CLOTHING RETAIL STORE

Database Management System

Final Report

CPS510: Database I

Section 13

Prof. Glaucia Melo

November 29, 2024

Piyush Patel, Ridham Aggarwal, Saiyon Jeyakumar

Table of Contents

Application Description

This clothing retail store application will be responsible for maintaining relevant information necessary for a clothing store business. It aims to manage all aspects of inventory, sales, customer information, and other tracking processes within a retail clothing store. This system caters to both small-scale and large-scale clothing retailers, in order to help them optimize inventory management, enhance customer satisfaction, and boost operational efficiency. This report provides a comprehensive overview of the application's functionality, the information expected from it, and the technical design required to achieve efficient and scalable operations.

The database management system intends to:

- Store and manage product inventory
- Process and track sales transactions
- Handle customer information
- Track purchase history of customers

Some of the functions that the system will be able to provide are outlined below:

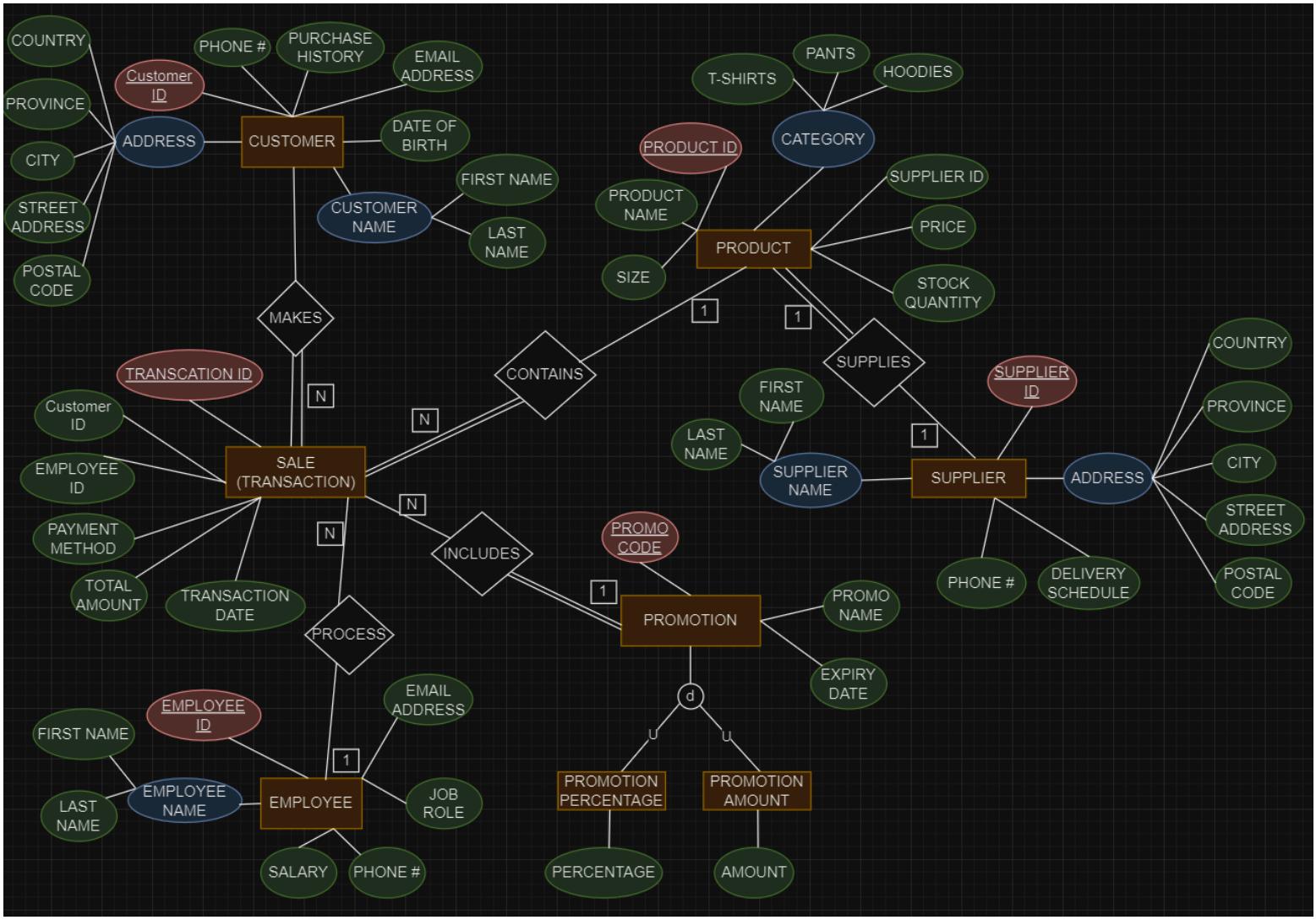
Function	Description
Account Management	The user can create an account, where they can manage and update their personal information.
Product Management	Stores the details of each product ranging from quantity to size.
Sales Management	Record sales details, including items sold, transaction amount, etc. Supports multiple payment methods.
Customer Management	Store customer details such as name, contact information,and purchase history.
Supplier Management	Store information about suppliers, including contact details, delivery schedules, and payment terms. Track the products they supply.
Employee Management	Maintain records of employee profiles, job roles, salaries, and performance metrics.

In the next section, the ER diagram illustrates the different components of our database management system.

- The orange-coloured items represent the entities that exist in our database management system
- The green-coloured item represents the attributes that each entity possesses

- The red-coloured items represent the primary key of each entity
- The black-coloured items represent the relationships between each entity

ER Model



Schema Design

Creating Tables

```
CREATE TABLE Address(
        Address_ID NUMBER PRIMARY KEY,
        Country_Name VARCHAR2(50) NOT NULL,
        Province_Name VARCHAR2(50) NOT NULL,
        City_Name VARCHAR2(15) NOT NULL,
        Street_Address VARCHAR2(100) NOT NULL,
        Postal_Code VARCHAR2(100) NOT NULL
);

CREATE TABLE Customer (
        Customer_ID NUMBER PRIMARY KEY,
        First_Name VARCHAR2(50) NOT NULL,
        Last_Name VARCHAR2(50) NOT NULL,
        Phone_Number VARCHAR2(15) UNIQUE,
        Email_Address VARCHAR2(100) UNIQUE,
        Date_of_Birth DATE,
        Address_ID NUMBER DEFAULT 1,
        FOREIGN KEY (Address_ID) REFERENCES Address(Address_ID)
);

CREATE TABLE Sale (
        Transaction_ID NUMBER PRIMARY KEY,
        Transaction_Date DATE,
        Total_Amount NUMBER(10, 2),
        Payment_Method VARCHAR2(50),
        Customer_ID NUMBER,
        Employee_ID NUMBER,
        FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID),
        FOREIGN KEY (Employee_ID) REFERENCES Employee(Employee_ID)
);

CREATE TABLE Supplier (
        Supplier_ID NUMBER PRIMARY KEY,
        First_Name VARCHAR2(50) NOT NULL,
        Last_Name VARCHAR2(50) NOT NULL,
        Phone_Number VARCHAR2(15) UNIQUE,
        Delivery_Schedule DATE,
        Address_ID NUMBER,
```

```
    FOREIGN KEY (Address_ID) REFERENCES Address(Address_ID)
);

CREATE TABLE Product (
    Product_ID NUMBER PRIMARY KEY,
    Product_Name VARCHAR2(50) NOT NULL,
    Product_Size VARCHAR2(25) NOT NULL,
    Product_Price VARCHAR2(5) NOT NULL,
    Stock_Quantity VARCHAR2(15) NOT NULL,
    Product_Category VARCHAR2(15) NOT NULL,
    Supplier_ID Number,
    FOREIGN KEY (Supplier_ID) REFERENCES Supplier(Supplier_ID)
);

CREATE TABLE Employee (
    Employee_ID INT PRIMARY KEY,
    FirstName VARCHAR2(50) NOT NULL,
    LastName VARCHAR2(50) NOT NULL,
    Salary DECIMAL(10, 2),
    PhoneNumber VARCHAR2(15),
    JobRole VARCHAR2(50),
    EmailAddress VARCHAR2(100) UNIQUE
);

CREATE TABLE Promotion (
    PromoCod INT PRIMARY KEY,
    PromoName VARCHAR2(100) NOT NULL,
    ExpiryDate DATE NOT NULL,
    PromotionPercent DECIMAL(5, 2),
    PromotionAmount DECIMAL(10, 2)
);
```

Inserting Sample Data

```
INSERT INTO Address (Address_ID, Country_Name, Province_Name, City_Name,
Street_Address, Postal_Code)
VALUES
  (1, 'Canada', 'Ontario', 'Toronto', '126 Queen St', 'L3R 6Y6'),
  (2, 'Canada', 'Ontario', 'Toronto', '344 King St', 'L3R 4K9'),
  (3, 'Canada', 'Ontario', 'Markham', '51 Tangmere Cres', 'L3R 4J9');
```

```
INSERT INTO Supplier (Supplier_ID, First_Name, Last_Name, Phone_Number,
Delivery_Schedule, Address_ID)
VALUES
   (1, 'John', 'Doe', '4161231111', TO_DATE('2024-10-02', 'YYYY-MM-DD'), 1),
   (2, 'Jane', 'Smith', '6479084567', TO_DATE('2024-10-05', 'YYYY-MM-DD'), 2),
   (3, 'Emily', 'Davis', '9052945678', TO_DATE('2024-10-10', 'YYYY-MM-DD'), 3);

INSERT INTO Product (Product_ID, Product_Name, Product_Size, Product_Price,
Stock_Quantity, Product_Category, Supplier_ID)
VALUES
   (1, 'T-shirt', 'Medium', '20', '150', 'Top', 1),
   (2, 'Jeans', '32x30', '40', '100', 'Bottom', 2),
   (3, 'Jacket', 'Large', '60', '75', 'Top', 3);

INSERT INTO Customer (Customer_ID, First_Name, Last_Name, Phone_Number,
Email_Address, Date_of_Birth, Address_ID)
VALUES (1, 'John', 'Doe', '555-555-1234', 'john.doe@example.com', TO_DATE('1985-05-10',
'YYYY-MM-DD'), 2);

INSERT INTO Sale (Transaction_ID, Transaction_Date, Total_Amount, Payment_Method,
Customer_ID, Employee_ID)
VALUES (1001, TO_DATE('2024-10-03', 'YYYY-MM-DD'), 250.75, 'Credit Card', 1, 101);

INSERT INTO Employee (Employee_ID, FirstName, LastName, Salary, PhoneNumber,
JobRole, EmailAddress)
VALUES (1, 'Alice', 'Johnson', 75000.00, '555-1234', 'Software Engineer',
'alice.johnson@example.com');

INSERT INTO Promotion (PromoCode, PromoName, ExpiryDate, PromotionPercent,
PromotionAmount)
VALUES (1, 'Fall Discount', TO_DATE('2024-12-31', 'YYYY-MM-DD'), 20.00, 10.00);
```

Simple/Advanced Queries & Views

```
CREATE VIEW CustomerOrdersView AS
SELECT c.Customer_ID,
        c.First_Name || ' ' || c.Last_Name AS Customer_Name,
        c.Email_Address,
        COUNT(s.Transaction_ID) AS Total_Transactions,
        SUM(s.Total_Amount) AS Total_Spent
FROM Customer c
JOIN Sale s ON c.Customer_ID = s.Customer_ID
GROUP BY c.Customer_ID, c.First_Name, c.Last_Name, c.Email_Address;

CREATE VIEW SupplierProductView AS
SELECT s.Supplier_ID,
        s.First_Name || ' ' || s.Last_Name AS Supplier_Name,
        p.Product_Category,
        COUNT(p.Product_ID) AS Product_Count,
        SUM(TO_NUMBER(p.Stock_Quantity)) AS Total_Stock
FROM Supplier s
JOIN Product p ON s.Supplier_ID = p.Supplier_ID
GROUP BY s.Supplier_ID, s.First_Name, s.Last_Name, p.Product_Category;

CREATE VIEW EmployeeSalesPerformanceView AS
SELECT e.Employee_ID,
        e.FirstName || ' ' || e.LastName AS Employee_Name,
        e.JobRole,
        COUNT(s.Transaction_ID) AS Transactions_Handled,
        SUM(s.Total_Amount) AS Total_Sales_Amount
FROM Employee e
JOIN Sale s ON e.Employee_ID = s.Employee_ID
GROUP BY e.Employee_ID, e.FirstName, e.LastName, e.JobRole;

SELECT a.Country_Name,
    a.Province_Name,
    a.City_Name,
    COUNT(s.Supplier_ID) AS Supplier_Count,
    AVG(s.Delivery_Schedule) AS Average_Delivery_Schedule
FROM Supplier s
JOIN Address a ON s.Address_ID = a.Address_ID
GROUP BY a.Country_Name, a.Province_Name, a.City_Name
ORDER BY a.Country_Name, a.Province_Name, a.City_Name;

SELECT p.Product_ID, AVG(p.Product_Price) AS Average_Product_Price
FROM Product p
GROUP BY p.Product_ID;
```

```
SELECT Product_ID, Product_Name
FROM Product
MINUS
SELECT Product_ID, Product_Name
FROM Product
WHERE Supplier_ID IS NOT NULL;

SELECT s.Supplier_ID,
        s.First_Name || ' ' || s.Last_Name AS Supplier_Name,
        COUNT(DISTINCT p.Product_Category) AS Category_Count
FROM Supplier s
JOIN Product p ON s.Supplier_ID = p.Supplier_ID
GROUP BY s.Supplier_ID, s.First_Name, s.Last_Name
HAVING COUNT(DISTINCT p.Product_Category) > 1;

SELECT c.Customer_ID, c.First_Name || ' ' || c.Last_Name AS Customer_Name
FROM Customer c
WHERE NOT EXISTS (
   SELECT 1
   FROM Sale s
   WHERE s.Customer_ID = c.Customer_ID
     AND s.Transaction_Date >= ADD_MONTHS(SYSDATE, -12)
);

SELECT e.Employee_ID, e.FirstName || ' ' || e.LastName AS Employee_Name,
      STDDEV(s.Total_Amount) AS Sales_Amount_StdDev
FROM Employee e
JOIN Sale s ON e.Employee_ID = s.Employee_ID
GROUP BY e.Employee_ID, e.FirstName, e.LastName
HAVING STDDEV(s.Total_Amount) > 0;

SELECT Product_ID, Product_Name, 'Supplied' AS Status
FROM Product
WHERE Supplier_ID IS NOT NULL

UNION

SELECT Product_ID, Product_Name, 'Not Supplied' AS Status
FROM Product
WHERE Supplier_ID IS NULL;
```

UNIX Shell

```
CREATE VIEW CustomerOrdersView AS
SELECT c.Customer_ID,
        c.First_Name || ' ' || c.Last_Name AS Customer_Name,
        c.Email_Address,
        COUNT(s.Transaction_ID) AS Total_Transactions,
        SUM(s.Total_Amount) AS Total_Spent
FROM Customer c
JOIN Sale s ON c.Customer_ID = s.Customer_ID
GROUP BY c.Customer_ID, c.First_Name, c.Last_Name, c.Email_Address;



CREATE VIEW SupplierProductView AS
SELECT s.Supplier_ID,
        s.First_Name || ' ' || s.Last_Name AS Supplier_Name,
        p.Product_Category,
        COUNT(p.Product_ID) AS Product_Count,
        SUM(TO_NUMBER(p.Stock_Quantity)) AS Total_Stock
FROM Supplier s
JOIN Product p ON s.Supplier_ID = p.Supplier_ID
GROUP BY s.Supplier_ID, s.First_Name, s.Last_Name, p.Product_Category;



CREATE VIEW EmployeeSalesPerformanceView AS
SELECT e.Employee_ID,
        e.FirstName || ' ' || e.LastName AS Employee_Name,
        e.JobRole,
        COUNT(s.Transaction_ID) AS Transactions_Handled,
        SUM(s.Total_Amount) AS Total_Sales_Amount
FROM Employee e
JOIN Sale s ON e.Employee_ID = s.Employee_ID
GROUP BY e.Employee_ID, e.FirstName, e.LastName, e.JobRole;
```

```
SELECT a.Country_Name,
       a.Province_Name,
       a.City_Name,
       COUNT(s.Supplier_ID) AS Supplier_Count,
       AVG(s.Delivery_Schedule) AS Average_Delivery_Schedule
FROM Supplier s
JOIN Address a ON s.Address_ID = a.Address_ID
GROUP BY a.Country_Name, a.Province_Name, a.City_Name
ORDER BY a.Country_Name, a.Province_Name, a.City_Name;
```

```
SELECT p.Product_ID, AVG(p.Product_Price) AS Average_Product_Price
FROM Product p
GROUP BY p.Product_ID;
```

```
SELECT Product_ID, Product_Name
FROM Product
MINUS
SELECT Product_ID, Product_Name
FROM Product
WHERE Supplier_ID IS NOT NULL;
```

```
SELECT s.Supplier_ID,
          s.First_Name || ' ' || s.Last_Name AS Supplier_Name,
          COUNT(DISTINCT p.Product_Category) AS Category_Count
FROM Supplier s
JOIN Product p ON s.Supplier_ID = p.Supplier_ID
GROUP BY s.Supplier_ID, s.First_Name, s.Last_Name
HAVING COUNT(DISTINCT p.Product_Category) > 1;
```

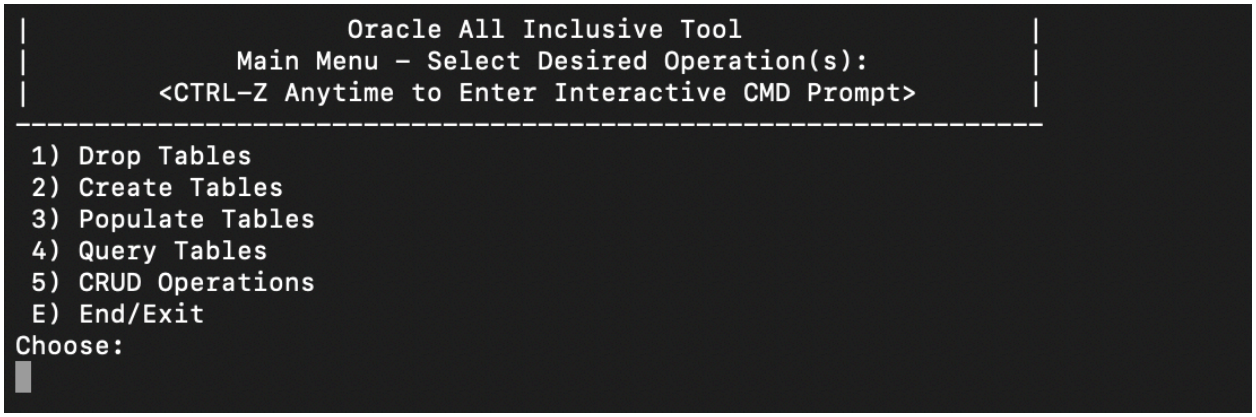
```
SELECT c.Customer_ID, c.First_Name || ' ' || c.Last_Name AS Customer_Name
FROM Customer c
WHERE NOT EXISTS (
   SELECT 1
   FROM Sale s
   WHERE s.Customer_ID = c.Customer_ID
     AND s.Transaction_Date >= ADD_MONTHS(SYSDATE, -12)
);
```

```
SELECT e.Employee_ID, e.FirstName || ' ' || e.LastName AS Employee_Name,
      STDDEV(s.Total_Amount) AS Sales_Amount_StdDev
FROM Employee e
JOIN Sale s ON e.Employee_ID = s.Employee_ID
GROUP BY e.Employee_ID, e.FirstName, e.LastName
HAVING STDDEV(s.Total_Amount) > 0;
```

```
SELECT Product_ID, Product_Name, 'Supplied' AS Status
FROM Product
WHERE Supplier_ID IS NOT NULL

UNION

SELECT Product_ID, Product_Name, 'Not Supplied' AS Status
FROM Product
WHERE Supplier_ID IS NULL;
```

RESULTS

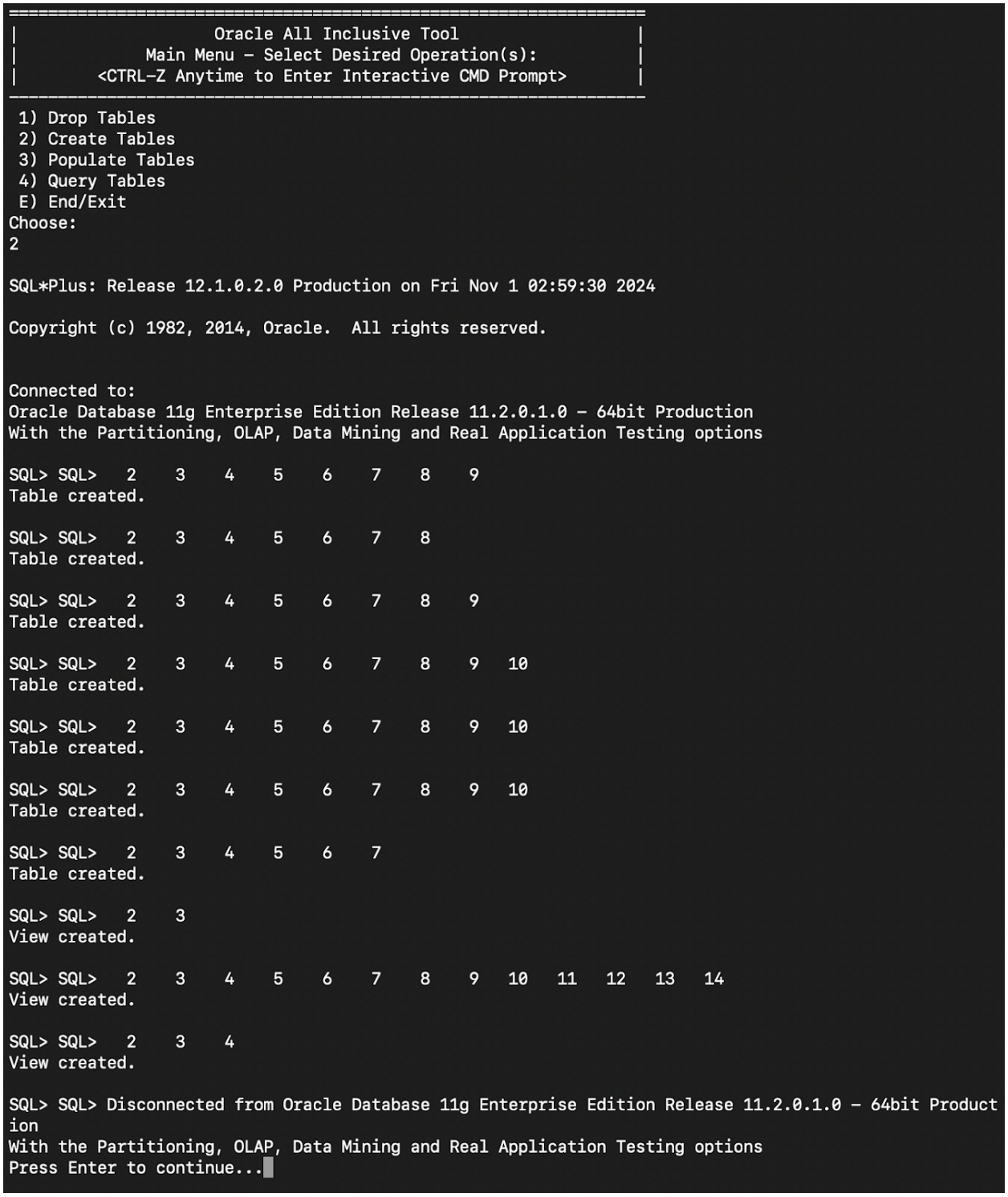
Menu:



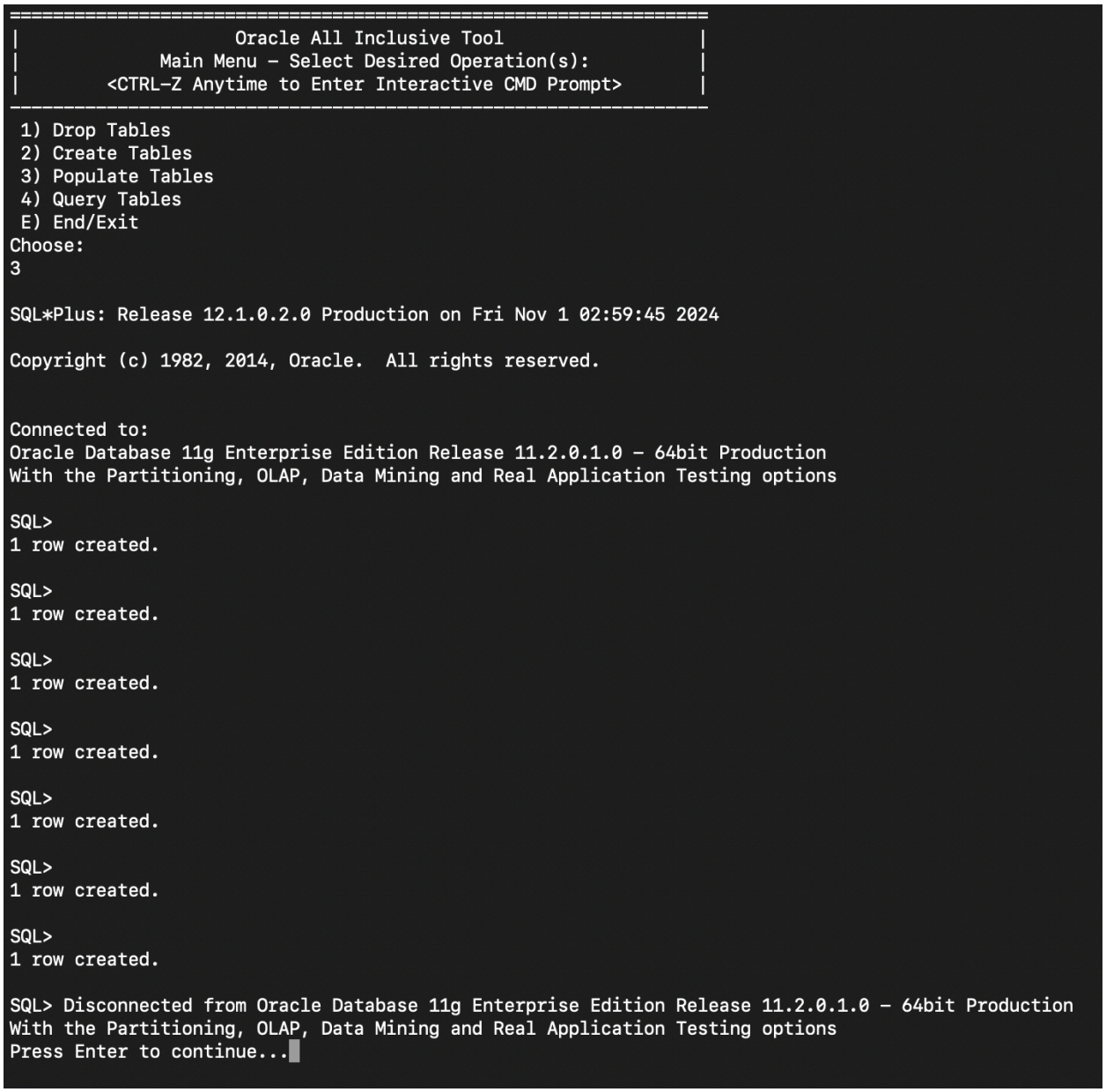
Drop table Command:

```
================================================================
|                  Oracle All Inclusive Tool                   |
|            Main Menu - Select Desired Operation(s):          |
|        <CTRL-Z Anytime to Enter Interactive CMD Prompt>      |
----------------------------------------------------------------
 1) Drop Tables
 2) Create Tables
 3) Populate Tables
 4) Query Tables
 E) End/Exit
Choose:
1

SQL*Plus: Release 12.1.0.2.0 Production on Fri Nov 1 02:59:08 2024

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>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
View dropped.

SQL>
View dropped.

SQL>
View 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
Press Enter to continue...
```

Create table Command:



Populate table Command:



Query table Command:

```
================================================================
|                  Oracle All Inclusive Tool                   |
|            Main Menu - Select Desired Operation(s):          |
|        <CTRL-Z Anytime to Enter Interactive CMD Prompt>      |
----------------------------------------------------------------
 1) Drop Tables
 2) Create Tables
 3) Populate Tables
 4) Query Tables
 E) End/Exit
Choose:
4
Select a Query to Run:
1. View Customer Orders
2. View Supplier Product Summary
3. View Employee Sales Performance
Choose a query:
```

Query 1:

```
================================================================
|                  Oracle All Inclusive Tool                   |
|            Main Menu - Select Desired Operation(s):          |
|        <CTRL-Z Anytime to Enter Interactive CMD Prompt>      |
----------------------------------------------------------------
 1) Drop Tables
 2) Create Tables
 3) Populate Tables
 4) Query Tables
 E) End/Exit
Choose:
4
Select a Query to Run:
1. View Customer Orders
2. View Supplier Product Summary
3. View Employee Sales Performance
Choose a query: 1

CUSTOMER_ID TRANSACTION_ID TOTAL_AMOUNT TRANSACTI
----------- -------------- ------------ ---------
          1           1001       250.75 03-OCT-24

Press Enter to continue...
```

Query 2:

```
============================================================
|                  Oracle All Inclusive Tool                 |
|            Main Menu - Select Desired Operation(s):        |
|       <CTRL-Z Anytime to Enter Interactive CMD Prompt>     |
------------------------------------------------------------
 1) Drop Tables
 2) Create Tables
 3) Populate Tables
 4) Query Tables
 E) End/Exit
Choose:
4
Select a Query to Run:
1. View Customer Orders
2. View Supplier Product Summary
3. View Employee Sales Performance
Choose a query: 2

SUPPLIER_ID SUPPLIER_FIRST_NAME  SUPPLIER_LAST_NAME   SUPPLIER_PHONE PRODUCT_ID PRODUCT_NAME PRODUCT_SIZE PRODUCT_PRICE STOCK_QUANTITY  PRODUCT_CATEGORY
----------- -------------------- -------------------- -------------- ---------- ------------ ------------ ------------- --------------- ----------------
 ########## John                 Doe                  4161231111     ########## T-shirt      Medium       20            150             Top

Press Enter to continue...
```

Query 3:

```
======================================================================
|                    Oracle All Inclusive Tool                       |
|              Main Menu - Select Desired Operation(s):              |
|        <CTRL-Z Anytime to Enter Interactive CMD Prompt>            |
----------------------------------------------------------------------
 1) Drop Tables
 2) Create Tables
 3) Populate Tables
 4) Query Tables
 E) End/Exit
Choose:
4
Select a Query to Run:
1. View Customer Orders
2. View Supplier Product Summary
3. View Employee Sales Performance
Choose a query: 3

EMPLOYEE_ID SALESCOUNT TOTALSALES
----------- ---------- ----------
          1          1     250.75

Press Enter to continue...
```

CRUD Operations:

```
| CRUD Operations                      |
========================================
| 1) Create Entry                      |
| 2) Read Entries                      |
| 3) Update Entry                      |
| 4) Delete Entry                      |
| E) Exit                              |
========================================
Choose an option:
```

Database Normalization

Normalization of the database/ Functional Dependencies:

Table	Primary Key	Functional Dependencies
Address	Address_ID	Address_ID → Country_Name, Province_Name, City_Name, Street_Address, Postal_Code
Supplier	Supplier_ID	Supplier_ID → First_Name, Last_Name, Phone_Number, Delivery_Schedule, Address_ID
Product	Product_ID	Product_ID → Product_Name, Product_Size, Product_Price, Stock_Quantity, Product_Category, Supplier_ID
Customer	Customer_ID	Customer_ID → First_Name, Last_Name, Phone_Number, Email_Address, Date_of_Birth, Address_ID
Sale	Transaction_ID	Transaction_ID → Transaction_Date, Total_Amount, Payment_Method, Customer_ID, Employee_ID
Product	Product_ID	Product_ID → Product_Name, Product_Size, Product_Price, Stock_Quantity, Product_Category, Supplier_ID
Employee	Employee_ID	Employee_ID → FirstName, LastName, Salary, PhoneNumber, JobRole, EmailAddress
Promotion	PromoCod	PromoCod → PromoName, ExpiryDate, PromotionPercent, PromotionAmount

Normalization / 3NF:

A table is in 3NF if it is in 2NF and does not have any transitive dependencies. This means that: It has no partial dependencies (all non-key attributes are fully functionally dependent on the primary key) and there is no attribute that depends on a non-key attribute.

Query 1:

The CustomerOrdersView can be verified as being in Third Normal Form (3NF) as follows. The primary key for the view is Customer_ID, and the functional dependencies are Customer_ID → Customer_Name, Email_Address, Total_Transactions, Total_Spent. Each non-key attribute depends entirely on the primary key, ensuring there are no partial

dependencies. Additionally, there are no transitive dependencies, as no non-key attribute depends on another non-key attribute. For example, Customer_Name and Email_Address are directly dependent on Customer_ID, not on each other. Similarly, the aggregated fields Total_Transactions and Total_Spent depend solely on Customer_ID. Therefore, the view satisfies the requirements of 3NF.

Query 2:

```
CREATE VIEW SupplierProductView AS
SELECT s.Supplier_ID,
        s.First_Name || ' ' || s.Last_Name AS Supplier_Name,
        p.Product_Category,
        COUNT(p.Product_ID) AS Product_Count,
        SUM(TO_NUMBER(p.Stock_Quantity)) AS Total_Stock
FROM Supplier s
JOIN Product p ON s.Supplier_ID = p.Supplier_ID
GROUP BY s.Supplier_ID, s.First_Name, s.Last_Name, p.Product_Category;
```

Analysis:

This view is derived from the Supplier and Product tables. The functional dependencies are as follows:

Supplier_ID → First_Name, Last_Name (from Supplier table).
Product_ID → Product_Category, Stock_Quantity, Supplier_ID (from Product table).
Supplier_ID, Product_Category → Product_Count, Total_Stock (aggregated fields).

Since all non-key attributes are fully dependent on the composite primary key (Supplier_ID, Product_Category), and there are no transitive dependencies, the view satisfies 3NF.

Query 3:

```
CREATE VIEW EmployeeSalesPerformanceView AS
SELECT e.Employee_ID,
e.FirstName || ' ' || e.LastName AS Employee_Name,
e.JobRole,
COUNT(s.Transaction_ID) AS Transactions_Handled,
SUM(s.Total_Amount) AS Total_Sales_Amount
FROM Employee e
JOIN Sale s ON e.Employee_ID = s.Employee_ID
GROUP BY e.Employee_ID, e.FirstName, e.LastName, e.JobRole;
```

Analysis:

This view is derived from the Employee and Sale tables. The functional dependencies are

as follows:

- Employee_ID → FirstName, LastName, JobRole (from Employee table).
- Employee_ID → Transactions_Handled, Total_Sales_Amount (aggregated fields from Sale table).

Since all non-key attributes are fully dependent on the primary key (Employee_ID), and there are no transitive dependencies, the view satisfies 3NF.

NOT in 3NF:

```
SELECT a.Country_Name,
    a.Province_Name,
    a.City_Name,
    a.Region_Name,
    COUNT(s.Supplier_ID) AS Supplier_Count,
    AVG(s.Delivery_Schedule) AS Average_Delivery_Schedule
FROM Supplier s
JOIN Address a ON s.Address_ID = a.Address_ID
GROUP BY a.Country_Name, a.Province_Name, a.City_Name, a.Region_Name
ORDER BY a.Country_Name, a.Province_Name, a.City_Name;
```

Analysis:

- **Primary Key:** The composite key remains Country_Name, Province_Name, City_Name.
- **New Functional Dependency:** Country_Name → Region_Name (indicating a transitive dependency).
- **Violation:** Region_Name is dependent on Country_Name alone, creating a transitive dependency through the primary key.

Bernstein's Algorithm to convert to 3NF:

1. **Minimize Functional Dependencies**
    a. Country_Name, Province_Name, City_Name → Supplier_Count, Average_Delivery_Schedule
    b. Country_Name → Region_Name (focus on removing this transitive dependency).
2. **Identify Candidate Keys**
    a. Confirm primary candidate key: (Country_Name, Province_Name, City_Name).
    b. Note secondary functional dependency key for decomposition: Country_Name.
3. **Decompose Tables to Remove Transitive Dependency**
    a. Contains: Country_Name, Region_Name.
    b. Ensures Country_Name → Region_Name.
4. **Location Table**: Maintains remaining attributes and dependencies.
    a. Contains: Country_Name, Province_Name, City_Name, Supplier_Count, Average_Delivery_Schedule.
    b. Preserves: Country_Name, Province_Name, City_Name → Supplier_Count, Average_Delivery_Schedule.

5. **Ensure Lossless Join and Dependency Preservation**
    a. **Lossless Join**: Verify that joining Region and Location on Country_Name reconstructs the original table without information loss.
    b. **Dependency Preservation**: Check that all original functional dependencies are maintained in the decomposed schema.

After applying Bernstein's Algorithm, the query has been decomposed into two 3NF compliant tables (Region and Location), ensuring no loss of information and maintaining all original dependencies. This normalization addresses the identified transitive dependency, improving the database schema's robustness and integrity.

Normalization/ BCNF:

Introduction

This report focuses on verifying the Boyce-Codd Normal Form (BCNF) compliance of our database schema, introducing a controlled violation of BCNF, and then using an algorithmic approach to restore compliance through decomposition. By following these steps, we aim to demonstrate a thorough understanding of normalization principles, the importance of BCNF in reducing redundancy, and ensuring robust database design.

Verification of BCNF Compliance

To begin, we examined each table within our database to confirm BCNF compliance. According to the BCNF rule, every non-trivial functional dependency X→Y must have X as a superkey. Below are the tables, their functional dependencies, and the BCNF verification for each.

1. **Address Table**
    - **Primary Key**: Address_ID
    - **Functional Dependencies**: Address_ID → Country_Name, Province_Name, City_Name, Street_Address, Postal_Code
    - **BCNF Verification**: Address_ID is the primary key and determines all other attributes. **This table is in BCNF**.
2. **Supplier Table**
    - **Primary Key**: Supplier_ID
    - **Functional Dependencies**: Supplier_ID → First_Name, Last_Name, Phone_Number, Delivery_Schedule, Address_ID
    - **BCNF Verification**: Supplier_ID is a superkey and determines all attributes. **This table is in BCNF**.
3. **Product Table**

    - **Primary Key**: Product_ID
    - **Functional Dependencies**: Product_ID → Product_Name, Product_Size, Product_Price, Stock_Quantity, Product_Category, Supplier_ID
    - **BCNF Verification**: Product_ID is the primary key and a superkey for all attributes. **This table is in BCNF**.
4. **Customer Table**
    - **Primary Key**: Customer_ID
    - **Functional Dependencies**: Customer_ID → First_Name, Last_Name, Phone_Number, Email_Address, Date_of_Birth, Address_ID
    - **BCNF Verification**: Customer_ID determines all attributes and is the primary key, satisfying BCNF. **This table is in BCNF**.
5. **Sale Table**
    - **Primary Key**: Transaction_ID
    - **Functional Dependencies**: Transaction_ID → Transaction_Date, Total_Amount, Payment_Method, Customer_ID, Employee_ID
    - **BCNF Verification**: Transaction_ID is a superkey for all other attributes. **This table is in BCNF**.
6. **Employee Table**
    - **Primary Key**: Employee_ID
    - **Functional Dependencies**: Employee_ID → FirstName, LastName, Salary, PhoneNumber, JobRole, EmailAddress
    - **BCNF Verification**: Employee_ID is the determinant for all attributes, making this table BCNF compliant. **This table is in BCNF**.
7. **Promotion Table**
    - **Primary Key**: PromoCode
    - **Functional Dependencies**: PromoCode → PromoName, ExpiryDate, PromotionPercent, PromotionAmount
    - **BCNF Verification**: PromoCode is a superkey for all attributes. **This table is in BCNF**.

Conclusion of Verification: After reviewing all tables and confirming that each non-trivial functional dependency has a superkey as its determinant, we conclude that **all tables are initially in BCNF**.

Introduction of a Non-BCNF Condition

To demonstrate the process of resolving BCNF violations, we intentionally introduced a non-BCNF dependency in the **Employee** table.

- **New Functional Dependency Introduced**: JobRole → Salary
- **Analysis**: This functional dependency indicates that a job role determines the salary, which violates BCNF because **JobRole is not a superkey**.

- **Resulting Violation**: Since JobRole is not a candidate key, the dependency JobRole → Salary does not satisfy BCNF requirements.

Resolving the BCNF Violation with Decomposition

To restore BCNF compliance, we used a decomposition algorithm. Below are the detailed steps and SQL commands.

Step 1: Identify the Violation

We identified that **JobRole → Salary** is a functional dependency where JobRole is not a superkey. Thus, this dependency violates BCNF.

Step 2: Decompose the Table

To resolve the violation, we decompose the **Employee** table into two separate tables:

1. **EmployeeRole**: This new table contains attributes (JobRole, Salary), capturing the relationship between JobRole and Salary independently.
2. **EmployeeDetails**: This table retains the other employee details, with a foreign key referencing the JobRole in **EmployeeRole**.

Step 3: SQL Commands for Decomposition

The decomposition into two tables is represented by the following SQL statements:

```
-- Creating the EmployeeRole table to store JobRole and Salary
CREATE TABLE EmployeeRole (
    JobRole VARCHAR2(50) PRIMARY KEY,
    Salary DECIMAL(10, 2)
);

-- Creating the EmployeeDetails table to store other employee
information, linking to EmployeeRole
CREATE TABLE EmployeeDetails (
    Employee_ID INT PRIMARY KEY,
    FirstName VARCHAR2(50) NOT NULL,
    LastName VARCHAR2(50) NOT NULL,
    PhoneNumber VARCHAR2(15),
    EmailAddress VARCHAR2(100) UNIQUE,
    JobRole VARCHAR2(50),
    FOREIGN KEY (JobRole) REFERENCES EmployeeRole(JobRole)
);
```

Verify BCNF Compliance Post-Decomposition

After decomposition, each table is verified to ensure BCNF compliance:

- **EmployeeRole Table**:
  - **Primary Key**: JobRole
  - **Functional Dependency**: JobRole → Salary
  - **BCNF Compliance**: JobRole is the primary key and determines Salary, meeting BCNF requirements.
- **EmployeeDetails Table**:
  - **Primary Key**: Employee_ID
  - **Foreign Key**: JobRole (references EmployeeRole)
  - **Functional Dependencies**: Employee_ID → FirstName, LastName, PhoneNumber, EmailAddress, JobRole
  - **BCNF Compliance**: Employee_ID determines all other attributes, including JobRole, without additional dependencies.

Conclusion of Decomposition: The decomposition resolves the BCNF violation by isolating the dependency on JobRole, ensuring that both **EmployeeRole** and **EmployeeDetails** tables now satisfy BCNF.

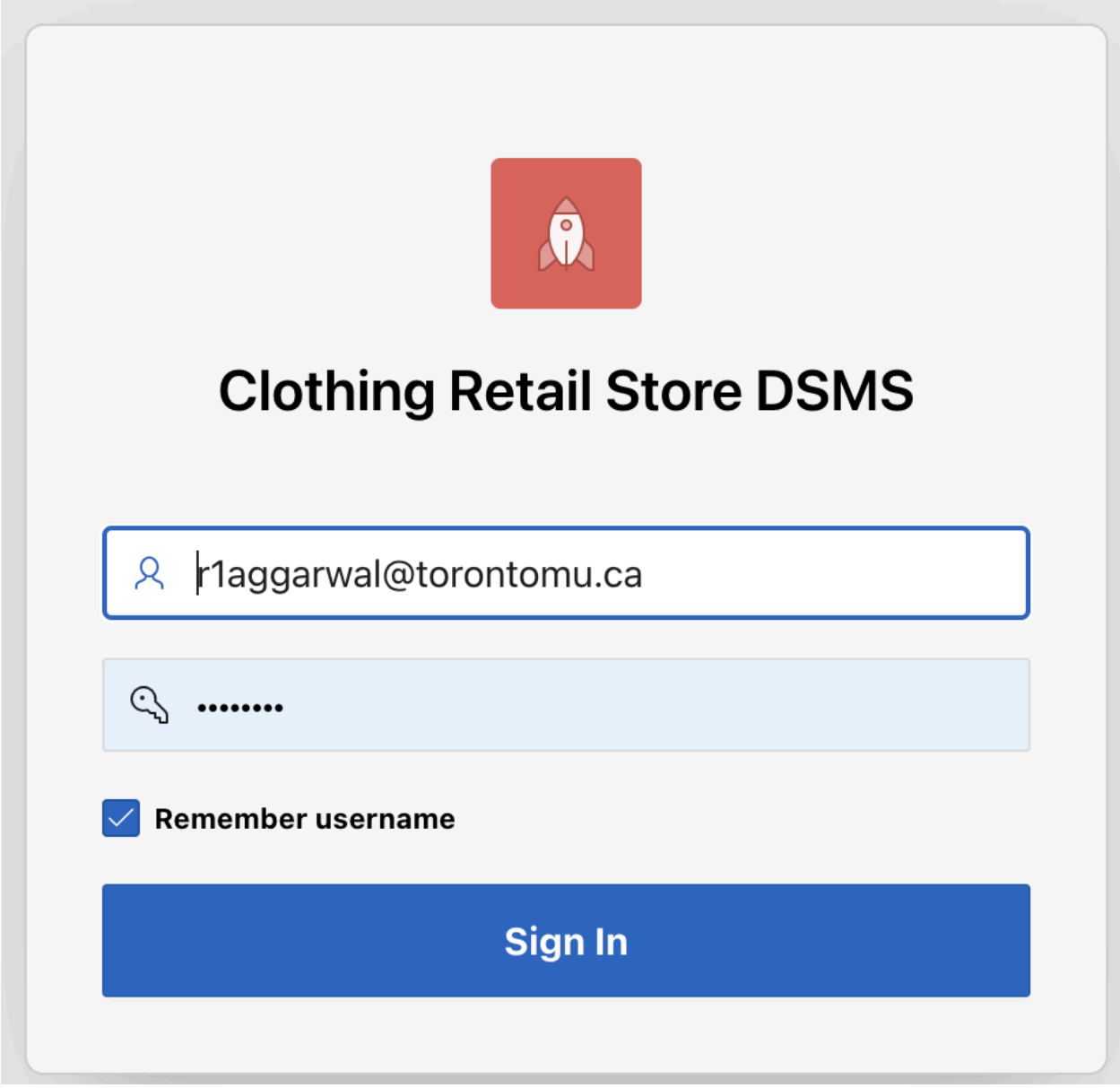
Conclusion

Through this exercise, we confirmed the initial BCNF compliance of all tables, introduced a non-BCNF condition, and resolved it using the BCNF decomposition algorithm. By following this systematic approach, we achieved a normalized database schema that reduces redundancy, enhances integrity, and prevents potential update anomalies.

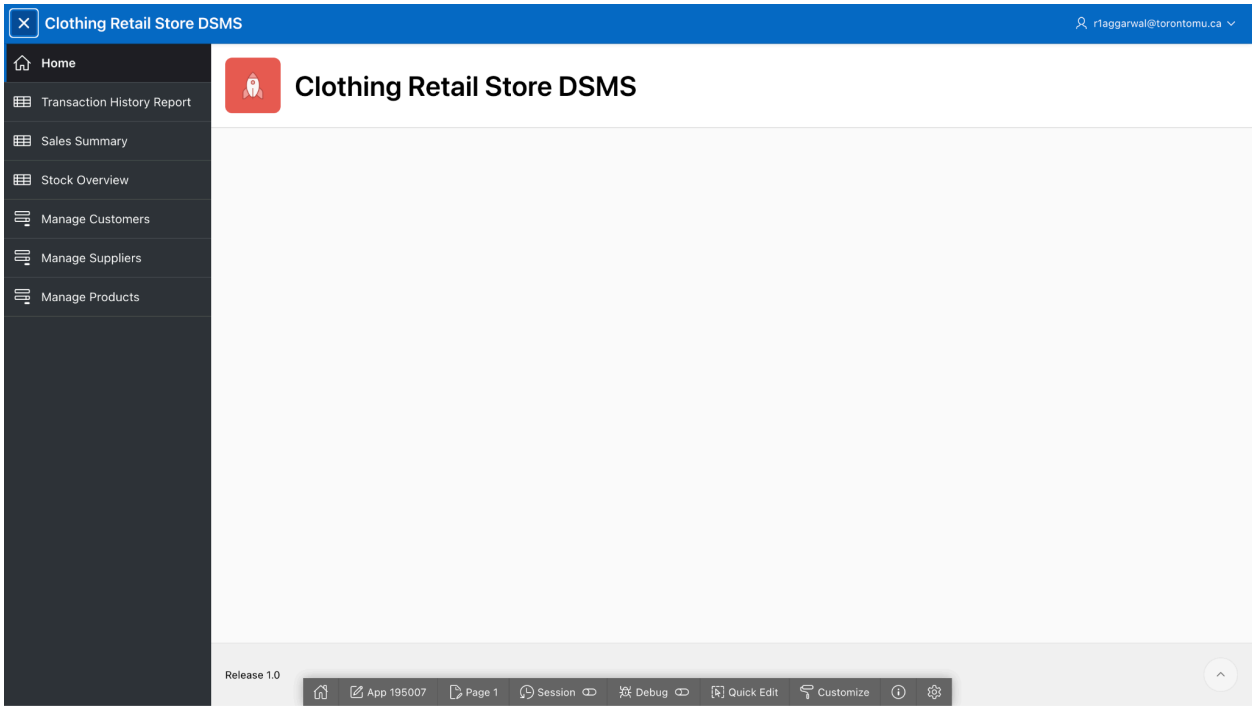
This report underscores the importance of BCNF in database design and highlights our proficiency in applying normalization techniques for optimal data management.

GUI

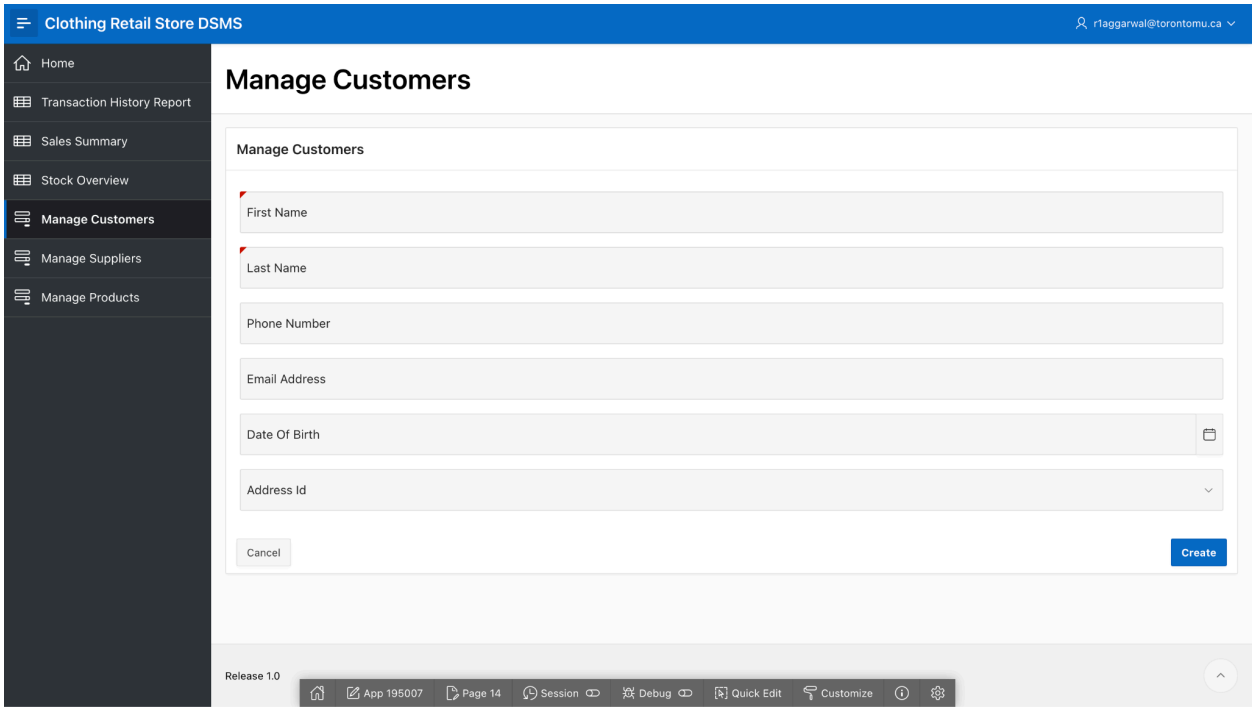
Initial Database login:



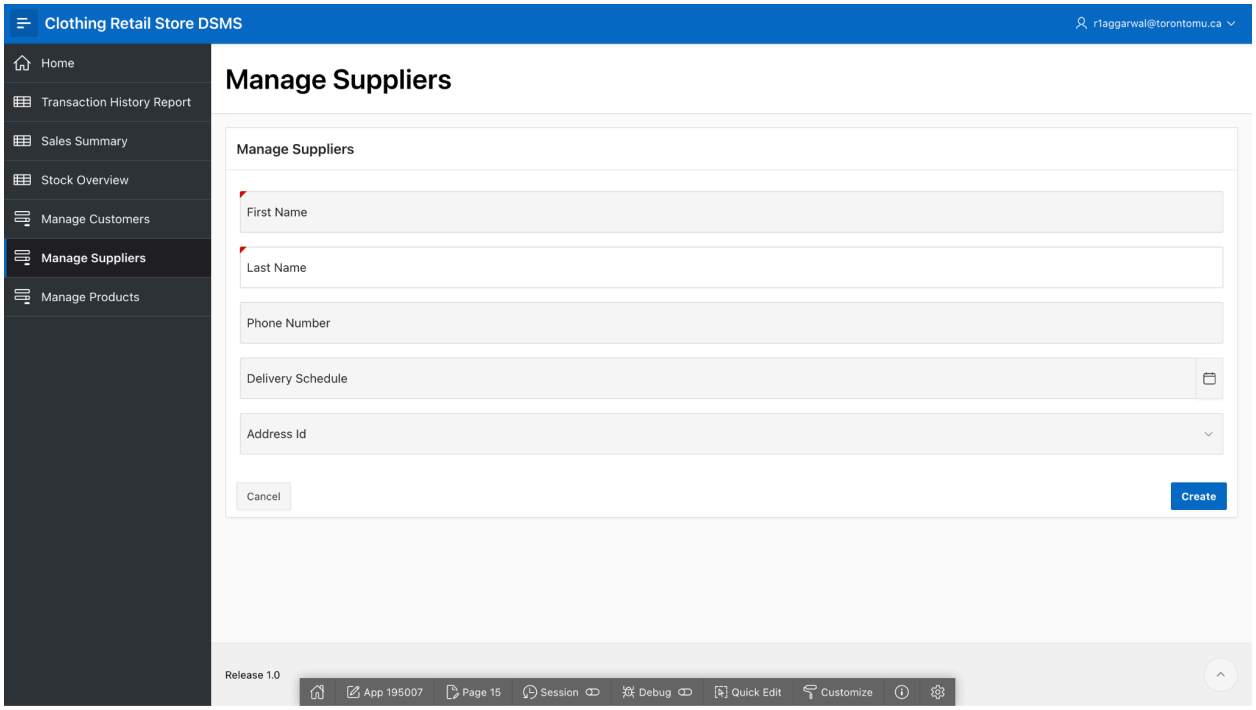
Menu:



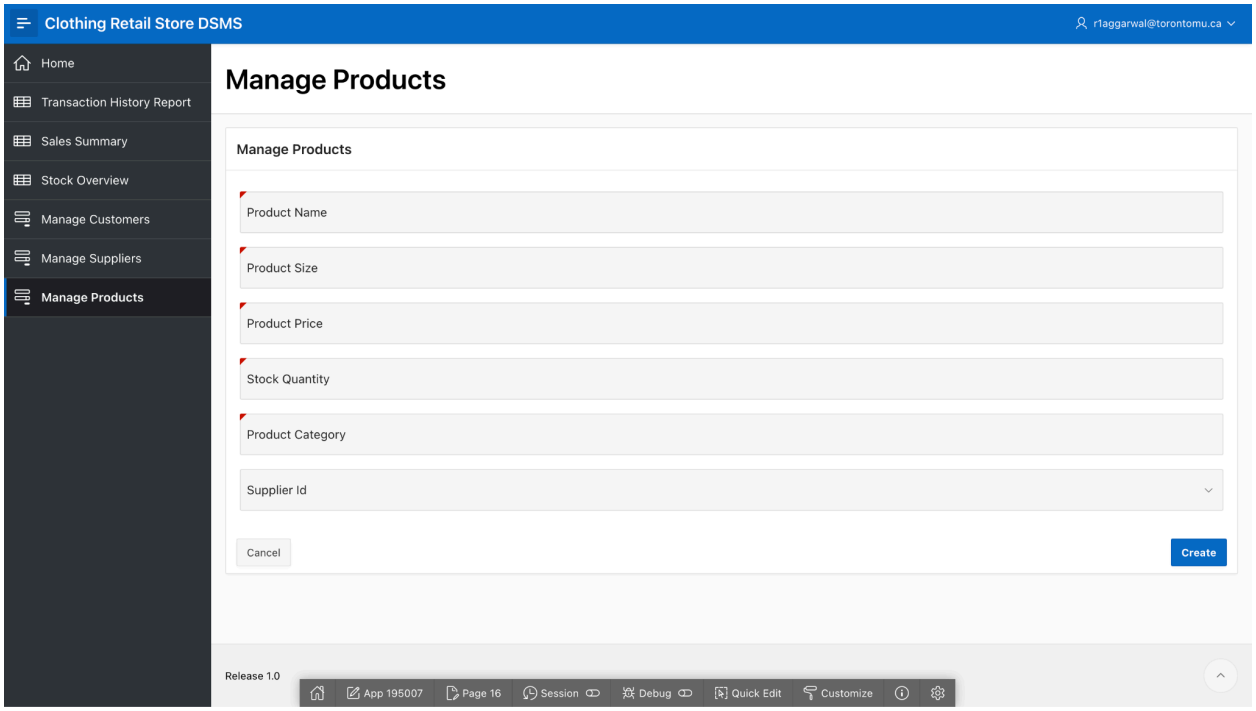
Manage Customers:



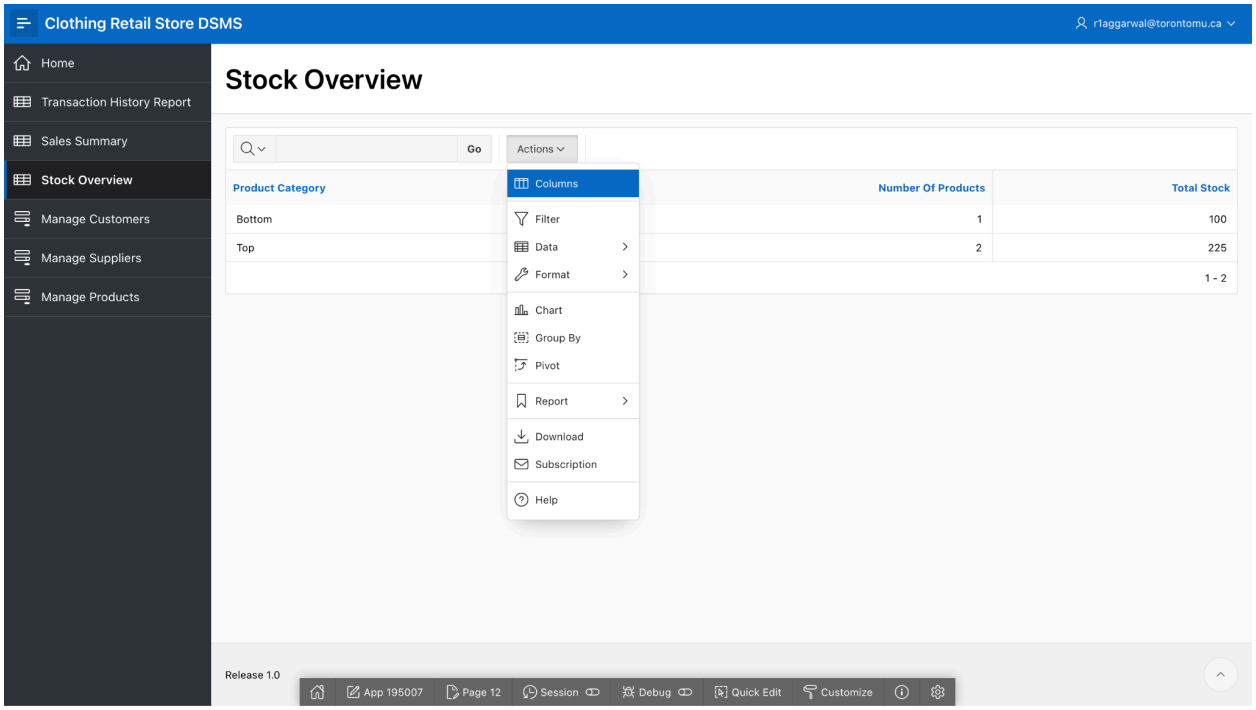
Manage Suppliers:



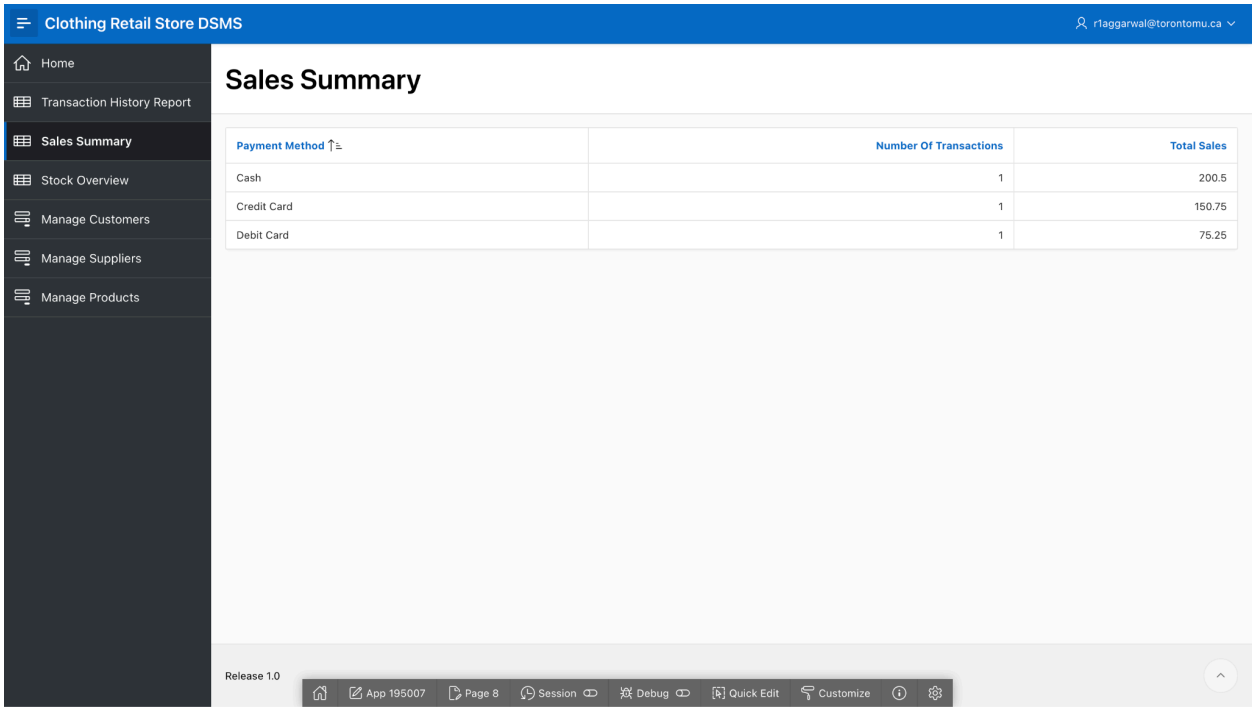
Manage Products:



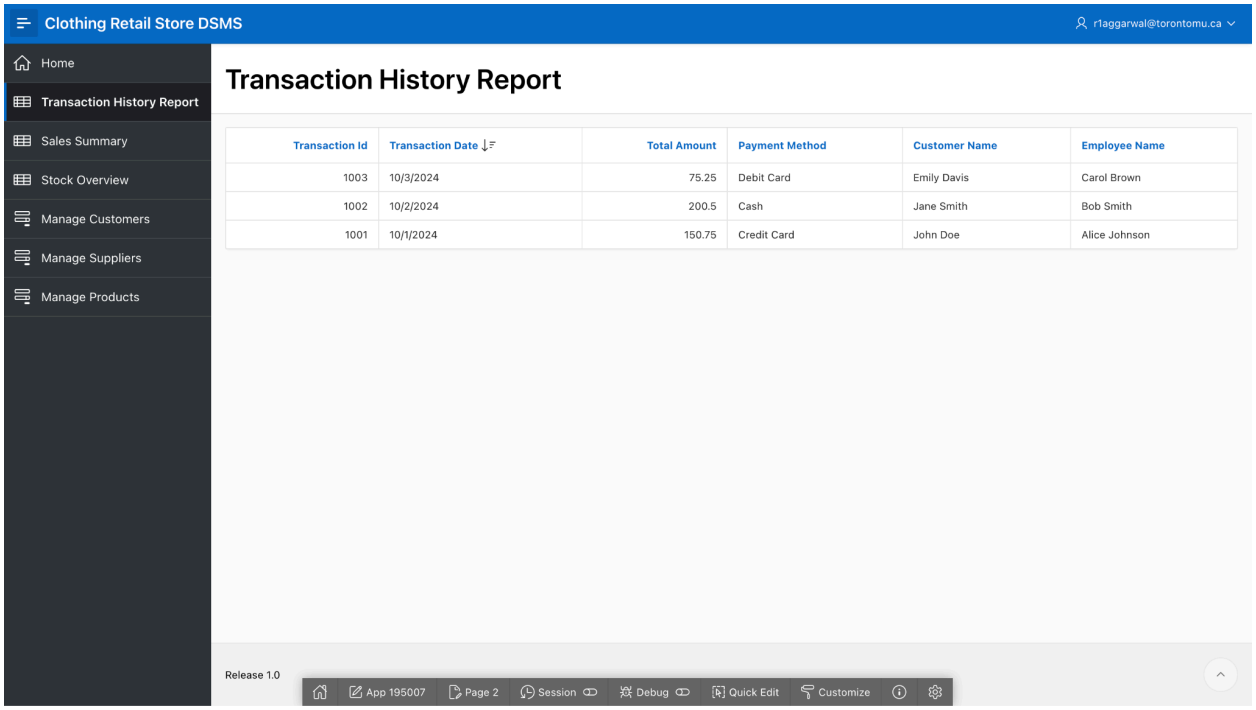
Stock Overview:



Sales Summary:



Transaction History Report:



Relational Algebra for Queries

Queries	Relational Algebra
CREATE VIEW CustomerOrdersView AS SELECT c.Customer_ID, c.First_Name || ' ' || c.Last_Name AS Customer_Name, c.Email_Address, COUNT(s.Transaction_ID) AS Total_Transactions, SUM(s.Total_Amount) AS Total_Spent FROM Customer c JOIN Sale s ON c.Customer_ID = s.Customer_ID GROUP BY c.Customer_ID, c.First_Name, c.Last_Name, c.Email_Address;	CustomerOrdersView = π_Customer_ID, Customer_Name, Email_Address, Total_Transactions, Total_Spent (γ_Customer_ID, (First_Name || ' ' || Last_Name) → Customer_Name, Email_Address; COUNT(Transaction_ID) → Total_Transactions, SUM(Total_Amount) → Total_Spent (Customer ⋈ Customer_ID = Customer_ID Sale))
CREATE VIEW SupplierProductView AS SELECT s.Supplier_ID, s.First_Name || ' ' || s.Last_Name AS Supplier_Name, p.Product_Category, COUNT(p.Product_ID) AS Product_Count, SUM(TO_NUMBER(p.Stock_Quantity)) AS	SupplierProductView = π_Supplier_ID, Supplier_Name, Product_Category, Product_Count, Total_Stock (γ_Supplier_ID, (First_Name || ' ' || Last_Name) → Supplier_Name, Product_Category; COUNT(Product_ID) → Product_Count, SUM(TO_NUMBER(Stock_Quantity)) →

Total_Stock FROM Supplier s JOIN Product p ON s.Supplier_ID = p.Supplier_ID GROUP BY s.Supplier_ID, s.First_Name, s.Last_Name, p.Product_Category; CREATE VIEW EmployeeSalesPerformanceView AS SELECT e.Employee_ID, e.FirstName || ' ' || e.LastName AS Employee_Name, e.JobRole, COUNT(s.Transaction_ID) AS Transactions_Handled, SUM(s.Total_Amount) AS Total_Sales_Amount FROM Employee e JOIN Sale s ON e.Employee_ID = s.Employee_ID GROUP BY e.Employee_ID, e.FirstName, e.LastName, e.JobRole;	Total_Stock (Supplier ⋈ Supplier_ID = Supplier_ID Product)) EmployeeSalesPerformanceView = π_Employee_ID, Employee_Name, JobRole, Transactions_Handled, Total_Sales_Amount (γ_Employee_ID, (FirstName || ' ' || LastName) → Employee_Name, JobRole; COUNT(Transaction_ID) → Transactions_Handled, SUM(Total_Amount) → Total_Sales_Amount (Employee ⋈ Employee_ID = Employee_ID Sale))
SELECT a.Country_Name, a.Province_Name, a.City_Name, COUNT(s.Supplier_ID) AS Supplier_Count, AVG(s.Delivery_Schedule) AS Average_Delivery_Schedule FROM Supplier s JOIN Address a ON s.Address_ID = a.Address_ID GROUP BY a.Country_Name, a.Province_Name, a.City_Name ORDER BY a.Country_Name, a.Province_Name, a.City_Name;	π_Country_Name, Province_Name, City_Name, Supplier_Count, Average_Delivery_Schedule (γ_Country_Name, Province_Name, City_Name; COUNT(Supplier_ID) → Supplier_Count, AVG(Delivery_Schedule) → Average_Delivery_Schedule (Supplier ⋈ Address_ID = Address_ID Address))
SELECT p.Product_ID, AVG(p.Product_Price) AS Average_Product_Price FROM Product p GROUP BY p.Product_ID;	π_Product_ID, Average_Product_Price (γ_Product_ID; AVG(Product_Price) → Average_Product_Price (Product))
SELECT Product_ID, Product_Name FROM Product MINUS	π_Product_ID, Product_Name (Product) - π_Product_ID, Product_Name (σ_Supplier_ID IS

SELECT Product_ID, Product_Name FROM Product WHERE Supplier_ID IS NOT NULL;	NOT NULL (Product))
SELECT s.Supplier_ID, s.First_Name || ' ' || s.Last_Name AS Supplier_Name, COUNT(DISTINCT p.Product_Category) AS Category_Count FROM Supplier s JOIN Product p ON s.Supplier_ID = p.Supplier_ID GROUP BY s.Supplier_ID, s.First_Name, s.Last_Name HAVING COUNT(DISTINCT p.Product_Category) > 1;	π_Supplier_ID, Supplier_Name, Category_Count (σ_Category_Count > 1 (γ_Supplier_ID, (First_Name || ' ' || Last_Name) → Supplier_Name; COUNT(DISTINCT Product_Category) → Category_Count (Supplier ⋈ Supplier_ID = Supplier_ID Product)))
SELECT c.Customer_ID, c.First_Name || ' ' || c.Last_Name AS Customer_Name FROM Customer c WHERE NOT EXISTS (SELECT 1 FROM Sale s WHERE s.Customer_ID = c.Customer_ID AND s.Transaction_Date >= ADD_MONTHS(SYSDATE, -12));	π_Customer_ID, Customer_Name (Customer - π_Customer_ID, (First_Name || ' ' || Last_Name) → Customer_Name (σ_Transaction_Date >= ADD_MONTHS(SYSDATE, -12) (Customer ⋈ Customer_ID = Customer_ID Sale)))
SELECT e.Employee_ID, e.FirstName || ' ' || e.LastName AS Employee_Name, STDDEV(s.Total_Amount) AS Sales_Amount_StdDev FROM Employee e JOIN Sale s ON e.Employee_ID = s.Employee_ID GROUP BY e.Employee_ID, e.FirstName, e.LastName HAVING STDDEV(s.Total_Amount) > 0;	π_Employee_ID, Employee_Name, Sales_Amount_StdDev (σ_Sales_Amount_StdDev > 0 (γ_Employee_ID, (FirstName || ' ' || LastName) → Employee_Name; STDDEV(Total_Amount) → Sales_Amount_StdDev (Employee ⋈ Employee_ID = Employee_ID Sale)))
SELECT Product_ID, Product_Name, 'Supplied' AS Status FROM Product	π_Product_ID, Product_Name, 'Supplied' AS Status (σ_Supplier_ID IS NOT NULL (Product)

WHERE Supplier_ID IS NOT NULL UNION SELECT Product_ID, Product_Name, 'Not Supplied' AS Status FROM Product WHERE Supplier_ID IS NULL;	) ∪ π_Product_ID, Product_Name, 'Not Supplied' AS Status (σ_Supplier_ID IS NULL (Product))

Conclusion

In conclusion, this lab demonstrated the effective use of SQL, Oracle, and UNIX in managing a clothing retail store database. Through queries and views, we efficiently retrieved and analyzed data for inventory, sales, and customer management. Oracle's database capabilities, combined with UNIX for automation and maintenance, highlighted the importance of a robust system in optimizing retail operations and decision-making.