Retail Business Database for "BuyX”

Higher National Diploma in Software Engineering

Data Management - 2

Coursework Report

24.1F/CO

Submitted By:

Index No.	Name
SANJULA W.M.M	COHNDSE241F-002
GUNARATHNA M. N	COHNDSE241F-007
DEWNAKA W.K.C	COHNDSE241F-023
BUCKSIMAR M.R	COHNDSE241F-037
RANASINGHE R.A.A.C	COHNDSE241F-070
ADHIKARI A.M.K.H.T	COHNDSE241F-085

Visit Git-hub: ManuraSanjula/ADBMS-NIBM (github.com) or https://github.com/ManuraSanjula/ADBMS-NIBM

School of Computing and Engineering

National Institute of Business Management

Colombo-7

Table of Contents

1. Introduction

Retail Business Database for buyX leverages the enterprise functionalities and robustness of Oracle Database Server. The goal is to design and implement a relational database system for a retail business focusing on managing product inventories, orders, suppliers, customer interactions, and detailed transaction histories. The database system will ensure scalability, high availability, and enterprise-grade security.

2. Database Platform Justification: Oracle Database Server

- Provides excellent performance while handling massive databases.
- Supports partitioning, PL/SQL for sophisticated, complex operations, and Real Application Clusters (RAC).
- Provides enterprise-grade access control, auditing, and encryption to safeguard sensitive data.
- Continuous availability can be ensured by data replication and backup/recovery solutions.
- Large community support.
- Provides future scalability through seamless integration with a variety of applications, including cloud services.

3. Requirements Analysis

- **User Management**: Registration and authentication for users, supporting unique emails and NICs. Manage multiple user roles (Admin, Manager, Driver, Customer, Supplier) and role-specific attributes.
- **Product Management**: Admins can manage categories, suppliers, and products. Products are linked to categories and suppliers with attributes like name, price, and stock level. Low stock alerts.
- **Order Processing**: Customers can place orders, manage order items, and view order details (date, total amount, status). The system calculates prices and tracks payments.
- **Payment Handling**: Support for payments via credit card, debit card, PayPal, and bank transfer. Payments are linked to orders and ensure the total matches the order amount.
- **Shipping Logistics**: Manage shipping records (carrier, tracking number, dates, costs) and assign drivers for delivery. Customers can track their orders.
- **Customer Reviews**: Allow customers to write reviews for purchased products, including ratings and text.
- **Shopping Cart Management**: Each customer has an active cart to add, update, or remove items, with automatic total calculation.
- **Address Management**: Customers and suppliers can manage shipping and billing addresses. Orders reference shipping addresses.
- **Reporting & Analytics**: Admins and managers can generate sales, inventory, and activity reports with real-time dashboards.

4. Database Design

The database consists of the following tables, designed with relationships, constraints, and security in mind:

User Table: Stores information about the User

```
CREATE TABLE Users (
    UserID NUMBER PRIMARY KEY,
    UserName VARCHAR2(100) UNIQUE NOT NULL,
    Password VARCHAR2(100) NOT NULL,
    NIC VARCHAR2(20) UNIQUE,
    Type VARCHAR2(20) CHECK (Type IN ('Admin', 'Manager', 'Driver', 'Customer','Supplier'))
);
```

Admin Table: Stores information about Admin.

```
CREATE TABLE Admin (
    AdminID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    FirstName VARCHAR2(100),
    LastName VARCHAR2(100),
    UserID NUMBER UNIQUE,
    Email VARCHAR2(100) CHECK (Email LIKE '%@%.%'),
    Phone VARCHAR2(20),
    Address VARCHAR2(200),
    City VARCHAR2(100),
    State VARCHAR2(100),
    ZipCode VARCHAR2(10),
    CONSTRAINT fk_admin_user FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

Manager Table: Stores information about Manager.

```
CREATE TABLE Manager (
    ManagerID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    FirstName VARCHAR2(100),
    LastName VARCHAR2(100),
    UserID NUMBER UNIQUE,
    Email VARCHAR2(100) CHECK (Email LIKE '%@%.%'),
    Phone VARCHAR2(20),
    Address VARCHAR2(200),
    City VARCHAR2(100),
    State VARCHAR2(100),
    ZipCode VARCHAR2(10),
    CONSTRAINT fk_manager_user FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

Driver Table: Stores information about Driver.

```
CREATE TABLE Driver (
    DriverID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    FirstName VARCHAR2(100),
    LastName VARCHAR2(100),
    UserID NUMBER UNIQUE,
    LicenseNumber VARCHAR2(50),
    Email VARCHAR2(100) CHECK (Email LIKE '%@%.%'),
    Phone VARCHAR2(20),
    Address VARCHAR2(200),
    City VARCHAR2(100),
    State VARCHAR2(100),
    ZipCode VARCHAR2(10),
    CONSTRAINT fk_driver_user FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

Category Table: Stores information about Different Product Categories.

```
CREATE TABLE Category (
    CategoryID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    CategoryName VARCHAR2(100) NOT NULL,
    Description VARCHAR2(500)
);
```

Supplier Table: Stores information about Supplier.

```
CREATE TABLE Supplier (
    SupplierID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    SupplierName VARCHAR2(100) NOT NULL,
    ContactName VARCHAR2(100),
    Email VARCHAR2(100) CHECK (Email LIKE '%@%.%'),
    Phone VARCHAR2(20),
    Address VARCHAR2(200),
    City VARCHAR2(100),
    State VARCHAR2(100),
    ZipCode VARCHAR2(10),
    UserID NUMBER UNIQUE,
    CONSTRAINT fk_supplier_user FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

Product Table: Stores information about Product.

```
CREATE TABLE Product (
    ProductID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    ProductName VARCHAR2(100) NOT NULL,
    CategoryID NUMBER NOT NULL,
    SupplierID NUMBER NOT NULL,
    Price NUMBER(10,2) NOT NULL CHECK (Price > 0),
```

```
    StockQuantity NUMBER NOT NULL CHECK (StockQuantity >= 0),
    Description VARCHAR2(1000),
    CONSTRAINT fk_product_category
        FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID),
    CONSTRAINT fk_product_supplier
        FOREIGN KEY (SupplierID) REFERENCES Supplier(SupplierID)
);
```

Customer Table: Stores information about Customer.

```
CREATE TABLE Customer (
    CustomerID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    FirstName VARCHAR2(50) NOT NULL,
    LastName VARCHAR2(50) NOT NULL,
    Email VARCHAR2(100) UNIQUE NOT NULL,
    Phone VARCHAR2(20),
    RegistrationDate DATE DEFAULT SYSDATE,
    UserID NUMBER UNIQUE,
    ShippingAddressID NUMBER UNIQUE,
    BillingAddressID NUMBER UNIQUE,
    CONSTRAINT fk_customer_user FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

Address Table: Stores information about Address.

```
CREATE TABLE Address (
    AddressID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    CustomerID NUMBER NOT NULL,
    Street VARCHAR2(200) NOT NULL,
    City VARCHAR2(100) NOT NULL,
    State VARCHAR2(100) NOT NULL,
    ZipCode VARCHAR2(10) NOT NULL,
```

```
    AddressType VARCHAR2(50) CHECK (AddressType IN ('Home', 'Work', 'Other')),
    CONSTRAINT fk_address_customer FOREIGN KEY (CustomerID) REFERENCES
Customer(CustomerID)
);
```

Alter the Customer Table .

```
ALTER TABLE Customer
    ADD (
        CONSTRAINT fk_customer_shipping_address
            FOREIGN KEY (ShippingAddressID)
            REFERENCES Address(AddressID),

        CONSTRAINT fk_customer_billing_address
            FOREIGN KEY (BillingAddressID)
            REFERENCES Address (AddressID)
    );
```

Orders Table: Stores information about Orders.

```
CREATE TABLE Orders (
    OrderID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    CustomerID NUMBER NOT NULL,
    OrderDate DATE DEFAULT SYSDATE,
    ShipDate DATE,
    ShippingAddress VARCHAR2(200),
    TotalAmount NUMBER(10,2) CHECK (TotalAmount >= 0),
    Status VARCHAR2(50) NOT NULL CHECK (Status IN ('Pending', 'Shipped', 'Delivered', 'Cancelled')),
    CONSTRAINT fk_orders_customer FOREIGN KEY (CustomerID) REFERENCES
Customer(CustomerID)
);
```

Order Item Table: Stores information about Order Item.

```
CREATE TABLE OrderItem (
    OrderItemID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
```

```
    OrderID NUMBER NOT NULL,
    ProductID NUMBER NOT NULL,
    Quantity NUMBER NOT NULL CHECK (Quantity > 0),
    UnitPrice NUMBER(10,2) NOT NULL CHECK (UnitPrice > 0),
    TotalPrice NUMBER GENERATED ALWAYS AS (Quantity * UnitPrice) VIRTUAL,
    CONSTRAINT fk_orderitem_order FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
    CONSTRAINT fk_orderitem_product FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);
```

Payment Table: Stores information about Payment.

```
CREATE TABLE Payment (
    PaymentID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    OrderID NUMBER NOT NULL,
    PaymentDate DATE DEFAULT SYSDATE,
    PaymentMethod VARCHAR2(50) NOT NULL CHECK (PaymentMethod IN ('Credit Card', 'Debit Card', 'PayPal', 'Bank Transfer')),
    Amount NUMBER(10,2) NOT NULL CHECK (Amount >= 0),
    CONSTRAINT fk_payment_order FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)
);
```

Shipping Table: Stores information about Shipping.

```
CREATE TABLE Shipping (
    ShippingID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    OrderID NUMBER NOT NULL,
    Carrier VARCHAR2(100),
    TrackingNumber VARCHAR2(50),
    ShippedDate DATE,
    DeliveredDate DATE,
    ShippingCost NUMBER(10, 2),
    CONSTRAINT fk_shipping_order FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)
);
```

Review Table: Stores information about Review.

```
CREATE TABLE Review (
   ReviewID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
   ProductID NUMBER NOT NULL,
   CustomerID NUMBER NOT NULL,
   Rating NUMBER CHECK (Rating BETWEEN 1 AND 5),
   ReviewText VARCHAR2(1000),
   ReviewDate DATE DEFAULT SYSDATE,
   CONSTRAINT fk_review_product FOREIGN KEY (ProductID) REFERENCES Product(ProductID),
   CONSTRAINT fk_review_customer FOREIGN KEY (CustomerID) REFERENCES
Customer(CustomerID)
);
```

Cart Table: Stores information about Cart.

```
CREATE TABLE Cart (
   CartID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
   CustomerID NUMBER NOT NULL,
   CONSTRAINT fk_cart_customer FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
);
```

Cart Item Table: Stores information about Cart Item.

```
CREATE TABLE CartItem (
   CartItemID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
   CartID NUMBER NOT NULL,
   ProductID NUMBER NOT NULL,
   Quantity NUMBER NOT NULL,
   CONSTRAINT fk_cartitem_cart FOREIGN KEY (CartID) REFERENCES Cart(CartID),
   CONSTRAINT fk_cartitem_product FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);
```

5. Database Tables and Relationships

Tables:

Customer, Product, Supplier, Order, Order Item, Shipping, Review, Cart, Cart Item, User, Admin, Manager, Driver

Complex Relationships:

• User - **User Type: One-to-One (1:1)**

- **Users - Admin:** Each user can be an Admin, linked uniquely to the `admin` table.
- **Users - Manager:** Each user can be a manager, linked uniquely to the `Manager` table.
- **Users - Driver:** Each user can be a Driver, linked uniquely to the `Driver` table.
- **Users - Customer:** Each user can be a customer, linked uniquely to the `Customer` table.
- **Users - Supplier:** Each user can be a Supplier, linked uniquely to the `Supplier` table.

• Customer - **Order: One-to-Many (1) -** A customer can place multiple orders over time, each uniquely linked to that customer.

• Order - **Order Item: One-to-Many (1) -** Each order can contain multiple items. Advanced indexing will be implemented for faster querying.

• Order **Item - Product: Many-to-One (N:1)** - Each order item relates to a single product, while a product can be part of multiple order items across different orders.

• Product - **Supplier: Many-to-One (N:1)** - Each product is provided by a single supplier, but a supplier can supply many products.

• Order - **Shipping: One-to-One (1:1)** - Each order corresponds to one shipping record, ensuring efficient tracking.

• Customer - **Cart: One-to-One (1:1)** - Each customer can have one active cart linked to their account.

• Cart - **Cart Item: One-to-Many (1) -** A cart contains multiple cart items.

• Cart **Item - Product: Many-to-One (N:1)** - Each cart item relates to a single product, but a product can be part of different customers' carts.

• Customer - **Review: One-to-Many (1) -** Each customer can write multiple reviews, each associated with that customer.

• Product - **Review: One-to-Many (1) -** Each product can have multiple reviews.

• Customer - **Address: One-to-Many (1) -** Each customer can have multiple addresses, designated as shipping and billing addresses.

• Order - **Payment: One-to-Many (1) -** Each order can have multiple payment records, depending on the payment method and transactions.

• Product - **Category: Many-to-One (N:1)** - Each product belongs to a single category, while a category can encompass multiple products.

6. Database Administrator (DBA)

Roles and Responsibilities:

- Maintenance and Monitoring: Regular monitoring of database health using tools like AWS CloudWatch and SQL query performance analyzers.
- Capacity Planning: Perform capacity planning to prepare for growth in user base and data size. Implement auto-scaling policies in the cloud to expand resources as needed.
- User Access Management: Define and manage roles using role-based access control (RBAC). Enforce the principle of least privilege, ensuring users only have access to the resources required for their roles.
- Performance Optimization: Use indexes, partitioning, and in-memory caching for optimizing performance. Identify slow queries using query optimization tools.
- Security Enforcement: Establish security baselines, encryption policies, and authentication protocols to maintain database security.

7. Backup Plan

Backup Strategy:

- Full Backups: Scheduled daily, encompassing all data, schemas, and indexes.
- Incremental Backups: Performed every 4 hours to cover changes since the last backup, reducing storage overhead and downtime during recovery.
- Transactional Log Backup: Back up transaction logs every 15 minutes to ensure point-in-time recovery.

Storage Options:

- Utilize multi-region cloud storage to enhance redundancy and resilience. Each backup will be stored across multiple regions for disaster resilience.
- On-premises Copies: Critical backups will also be archived in a secure on-premises storage system.

Disaster Recovery Plan:

- Recovery Point Objective (RPO): Set to 30 minutes to ensure minimal data loss.
- Recovery Time Objective (RTO): Less than 2 hours to ensure fast recovery in case of a failure.
- Implement AWS CloudEndure for continuous data replication to secondary regions.

8. Logical and Physical Database Structure

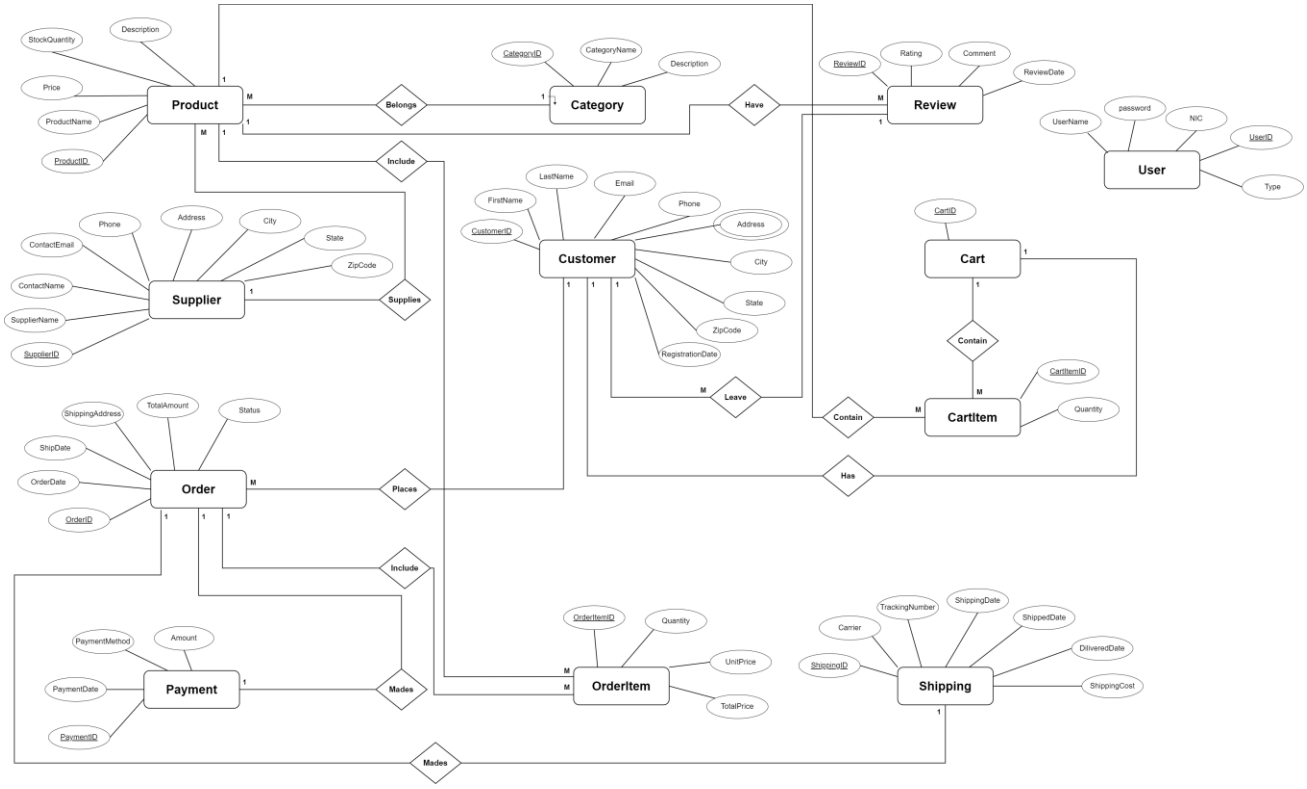


Figure 1

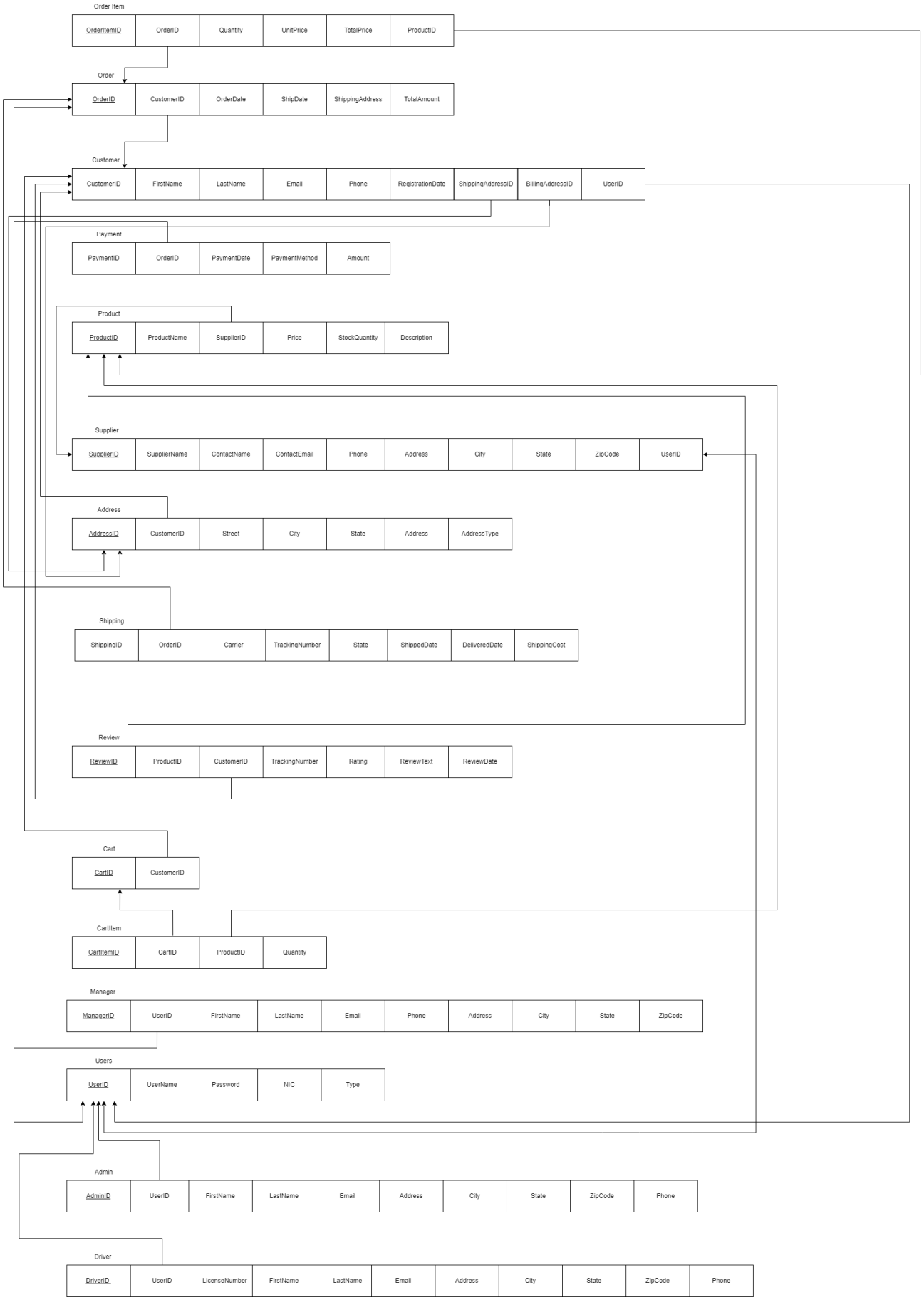

Order Item
OrderItemID
OrderID
Quantity
UnitPrice
TotalPrice
ProductID
Order
OrderID
CustomerID
OrderDate
ShipDate
ShippingAddress
TotalAmount
Customer
CustomerID
FirstName
LastName
Email
Phone
RegistrationDate
ShippingAddressID
BillingAddressID
UserID
Payment
PaymentID
OrderID
PaymentDate
PaymentMethod
Amount
Product
ProductID
ProductName
SupplierID
Price
StockQuantity
Description
Supplier
SupplierID
SupplierName
ContactName
ContactEmail
Phone
Address
City
State
ZipCode
UserID
Address
AddressID
CustomerID
Street
City
State
Address
AddressType
Shipping
ShippingID
OrderID
Carrier
TrackingNumber
State
ShippedDate
DeliveredDate
ShippingCost
Review
ReviewID
ProductID
CustomerID
TrackingNumber
Rating
ReviewText
ReviewDate
Cart
CartID
CustomerID
CartItem
CartItemID
CartID
ProductID
Quantity
Manager
ManagerID
UserID
FirstName
LastName
Email
Phone
Address
City
State
ZipCode
Users
UserID
UserName
Password
NIC
Type
Admin
AdminID
UserID
FirstName
LastName
Email
Address
City
State
ZipCode
Phone
Driver
DriverID
UserID
LicenseNumber
FirstName
LastName
Email
Address
City
State
ZipCode
Phone

Figure 2

9. Oracle PL/SQL Programs for CRUD Operations

Visit Git-hub: ManuraSanjula/ADBMS-NIBM (github.com) or https://github.com/ManuraSanjula/ADBMS-NIBM

10. Business Reports

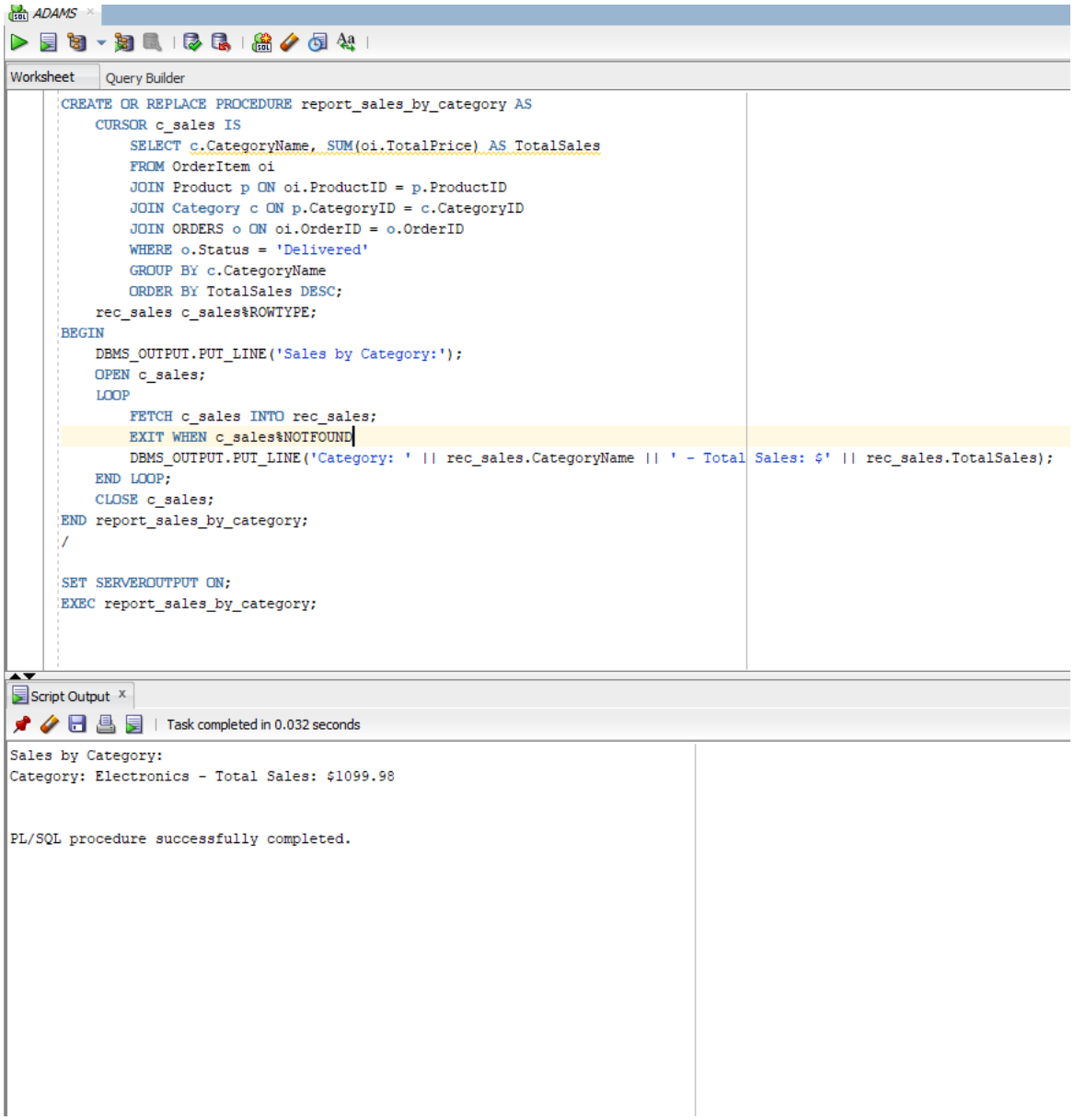
10.1 report_sales_by_category

```
CREATE OR REPLACE PROCEDURE report_sales_by_category AS
    CURSOR c_sales IS
        SELECT c.CategoryName, SUM(oi.TotalPrice) AS TotalSales
        FROM OrderItem oi
        JOIN Product p ON oi.ProductID = p.ProductID
        JOIN Category c ON p.CategoryID = c.CategoryID
        JOIN ORDERS o ON oi.OrderID = o.OrderID
        WHERE o.Status = 'Delivered'
        GROUP BY c.CategoryName
        ORDER BY TotalSales DESC;
    rec_sales c_sales%ROWTYPE;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Sales by Category:');
    OPEN c_sales;
    LOOP
        FETCH c_sales INTO rec_sales;
        EXIT WHEN c_sales%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Category: ' || rec_sales.CategoryName || ' - Total Sales: $' ||
rec_sales.TotalSales);
    END LOOP;
```

```
    CLOSE c_sales;

END report_sales_by_category;

/
```



report_sales_by_category 1

```
Sales by Category:
Category: Electronics - Total Sales: $1099.98


PL/SQL procedure successfully completed.
```

report_sales_by_category 2

10.2 report_top_customers

```
CREATE OR REPLACE PROCEDURE report_top_customers AS
   CURSOR c_top_customers IS
      SELECT c.FirstName || ' ' || c.LastName AS CustomerName, SUM(o.TotalAmount) AS
TotalPurchases
      FROM Customer c
      JOIN ORDERS o ON c.CustomerID = o.CustomerID
      WHERE o.Status = 'Delivered'
      GROUP BY c.FirstName, c.LastName
      HAVING SUM(o.TotalAmount) > 0
      ORDER BY TotalPurchases DESC
      FETCH FIRST 10 ROWS ONLY;
   rec_customer c_top_customers%ROWTYPE;
BEGIN
```

```
    DBMS_OUTPUT.PUT_LINE('Top 10 Customers by Purchase Amount:');
    OPEN c_top_customers;
    LOOP
        FETCH c_top_customers INTO rec_customer;
        EXIT WHEN c_top_customers%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Customer: ' || rec_customer.CustomerName || ' - Total
Purchases: $' || rec_customer.TotalPurchases);
    END LOOP;
    CLOSE c_top_customers;
END report_top_customers;
/


SET SERVEROUTPUT ON;
EXEC report_top_customers
```

Worksheet | Query Builder

```
CREATE OR REPLACE PROCEDURE report_top_customers AS
    CURSOR c_top_customers IS
        SELECT c.FirstName || ' ' || c.LastName AS CustomerName, SUM(o.TotalAmount) AS TotalPurchases
        FROM Customer c
        JOIN ORDERS o ON c.CustomerID = o.CustomerID
        WHERE o.Status = 'Delivered'
        GROUP BY c.FirstName, c.LastName
        HAVING SUM(o.TotalAmount) > 0
        ORDER BY TotalPurchases DESC
        FETCH FIRST 10 ROWS ONLY;
    rec_customer c_top_customers%ROWTYPE;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Top 10 Customers by Purchase Amount:');
    OPEN c_top_customers;
    LOOP
        FETCH c_top_customers INTO rec_customer;
        EXIT WHEN c_top_customers%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Customer: ' || rec_customer.CustomerName || ' - Total Purchases: $' || rec_customer.TotalPurchases);
    END LOOP;
    CLOSE c_top_customers;
END report_top_customers;
/

SET SERVEROUTPUT ON;
EXEC report_top_customers;
```

Script Output | Task completed in 0.066 seconds

```
Top 10 Customers by Purchase Amount:
Customer: Jane Smith - Total Purchases: $549.99


PL/SQL procedure successfully completed.
```

report_top_customers 1

```
Top 10 Customers by Purchase Amount:
Customer: Jane Smith - Total Purchases: $549.99


PL/SQL procedure successfully completed.
```

report_top_customers 2

10.3 report_low_inventory

```
CREATE OR REPLACE PROCEDURE report_low_inventory(p_threshold IN NUMBER) AS
    CURSOR c_low_inventory IS
        SELECT p.ProductName, p.StockQuantity
        FROM Product p
        WHERE p.StockQuantity < p_threshold
        ORDER BY p.StockQuantity ASC;
    rec_product c_low_inventory%ROWTYPE;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Inventory Status Report (Stock below ' || p_threshold || '):');
    OPEN c_low_inventory;
    LOOP
        FETCH c_low_inventory INTO rec_product;
        EXIT WHEN c_low_inventory%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Product: ' || rec_product.ProductName || ' - Stock Quantity: ' ||
rec_product.StockQuantity);
    END LOOP;
    CLOSE c_low_inventory;
END report_low_inventory;
/


SET SERVEROUTPUT ON;
EXEC report_low_inventory(50);
```

Worksheet | Query Builder

```
CREATE OR REPLACE PROCEDURE report_low_inventory(p_threshold IN NUMBER) AS
    CURSOR c_low_inventory IS
        SELECT p.ProductName, p.StockQuantity
        FROM Product p
        WHERE p.StockQuantity < p_threshold
        ORDER BY p.StockQuantity ASC;
    rec_product c_low_inventory%ROWTYPE;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Inventory Status Report (Stock below ' || p_threshold || '):');
    OPEN c_low_inventory;
    LOOP
        FETCH c_low_inventory INTO rec_product;
        EXIT WHEN c_low_inventory%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Product: ' || rec_product.ProductName || ' - Stock Quantity: ' || rec_product.StockQuantity);
    END LOOP;
    CLOSE c_low_inventory;
END report_low_inventory;
/

SET SERVEROUTPUT ON;
EXEC report_low_inventory(50);
```

Script Output | Task completed in 0.042 seconds

```
Inventory Status Report (Stock below 50):
Product: Smartphone - Stock Quantity: 45


PL/SQL procedure successfully completed.
```

report_low_inventory 1

```
Inventory Status Report (Stock below 50):
Product: Smartphone - Stock Quantity: 45


PL/SQL procedure successfully completed.
```

report_low_inventory 2

10.4 report_monthly_sales

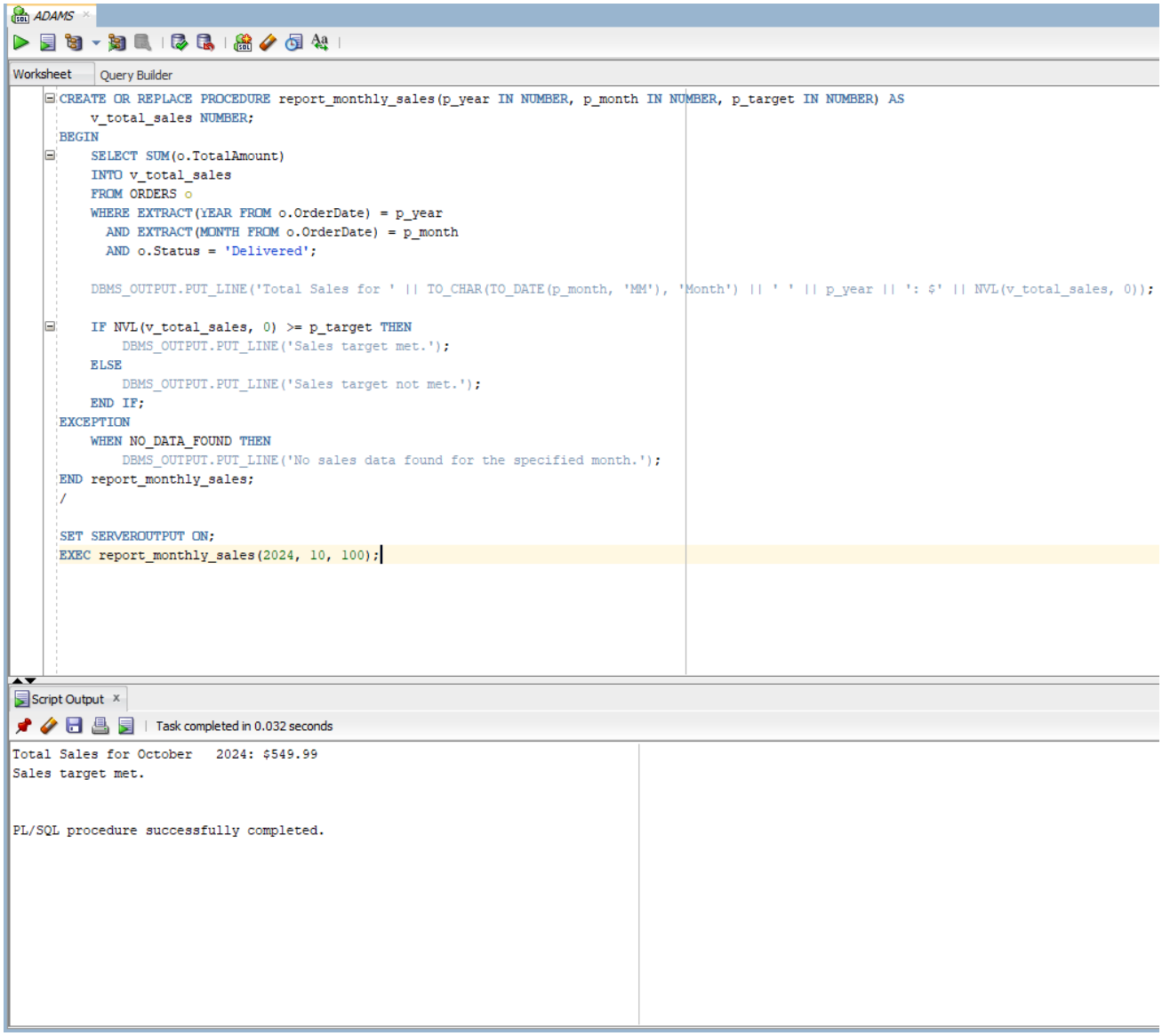
```
CREATE OR REPLACE PROCEDURE report_monthly_sales(p_year IN NUMBER, p_month IN
NUMBER, p_target IN NUMBER) AS
   v_total_sales NUMBER;
BEGIN
   SELECT SUM(o.TotalAmount)
   INTO v_total_sales
   FROM ORDERS o
   WHERE EXTRACT(YEAR FROM o.OrderDate) = p_year
     AND EXTRACT(MONTH FROM o.OrderDate) = p_month
     AND o.Status = 'Delivered';

   DBMS_OUTPUT.PUT_LINE('Total Sales for ' || TO_CHAR(TO_DATE(p_month, 'MM'), 'Month') || ' ' ||
p_year || ': $' || NVL(v_total_sales, 0));

   IF NVL(v_total_sales, 0) >= p_target THEN
      DBMS_OUTPUT.PUT_LINE('Sales target met.');
   ELSE
      DBMS_OUTPUT.PUT_LINE('Sales target not met.');
   END IF;
EXCEPTION
```

```
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No sales data found for the specified month.');
END report_monthly_sales;
/

SET SERVEROUTPUT ON;
EXEC report_monthly_sales(2024, 10, 100);
```



report_monthly_sales 1

```
Total Sales for October   2024: $549.99
Sales target met.


PL/SQL procedure successfully completed.
```

report_monthly_sales 2

10.5 report_product_sales_trend

```
CREATE OR REPLACE PROCEDURE report_product_sales_trend(p_product_id IN NUMBER)
AS
   TYPE sales_array IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
   v_sales sales_array;
BEGIN
   -- Populate sales per month
   FOR i IN 1..12 LOOP
      SELECT NVL(SUM(oi.TotalPrice), 0)
      INTO v_sales(i)
      FROM OrderItem oi
      JOIN ORDERS o ON oi.OrderID = o.OrderID
      WHERE oi.ProductID = p_product_id
        AND EXTRACT(MONTH FROM o.OrderDate) = i
        AND o.Status = 'Delivered';
   END LOOP;
```

```
    DBMS_OUTPUT.PUT_LINE('Sales Trend for Product ID ' || p_product_id || ':');

    FOR i IN 1..12 LOOP
        DBMS_OUTPUT.PUT_LINE('Month ' || i || ': $' || v_sales(i) || ' ' ||
            CASE
                WHEN v_sales(i) > 10000 THEN 'High'
                WHEN v_sales(i) BETWEEN 5000 AND 10000 THEN 'Medium'
                ELSE 'Low'
            END);
    END LOOP;
END report_product_sales_trend;
/

SET SERVEROUTPUT ON;
EXEC report_product_sales_trend(1);
```

ADAMS

Worksheet | Query Builder

```
CREATE OR REPLACE PROCEDURE report_product_sales_trend(p_product_id IN NUMBER) AS
    TYPE sales_array IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
    v_sales sales_array;
BEGIN
    -- Populate sales per month
    FOR i IN 1..12 LOOP
        SELECT NVL(SUM(oi.TotalPrice), 0)
        INTO v_sales(i)
        FROM OrderItem oi
        JOIN ORDERS o ON oi.OrderID = o.OrderID
        WHERE oi.ProductID = p_product_id
          AND EXTRACT(MONTH FROM o.OrderDate) = i
          AND o.Status = 'Delivered';
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('Sales Trend for Product ID ' || p_product_id || ':');

    FOR i IN 1..12 LOOP
        DBMS_OUTPUT.PUT_LINE('Month ' || i || ': $' || v_sales(i) || ' ' ||
            CASE
                WHEN v_sales(i) > 10000 THEN 'High'
                WHEN v_sales(i) BETWEEN 5000 AND 10000 THEN 'Medium'
                ELSE 'Low'
            END);
    END LOOP;
END report_product_sales_trend;
/
```

Script Output

Task completed in 0.029 seconds

```
Sales Trend for Product ID 1:
Month 1: $0 Low
Month 2: $0 Low
Month 3: $0 Low
Month 4: $0 Low
Month 5: $0 Low
Month 6: $0 Low
Month 7: $0 Low
Month 8: $0 Low
Month 9: $0 Low
Month 10: $1099.98 Low
Month 11: $0 Low
Month 12: $0 Low


PL/SQL procedure successfully completed.
```

report_product_sales_trend 1

```
Sales Trend for Product ID 1:
Month 1: $0 Low
Month 2: $0 Low
Month 3: $0 Low
Month 4: $0 Low
Month 5: $0 Low
Month 6: $0 Low
Month 7: $0 Low
Month 8: $0 Low
Month 9: $0 Low
Month 10: $1099.98 Low
Month 11: $0 Low
Month 12: $0 Low


PL/SQL procedure successfully completed.
```

report_product_sales_trend 2