//create rating table

Create Table Rating (Rating\_id number,Rat\_date date,Rat\_Comment varchar(1000))

alter table rating

modify rating\_id primary key

//create customer\_rating table

CReate table customer\_rating(Customer\_id number,Rating\_id number,product\_id number)

alter table customer\_rating

add FOREIGN key (customer\_id) REFERENCES customers(customer\_id)

alter table customer\_rating

add FOREIGN key (rating\_id) REFERENCES rating(rating\_id)

alter table customer\_rating

add FOREIGN key (product\_id) REFERENCES products(product\_id)

//create survey table

create table order\_survey (Survey\_id number, customer\_id number,surver\_date date,survey\_method varchar(15),purchase\_id number,delivey\_id number,survey\_comments varchar(100),survey\_rating varchar(2))

alter table order\_survey

modify survey\_id primary key

alter table order\_survey

add FOREIGN key (delivey\_id) REFERENCES delivery(delivery\_id)

alter table order\_survey

add FOREIGN key (customer\_id) REFERENCES customers(customer\_id)

//create tips table

create table tips (tips\_id number,delivery\_id number,tips\_amount number)

alter table tips

modify tips\_id primary key

alter table tips

add FOREIGN key (delivery\_id) REFERENCES delivery(delivery\_id)

//add 5 product ratings and the corresponding values to customer\_rating

insert into rating values (1100,'4-apr-2020','works great i got relief from headache very quickly');

insert into customer\_rating values(101,1100,422)

insert into rating values (1101,'3-feb-2020','tastes great');

insert into customer\_rating values(102,1101,405)

insert into rating values (1102,'18-mar-2020','not the greatest quality but it tastes alright');

insert into customer\_rating values(102,1102,413)

insert into rating values (1103,'20-jan-2020','doesn’t taste good wouldn’t recommend');

insert into customer\_rating values(105,1103,414)

insert into rating values (1104,'5-may-2020','bagel is not soft at all');

insert into customer\_rating values(105,1104,400)

//insert order survey values

Insert into order\_survey values(1200,101,'5-feb-2020','stars','511',null,'very satisfied with the price',5)

Insert into order\_survey values(1201,106,'10-mar-2020','number','512',null,'affordable price for a premium product',10)

Insert into order\_survey values(1202,108,'15-apr-2020','stars',null,612, null,5)

Insert into order\_survey values(1203,112,'18-jan-2020','number’, null,605,null,6)

Insert into order\_survey values(1204,114,'5-may-2020','stars',509,null,'price is a bit too expensive',3)

Insert into order\_survey values(1205,110, '10-apr-2020','stars',null,604,'the driver arrived very late he seemed to not care about the customer at all. His phone number is 5673019988.',1)

//insert 5 tips values

insert into tips values(1300,610,10)

insert into tips values(1301,601,8)

insert into tips values(1302,602,5)

insert into tips values(1303,603,15)

insert into tips values(1304,605,5)

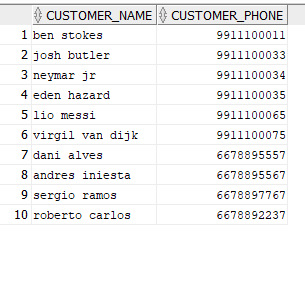
**questions:**

1. Identify customers who have not completed a purchase/delivery survey this year. Display the customer name and phone number.

select customer\_name, customer\_phone

from customers where customer\_id not in

(select customer\_id from order\_survey where survey\_date >='01-jan-2020')



2. Identify the most popular product purchased this year. Display four

columns: warehouse, product name, product type and number of orders. Display

one distinct row for each warehouse, product, and product type. Display the

product with the most orders first.

select warehouse\_location, product\_name,product\_type,count(purchase.product\_id) AS "number of orders"

from warehouse,products\_warehouse,products,purchase

where products.product\_id=purchase.product\_id

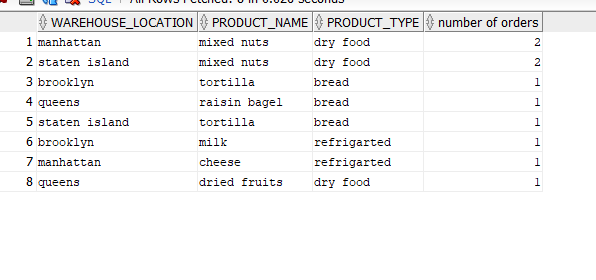
and products\_warehouse.product\_id=purchase.product\_id

and warehouse.warehouse\_id=products\_warehouse.warehouse\_id

and purchase\_date>='01-feb-2020'

group by warehouse. warehouse\_location,products.product\_name,products.product\_type,purchase.product\_id

order by 4 desc;



3. Identify customers with the most purchases of mixed nuts this year by customer

location. Display five rows in your output – one row for each borough. Display

three columns: borough, number of orders, total dollar amount of order. The

borough with the most orders is displayed first.

select customer\_borough,count(purchase.product\_id) as "number of orders",sum(purchase.purchase\_price) as "total spent"

from customers,purchase,products

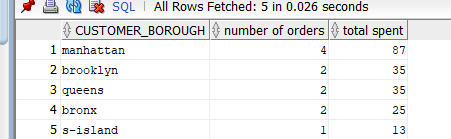
where customers.customer\_id=purchase.customer\_id

and products.product\_id=purchase.product\_id

and purchase\_date>='01-feb-2020'

group by customers.customer\_borough

order by 2 desc



4. Identify customers with no comments in the product survey. Display the

customer name.

select customer\_name

from customers where customer\_id in

(select customer\_id from order\_survey where survey\_comments is null)



5. Search the open-ended narrative text/comments in the product and delivery

comments to identify personally identifiable information (PII). This includes any

data that could potentially be used to identify a person. For instance, examples of

PII include email address, date of birth, Social Security number, bank account

number, home address, and full name. Display the customer who created the

comment, date of comment and the comment. Order the output by customer

name.

select customer\_name,survey\_date,survey\_comments

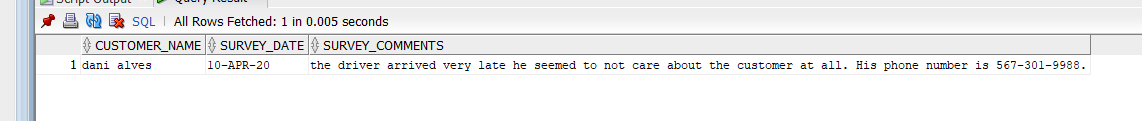
from customers,order\_survey

where customers.customer\_id=order\_survey.customer\_id

and customers.customer\_id in

(select customer\_id from order\_survey

where survey\_comments like '%1-9%')



6. Customers can view, but not change past orders. Create SQL to implement.

Demonstrate your implementation will not edit past orders by attempting to

change data.

//we have user Neymar and. lets’s give him access to see past orders

grant select on purchase TO Neymar

//so he only can see the data but cannot change it

7. Staff are restricted from accessing customer credit card number and expiration

dates. Create SQL to implement. Demonstrate your implementation will prevent

staff from viewing customer credit card data.

//we have staff user for kante. let’s give him privileges, then revoke from customers table to so he can’t access their credit card Info, expiration date

grant all PRIVILEGES TO kante

revoke all PRIVILEGES on customers from kante

8. Staff can’t delete purchases after they are entered in the database. Create SQL to

implement. Demonstrate your implementation will prevent staff from deleting

purchases.

//give kante the privilege to only select insert and update on purchase but not delete

revoke all PRIVILEGES on purchase from kante

grant select on purchase to kante

grant insert on purchase to kante

grant update on purchase to kante

9. The product green is no longer being offered by the grocery store and being

available for 3 years. Identify the SQL to implement.

Delete from products

Where product\_name like 'green tea'

10. In one SQL window, delete all customers. Don’t commit. In another SQL

window, add five new customers. Don’t commit. In each SQL window, identify

the number of customers. Explain your results. Disable the auto commit flag at

the top of the window before performing this operation. Show all SQL to

perform these operations. Demonstrate the functionally of your SQL by

displaying the before and after results.

delete from customer\_copy

where customer\_id>=100

insert into sally.customer\_copy

values(117,'harry kane','h.k',9911103033,'ball','sunnyside','ny',11378);

insert into sally.customer\_copy

values(118,'sadio mane','s.m',9911120022,'bat','woodside','ny',11375);

insert into sally.customer\_copy

values(119,'jose morinho','j.m',8811120022,'bat','woodside','ny',11375);

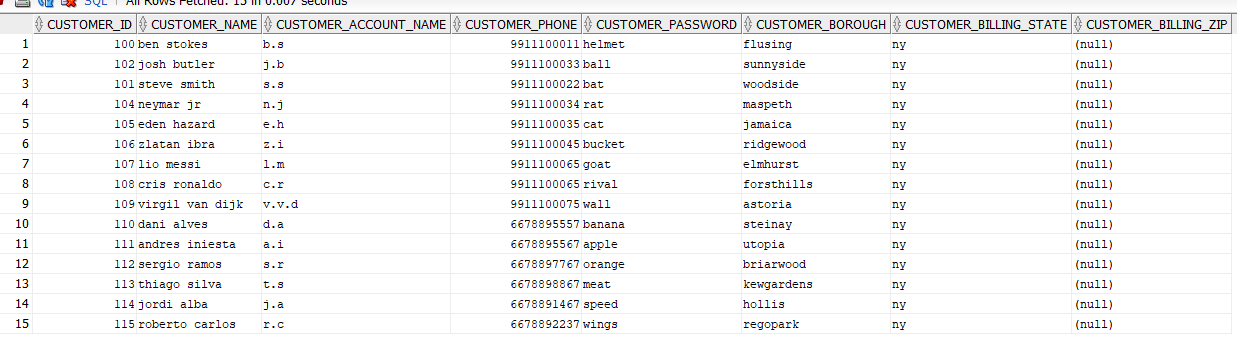
insert into sally.customer\_copy

values(120,'dani carvajal','n.j',9911100034,'rat','maspeth','ny',11372);

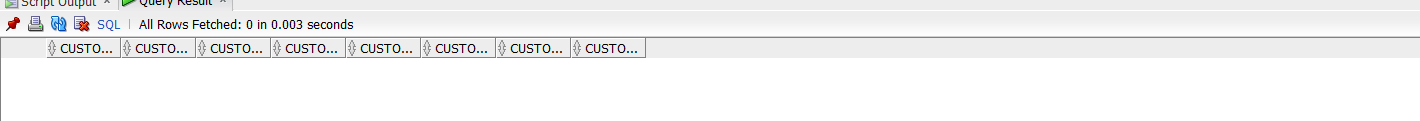
insert into sally.customer\_copy

values(121,'luis','n.j',7911100034,'rat','maspeth','ny',11372);

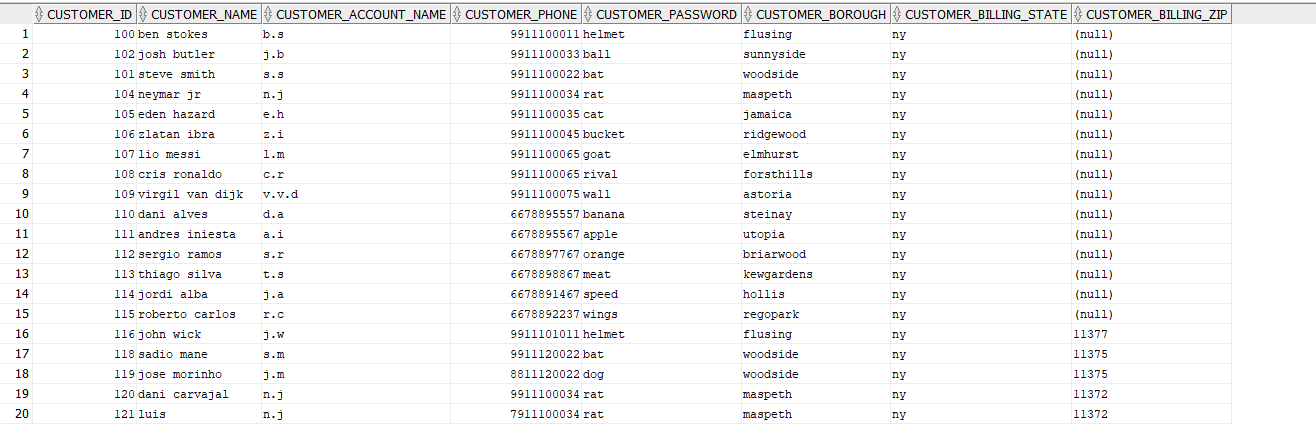
on both window before:



After (the window where I deleted all the customers):



After(the window where I added 5 customers):



So, as we can see we have two different results because we deleted all the customers in the first window, but we did not commit. Therefore, in the second window we have the 15 existing customers plus 5 new ones.

11. In one SQL window, delete all dry products. Don’t commit. In another SQL

window, delete all products Don’t commit. Explain your results. Resolve the

problem. Create a backup of your table before implementing. To create a backup

table, enter CREATE TABLE <NEWTABLE> AS SELECT \* FROM

<ORIGINALTABLE>; COMMIT; Then you can rename a table using the

RENAME TABLE commit. Disable the auto commit flag at the top of the

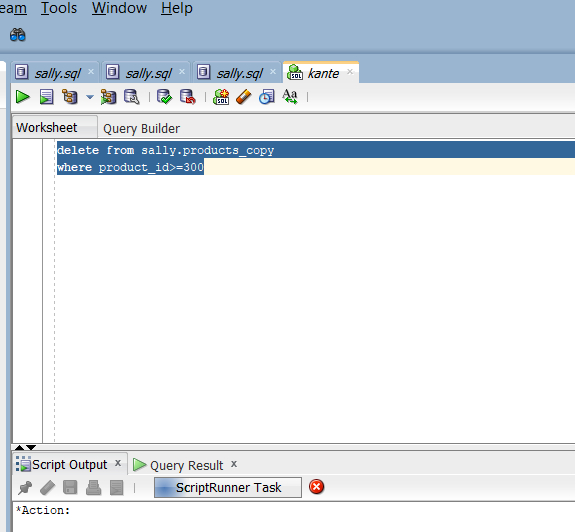
window before performing this operation. Show all SQL to perform these

operations. Demonstrate the functionally of your SQL by displaying the before

and after results.

delete from products\_copy

where product\_type= 'dry food'



So after we perform delete on the first window we basically lock the products\_copy table, therefore when we perform delete on the second window we can see in the picture above that its waiting for first window to release the lock. So, we have to commit or rollback on the first window in order to execute the operation the second window.

12. In one SQL window, change the password for the customer ben stokes. Don’t commit.

In another SQL window, change the last name of customer ben stokes (use the same

name as above). Don’t commit. Quit both Oracle sessions. Login to Oracle again

and display all columns for the customer ben stokes . Explain your results. Disable the

auto commit flag at the top of the windows before performing this operation.

Show all SQL to perform these operations. Demonstrate the functionally of your

SQL by displaying the before and after results.

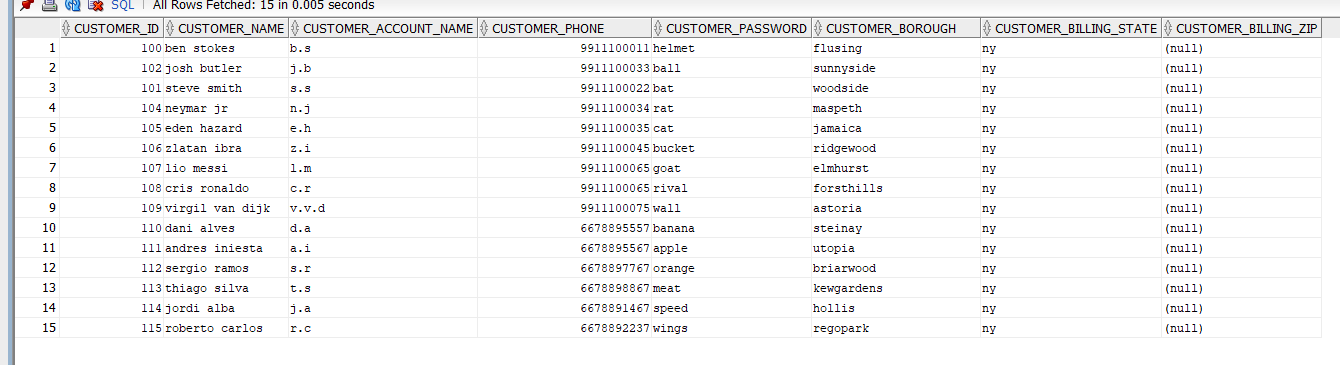
update customers

set customer\_name='ben smith' where customer\_id=100

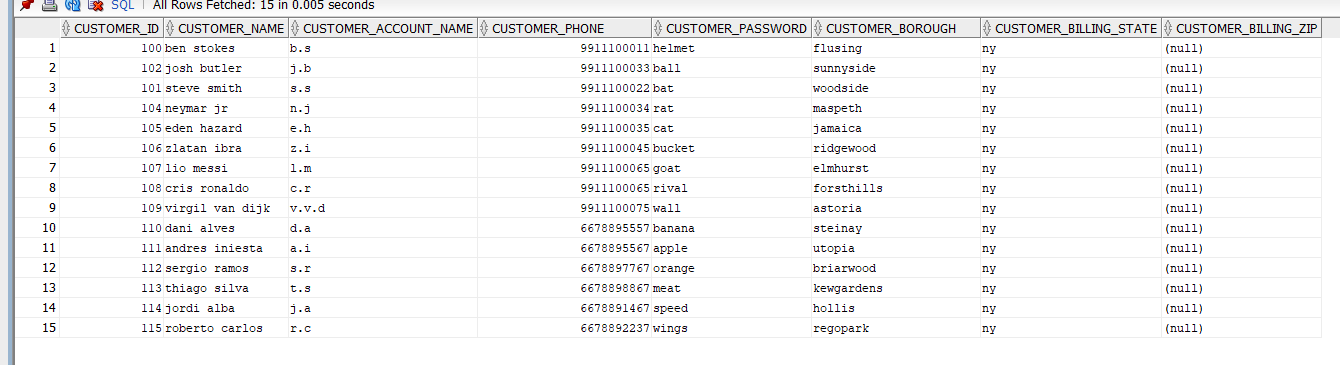
update sally.customers

set customer\_password= 'bucket' where customer\_id=100;

before(on both windows):



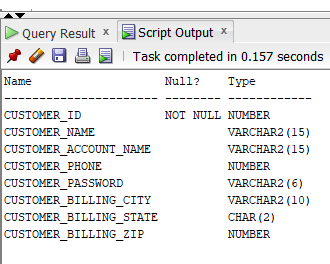
After(on both windows):



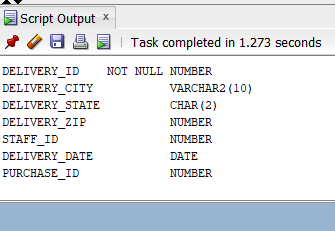
So as we can see there is no change the date after we quit the session and come back because we didn’t commit.

13) Use the SQL DESCRIBE operation to display the structure for all tables.

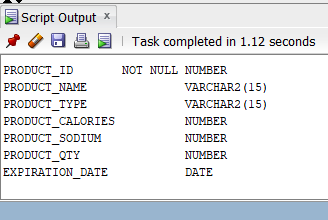
Desc customers



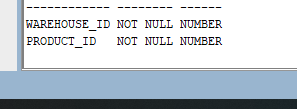
Desc delivery



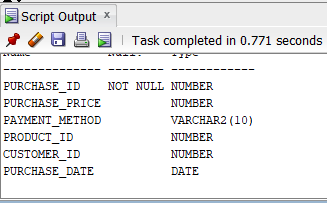
Desc products



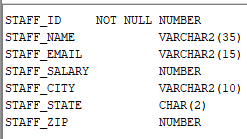
Desc Product\_warehouse



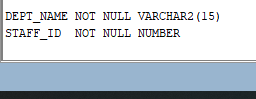
Desc purchase



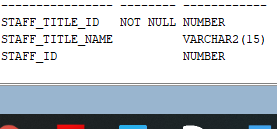
Desc staff



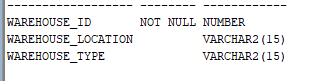
Desc staff\_dept



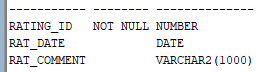
Desc staff\_title



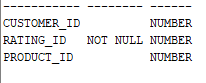
Desc Warehouse



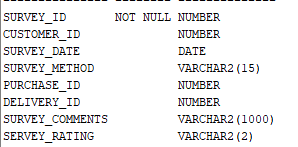
Desc rating



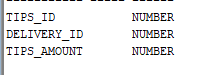
Desc customer\_rating



Desc order\_survey



Desc tips



14. Display the version of Oracle.

SELECT \*

FROM v$version;

