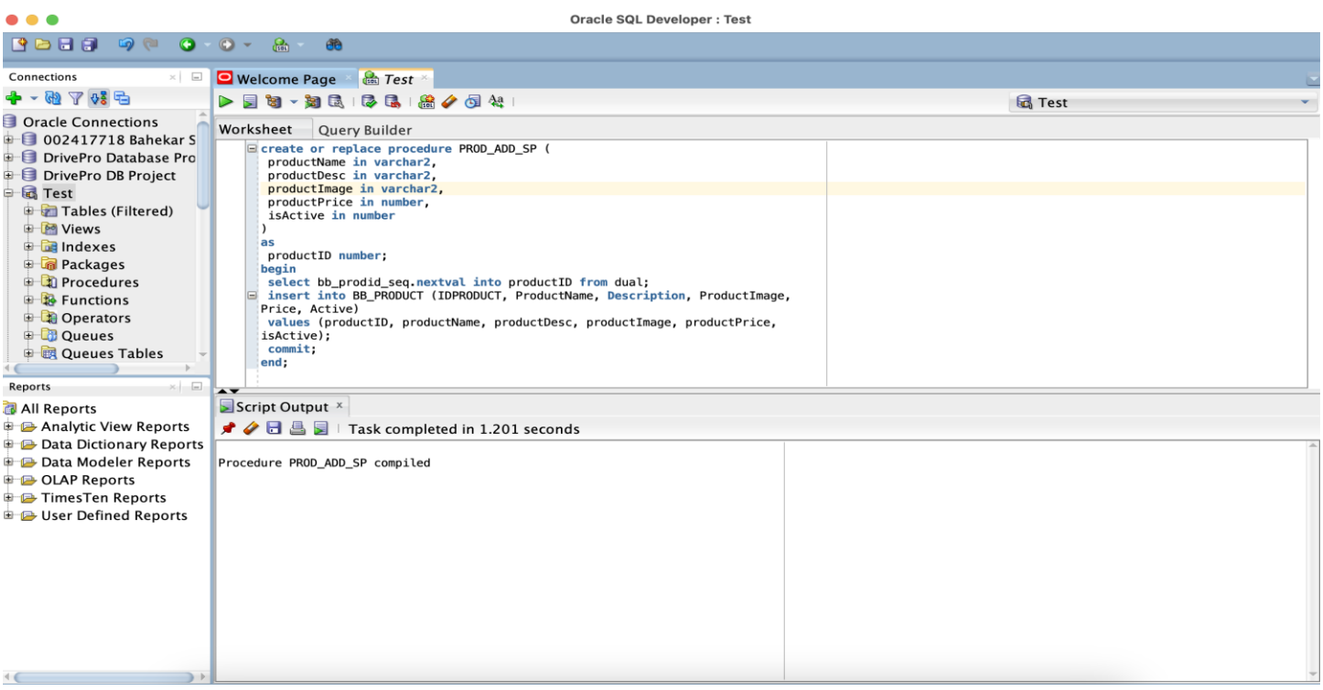
Assignment 3 – Procedures and Functions – Siddharth Bahekar – 002417718

Question 1: Using a Procedure with IN Parameters

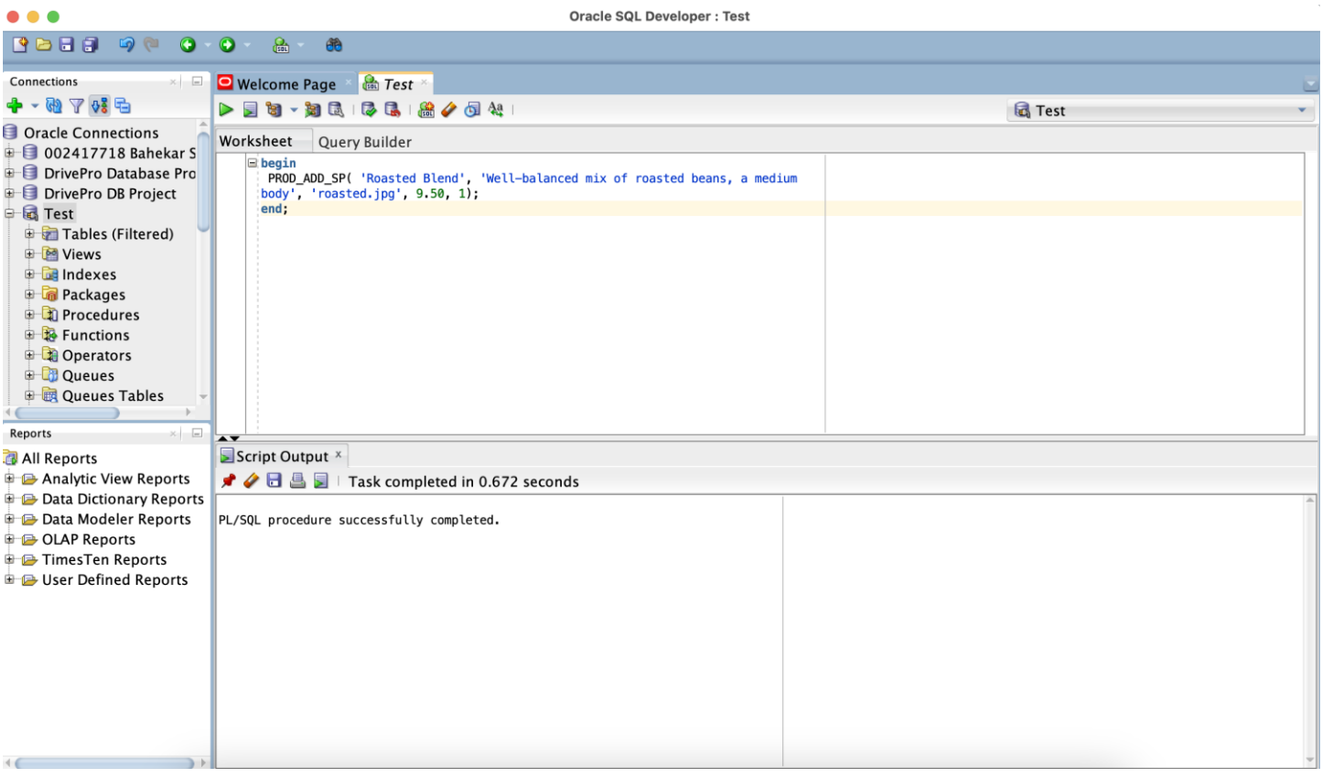
Follow these steps to create a procedure that allows a company employee to add a new product to the database. This procedure needs only `IN` parameters.

1. In SQL Developer, create a procedure named **`PROD_ADD_SP`** that adds a row for a new product in the BB_PRODUCT table. Keep in mind that the user provides values for the product name, description, image filename, price, and active status. Address the input values or parameters in the same order as in the preceding sentence.
2. Call the procedure with these parameter values: **`('Roasted Blend', 'Well-balanced mix of roasted beans, a medium body', 'roasted.jpg',9.50,1)`**.
3. Check whether the update was successful by querying the BB_PRODUCT table.

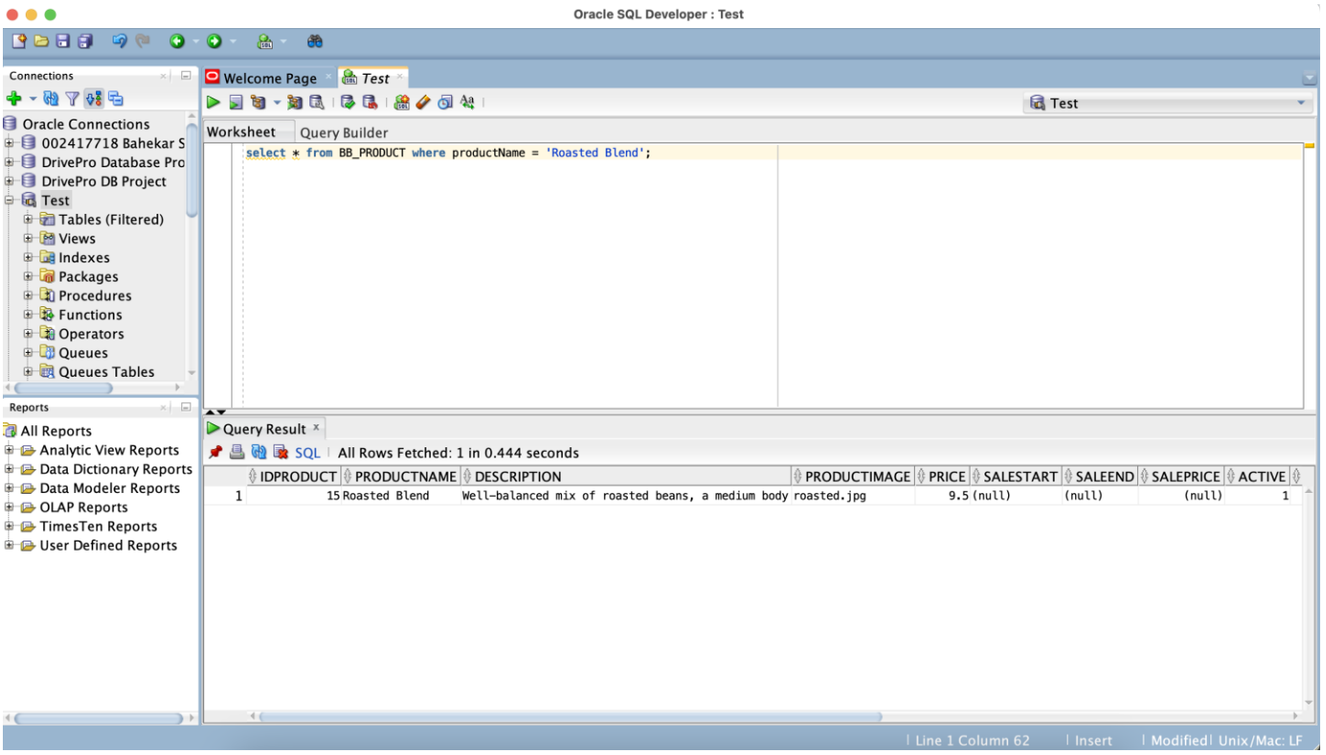
```
create or replace procedure PROD_ADD_SP (
 productName in varchar2,
 productDesc in varchar2,
 productImage in varchar2,
 productPrice in number,
 isActive in number
)
as
 productID number;
begin
 select bb_prodid_seq.nextval into productID from dual;
 insert into BB_PRODUCT (IDPRODUCT, ProductName, Description, ProductImage,
Price, Active)
 values (productID, productName, productDesc, productImage, productPrice,
isActive);
 commit;
end;
```



```
begin
 PROD_ADD_SP( 'Roasted Blend', 'Well-balanced mix of roasted beans, a medium
body', 'roasted.jpg', 9.50, 1);
end;
```




```
select * from BB_PRODUCT where productName = 'Roasted Blend';
```

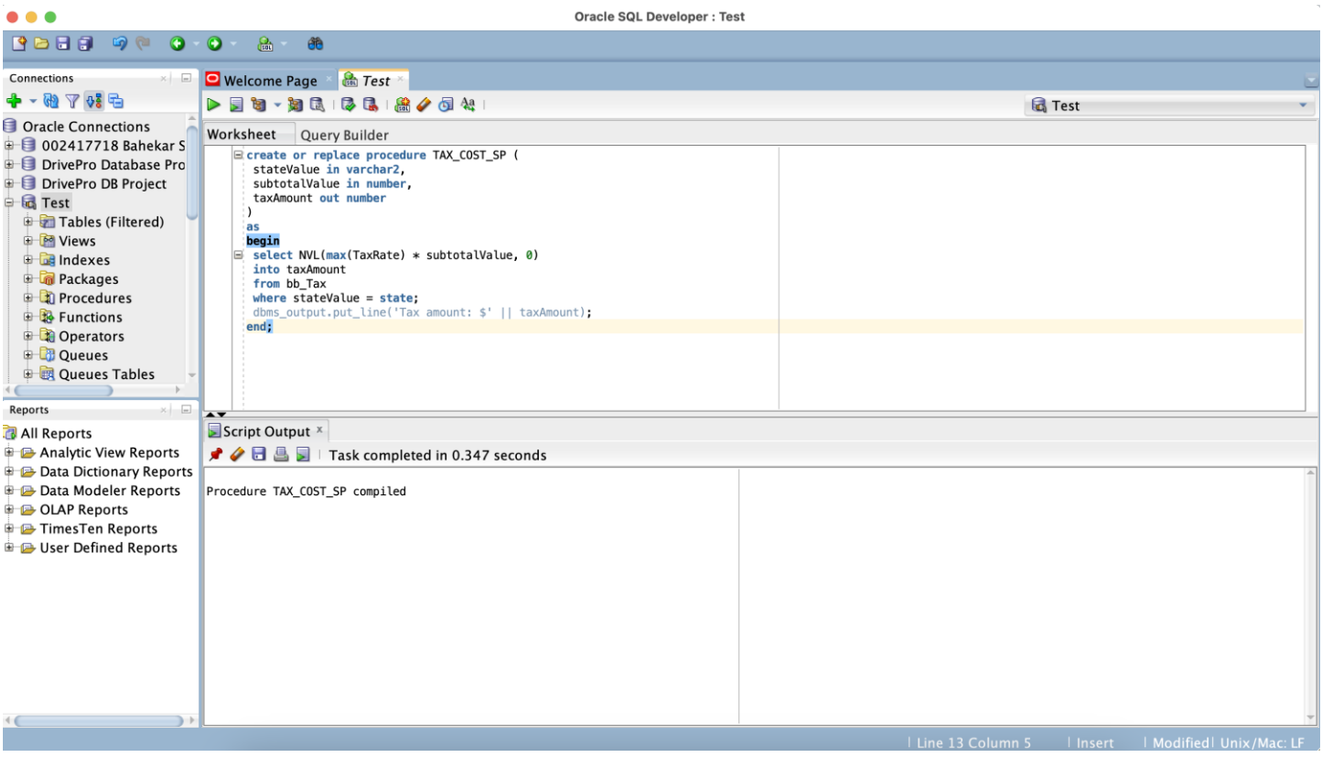


Question 2: Calculating the Tax on an Order

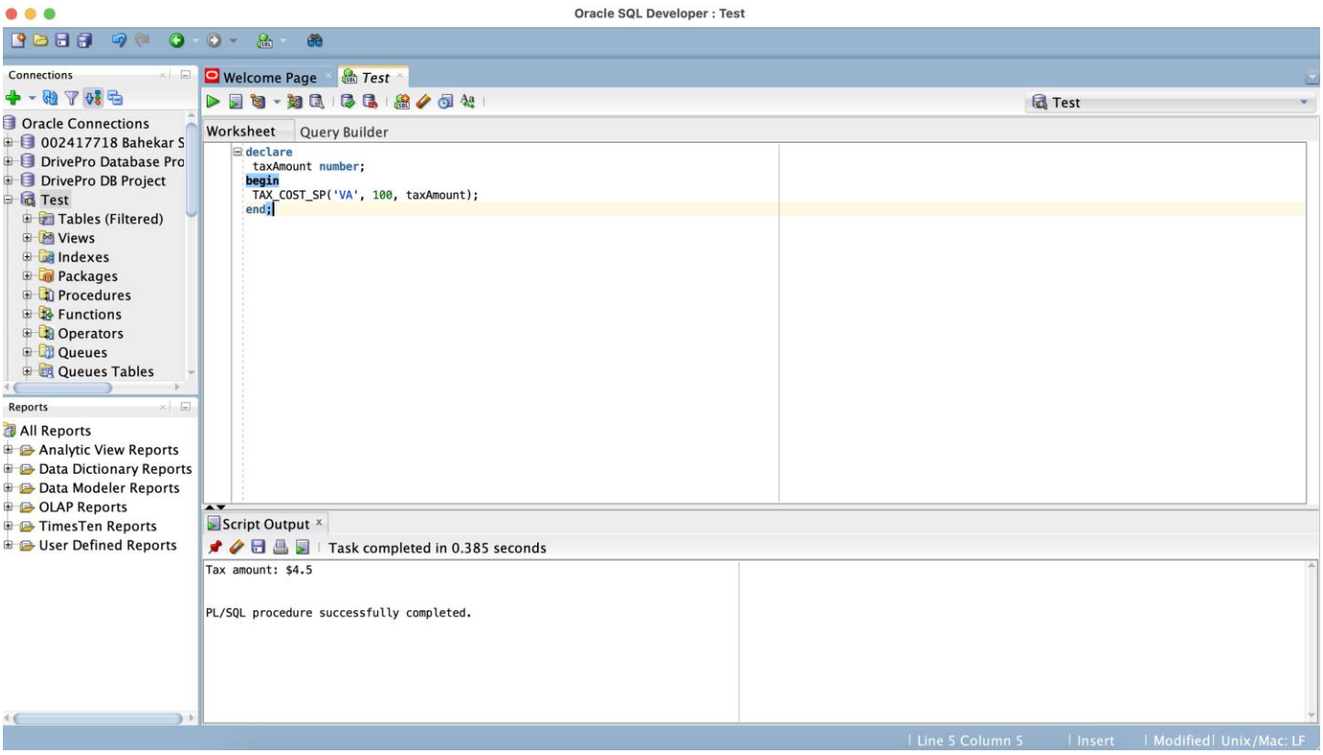
Follow these steps to create a procedure for calculating the tax on an order. The BB_TAX table contains states that require submitting taxes for Internet sales. If the state isn't listed in the table, no tax should be assessed on the order. The shopper's state and basket subtotal are the inputs to the procedure, and the tax amount should be returned.

1. In SQL Developer, create a procedure named **TAX_COST_SP**. Remember that the state and subtotal values are inputs to the procedure, which should return the tax amount. Review the BB_TAX table, which contains the tax rate for each applicable state.
2. Call the procedure with the values **VA** for the state and **$100** for the subtotal. Display the tax amount the procedure returns. (It should be $4.50.)

```
create or replace procedure TAX_COST_SP (
stateValue in varchar2,
subtotalValue in number,
taxAmount out number
)
as
begin
select NVL(max(TaxRate) * subtotalValue, 0)
into taxAmount
from bb_Tax
where stateValue = state;
dbms_output.put_line('Tax amount: $' || taxAmount);
end;
```



```
declare
 taxAmount number;
begin
 TAX_COST_SP('VA', 100, taxAmount);
end;
```



Question 3: Updating Columns in a Table

After a shopper completes an order, a procedure is called to update the following columns in the BASKET table: ORDERPLACED, SUBTOTAL, SHIPPING, TAX, and TOTAL. The value 1 entered in the ORDERPLACED column indicates that the shopper has completed an order. Inputs to the procedure are the basket ID and amounts for the subtotal, shipping, tax, and total.

1. In SQL Developer, create a procedure named **BASKET_CONFIRM_SP** that accepts the input values specified in the preceding description. Keep in mind that you're modifying an existing row of the BB_BASKET table in this procedure.

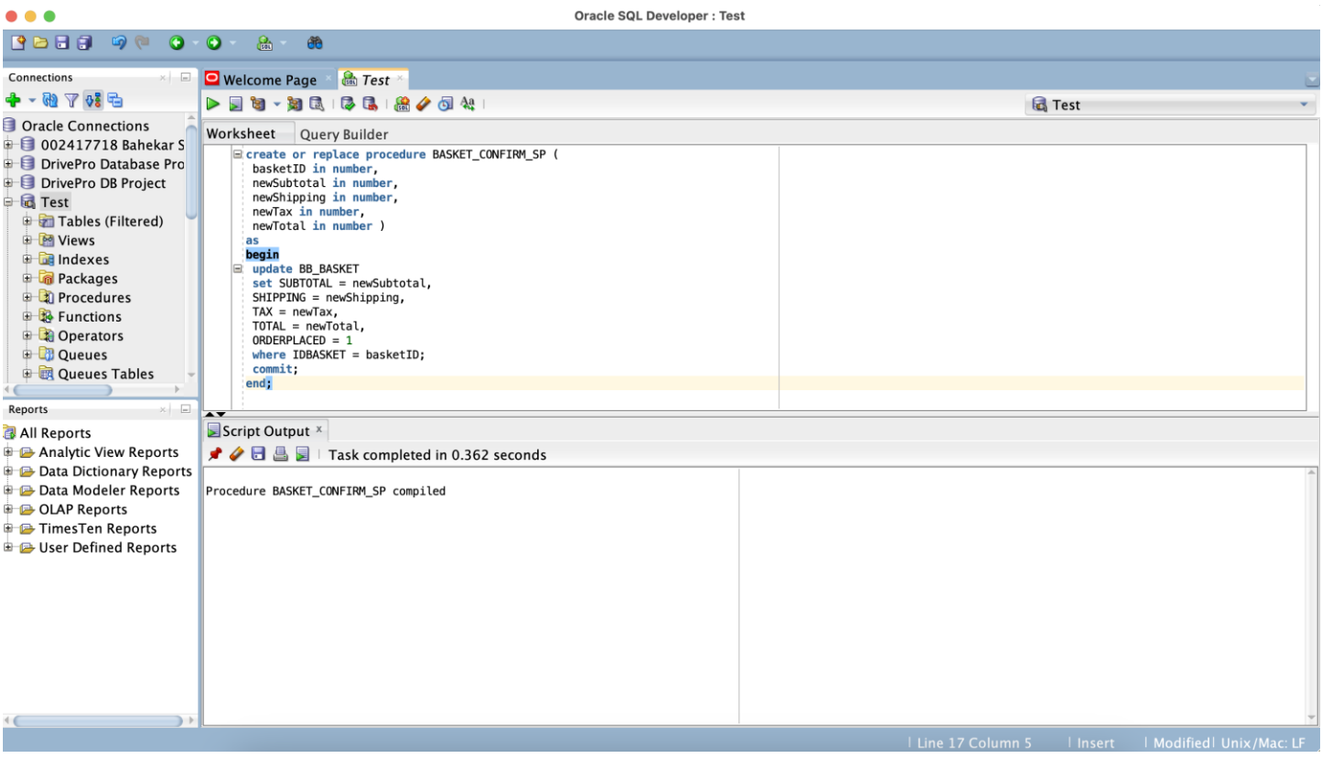
2. Enter the following statements to create a new basket containing two items:

```
INSERT INTO BB_BASKET (IDBASKET, QUANTITY, IDSHOPPER,
                       ORDERPLACED, SUBTOTAL, TOTAL,
                       SHIPPING, TAX, DTCREATED, PROMO)
    VALUES (17, 2, 22, 0, 0, 0, 0, 0, '28-FEB-12', 0);
INSERT INTO BB_BASKETITEM (IDBASKETITEM, IDPRODUCT, PRICE,
                           QUANTITY, IDBASKET, OPTION1, OPTION2)
    VALUES (44, 7, 10.8, 3, 17, 2, 3);
INSERT INTO BB_BASKETITEM (IDBASKETITEM, IDPRODUCT, PRICE,
                           QUANTITY, IDBASKET, OPTION1, OPTION2)
    VALUES (45, 8, 10.8, 3, 17, 2, 3);
```

3. Type and run `COMMIT;` to save the data from these statements.
4. Call the procedure with the following parameter values: `17, 64.80, 8.00, 1.94, 74.74`. As mentioned, these values represent the basket ID and the amounts for the subtotal, shipping, tax, and total.
5. Query the BB_BASKET table to confirm that the procedure was successful:

```
SELECT subtotal, shipping, tax, total, orderplaced
  FROM bb_basket
  WHERE idbasket = 17;
```

```
create or replace procedure BASKET_CONFIRM_SP (
 basketID in number,
 newSubtotal in number,
 newShipping in number,
 newTax in number,
 newTotal in number )
as
begin
 update BB_BASKET
 set SUBTOTAL = newSubtotal,
 SHIPPING = newShipping,
 TAX = newTax,
 TOTAL = newTotal,
 ORDERPLACED = 1
 where IDBASKET = basketID;
 commit;
end;
```



INSERT INTO BB_BASKET (IDBASKET, QUANTITY, IDSHOPPER,

ORDERPLACED, SUBTOTAL, TOTAL, SHIPPING, TAX, DTCREATED, PROMO)

VALUES (17, 2, 22, 0, 0, 0, 0, 0, '28-FEB-12', 0);

INSERT INTO BB_BASKETITEM (IDBASKETITEM, IDPRODUCT, PRICE,

QUANTITY, IDBASKET, OPTION1, OPTION2)

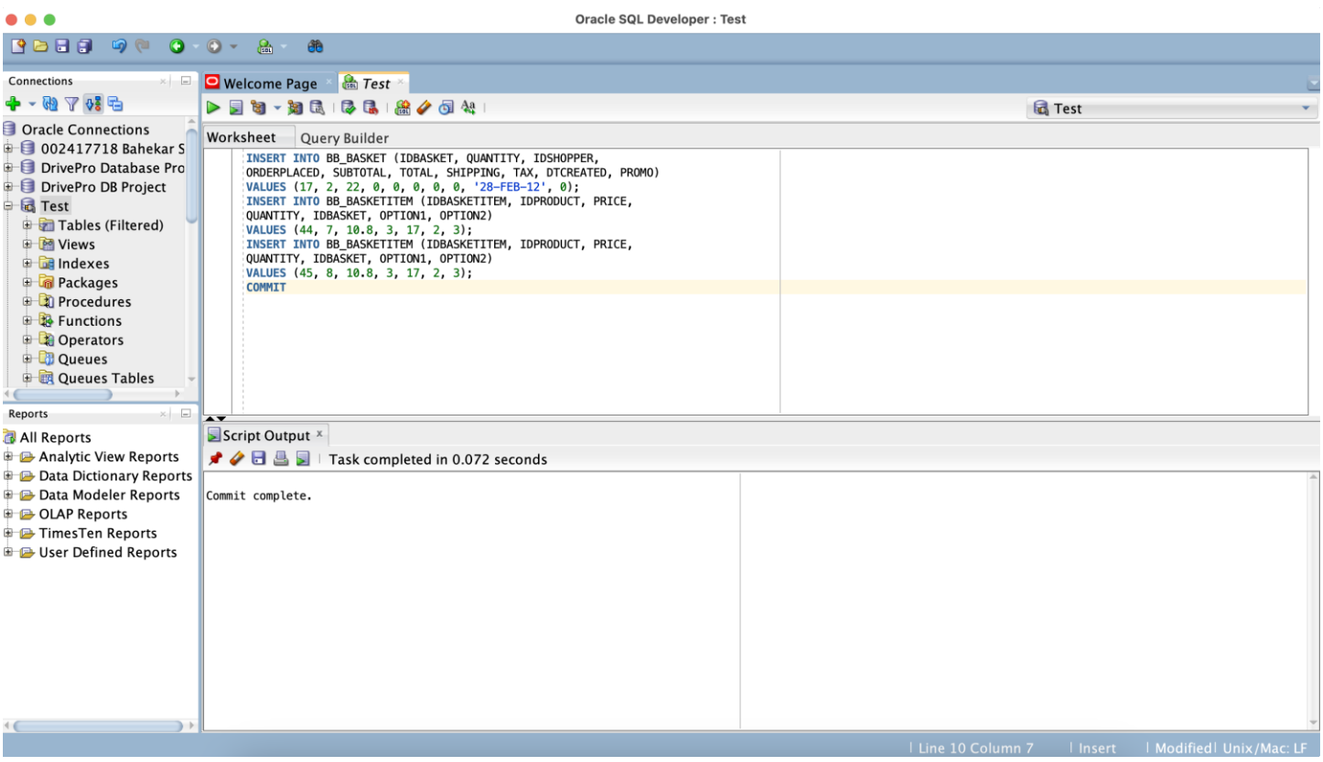
VALUES (44, 7, 10.8, 3, 17, 2, 3);

INSERT INTO BB_BASKETITEM (IDBASKETITEM, IDPRODUCT, PRICE,

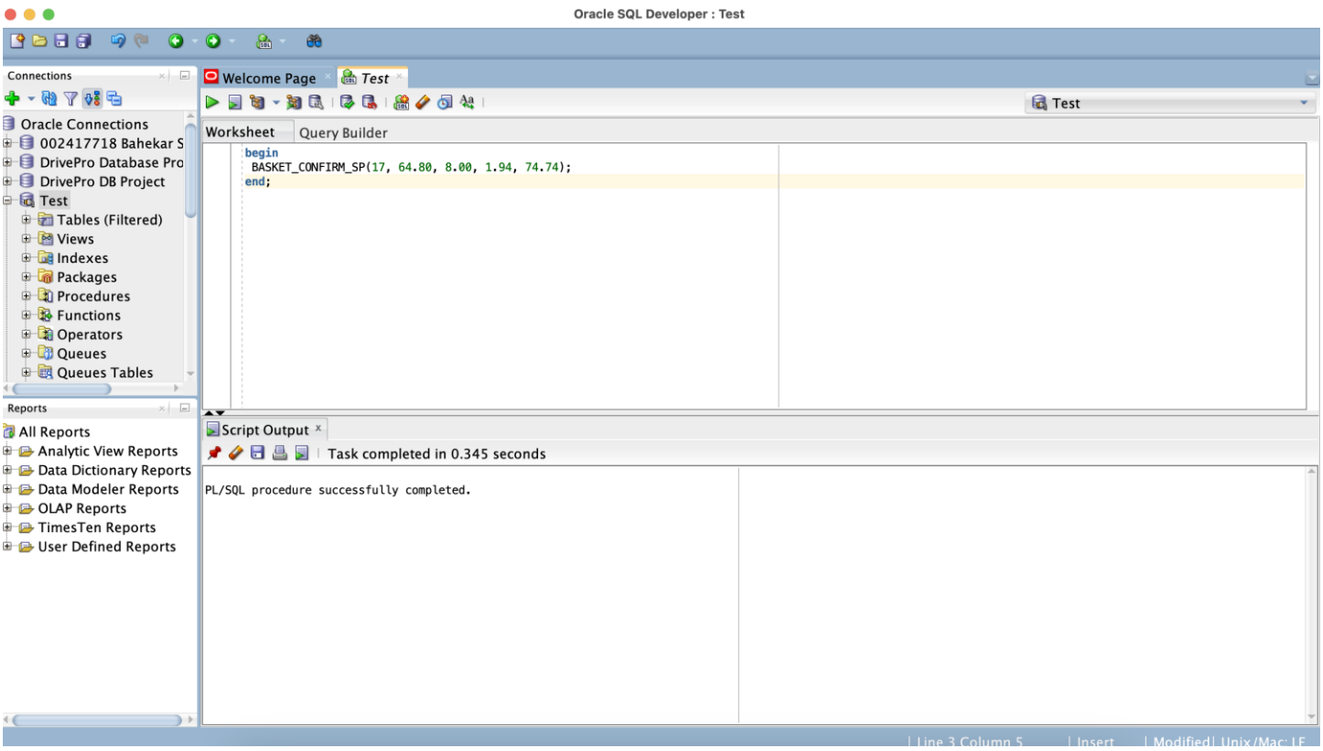
QUANTITY, IDBASKET, OPTION1, OPTION2)

VALUES (45, 8, 10.8, 3, 17, 2, 3);

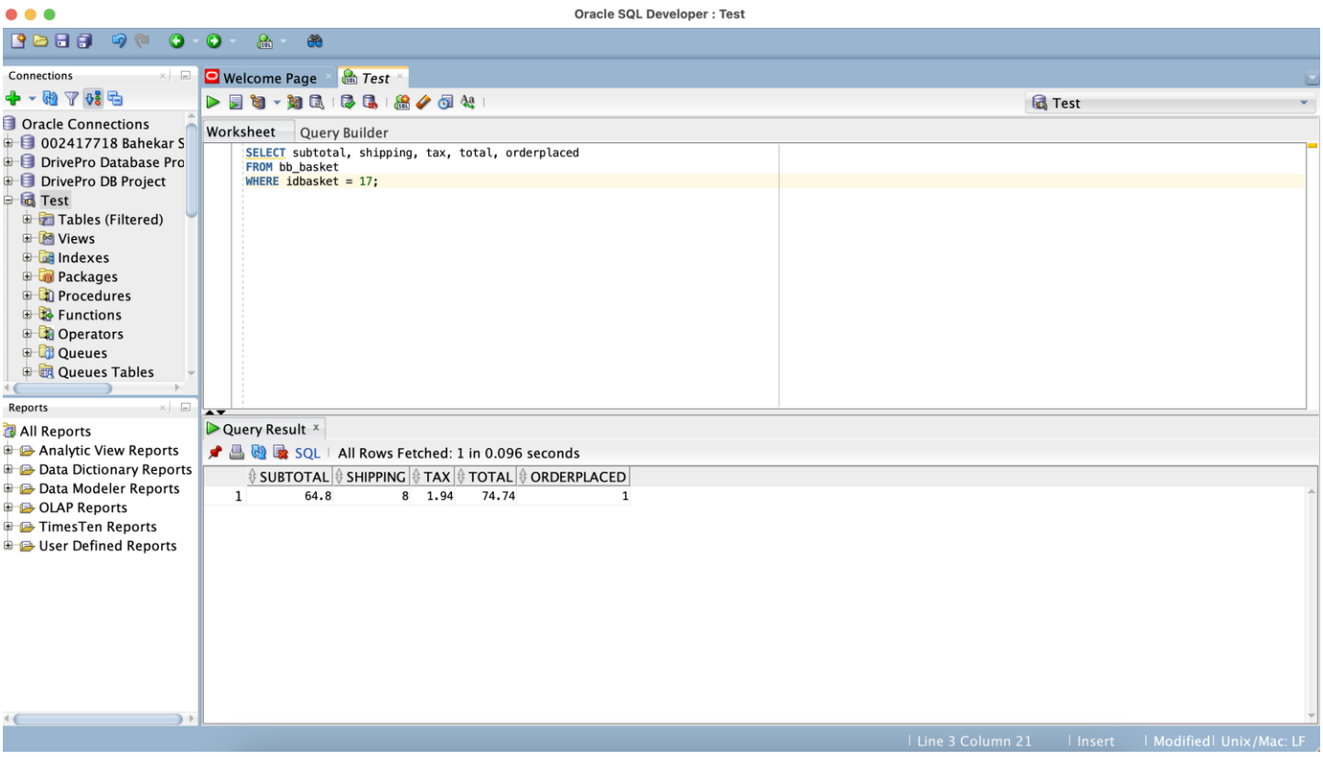
COMMIT



```
begin
 BASKET_CONFIRM_SP(17, 64.80, 8.00, 1.94, 74.74);
end;
```




```
SELECT subtotal, shipping, tax, total, orderplaced
FROM bb_basket
WHERE idbasket = 17;
```



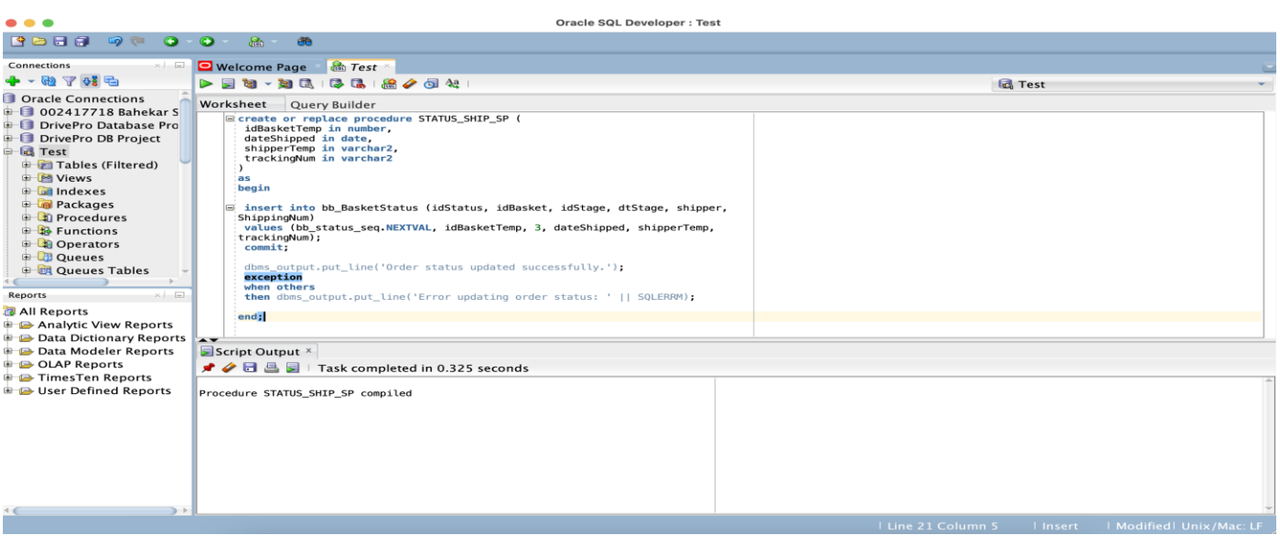
Question 4: Updating Order status:

Create a procedure named **STATUS_SHIP_SP** that allows an employee in the Shipping Department to update an order status to add shipping information. The BB_BASKETSTATUS table lists events for each order so that a shopper can see the current status, date, and comments as each stage of the order process is finished. The IDSTAGE column of the BB_BASKETSTATUS table identifies each stage; the value 3 in this column indicates that an order has been shipped.

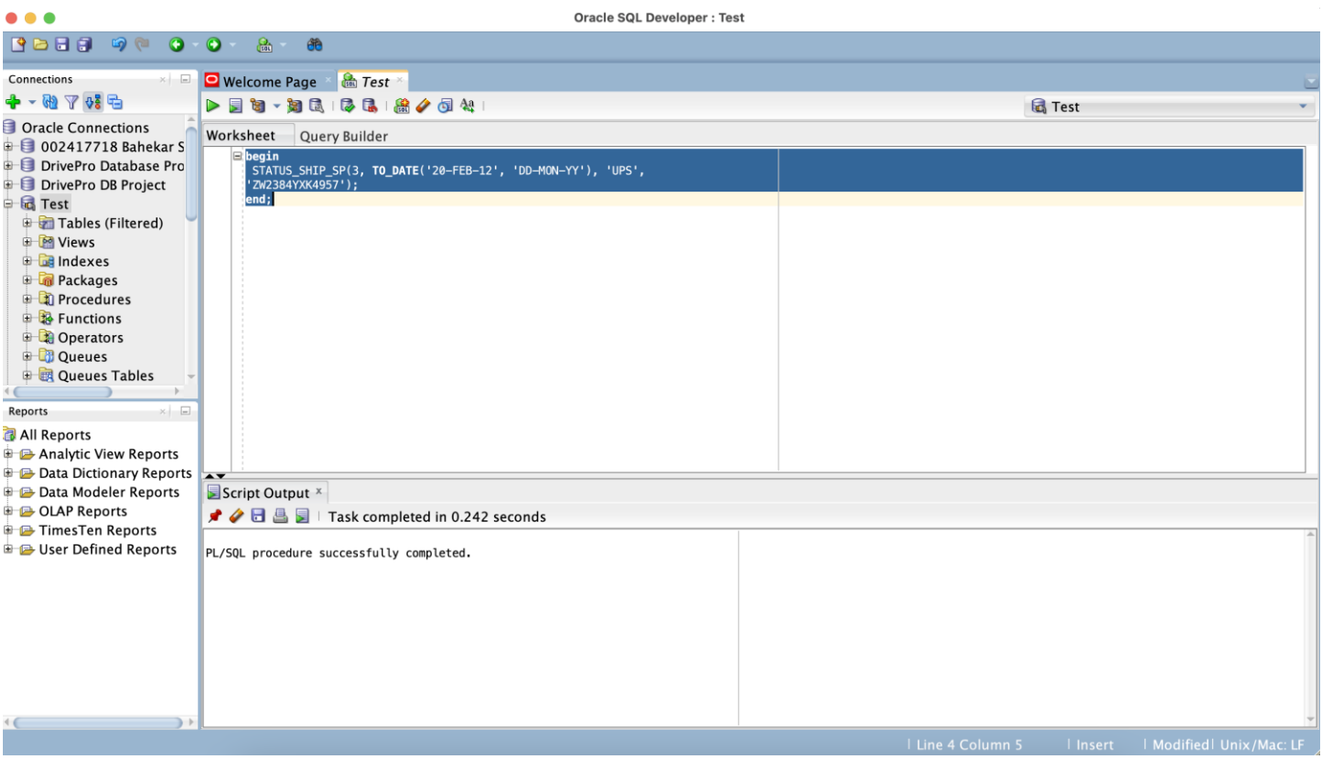
The procedure should allow adding a row with an IDSTAGE of 3, date shipped, tracking number, and shipper. The BB_STATUS_SEQ sequence is used to provide a value for the primary key column. Test the procedure with the following information:

```
Basket # = 3
Date shipped = 20-FEB-12
Shipper = UPS
Tracking # = ZW2384YXK4957
```

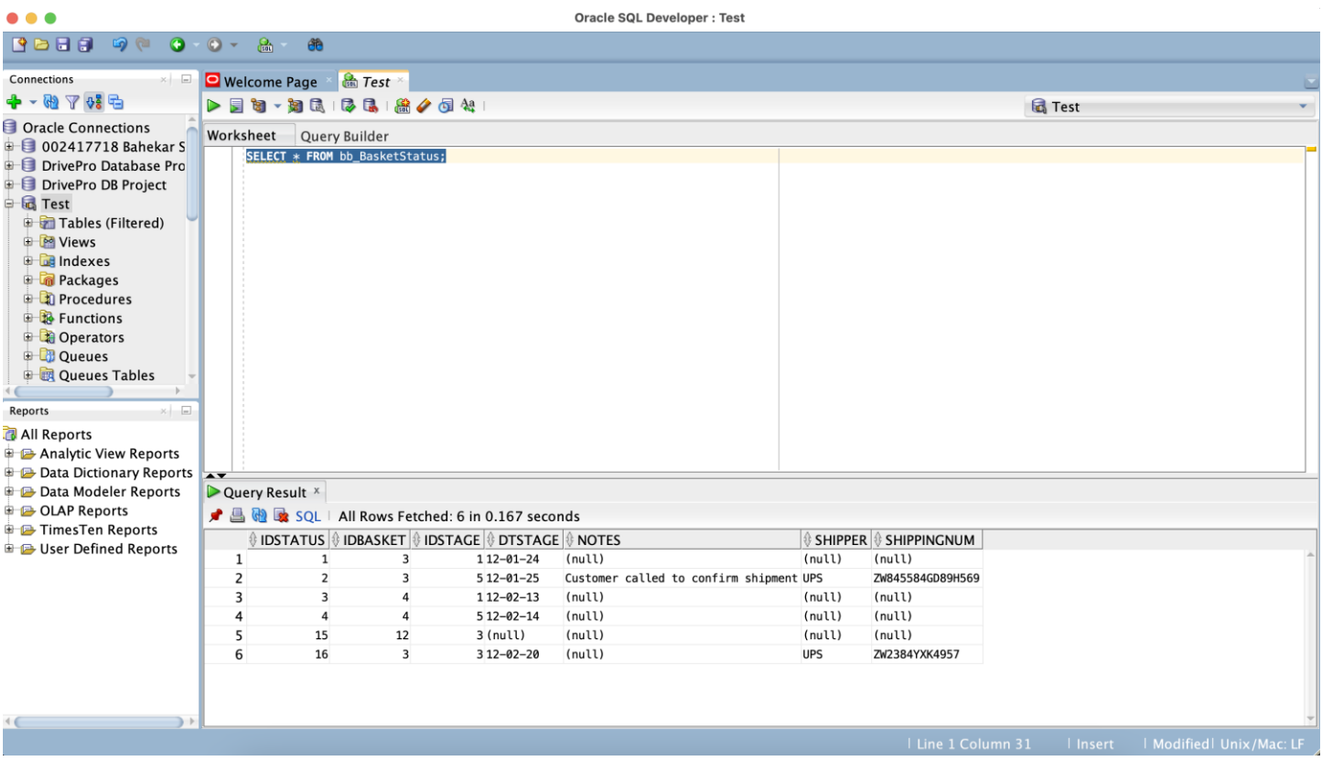
```
create or replace procedure STATUS_SHIP_SP (
 idBasketTemp in number,
 dateShipped in date,
 shipperTemp in varchar2,
 trackingNum in varchar2
 )
as
begin


 insert into bb_BasketStatus (idStatus, idBasket, idStage, dtStage, shipper,
 ShippingNum)
 values (bb_status_seq.NEXTVAL, idBasketTemp, 3, dateShipped, shipperTemp,
 trackingNum);
 commit;


 dbms_output.put_line('Order status updated successfully.');
 exception
 when others
  then dbms_output.put_line('Error updating order status: ' || SQLERRM);
end;
```



```
begin
 STATUS_SHIP_SP(3, TO_DATE('20-FEB-12', 'DD-MON-YY'), 'UPS',
 'ZW2384YXK4957');
end;
```




```
SELECT * FROM bb_BasketStatus;
```



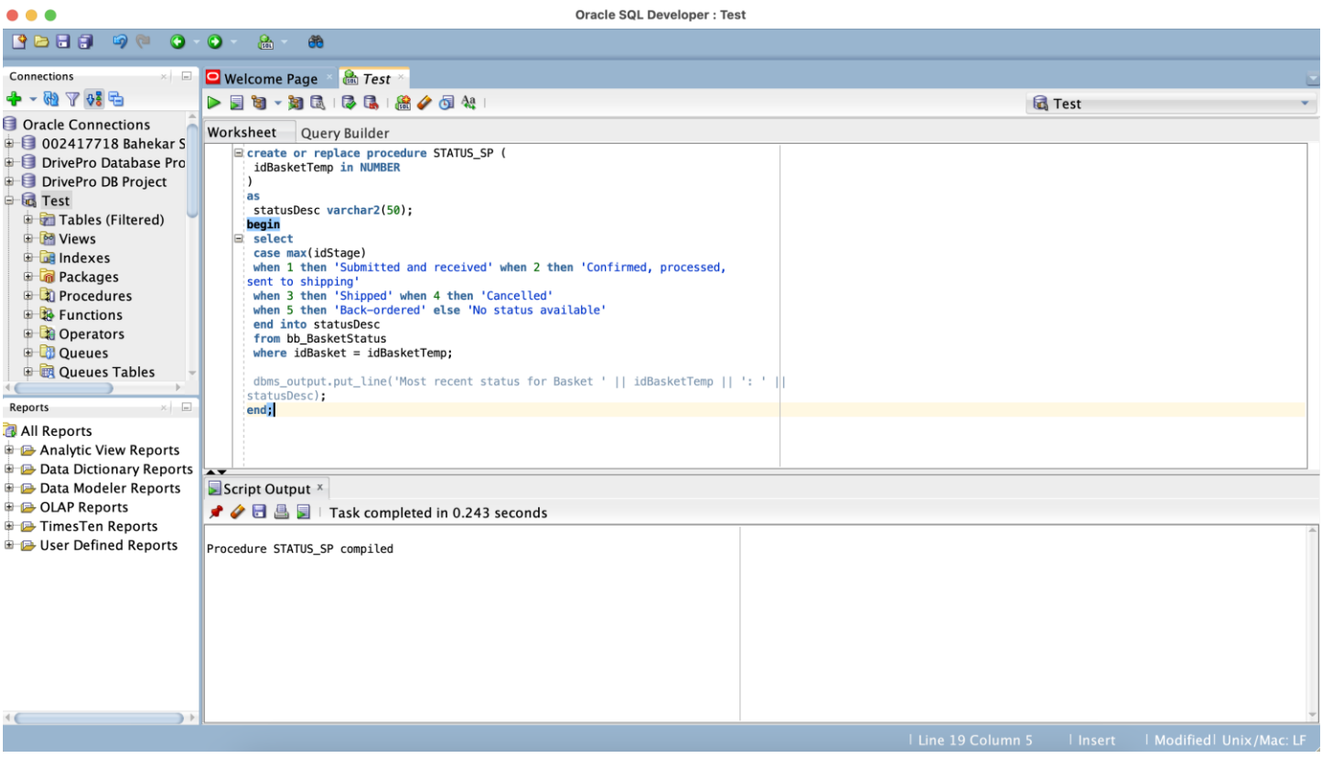
Question 5: Returning Order status Information:

Create a procedure that returns the most recent order status information for a specified basket. This procedure should determine the most recent ordering-stage entry in the BB_BASKETSTATUS table and return the data. Use an `IF` or `CASE` clause to return a stage description instead of an IDSTAGE number, which means little to shoppers. The IDSTAGE column of the BB_BASKETSTATUS table identifies each stage as follows:

- 1—Submitted and received
- 2—Confirmed, processed, sent to shipping
- 3—Shipped
- 4—Cancelled
- 5—Back-ordered

The procedure should accept a basket ID number and return the most recent status description and date the status was recorded. If no status is available for the specified basket ID, return a message stating that no status is available. Name the procedure STATUS_SP. Test the procedure twice with the basket ID 4 and then 6.

```
create or replace procedure STATUS_SP (
 idBasketTemp in NUMBER
)
as
 statusDesc varchar2(50);
begin
 select
 case max(idStage)
 when 1 then 'Submitted and received' when 2 then 'Confirmed, processed,
 sent to shipping'
 when 3 then 'Shipped'
 when 4 then 'Cancelled'
 when 5 then 'Back-ordered'
 else 'No status available'
 end into statusDesc
 from bb_BasketStatus
 where idBasket = idBasketTemp;

 dbms_output.put_line('Most recent status for Basket ' || idBasketTemp || ': ' ||
statusDesc);
end;
```

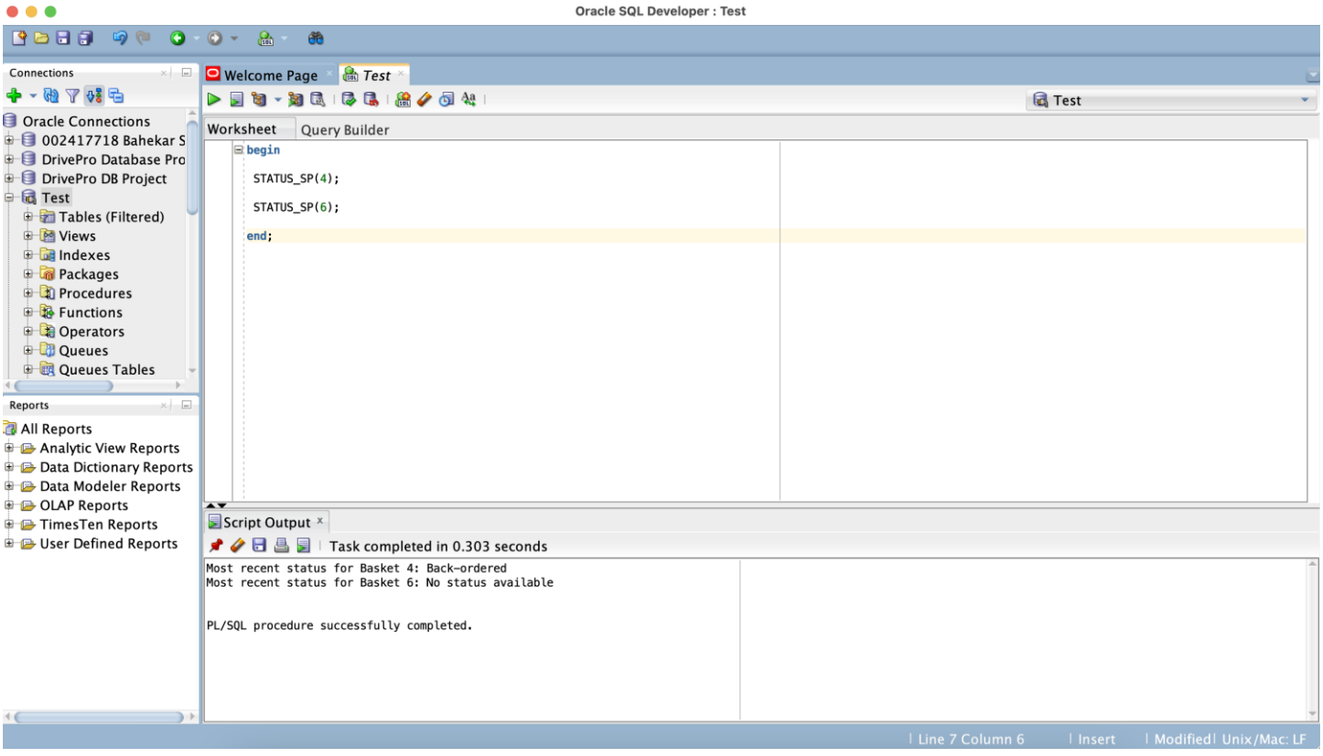


begin

STATUS_SP(4);

STATUS_SP(6);

end;



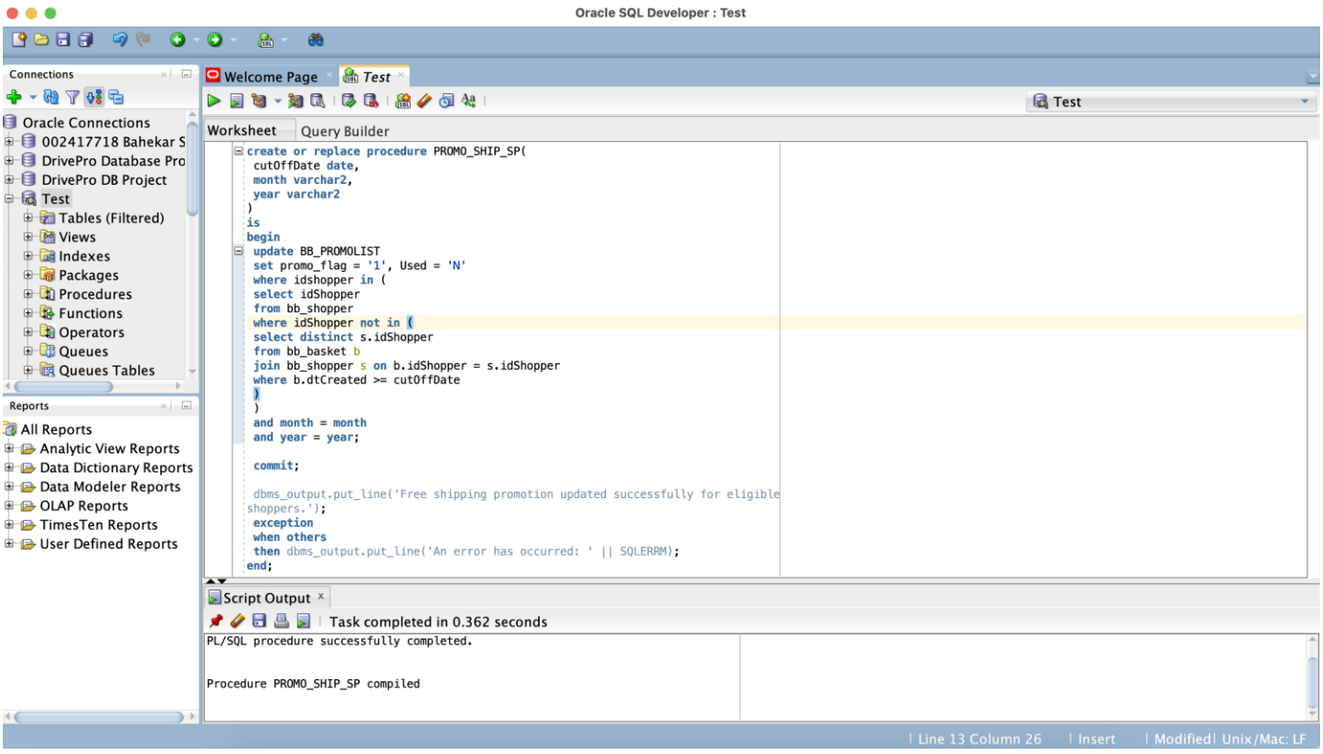
Question 6: Identifying Customer:

Brewbean's wants to offer an incentive of free shipping to customers who haven't returned to the site since a specified date. Create a procedure named **PROMO_SHIP_SP** that determines who these customers are and then updates the BB_PROMOLIST table accordingly. The procedure uses the following information:

- *Date cutoff*—Any customers who haven't shopped on the site since this date should be included as incentive participants. Use the basket creation date to reflect shopper activity dates.
- *Month*—A three-character month (such as APR) should be added to the promotion table to indicate which month free shipping is effective.
- *Year*—A four-digit year indicates the year the promotion is effective.
- `promo_flag`—1 represents free shipping.

The BB_PROMOLIST table also has a USED column, which contains the default value `N` and is updated to `Y` when the shopper uses the promotion. Test the procedure with the cutoff date 15-FEB-12. Assign free shipping for the month APR and the year 2012.

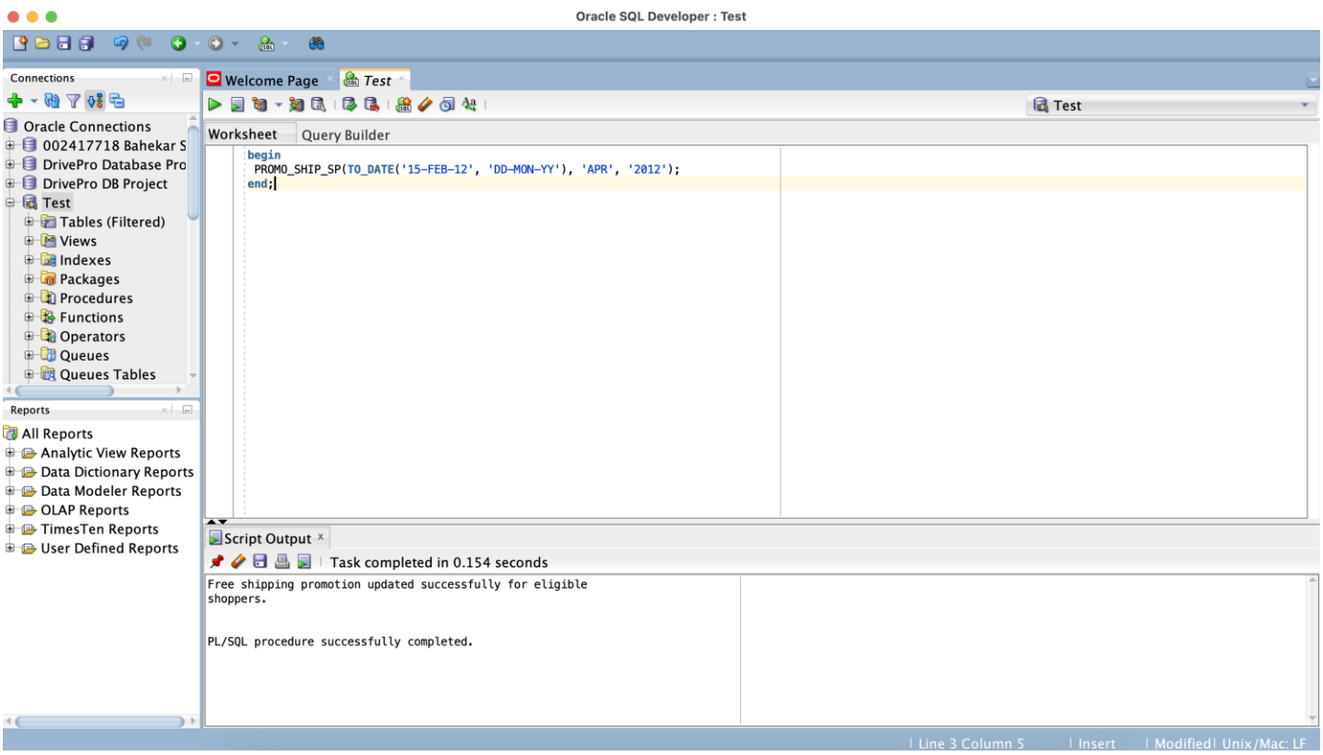
```
create or replace procedure PROMO_SHIP_SP(
cutOffDate date,
month varchar2,
year varchar2
)
is
begin
update BB_PROMOLIST
set promo_flag = '1', Used = 'N'
where idshopper in (
select idShopper
from bb_shopper
where idShopper not in (
select distinct s.idShopper
from bb_basket b
join bb_shopper s on b.idShopper = s.idShopper
where b.dtCreated >= cutOffDate
)
)
and month = month
and year = year;
commit;
dbms_output.put_line('Free shipping promotion updated successfully for eligible
shoppers.');
exception
when others
then dbms_output.put_line('An error has occurred: ' || SQLERRM);
end;
```



begin

PROMO_SHIP_SP(TO_DATE('15-FEB-12', 'DD-MON-YY'), 'APR', '2012');

end;

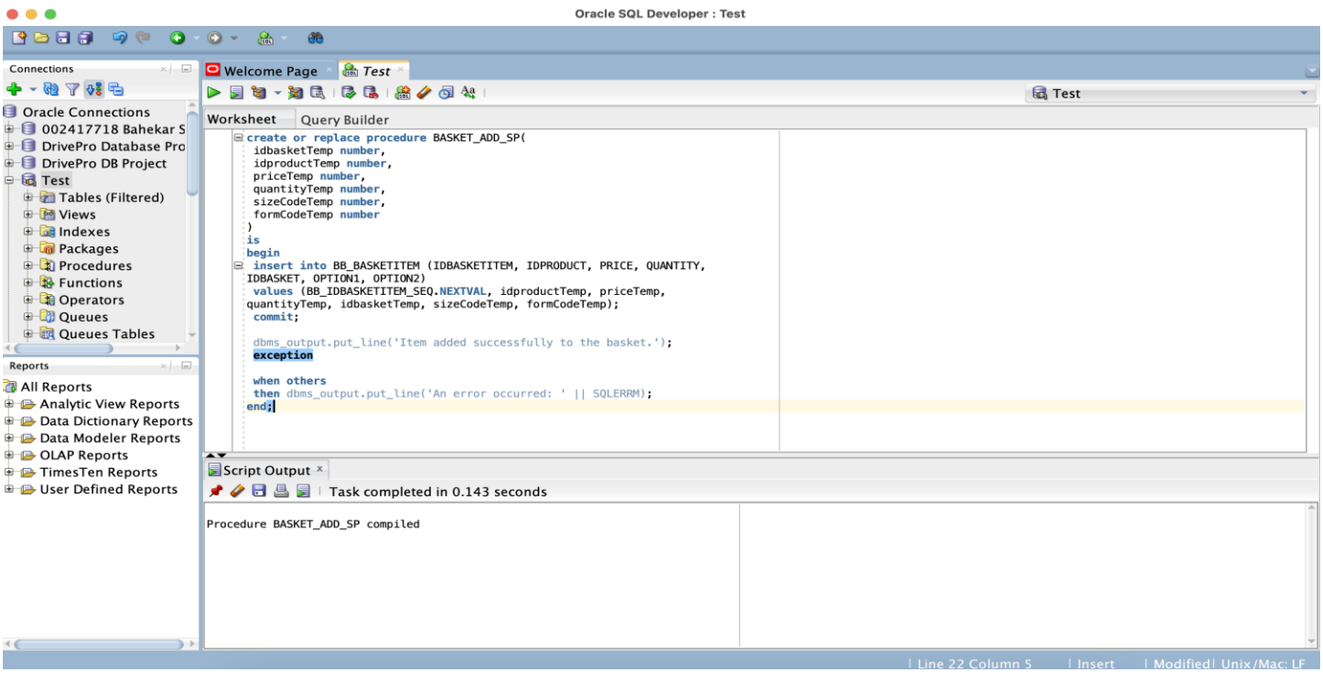


Question 7: Adding Items to a Basket:

As a shopper selects products on the Brewbean's site, a procedure is needed to add a newly selected item to the current shopper's basket. Create a procedure named **BASKET_ADD_SP** that accepts a product ID, basket ID, price, quantity, size code option (1 or 2), and form code option (3 or 4) and uses this information to add a new item to the BB_BASKETITEM table. The table's PRIMARY KEY column is generated by `BB_IDBASKETITEM_SEQ`. Run the procedure with the following values:

- Basket ID—14
- Product ID—8
- Price—10.80
- Quantity—1
- Size code—2
- Form code—4

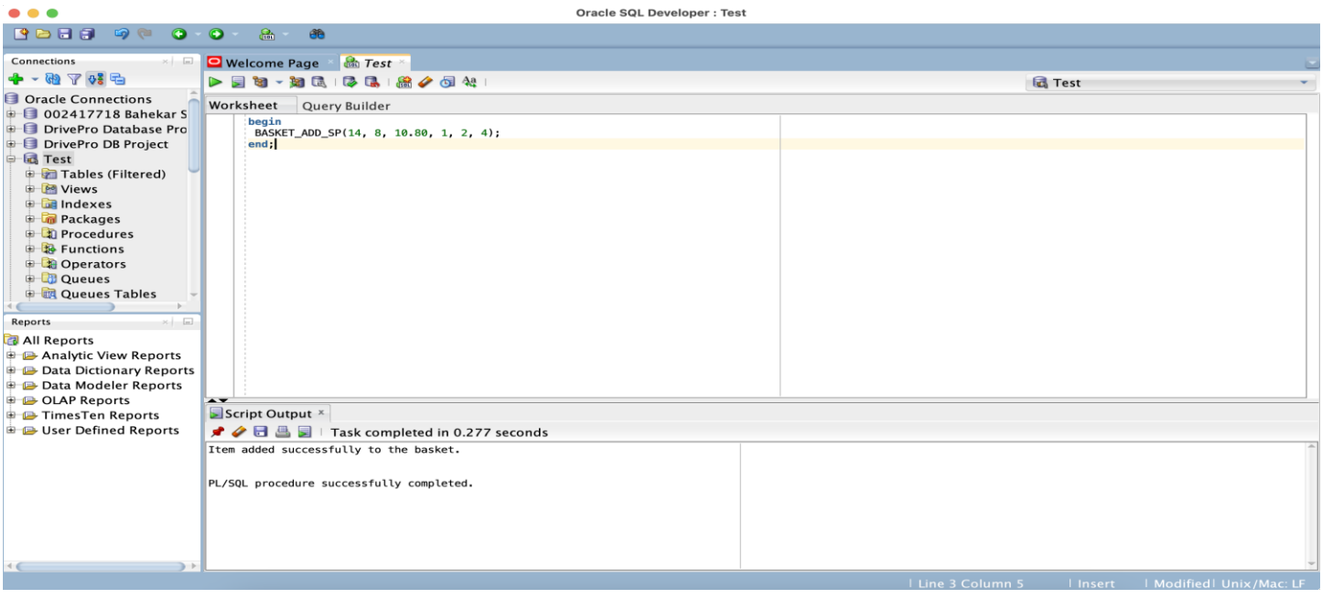
```
create or replace procedure BASKET_ADD_SP(
idbasketTemp number,
idproductTemp number,
priceTemp number,
quantityTemp number,
sizeCodeTemp number,
formCodeTemp number
)
is
begin
insert into BB_BASKETITEM (IDBASKETITEM, IDPRODUCT, PRICE, QUANTITY,
IDBASKET, OPTION1, OPTION2)
values (BB_IDBASKETITEM_SEQ.NEXTVAL, idproductTemp, priceTemp,
quantityTemp, idbasketTemp, sizeCodeTemp, formCodeTemp);
commit;
dbms_output.put_line('Item added successfully to the basket.');
exception
when others
then dbms_output.put_line('An error occurred: ' || SQLERRM);
end;
```



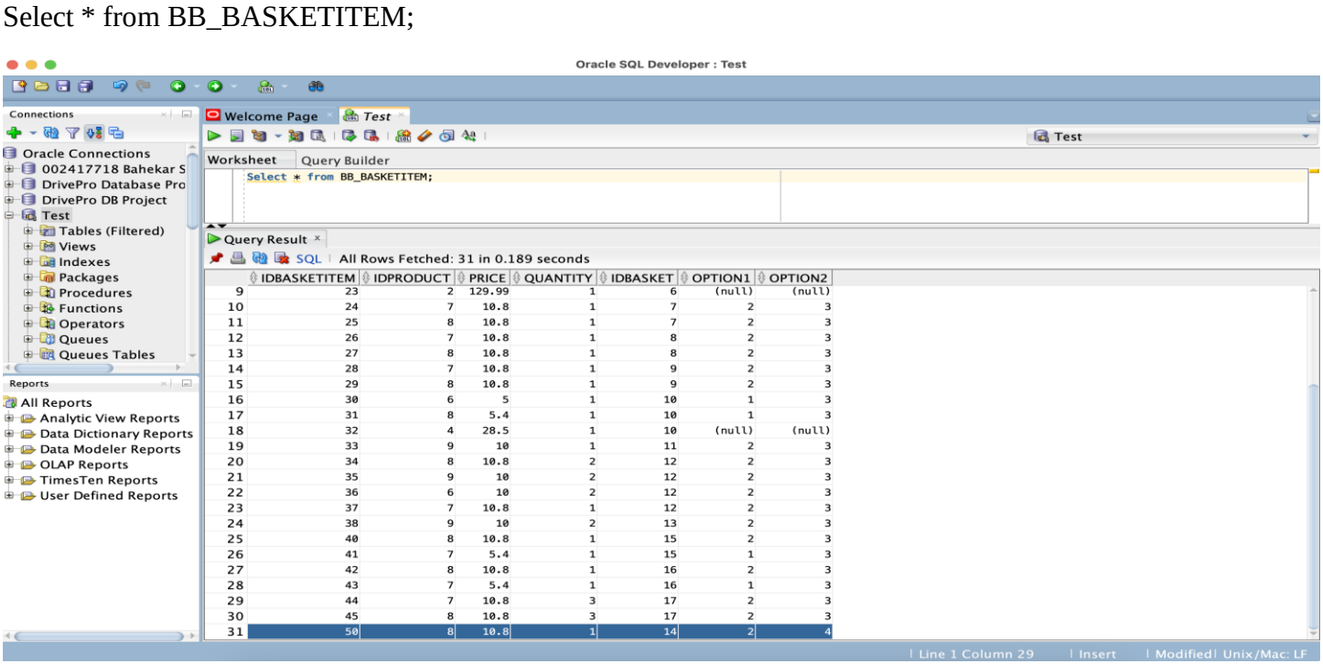
begin

BASKET_ADD_SP(14, 8, 10.80, 1, 2, 4);

end;



Select * from BB_BASKETITEM;

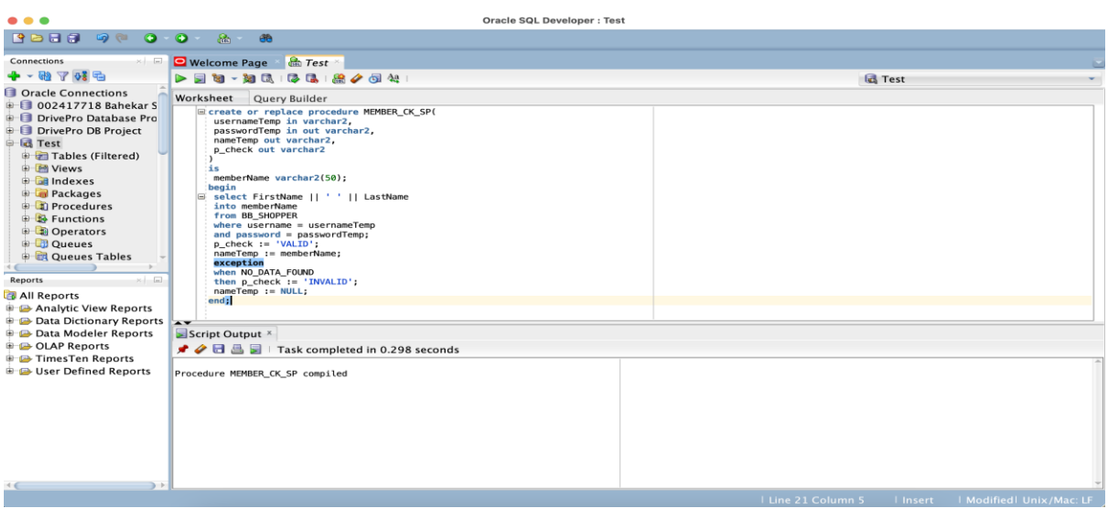


Question 8: Create a Login Procedure

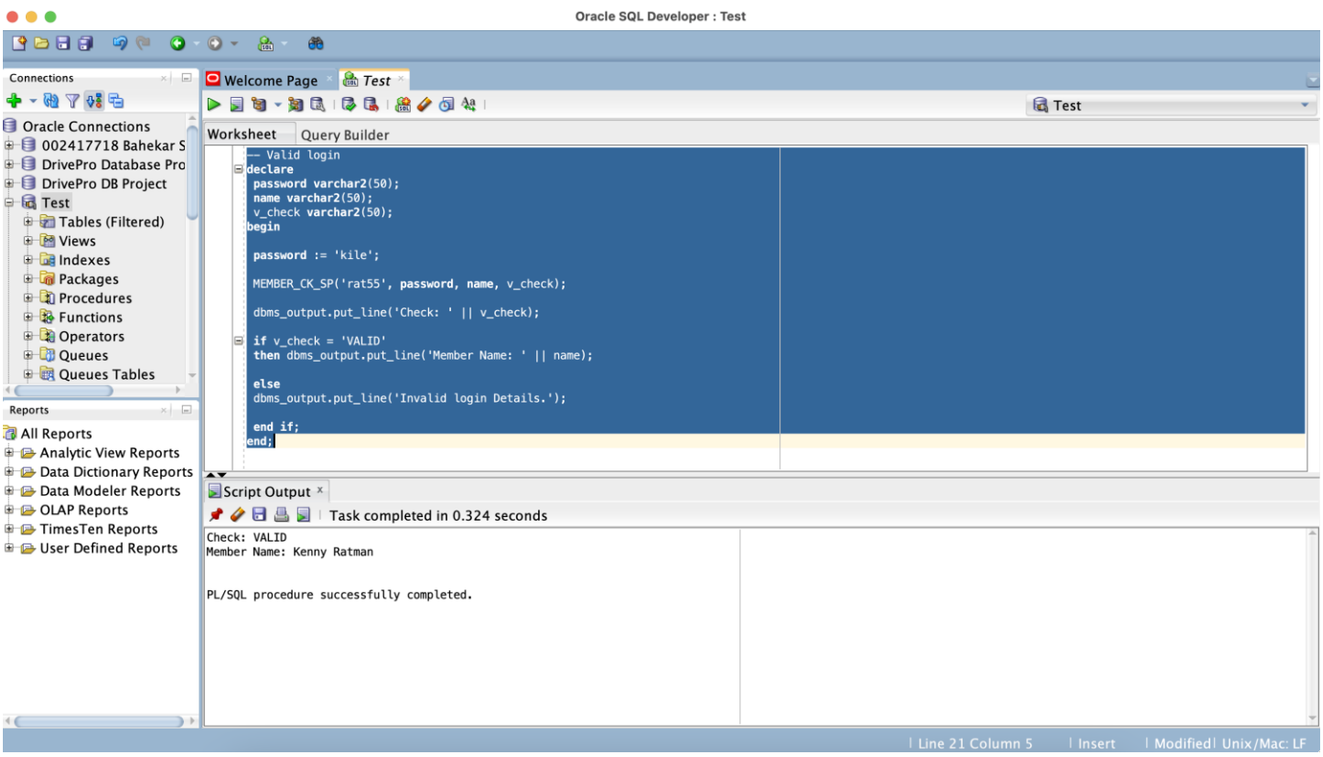
The home page of the Brewbean's Web site has an option for members to log on with their IDs and passwords. Develop a procedure named **`MEMBER_CK_SP`** that accepts the ID and password as inputs, checks whether they make up a valid logon, and returns the member name and cookie value. The name should be returned as a single text string containing the first and last name.

The head developer wants the number of parameters minimized so that the same parameter is used to accept the password and return the name value. Also, if the user doesn't enter a valid username and password, return the value **`INVALID`** in a parameter named **`p_check`**. Test the procedure using a valid logon first, with the username **`rat55`** and password **`kile`**. Then try it with an invalid logon by changing the username to **`rat`**.

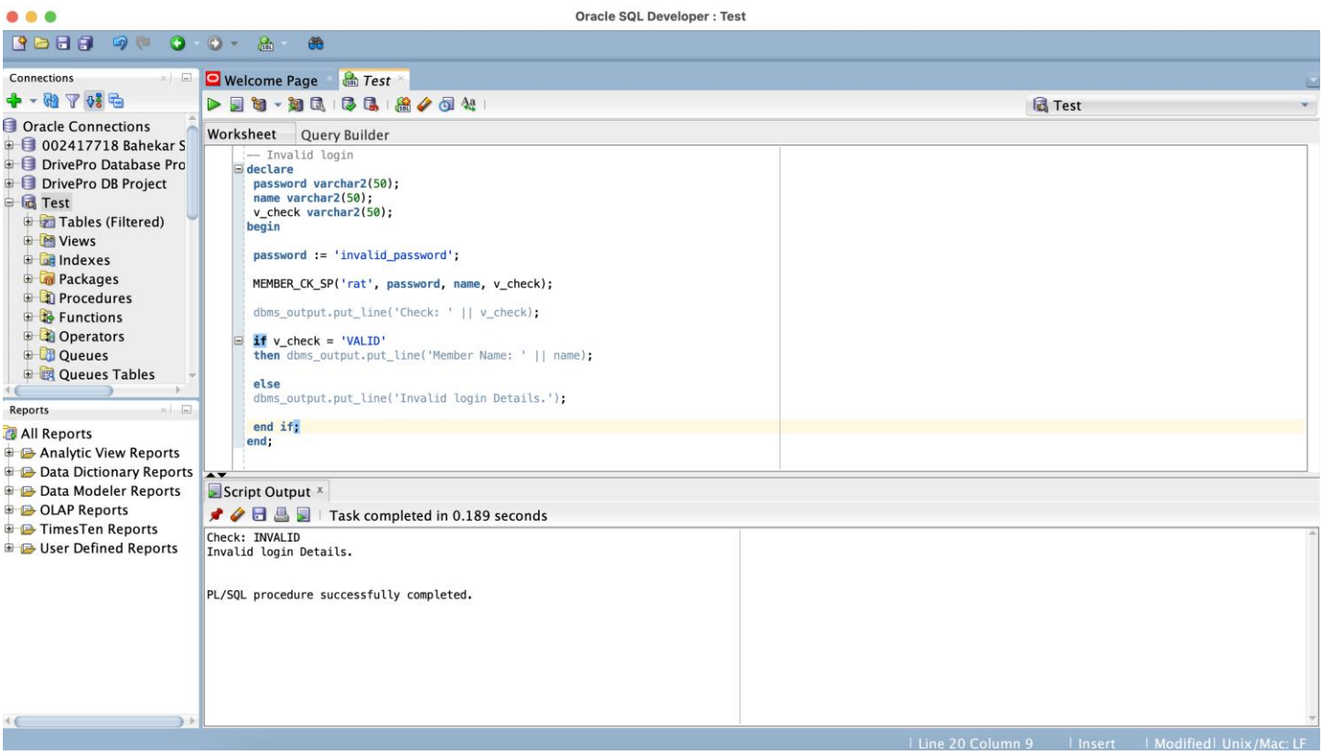
```
create or replace procedure MEMBER_CK_SP(
usernameTemp in varchar2,
passwordTemp in out varchar2,
nameTemp out varchar2,
p_check out varchar2
)
is
memberName varchar2(50);
begin
select FirstName || ' ' || LastName
into memberName
from BB_SHOPPER
where username = usernameTemp
and password = passwordTemp;
p_check := 'VALID';
nameTemp := memberName;
exception
when NO_DATA_FOUND
then p_check := 'INVALID';
nameTemp := NULL;
end;
```



```
-- Valid login
declare
password varchar2(50);
name varchar2(50);
v_check varchar2(50);
begin

password := 'kile';

MEMBER_CK_SP('rat55', password, name, v_check);

dbms_output.put_line('Check: ' || v_check);

if v_check = 'VALID'
then dbms_output.put_line('Member Name: ' || name);

else
dbms_output.put_line('Invalid login Details.');

end if;
end;
```



```
-- Invalid login
declare
password varchar2(50);
name varchar2(50);
v_check varchar2(50);
begin

password := 'invalid_password';

MEMBER_CK_SP('rat', password, name, v_check);

dbms_output.put_line('Check: ' || v_check);

if v_check = 'VALID'
then dbms_output.put_line('Member Name: ' || name);

else
dbms_output.put_line('Invalid login Details.');

end if;
end;
```



Function:

Question 9: Calculating a Shopper's Total Number of Orders

Another commonly used statistic in reports is the total number of orders a shopper has placed. Follow these steps to create a function named `NUM_PURCH_SF` that accepts a shopper ID and returns a shopper's total number of orders. Use the function in a `SELECT` statement to display the number of orders for shopper 23.

1. Develop and run a `CREATE FUNCTION` statement to create the **`NUM_PURCH_SF`** function. The function code needs to tally the number of orders (using an Oracle built-in function) by shopper. Keep in mind that the ORDERPLACED column contains a 1 if an order has been placed.
2. Create a `SELECT` query by using the `NUM_PURCH_SF` function on the IDSHOPPER column of the BB_SHOPPER table. Be sure to select only shopper 23.

create or replace function NUM_PURCH_SF(p_shopper_id in number) return number is

numOfPurchases number;

begin

select count(*) into numOfPurchases

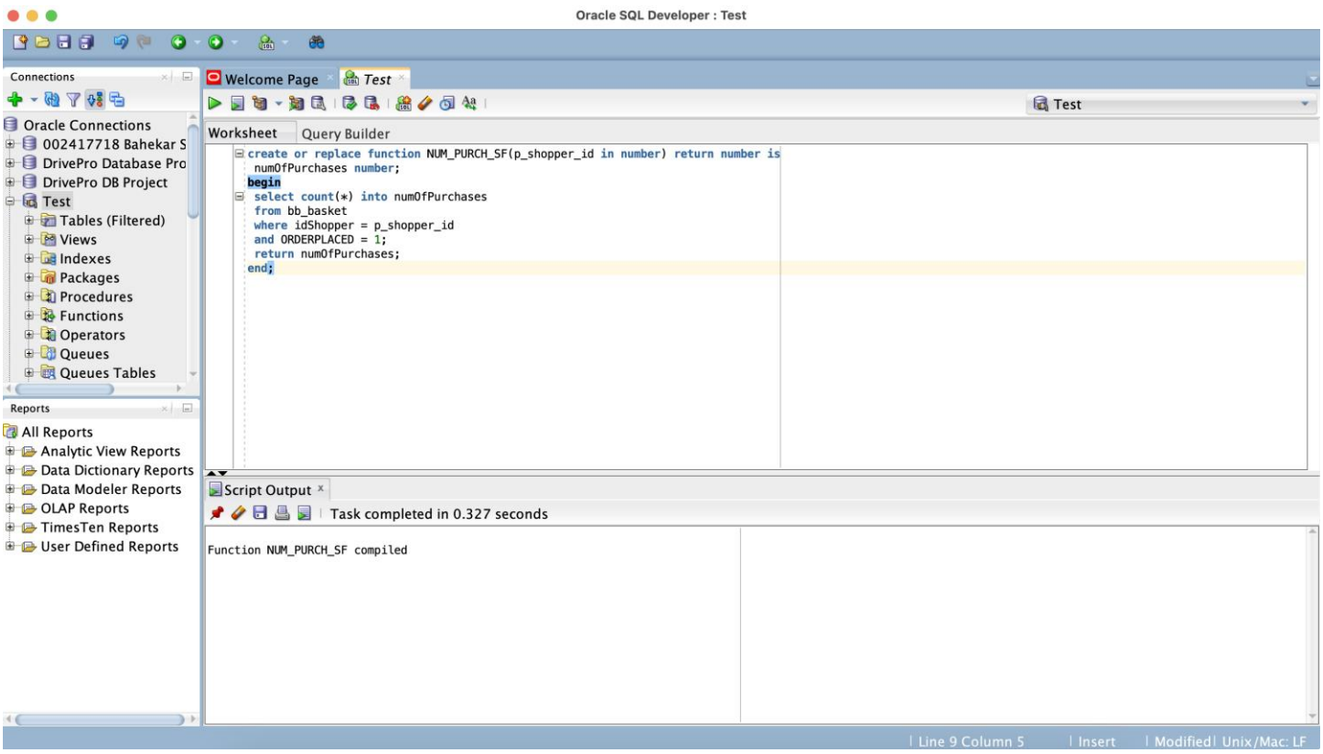
from bb_basket

where idShopper = p_shopper_id

and ORDERPLACED = 1;

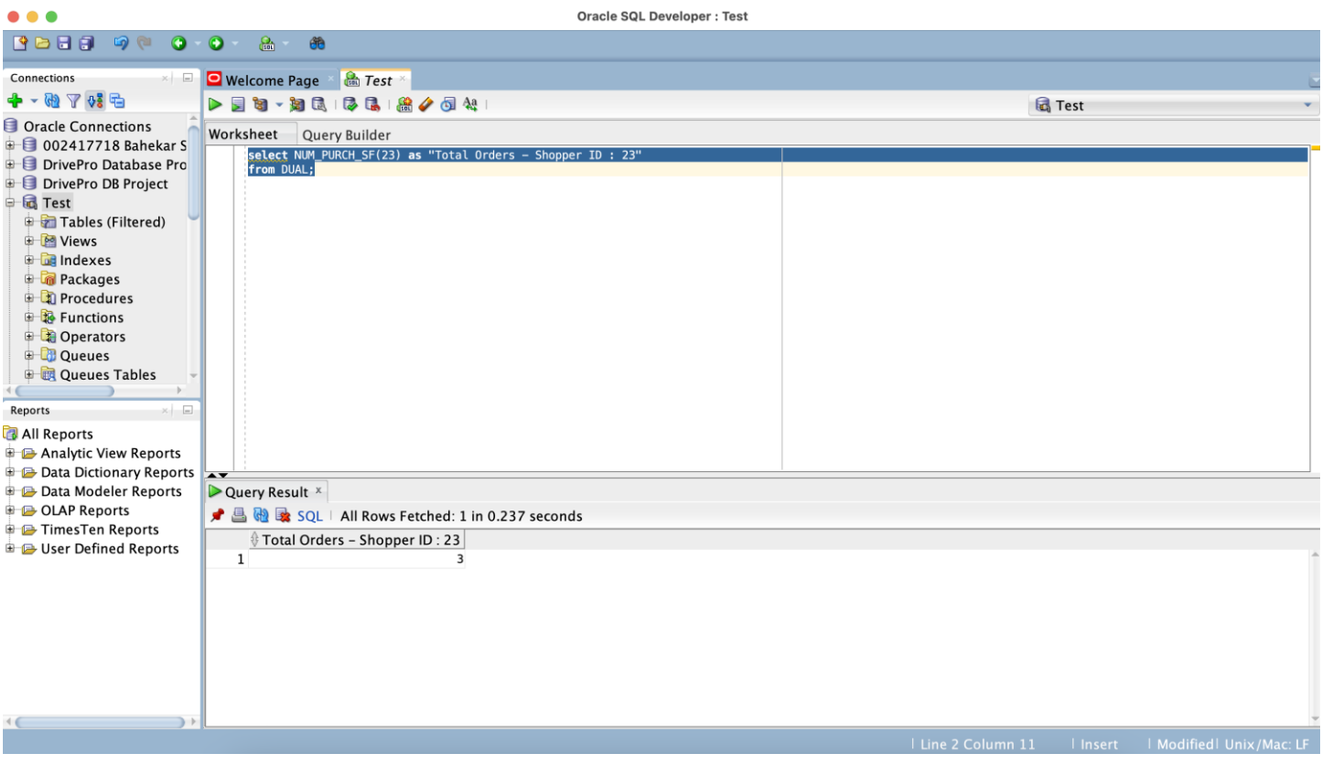
return numOfPurchases;

end;



select NUM_PURCH_SF(23) as "Total Orders - Shopper ID : 23"

from DUAL;



Question 10: Identifying the Weekday for an Order Date:

The day of the week that baskets are created is often analyzed to determine consumer-shopping patterns. Create a function named DAY_ORD_SF that accepts an order date and returns the weekday. Use the function in a SELECT statement to display each basket ID and the weekday the order was created. Write a second SELECT statement, using this function to display the total number of orders for each weekday. (*Hint*: Call the TO_CHAR function to retrieve the weekday from a date.)

1. Develop and run a CREATE FUNCTION statement to create the **DAY_ORD_SF** function. Use the DTCREATED column of the BB_BASKET table as the date the basket is created. Call the TO_CHAR function with the DAY option to retrieve the weekday for a date value.
2. Create a SELECT statement that lists the basket ID and weekday for every basket.
3. Create a SELECT statement, using a GROUP BY clause to list the total number of baskets per weekday. Based on the results, what's the most popular shopping day?

create or replace function DAY_ORD_SP(orderDate in date)

return varchar2

is

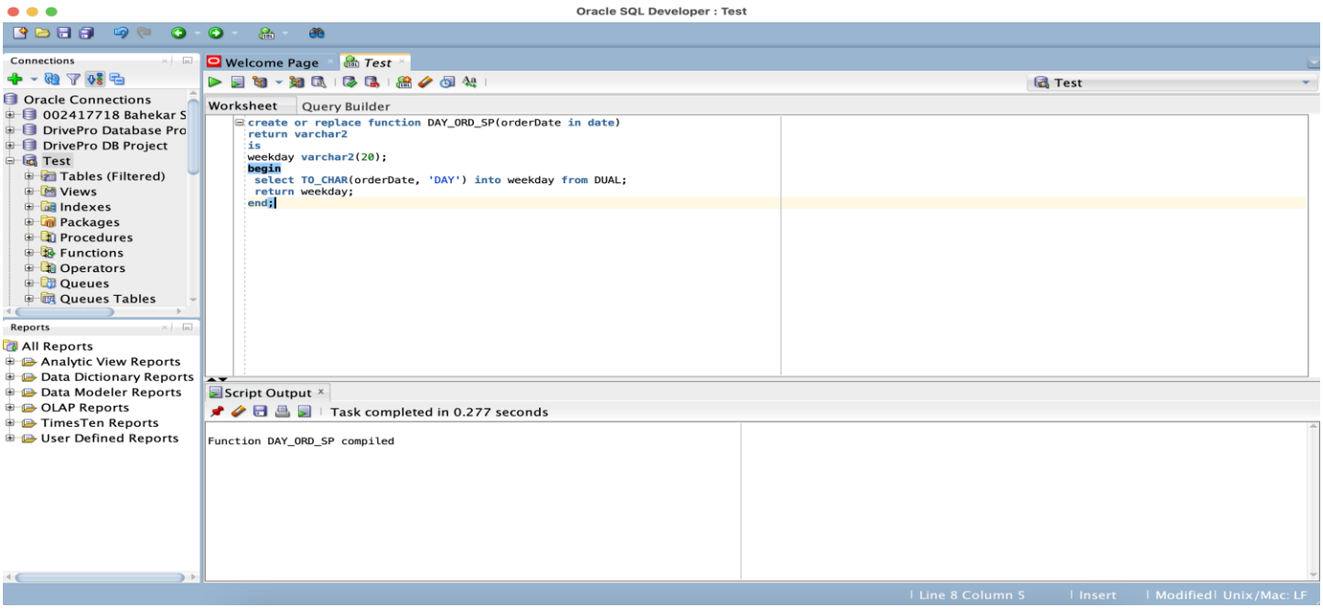
weekday varchar2(20);

begin

select TO_CHAR(orderDate, 'DAY') into weekday from DUAL;

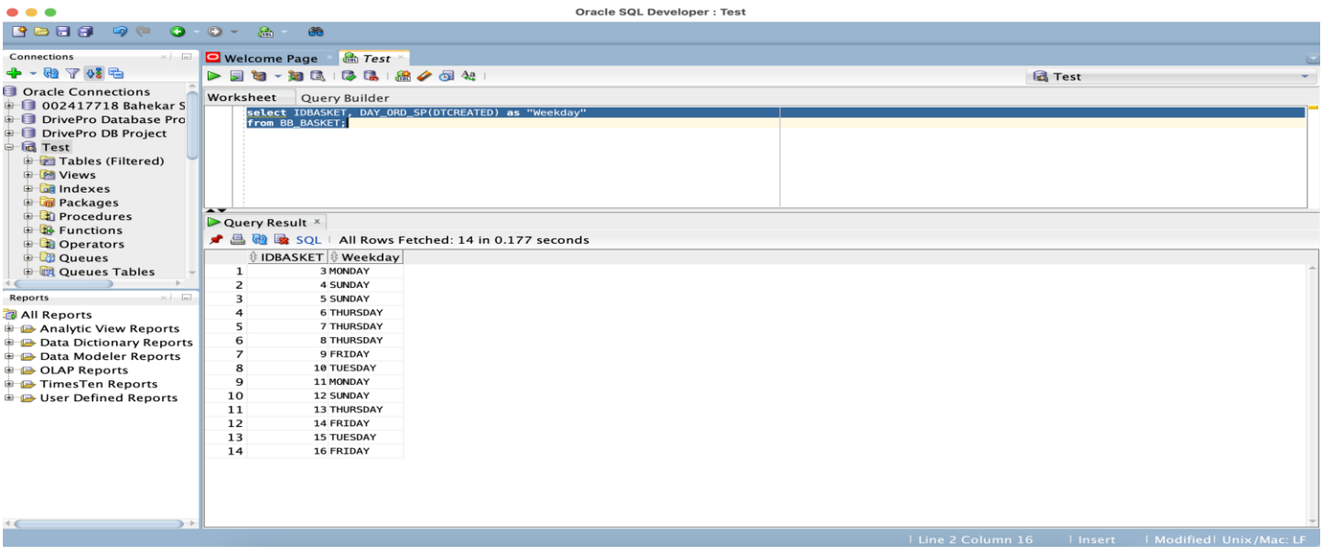
return weekday;

end;



select IDBASKET, DAY_ORD_SP(DTCREATED) as "Weekday"

from BB_BASKET;



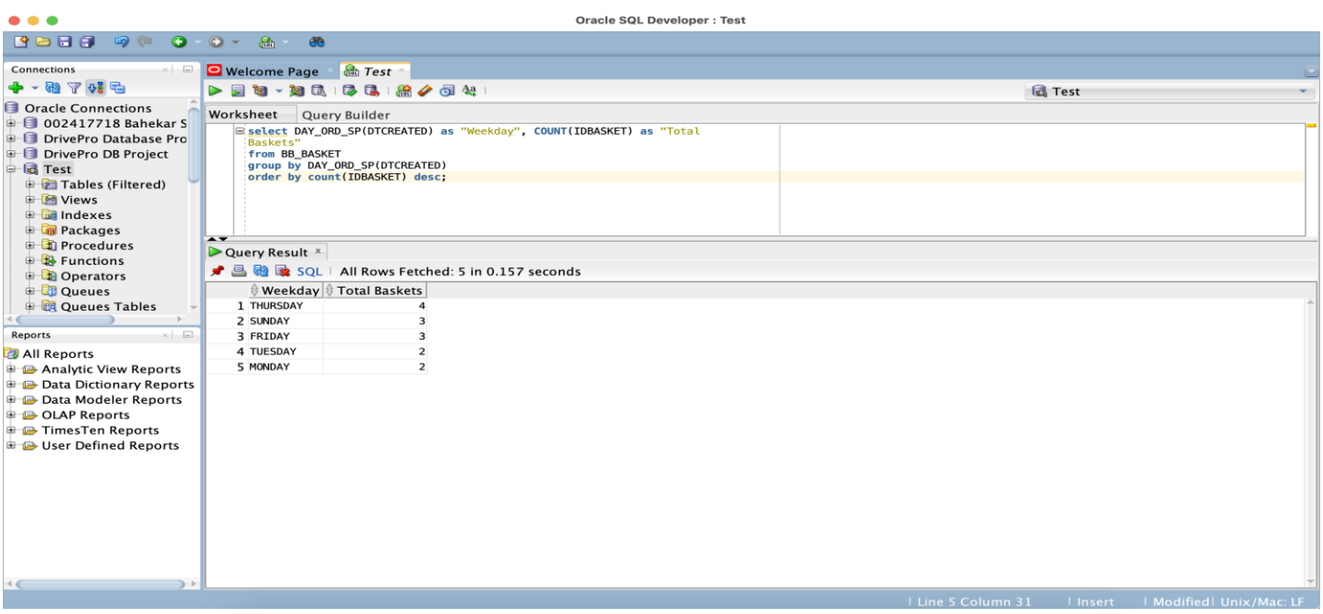
select DAY_ORD_SP(DTCREATED) as "Weekday", COUNT(IDBASKET) as "Total

Baskets"

from BB_BASKET

group by DAY_ORD_SP(DTCREATED)

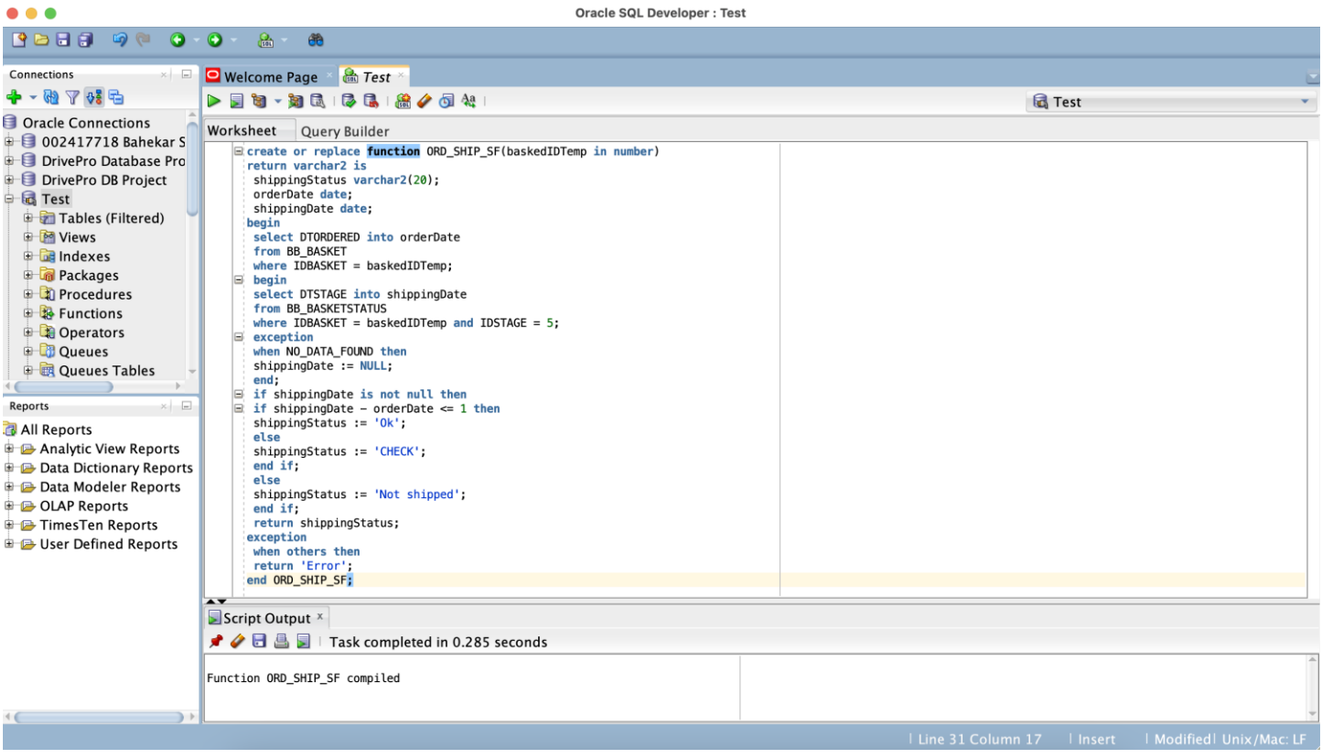
order by count(IDBASKET) desc;



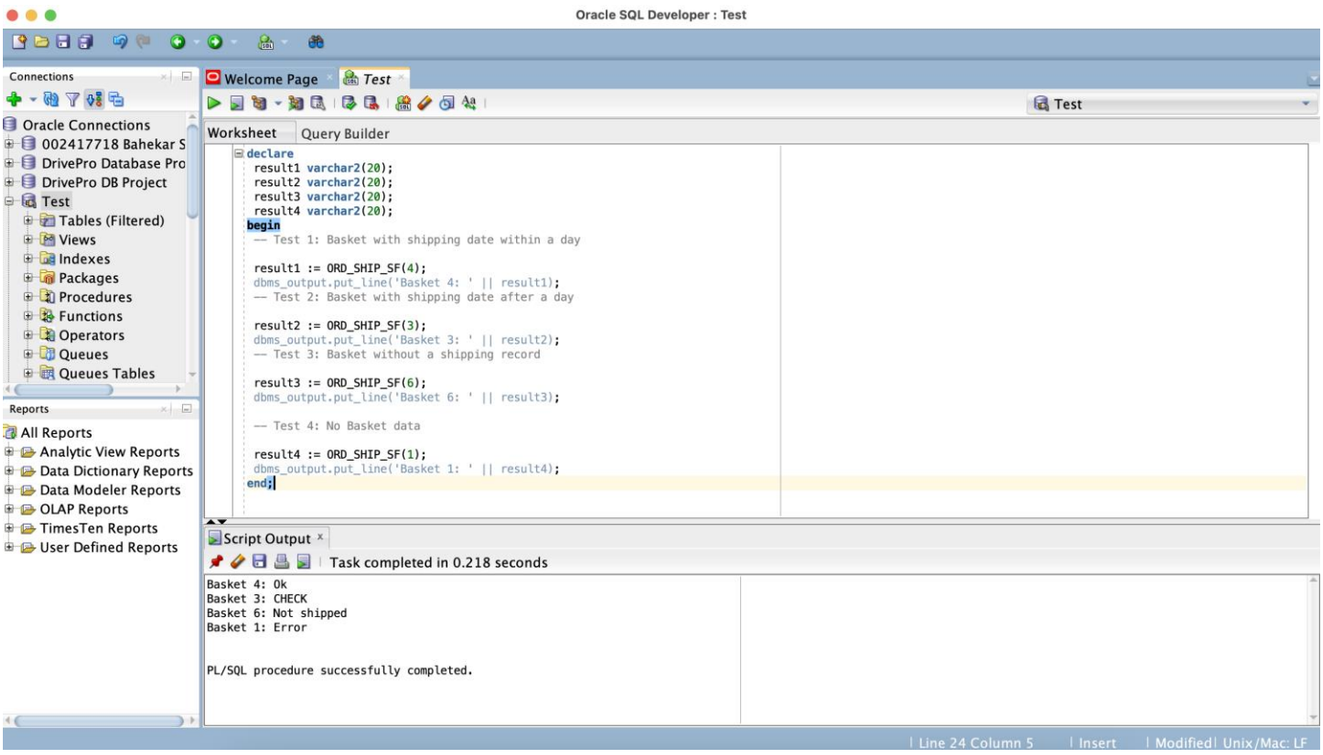
Question 11: Calculating Days Between Ordering and Shipping:

An analyst in the quality assurance office reviews the time elapsed between receiving an order and shipping the order. Any orders that haven't been shipped within a day of the order being placed are investigated. Create a function named **ORD_SHIP_SF** that calculates the number of days between the basket's creation date and the shipping date. The function should return a character string that states `OK` if the order was shipped within a day or `CHECK` if it wasn't. If the order hasn't shipped, return the string `Not shipped`. The IDSTAGE column of the BB_BASKETSTATUS table indicates a shipped item with the value 5, and the DTSTAGE column is the shipping date. The DTORDERED column of the BB_BASKET table is the order date. Review data in the BB_BASKETSTATUS table, and create an anonymous block to test all three outcomes the function should handle.

```
create or replace function ORD_SHIP_SF(baskedIDTemp in number)
return varchar2 is
shippingStatus varchar2(20);
orderDate date;
shippingDate date;
begin
select DTORDERED into orderDate
from BB_BASKET
where IDBASKET = baskedIDTemp;
begin
select DTSTAGE into shippingDate
from BB_BASKETSTATUS
where IDBASKET = baskedIDTemp and IDSTAGE = 5;
exception
when NO_DATA_FOUND then
shippingDate := NULL;
end;
if shippingDate is not null then
if shippingDate - orderDate <= 1 then
shippingStatus := 'Ok';
else
shippingStatus := 'CHECK';
end if;
else
shippingStatus := 'Not shipped';
end if;
return shippingStatus;
exception
when others then
return 'Error';
end ORD_SHIP_SF;
```




```
declare
result1 varchar2(20);
result2 varchar2(20);
result3 varchar2(20);
result4 varchar2(20);
begin
-- Test 1: Basket with shipping date within a day

result1 := ORD_SHIP_SF(4);
dbms_output.put_line('Basket 4: ' || result1);
-- Test 2: Basket with shipping date after a day

result2 := ORD_SHIP_SF(3);
dbms_output.put_line('Basket 3: ' || result2);
-- Test 3: Basket without a shipping record

result3 := ORD_SHIP_SF(6);
dbms_output.put_line('Basket 6: ' || result3);

-- Test 4: No Basket data

result4 := ORD_SHIP_SF(1);
dbms_output.put_line('Basket 1: ' || result4);
end;
```

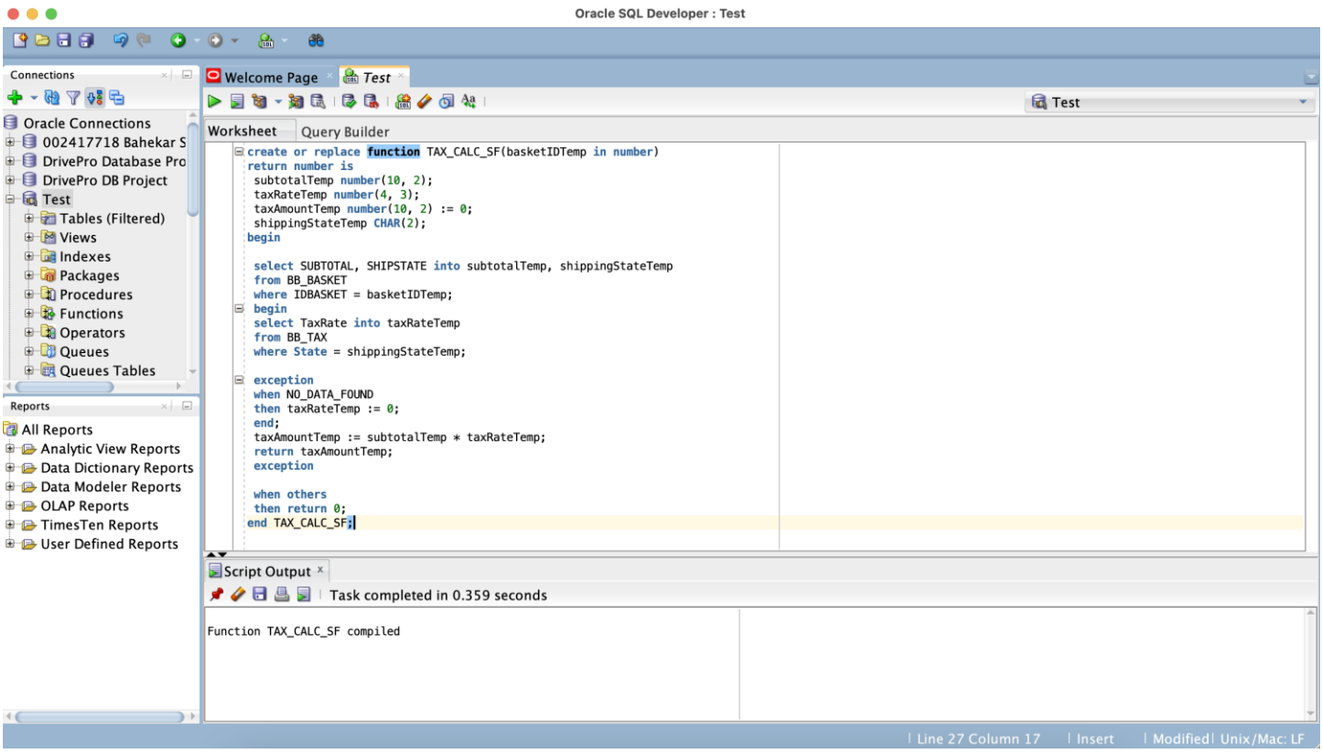


Question 12: Calculating an Order's Tax Amount:

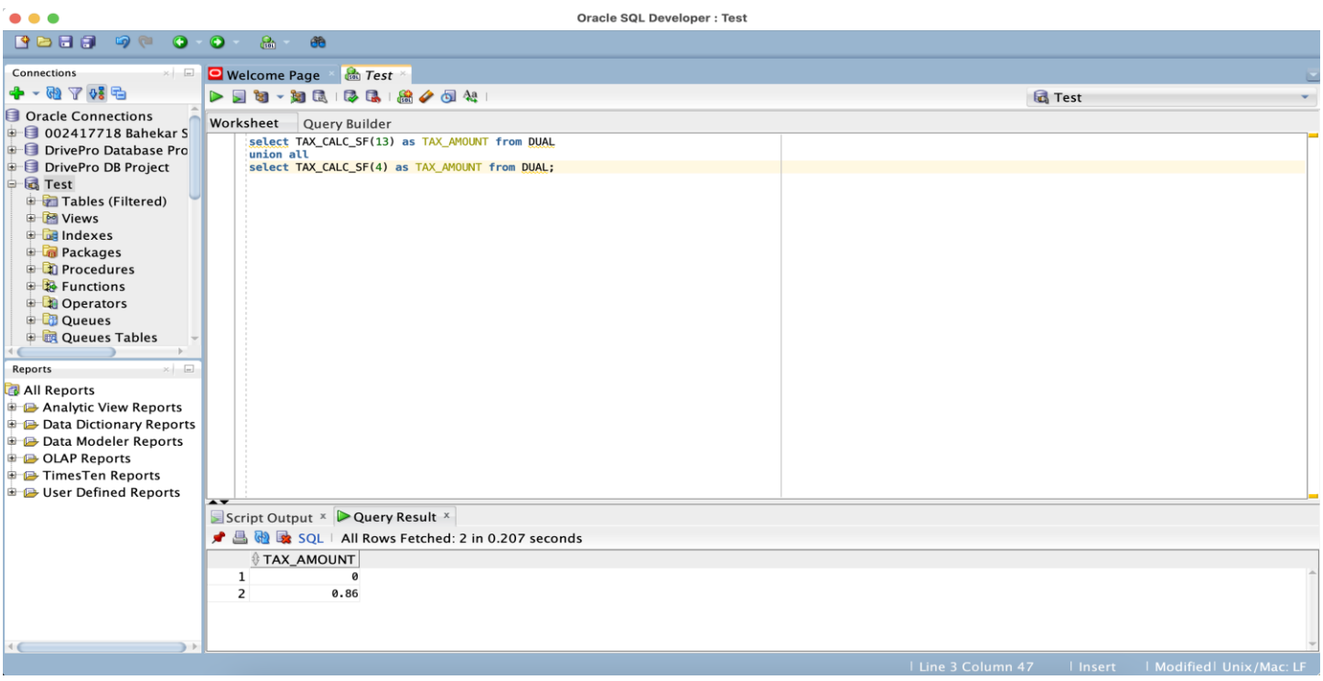
Create a function named TAX_CALC_SF that accepts a basket ID, calculates the tax amount by using the basket subtotal, and returns the correct tax amount for the order. The tax is determined by the shipping state, which is stored in the BB_BASKET table. The BB_TAX table contains the tax rate for states that require taxes on Internet purchases. If the state isn't listed in the tax table or no shipping state is assigned to the basket, a tax amount of zero should be applied to the order. Use the function in a SELECT statement that displays the shipping costs fora basket that has tax applied and a basket with no shipping state.

```
create or replace function TAX_CALC_SF(basketIDTemp in number)
return number is
subtotalTemp number(10, 2);
taxRateTemp number(4, 3);
taxAmountTemp number(10, 2) := 0;
shippingStateTemp CHAR(2);
begin

select SUBTOTAL, SHIPSTATE into subtotalTemp, shippingStateTemp
from BB_BASKET
where IDBASKET = basketIDTemp;
begin
select TaxRate into taxRateTemp
from BB_TAX
where State = shippingStateTemp;

exception
when NO_DATA_FOUND
then taxRateTemp := 0;
end;
```

```
taxAmountTemp := subtotalTemp * taxRateTemp;
return taxAmountTemp;
exception

when others
then return 0;
end TAX_CALC_SF;
```



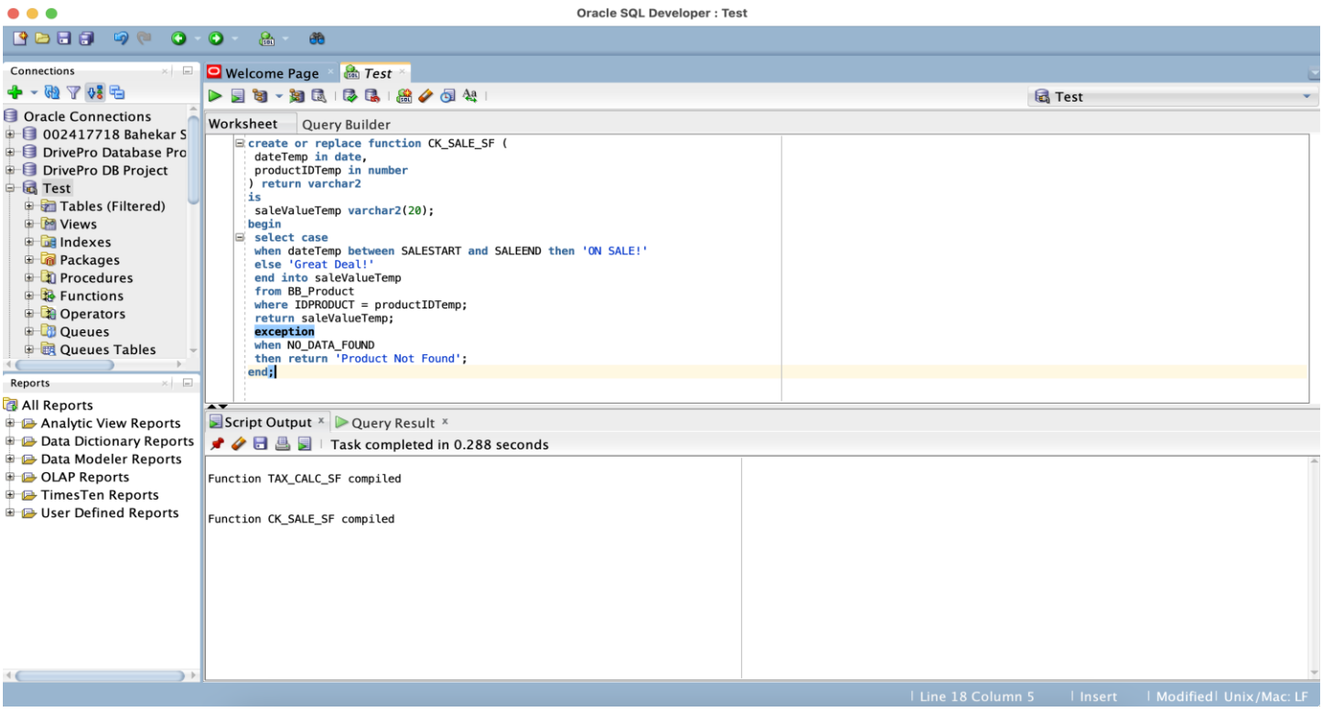

```
select TAX_CALC_SF(13) as TAX_AMOUNT from DUAL
union all
select TAX_CALC_SF(4) as TAX_AMOUNT from DUAL;
```



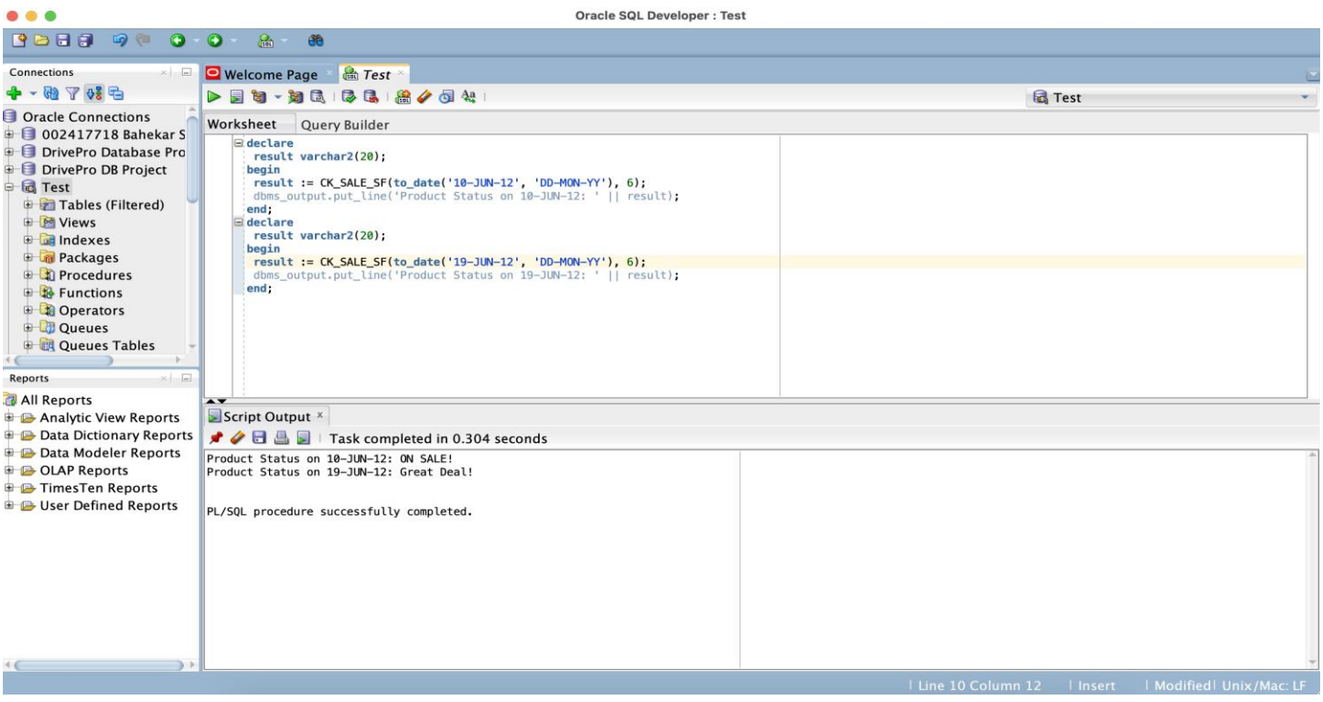
Question 13: Identifying Sale Products:

When a product is placed on sale, Brewbean's records the sale's start and end dates in columns of the BB_PRODUCT table. A function is needed to provide sales information when a shopper selects an item. If a product is on sale, the function should return the value ON SALE!. However, if it isn't on sale, the function should return the value Great Deal!. These values are used on the product display page. Create a function named CK_SALE_SF that accepts a date and product ID as arguments, checks whether the date falls within the product's sale period, and returns the corresponding string value. Test the function with the product ID 6 and two dates: 10-JUN-12and 19-JUN-12. Verify your results by reviewing the product sales information.

```
create or replace function CK_SALE_SF (
dateTemp in date,
productIDTemp in number
) return varchar2
is
saleValueTemp varchar2(20);
begin
select case
when dateTemp between SALESTART and SALEEND then 'ON SALE!'
else 'Great Deal!'
end into saleValueTemp
from BB_Product
where IDPRODUCT = productIDTemp;
return saleValueTemp;
exception
when NO_DATA_FOUND
then return 'Product Not Found';
end;
```



```
declare
result varchar2(20);
begin
result := CK_SALE_SF(to_date('10-JUN-12', 'DD-MON-YY'), 6);
dbms_output.put_line('Product Status on 10-JUN-12: ' || result);
end;
declare
result varchar2(20);
begin
result := CK_SALE_SF(to_date('19-JUN-12', 'DD-MON-YY'), 6);
dbms_output.put_line('Product Status on 19-JUN-12: ' || result);
end;
```




```
select IDPRODUCT, SALESTART, SALEEND
from BB_PRODUCT
where IDPRODUCT = 6;
```