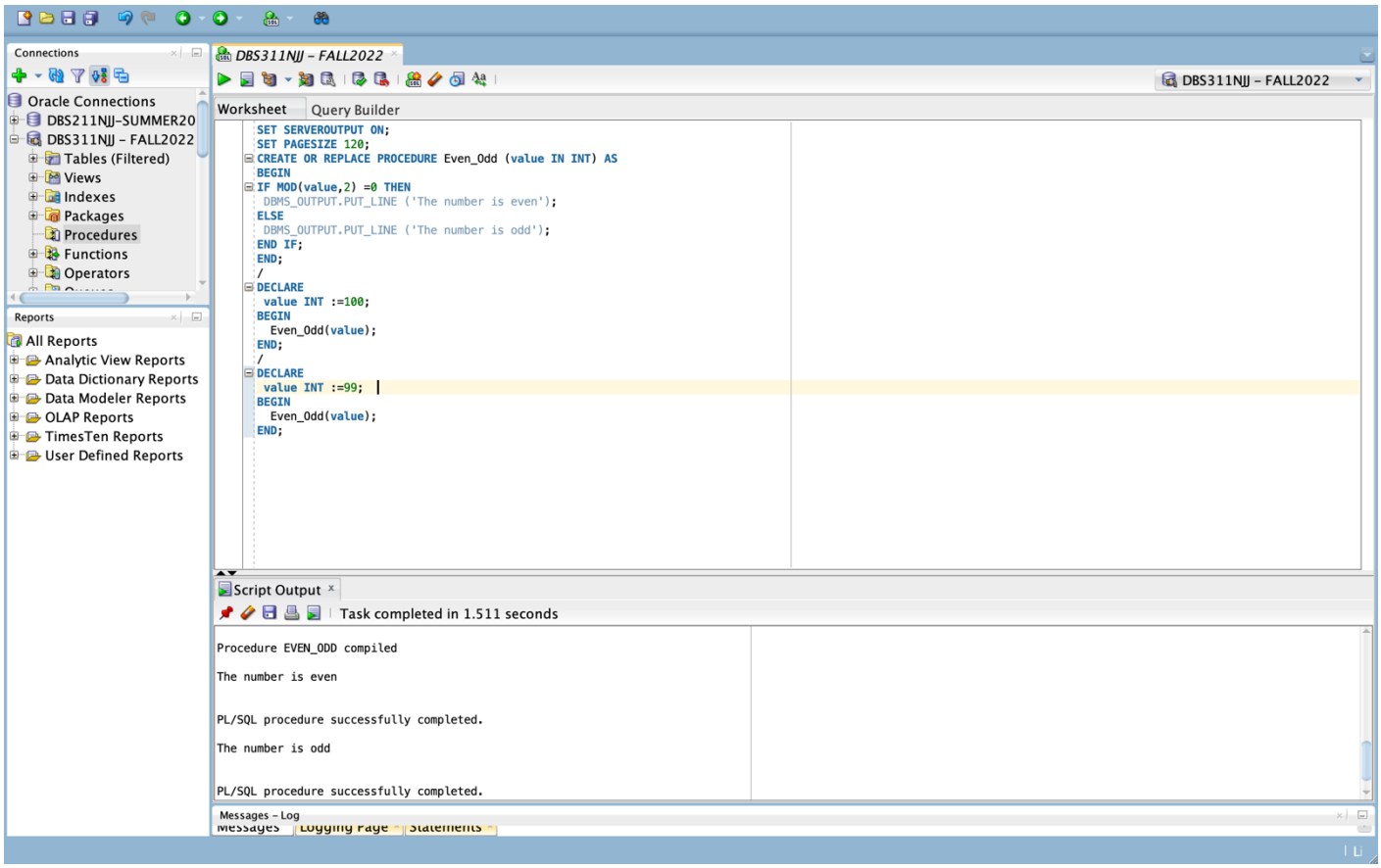
DBS311 Lab 6

1. Write a store procedure called *Even_Odd* that gets an integer number and prints
   The number is even.
   If a number is divisible by 2.
   Otherwise, it prints
   The number is odd.

Show testing with one even and one odd integer.

```
SET SERVEROUTPUT ON;
SET PAGESIZE 120;
CREATE OR REPLACE PROCEDURE Even_Odd (value IN INT) AS
BEGIN
IF MOD(value,2) =0 THEN
 DBMS_OUTPUT.PUT_LINE ('The number is even');
ELSE
 DBMS_OUTPUT.PUT_LINE ('The number is odd');
END IF;
END;
/
DECLARE
 value INT :=100;
BEGIN
  Even_Odd(value);
END;
/
DECLARE
 value INT :=99;
BEGIN
  Even_Odd(value);
END;
```

```
Procedure EVEN_ODD compiled

The number is even


PL/SQL procedure successfully completed.

The number is odd


PL/SQL procedure successfully completed.
```



2. Create a stored procedure named *Find_Employee*. This procedure gets an employee number and prints the following employee information:
   First name
   Last name
   Email
   Phone
   Hire date
   Job title

   The procedure gets a value as the employeeID of type NUMBER.
   See the following example for employeeID 107:

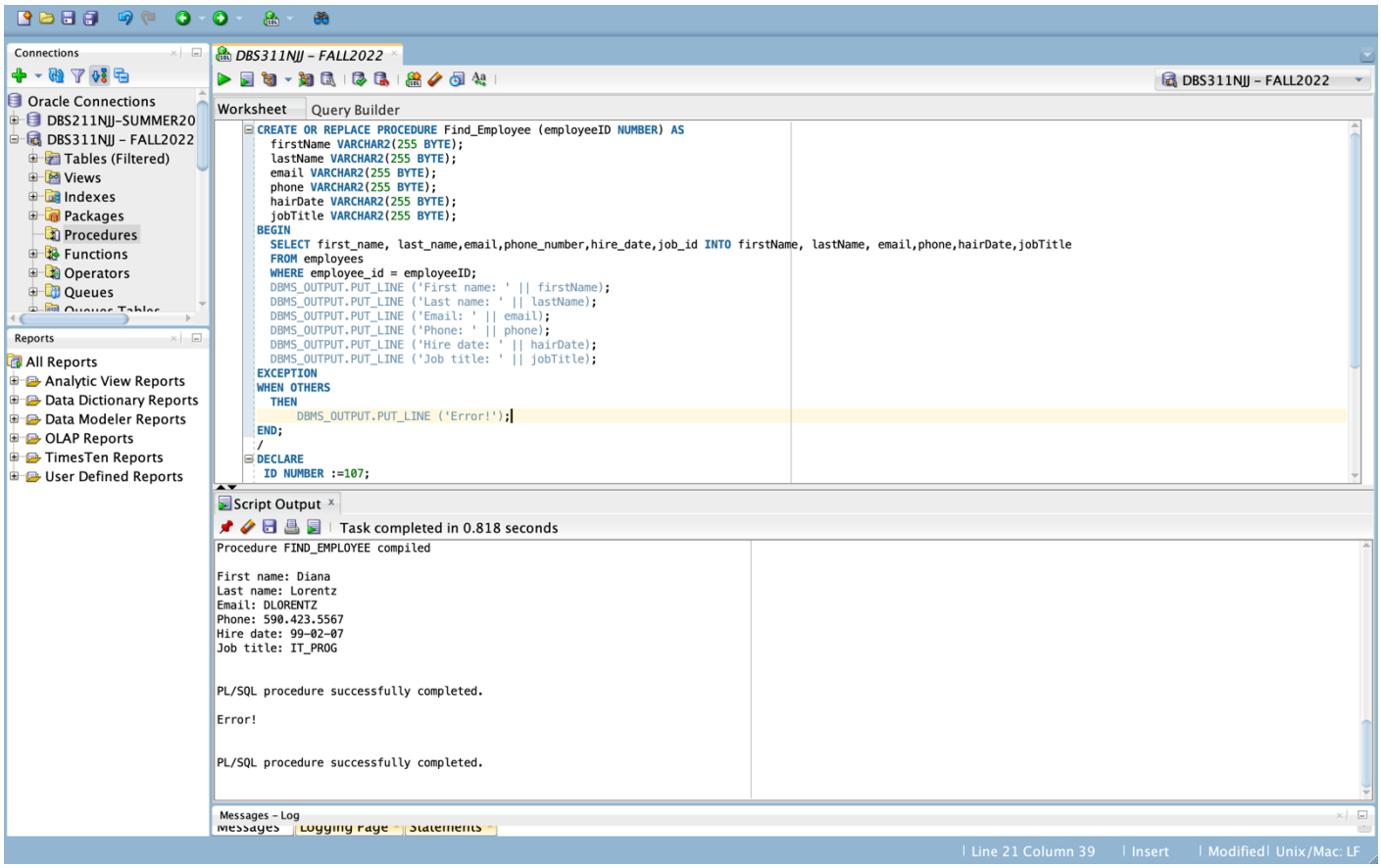
```
First name: Summer
Last name: Payn
Email: summer.payne@example.com
Phone: 515.123.8181
Hire date: 07-JUN-16
Job title: Public Accountant
```

   The procedure displays a proper error message if any error occurs.
   Show testing with one invalid employee Id and one valid Id.

```
CREATE OR REPLACE PROCEDURE Find_Employee (employeeID NUMBER) AS
  firstName VARCHAR2(255 BYTE);
  lastName VARCHAR2(255 BYTE);
  email VARCHAR2(255 BYTE);
  phone VARCHAR2(255 BYTE);
  hairDate VARCHAR2(255 BYTE);
```

```
  jobTitle VARCHAR2(255 BYTE);
BEGIN
  SELECT first_name, last_name,email,phone_number,hire_date,job_id
  INTO firstName, lastName, email,phone,hairDate,jobTitle
  FROM employees
  WHERE employee_id = employeeID;
  DBMS_OUTPUT.PUT_LINE ('First name: ' || firstName);
  DBMS_OUTPUT.PUT_LINE ('Last name: ' || lastName);
  DBMS_OUTPUT.PUT_LINE ('Email: ' || email);
  DBMS_OUTPUT.PUT_LINE ('Phone: ' || phone);
  DBMS_OUTPUT.PUT_LINE ('Hire date: ' || hairDate);
  DBMS_OUTPUT.PUT_LINE ('Job title: ' || jobTitle);
EXCEPTION
WHEN OTHERS
  THEN
      DBMS_OUTPUT.PUT_LINE ('Error!');
END;
/
DECLARE
 ID NUMBER :=107;
BEGIN
  Find_Employee(ID);
END;
/
DECLARE
 ID NUMBER :=99999;
BEGIN
  Find_Employee(ID);
END;
```

```
Procedure FIND_EMPLOYEE compiled

First name: Summer
Last name: Payne
Email: summer.payne@example.com
Phone: 515.123.8181
Hire date: 16-06-07
Job title: Public Accountant


PL/SQL procedure successfully completed.

Error!


PL/SQL procedure successfully completed.
```



3. Every year, the company increases the price of all products in one category. For example, the company wants to increase the price (list_price) of products in category 1 by $5. Write a procedure named *Update_Price_by_Cat* to update the price of all products in a given category and the given amount to be added to the current price if the price is greater than 0. The procedure shows the number of updated rows if the update is successful or shows 0 rows updated, if the input was an invalid category Id.
   The procedure gets two parameters:
   - category_id IN NUMBER
   - amount NUMBER(9,2)

   To define the type of variables that store values of a table' column, you can also write:

   ```
   vriable_name table_name.column_name%type;
   ```

   The above statement defines a variable of the same type as the type of the table' column.

   ```
   category_id products.category_id%type;
   ```

   Or you need to see the table definition to find the type of the category_id column. Make sure the type of your variable is compatible with the value that is stored in your variable.

   To show the number of affected rows the update query, declare a variable named `rows_updated` of type NUMBER and use the SQL variable `sql%rowcount` to set your variable. Then, print its value in your stored procedure.

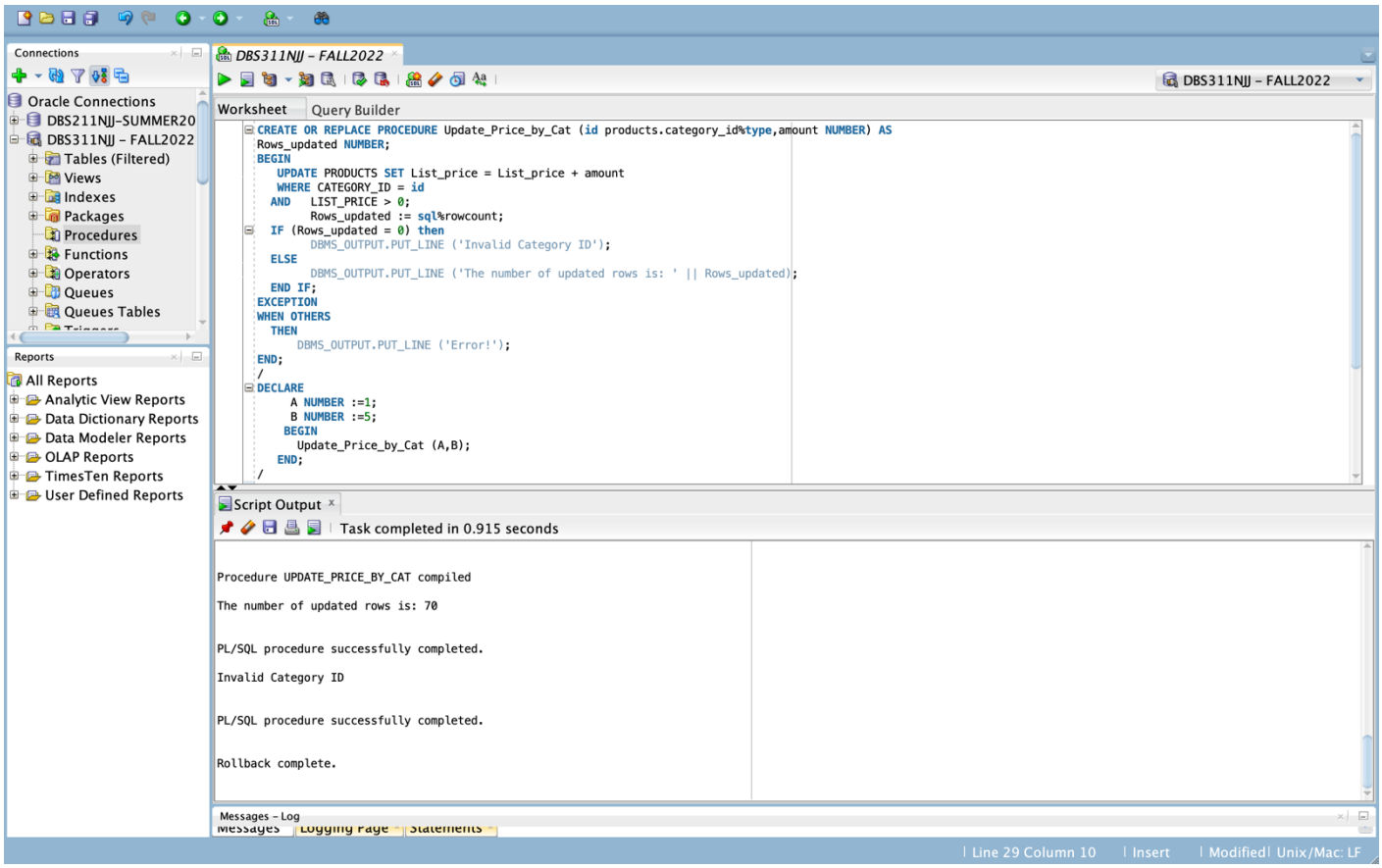
   ```
   Rows_updated := sql%rowcount;
   ```

   SQL%*ROWCOUNT* stores the number of rows affected by an INSERT, UPDATE, or DELETE.

Show testing with one invalid category Id and one valid Id.
Undo your Update > Rollback

```
CREATE OR REPLACE PROCEDURE Update_Price_by_Cat (id
products.category_id%type,amount NUMBER) AS
Rows_updated NUMBER;
BEGIN
  UPDATE PRODUCTS SET List_price = List_price + amount
  WHERE CATEGORY_ID = id
 AND   LIST_PRICE > 0;
      Rows_updated := sql%rowcount;
 IF (Rows_updated = 0) then
      DBMS_OUTPUT.PUT_LINE ('Invalid Category ID');
 ELSE
      DBMS_OUTPUT.PUT_LINE ('The number of updated rows is: ' || Rows_updated);
 END IF;
EXCEPTION
WHEN OTHERS
 THEN
    DBMS_OUTPUT.PUT_LINE ('Error!');
END;
/
DECLARE
    A NUMBER :=1;
    B NUMBER :=5;
   BEGIN
     Update_Price_by_Cat (A,B);
  END;
/
DECLARE
    A NUMBER :=99999;
    B NUMBER :=5;
   BEGIN
     Update_Price_by_Cat (A,B);
  END;
/
ROLLBACK;
```

```
Procedure UPDATE_PRICE_BY_CAT compiled

The number of updated rows is: 70


PL/SQL procedure successfully completed.

Invalid Category ID


PL/SQL procedure successfully completed.


Rollback complete.
```



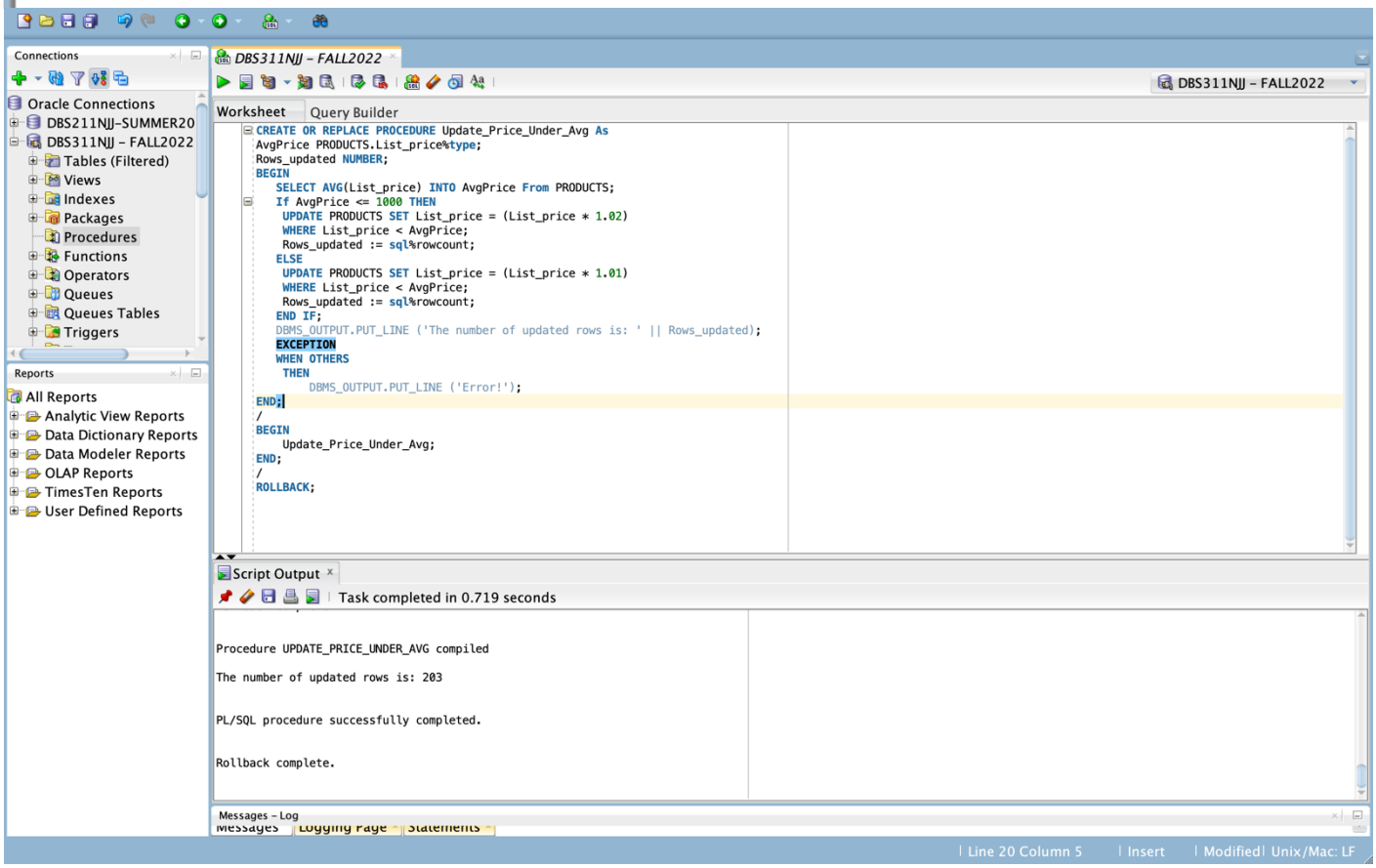
4. Every year, the company increase the price of products whose price is less than the average price of all products by 1%. (list_price * 1.01). Write a stored procedure named *Update_Price_Under_Avg*. This procedure does not have any parameters. You need to find the average price of all products and store it into a variable of the same type. If the average price is less than or equal to $1000, update products' price by 2% if the price of the product is less than the calculated average. If the average price is greater than $1000, update products' price by 1% if the price of the product is less than the calculated average. The query displays an error message if any error occurs. Otherwise, it displays the number of updated rows.

Show your testing.
Undo your Update > Rollback

```
CREATE OR REPLACE PROCEDURE Update_Price_Under_Avg As
AvgPrice PRODUCTS.List_price%type;
Rows_updated NUMBER;
BEGIN
  SELECT AVG(List_price) INTO AvgPrice From PRODUCTS;
  If AvgPrice <= 1000 THEN
   UPDATE PRODUCTS SET List_price = (List_price * 1.02)
   WHERE List_price < AvgPrice;
   Rows_updated := sql%rowcount;
  ELSE
   UPDATE PRODUCTS SET List_price = (List_price * 1.01)
   WHERE List_price < AvgPrice;
   Rows_updated := sql%rowcount;
  END IF;
  DBMS_OUTPUT.PUT_LINE ('The number of updated rows is: ' || Rows_updated);
  EXCEPTION
  WHEN OTHERS
```

```
    THEN
        DBMS_OUTPUT.PUT_LINE ('Error!');
END;
/
BEGIN
    Update_Price_Under_Avg;
END;
/
ROLLBACK;
```

```
Procedure UPDATE_PRICE_UNDER_AVG compiled

The number of updated rows is: 201


PL/SQL procedure successfully completed.


Rollback complete.
```



5. The company needs a report that shows three category of products based their prices. The company needs to know if the product price is cheap, fair, or expensive. Let's assume that
   - If the list price is less than
     - (avg_price - min_price) / 2

     The product's price is cheap.
   - If the list price is greater than
     - (max_price - avg_price) / 2

     The product' price is expensive.
   - If the list price is between

- (avg_price - min_price) / 2
- and
- (max_price - avg_price) / 2
- the end values included

The product's price is fair.

Write a procedure named *Product_Price_Report* to show the number of products in each price category:

The following is a sample output of the procedure if no error occurs:

```
Cheap: 10
Fair: 50
Expensive: 18
```

The values in the above examples are just random values and may not match the real numbers in your result.

The procedure has no parameter. First, you need to find the average, minimum, and maximum prices (list_price) in your database and store them into varibles avg_price, min_price, and max_price.

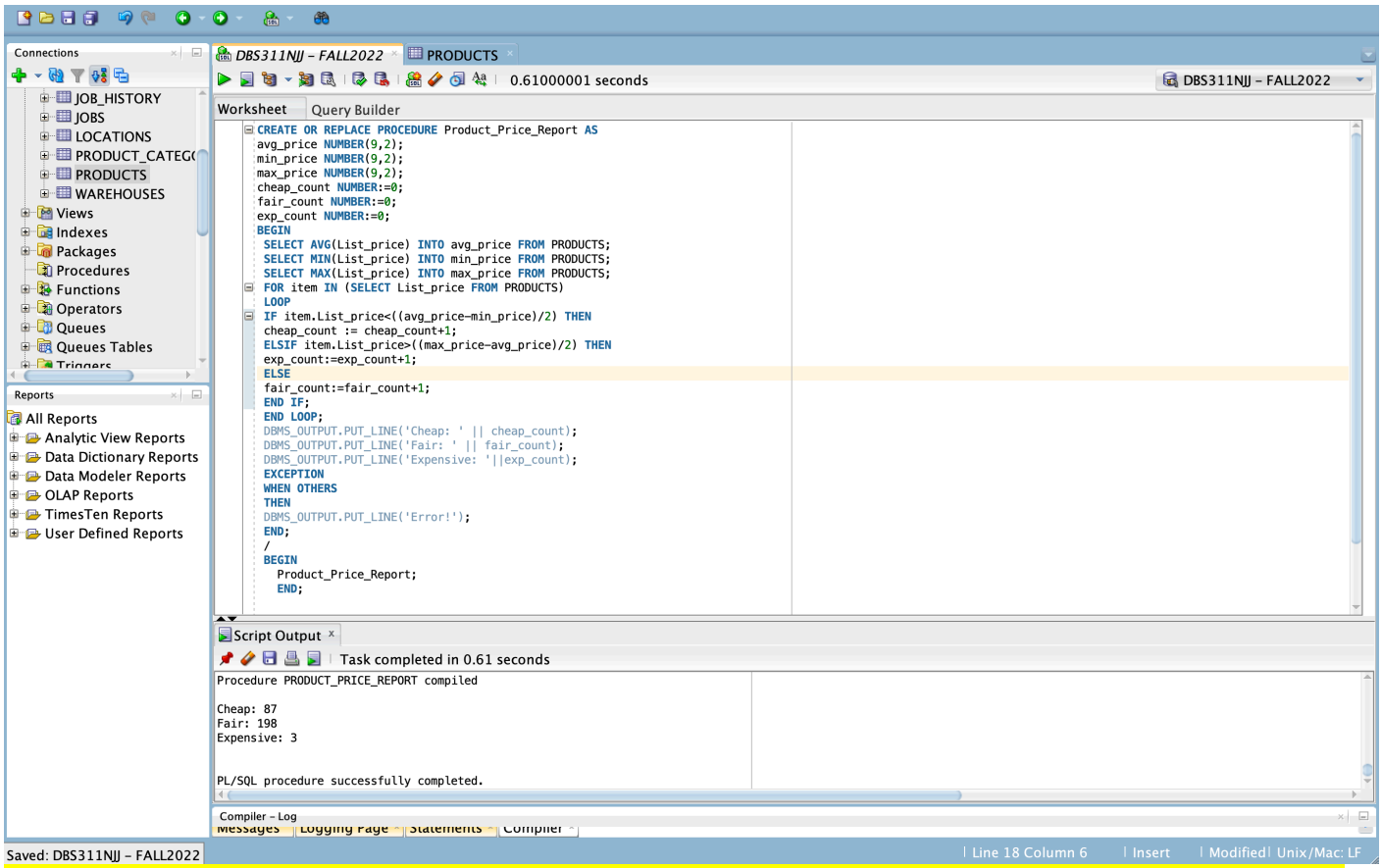
You need more three varaibles to store the number of products in each price category:

cheap_count
fair_count
exp_count

Make sure you choose a proper type for each variable. You may need to define more variables based on your solution.

Show your testing.

```
Procedure PRODUCT_PRICE_REPORT compiled

Cheap: 87
Fair: 198
Expensive: 3


PL/SQL procedure successfully completed.
```



Note: Some of the above output displayed may not match exactly with your produced output. This is because the script file supplied to you was modified after creation of this lab requirements.