

CSD 4203 – Database Programming

Student ID : C0930321

Student Name : Shreejana Shrestha

Assignment # 11

TASK 1: Tables creation

The screenshot displays the Oracle SQL Developer interface for a database named 'ShreejanaDB'. The 'Query Builder' window contains the following SQL script:

```
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)  
);
```

The 'Script Output' window at the bottom shows the execution results:

```
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 screenshot displays the Oracle SQL Developer interface with the 'ShreejanaDB' connection. The main workspace shows the SQL script for creating the 'inventory_pkg' package. The script is as follows:

```
-- 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;
    END update_stock_321;

    PROCEDURE record_sale_321(p_product_id NUMBER, p_quantity NUMBER) IS
    BEGIN
        UPDATE Products_321
        SET stock = stock - p_quantity
        WHERE product_id = p_product_id;
    END record_sale_321;

    FUNCTION get_product_info_321(p_product_id NUMBER) RETURN VARCHAR2 IS
    BEGIN
        SELECT product_name || ' - ' || price || ' - ' || stock
        FROM Products_321
        WHERE product_id = p_product_id;
    END get_product_info_321;

    PROCEDURE display_products_321 IS
    BEGIN
        SELECT * FROM Products_321;
    END display_products_321;

    FUNCTION calculate_discounted_price_321(p_product_id NUMBER) RETURN NUMBER IS
    BEGIN
        SELECT price * 0.9
        FROM Products_321
        WHERE product_id = p_product_id;
    END calculate_discounted_price_321;

END inventory_pkg;
```

The 'Script Output' window at the bottom shows the message: 'Package INVENTORY_PKG compiled'.

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 body

The screenshot displays the Oracle SQL Developer interface with the 'ShreejanaDB' connection. The 'Connections' pane on the left shows the database structure, including tables like ACCTMAN, ALIASES, APPEALS, and others. The main workspace shows the SQL script for creating or replacing the package body for 'inventory_pkg'. The script defines three procedures: 'add_product_321', 'update_stock_321', and 'record_sale_321'. The 'Script Output' pane at the bottom shows the message 'Package Body INVENTORY_PKG compiled', indicating successful compilation.

```
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
```

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.

Package Body INVENTORY_PKG compiled

Oracle SQL Developer : ShreejanaDB

File Edit View Navigate Run Source Team Tools Window Help

Connections

Oracle Connections

ShreejanaDB

Tables (Filtered)

- ACCTMAN
- ACCTMAN
- ALIASES
- APPEALS
- AQ\$ INTE
- AQ\$ INTE
- AQ\$ KEY
- AQ\$ QUE
- AQ\$ SCHI
- ATTENDEE
- ATTENDEE
- AUTHOR
- BOOK

Reports

- All Reports
- Analytic View Reports
- Data Dictionary Reports
- Data Modeler Reports
- OLAP Reports
- TimesTen Reports
- User Defined Reports

Worksheet

```

-- 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

```

Oracle SQL Developer : ShreejanaDB

File Edit View Navigate Run Source Team Tools Window Help

Connections

Oracle Connections

ShreejanaDB

Tables (Filter)

- ACCTMAN
- ACCTMAN
- ALIASES
- APPEALS
- AQ\$ INTE
- AQ\$ INTE
- AQ\$ KEY
- AQ\$ QUE
- AQ\$ QUE
- AQ\$ SCHI
- ATTENDEE
- ATTENDEE
- AUTHOR
- BOOK&IT

Reports

- All Reports
- Analytic View Reports
- Data Dictionary Report
- Data Modeler Reports
- OLAP Reports
- TimesTen Reports
- User Defined Reports

Worksheet

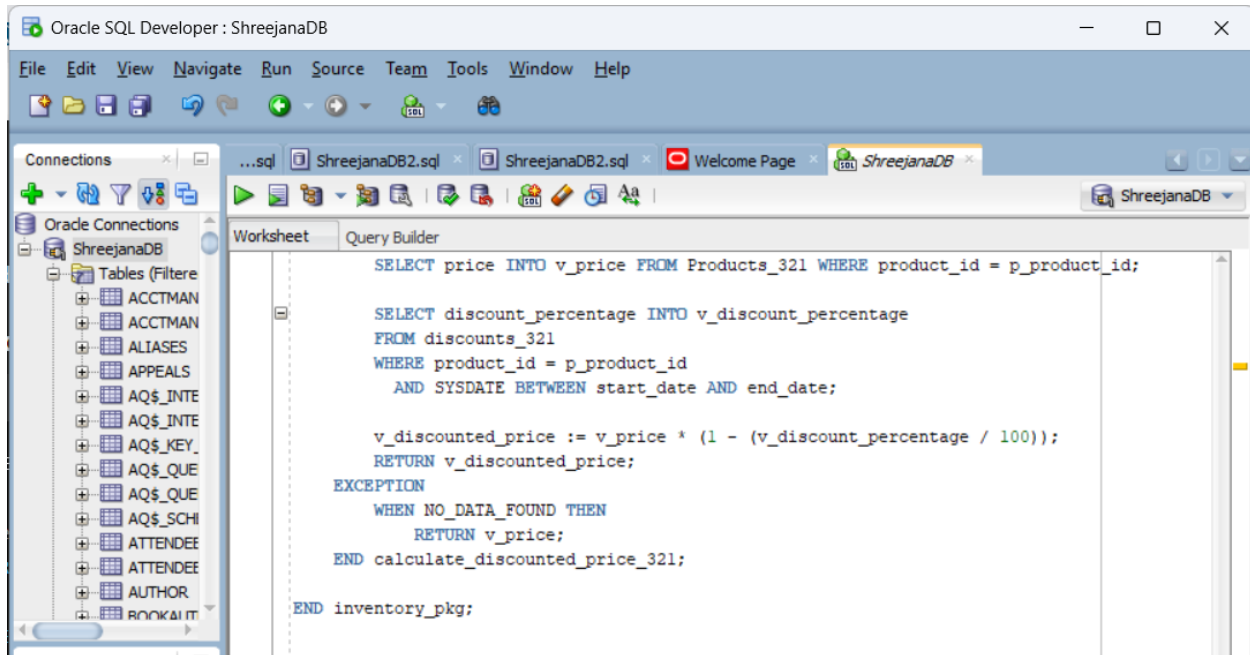
```
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN 'Product not found';
END get_product_info_321;

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
```

TASK 3: Triggers

1. Trigger to update the stock

The screenshot displays the Oracle SQL Developer interface with the 'ShreejanaDB' connection selected. The 'Query Builder' tab is active, showing the following SQL script:

```
-- 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;
```

The 'Script Output' window at the bottom shows the message: 'Trigger TRG_UPDATE_STOCK compiled'. A status bar indicates 'Task completed in 0.074 seconds'.

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

The screenshot displays the Oracle SQL Developer interface for a database named 'ShreejanaDB'. The 'Connections' pane on the left shows the database structure, including tables like ACCTMAN, ALIASES, APPEALS, and others. The main workspace is divided into a 'Worksheet' and a 'Query Builder'. The 'Worksheet' contains the following SQL code:

```
-- 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;
```

The 'Script Output' pane at the bottom shows the message: 'Trigger TRG_CHECK_STOCK compiled'. A red box highlights this message.

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

The screenshot displays the Oracle SQL Developer interface for the 'ShreejanaDB' connection. The main window shows a SQL script in the 'Query Builder' tab. The script is a PL/SQL trigger named 'trg_apply_discount' that is created or replaced. It is a BEFORE INSERT trigger on the 'sales_321' table, firing for each row. The trigger declares a variable 'v_discounted_price' of type NUMBER. In the BEGIN block, it calls the 'inventory_pkg.calculate_discounted_price_321' function with ':NEW.product_id' as the argument. It then uses 'DBMS_OUTPUT.PUT_LINE' to output a message showing the product ID and the calculated discounted price. The trigger ends with 'END;'. A red box highlights the entire script.

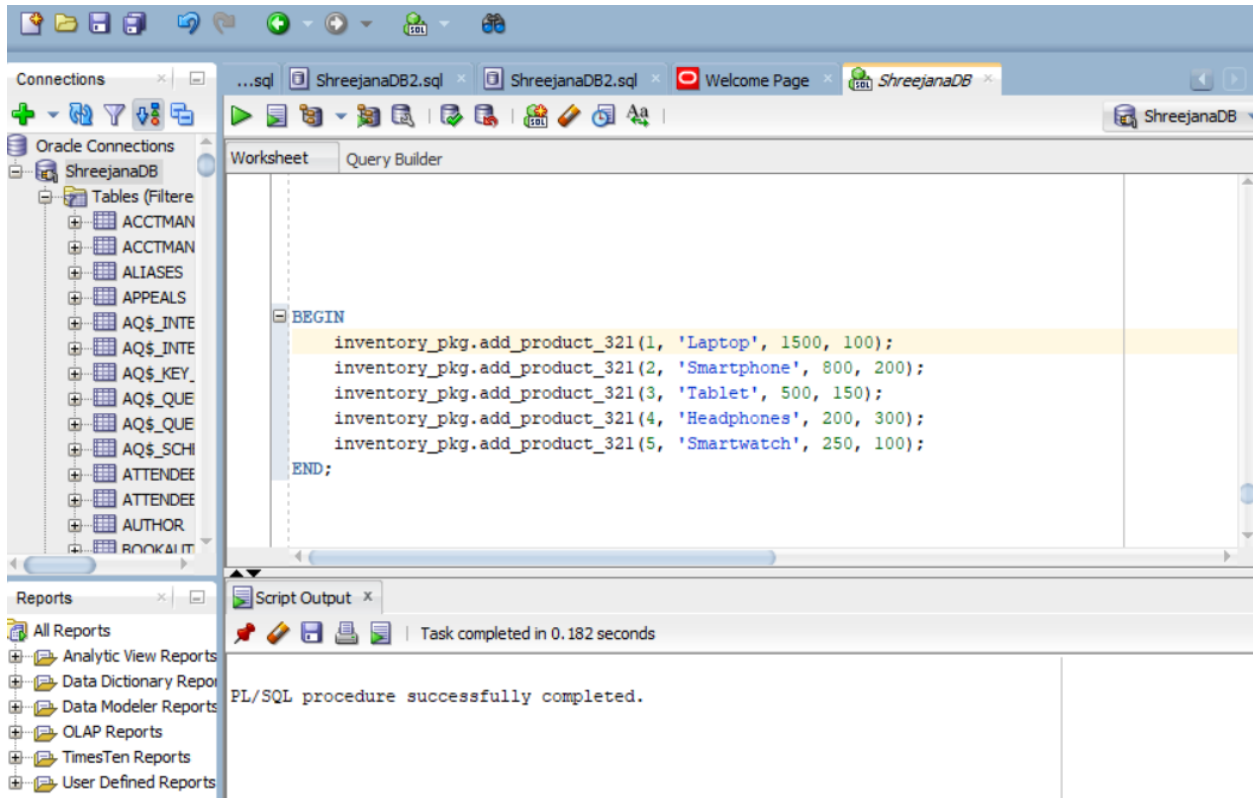
```
-- Trigger to Apply Discount During Sale
CREATE OR REPLACE TRIGGER trg_apply_discount
BEFORE INSERT ON sales_321
FOR EACH ROW
DECLARE
    v_discounted_price NUMBER;
BEGIN
    v_discounted_price := inventory_pkg.calculate_discounted_price_321(:NEW.product_id);
    DBMS_OUTPUT.PUT_LINE('Discounted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

Below the script, the 'Script Output' window shows the message: 'Trigger TRG_APPLY_DISCOUNT compiled'. A red box highlights this message.

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



discounts adding

The screenshot displays the Oracle SQL Developer interface. The top pane shows a SQL script with five INSERT statements into the Discounts_321 table. The bottom pane shows the script output, indicating that each row was successfully inserted.

Connections: ShreejanaDB

Tables (Filtered): ACCTMAN, ACCTMAN, ALIASES, APPEALS, AQ\$_INTE, AQ\$_INTE, AQ\$_KEY, AQ\$_QUE, AQ\$_QUE, AQ\$_SCHI, ATTENDEE, ATTENDEE, AUTHOR, BOOK&IT

Worksheet: Query Builder

```
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);
```

Script Output: Task completed in 0.085 seconds

```
1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.
```

Recording sale

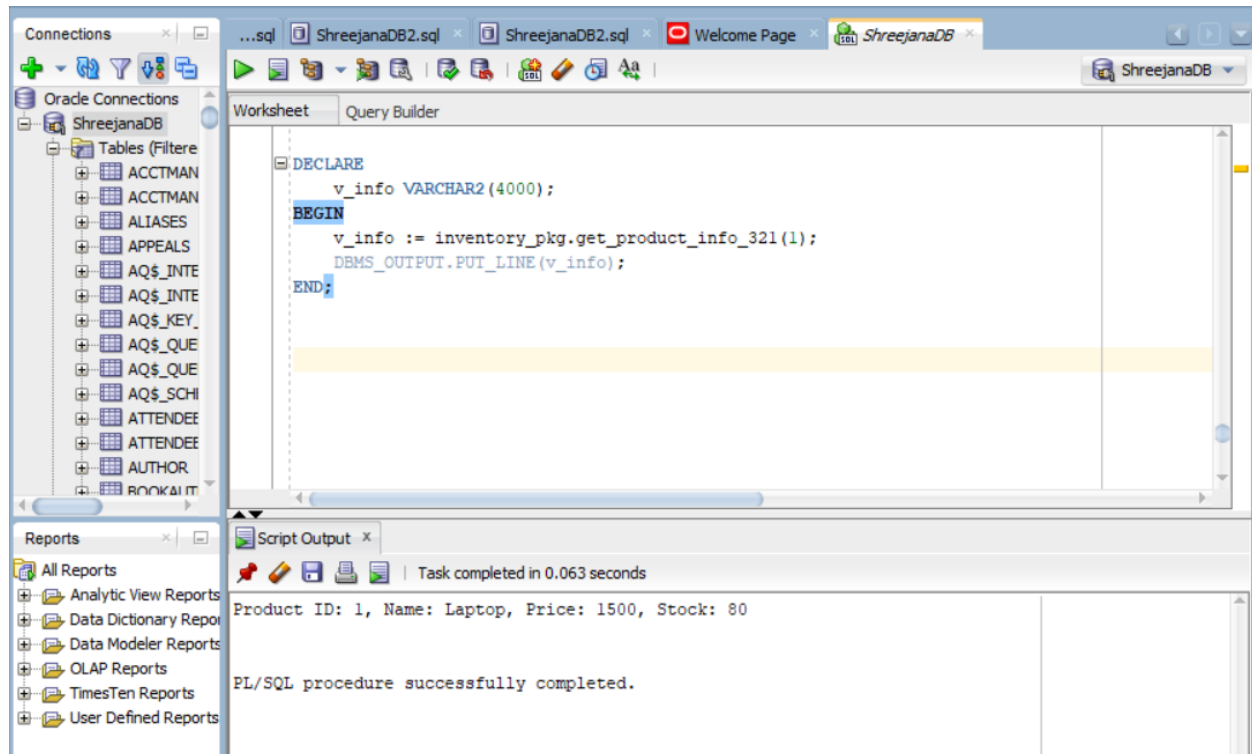
The screenshot displays the Oracle SQL Developer environment. The top toolbar includes icons for file operations, execution, and formatting. The main window is divided into several panes:

- Connections:** Shows the 'ShreejanaDB' connection selected under 'Oracle Connections'.
- Tables (Filter):** Lists various tables including ACCTMAN, ALIASES, APPEALS, AQ\$_INTE, AQ\$_KEY_, AQ\$_QUE, AQ\$_SCHI, ATTENDEE, AUTHOR, and BOOK&IT.
- Worksheet:** Contains a SQL script with the following code:

```
-- Insert a sale
BEGIN
inventory_pkg.record_sale_321(1, 10); -- Sale for Product ID 1, Quantity 10
END;
```
- Script Output:** Displays the results of the execution:

```
Task completed in 0.074 seconds
Discounted Price for Product ID 1 is: 1350
PL/SQL procedure successfully completed.
```

Displaying Single Product Info



The screenshot displays the Oracle SQL Developer interface. On the left, the 'Connections' pane shows 'ShreejanaDB' selected. Below it, the 'Tables (Filter)' list includes ACCTMAN, ALIASES, APPEALS, AQ\$_INTE, AQ\$_KEY_, AQ\$_QUE, AQ\$_SCHI, ATTENDEE, AUTHOR, and BOOKAID. The main workspace is divided into 'Worksheet' and 'Query Builder' tabs. The 'Worksheet' tab contains a PL/SQL procedure:

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

Below the code editor, the 'Script Output' pane shows the execution results:

Task completed in 0.063 seconds

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

PL/SQL procedure successfully completed.

Displaying All Products

The screenshot displays the Oracle SQL Developer environment. The 'Connections' pane on the left shows a connection to 'ShreejanaDB'. The 'Query Builder' pane is active, showing a PL/SQL block with the following code:

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

The 'Script Output' pane at the bottom shows the results of the procedure execution, indicating that the task completed in 0.068 seconds. The output lists five products with their IDs, names, prices, and stock levels:

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

Below the table, the message 'PL/SQL procedure successfully completed.' is displayed.