395pate

Populating dummy data...
Dummy data inserted successfully.
Done populating data.
Connected to Oracle as s395pate

User tables:
User tables:
  CUSTOMER
  DISCOUNTS
  EMPLOYEE
  INVENTORY
  ORDERS
  ORDER_ITE
  PRODUCT
  STORE

Rows in CUS
  CUSTOMER_                                                                      2024-01-10 00:00:00 | VIP_TIER=SILVER
  CUSTOMER_ID=1002 | NAME=Abisa Rajkumar | EMAIL=abisa.rajkumar@gmail.com | PHONE=647-555-0202 | DATE_JOINED=2024-02-05 00:00:00 | VIP_TIER=GOLD
  CUSTOMER_ID=1003 | NAME=Nehal Goel | EMAIL=nehal.goel@yahoo.com | PHONE=null | DATE_JOINED=2024-03-01 00:00:00 | VIP_TIER=NONE
Connected to Oracle as s395pate

Result for CUSTOMER:
  CUSTOMER_ID=1002 | NAME=Abisa Rajkumar | EMAIL=abisa.rajkumar@gmail.com | PHONE=647-555-0202 | DATE_JOINED=2024-02-05 00:00:00 | VIP_TIER=GOLD
Connected to Oracle as s395pate
Updated 1 row(s) in CUSTOMER.
Connected to Oracle as s395pate

Result for CUSTOMER:
  CUSTOMER_ID=1002 | NAME=Abisa Rajkumar | EMAIL=abisa.rajkumar@yahoo.com | PHONE=647-555-0202 | DATE_JOINED=2024-02-05 00:00:00 | VIP_TIER=GOLD
Connected to Oracle as s395pate
Connected to Oracle as s395pate
```

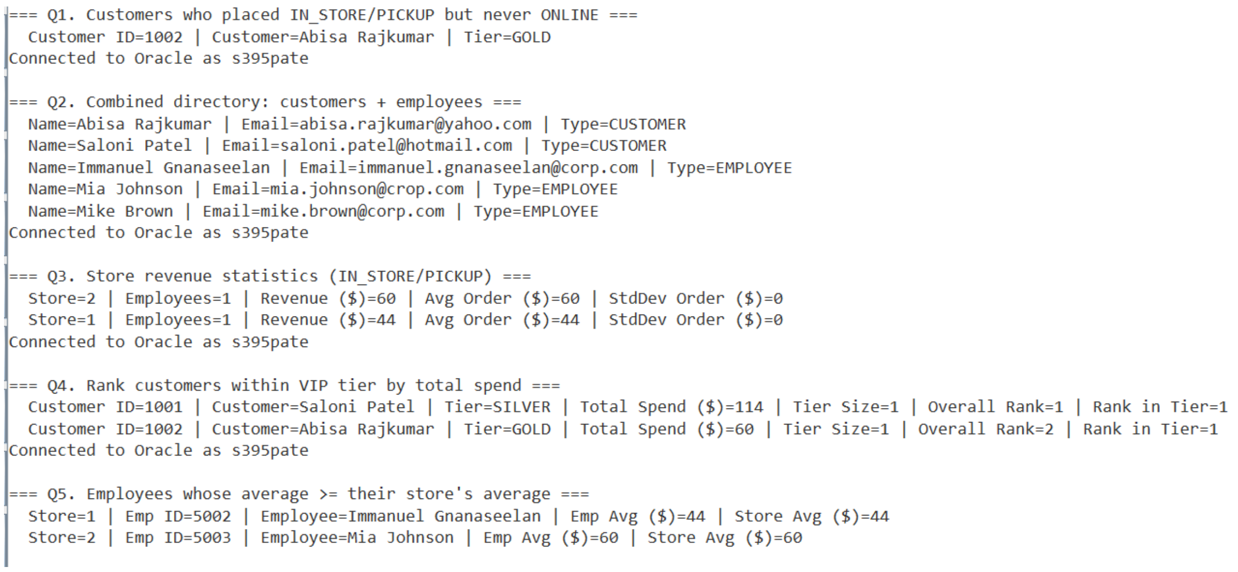
Enter table name (Store, Customer, Product, Employee, Orders, Inventory, Order_Item, Discounts): Customer



```
Done creating tables.
Connected to Oracle as s395pate

Populating dummy data...
Dummy data inserted successfully.
Done populating data.
Connected to Oracle as s395pate

User tables:
User tables:
  CUSTOMER
  DISCOUNTS
  EMPLOYEE
  INVENTORY
  ORDERS
  ORDER_ITEM
  PRODUCT
  STORE

Rows in CUSTOMER:
  CUSTOMER_ID=1001 | NAME=Saloni Patel | [illegible]E=647-555-0101 | DATE_JOINED=2024-01-10 00:00:00 | VIP_TIER=SILVER
  CUSTOMER_ID=1002 | NAME=Abisa Rajkumar | EMAIL=abisa.rajkumar@gmail.com | PHONE=647-555-0202 | DATE_JOINED=2024-02-05 00:00:00 | VIP_TIER=GOLD
  CUSTOMER_ID=1003 | NAME=Nehal Goel | EMAIL=nehal.goel@yahoo.com | PHONE=null | DATE_JOINED=2024-03-01 00:00:00 | VIP_TIER=NONE
Connected to Oracle as s395pate

Result for CUSTOMER:
  CUSTOMER_ID=1002 | NAME=Abisa Rajkumar | EMAIL=abisa.rajkumar@gmail.com | PHONE=647-555-0202 | DATE_JOINED=2024-02-05 00:00:00 | VIP_TIER=GOLD
Connected to Oracle as s395pate
Updated 1 row(s) in CUSTOMER.
Connected to Oracle as s395pate

Result for CUSTOMER:
  CUSTOMER_ID=1002 | NAME=Abisa Rajkumar | EMAIL=abisa.rajkumar@yahoo.com | PHONE=647-555-0202 | DATE_JOINED=2024-02-05 00:00:00 | VIP_TIER=GOLD
Connected to Oracle as s395pate
Connected to Oracle as s395pate
```



```
Deleted 1 row(s) from CUSTOMER.
Connected to Oracle as s395pate

User tables:
User tables:
  CUSTOMER
  DISCOUNTS
  EMPLOYEE
  INVENTORY
  ORDERS
  ORDER_ITEM
  PRODUCT
  STORE

Rows in CUSTOMER:
  CUSTOMER_ID=1001 | NAME=Saloni Patel | EMAIL=saloni.patel@hotmail.com | PHONE=647-555-0101 | DATE_JOINED=2024-01-10 00:00:00 | VIP_TIER=SILVER
  CUSTOMER_ID=1002 | NAME=Abisa Rajkumar | EMAIL=abisa.rajkumar@yahoo.com | PHONE=647-555-0202 | DATE_JOINED=2024-02-05 00:00:00 | VIP_TIER=GOLD
```



```
=== Q1. Customers who placed IN_STORE/PICKUP but never ONLINE ===
  Customer ID=1002 | Customer=Abisa Rajkumar | Tier=GOLD
Connected to Oracle as s395pate

=== Q2. Combined directory: customers + employees ===
  Name=Abisa Rajkumar | Email=abisa.rajkumar@yahoo.com | Type=CUSTOMER
  Name=Saloni Patel | Email=saloni.patel@hotmail.com | Type=CUSTOMER
  Name=Immanuel Gnanaseelan | Email=immanuel.gnanaseelan@corp.com | Type=EMPLOYEE
  Name=Mia Johnson | Email=mia.johnson@crop.com | Type=EMPLOYEE
  Name=Mike Brown | Email=mike.brown@corp.com | Type=EMPLOYEE
Connected to Oracle as s395pate

=== Q3. Store revenue statistics (IN_STORE/PICKUP) ===
  Store=2 | Employees=1 | Revenue ($)=60 | Avg Order ($)=60 | StdDev Order ($)=0
  Store=1 | Employees=1 | Revenue ($)=44 | Avg Order ($)=44 | StdDev Order ($)=0
Connected to Oracle as s395pate

=== Q4. Rank customers within VIP tier by total spend ===
  Customer ID=1001 | Customer=Saloni Patel | Tier=SILVER | Total Spend ($)=114 | Tier Size=1 | Overall Rank=1 | Rank in Tier=1
  Customer ID=1002 | Customer=Abisa Rajkumar | Tier=GOLD | Total Spend ($)=60 | Tier Size=1 | Overall Rank=2 | Rank in Tier=1
Connected to Oracle as s395pate

=== Q5. Employees whose average >= their store's average ===
  Store=1 | Emp ID=5002 | Employee=Immanuel Gnanaseelan | Emp Avg ($)=44 | Store Avg ($)=44
  Store=2 | Emp ID=5003 | Employee=Mia Johnson | Emp Avg ($)=60 | Store Avg ($)=60
```

Code:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.sql.*;

public class A9GUI extends JFrame {
  // change the DB_USER and DB_PASSWORD
  private static final String DB_URL = "jdbc:oracle:thin:@oracle.cs.torontomu.ca:1521:orcl";
  private static final String DB_USER = "s395pate";
  private static final String DB_PASSWORD = "03096930";

  // Advanced Queries (Q1–Q5)
  private static final String SQL_Q1 = """
        SELECT c.customer_id AS "Customer ID",
              c.name        AS "Customer",
              c.vip_tier    AS "Tier"
        FROM   Customer c
        WHERE  EXISTS (
                SELECT 1 FROM Orders o
                WHERE  o.customer_id = c.customer_id
                AND    o.order_type IN ('IN_STORE','PICKUP')
              )
        AND    NOT EXISTS (
                SELECT 1 FROM Orders o2
                WHERE  o2.customer_id = c.customer_id
                AND    o2.order_type = 'ONLINE'
              )
        ORDER BY "Customer"
        """;

  private static final String SQL_Q2 = """
        SELECT
          RTRIM(name)  AS "Name",
          RTRIM(email) AS "Email",
          'CUSTOMER'   AS "Type"
        FROM Customer
        UNION ALL
        SELECT
          RTRIM(name)  AS "Name",
          RTRIM(email) AS "Email",
          'EMPLOYEE'   AS "Type"
        FROM Employee
        ORDER BY
          "Type", "Name"
        """;

  private static final String SQL_Q3 = """
```

```
        SELECT s.store_id                    AS "Store",
               COUNT(DISTINCT e.employee_id) AS "Employees",
               ROUND(SUM(o.order_total),2)   AS "Revenue ($)",
               ROUND(AVG(o.order_total),2)   AS "Avg Order ($)",
               ROUND(STDDEV(o.order_total),2) AS "StdDev Order ($)"
        FROM   Orders   o
        JOIN   Employee e ON e.employee_id = o.employee_id
        JOIN   Store    s ON s.store_id    = e.store_id
        WHERE  o.order_type IN ('IN_STORE','PICKUP')
        GROUP  BY s.store_id
        HAVING SUM(o.order_total) > 0
        ORDER  BY "Revenue ($)" DESC, "Store"
        """;

    private static final String SQL_Q4 = """
        WITH spends AS (
          SELECT
             c.customer_id,
             c.name                 AS customer_name,
             NVL(c.vip_tier,'NONE') AS vip_tier,
             NVL(SUM(o.order_total),0) AS total_spend
          FROM   Customer c
          LEFT JOIN Orders o
                ON o.customer_id = c.customer_id
          GROUP  BY c.customer_id, c.name, c.vip_tier
        )
        SELECT
           customer_id                        AS "Customer ID",
           customer_name                      AS "Customer",
           vip_tier                           AS "Tier",
           ROUND(total_spend,2)               AS "Total Spend ($)",
           COUNT(*) OVER (PARTITION BY vip_tier)        AS "Tier Size",
           DENSE_RANK() OVER (ORDER BY total_spend DESC) AS "Overall Rank",
           DENSE_RANK() OVER (PARTITION BY vip_tier
                              ORDER BY total_spend DESC) AS "Rank in Tier"
        FROM spends
        ORDER BY "Overall Rank", "Tier", "Rank in Tier", "Customer"
        """;

    private static final String SQL_Q5 = """
        SELECT s.store_id AS "Store",
               e.employee_id AS "Emp ID",
               e.name AS "Employee",
               ROUND(AVG(o.order_total),2) AS "Emp Avg ($)",
               ROUND( (SELECT AVG(o2.order_total)
                       FROM Orders o2
                       JOIN Employee e2 ON e2.employee_id = o2.employee_id
                       WHERE e2.store_id = s.store_id
                         AND o2.order_type IN ('IN_STORE','PICKUP')), 2) AS "Store Avg ($)"
        FROM   Orders   o
        JOIN   Employee e ON e.employee_id = o.employee_id
        JOIN   Store    s ON s.store_id    = e.store_id
        WHERE  o.order_type IN ('IN_STORE','PICKUP')
        GROUP  BY s.store_id, e.employee_id, e.name
        HAVING AVG(o.order_total) >= (SELECT AVG(o2.order_total)
                               FROM Orders o2
                               JOIN Employee e2 ON e2.employee_id = o2.employee_id
                               WHERE e2.store_id = s.store_id
                                 AND o2.order_type IN ('IN_STORE','PICKUP'))
        ORDER  BY "Store","Emp Avg ($)" DESC,"Employee"
        """;

    private final JTextArea outputArea;

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            A9GUI gui = new A9GUI();
            gui.setVisible(true);
        });
    }

    public A9GUI() {
        setTitle("CPS510 Assignment 10 - Store DB GUI");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(950, 600);
        setLocationRelativeTo(null);

        setLayout(new BorderLayout());

        // Left panel with buttons
        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new GridLayout(0, 1, 5, 5));

        JButton btnDrop = new JButton("Drop Tables");
        JButton btnCreate = new JButton("Create Tables (BCNF)");
        JButton btnPopulate = new JButton("Populate Data");
        JButton btnShowTbls = new JButton("Show Tables");
        JButton btnSearchPK = new JButton("Search");
        JButton btnUpdatePK = new JButton("Update");
        JButton btnDeletePK = new JButton("Delete");

        JButton btnQ1 = new JButton("Q1 Customers IN_STORE/PICKUP only");
        JButton btnQ2 = new JButton("Q2 Customers + Employees");
        JButton btnQ3 = new JButton("Q3 Store Revenue Stats");
        JButton btnQ4 = new JButton("Q4 Rank Customers by Spend");
        JButton btnQ5 = new JButton("Q5 Employees >= Store Avg");

        JButton btnExit = new JButton("Exit");
        buttonPanel.add(btnDrop);
        buttonPanel.add(btnCreate);
        buttonPanel.add(btnPopulate);
        buttonPanel.add(btnShowTbls);
        buttonPanel.add(btnSearchPK);
        buttonPanel.add(btnUpdatePK);
        buttonPanel.add(btnDeletePK);
        buttonPanel.add(btnQ1);
        buttonPanel.add(btnQ2);
        buttonPanel.add(btnQ3);
        buttonPanel.add(btnQ4);
        buttonPanel.add(btnQ5);
        buttonPanel.add(btnExit);

        outputArea = new JTextArea();
        outputArea.setEditable(false);
        outputArea.setFont(new Font("Consolas", Font.PLAIN, 12));
        JScrollPane scrollPane = new JScrollPane(outputArea);

        add(buttonPanel, BorderLayout.WEST);
        add(scrollPane, BorderLayout.CENTER);

        // Hook up actions
        btnDrop.addActionListener(this::handleDropTables);
        btnCreate.addActionListener(this::handleCreateTables);
        btnPopulate.addActionListener(this::handlePopulate);
        btnShowTbls.addActionListener(this::handleShowTables);
        btnSearchPK.addActionListener(this::handleSearchByPK);
        btnUpdatePK.addActionListener(this::handleUpdateByPK);
        btnDeletePK.addActionListener(this::handleDeleteByPK);
        btnQ1.addActionListener(e -> withConnection(
                conn -> runQuery(conn, "Q1. Customers who placed IN_STORE/PICKUP but never ONLINE",
SQL_Q1)));
        btnQ2.addActionListener(
                e -> withConnection(conn -> runQuery(conn, "Q2. Combined directory: customers + employees",
SQL_Q2)));
        btnQ3.addActionListener(
                e -> withConnection(conn -> runQuery(conn, "Q3. Store revenue statistics (IN_STORE/PICKUP)",
SQL_Q3)));
        btnQ4.addActionListener(e -> withConnection(
                conn -> runQuery(conn, "Q4. Rank customers within VIP tier by total spend", SQL_Q4)));
        btnQ5.addActionListener(e -> withConnection(
                conn -> runQuery(conn, "Q5. Employees whose average >= their store's average", SQL_Q5)));
        btnExit.addActionListener(e -> System.exit(0));
    }

    // Connection helper
    private void withConnection(DBAction action) {
        try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
            appendLine("Connected to Oracle as " + DB_USER);
            action.run(conn);
        } catch (SQLException ex) {
            appendLine("Database error: " + ex.getMessage());
        }
    }

    private interface DBAction {
        void run(Connection conn) throws SQLException;
    }

    // Button handlers
    private void handleDropTables(ActionEvent e) {
        withConnection(conn -> {
            appendLine("\nDropping tables...");
            dropTables(conn);
            appendLine("Done dropping tables.");
        });
    }

    private void handleCreateTables(ActionEvent e) {
        withConnection(conn -> {
            appendLine("\nCreating BCNF tables...");
            createTables(conn);
            appendLine("Done creating tables.");
        });
    }

    private void handlePopulate(ActionEvent e) {
        withConnection(conn -> {
            appendLine("\nPopulating dummy data...");
            populateTables(conn);
            appendLine("Done populating data.");
        });
    }

    private void handleShowTables(ActionEvent e) {
        withConnection(conn -> {
            appendLine("\nUser tables:");
            showTables(conn);
        });
    }

    private void handleSearchByPK(ActionEvent e) {
        withConnection(conn -> {
            String table = JOptionPane.showInputDialog(this,
                    "Enter table name (Store, Customer, Product, Employee, Orders, Inventory, Order_Item, Discounts):",
                    "Search by Primary Key",
                    JOptionPane.QUESTION_MESSAGE);
            if (table == null || table.isBlank())
                return;

            searchByPrimaryKey(conn, table.trim());
        });
    }

    private void handleUpdateByPK(ActionEvent e) {
        withConnection(conn -> {
            String table = JOptionPane.showInputDialog(this,
                    "Enter table name (Store, Customer, Product, Employee, Orders, Inventory, Order_Item, Discounts):",
                    "Update by Primary Key",
                    JOptionPane.QUESTION_MESSAGE);
            if (table == null || table.isBlank())
                return;

            updateByPrimaryKey(conn, table.trim());
        });
    }

    private void handleDeleteByPK(ActionEvent e) {
        withConnection(conn -> {
            String table = JOptionPane.showInputDialog(this,
                    "Enter table name (Store, Customer, Product, Employee, Orders, Inventory, Order_Item, Discounts):",
                    "Delete by Primary Key",
                    JOptionPane.QUESTION_MESSAGE);
            if (table == null || table.isBlank())
                return;

            deleteByPrimaryKey(conn, table.trim());
        });
    }

    // Drop all tables
    private void dropTables(Connection conn) throws SQLException {
        String[] dropStatements = {
                "DROP TABLE Discounts CASCADE CONSTRAINTS",
                "DROP TABLE Order_Item CASCADE CONSTRAINTS",
                "DROP TABLE Inventory CASCADE CONSTRAINTS",
                "DROP TABLE Orders CASCADE CONSTRAINTS",
                "DROP TABLE Employee CASCADE CONSTRAINTS",
                "DROP TABLE Product CASCADE CONSTRAINTS",
                "DROP TABLE Customer CASCADE CONSTRAINTS",
                "DROP TABLE Store CASCADE CONSTRAINTS"
        };

        for (String sql : dropStatements) {
            try (Statement st = conn.createStatement()) {
                st.executeUpdate(sql);
                appendLine("Executed: " + sql);
            } catch (SQLException ex) {
                if (!ex.getMessage().toUpperCase().contains("ORA-00942")) {
                    appendLine("Error executing: " + sql);
                    appendLine("  -> " + ex.getMessage());
                }
            }
        }
    }

    // Create BCNF tables
    private void createTables(Connection conn) throws SQLException {
        String createStore = """
                CREATE TABLE Store (
                    store_id       NUMBER              PRIMARY KEY,
                    address        VARCHAR2(200)       NOT NULL,
                    opened_date    DATE                DEFAULT SYSDATE NOT NULL,
                    status         VARCHAR2(20)        DEFAULT 'OPEN' NOT NULL,
                    phone          VARCHAR2(20),
                    store_hours    VARCHAR2(100),
                    CONSTRAINT uq_store_phone UNIQUE (phone),
                    CONSTRAINT chk_store_status CHECK (status IN ('OPEN', 'CLOSED', 'RENOVATION'))
                )
                """;

        String createCustomer = """
                CREATE TABLE Customer (
                    customer_id    NUMBER(10)          PRIMARY KEY,
                    name           VARCHAR2(100)       NOT NULL,
                    email          VARCHAR2(200)       NOT NULL,
                    phone          VARCHAR2(20),
                    date_joined    DATE                DEFAULT SYSDATE NOT NULL,
                    vip_tier       VARCHAR2(20)        DEFAULT 'NONE' NOT NULL,
                    CONSTRAINT uq_customer_email UNIQUE (email),
                    CONSTRAINT chk_customer_viptier CHECK (vip_tier IN ('NONE', 'SILVER', 'GOLD',
'PLATINUM'))
                )
                """;

        String createProduct = """
                CREATE TABLE Product (
                    product_id     NUMBER(10)          PRIMARY KEY,
                    category       VARCHAR2(50)        NOT NULL,
                    base_price     NUMBER(10,2)        NOT NULL,
                    CONSTRAINT chk_product_price_nonneg CHECK (base_price >= 0)
                )
                """;
```

```
String createEmployee = """
    CREATE TABLE Employee (
      employee_id    NUMBER              PRIMARY KEY,
      store_id       NUMBER(10)          NOT NULL,
      name           VARCHAR2(100)       NOT NULL,
      email          VARCHAR2(100)       UNIQUE,
      role           VARCHAR2(50),
      sales_per_week NUMBER(10,2),
      CONSTRAINT fk_employee_store
        FOREIGN KEY (store_id)
        REFERENCES Store(store_id)
        ON DELETE CASCADE
    )
    """;

String createOrders = """
    CREATE TABLE Orders (
      order_id       NUMBER(12)          PRIMARY KEY,
      customer_id    NUMBER(10)          NOT NULL,
      employee_id    NUMBER              NOT NULL,
      order_type     VARCHAR2(20)        DEFAULT 'ONLINE' NOT NULL,
      order_total    NUMBER(10,2)        NOT NULL,
      order_date     DATE                DEFAULT SYSDATE NOT NULL,
      CONSTRAINT chk_orders_type CHECK (order_type IN ('ONLINE', 'IN_STORE', 'PICKUP')),
      CONSTRAINT chk_orders_total_nonneg CHECK (order_total >= 0),
      CONSTRAINT fk_orders_customer
        FOREIGN KEY (customer_id) REFERENCES Customer(customer_id) ON DELETE CASCADE,
      CONSTRAINT fk_orders_employee
        FOREIGN KEY (employee_id) REFERENCES Employee(employee_id) ON DELETE CASCADE
    )
    """;

String createInventory = """
    CREATE TABLE Inventory (
      store_id           NUMBER(10)      NOT NULL,
      product_id         NUMBER(10)      NOT NULL,
      quantity_available  NUMBER(10)     DEFAULT 0 NOT NULL,
      last_restocked      DATE,
      store_price        NUMBER(10,2)    NOT NULL,
      CONSTRAINT pk_inventory PRIMARY KEY (store_id, product_id),
      CONSTRAINT chk_inventory_qty_nonneg CHECK (quantity_available >= 0),
      CONSTRAINT chk_inventory_price_nonneg CHECK (store_price >= 0),
      CONSTRAINT fk_inventory_store
        FOREIGN KEY (store_id) REFERENCES Store(store_id) ON DELETE CASCADE,
      CONSTRAINT fk_inventory_product
        FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE CASCADE
    )
    """;
```

```
String createOrderItem = """
        CREATE TABLE Order_Item (
            order_id     NUMBER(12)           NOT NULL,
            product_id   NUMBER(10)           NOT NULL,
            quantity     NUMBER(10)           DEFAULT 1 NOT NULL,
            unit_price   NUMBER(10,2)         NOT NULL,
            CONSTRAINT pk_order_item PRIMARY KEY (order_id, product_id),
            CONSTRAINT chk_order_item_qty_pos CHECK (quantity > 0),
            CONSTRAINT chk_order_item_price_nonneg CHECK (unit_price >= 0),
            CONSTRAINT fk_order_item_order
                FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE,
            CONSTRAINT fk_order_item_product
                FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE CASCADE
        )
        """;

String createDiscounts = """
        CREATE TABLE Discounts (
            discount_id      NUMBER(12)        PRIMARY KEY,
            vip_tier         VARCHAR2(20)      NOT NULL,
            start_date       DATE              NOT NULL,
            end_date         DATE              NOT NULL,
            discount_amount  NUMBER(5,2)       NOT NULL,
            order_id         NUMBER(12)        NOT NULL,
            CONSTRAINT chk_discounts_viptier CHECK (vip_tier IN ('NONE','SILVER','GOLD','PLATINUM')),
            CONSTRAINT chk_discounts_pct CHECK (discount_amount >= 0 AND discount_amount <= 100),
            CONSTRAINT chk_discounts_dates CHECK (end_date >= start_date),
            CONSTRAINT fk_discounts_order
                FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE
        )
        """;

String[] ddl = {
        createStore,
        createCustomer,
        createProduct,
        createEmployee,
        createOrders,
        createInventory,
        createOrderItem,
        createDiscounts
};

for (String sql : ddl) {
    try (Statement st = conn.createStatement()) {
        st.executeUpdate(sql);
        appendLine("Created table with statement:\n" + sql);
    } catch (SQLException e) {
        appendLine("Error executing DDL:\n" + sql);
```

```
            appendLine("  -> " + e.getMessage());
        }
    }
}

// Insert sample data into all tables
private void populateTables(Connection conn) throws SQLException {
    try (Statement st = conn.createStatement()) {

        // STORE
        st.executeUpdate(
            """
                INSERT INTO Store (store_id, address, opened_date, status, phone, store_hours)
                VALUES (1, '100 King St W, Toronto, ON', DATE '2022-03-01', 'OPEN', '416-555-1000', 'Mon-Fri
10:00-21:00')
                """);
        st.executeUpdate(
            """
                INSERT INTO Store (store_id, address, opened_date, status, phone, store_hours)
                VALUES (2, '200 Queen St W, Toronto, ON', DATE '2023-05-15', 'OPEN', '416-555-2000',
'Mon-Sun 11:00-19:00')
                """);

        // CUSTOMER
        st.executeUpdate(
            """
                INSERT INTO Customer (customer_id, name, email, phone, date_joined, vip_tier)
                VALUES (1001, 'Saloni Patel',  'saloni.patel@hotmail.com',  '647-555-0101', DATE '2024-01-10',
'SILVER')
                """);
        st.executeUpdate(
            """
                INSERT INTO Customer (customer_id, name, email, phone, date_joined, vip_tier)
                VALUES (1002, 'Abisa Rajkumar', 'abisa.rajkumar@gmail.com', '647-555-0202', DATE
'2024-02-05', 'GOLD')
                """);
        st.executeUpdate("""
            INSERT INTO Customer (customer_id, name, email, phone, date_joined, vip_tier)
            VALUES (1003, 'Nehal Goel', 'nehal.goel@yahoo.com', NULL, DATE '2024-03-01', 'NONE')
            """);

        // PRODUCT
        st.executeUpdate("""
            INSERT INTO Product (product_id, category, base_price)
            VALUES (100, 'Tops', 30.00)
            """);
        st.executeUpdate("""
            INSERT INTO Product (product_id, category, base_price)
            VALUES (101, 'Jeans', 60.00)
            """);
        st.executeUpdate("""
            INSERT INTO Product (product_id, category, base_price)
            VALUES (102, 'Accessories', 15.00)
            """);

        // EMPLOYEE
        st.executeUpdate("""
            INSERT INTO Employee (employee_id, store_id, name, email, role, sales_per_week)
            VALUES (5001, 1, 'Mike Brown',  'mike.brown@corp.com',  'Manager',        15000.00)
            """);
        st.executeUpdate(
            """
                INSERT INTO Employee (employee_id, store_id, name, email, role, sales_per_week)
                VALUES (5002, 1, 'Immanuel Gnanaseelan',   'immanuel.gnanaseelan@corp.com',   'Sales
Associate',  8000.00)
                """);
        st.executeUpdate("""
            INSERT INTO Employee (employee_id, store_id, name, email, role, sales_per_week)
            VALUES (5003, 2, 'Mia Johnson',   'mia.johnson@crop.com',   'Sales Associate',  9000.00)
            """);

        // ORDERS
        st.executeUpdate("""
            INSERT INTO Orders (order_id, customer_id, employee_id, order_type, order_total, order_date)
            VALUES (90001, 1001, 5002, 'ONLINE', 70.00, DATE '2024-11-01')
            """);
        st.executeUpdate("""
            INSERT INTO Orders (order_id, customer_id, employee_id, order_type, order_total, order_date)
            VALUES (90002, 1002, 5003, 'IN_STORE', 60.00, DATE '2024-11-02')
            """);
        st.executeUpdate("""
            INSERT INTO Orders (order_id, customer_id, employee_id, order_type, order_total, order_date)
            VALUES (90003, 1001, 5002, 'IN_STORE', 44.00, DATE '2024-11-03')
            """);

        // INVENTORY
        st.executeUpdate("""
            INSERT INTO Inventory (store_id, product_id, quantity_available, last_restocked, store_price)
            VALUES (1, 100, 50, DATE '2024-10-01', 28.00)
            """);
        st.executeUpdate("""
            INSERT INTO Inventory (store_id, product_id, quantity_available, last_restocked, store_price)
            VALUES (1, 101, 35, DATE '2024-10-05', 60.00)
            """);
        st.executeUpdate("""
            INSERT INTO Inventory (store_id, product_id, quantity_available, last_restocked, store_price)
            VALUES (2, 100, 20, DATE '2024-10-03', 30.00)
            """);
        st.executeUpdate("""
            INSERT INTO Inventory (store_id, product_id, quantity_available, last_restocked, store_price)
            VALUES (2, 102, 60, DATE '2024-10-06', 14.00)
            """);

        // ORDER_ITEM
        st.executeUpdate("""
            INSERT INTO Order_Item (order_id, product_id, quantity, unit_price)
            VALUES (90001, 100, 2, 28.00)
            """);
        st.executeUpdate("""
            INSERT INTO Order_Item (order_id, product_id, quantity, unit_price)
            VALUES (90001, 102, 1, 14.00)
            """);
        st.executeUpdate("""
            INSERT INTO Order_Item (order_id, product_id, quantity, unit_price)
            VALUES (90002, 101, 1, 60.00)
            """);
        st.executeUpdate("""
            INSERT INTO Order_Item (order_id, product_id, quantity, unit_price)
            VALUES (90003, 100, 1, 30.00)
            """);
        st.executeUpdate("""
            INSERT INTO Order_Item (order_id, product_id, quantity, unit_price)
            VALUES (90003, 102, 1, 14.00)
            """);

        // DISCOUNTS
        st.executeUpdate("""
            INSERT INTO Discounts (discount_id, vip_tier, start_date, end_date, discount_amount, order_id)
            VALUES (1, 'SILVER', DATE '2024-11-01', DATE '2024-11-30', 10.00, 90001)
            """);
        st.executeUpdate("""
            INSERT INTO Discounts (discount_id, vip_tier, start_date, end_date, discount_amount, order_id)
            VALUES (2, 'GOLD',   DATE '2024-11-01', DATE '2024-11-15', 15.00, 90002)
            """);
        st.executeUpdate("""
            INSERT INTO Discounts (discount_id, vip_tier, start_date, end_date, discount_amount, order_id)
            VALUES (3, 'NONE',   DATE '2024-11-03', DATE '2024-11-30',  0.00, 90003)
            """);

        appendLine("Dummy data inserted successfully.");
    }
}

// Show tables
private void showTables(Connection conn) throws SQLException {
    String sql = "SELECT table_name FROM user_tables ORDER BY table_name";
    try (Statement st = conn.createStatement();
         ResultSet rs = st.executeQuery(sql)) {
      appendLine("User tables:");
      while (rs.next()) {
         appendLine("  " + rs.getString(1));
      }
   }
   String tableName = ask("Enter a table name to show its rows (or Cancel to skip):");
   if (tableName == null || tableName.isBlank()) {
      return; // user cancelled
   }

   tableName = tableName.trim().toUpperCase();
   // Select all rows from that table and print them
   String query = "SELECT * FROM " + tableName;
   try (Statement st2 = conn.createStatement();
         ResultSet rs2 = st2.executeQuery(query)) {

      ResultSetMetaData md = rs2.getMetaData();
      int cols = md.getColumnCount();

      appendLine("\nRows in " + tableName + ":");
      boolean any = false;

      while (rs2.next()) {
         any = true;
         StringBuilder row = new StringBuilder("  ");
         for (int i = 1; i <= cols; i++) {
            row.append(md.getColumnName(i))
                  .append("=")
                  .append(rs2.getString(i));
            if (i < cols)
               row.append(" | ");
         }
         appendLine(row.toString());
      }

      if (!any) {
         appendLine("  (no rows)");
      }
   } catch (SQLException e) {
      appendLine("Error showing data for " + tableName + ": " + e.getMessage());
   }
}

// SEARCH by the Primary Key
private void searchByPrimaryKey(Connection conn, String tableName) throws SQLException {
   String t = tableName.toUpperCase();
   String sql;
   PreparedStatement ps;
```

```
switch (t) {
    case "STORE" -> {
        String id = ask("Enter store_id:");
        if (id == null)
            return;
        sql = "SELECT * FROM Store WHERE store_id = ?";
        ps = conn.prepareStatement(sql);
        ps.setInt(1, Integer.parseInt(id));
    }
    case "CUSTOMER" -> {
        String id = ask("Enter customer_id:");
        if (id == null)
            return;
        sql = "SELECT * FROM Customer WHERE customer_id = ?";
        ps = conn.prepareStatement(sql);
        ps.setInt(1, Integer.parseInt(id));
    }
    case "PRODUCT" -> {
        String id = ask("Enter product_id:");
        if (id == null)
            return;
        sql = "SELECT * FROM Product WHERE product_id = ?";
        ps = conn.prepareStatement(sql);
        ps.setInt(1, Integer.parseInt(id));
    }
    case "EMPLOYEE" -> {
        String id = ask("Enter employee_id:");
        if (id == null)
            return;
        sql = "SELECT * FROM Employee WHERE employee_id = ?";
        ps = conn.prepareStatement(sql);
        ps.setInt(1, Integer.parseInt(id));
    }
    case "ORDERS" -> {
        String id = ask("Enter order_id:");
        if (id == null)
            return;
        sql = "SELECT * FROM Orders WHERE order_id = ?";
        ps = conn.prepareStatement(sql);
        ps.setInt(1, Integer.parseInt(id));
    }
    case "INVENTORY" -> {
        String sid = ask("Enter store_id:");
        if (sid == null)
            return;
        String pid = ask("Enter product_id:");
        if (pid == null)
            return;
        sql = "SELECT * FROM Inventory WHERE store_id = ? AND product_id = ?";
        ps = conn.prepareStatement(sql);
        ps.setInt(1, Integer.parseInt(sid));
        ps.setInt(2, Integer.parseInt(pid));
    }
    case "ORDER_ITEM" -> {
        String oid = ask("Enter order_id:");
        if (oid == null)
            return;
        String pid = ask("Enter product_id:");
        if (pid == null)
            return;
        sql = "SELECT * FROM Order_Item WHERE order_id = ? AND product_id = ?";
        ps = conn.prepareStatement(sql);
        ps.setInt(1, Integer.parseInt(oid));
        ps.setInt(2, Integer.parseInt(pid));
    }
    case "DISCOUNTS" -> {
        String id = ask("Enter discount_id:");
        if (id == null)
            return;
        sql = "SELECT * FROM Discounts WHERE discount_id = ?";
        ps = conn.prepareStatement(sql);
        ps.setInt(1, Integer.parseInt(id));
    }
    default -> {
        appendLine("Unknown table: " + tableName);
        return;
    }
}

try (ps; ResultSet rs = ps.executeQuery()) {
    ResultSetMetaData md = rs.getMetaData();
    int cols = md.getColumnCount();
    appendLine("\nResult for " + t + ":");
    if (!rs.next()) {
        appendLine("  No row found.");
        return;
    }
    do {
        StringBuilder row = new StringBuilder("  ");
        for (int i = 1; i <= cols; i++) {
            row.append(md.getColumnName(i)).append("=").append(rs.getString(i));
            if (i < cols)
                row.append(" | ");
        }
        appendLine(row.toString());
    } while (rs.next());
}
```

```
}

// UPDATE by Primary Key
private void updateByPrimaryKey(Connection conn, String tableName) throws SQLException {
    String t = tableName.toUpperCase();

    String column = ask("Enter column name to update:");
    if (column == null || column.isBlank())
        return;

    String newValue = ask("Enter new value:");
    if (newValue == null)
        return;

    String whereClause;
    PreparedStatement ps;

    switch (t) {
        case "STORE" -> {
            String id = ask("Enter store_id:");
            if (id == null)
                return;
            whereClause = "store_id = ?";
            ps = conn.prepareStatement("UPDATE Store SET " + column + " = ? WHERE " + whereClause);
            ps.setString(1, newValue);
            ps.setInt(2, Integer.parseInt(id));
        }
        case "CUSTOMER" -> {
            String id = ask("Enter customer_id:");
            if (id == null)
                return;
            whereClause = "customer_id = ?";
            ps = conn.prepareStatement("UPDATE Customer SET " + column + " = ? WHERE " + whereClause);
            ps.setString(1, newValue);
            ps.setInt(2, Integer.parseInt(id));
        }
        case "PRODUCT" -> {
            String id = ask("Enter product_id:");
            if (id == null)
                return;
            whereClause = "product_id = ?";
            ps = conn.prepareStatement("UPDATE Product SET " + column + " = ? WHERE " + whereClause);
            ps.setString(1, newValue);
            ps.setInt(2, Integer.parseInt(id));
        }
        case "EMPLOYEE" -> {
            String id = ask("Enter employee_id:");
            if (id == null)
                return;
        whereClause = "employee_id = ?";
        ps = conn.prepareStatement("UPDATE Employee SET " + column + " = ? WHERE " + whereClause);
        ps.setString(1, newValue);
        ps.setInt(2, Integer.parseInt(id));
    }
    case "ORDERS" -> {
        String id = ask("Enter order_id:");
        if (id == null)
            return;
        whereClause = "order_id = ?";
        ps = conn.prepareStatement("UPDATE Orders SET " + column + " = ? WHERE " + whereClause);
        ps.setString(1, newValue);
        ps.setInt(2, Integer.parseInt(id));
    }
    case "INVENTORY" -> {
        String sid = ask("Enter store_id:");
        if (sid == null)
            return;
        String pid = ask("Enter product_id:");
        if (pid == null)
            return;
        whereClause = "store_id = ? AND product_id = ?";
        ps = conn.prepareStatement("UPDATE Inventory SET " + column + " = ? WHERE " + whereClause);
        ps.setString(1, newValue);
        ps.setInt(2, Integer.parseInt(sid));
        ps.setInt(3, Integer.parseInt(pid));
    }
    case "ORDER_ITEM" -> {
        String oid = ask("Enter order_id:");
        if (oid == null)
            return;
        String pid = ask("Enter product_id:");
        if (pid == null)
            return;
        whereClause = "order_id = ? AND product_id = ?";
        ps = conn.prepareStatement("UPDATE Order_Item SET " + column + " = ? WHERE " + whereClause);
        ps.setString(1, newValue);
        ps.setInt(2, Integer.parseInt(oid));
        ps.setInt(3, Integer.parseInt(pid));
    }
    case "DISCOUNTS" -> {
        String id = ask("Enter discount_id:");
        if (id == null)
            return;
        whereClause = "discount_id = ?";
        ps = conn.prepareStatement("UPDATE Discounts SET " + column + " = ? WHERE " + whereClause);
        ps.setString(1, newValue);
        ps.setInt(2, Integer.parseInt(id));
    }
```

```
        default -> {
            appendLine("Unknown table: " + tableName);
            return;
        }
    }

    try (ps) {
        int rows = ps.executeUpdate();
        appendLine("Updated " + rows + " row(s) in " + t + ".");
    }
}

// DELETE by Primary Key
private void deleteByPrimaryKey(Connection conn, String tableName) throws SQLException {
    String t = tableName.toUpperCase();
    String sql;
    PreparedStatement ps;

    switch (t) {
        case "STORE" -> {
            String id = ask("Enter store_id:");
            if (id == null)
                return;
            sql = "DELETE FROM Store WHERE store_id = ?";
            ps = conn.prepareStatement(sql);
            ps.setInt(1, Integer.parseInt(id));
        }
        case "CUSTOMER" -> {
            String id = ask("Enter customer_id:");
            if (id == null)
                return;
            sql = "DELETE FROM Customer WHERE customer_id = ?";
            ps = conn.prepareStatement(sql);
            ps.setInt(1, Integer.parseInt(id));
        }
        case "PRODUCT" -> {
            String id = ask("Enter product_id:");
            if (id == null)
                return;
            sql = "DELETE FROM Product WHERE product_id = ?";
            ps = conn.prepareStatement(sql);
            ps.setInt(1, Integer.parseInt(id));
        }
        case "EMPLOYEE" -> {
            String id = ask("Enter employee_id:");
            if (id == null)
                return;
            sql = "DELETE FROM Employee WHERE employee_id = ?";
            ps = conn.prepareStatement(sql);
        ps.setInt(1, Integer.parseInt(id));
    }
    case "ORDERS" -> {
        String id = ask("Enter order_id:");
        if (id == null)
            return;
        sql = "DELETE FROM Orders WHERE order_id = ?";
        ps = conn.prepareStatement(sql);
        ps.setInt(1, Integer.parseInt(id));
    }
    case "INVENTORY" -> {
        String sid = ask("Enter store_id:");
        if (sid == null)
            return;
        String pid = ask("Enter product_id:");
        if (pid == null)
            return;
        sql = "DELETE FROM Inventory WHERE store_id = ? AND product_id = ?";
        ps = conn.prepareStatement(sql);
        ps.setInt(1, Integer.parseInt(sid));
        ps.setInt(2, Integer.parseInt(pid));
    }
    case "ORDER_ITEM" -> {
        String oid = ask("Enter order_id:");
        if (oid == null)
            return;
        String pid = ask("Enter product_id:");
        if (pid == null)
            return;
        sql = "DELETE FROM Order_Item WHERE order_id = ? AND product_id = ?";
        ps = conn.prepareStatement(sql);
        ps.setInt(1, Integer.parseInt(oid));
        ps.setInt(2, Integer.parseInt(pid));
    }
    case "DISCOUNTS" -> {
        String id = ask("Enter discount_id:");
        if (id == null)
            return;
        sql = "DELETE FROM Discounts WHERE discount_id = ?";
        ps = conn.prepareStatement(sql);
        ps.setInt(1, Integer.parseInt(id));
    }
    default -> {
        appendLine("Unknown table: " + tableName);
        return;
    }
}

try (ps) {
```

```
            int rows = ps.executeUpdate();
            appendLine("Deleted " + rows + " row(s) from " + t + ".");
        }
    }

    // Small helpers
    private void runQuery(Connection conn, String title, String sql) throws SQLException {
        appendLine("\n=== " + title + " ===");

        try (Statement st = conn.createStatement();
                ResultSet rs = st.executeQuery(sql)) {

            ResultSetMetaData md = rs.getMetaData();
            int cols = md.getColumnCount();
            boolean any = false;

            while (rs.next()) {
                any = true;
                StringBuilder row = new StringBuilder("  ");
                for (int i = 1; i <= cols; i++) {
                    String label = md.getColumnLabel(i);
                    row.append(label)
                            .append("=")
                            .append(rs.getString(i));

                    if (i < cols)
                        row.append(" | ");
                }
                appendLine(row.toString());
            }

            if (!any) {
                appendLine("  (no rows)");
            }
        }
    }

    private String ask(String message) {
        return JOptionPane.showInputDialog(this, message);
    }

    private void appendLine(String text) {
        outputArea.append(text + "\n");
        outputArea.setCaretPosition(outputArea.getDocument().getLength());
    }
}
```

Conclusion

This project successfully evolved our retail database system from a conceptual model into a fully normalized and operational BCNF design. Throughout Assignments A1-A10, each stage focused on identifying redundancy, partial dependencies and transitive dependencies, in which we addressed through systemic normalization techniques like decompositions. The resulting schema is now consistent, non-redundant, and optimized for scalability, allowing strong data to be accessed across all operations. With this polished structure, the database supports accurate reports, efficient queries, and integration with the user interface that we developed at the end of this project. Overall, the final DBMS provides a reliable and maintainable foundation dor managing customers, employees, inventory, orders, product, order items and discounts within a modern retail environment.