Creating Tables

Customers

- **customer_id**: Unique identifier for each customer.
- **name**: Name of the customer.
- **phone**: Contact number of the customer.
- **street_no**: Street number of the customer's address.
- **house_no**: House number of the customer's address.
- **city**: City of the customer's address.

```
CREATE TABLE Customers (
  customer_id NUMBER PRIMARY KEY,
  name VARCHAR2(100),
  phone VARCHAR2(15) UNIQUE,
  street_no VARCHAR2(50),
  house_no VARCHAR2(50),
  city VARCHAR2(100),
);
```

Recipients

- **recipient_id**: Unique identifier for each recipient.
- **name**: Name of the recipient.
- **phone**: Contact number of the recipient.
- **street_no**: Street number of the recipient's address.
- **house_no**: House number of the recipient's address.
- **city**: City of the recipient's address.
- **customer_id**: Foreign key referencing the customer who placed the order.

```
CREATE TABLE Recipients (
  recipient_id NUMBER PRIMARY KEY,
  name VARCHAR2(100),
  phone VARCHAR2(15),
  street_no VARCHAR2(50),
  house_no VARCHAR2(50),
  city VARCHAR2(100),
  customer_id NUMBER,
  FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);
```

Riders

- **rider_id**: Unique identifier for each rider.
- **name**: Name of the rider.
- **phone**: Contact number of the rider.
- **current_location**: Current location of the rider.
- **status**: Status of the rider (available, delivering, unavailable).
- **rating**: Rating of the rider (1 to 5).
- **number_of_orders**: Number of orders assigned to the rider.

```
CREATE TABLE Riders (
    rider_id NUMBER PRIMARY KEY,
    name VARCHAR2(100),
    phone VARCHAR2(15),
    current_location VARCHAR2(255),
    status VARCHAR2(20) CHECK (status IN ('available', 'delivering',
'unavailable')),
    rating NUMBER(2,1) CHECK (rating BETWEEN 1 AND 5),
    nummber_of_orders NUMBER DEFAULT 0,
);
```

Managers

- **manager_id**: Unique identifier for each manager.
- **name**: Name of the manager.
- **phone**: Contact number of the manager.
- **store_id**: Foreign key referencing the dark store managed by the manager.

```
CREATE TABLE Managers (
    manager_id NUMBER PRIMARY KEY,
    name VARCHAR2(100),
    phone VARCHAR2(15) UNIQUE,
);
```

Dark Stores

```
CREATE TABLE DarkStores (
    store_id NUMBER PRIMARY KEY,
    city VARCHAR2(100),
    manager_id NUMBER,
    FOREIGN KEY (manager_id) REFERENCES Managers(manager_id),
);

-- Drop columns
ALTER TABLE DarkStores
DROP COLUMN IF EXISTS house_no, street_no, location;
```

Products

```
CREATE TABLE Products (
  product_id NUMBER PRIMARY KEY,
  name VARCHAR2(100),
  price NUMBER(10,2),
  category VARCHAR2(50) CHECK (category IN ('grocery', 'electronics',
'clothing', 'furniture')),
);
```

Inventory

```
CREATE TABLE Inventory (
  store_id NUMBER,
  product_id NUMBER,
  stock NUMBER,
  PRIMARY KEY (store_id, product_id),
  FOREIGN KEY (store_id) REFERENCES DarkStores(store_id),
  FOREIGN KEY (product_id) REFERENCES Products(product_id)
);
```

Orders

```
CREATE TABLE Orders (
  order_id NUMBER PRIMARY KEY,
  customer_id NUMBER,
  recipient_id NUMBER,
  rider_id NUMBER,
  store_id NUMBER,
  order_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  status VARCHAR2(30) CHECK (status IN ('pending', 'packed', 'dispatched',
'delivered', 'cancelled')),
  FOREIGN KEY (customer_id) REFERENCES Customers(customer_id),
  FOREIGN KEY (recipient_id) REFERENCES Recipients(recipient_id),
  FOREIGN KEY (rider_id) REFERENCES Riders(rider_id),
  FOREIGN KEY (store_id) REFERENCES DarkStores(store_id)
);
```

Contains

```
CREATE TABLE Contains (
  order_id NUMBER,
  product_id NUMBER,
  quantity NUMBER,
  PRIMARY KEY (order_id, product_id),
  FOREIGN KEY (order_id) REFERENCES Orders(order_id),
  FOREIGN KEY (product_id) REFERENCES Products(product_id)
);
```

Payments

```
CREATE TABLE Payments (
  payment_id NUMBER PRIMARY KEY,
  order_id NUMBER UNIQUE,
  customer_id NUMBER,
  amount NUMBER(10,2),
  payment_method VARCHAR2(20) CHECK (payment_method IN ('UPI', 'Card',
'Cash', 'NetBanking')),
  status VARCHAR2(20) CHECK (status IN ('pending', 'success', 'failed')),
  paid_at TIMESTAMP,
  FOREIGN KEY (order_id) REFERENCES Orders(order_id),
  FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);
```

Procedures

insert_customer

```
CREATE OR REPLACE PROCEDURE insert_customer (
  p_name       IN VARCHAR2,
  p_phone      IN VARCHAR2,
  p_email      IN VARCHAR2,
  p_street_no  IN VARCHAR2,
  p_house_no   IN VARCHAR2,
  p_city       IN VARCHAR2
)
AS
  v_id NUMBER;
BEGIN
  SELECT NVL(MAX(customer_id), 0) + 1 INTO v_id FROM Customers;

  INSERT INTO Customers (customer_id, name, phone, email, street_no,
house_no, city)
  VALUES (v_id, p_name, p_phone, p_email, p_street_no, p_house_no, p_city);

    DBMS_OUTPUT.PUT_LINE('Customer inserted with ID: ' || v_id);
END;
/
```

insert_recipient

```
CREATE OR REPLACE PROCEDURE insert_recipient (
  p_name        IN VARCHAR2,
  p_phone       IN VARCHAR2,
  p_street_no   IN VARCHAR2,
  p_house_no    IN VARCHAR2,
  p_city        IN VARCHAR2,
  p_customer_id IN NUMBER
)
AS
  v_id NUMBER;
BEGIN
  SELECT NVL(MAX(recipient_id), 0) + 1 INTO v_id FROM Recipients;

  INSERT INTO Recipients (recipient_id, name, phone, street_no, house_no,
city, customer_id)
  VALUES (v_id, p_name, p_phone, p_street_no, p_house_no, p_city,
p_customer_id);

  DBMS_OUTPUT.PUT_LINE('Recipient inserted successfully with ID: ' || v_id);
END;
```

insert_rider

```
CREATE OR REPLACE PROCEDURE insert_rider (
  p_name           IN VARCHAR2,
  p_contact_number IN VARCHAR2,
  p_status         IN VARCHAR2,
  p_location       IN VARCHAR2
)
AS
  v_id NUMBER;
BEGIN
  -- Generate next rider_id
  SELECT NVL(MAX(rider_id), 0) + 1 INTO v_id FROM Riders;

  -- Insert new rider with default rating and reviews
  INSERT INTO Riders (
    rider_id, name, contact_number, status, current_location, rating,
number_of_reviews
  ) VALUES (
    v_id, p_name, p_contact_number, p_status, p_location, 5, 0
  );

  -- Output success message
  DBMS_OUTPUT.PUT_LINE('Rider inserted successfully with ID: ' || v_id);
END;
/
```

insert_manager

```
CREATE OR REPLACE PROCEDURE insert_manager (
  p_name      IN VARCHAR2,
  p_phone     IN VARCHAR2,
)
AS
  v_id NUMBER;
BEGIN
  SELECT NVL(MAX(manager_id), 0) + 1 INTO v_id FROM Managers;

  INSERT INTO Managers (manager_id, name, phone)
  VALUES (v_id, p_name, p_phone);

  DBMS_OUTPUT.PUT_LINE('Manager inserted successfully with ID: ' || v_id);
END;
/
```

insert_dark_store

```
CREATE OR REPLACE PROCEDURE insert_dark_store (
  p_location IN VARCHAR2,
    p_manager_id IN NUMBER
)
AS
  v_id NUMBER;
BEGIN
  SELECT NVL(MAX(store_id), 0) + 1 INTO v_id FROM DarkStores;

  INSERT INTO DarkStores (store_id, city, manager_id)
  VALUES (v_id, p_location, p_manager_id);

  DBMS_OUTPUT.PUT_LINE('Dark Store inserted successfully with ID: ' ||
v_id);
END;
/
```

insert_product

```
CREATE OR REPLACE PROCEDURE insert_product (
  p_name  IN VARCHAR2,
  p_price IN NUMBER,
    p_category IN VARCHAR2
)
AS
  v_id NUMBER;
BEGIN
  SELECT NVL(MAX(product_id), 0) + 1 INTO v_id FROM Products;

  INSERT INTO Products (product_id, name, price, category)
  VALUES (v_id, p_name, p_price, p_category);

  DBMS_OUTPUT.PUT_LINE('Product inserted successfully with ID: ' || v_id);
END;
/
```

insert_inventory

```
CREATE OR REPLACE PROCEDURE insert_inventory (
  p_store_id   IN NUMBER,
  p_product_id IN NUMBER,
  p_stock      IN NUMBER
)
AS
BEGIN
  INSERT INTO Inventory (store_id, product_id, stock)
  VALUES (p_store_id, p_product_id, p_stock);

  DBMS_OUTPUT.PUT_LINE('Inventory record inserted for Store ID ' ||
p_store_id || ' and Product ID ' || p_product_id);
END;
/
```

insert_order

```
CREATE OR REPLACE PROCEDURE insert_order (
  p_customer_id  IN NUMBER,
  p_recipient_id IN NUMBER,
  p_store_id     IN NUMBER,
  p_status       IN VARCHAR2
)
AS
  v_id NUMBER;
BEGIN
  SELECT NVL(MAX(order_id), 0) + 1 INTO v_id FROM Orders;

  INSERT INTO Orders (order_id, customer_id, recipient_id, store_id,
order_time, status)
  VALUES (v_id, p_customer_id, p_recipient_id, p_store_id,
CURRENT_TIMESTAMP, p_status);

  DBMS_OUTPUT.PUT_LINE('Order inserted successfully with ID: ' || v_id);
END;
/
```

insert_contains

```
CREATE OR REPLACE PROCEDURE insert_contains (
  p_order_id   IN NUMBER,
  p_product_id IN NUMBER,
  p_quantity   IN NUMBER
)
AS
BEGIN
  INSERT INTO Contains (order_id, product_id, quantity)
  VALUES (p_order_id, p_product_id, p_quantity);

  DBMS_OUTPUT.PUT_LINE('Product ID ' || p_product_id || ' added to Order ID
' || p_order_id);
END;
/
```

insert_payment

```
CREATE OR REPLACE PROCEDURE insert_payment (
  p_order_id      IN NUMBER,
  p_method        IN VARCHAR2,
  p_status        IN VARCHAR2
)
AS
  v_id NUMBER;
BEGIN
  -- Get the next available payment_id
  SELECT NVL(MAX(payment_id), 0) + 1 INTO v_id FROM Payments;

  -- Insert into Payments table, setting paid_at to the current timestamp
  INSERT INTO Payments (payment_id, order_id, paid_at, payment_method,
status)
  VALUES (v_id, p_order_id, SYSTIMESTAMP, p_method, p_status);

  -- Output the inserted payment ID
  DBMS_OUTPUT.PUT_LINE('Payment inserted successfully with ID: ' || v_id);
END;
/
```

UpdateStock

```
CREATE OR REPLACE PROCEDURE update_stock (
  p_store_id   IN NUMBER,
  p_product_id IN NUMBER,
  p_quantity   IN NUMBER
)
```

```
AS
BEGIN
  -- Update the stock in the Inventory table
  UPDATE Inventory
  SET stock = p_quantity
  WHERE store_id = p_store_id AND product_id = p_product_id;

  -- Output success message
  DBMS_OUTPUT.PUT_LINE('Stock updated successfully for Store ID ' ||
p_store_id || ' and Product ID ' || p_product_id);
END;
/
```

Place Order for Self

- takes in customer_id, recipient_id, item_id, quantity
- finds nearest dark store
- checks if item is available in the dark store
- if available, creates an order
- updates inventory
- updates rider status
- updates order status
- updates payment status
- updates order status to 'packed'

```
CREATE OR REPLACE PROCEDURE place_order_for_self (
  p_customer_id  IN NUMBER,
  p_item_id      IN NUMBER,
  p_quantity     IN NUMBER
)
AS
  v_order_id     NUMBER;
  v_store_id     NUMBER;
  v_stock        NUMBER;
  v_rider_id     NUMBER;
  v_payment_id   NUMBER;
  v_amount       NUMBER;
  v_city         VARCHAR2(50);
  v_price        NUMBER;
BEGIN
  -- Get customer's city
  SELECT city INTO v_city FROM Customers WHERE customer_id = p_customer_id;

  -- Find the nearest dark store in the same city
  SELECT store_id INTO v_store_id
  FROM DarkStores
  WHERE city = v_city
  AND ROWNUM = 1;
```

```
  -- Assign a rider from the same city
  SELECT rider_id INTO v_rider_id
  FROM Riders
  WHERE status = 'available' AND CURRENT_LOCATION = v_city
  AND ROWNUM = 1;

  -- Check stock in inventory
  SELECT stock INTO v_stock
  FROM Inventory
  WHERE store_id = v_store_id AND product_id = p_item_id;

  IF v_stock < p_quantity THEN
    DBMS_OUTPUT.PUT_LINE('✖ Insufficient stock for item ID: ' || p_item_id);
    RETURN;
  END IF;

  -- Generate new order ID
  SELECT NVL(MAX(order_id), 0) + 1 INTO v_order_id FROM Orders;

  -- Insert into Orders
  INSERT INTO Orders (order_id, customer_id, recipient_id, rider_id,
store_id, order_time, status)
  VALUES (v_order_id, p_customer_id, p_customer_id, v_rider_id, v_store_id,
CURRENT_TIMESTAMP, 'pending');

  -- Insert order details into Contains table
  INSERT INTO Contains (order_id, product_id, quantity)
  VALUES (v_order_id, p_item_id, p_quantity);

  -- Update inventory
  UPDATE Inventory
  SET stock = stock - p_quantity
  WHERE store_id = v_store_id AND product_id = p_item_id;

  -- Update rider status to 'delivering'
  UPDATE Riders
  SET status = 'delivering'
  WHERE rider_id = v_rider_id;

  -- Get product price
  SELECT price INTO v_price FROM Products WHERE product_id = p_item_id;
  v_amount := v_price * p_quantity;

  -- Create payment record
  SELECT NVL(MAX(payment_id), 0) + 1 INTO v_payment_id FROM Payments;

  INSERT INTO Payments (payment_id, order_id, customer_id, amount,
payment_method, status, paid_at)
```

```
  VALUES (v_payment_id, v_order_id, p_customer_id, v_amount, 'UPI',
'pending', CURRENT_TIMESTAMP);

  -- Update order status
  UPDATE Orders SET status = 'packed' WHERE order_id = v_order_id;

  DBMS_OUTPUT.PUT_LINE('✅ Order placed successfully!');
  DBMS_OUTPUT.PUT_LINE('📦 Order ID: ' || v_order_id);
  DBMS_OUTPUT.PUT_LINE('💳 Payment ID: ' || v_payment_id);
  DBMS_OUTPUT.PUT_LINE('🚴 Rider ID: ' || v_rider_id);
END;
```

Place Order for Other

- takes in customer_id, recipient_id, item_id, quantity
- finds nearest dark store
- checks if item is available in the dark store
- if available, creates an order
- updates inventory
- updates rider status
- updates order status
- updates payment status

```
CREATE OR REPLACE PROCEDURE place_order_for_other (
  p_customer_id  IN NUMBER,
  p_recipient_id IN NUMBER,
  p_item_id      IN NUMBER,
  p_quantity     IN NUMBER
)
AS
  v_order_id     NUMBER;
  v_store_id     NUMBER;
  v_stock        NUMBER;
  v_rider_id     NUMBER;
  v_payment_id   NUMBER;
  v_amount       NUMBER;
  v_city         VARCHAR2(50);
  v_price        NUMBER;
BEGIN
    -- Get recipient's city
    SELECT city INTO v_city FROM Recipients WHERE recipient_id =
p_recipient_id;

    -- Find the nearest dark store in the same city
    SELECT store_id INTO v_store_id
    FROM DarkStores
    WHERE city = v_city
    AND ROWNUM = 1;
```

```
    -- Assign a rider from the same city
    SELECT rider_id INTO v_rider_id
    FROM Riders
    WHERE status = 'available' AND CURRENT_LOCATION = v_city
    AND ROWNUM = 1;

    -- Check stock in inventory
    SELECT stock INTO v_stock
    FROM Inventory
    WHERE store_id = v_store_id AND product_id = p_item_id;

    IF v_stock < p_quantity THEN
        DBMS_OUTPUT.PUT_LINE('✖ Insufficient stock for item ID: ' ||
p_item_id);
        RETURN;
    END IF;

    -- Generate new order ID
    SELECT NVL(MAX(order_id), 0) + 1 INTO v_order_id FROM Orders;

    -- Insert into Orders
    INSERT INTO Orders (order_id, customer_id, recipient_id, rider_id,
store_id, order_time, status)
    VALUES (v_order_id, p_customer_id, p_recipient_id, v_rider_id,
v_store_id, CURRENT_TIMESTAMP, 'pending');

    -- Insert order details into Contains table
    INSERT INTO Contains (order_id, product_id, quantity)
    VALUES (v_order_id, p_item_id, p_quantity);

    -- Update inventory
    UPDATE Inventory
    SET stock = stock - p_quantity
    WHERE store_id = v_store_id AND product_id = p_item_id;

    -- Update rider status to 'delivering'
    UPDATE Riders
    SET status = 'delivering'
    WHERE rider_id = v_rider_id;

    -- Get product price
    SELECT price INTO v_price FROM Products WHERE product_id = p_item_id;
    v_amount := v_price * p_quantity;
    -- Create payment record
    SELECT NVL(MAX(payment_id), 0) + 1 INTO v_payment_id FROM Payments;
    INSERT INTO Payments (payment_id, order_id, customer_id, amount,
payment_method, status, paid_at)
```

```
    VALUES (v_payment_id, v_order_id, p_customer_id, v_amount, 'UPI',
'pending', CURRENT_TIMESTAMP);
    -- Update order status
    UPDATE Orders SET status = 'packed' WHERE order_id = v_order_id;
    DBMS_OUTPUT.PUT_LINE('✅ Order placed successfully!');
    DBMS_OUTPUT.PUT_LINE('🆔 Order ID: ' || v_order_id);
    DBMS_OUTPUT.PUT_LINE('💳 Payment ID: ' || v_payment_id);
    DBMS_OUTPUT.PUT_LINE('🚴 Rider ID: ' || v_rider_id);
END;
/
```

UpdateInventory

```
CREATE OR REPLACE PROCEDURE update_inventory (
  p_store_id   IN NUMBER,
  p_product_id IN NUMBER,
  p_quantity   IN NUMBER
)
AS
BEGIN
  -- Update the stock in the Inventory table
  UPDATE Inventory
  SET stock = p_quantity
  WHERE store_id = p_store_id AND product_id = p_product_id;

  -- Output success message
  DBMS_OUTPUT.PUT_LINE('Stock updated successfully for Store ID ' ||
p_store_id || ' and Product ID ' || p_product_id);
END;
/
```

SubmitReview

```
CREATE OR REPLACE PROCEDURE submit_rider_review (
  p_rider_id    IN NUMBER,
  p_new_rating  IN FLOAT
)
AS
  v_old_rating        FLOAT;
  v_number_of_reviews NUMBER;
  v_new_average       FLOAT;
BEGIN
  -- Step 1: Validate rating input
  IF p_new_rating < 1 OR p_new_rating > 5 THEN
    DBMS_OUTPUT.PUT_LINE('❌ Rating must be between 1 and 5.');
    RETURN;
  END IF;
```

```
  -- Step 2: Fetch current rating and number of reviews
  BEGIN
    SELECT rating, number_of_reviews
    INTO v_old_rating, v_number_of_reviews
    FROM Riders
    WHERE rider_id = p_rider_id;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('❌ Rider not found for rider_id: ' ||
p_rider_id);
      RETURN;
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('❌ Error fetching rider data: ' || SQLERRM);
      RETURN;
  END;

  -- Step 3: Compute new average rating
  v_new_average := ((v_old_rating * v_number_of_reviews) + p_new_rating) /
(v_number_of_reviews + 1);

  -- Step 4: Update rider's rating and number of reviews
  BEGIN
    UPDATE Riders
    SET rating = v_new_average,
        number_of_reviews = number_of_reviews + 1
    WHERE rider_id = p_rider_id;
  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('❌ Error updating rider rating: ' || SQLERRM);
      RETURN;
  END;

  DBMS_OUTPUT.PUT_LINE('✅ Rider review submitted successfully!');
  DBMS_OUTPUT.PUT_LINE('⭐ New average rating: ' || ROUND(v_new_average, 2));

EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('❌ Unexpected error: ' || SQLERRM);
    ROLLBACK;
END;
/
```

UpdateOrderStatus

```
CREATE OR REPLACE PROCEDURE update_order_status (
  p_order_id IN NUMBER,
  p_new_status IN VARCHAR2
)
AS
```

```
  v_current_status VARCHAR2(30);
BEGIN
    -- Step 1: Fetch current status
    SELECT status INTO v_current_status FROM Orders WHERE order_id =
p_order_id;

    -- Step 2: Validate new status
    IF p_new_status NOT IN ('pending', 'packed', 'dispatched', 'delivered',
'cancelled') THEN
        DBMS_OUTPUT.PUT_LINE('❌ Invalid status: ' || p_new_status);
        RETURN;
    END IF;

    -- Step 3: Update order status
    UPDATE Orders SET status = p_new_status WHERE order_id = p_order_id;

    DBMS_OUTPUT.PUT_LINE('✅ Order status updated from ' || v_current_status
|| ' to ' || p_new_status);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('❌ Order not found for order_id: ' ||
p_order_id);
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('❌ Error updating order status: ' || SQLERRM);
END;
/
```

MarkOrderAsDelivered

```
CREATE OR REPLACE PROCEDURE mark_order_delivered (
  p_order_id IN NUMBER
)
AS
  v_payment_id NUMBER;
BEGIN
  -- Update the order status to 'delivered'
  UPDATE Orders
  SET status = 'delivered'
  WHERE order_id = p_order_id;

  -- Fetch the payment_id for the given order
  SELECT payment_id INTO v_payment_id
  FROM Payments
  WHERE order_id = p_order_id;

  -- Set payment status to 'success'
  UPDATE Payments
  SET status = 'success', paid_at = CURRENT_TIMESTAMP
  WHERE payment_id = v_payment_id;
```

```
  DBMS_OUTPUT.PUT_LINE('✅ Order marked as delivered');
  DBMS_OUTPUT.PUT_LINE('💳 Payment status updated to success');
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('❌ No payment found for this order.');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('❌ Error: ' || SQLERRM);
END;
/
```

GenerateReport

```
CREATE OR REPLACE PROCEDURE generate_report AS
  v_total_orders     NUMBER;
  v_total_revenue    NUMBER;
  v_avg_order_value  NUMBER;
BEGIN
  DBMS_OUTPUT.PUT_LINE('📊 Order Report (Last 30 Days)');
  DBMS_OUTPUT.PUT_LINE('---------------------------------');

  -- Total orders in last 30 days
  SELECT COUNT(*) INTO v_total_orders
  FROM Orders
  WHERE order_time ≥ SYSDATE - 30;

  -- Total revenue in last 30 days
  SELECT NVL(SUM(amount), 0) INTO v_total_revenue
  FROM Payments
  WHERE paid_at ≥ SYSDATE - 30 AND status = 'success';

  -- Average order value
  IF v_total_orders > 0 THEN
    v_avg_order_value := v_total_revenue / v_total_orders;
  ELSE
    v_avg_order_value := 0;
  END IF;

  DBMS_OUTPUT.PUT_LINE('📦 Total Orders      : ' || v_total_orders);
  DBMS_OUTPUT.PUT_LINE('💰 Total Revenue     : ₹' || v_total_revenue);
  DBMS_OUTPUT.PUT_LINE('📈 Avg Order Value   : ₹' || ROUND(v_avg_order_value,
2));
  DBMS_OUTPUT.PUT_LINE('---------------------------------');
  DBMS_OUTPUT.PUT_LINE('📋 Orders by Status:');

  -- Orders by status
  FOR r IN (
    SELECT status, COUNT(*) AS count
    FROM Orders
```

```
    WHERE order_time ≥ SYSDATE - 30
    GROUP BY status
  ) LOOP
    DBMS_OUTPUT.PUT_LINE('   - ' || r.status || ': ' || r.count);
  END LOOP;

END;
/
```

GetMostSoldProducts

```
CREATE OR REPLACE PROCEDURE get_most_sold_products_last_30_days
AS
  CURSOR most_sold_cursor IS
    SELECT c.product_id,
           p.product_name,
           SUM(c.quantity) AS total_quantity_sold
    FROM Contains c
    JOIN Orders o ON c.order_id = o.order_id
    JOIN Products p ON c.product_id = p.product_id
    WHERE o.order_time ≥ SYSDATE - 30 -- Last 30 days
    AND o.status = 'delivered'         -- Only completed (delivered) orders
    GROUP BY c.product_id, p.product_name
    ORDER BY total_quantity_sold DESC;

  v_product_id       NUMBER;
  v_product_name     VARCHAR2(100);
  v_total_quantity   NUMBER;
BEGIN
  -- Open cursor and fetch results
  OPEN most_sold_cursor;

  LOOP
    FETCH most_sold_cursor INTO v_product_id, v_product_name,
v_total_quantity;
    EXIT WHEN most_sold_cursor%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE('Product ID: ' || v_product_id ||
                         ', Product Name: ' || v_product_name ||
                         ', Total Quantity Sold: ' || v_total_quantity);
  END LOOP;

  -- Close the cursor
  CLOSE most_sold_cursor;

END get_most_sold_products_last_30_days;
```

GetMostPopularProductByCategory

```
CREATE OR REPLACE PROCEDURE get_most_popular_product_by_category (
  p_category IN VARCHAR2
)
AS
  CURSOR most_popular_cursor IS
    SELECT
      c.product_id,
      p.name AS product_name,
      SUM(c.quantity) AS total_quantity_sold
    FROM Contains c
    JOIN Orders o ON c.order_id = o.order_id
    JOIN Products p ON c.product_id = p.product_id
    WHERE p.category = p_category
      AND o.status = 'delivered'
    GROUP BY c.product_id, p.name
    ORDER BY total_quantity_sold DESC;

  v_product_id       NUMBER;
  v_product_name     VARCHAR2(100);
  v_total_quantity   NUMBER;
BEGIN
  -- Open cursor and fetch results
  OPEN most_popular_cursor;

  LOOP
    FETCH most_popular_cursor INTO v_product_id, v_product_name,
v_total_quantity;
    EXIT WHEN most_popular_cursor%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE('Product ID: ' || v_product_id ||
                         ', Product Name: ' || v_product_name ||
                         ', Total Quantity Sold: ' || v_total_quantity);
  END LOOP;

  -- Close the cursor
  CLOSE most_popular_cursor;

END get_most_popular_product_by_category;
```

Functions

CalculateOrderTotal

```
CREATE OR REPLACE FUNCTION calculate_order_total(p_order_id IN NUMBER)
RETURN NUMBER
AS
  v_total_amount NUMBER := 0;
BEGIN
  SELECT SUM(c.quantity * p.price)
  INTO v_total_amount
  FROM Contains c
  JOIN Products p ON c.product_id = p.product_id
  WHERE c.order_id = p_order_id;

  RETURN v_total_amount;
END calculate_order_total;
```

GetCustomerOrderHistory

```
CREATE OR REPLACE FUNCTION get_customer_order_history(p_customer_id IN
NUMBER)
RETURN SYS_REFCURSOR
AS
  v_cursor SYS_REFCURSOR;
BEGIN
  OPEN v_cursor FOR
    SELECT o.order_id, o.status, o.order_time
    FROM Orders o
    WHERE o.customer_id = p_customer_id;

  RETURN v_cursor;
END get_customer_order_history;
```

CheckProductAvailability

```
CREATE OR REPLACE FUNCTION check_product_availability(p_product_id IN
NUMBER, p_store_id IN NUMBER)
RETURN BOOLEAN
AS
  v_stock NUMBER;
BEGIN
  SELECT stock INTO v_stock
  FROM Inventory
  WHERE product_id = p_product_id AND store_id = p_store_id;

  IF v_stock > 0 THEN
```

```
    RETURN TRUE;
  ELSE
    RETURN FALSE;
  END IF;
END check_product_availability;
```

GetRiderDeliveryHistory

```
CREATE OR REPLACE FUNCTION get_rider_delivery_history(p_rider_id IN NUMBER)
RETURN SYS_REFCURSOR
AS
  v_cursor SYS_REFCURSOR;
BEGIN
    OPEN v_cursor FOR
        SELECT o.order_id, o.status, o.order_time
        FROM Orders o
        WHERE o.rider_id = p_rider_id;

    RETURN v_cursor;
END get_rider_delivery_history;
```

GetTotalProductSales

```
CREATE OR REPLACE FUNCTION get_total_product_sales(p_product_id IN NUMBER)
RETURN NUMBER
AS
  v_total_sales NUMBER;
BEGIN
    SELECT SUM(c.quantity)
    INTO v_total_sales
    FROM Contains c
    JOIN Orders o ON c.order_id = o.order_id
    WHERE c.product_id = p_product_id
        AND o.status = 'delivered';

    RETURN NVL(v_total_sales, 0);
END get_total_product_sales;
```

Triggers

applyDiscount

```
CREATE OR REPLACE TRIGGER applyDiscount
BEFORE INSERT OR UPDATE ON Products
FOR EACH ROW
BEGIN
  IF :NEW.price < 0 THEN
```

```
    RAISE_APPLICATION_ERROR(-20001, 'Price cannot be negative');
  END IF;

  IF :NEW.price > 1000 THEN
    :NEW.price := :NEW.price * 0.9; -- Apply a 10% discount
  END IF;
END;
/
```