**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)](https://github.com/ManuraSanjula/ADBMS-NIBM) or <https://github.com/ManuraSanjula/ADBMS-NIBM>



**School of Computing and Engineering**

**National Institute of Business Management**

**Colombo-7**

Table of Contents

[1. Introduction 4](#_Toc179300717)

[2. Database Platform Justification: Oracle Database Server 4](#_Toc179300718)

[3. Requirements Analysis 4](#_Toc179300719)

[4. Database Design 5](#_Toc179300720)

[User Table: Stores information about the User 5](#_Toc179300721)

[Admin Table: Stores information about Admin. 5](#_Toc179300722)

[Manager Table: Stores information about Manager. 6](#_Toc179300723)

[Driver Table: Stores information about Driver. 6](#_Toc179300724)

[Category Table: Stores information about Different Product Categories. 7](#_Toc179300725)

[Supplier Table: Stores information about Supplier. 7](#_Toc179300726)

[Product Table: Stores information about Product. 7](#_Toc179300727)

[Customer Table: Stores information about Customer. 8](#_Toc179300728)

[Address Table: Stores information about Address. 8](#_Toc179300729)

[Alter the Customer Table . 9](#_Toc179300730)

[Orders Table: Stores information about Orders. 9](#_Toc179300731)

[Order Item Table: Stores information about Order Item. 9](#_Toc179300732)

[Payment Table: Stores information about Payment. 10](#_Toc179300733)

[Shipping Table: Stores information about Shipping. 10](#_Toc179300734)

[Review Table: Stores information about Review. 11](#_Toc179300735)

[Cart Table: Stores information about Cart. 11](#_Toc179300736)

[Cart Item Table: Stores information about Cart Item. 11](#_Toc179300737)

[5. Database Tables and Relationships 12](#_Toc179300738)

[Tables: 12](#_Toc179300739)

[Complex Relationships: 12](#_Toc179300740)

[6. Database Administrator (DBA) 13](#_Toc179300741)

[Roles and Responsibilities: 13](#_Toc179300742)

[7. Backup Plan 13](#_Toc179300743)

[Backup Strategy: 13](#_Toc179300744)

[Storage Options: 13](#_Toc179300745)

[Disaster Recovery Plan: 13](#_Toc179300746)

[8. Logical and Physical Database Structure 14](#_Toc179300747)

[9. Oracle PL/SQL Programs for CRUD Operations 16](#_Toc179300748)

[10. Business Reports 16](#_Toc179300749)

[10.1 report\_sales\_by\_category 16](#_Toc179300750)

[10.2 report\_top\_customers 18](#_Toc179300751)

[10.3 report\_low\_inventory 21](#_Toc179300752)

[10.4 report\_monthly\_sales 23](#_Toc179300753)

[10.5 report\_product\_sales\_trend 25](#_Toc179300754)

## 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

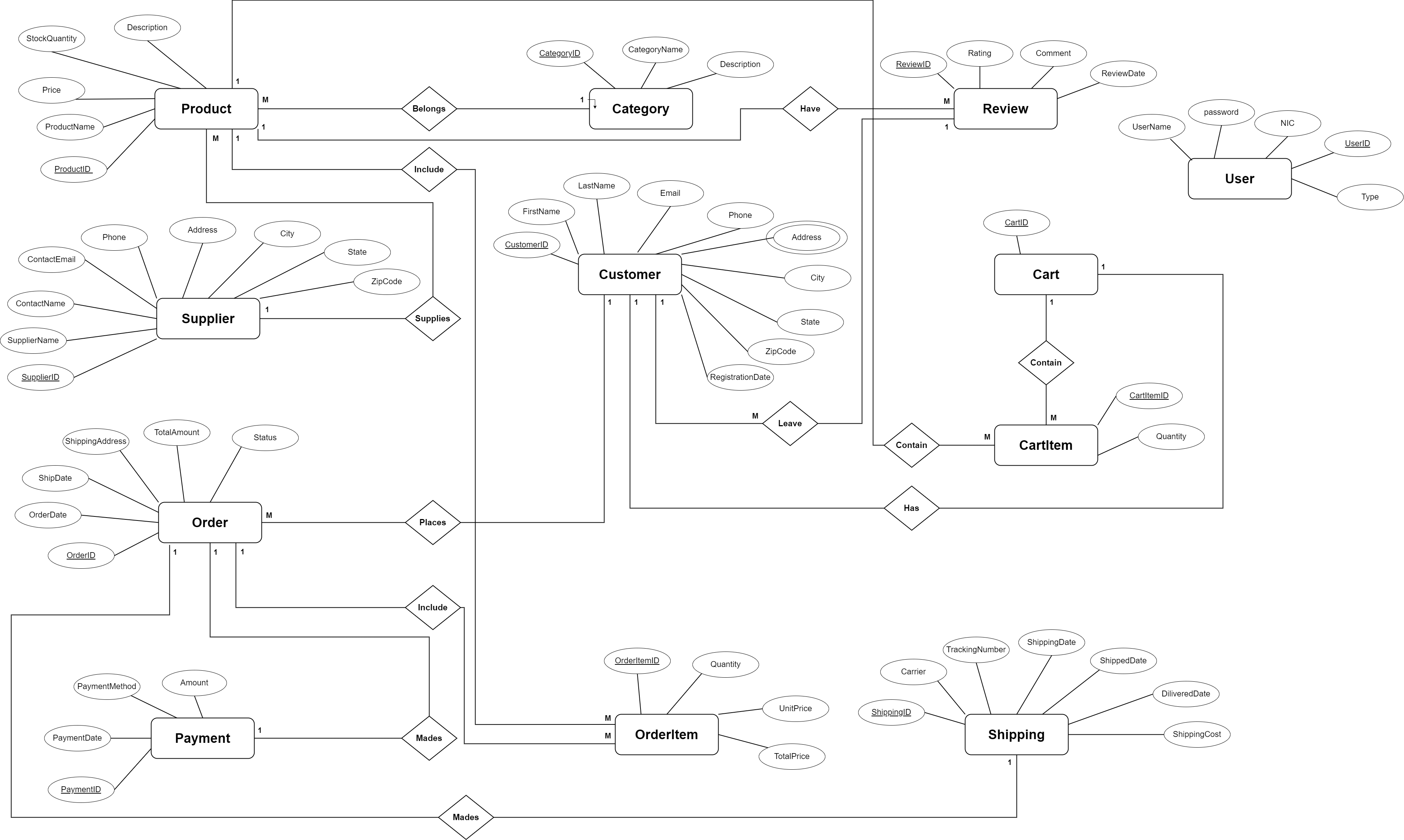


Figure 1

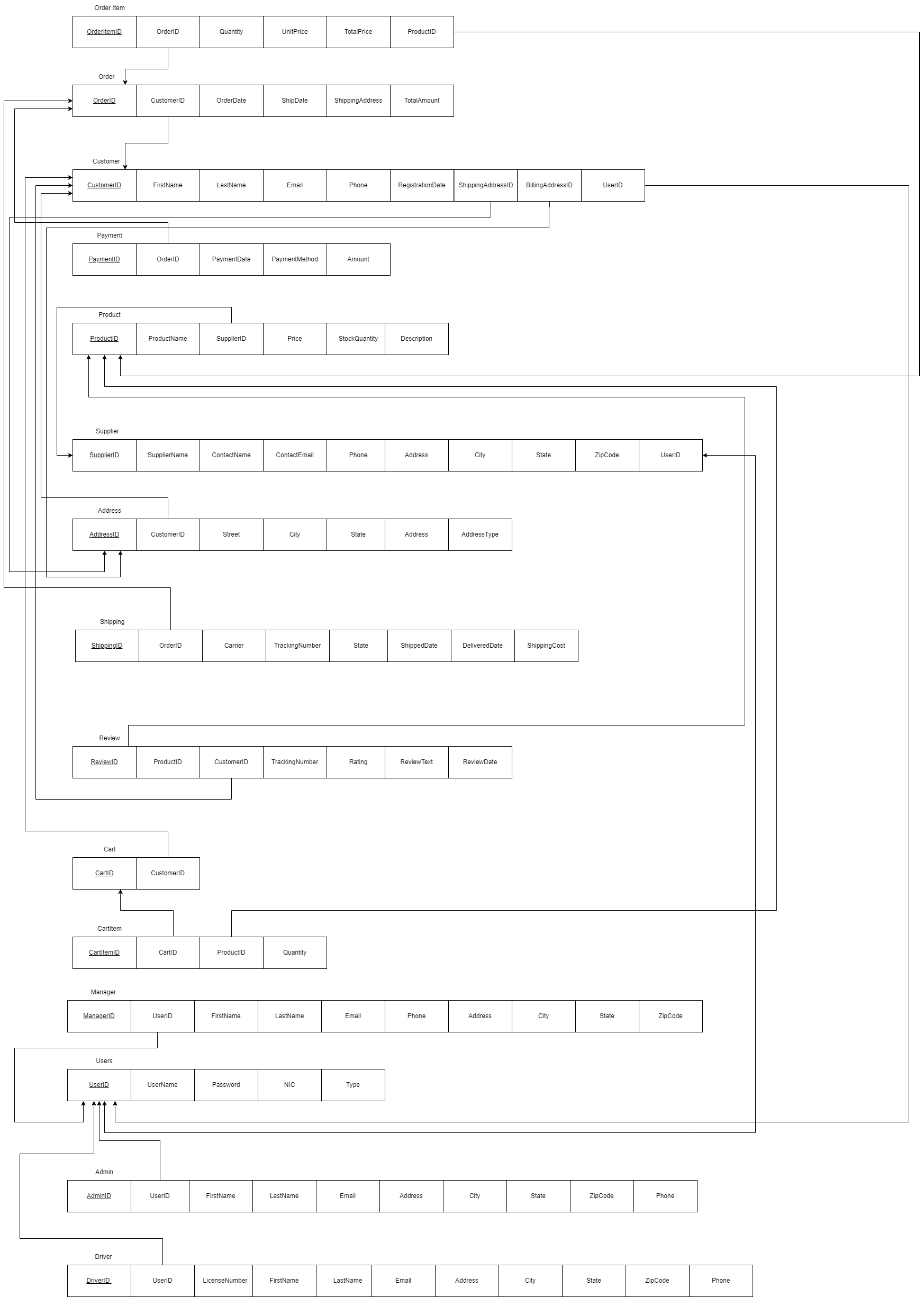


Figure 2

## 9. Oracle PL/SQL Programs for CRUD Operations

Visit Git-hub: [ManuraSanjula/ADBMS-NIBM (github.com)](https://github.com/ManuraSanjula/ADBMS-NIBM) 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;

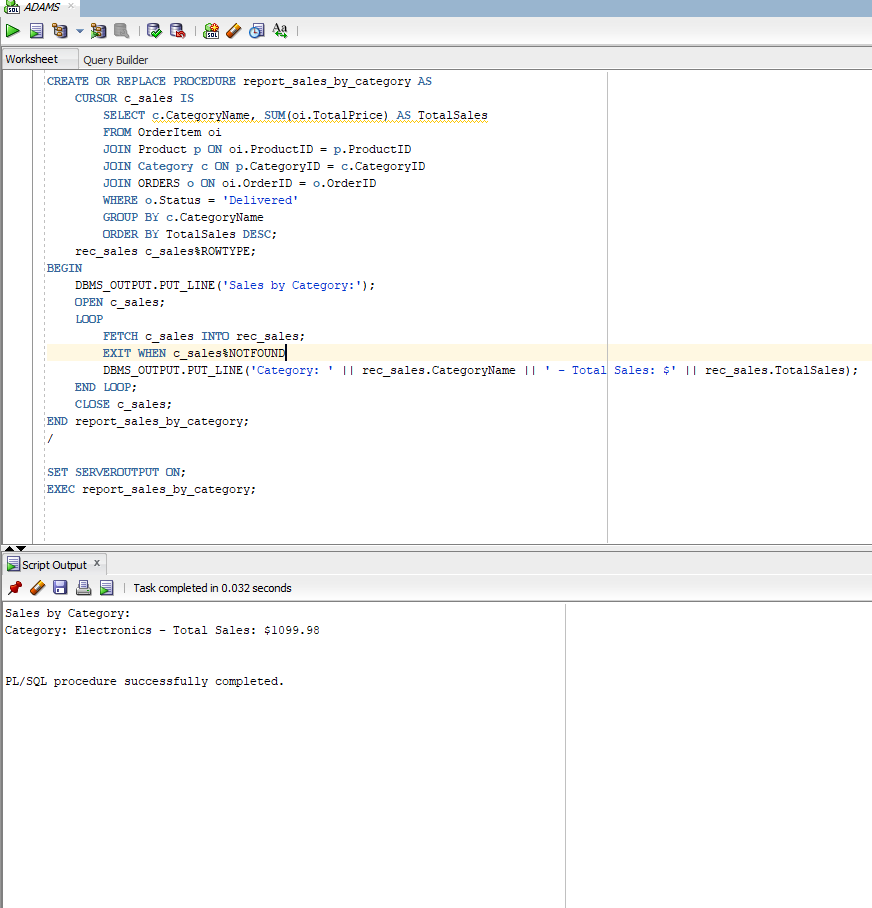
DBMS\_OUTPUT.PUT\_LINE('Category: ' || rec\_sales.CategoryName || ' - Total Sales: $' || rec\_sales.TotalSales);

END LOOP;

CLOSE c\_sales;

END report\_sales\_by\_category;

/



report\_sales\_by\_category 1



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

A screenshot of a computer

Description automatically generated

report\_top\_customers 1

A screenshot of a computer screen

Description automatically generated

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);

A screenshot of a computer

Description automatically generated

report\_low\_inventory 1

A screenshot of a computer

Description automatically generated

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);

A screenshot of a computer

Description automatically generated

report\_monthly\_sales 1

**A screenshot of a computer

Description automatically generated**

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);

A screenshot of a computer

Description automatically generated

report\_product\_sales\_trend 1

A screenshot of a computer

Description automatically generated

report\_product\_sales\_trend 2