**Phase 3: Final Database Reorganization**

**Erdenesukh (Eric) Tsendjav, Bibek Amatya, Gowtham Ravuru, Avinash Pentyala, Mounisri Muppidi**

# **Introduction**

**This report explains the process we followed to complete this assignment and demonstrates the correctness of our solutions. All items were completed successfully and the solutions for each item were incorporated into the Phase 3 database script.**

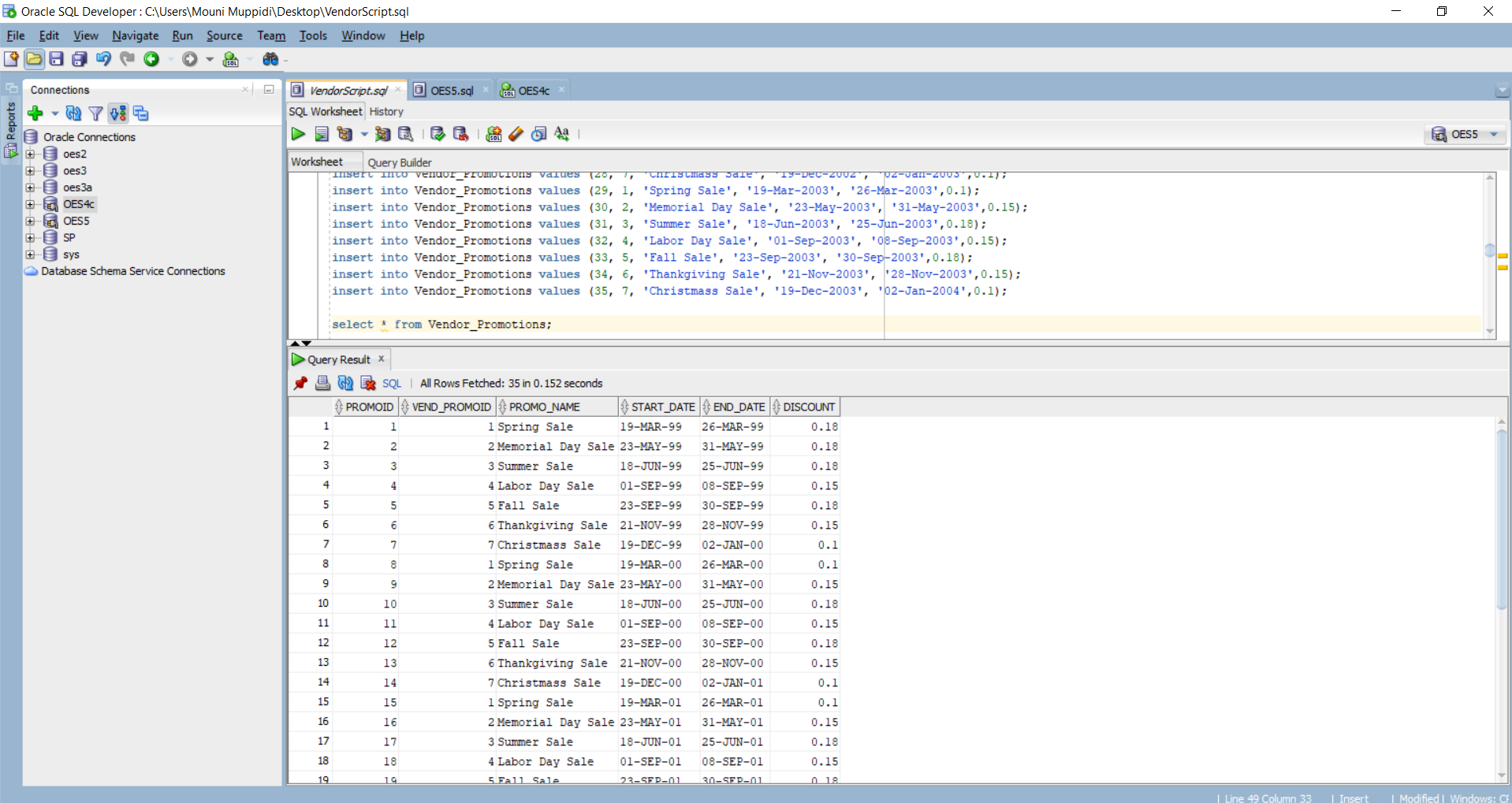
# **Item Results and Discussion**

**Many of the items in the problem set require proof to demonstrate that we implemented them correctly. This section contains a discussion of each item, along with the required proof.**

**Use case 7.1: Vendor Promotion**

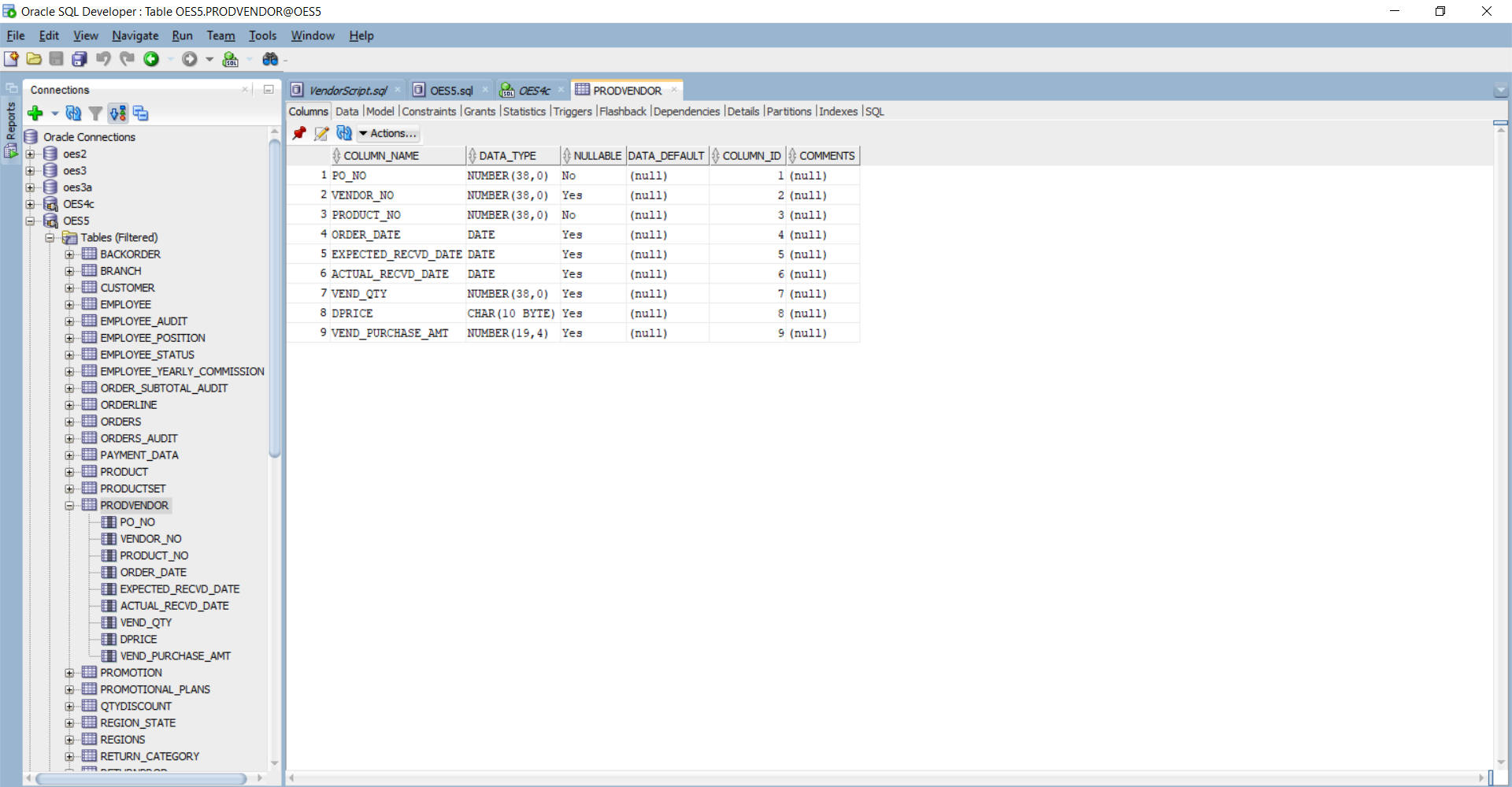
**Vendors has 7 promotional prices**

**The actual dates for applying promotional prices applies the same as sale**

****

The first task in the Vendor promotion is to create a table. So by using the create table script we have created a Vendor\_Promotions table. Given the actual date for applying vendor promotional prices applies the same as sale so by using the excel sheet we have created a table for different years like 2000, 2001, 2002, 2003 and inserted the values. From the above figure we can clearly see that the Vendor\_Promotions table was created and data was inserted and it is running successfully.

**Using ALTER command add a field to prodvendor table called Vend\_Purchase\_Amt**

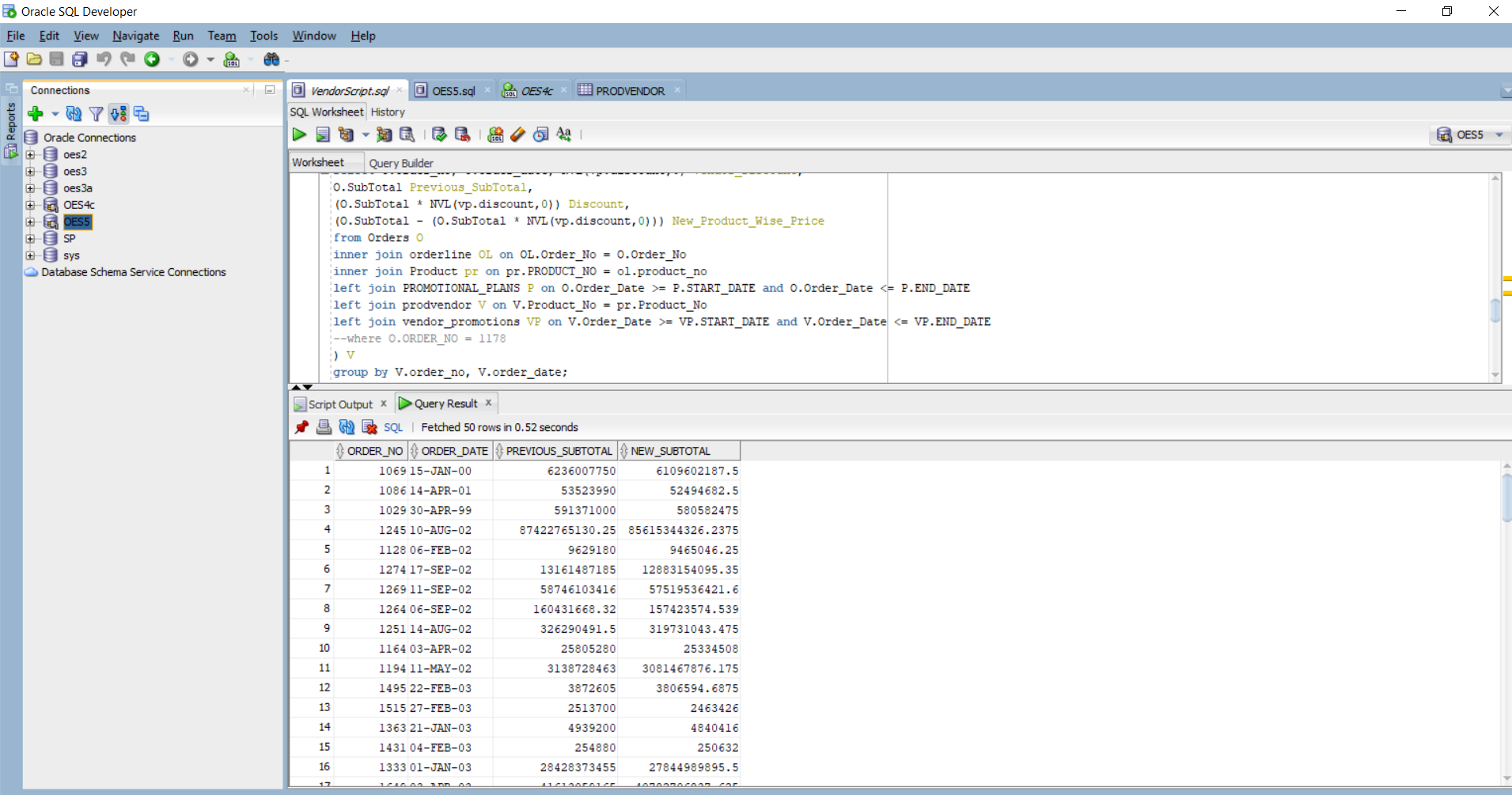
****

Altering means we have to add a column to an existing table with an alter statement. By using the alter command we have added the VEND\_PURCHASE\_AMT in the prodvendor table and updated it. The query can be seen below: It is successfully added to the prodvendor and runs successfully.

alter table prodvendor

add Vend\_Purchase\_Amt number(19, 4);

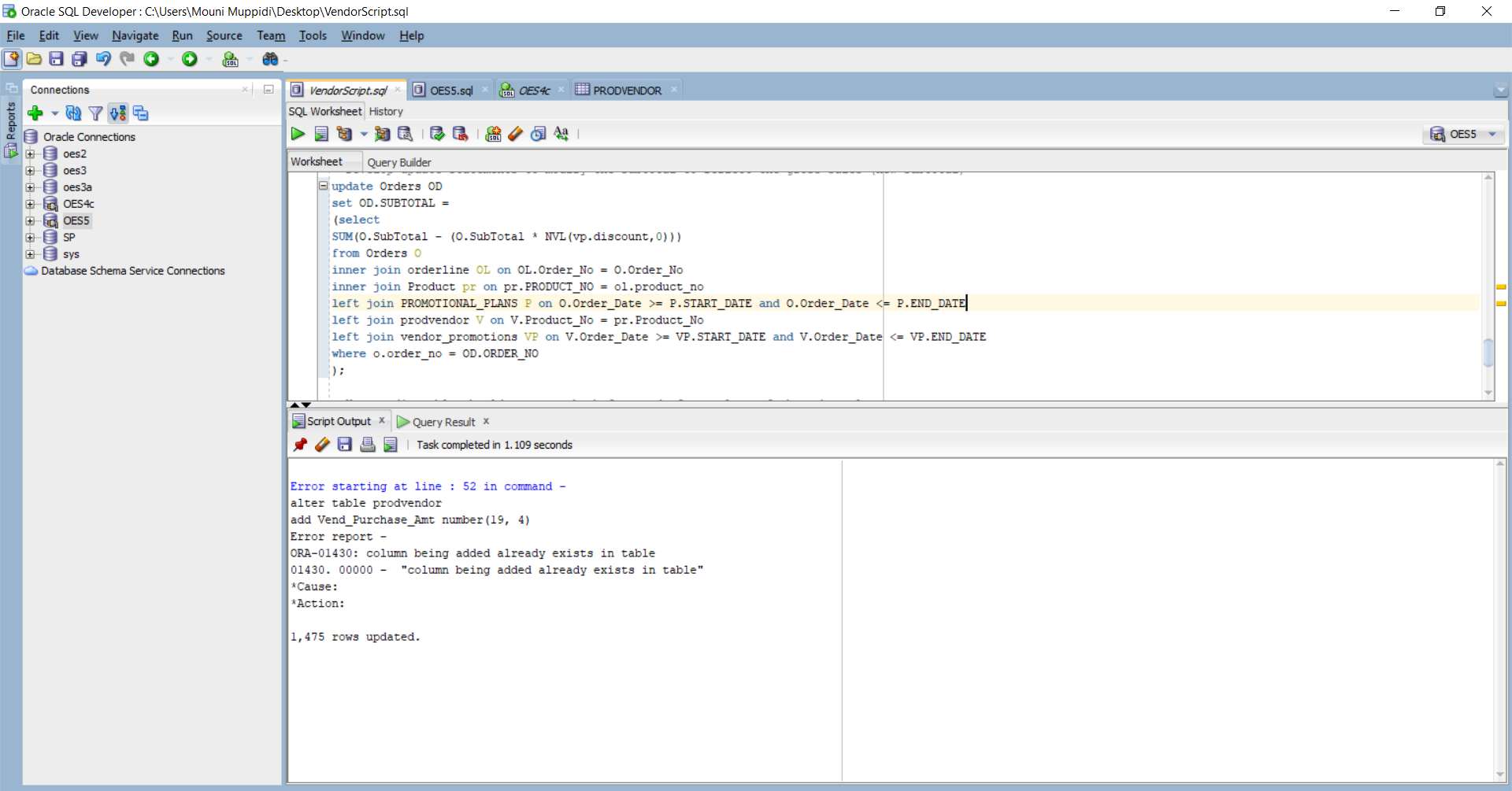
**Develop a report to show subtotal before applying promotional prices and after promotional prices**

****

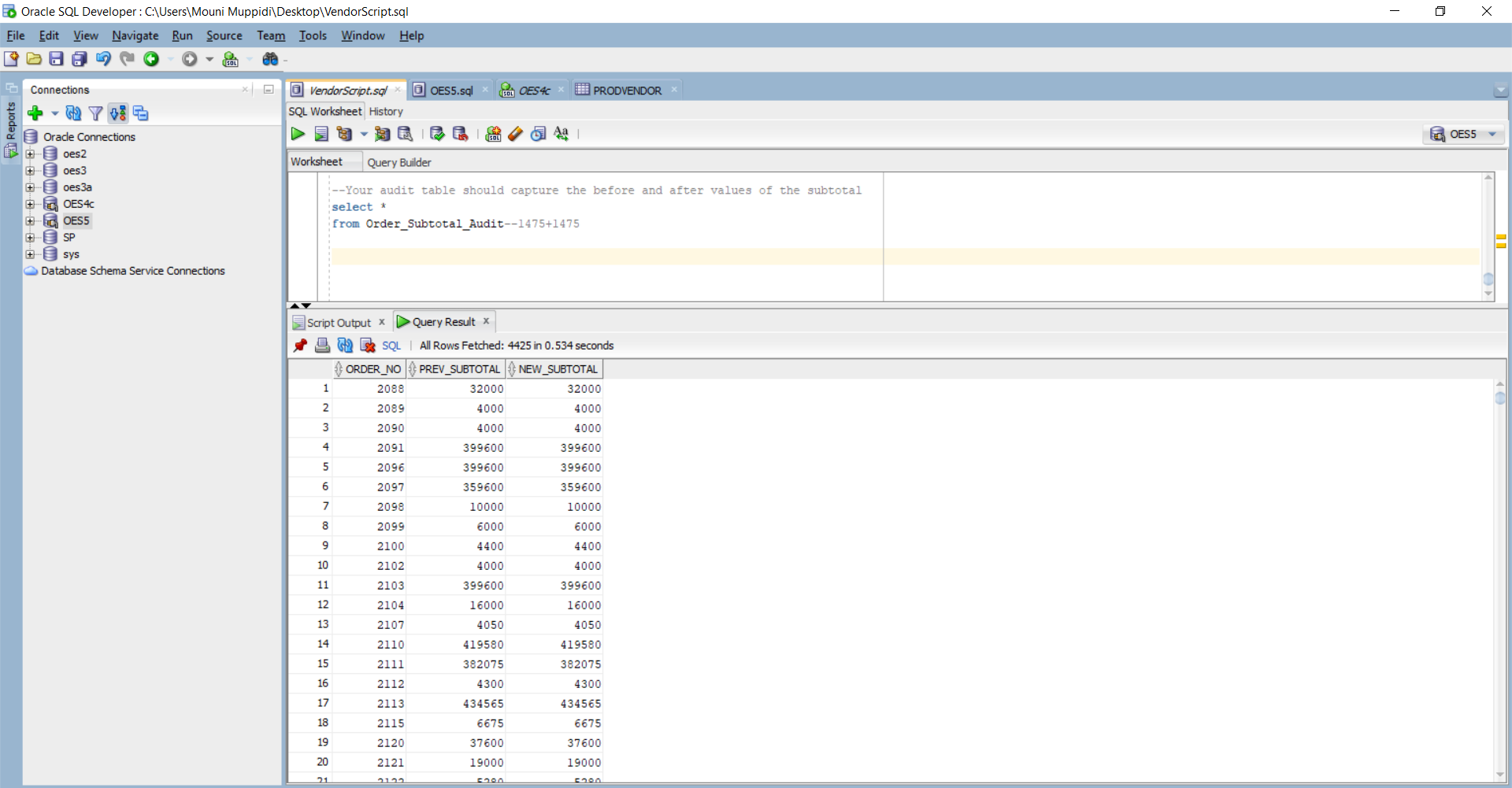
Here the prodvendor is related to the product table. And in the product table there is product\_no and the vendor table also has product\_no. So by using that relation we have joined the two tables. Now the order\_date is taken from the vendor\_promotions table. The vendor\_promotions discount is NVL(vp,discount,0). For the discount value we have multiplied subtotal and vendor\_promotions discount. And in the subquery we had done the summation as it is calculated product wise. From the above report we can clearly see the before and after subtotal values.

**Develop update statements to modify the subtotal to reflect the gross sales (new subtotal)**

We have used the same query as above. But here we have no use of subquery because we directly need to update summation. By adding the SUM statement we have updated the subtotal value of the order table. And during update all the values will be transferred to the audit log table. We have modified the subtotal values and all the statements are successfully updated.

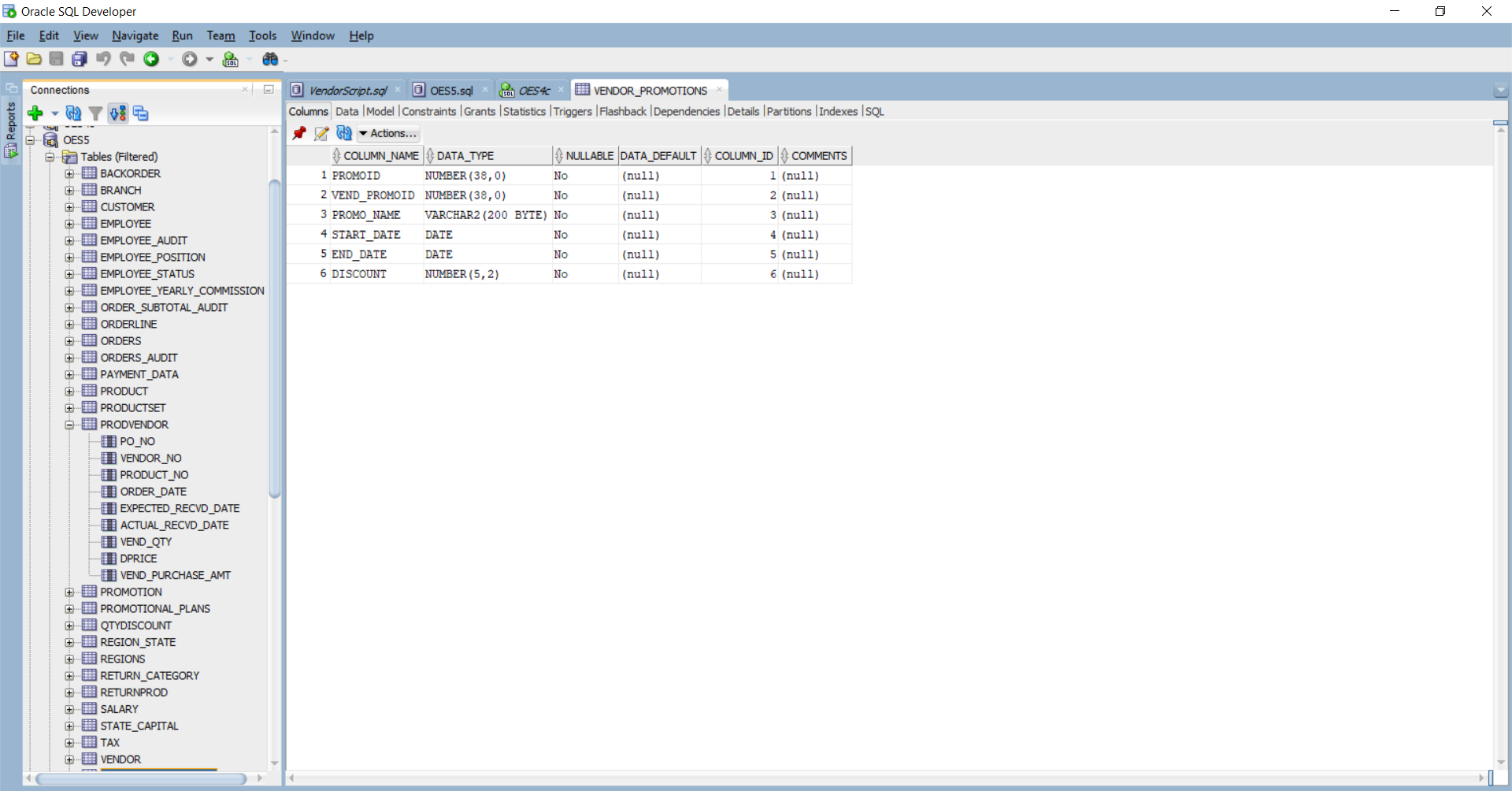
****

**Your audit table should capture the before and after values of the subtotal**

****

As we have updated the subtotal values all the values are successfully updated and also transferred to the audit table. Now we can see that the prev\_subtotal values and the new\_subtotal values are executed fine.

**After implementing the vendor sales promotional prices, append the changes to form OES5**

****

Now finally we have added all the changes to the new script to form an OES5 table. As we can see all the tables are successfully executed, the prodvendor table is altered and the vendor\_promotions table also created without any errors and has all the data in it.

## 

## 

## 

## 

## 

## 

## 

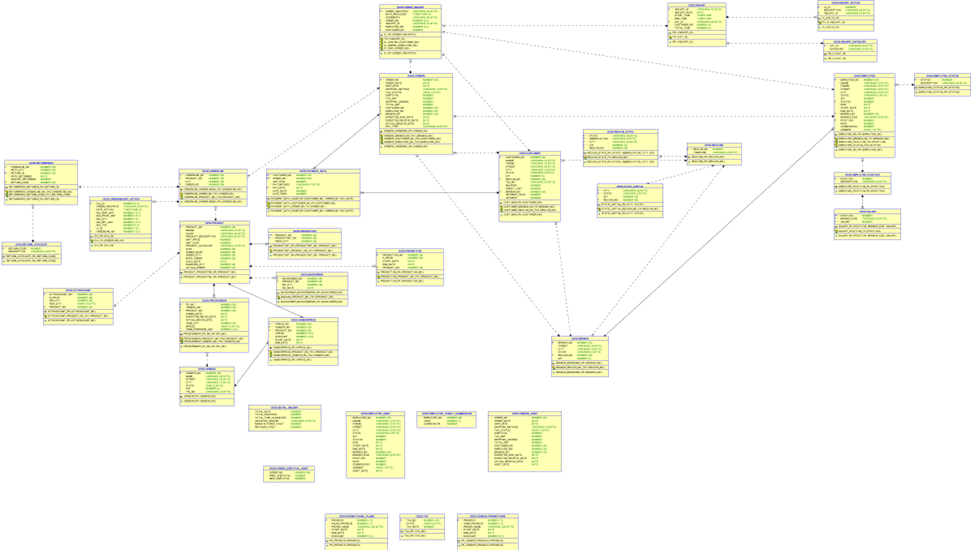
## **Use Case #7.2:**

#1

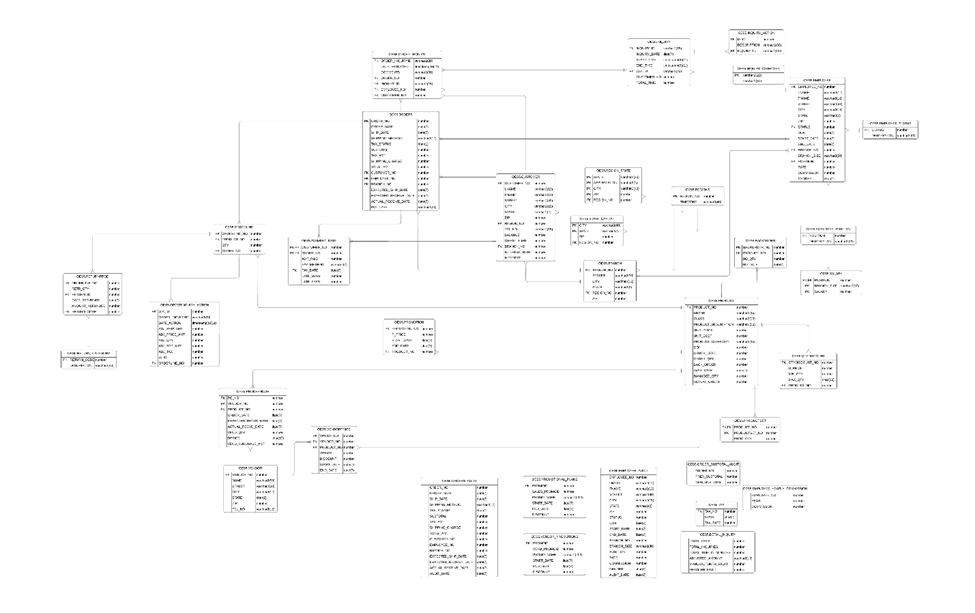
**Extend the database to support customer inquirie**s

The script to populate Inquiry tables called Inquiry\_Populate had so many issues with the insert statements, such as data types being different from what was specified on tables such as inquiry, Order\_inquiry, orderinquiry\_action. Their date columns were specified as DATE but the date value being inserted to said columns were very long timestamps that are not supported by the Date data type. So, I had to analyze all the date data being added to these tables and confirmed that extra decimal second values were not needed since they were all zeros. So, first I removed the extra decimal zero seconds, and also updated the said tables’ columns into TIMESTAMP from DATE, so I can successfully enter the inquiry data.

**Oracle Data Model**

****

**Lucidchart Data Model**

****

#2

**Description and action related to return categories**

| **Inquiry Categories** | **Description** | **Our Action** |
| --- | --- | --- |
| **Product Defect** | **Return to Vendor at the end of each month** | **Credit voucher will be rec’d after 4 days** |
| **Product Mismatch** | **Product will be restoked (added back to inventory** | **Correct product shipped one day air to customer** |
| **Shipping Delay** | **Customer rec’d product late** | **Shipping cost refunded to customer** |
| **Product Damage** | **Product damaged during shipping** | **Our cost. New product to customer with no shipping cost. Keep track of it in Product table: "Damaged Product"** |
| **Manufacturing Defect** | **Return to Vendor at the end of each month** | **Credit voucher will be rec'd after 4 days** |
| **Extra Charges** | **inaccurate charges** | **Refund the customer** |
| **Shipping Cost** | **Refund customer late shipping** | **Refund shipping cost for any late arrival (our credit voucher)** |

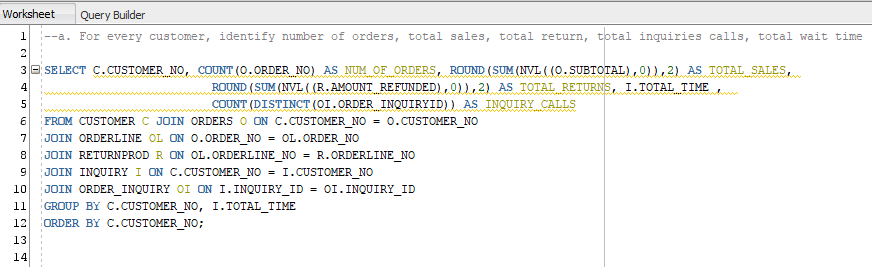
**List all changes to database**

* Obviously, first of all, I’ve added an audit table called INQUIRY\_CATEGORY\_AUDIT that has identical columns and metadata with the INQUIRY\_CATEGORY table.
* Then created a before update trigger called INQUIRY\_CATEGORY\_BEFORE\_UPDATE that inserts all the columns from the INQUIRY\_CATEGORY table into the INQUIRY\_CATEGORY\_AUDIT table whenever there are changes on the former.
* Following that, I had to add Description and Our Action columns from the above table to the INQUIRY\_CATEGORY table. The reason why I added these columns to the INQUIRY\_CATEGORY table instead of the said “RETURN\_CATEGORY” table is that there is no useful connection to that table from any of the other tables. It only has Return\_Code as a FK in the RETURNPROD table but I really didn’t see the upside of adding these columns into the Return\_Category table since it has no connection whatsoever to the Inquiry tables such as Inquiry\_Category. So, the way I decided to go was adding these columns into the INQUIRY\_CATEGORY table itself, so whenever any other table or business logic needs to use the inquiry categories, they can also reference the appropriate descriptions, as well as course of action to take from the internal business side.
* Finally, I have updated all the appropriate Inquiry\_Category records that were referenced in the above table and left out the rest of the ones that were not referenced in the table**.**

#3 Developing the following queries

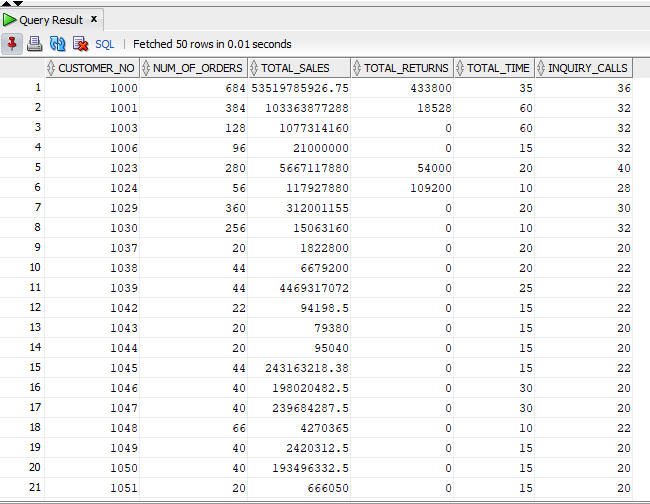
a. For every customer, identify number of orders, total sales, total return, total inquiries calls, total wait time

The first query was to identify the number of orders, total sales, total returns, total inquiries calls, and total wait time for every customer. We used the following SQL queries to generate the desired output.



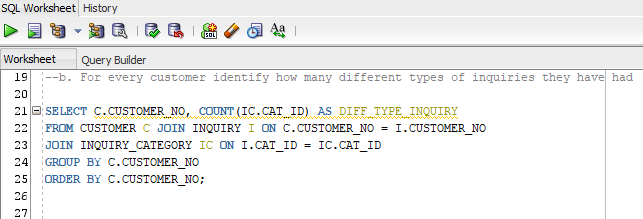
Output:-

The output below displays the number of orders, total sales, total returns, total time, and inquiry calls for every customer.

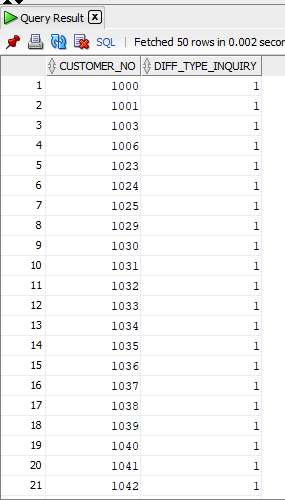


b. For every customer identify how many different types of inquiries they have had

The second query was to identify how many different types of inquiries they have had. And for that we have used the following SQL queries to generate the desired output.



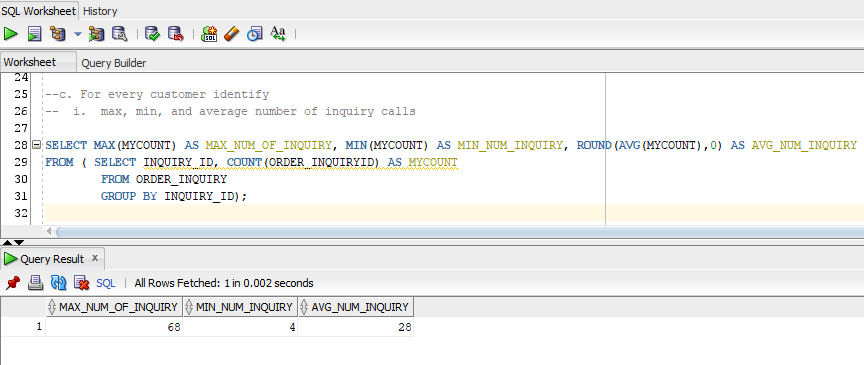
The output below displays a list of customers and different types of inquiries they have had.



c. For every customer identify

i. max, min, and average number of inquiry calls

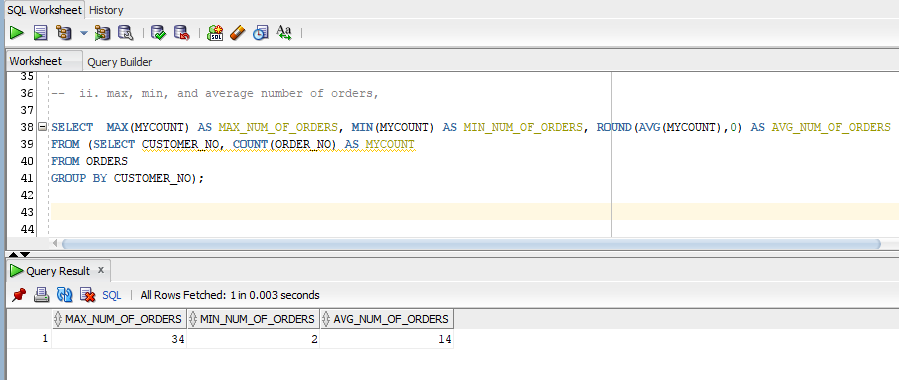
For this query, we have to find the max, min, and average number of inquiry calls. And for that we have used the following SQL queries to generate the desired output.



The output above displays the maximum number of inquiries, minimum number of inquiries, and average number of inquiries for any given customer.

ii. max, min, and average number of orders,

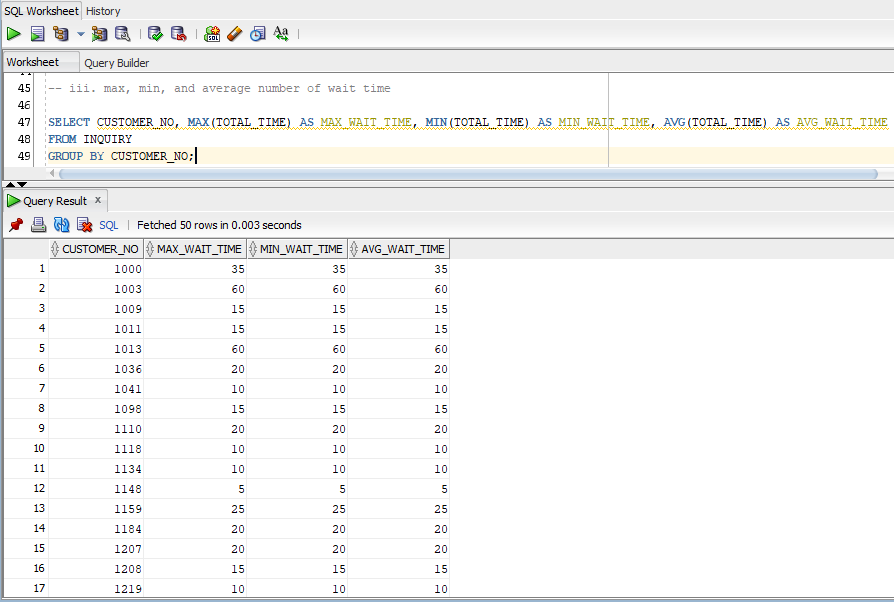
For this query, we have to find the max, min, and average number of orders. And for that we have used the following SQL queries to generate the desired output.



The output above displays the maximum number of orders, minimum number of orders, and average number of orders for any given customer.

iii. max, min, and average number of wait time

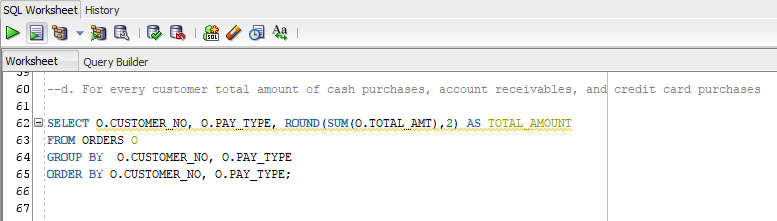
For this query, we have to find the max, min, and average number of wait times. And for that we have used the following SQL queries to generate the desired output.



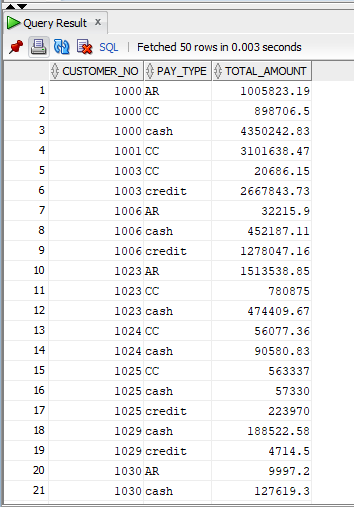
The output above displays the maximum number of wait times, minimum number of wait times, and average number of wait times for every customer.

d. For every customer total amount of cash purchases, account receivables, and credit card purchases

For this query, we had to generate a total amount of cash purchases, account receivables, and credit card purchases for every customer. We have used the following SQL queries to generate the desired output.

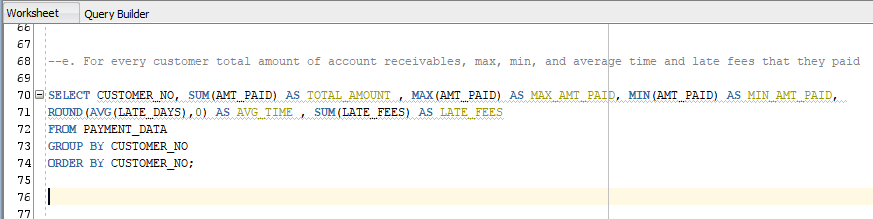


The output below displays the customer number with their payment type and total amount paid for every customer.



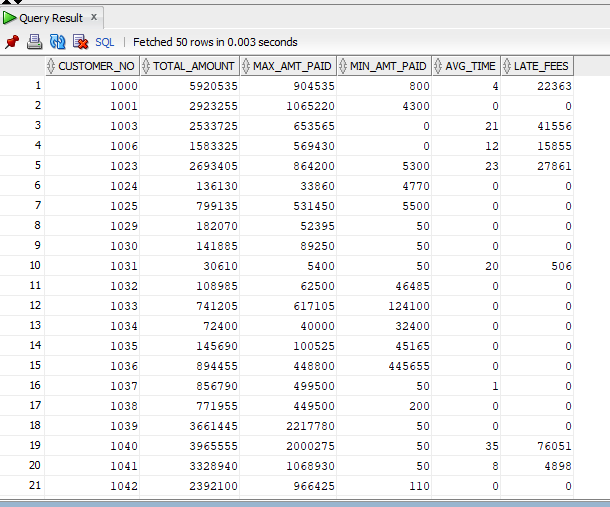
e. For every customer total amount of account receivables, max, min, and average time and late fees that they paid

For this query, we had to generate a total amount of account receivables, max, min, average time, and late fees paid for every customer. We have used the following SQL queries to generate the desired output.



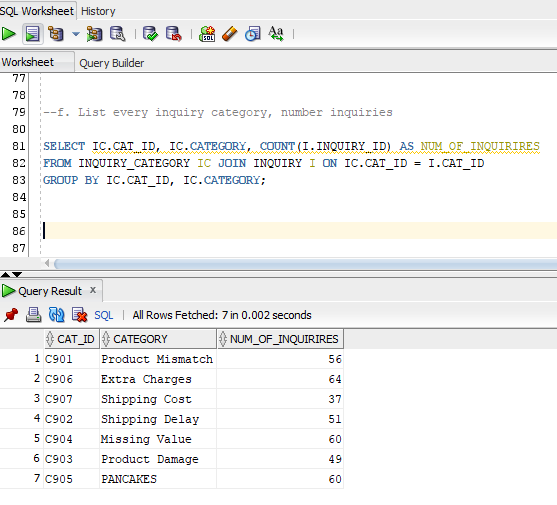
The output below displays the customer number, total amount paid by that particular customer, along with maximum amount of payment done, minimum amount of payment done, average time to pay the amount, and late fees if any.

Output:-



f. List every inquiry category, number inquiries

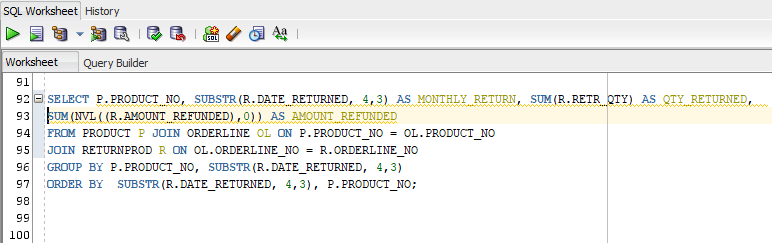
For this query, we had to generate a report listing every inquiry category and number of inquiries . We have used the following SQL queries to generate the desired output.



The output above displays the list of inquiry categories and number of inquiries made for those categories.

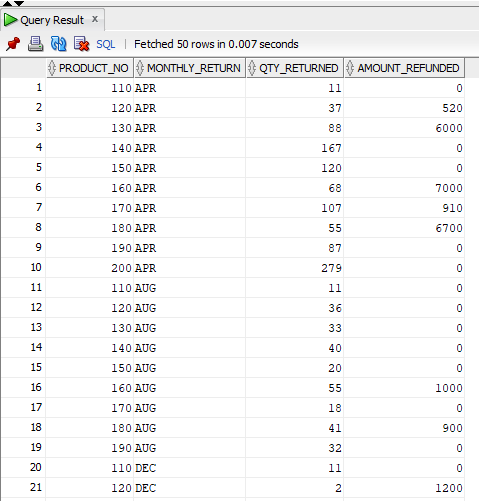
g. List products that must be shipped back to each vendor EVERY MONTH, their qty, amount to be refunded from the vendor in terms of credit voucher (similar to rebate to be used).

For this query, we had to generate a report that listed products which have to be shipped back to the vendor every month with returned quantity, and amount to be refunded. We have used the following SQL queries to generate the desired output.



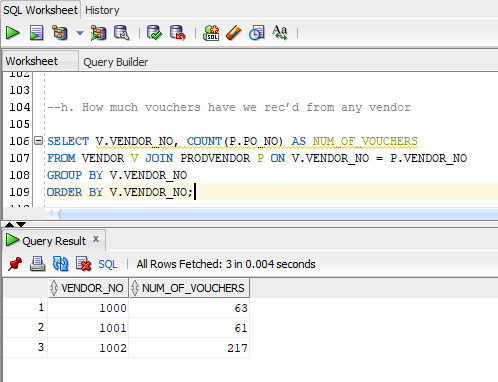
The output below displays the list of products to be shipped back every month along with quantity returned, and amount to be refunded.

Output:-



h. How much vouchers have we rec’d from any vendor

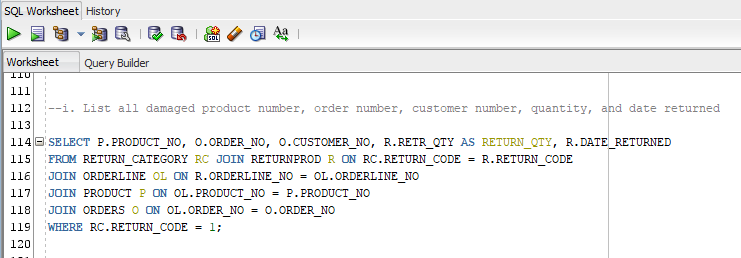
For this query, we had to generate a report that provides the number of vouchers received from any vendor. We have used the following SQL queries to generate the desired output.



The output above displays the list of vendors and number of vouchers received from every vendor.

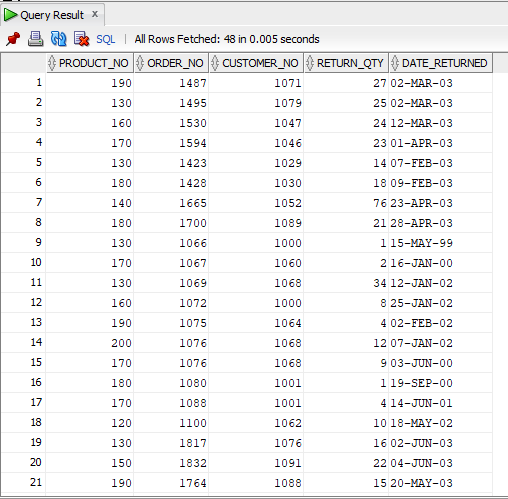
i. List all damaged product number, order number, customer number, quantity, and date returned

For this query, we had to generate a report to list all the damaged product numbers, order numbers, customer number, quantity, and date returned. We have used the following SQL queries to generate the desired output.



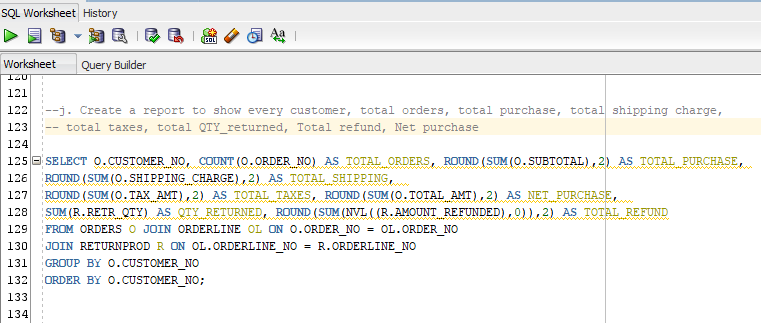
The output below displays the list of product number, order number, customer number, quantity returned, and returned date for all the damaged products.

Output:-



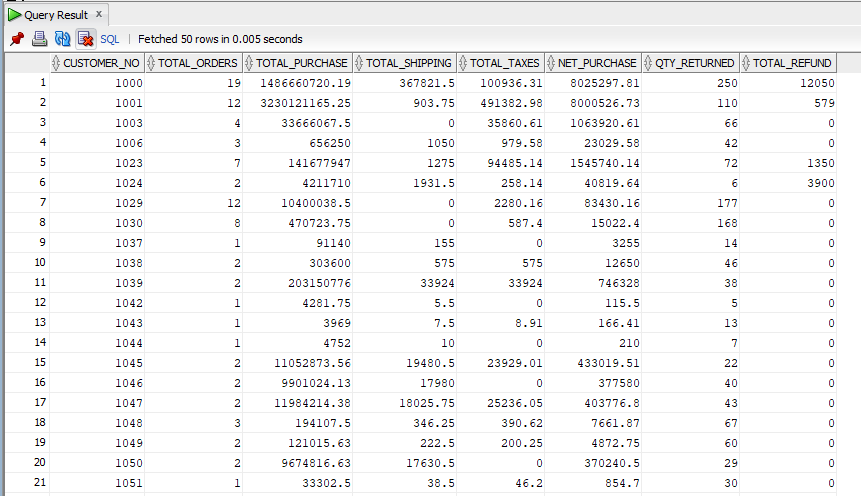
j. Create a report to show every customer, total orders, total purchase, total shipping charge, total taxes, total QTY\_returned, Total refund, Net purchase

For this query, we had to generate a report to show every customer, total orders, total purchase, total shipping charge, total taxes, total quantity returned, total refunded amount, and net purchase. We have used the following SQL queries to generate the desired output.



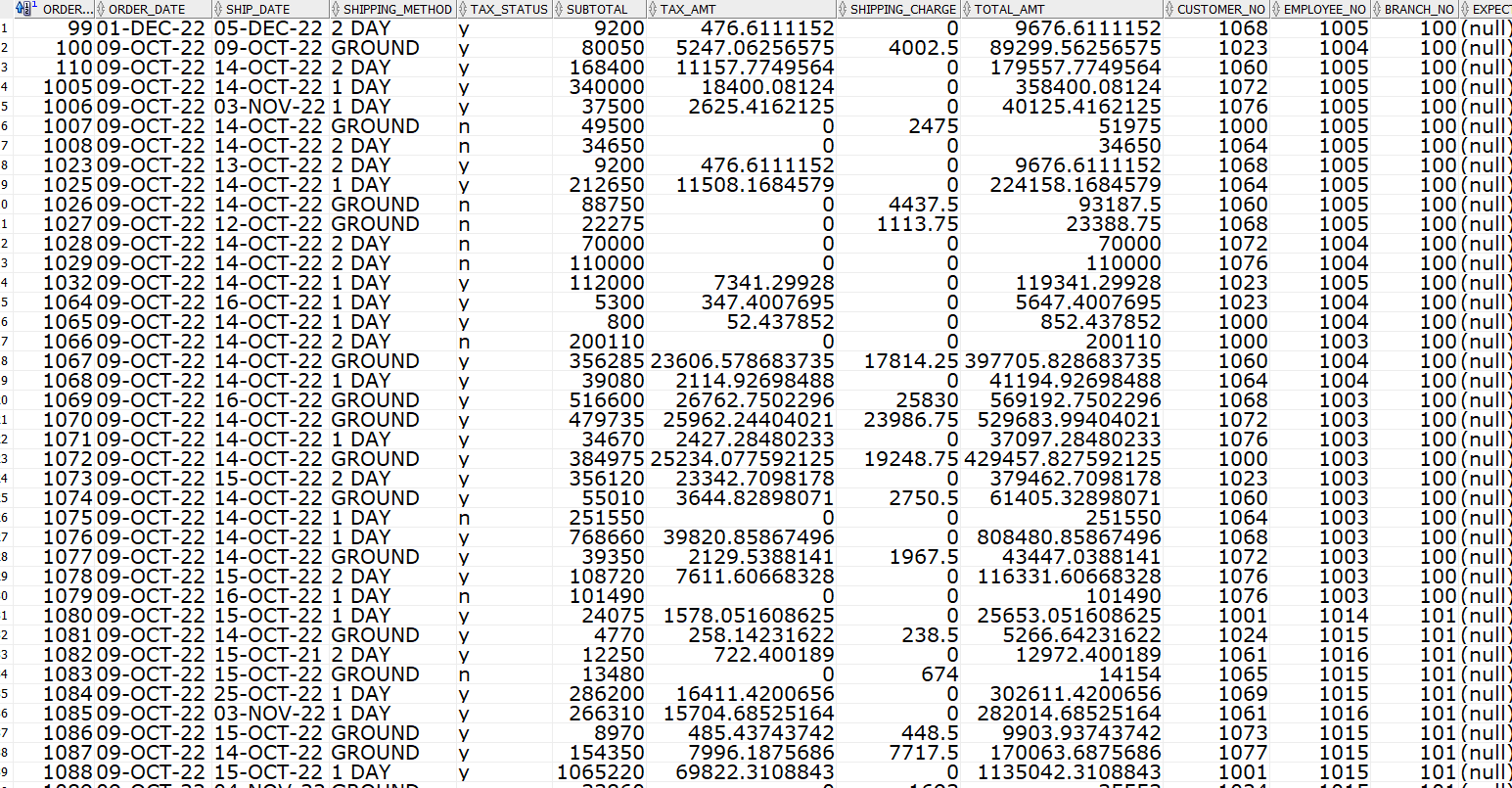
The output below displays a report to show every customer, total orders, total purchase, total shipping charge, total taxes paid, total quantity returned, total refunded amount, and net purchases.

Output:-

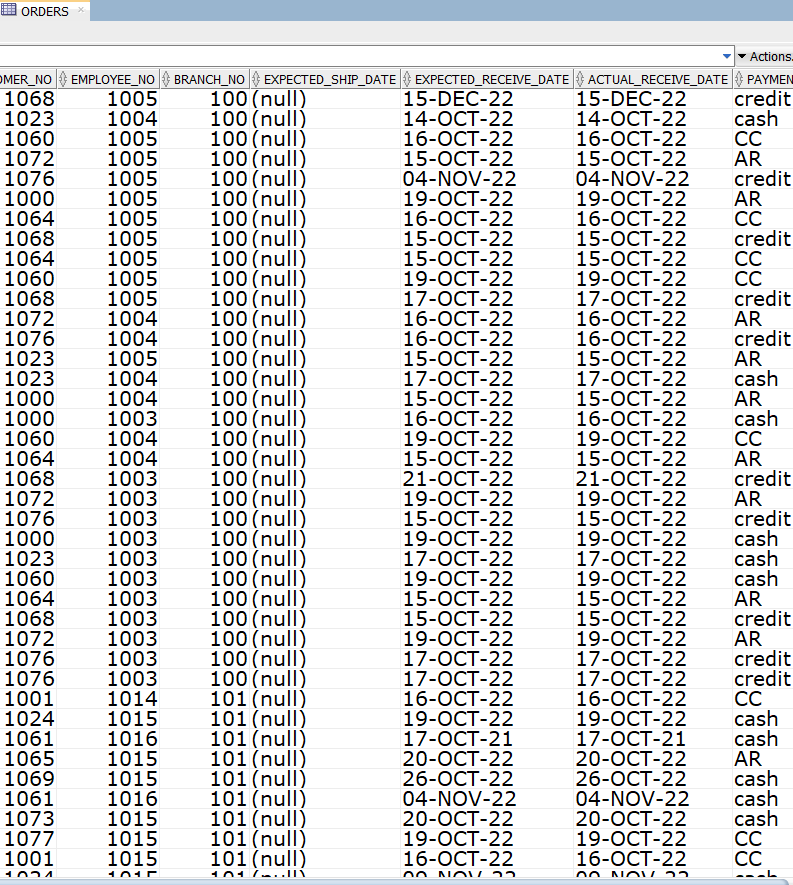


**Use Case #7.3: Reset database to current date**

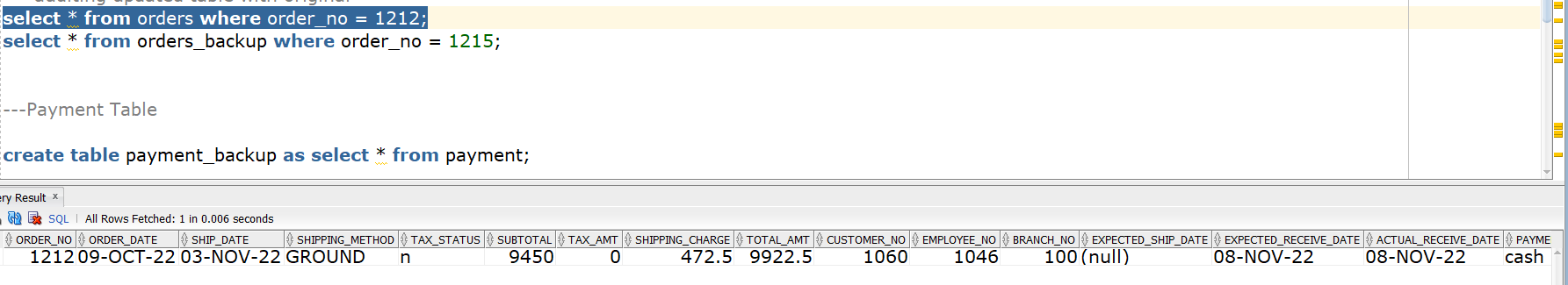
We updated all sales related dates in the database to change them to the current date(recent dates). Below are the screenshots of the updated dates.

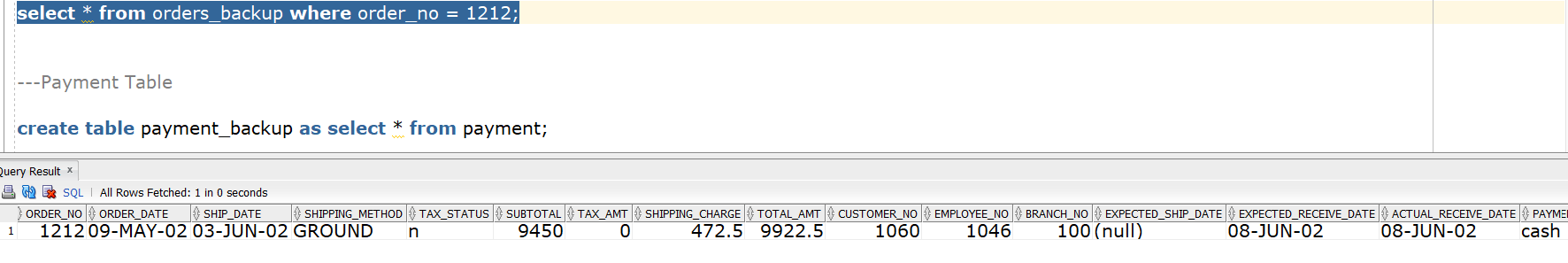
We first updated order table order data and ship date, below is the updated dates screenshot ****

We then updated the expected ship date and expected receive data in the orders table



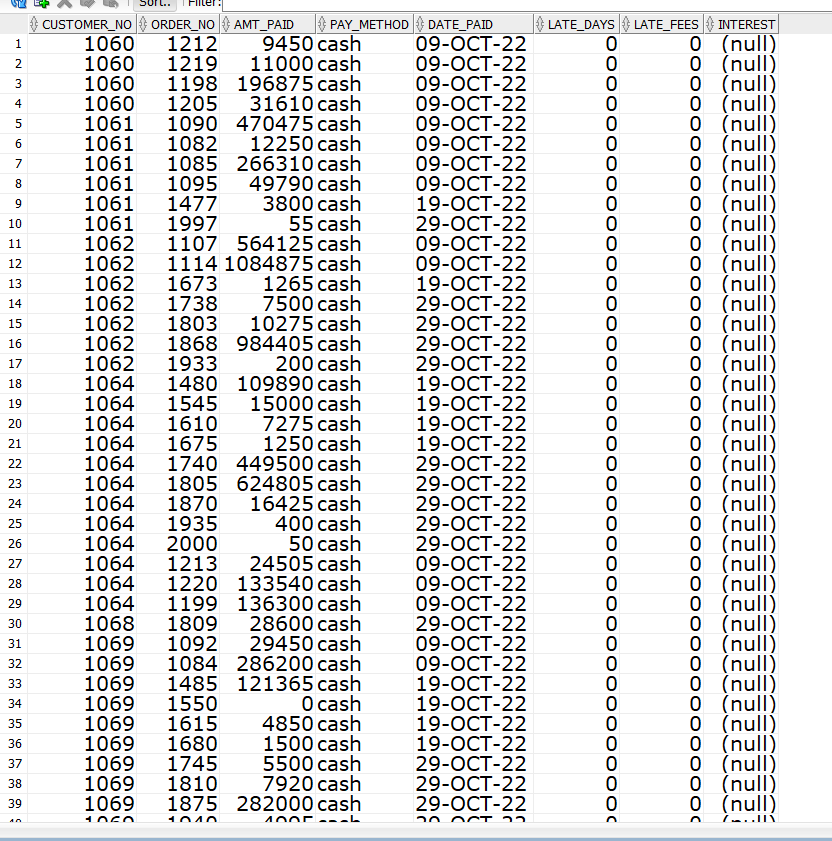
We created audit tables to keep track of all the updates to make sure they are correct by querying the database.



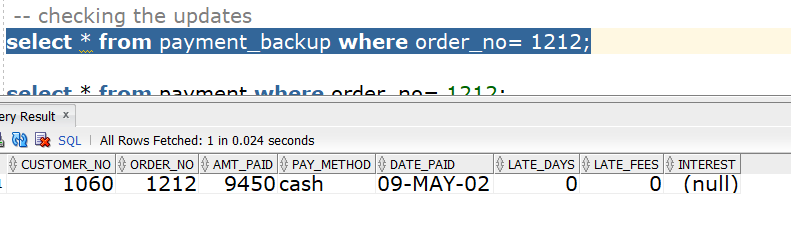


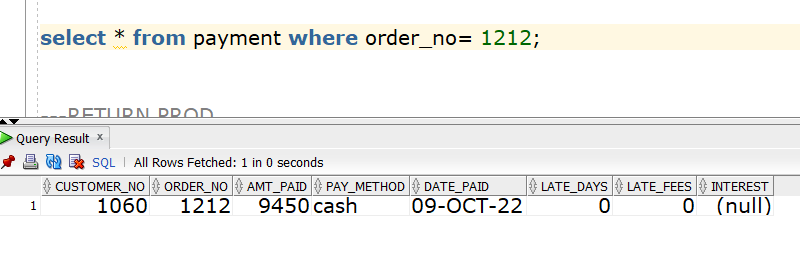
In the above two screenshots we can see that the date is updated to the current date.we maintained the same difference between the order\_date and ship date in the updated table the same way as the difference between expected ship date and actual receive date to make sure it makes sense.

Next we updated dates in the payment table. The screenshot below shows the updated dates in the payment table.



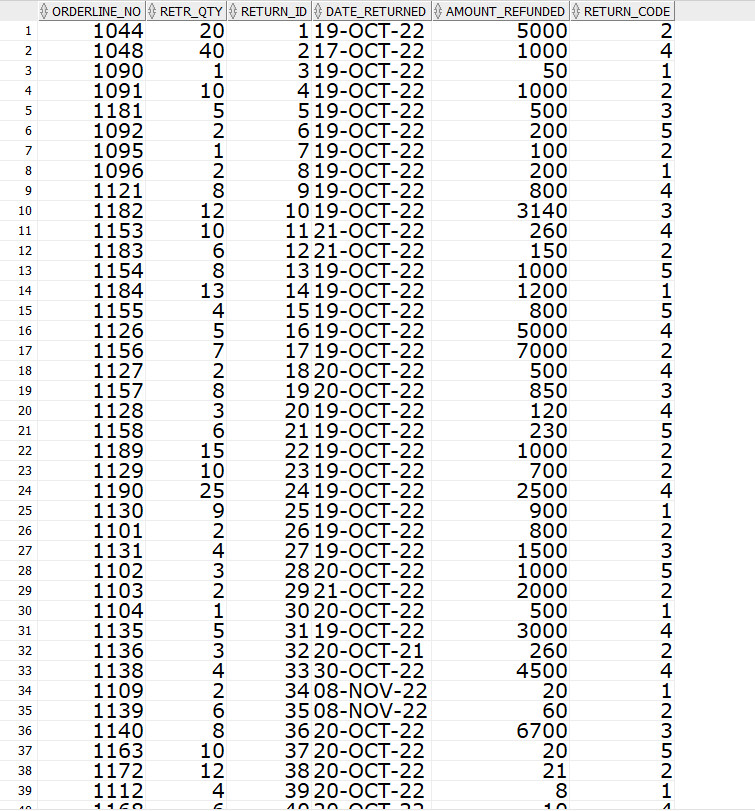
Audit tables to keep track changes of updated dates in payments table



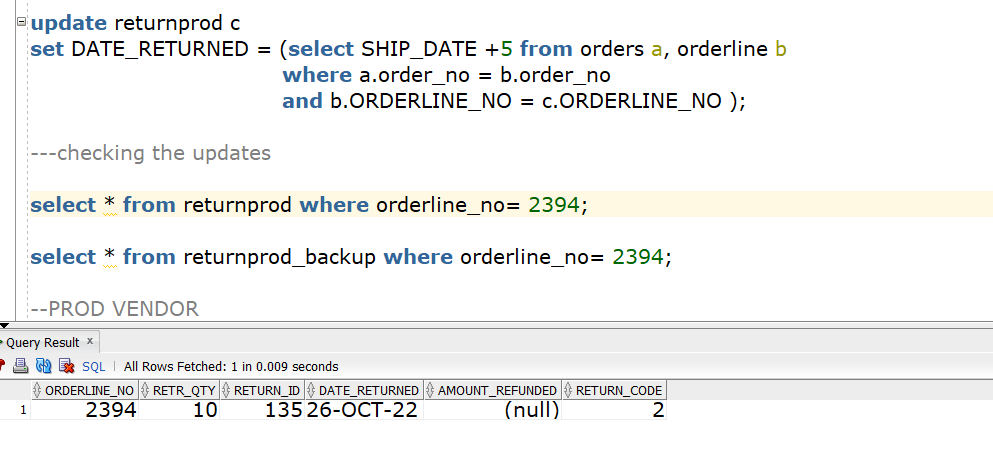


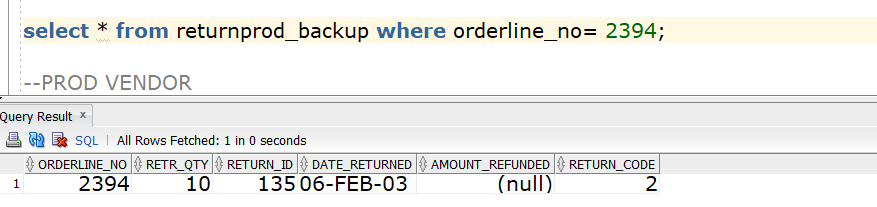
In the above two screenshots we can see that the date is updated .

Next we updated dates in the returnprod table. The screenshot below shows the updated dates.



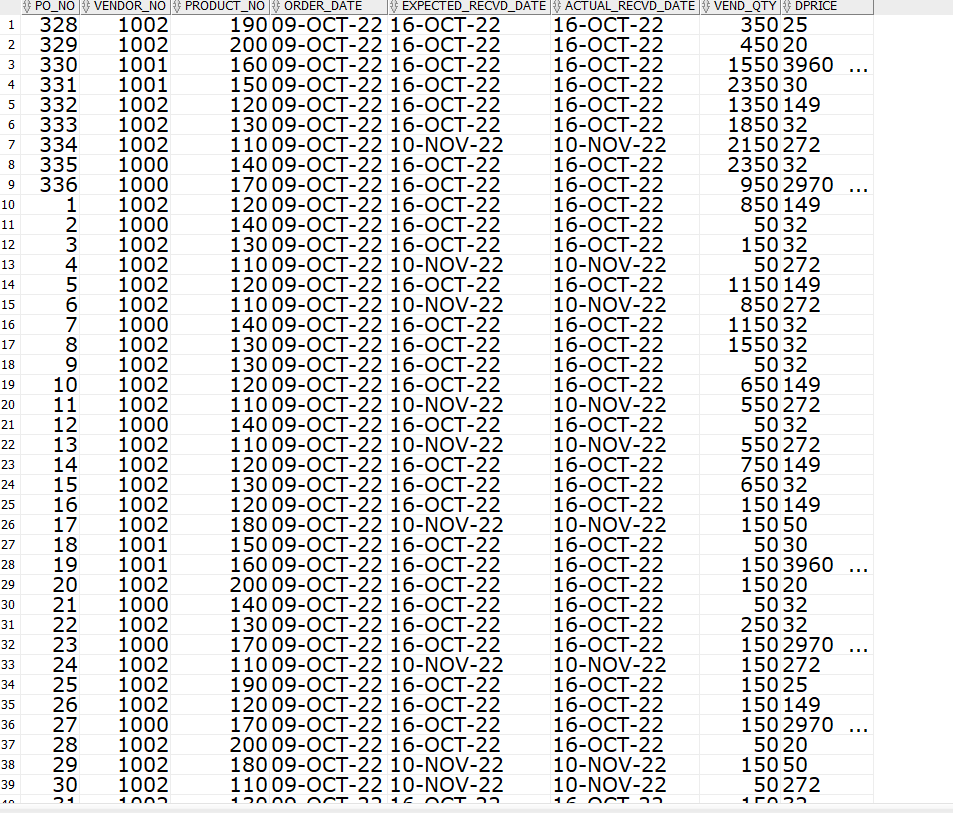
Audit tables to keep track changes of updated dates in payments table



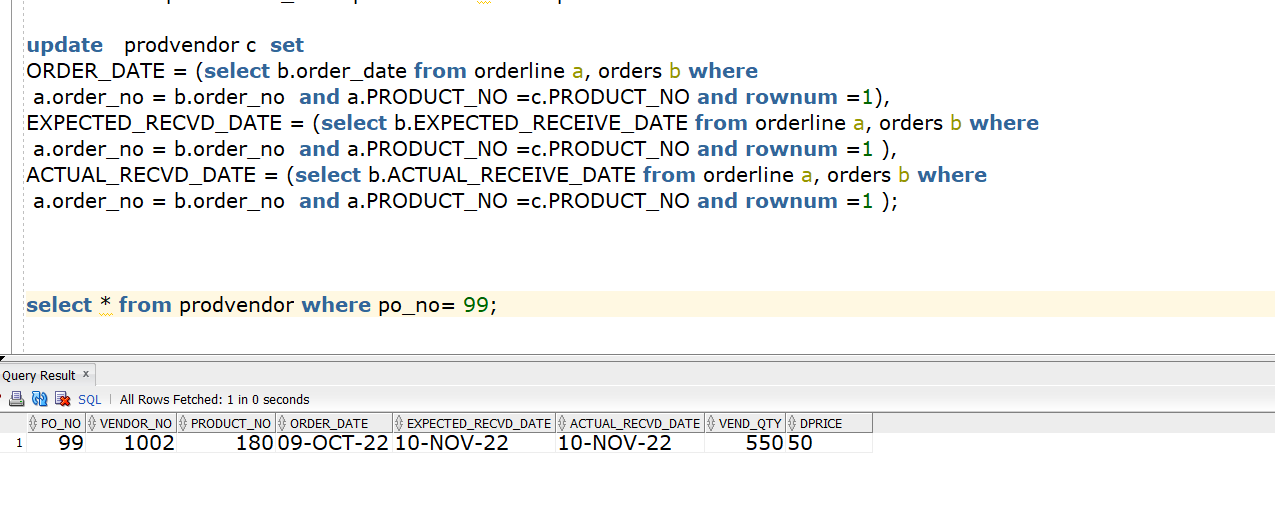


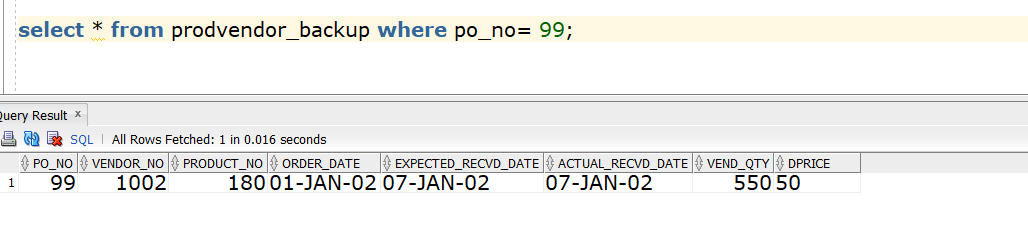
In the above two screenshots we can see that the date is updated .

Next we updated dates in the prod vendor table. The screenshot below shows the updated dates in the prod vendor table.



Audit tables to keep track changes of updated dates in prodvendor table





In the above two screenshots we can see that the date is updated . we updated the order date, expected received date and actual received date from the orders table by joining order and orderline.

We had resetted all sales related dates.

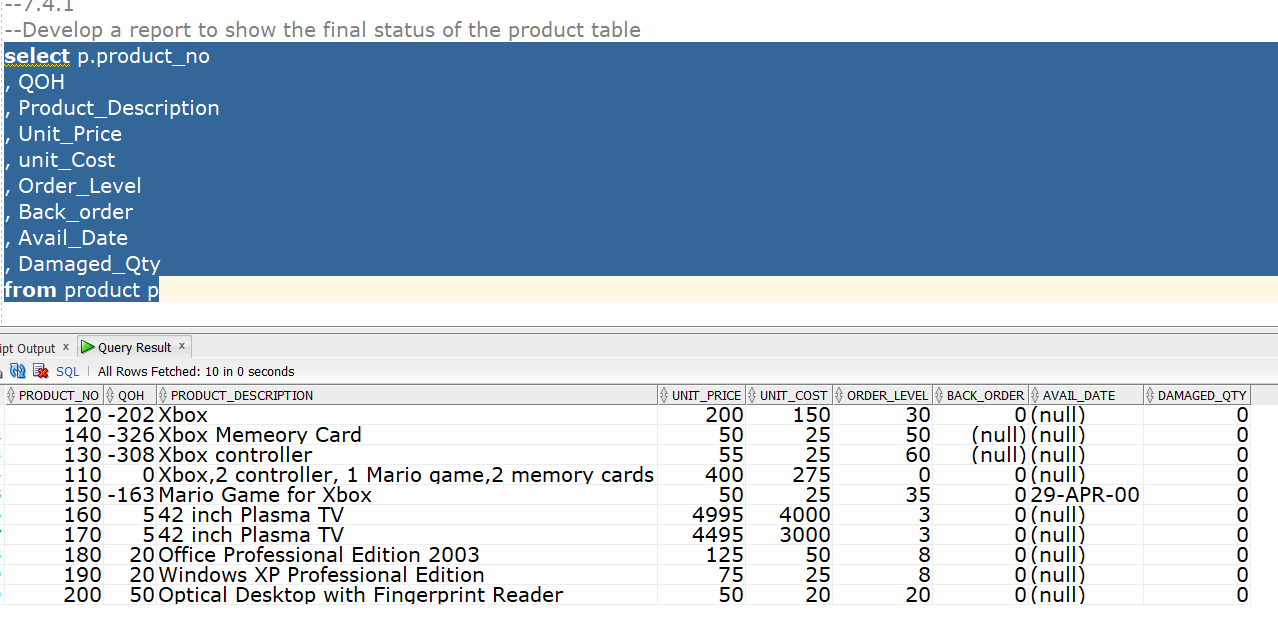
#Develop procedure to reset the database currency to a desired date

We had resetted all sales related dates. In future dates can be resetted to a desired date by using the above update scripts and methods. The script after use case 7.3 is OES5b.

**#USE CASE 7.4 : Implement Intelligent Order Entry System(IOES) based on current dates**

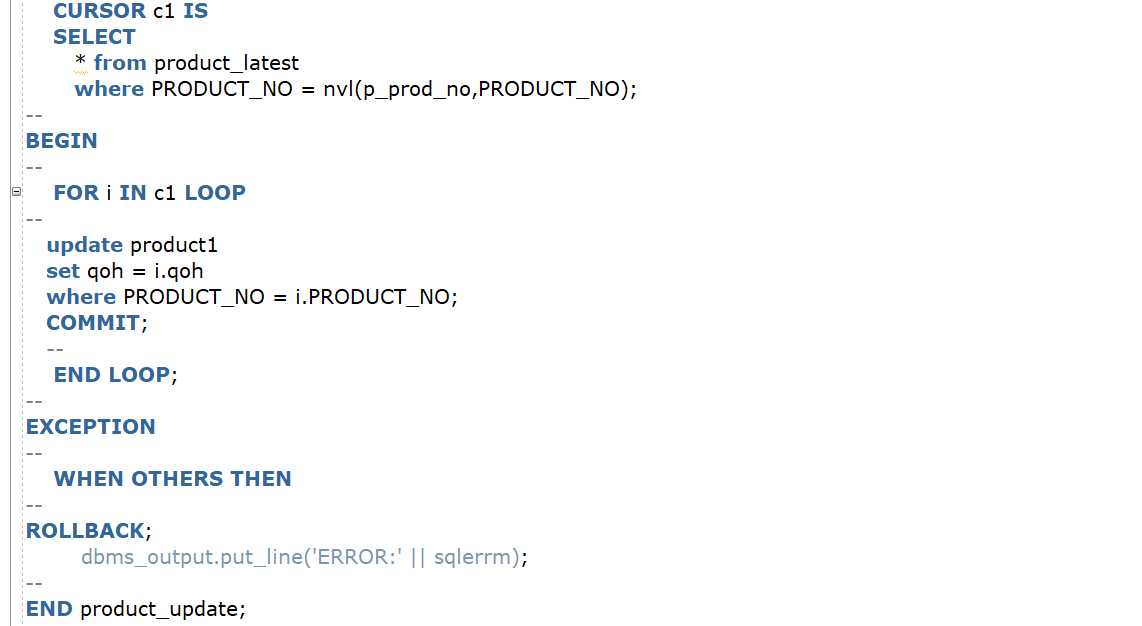
**7.4.1** Developing a report to show the final status of the product table.

We had a developed a report to show all the updates and final status of the product table and renamed the product table to product1

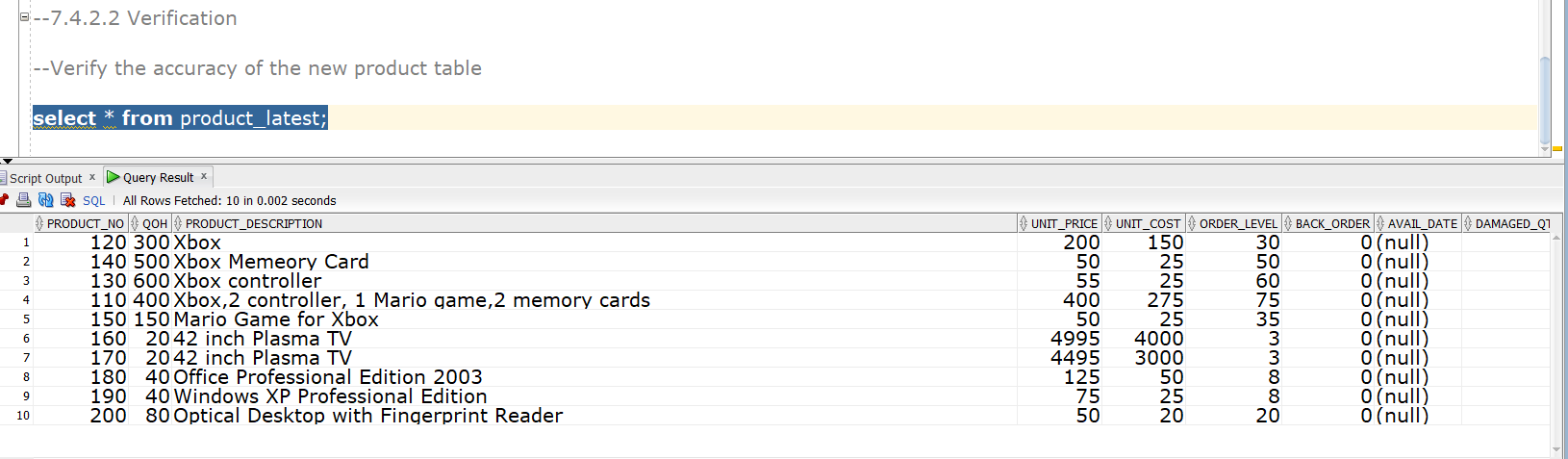


**7.4.2** Develop a procedure to reset to the product table to the given numbers

We developed a procedure to reset the product table to the provided balances in the assignment document.

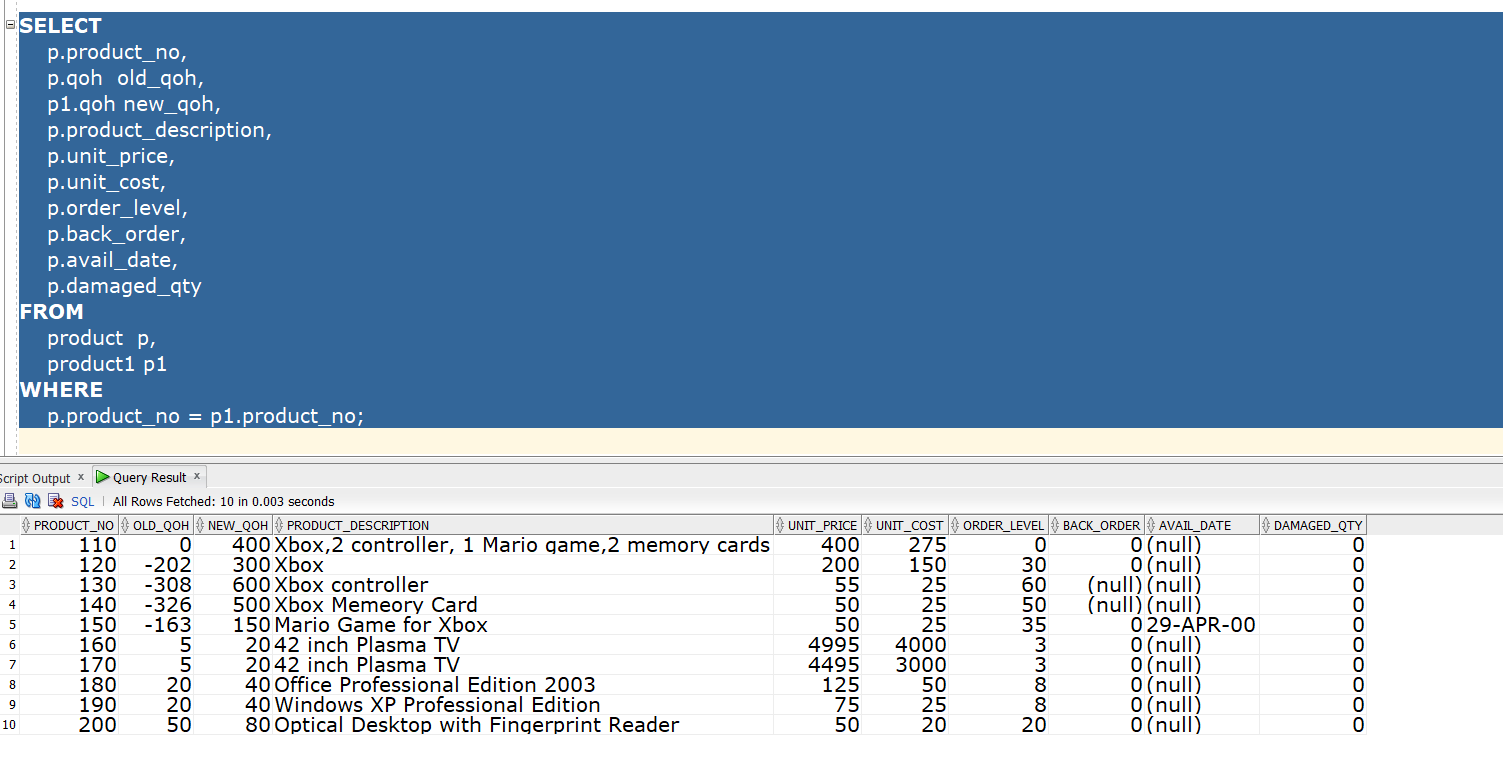


7.4.2 We verified the accuracy of the new product table as shown in the below screenshot



7.4.2

We developed a report to show the difference between product and product1 table



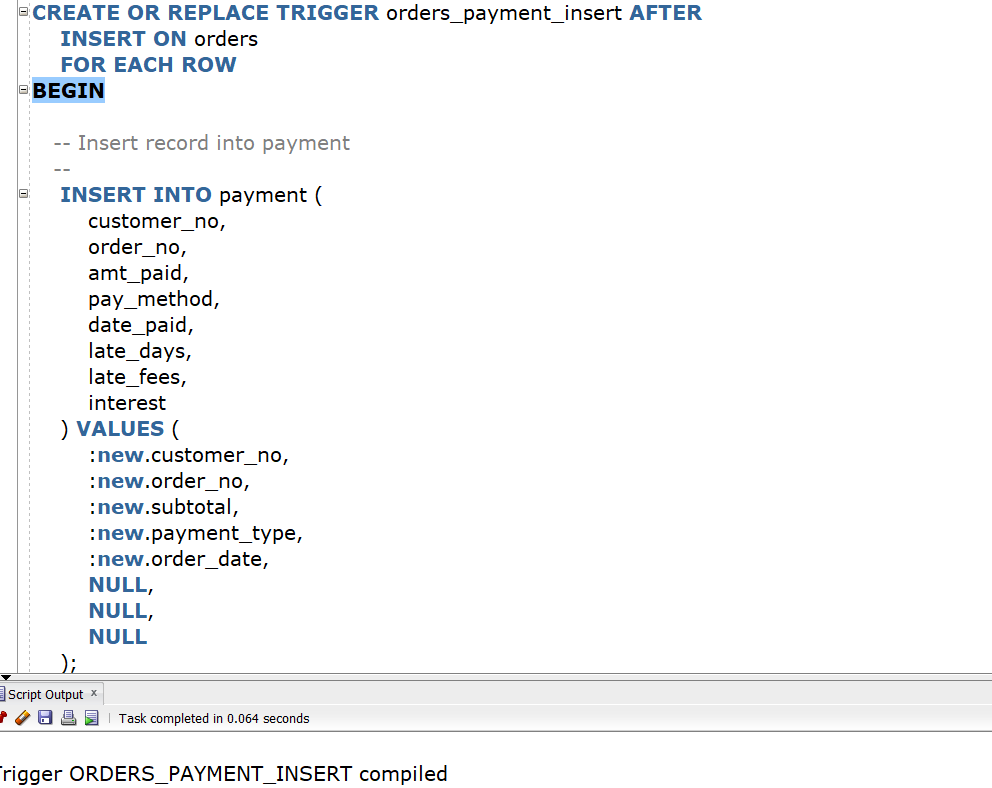
The script after updating the product table is oes 5c

7.4.3

We had written insert statements to update the repeated purchases of any customers by including their Customer\_no, product\_no, qty, shipping method, payment method. This helped us to setup a database to accept sales based on the current date(SYSDATE)

We added the appropriate triggers to convert OES5c to OES6.

We added a trigger which acts upon insert in orders table(basically a new order), the payments table is updated . Below trigger after inserting on the orders table its updated values in the payment table.

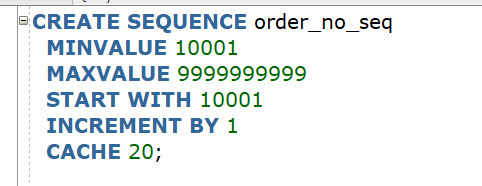


The script is now OES6

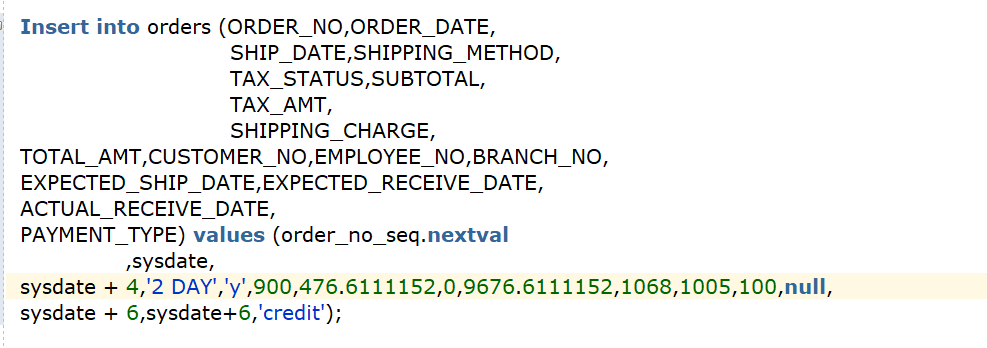
We tested AOES and verified it.

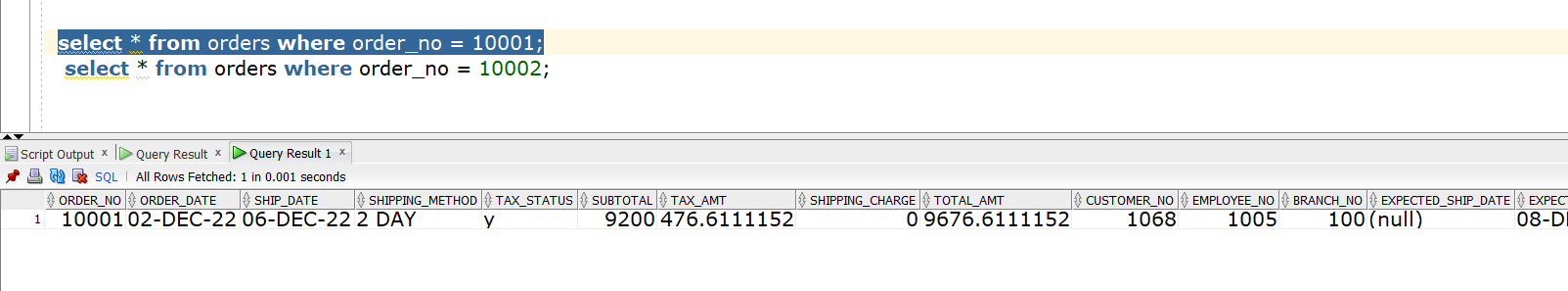
7.4.3

We developed a sequence to auto generate the order\_no and auto generation of Pks. We had given a minimum value of 10001.

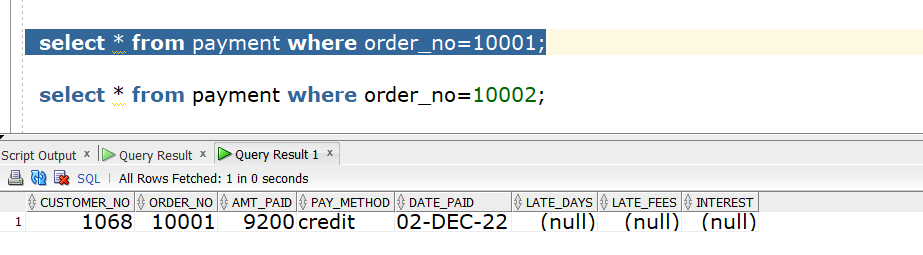


Based on the sequence we verified the orders table

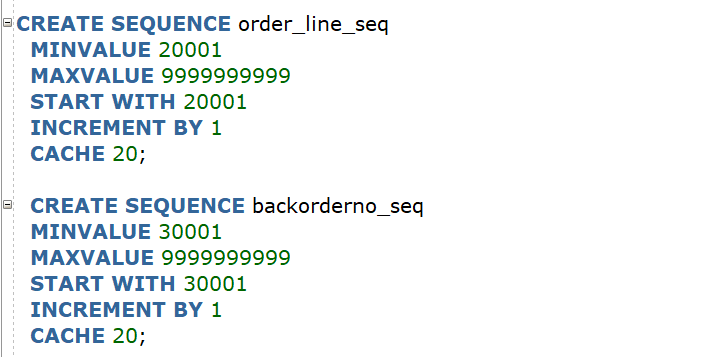




Based on the sequence we also verified the payments table.



We already developed a procedure to accept new orders.

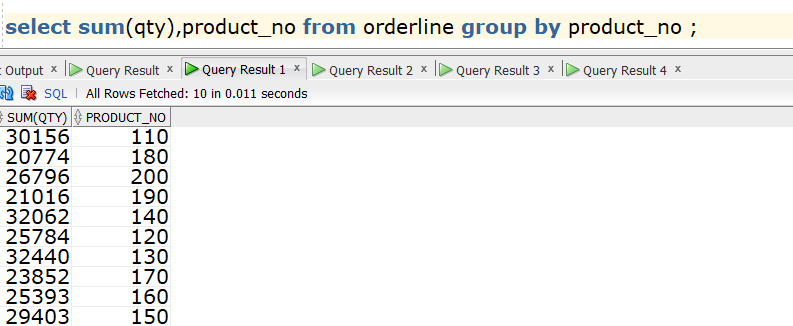


We developed a backorderno\_seq to auto generate the backorder\_no in the back orders table and order\_line seq for orderline\_no.

7.4.4

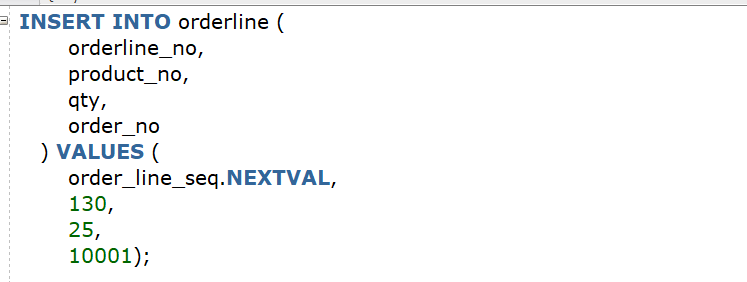
We exported the given data set to excel and then inserted it in orders and products table based on the attributes given in the dataset.

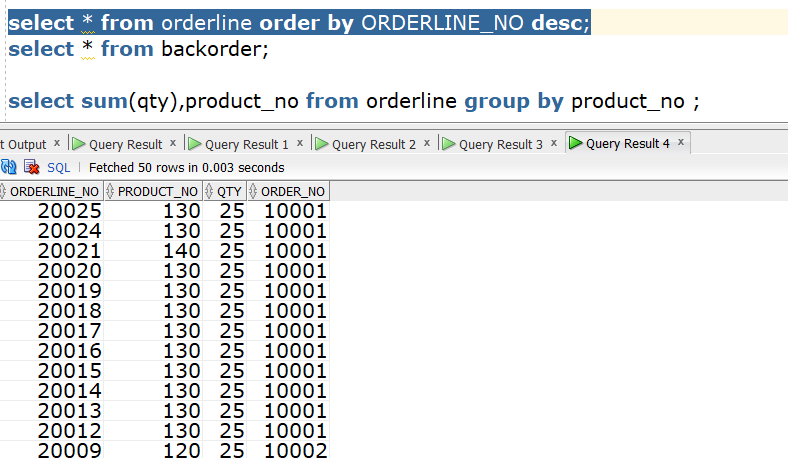
Here we created another trigger that works on the orderline when ever an orderline was created and the if the order quantity is more than the available quantity it creates a back order. We got this idea when we checked the sum of qty for a particular order\_no in the orderline table and surprisingly it was more when compared to the qty available in the product table.

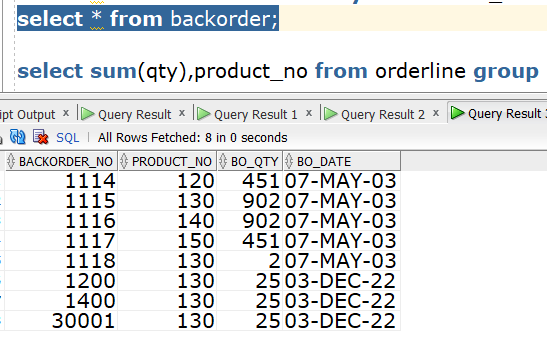




Using the Seq created for Backorder\_no. We auto generated the backorder order\_no and inserterted the values in the trigger when we enter a orderline value. In the below screen shot we inserted the values into the orderline and instead of using orderline\_no we used the seq created for orderlin\_no.







After Inserting the values we checked in the orderline table and backorder table the orders are updated.