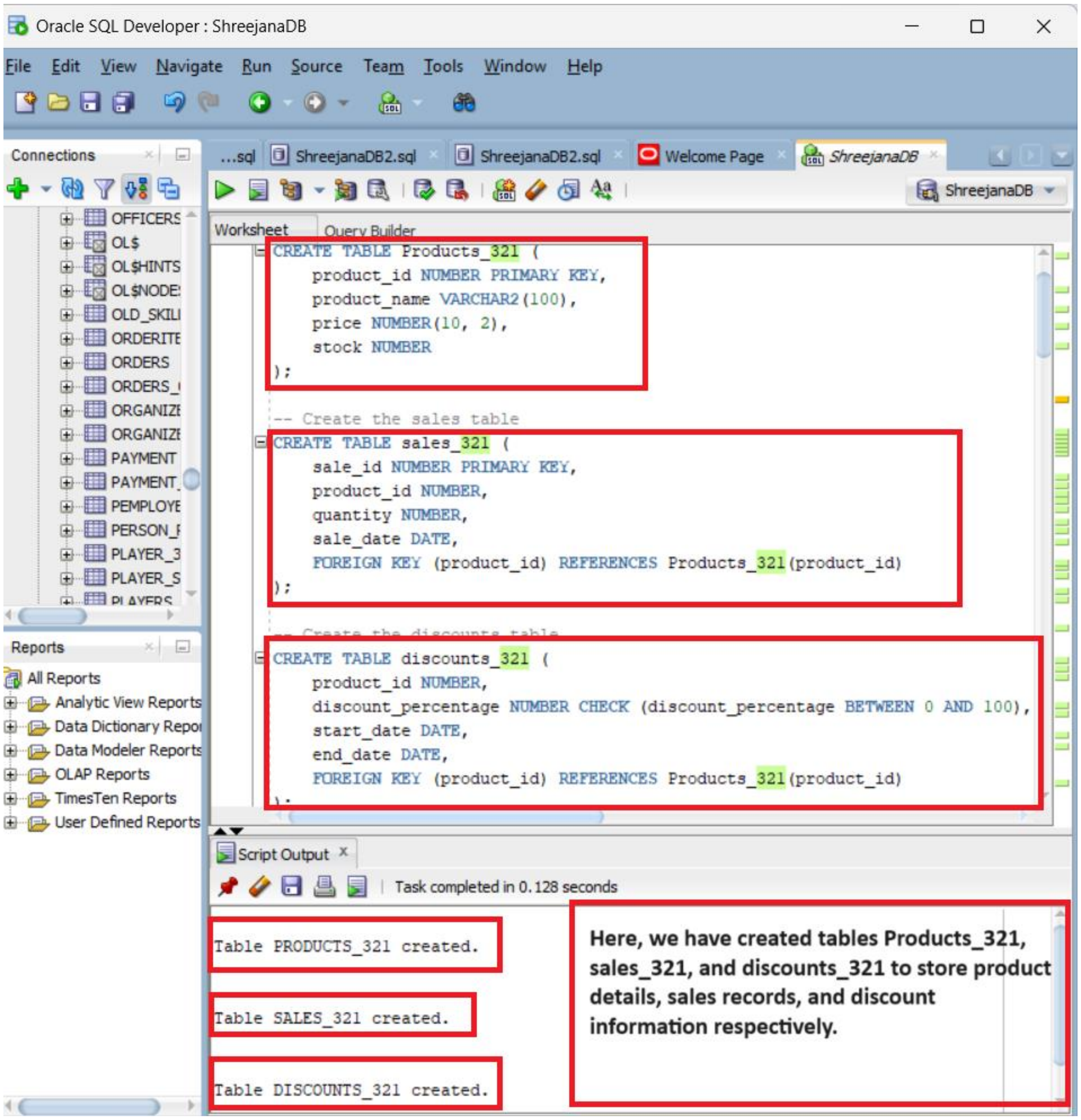
CSD 4203 – Database Programming

Student ID : C0930321

Student Name : Shreejana Shrestha

Assignment # 11

TASK 1: Tables creation



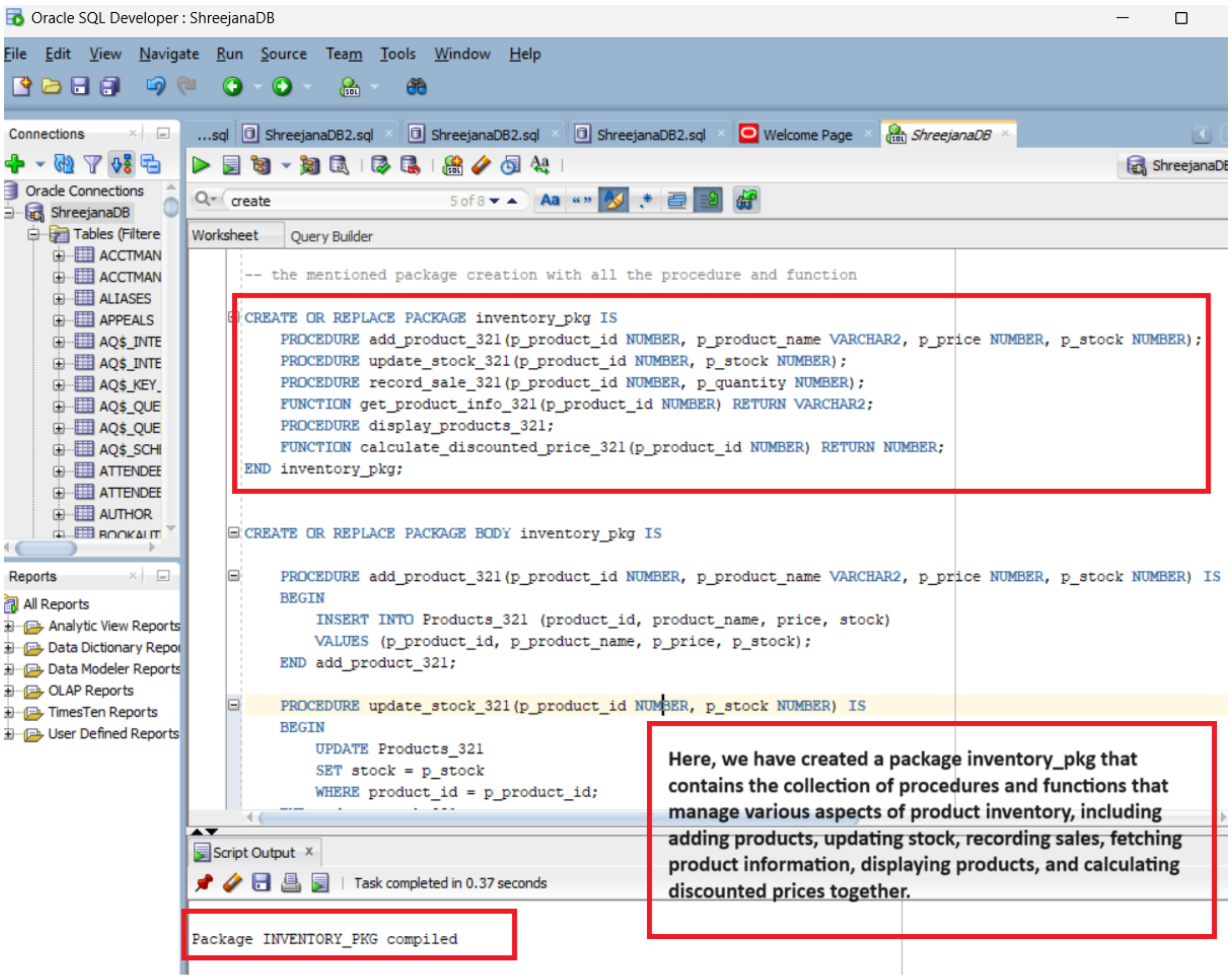
```
CREATE TABLE Products_321 (
    product_id NUMBER PRIMARY KEY,
    product_name VARCHAR2(100),
    price NUMBER(10, 2),
    stock NUMBER
);

-- Create the sales table
CREATE TABLE sales_321 (
    sale_id NUMBER PRIMARY KEY,
    product_id NUMBER,
    quantity NUMBER,
    sale_date DATE,
    FOREIGN KEY (product_id) REFERENCES Products_321(product_id)
);

-- Create the discounts table
CREATE TABLE discounts_321 (
    product_id NUMBER,
    discount_percentage NUMBER CHECK (discount_percentage BETWEEN 0 AND 100),
    start_date DATE,
    end_date DATE,
    FOREIGN KEY (product_id) REFERENCES Products_321(product_id)
);
```

Task completed in 0.128 seconds

```
Table PRODUCTS_321 created.

Table SALES_321 created.

Table DISCOUNTS_321 created.
```

Here, we have created tables Products_321, sales_321, and discounts_321 to store product details, sales records, and discount information respectively.

TASK 2: Creation of Package (inventory_pkg)

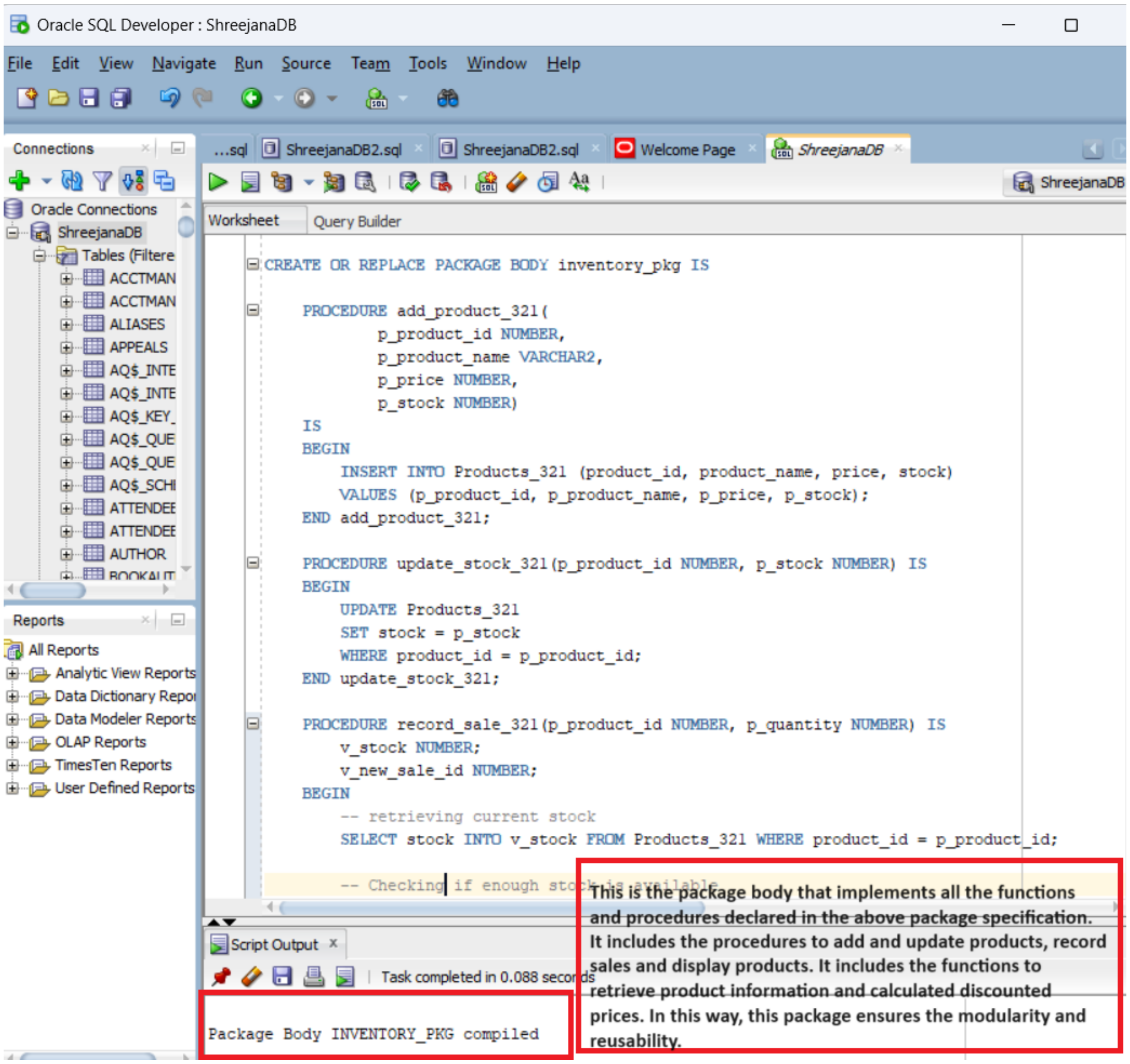


```
-- the mentioned package creation with all the procedure and function

CREATE OR REPLACE PACKAGE inventory_pkg IS
    PROCEDURE add_product_321(p_product_id NUMBER, p_product_name VARCHAR2, p_price NUMBER, p_stock NUMBER);
    PROCEDURE update_stock_321(p_product_id NUMBER, p_stock NUMBER);
    PROCEDURE record_sale_321(p_product_id NUMBER, p_quantity NUMBER);
    FUNCTION get_product_info_321(p_product_id NUMBER) RETURN VARCHAR2;
    PROCEDURE display_products_321;
    FUNCTION calculate_discounted_price_321(p_product_id NUMBER) RETURN NUMBER;
END inventory_pkg;

CREATE OR REPLACE PACKAGE BODY inventory_pkg IS

    PROCEDURE add_product_321(p_product_id NUMBER, p_product_name VARCHAR2, p_price NUMBER, p_stock NUMBER) IS
    BEGIN
        INSERT INTO Products_321 (product_id, product_name, price, stock)
        VALUES (p_product_id, p_product_name, p_price, p_stock);
    END add_product_321;

    PROCEDURE update_stock_321(p_product_id NUMBER, p_stock NUMBER) IS
    BEGIN
        UPDATE Products_321
        SET stock = p_stock
        WHERE product_id = p_product_id;
```

Here, we have created a package inventory_pkg that contains the collection of procedures and functions that manage various aspects of product inventory, including adding products, updating stock, recording sales, fetching product information, displaying products, and calculating discounted prices together.

```
Package INVENTORY_PKG compiled
```

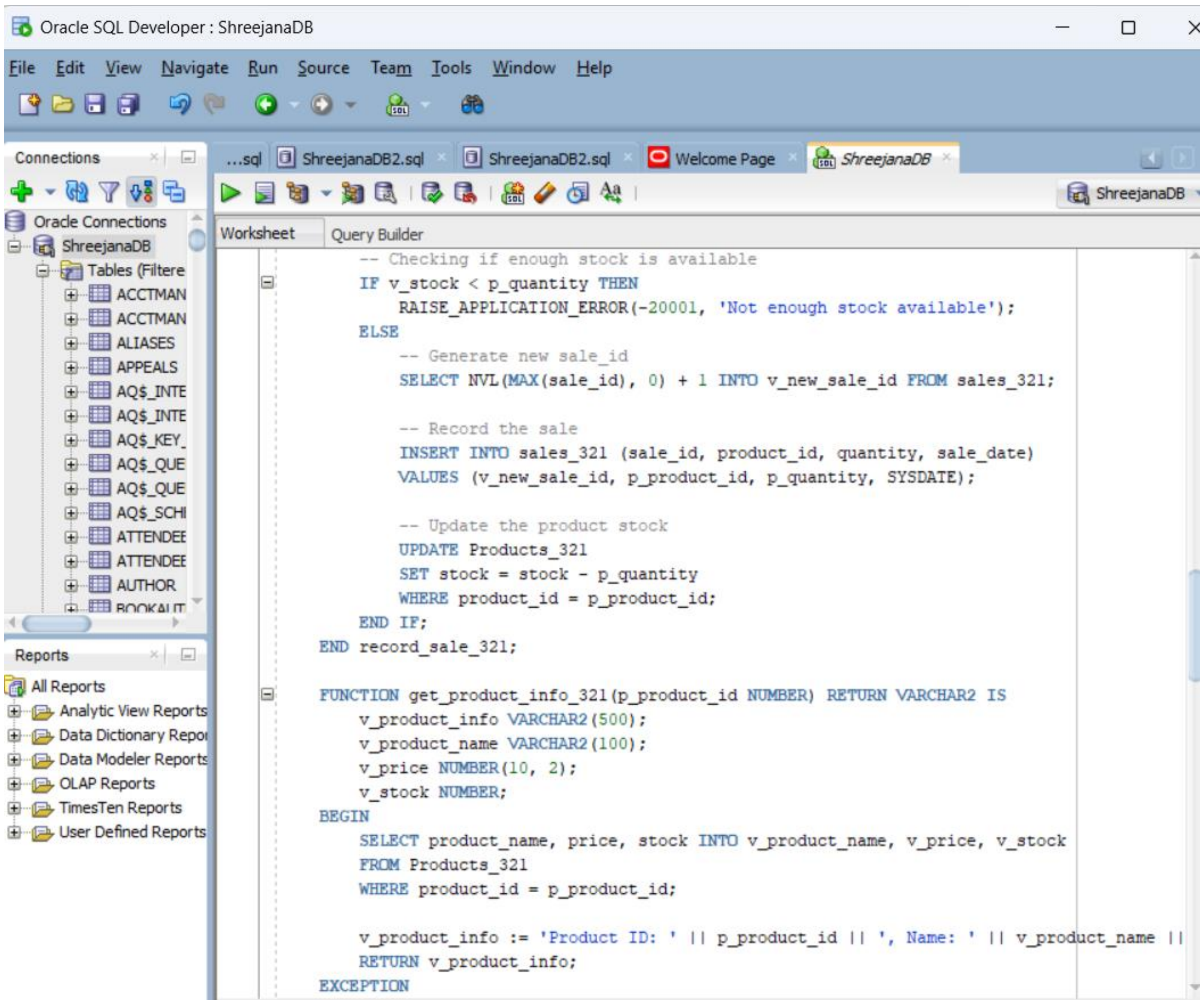
# package body



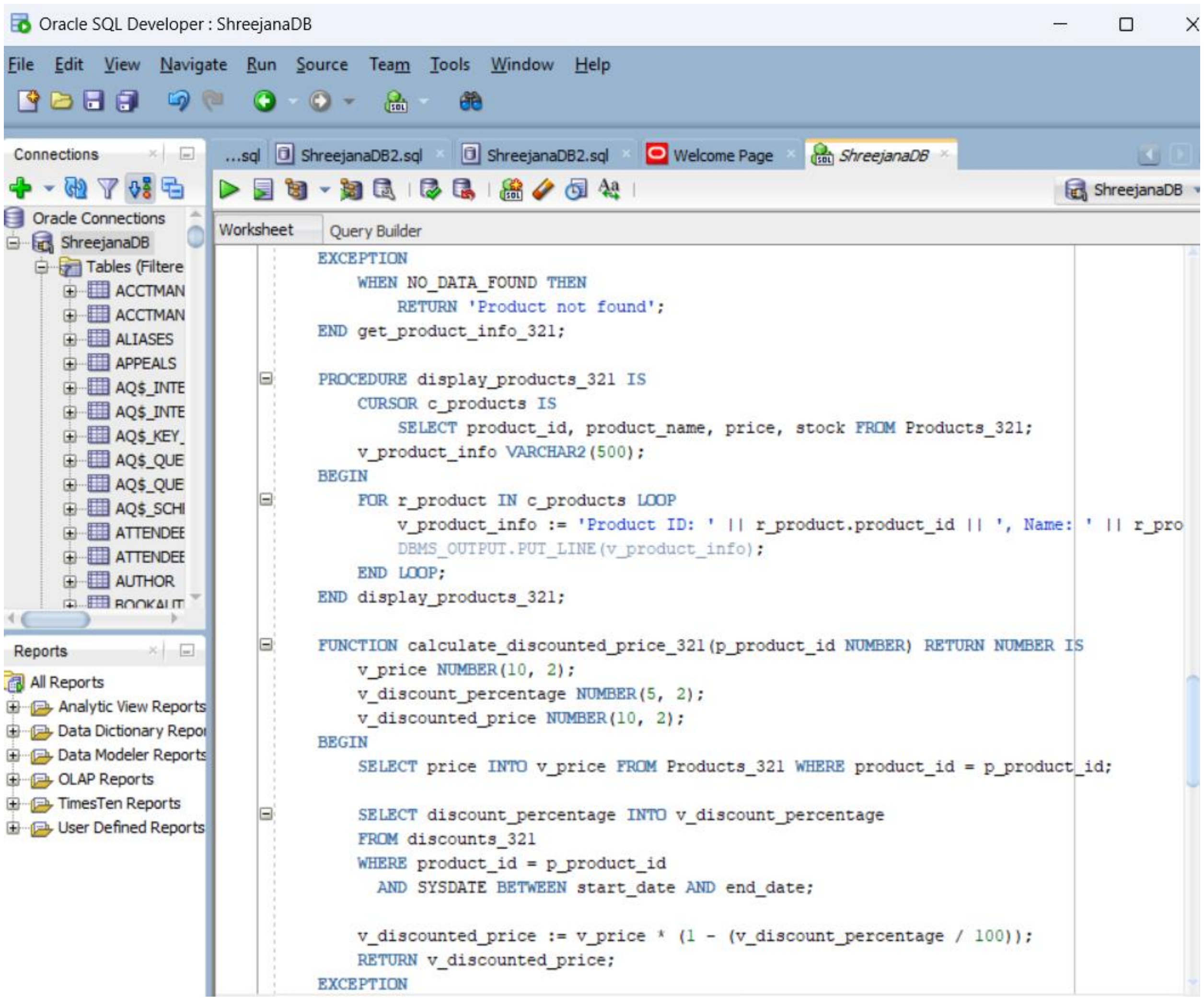
```
CREATE OR REPLACE PACKAGE BODY inventory_pkg IS

    PROCEDURE add_product_321(
            p_product_id NUMBER,
            p_product_name VARCHAR2,
            p_price NUMBER,
            p_stock NUMBER)
    IS
    BEGIN
        INSERT INTO Products_321 (product_id, product_name, price, stock)
        VALUES (p_product_id, p_product_name, p_price, p_stock);
    END add_product_321;

    PROCEDURE update_stock_321(p_product_id NUMBER, p_stock NUMBER) IS
    BEGIN
        UPDATE Products_321
        SET stock = p_stock
        WHERE product_id = p_product_id;
    END update_stock_321;

    PROCEDURE record_sale_321(p_product_id NUMBER, p_quantity NUMBER) IS
        v_stock NUMBER;
        v_new_sale_id NUMBER;
    BEGIN
        -- retrieving current stock
        SELECT stock INTO v_stock FROM Products_321 WHERE product_id = p_product_id;

        -- Checking if enough stock is available
```

```
Task completed in 0.088 seconds

Package Body INVENTORY_PKG compiled
```

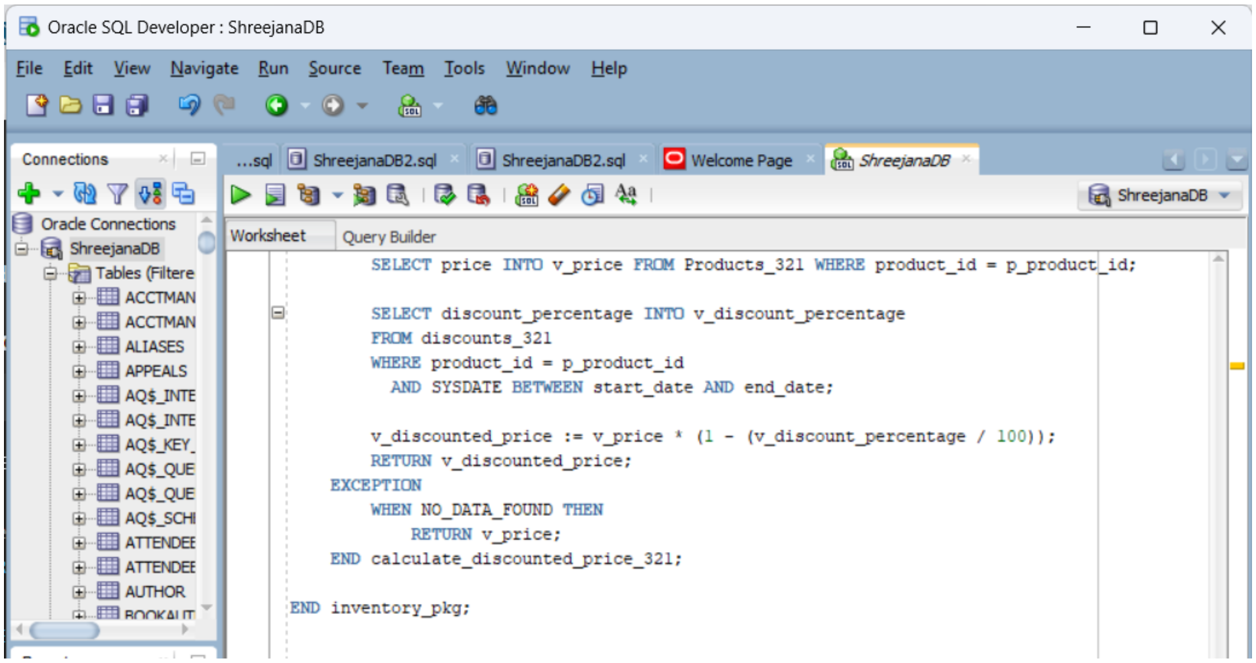
This is the package body that implements all the functions and procedures declared in the above package specification. It includes the procedures to add and update products, record sales and display products. It includes the functions to retrieve product information and calculated discounted prices. In this way, this package ensures the modularity and reusability.



```
        -- Checking if enough stock is available
        IF v_stock < p_quantity THEN
            RAISE_APPLICATION_ERROR(-20001, 'Not enough stock available');
        ELSE
            -- Generate new sale_id
            SELECT NVL(MAX(sale_id), 0) + 1 INTO v_new_sale_id FROM sales_321;

            -- Record the sale
            INSERT INTO sales_321 (sale_id, product_id, quantity, sale_date)
            VALUES (v_new_sale_id, p_product_id, p_quantity, SYSDATE);

            -- Update the product stock
            UPDATE Products_321
            SET stock = stock - p_quantity
            WHERE product_id = p_product_id;
        END IF;
    END record_sale_321;

    FUNCTION get_product_info_321(p_product_id NUMBER) RETURN VARCHAR2 IS
        v_product_info VARCHAR2(500);
        v_product_name VARCHAR2(100);
        v_price NUMBER(10, 2);
        v_stock NUMBER;
    BEGIN
        SELECT product_name, price, stock INTO v_product_name, v_price, v_stock
        FROM Products_321
        WHERE product_id = p_product_id;

        v_product_info := 'Product ID: ' || p_product_id || ', Name: ' || v_product_name ||
        RETURN v_product_info;
    EXCEPTION
```



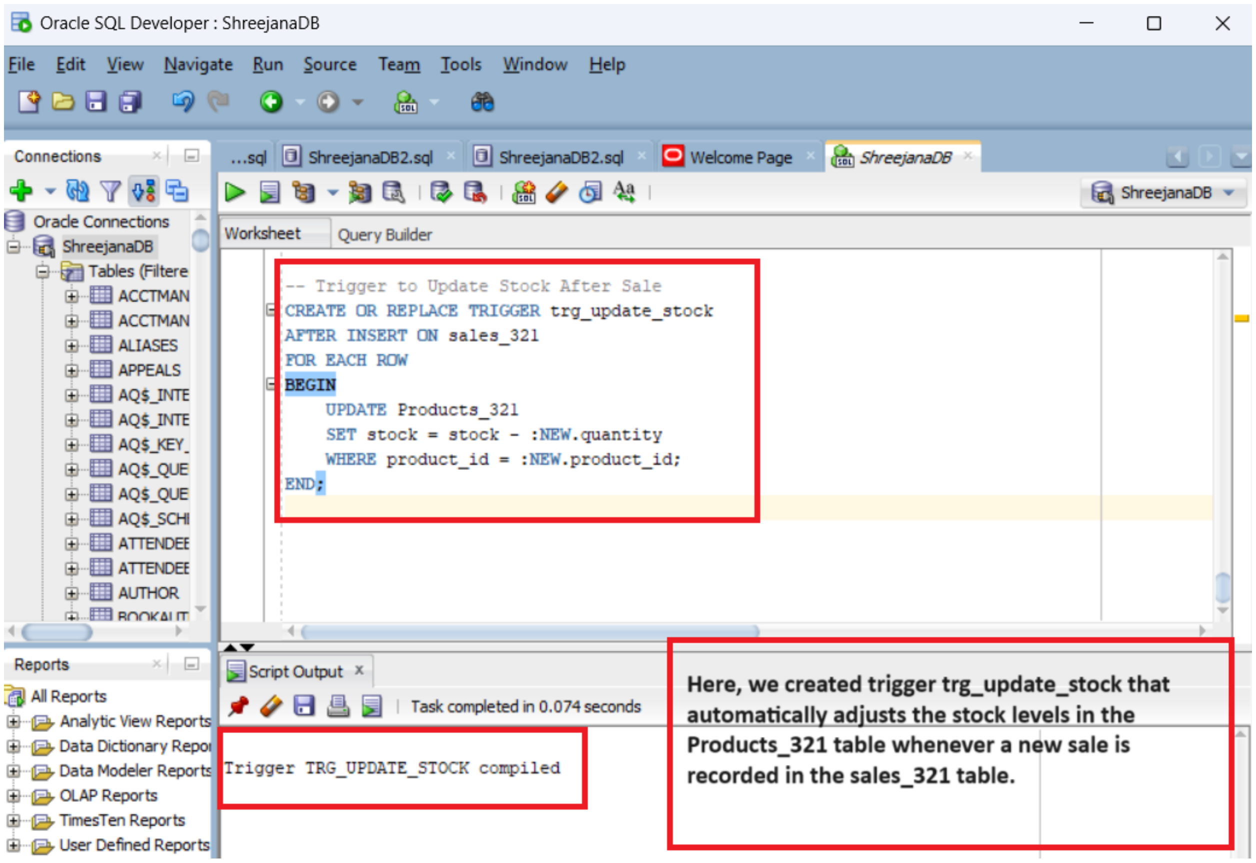
```
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN 'Product not found';
    END get_product_info_321;

    PROCEDURE display_products_321 IS
        CURSOR c_products IS
            SELECT product_id, product_name, price, stock FROM Products_321;
        v_product_info VARCHAR2(500);
    BEGIN
        FOR r_product IN c_products LOOP
            v_product_info := 'Product ID: ' || r_product.product_id || ', Name: ' || r_pro
            DBMS_OUTPUT.PUT_LINE(v_product_info);
        END LOOP;
    END display_products_321;

    FUNCTION calculate_discounted_price_321(p_product_id NUMBER) RETURN NUMBER IS
        v_price NUMBER(10, 2);
        v_discount_percentage NUMBER(5, 2);
        v_discounted_price NUMBER(10, 2);
    BEGIN
        SELECT price INTO v_price FROM Products_321 WHERE product_id = p_product_id;

        SELECT discount_percentage INTO v_discount_percentage
        FROM discounts_321
        WHERE product_id = p_product_id
          AND SYSDATE BETWEEN start_date AND end_date;

        v_discounted_price := v_price * (1 - (v_discount_percentage / 100));
        RETURN v_discounted_price;
    EXCEPTION
```



```
        SELECT price INTO v_price FROM Products_321 WHERE product_id = p_product_id;

        SELECT discount_percentage INTO v_discount_percentage
        FROM discounts_321
        WHERE product_id = p_product_id
          AND SYSDATE BETWEEN start_date AND end_date;

        v_discounted_price := v_price * (1 - (v_discount_percentage / 100));
        RETURN v_discounted_price;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN v_price;
    END calculate_discounted_price_321;

END inventory_pkg;
```

# TASK 3: Triggers

1. Trigger to update the stock



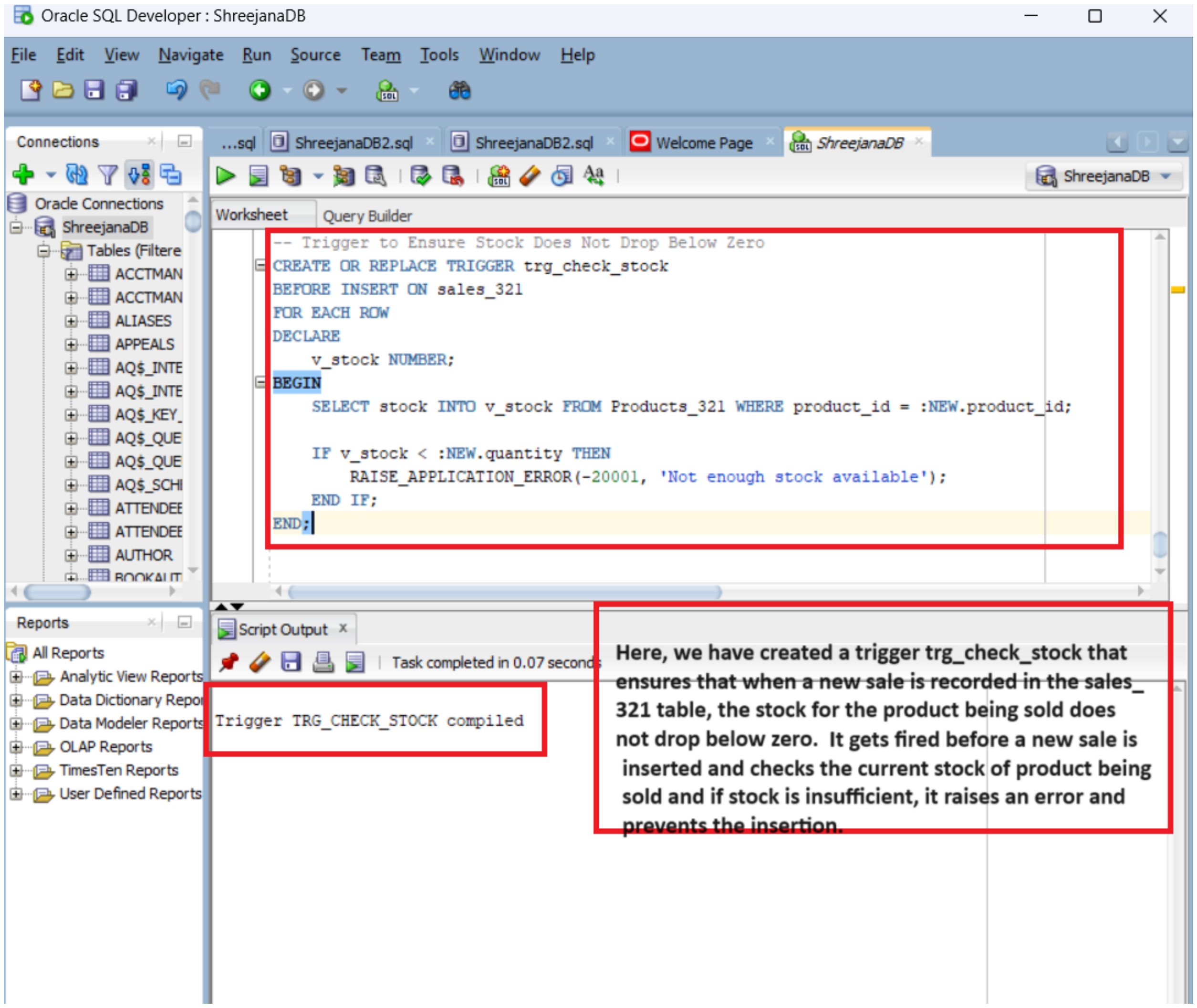
```
-- Trigger to Update Stock After Sale
CREATE OR REPLACE TRIGGER trg_update_stock
AFTER INSERT ON sales_321
FOR EACH ROW
BEGIN
    UPDATE Products_321
    SET stock = stock - :NEW.quantity
    WHERE product_id = :NEW.product_id;
END;
```

Task completed in 0.074 seconds

```
Trigger TRG_UPDATE_STOCK compiled
```

Here, we created trigger trg_update_stock that automatically adjusts the stock levels in the Products_321 table whenever a new sale is recorded in the sales_321 table.

2. Trigger to ensure the stock does not drop below zero

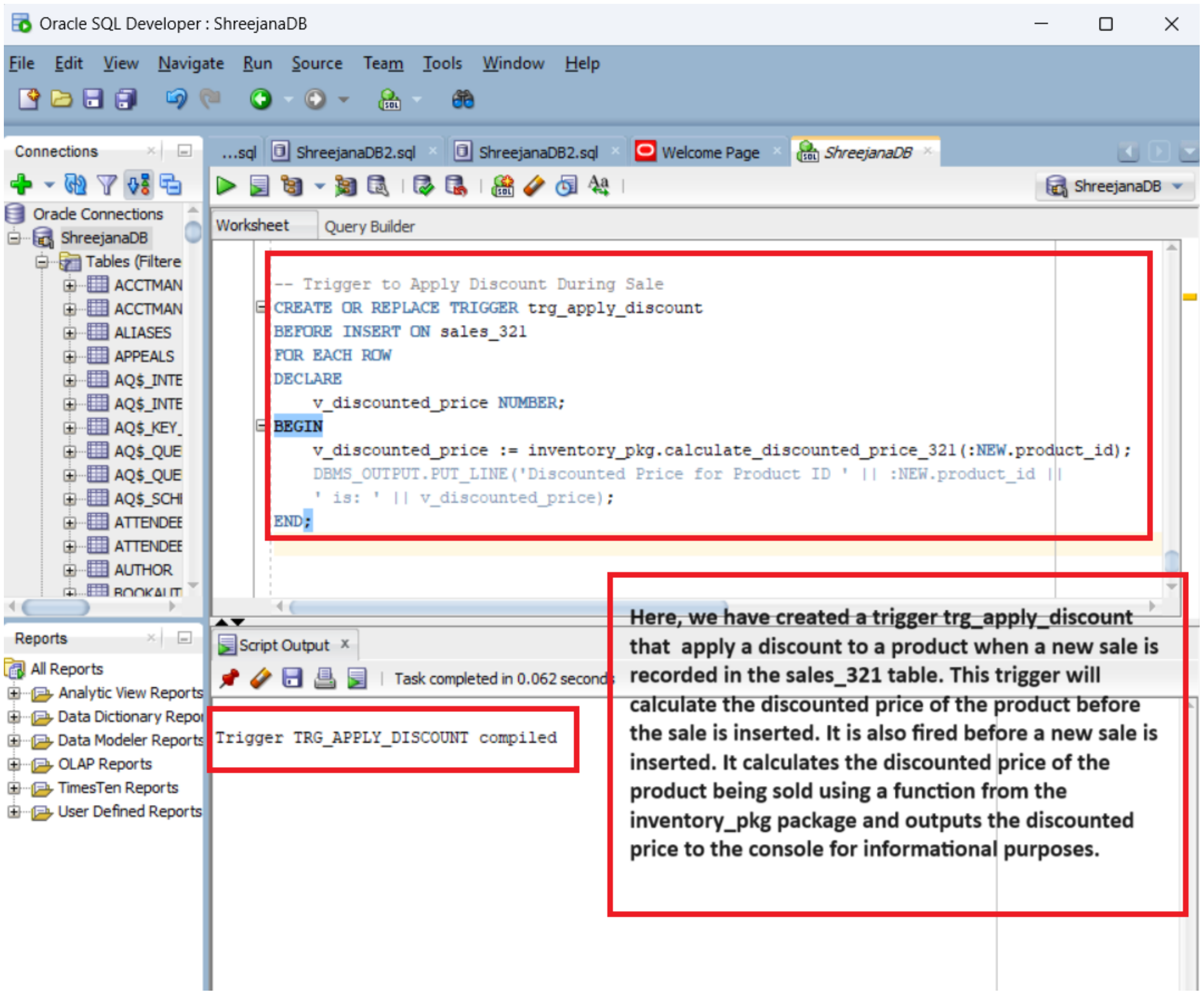


```sql
-- Trigger to Ensure Stock Does Not Drop Below Zero
CREATE OR REPLACE TRIGGER trg_check_stock
BEFORE INSERT ON sales_321
FOR EACH ROW
DECLARE
    v_stock NUMBER;
BEGIN
    SELECT stock INTO v_stock FROM Products_321 WHERE product_id = :NEW.product_id;

    IF v_stock < :NEW.quantity THEN
        RAISE_APPLICATION_ERROR(-20001, 'Not enough stock available');
    END IF;
END;
```

```
Trigger TRG_CHECK_STOCK compiled
```

Here, we have created a trigger trg_check_stock that ensures that when a new sale is recorded in the sales_ 321 table, the stock for the product being sold does not drop below zero. It gets fired before a new sale is inserted and checks the current stock of product being sold and if stock is insufficient, it raises an error and prevents the insertion.

3. Trigger to apply discount using calculate_discounted_price_321 function



```
-- Trigger to Apply Discount During Sale
CREATE OR REPLACE TRIGGER trg_apply_discount
BEFORE INSERT ON sales_321
FOR EACH ROW
DECLARE
    v_discounted_price NUMBER;
BEGIN
    v_discounted_price := inventory_pkg.calculate_discounted_price_321(:NEW.product_id);
    DBMS_OUTPUT.PUT_LINE('Discounted Price for Product ID ' || :NEW.product_id ||
    ' is: ' || v_discounted_price);
END;
```

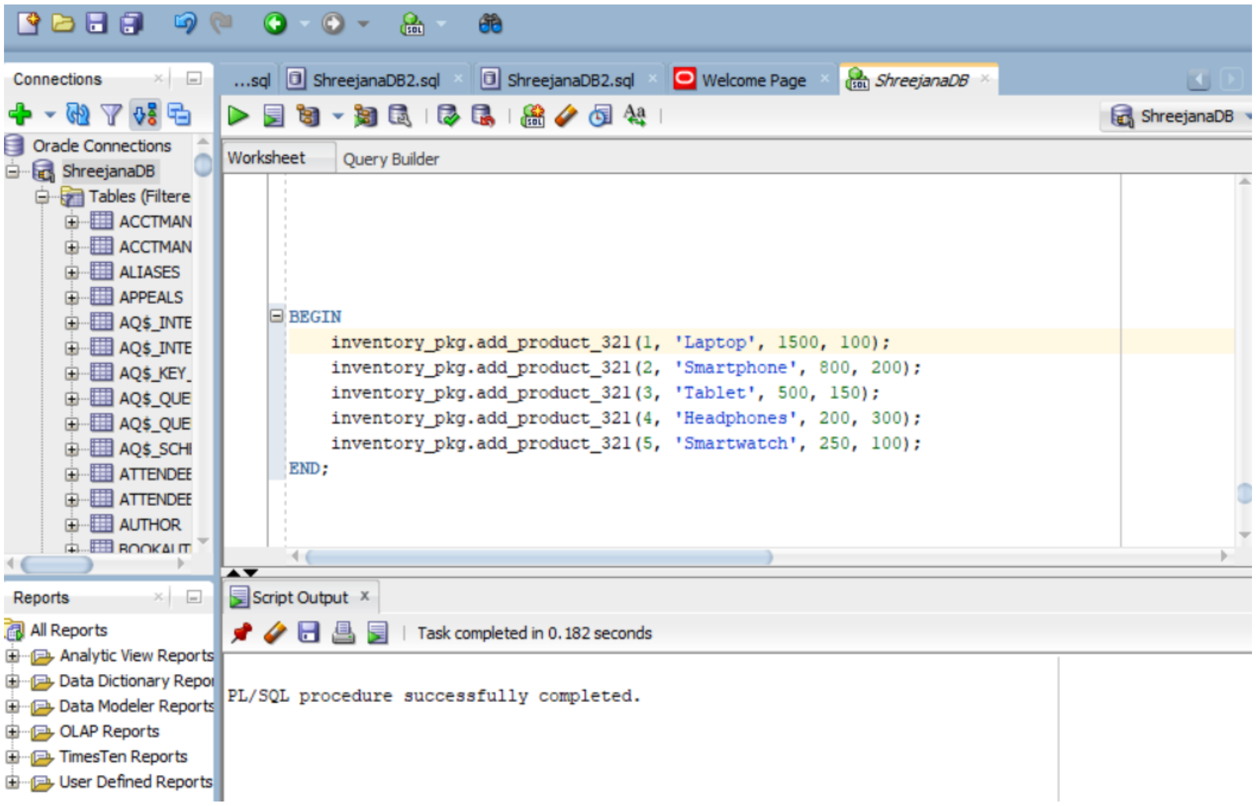
Task completed in 0.062 seconds

```
Trigger TRG_APPLY_DISCOUNT compiled
```

Here, we have created a trigger trg_apply_discount that apply a discount to a product when a new sale is recorded in the sales_321 table. This trigger will calculate the discounted price of the product before the sale is inserted. It is also fired before a new sale is inserted. It calculates the discounted price of the product being sold using a function from the inventory_pkg package and outputs the discounted price to the console for informational purposes.

# Implementation

# Product adding

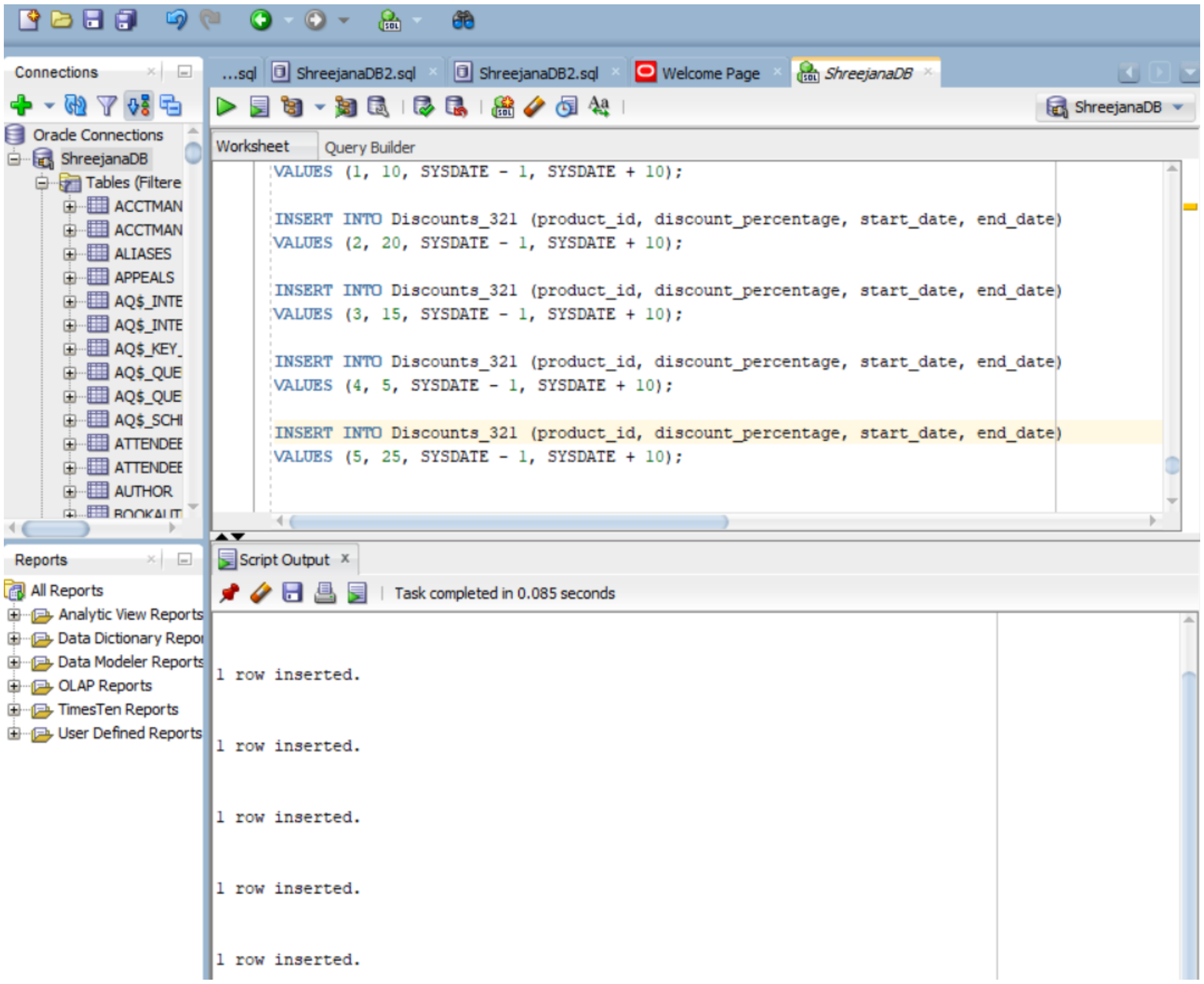


```
BEGIN
    inventory_pkg.add_product_321(1, 'Laptop', 1500, 100);
    inventory_pkg.add_product_321(2, 'Smartphone', 800, 200);
    inventory_pkg.add_product_321(3, 'Tablet', 500, 150);
    inventory_pkg.add_product_321(4, 'Headphones', 200, 300);
    inventory_pkg.add_product_321(5, 'Smartwatch', 250, 100);
END;
```

Task completed in 0.182 seconds

```
PL/SQL procedure successfully completed.
```

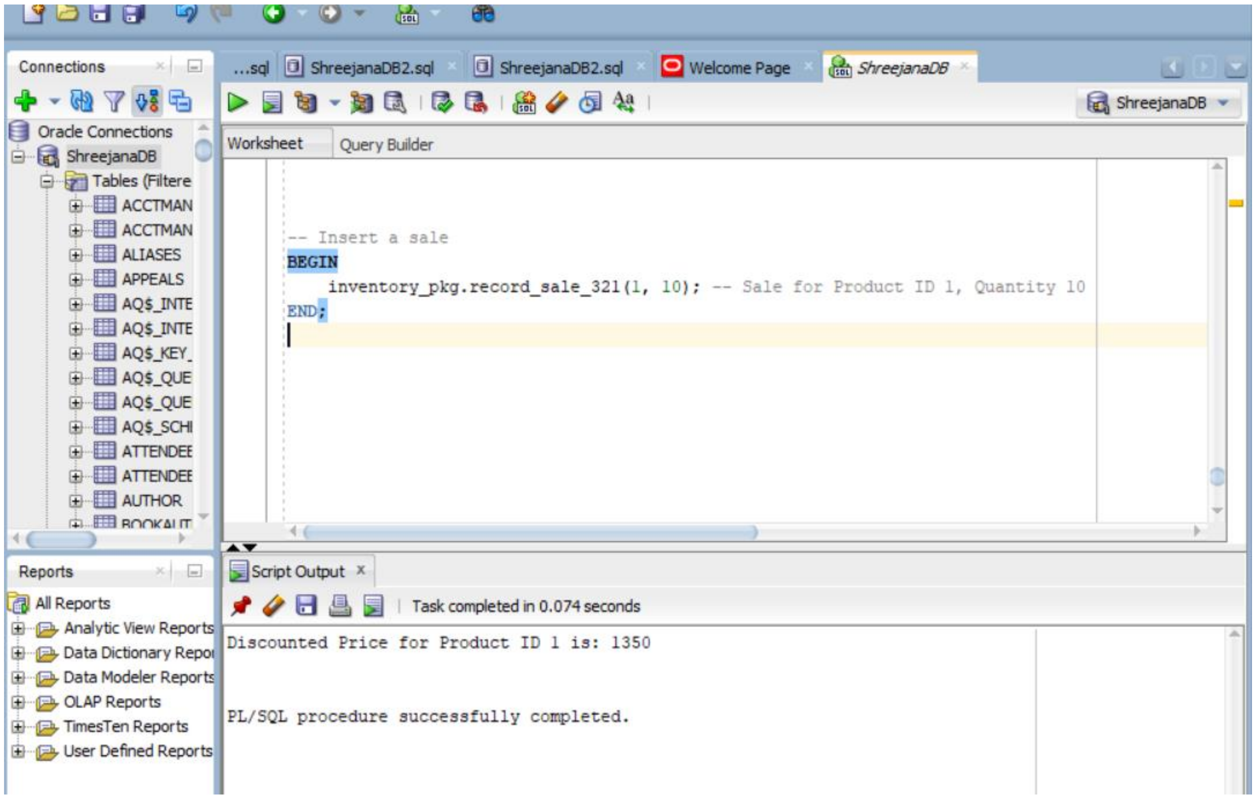
# discounts adding



```sql
VALUES (1, 10, SYSDATE - 1, SYSDATE + 10);

INSERT INTO Discounts_321 (product_id, discount_percentage, start_date, end_date)
VALUES (2, 20, SYSDATE - 1, SYSDATE + 10);

INSERT INTO Discounts_321 (product_id, discount_percentage, start_date, end_date)
VALUES (3, 15, SYSDATE - 1, SYSDATE + 10);

INSERT INTO Discounts_321 (product_id, discount_percentage, start_date, end_date)
VALUES (4, 5, SYSDATE - 1, SYSDATE + 10);

INSERT INTO Discounts_321 (product_id, discount_percentage, start_date, end_date)
VALUES (5, 25, SYSDATE - 1, SYSDATE + 10);
```

```
1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.
```

# Recording sale

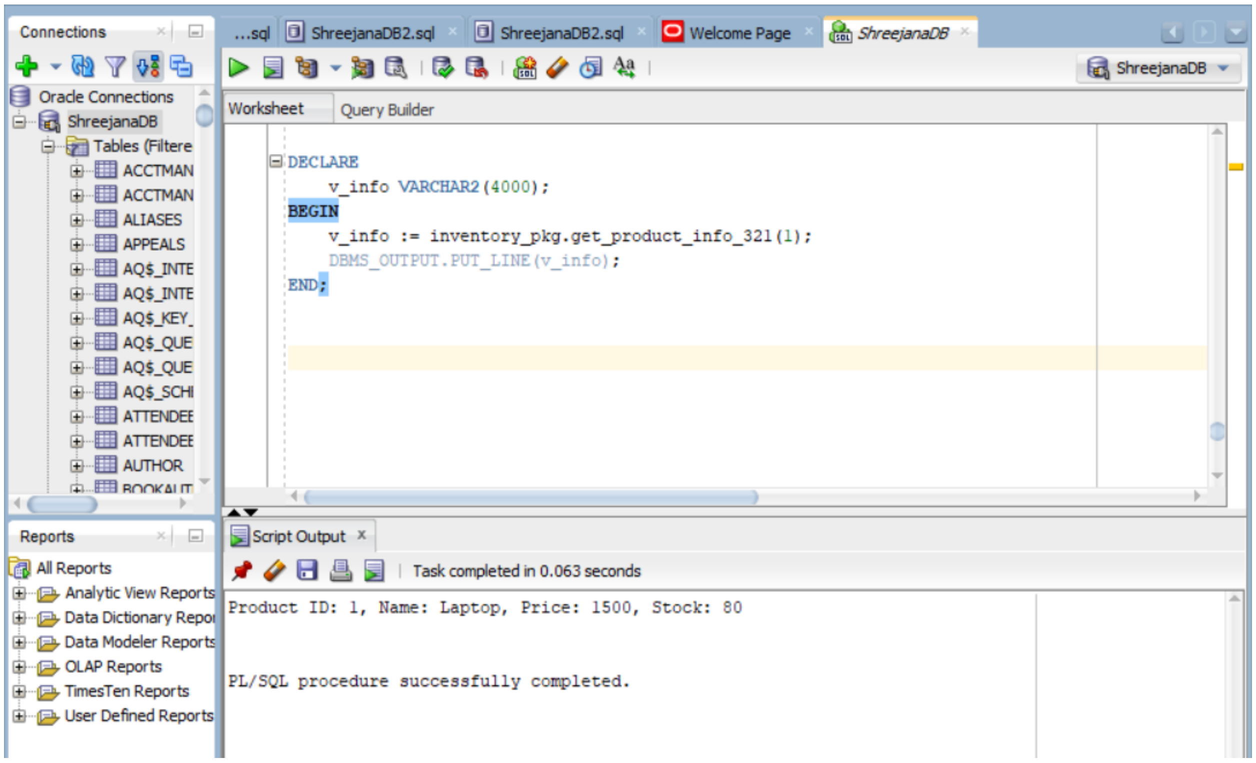


```
-- Insert a sale
BEGIN
    inventory_pkg.record_sale_321(1, 10); -- Sale for Product ID 1, Quantity 10
END;
```

Task completed in 0.074 seconds

```
Discounted Price for Product ID 1 is: 1350


PL/SQL procedure successfully completed.
```

# Displaying Single Product Info

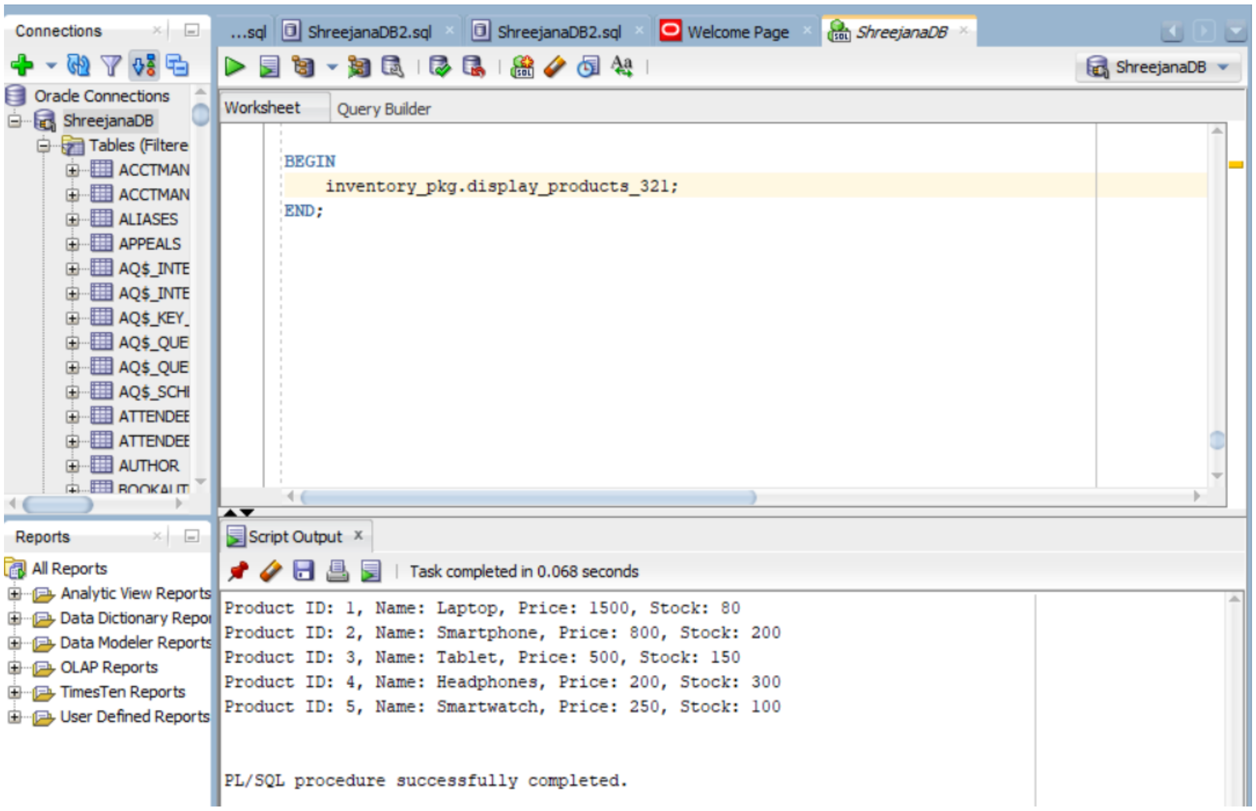


```
DECLARE
    v_info VARCHAR2(4000);
BEGIN
    v_info := inventory_pkg.get_product_info_321(1);
    DBMS_OUTPUT.PUT_LINE(v_info);
END;
```

Task completed in 0.063 seconds

```
Product ID: 1, Name: Laptop, Price: 1500, Stock: 80


PL/SQL procedure successfully completed.
```

Displaying All Products



```
BEGIN
    inventory_pkg.display_products_321;
END;
```

```
Product ID: 1, Name: Laptop, Price: 1500, Stock: 80
Product ID: 2, Name: Smartphone, Price: 800, Stock: 200
Product ID: 3, Name: Tablet, Price: 500, Stock: 150
Product ID: 4, Name: Headphones, Price: 200, Stock: 300
Product ID: 5, Name: Smartwatch, Price: 250, Stock: 100


PL/SQL procedure successfully completed.
```