מיני פרוייקט בסיסי נתונים

סופרמרקט

Une image contenant bâtiment, extérieur, route, rue

Description générée automatiquement

מגישים:

אליאור קליין 329843999

אברהם חרר 324215458

מרצה:

אריה ויזן

Step 1)

* 1. Definition of the system
  2. List of entities

a) sales

b) customer

c) supplier

d) products

e) employee

f) diary

* 1. Erd diagram

Une image contenant texte

Description générée automatiquement

* 1. Dsd diagram

Une image contenant capture d’écran, carte

Description générée automatiquement

* 1. Creation of the tables (Sql script)

CREATE TABLE supplier

(

supID INT NOT NULL,

supName VARCHAR(50) NOT NULL,

supPhone INT NOT NULL,

supEmail VARCHAR(50) NOT NULL,

supCompany VARCHAR(30) NOT NULL,

PRIMARY KEY (supID)

);

CREATE TABLE products

(

prodID INT NOT NULL,

numAvailable INT NOT NULL,

price INT NOT NULL,

PRIMARY KEY (prodID)

);

CREATE TABLE diary

(

diarydate DATE NOT NULL,

campaign VARCHAR(50) NOT NULL,

PRIMARY KEY (diarydate)

);

CREATE TABLE customer

(

ID INT NOT NULL,

birthday DATE NOT NULL,

address VARCHAR(50) NOT NULL,

phone INT NOT NULL,

email VARCHAR(50) NOT NULL,

name VARCHAR(50) NOT NULL,

PRIMARY KEY (ID)

);

CREATE TABLE sales

(

saleID INT NOT NULL,

saleDate DATE NOT NULL,

customerID INT NOT NULL,

EmployeeID INT NOT NULL,

PRIMARY KEY (saleID)

);

CREATE TABLE employee

(

empID INT NOT NULL,

empName VARCHAR(50) NOT NULL,

salary INT NOT NULL,

empRating INT NOT NULL,

joinDate DATE NOT NULL,

PRIMARY KEY (empID)

);

CREATE TABLE supply

(

supplyAmount INT NOT NULL,

suplyDate DATE NOT NULL,

orderPrice INT NOT NULL,

supID INT NOT NULL,

prodID INT NOT NULL,

PRIMARY KEY (supID, prodID),

FOREIGN KEY (supID) REFERENCES supplier(supID),

FOREIGN KEY (prodID) REFERENCES products(prodID)

);

CREATE TABLE purchase

(

purchaseAmount INT NOT NULL,

saleID INT NOT NULL,

prodID INT NOT NULL,

PRIMARY KEY (saleID, prodID),

FOREIGN KEY (saleID) REFERENCES sales(saleID),

FOREIGN KEY (prodID) REFERENCES products(prodID)

);

CREATE TABLE diaryUpdate

(

empID INT NOT NULL,

Updatedate DATE NOT NULL,

PRIMARY KEY (empID, Updatedate),

FOREIGN KEY (empID) REFERENCES employee(empID),

FOREIGN KEY (Updatedate) REFERENCES diary(diarydate)

);CREATE TABLE supplier

(

supID INT NOT NULL,

supName VARCHAR(50) NOT NULL,

supPhone INT NOT NULL,

supEmail VARCHAR(50) NOT NULL,

supCompany VARCHAR(30) NOT NULL,

PRIMARY KEY (supID)

);

CREATE TABLE products

(

prodID INT NOT NULL,

numAvailable INT NOT NULL,

price INT NOT NULL,

PRIMARY KEY (prodID)

);

CREATE TABLE diary

(

diarydate DATE NOT NULL,

campaign VARCHAR(50) NOT NULL,

PRIMARY KEY (diarydate)

);

CREATE TABLE customer

(

ID INT NOT NULL,

birthday DATE NOT NULL,

address VARCHAR(50) NOT NULL,

phone INT NOT NULL,

email VARCHAR(50) NOT NULL,

name VARCHAR(50) NOT NULL,

PRIMARY KEY (ID)

);

CREATE TABLE sales

(

saleID INT NOT NULL,

saleDate DATE NOT NULL,

customerID INT NOT NULL,

EmployeeID INT NOT NULL,

PRIMARY KEY (saleID)

);

CREATE TABLE employee

(

empID INT NOT NULL,

empName VARCHAR(50) NOT NULL,

salary INT NOT NULL,

empRating INT NOT NULL,

joinDate DATE NOT NULL,

PRIMARY KEY (empID)

);

CREATE TABLE supply

(

supplyAmount INT NOT NULL,

suplyDate DATE NOT NULL,

orderPrice INT NOT NULL,

supID INT NOT NULL,

prodID INT NOT NULL,

PRIMARY KEY (supID, prodID),

FOREIGN KEY (supID) REFERENCES supplier(supID),

FOREIGN KEY (prodID) REFERENCES products(prodID)

);

CREATE TABLE purchase

(

purchaseAmount INT NOT NULL,

saleID INT NOT NULL,

prodID INT NOT NULL,

PRIMARY KEY (saleID, prodID),

FOREIGN KEY (saleID) REFERENCES sales(saleID),

FOREIGN KEY (prodID) REFERENCES products(prodID)

);

CREATE TABLE diaryUpdate

(

empID INT NOT NULL,

Updatedate DATE NOT NULL,

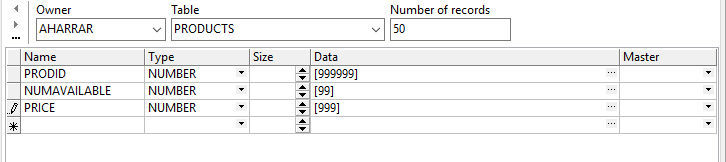
PRIMARY KEY (empID, Updatedate),

FOREIGN KEY (empID) REFERENCES employee(empID),

FOREIGN KEY (Updatedate) REFERENCES diary(diarydate)

);

* 1. Insert example and data generation



insert into AHARRAR.PRODUCTS (PRODID, NUMAVAILABLE, PRICE)

values (768702, 76, 486);

insert into AHARRAR.PRODUCTS (PRODID, NUMAVAILABLE, PRICE)

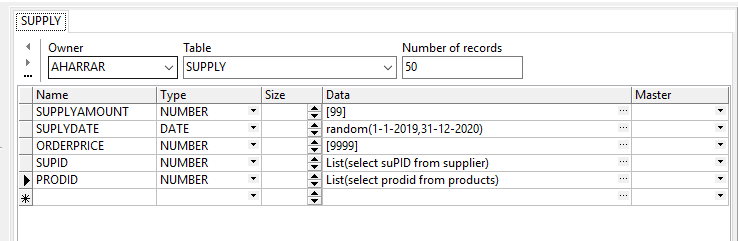
values (215535, 14, 911);

insert into AHARRAR.PRODUCTS (PRODID, NUMAVAILABLE, PRICE)

values (787905, 71, 529);

insert into AHARRAR.PRODUCTS (PRODID, NUMAVAILABLE, PRICE)

values (017244, 52, 584);……………..

insert into AHARRAR.SUPPLY (SUPPLYAMOUNT, SUPLYDATE, ORDERPRICE, SUPID, PRODID)

values (52, to\_date('18-07-2019', 'dd-mm-yyyy'), 7530, 688418, 236958);

insert into AHARRAR.SUPPLY (SUPPLYAMOUNT, SUPLYDATE, ORDERPRICE, SUPID, PRODID)

values (21, to\_date('31-12-2019', 'dd-mm-yyyy'), 7669, 498358, 575360);

insert into AHARRAR.SUPPLY (SUPPLYAMOUNT, SUPLYDATE, ORDERPRICE, SUPID, PRODID)

values (49, to\_date('25-06-2019', 'dd-mm-yyyy'), 5999, 625327, 674490);

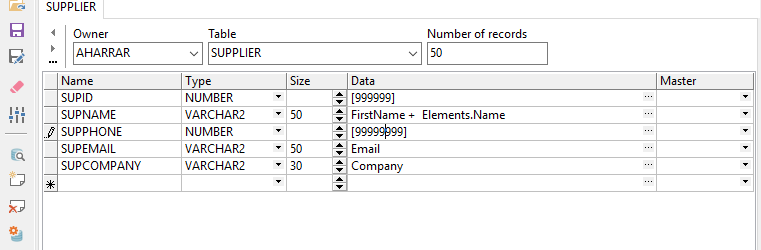
insert into AHARRAR.SUPPLY (SUPPLYAMOUNT, SUPLYDATE, ORDERPRICE, SUPID, PRODID)

values (92, to\_date('05-01-2019', 'dd-mm-yyyy'), 8055, 721069, 749560);

insert into AHARRAR.SUPPLY (SUPPLYAMOUNT, SUPLYDATE, ORDERPRICE, SUPID, PRODID)

values (75, to\_date('24-01-2020', 'dd-mm-yyyy'), 9415, 863737, 792739);

………….



insert into AHARRAR.SUPPLIER (SUPID, SUPNAME, SUPPHONE, SUPEMAIL, SUPCOMPANY)

values (213485, 'Carlerbium', 97496278, 'carl.diesel@marathonheater.ca', 'Marathon Heater');

insert into AHARRAR.SUPPLIER (SUPID, SUPNAME, SUPPHONE, SUPEMAIL, SUPCOMPANY)

values (001216, 'Judethallium', 66696870, 'jude.gaynor@pulaskifinancial.com', 'Pulaski Financial');

insert into AHARRAR.SUPPLIER (SUPID, SUPNAME, SUPPHONE, SUPEMAIL, SUPCOMPANY)

values (346311, 'Ellepraseodymium', 79402002, 'elle@timberlanewoodcrafters.com', 'Timberlane Woodcrafters');

insert into AHARRAR.SUPPLIER (SUPID, SUPNAME, SUPPHONE, SUPEMAIL, SUPCOMPANY)

values (634497, 'Simonberkelium', 29505771, 'simon.haysbert@pioneermortgage.com', 'Pioneer Mortgage');

insert into AHARRAR.SUPPLIER (SUPID, SUPNAME, SUPPHONE, SUPEMAIL, SUPCOMPANY)

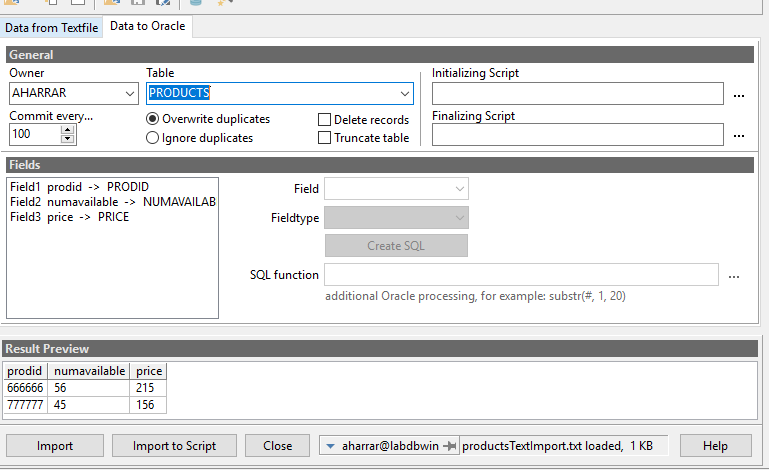
values (324553, 'Vinneon', 29239261, 'v.borgnine@trx.com', 'TRX');…………….

Text importer:

prodid, numavailable, price

666666, 56, 215

777777, 45, 156



Step 2)

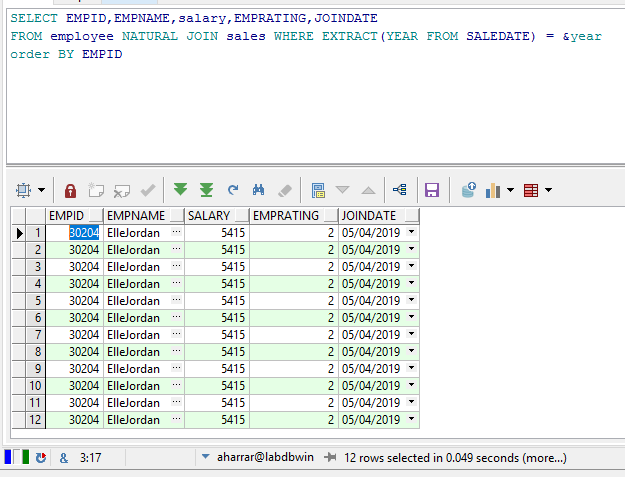
2.1) Sql queries

Return every employee who work at the variable “year”:

SELECT EMPID,EMPNAME,salary,EMPRATING,JOINDATE

FROM employee NATURAL JOIN sales WHERE EXTRACT(YEAR FROM SALEDATE) = &year

order BY EMPID



The query returns every products that doesn’t left much(numAvailable<10) , products that we need to make an order and their price(calculated by the price of the order divided by the amount):

SELECT p.prodid,p.NUMAVAILABLE,

SUM(s.ORDERPRICE)/SUM(s.SUPPLYAMOUNT) AS avarage\_price\_for\_unit

FROM products p, supply s

WHERE p.prodid = s.prodid AND p.NUMAVAILABLE< 10

GROUP BY p.prodid,p.NUMAVAILABLE

ORDER BY p.prodid

Une image contenant capture d’écran

Description générée automatiquement

Query that return the price of a sales during a date with a special event:

select SALEID, sum(price)

from purchase natural join products natural join sales

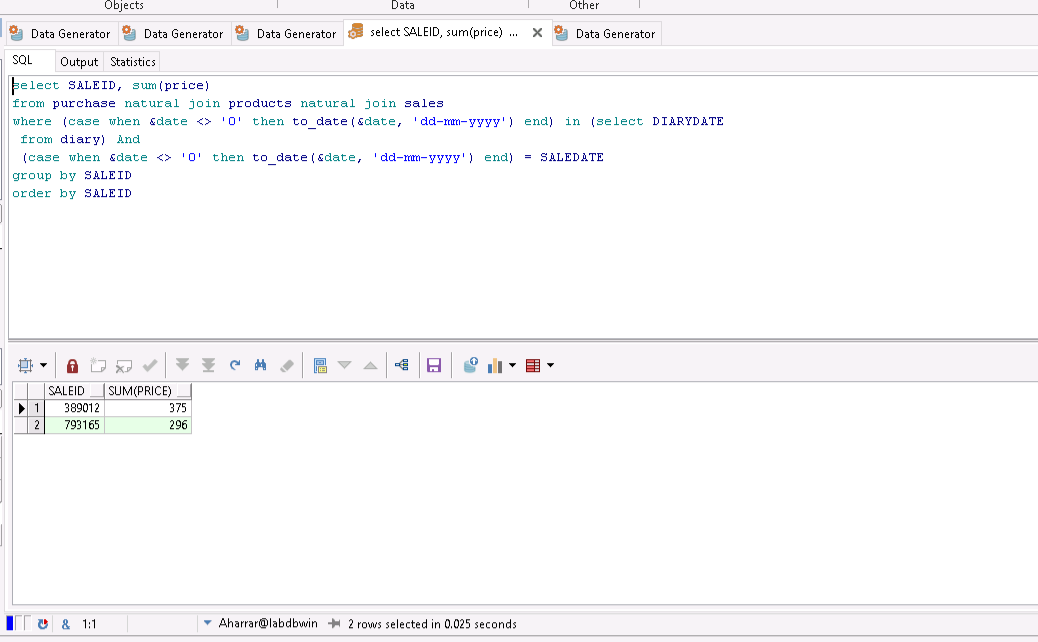
where (case when &date <> '0' then to\_date(&date, 'dd-mm-yyyy') end) in (select DIARYDATE

from diary) And

(case when &date <> '0' then to\_date(&date, 'dd-mm-yyyy') end) = SALEDATE

group by SALEID

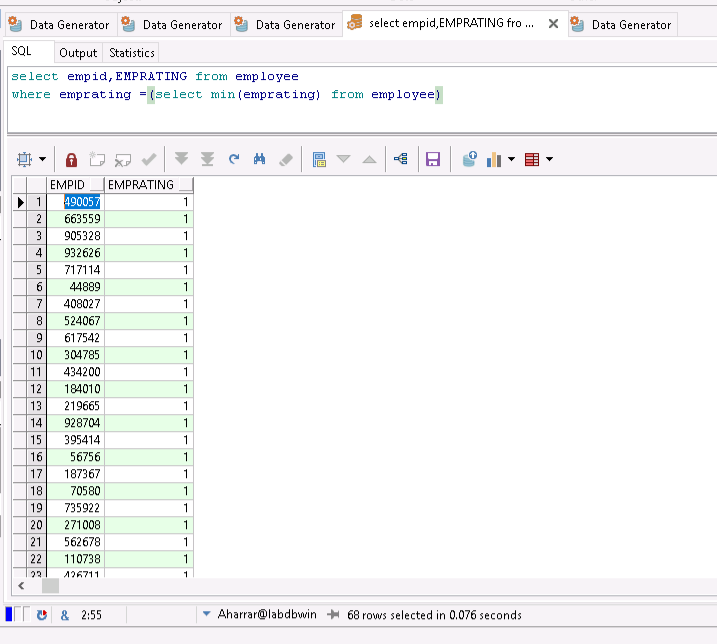
order by SALEID



That query return the employees with the lowest rating:

select empid,EMPRATING from employee

where emprating =(select min(emprating) from employee)



Return supply orders over a month(s) (in this case, from 2019-11-01 to 2020-01-01):

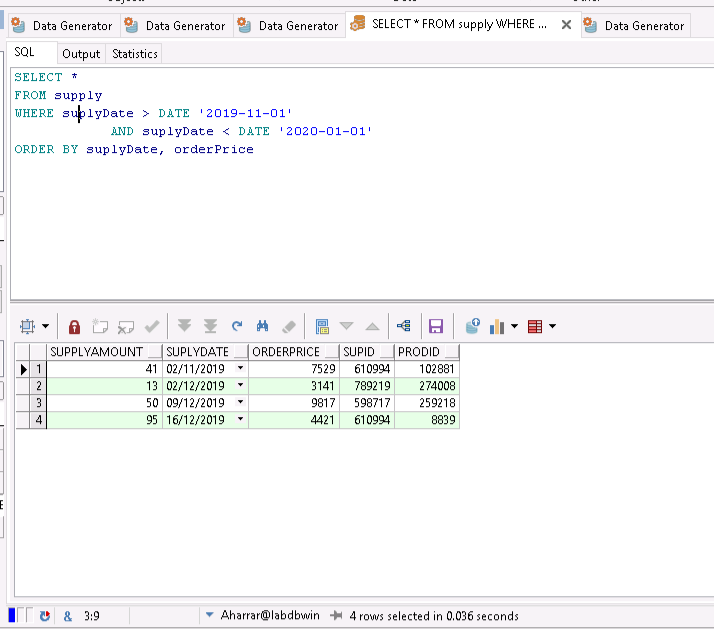
SELECT \*

FROM supply

WHERE suplyDate > DATE '2019-11-01'

AND suplyDate < DATE '2020-01-01'

ORDER BY suplyDate, orderPrice



Return the benefits on a year:

select distinct (select sum(purchaseamount\*price)

from purchase natural join sales natural join products

where saledate < to\_date('01-01-2020', 'dd/mm/yyyy') and saledate > to\_date('01-01-2019', 'dd/mm/yyyy')

)

-

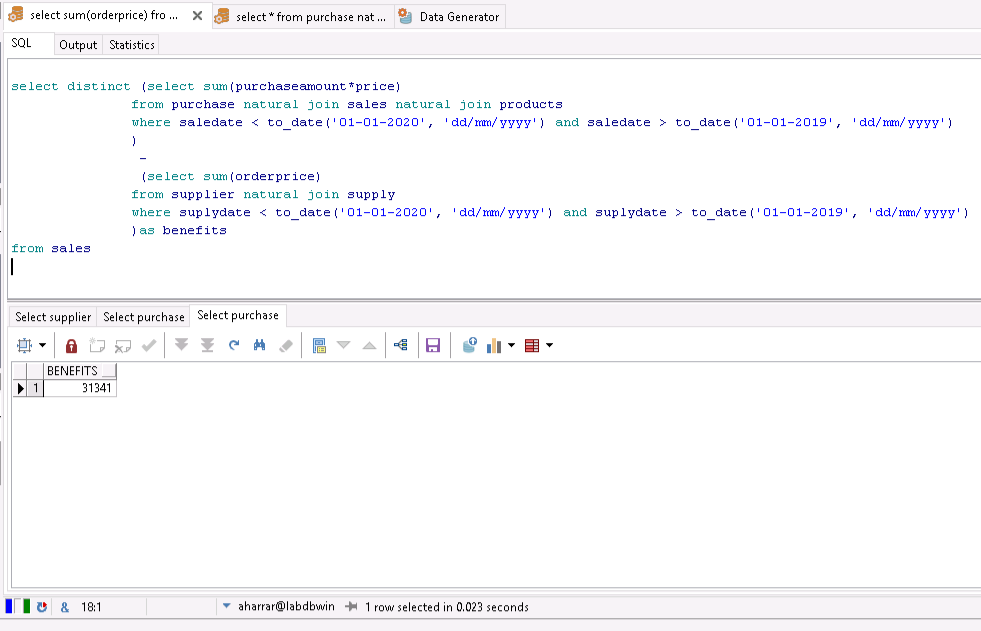
(select sum(orderprice)

from supplier natural join supply

where suplydate < to\_date('01-01-2020', 'dd/mm/yyyy') and suplydate > to\_date('01-01-2019', 'dd/mm/yyyy')

)as benefits

From sales --we could also used ‘dual’



That query return the employees made the most transaction:

select \*

from employee

where empid in (

select EMPLOYEEID from (select count(EMPLOYEEID) as c, employeeid

from sales

group by EMPLOYEEID)

where c = (select max(c)

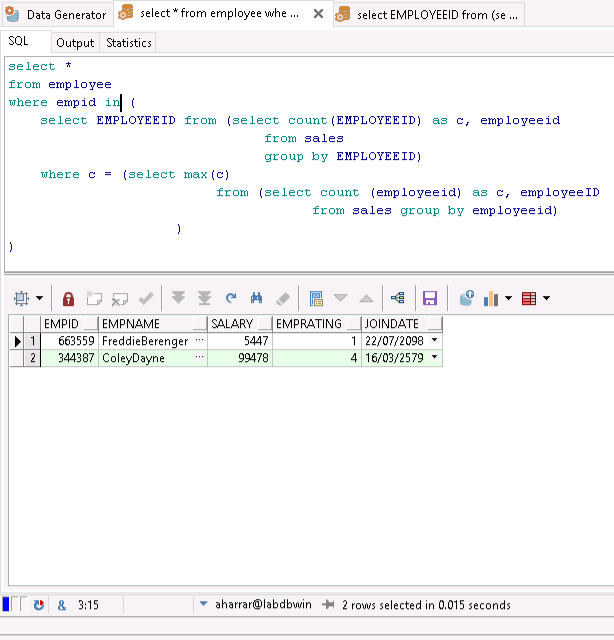
from (select count (employeeid) as c, employeeID

from sales

group by employeeid)

)

)



The following query return the client who bought the most:

select \*

from customer

where id = (

select id from (select count(id) as c, id

from sales

group by id)

where c = (select max(c)

from (select count (id) as c, id

from sales group by id)

)

)

Une image contenant capture d’écran

Description générée automatiquement

2.2) Indexes

1)

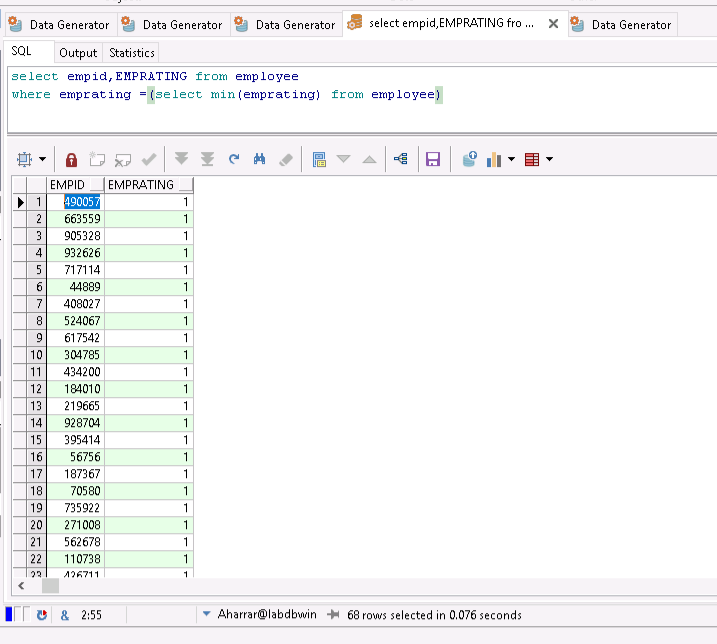
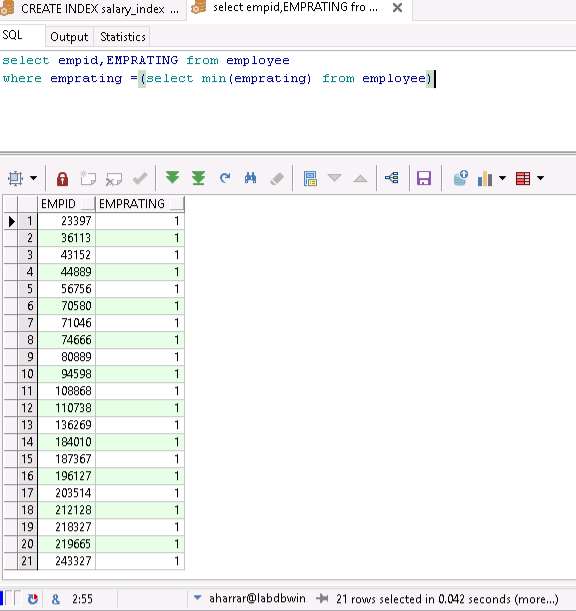
The index we created:

CREATE INDEX salary\_index  
ON employee (empid,EMPRATING)  
COMPUTE STATISTICS;

And the query we use as example:

select empid,EMPRATING from employee

where emprating =(select min(emprating) from employee)



The first time we ran the query(without the index) it was done in 0.076 seconds, now with the index it is done in 0.042 seconds

2)

The index:

CREATE INDEX supply\_index  
ON supply (supplyamount, suplydate, orderprice, supid, prodid)  
COMPUTE STATISTICS;

The query:

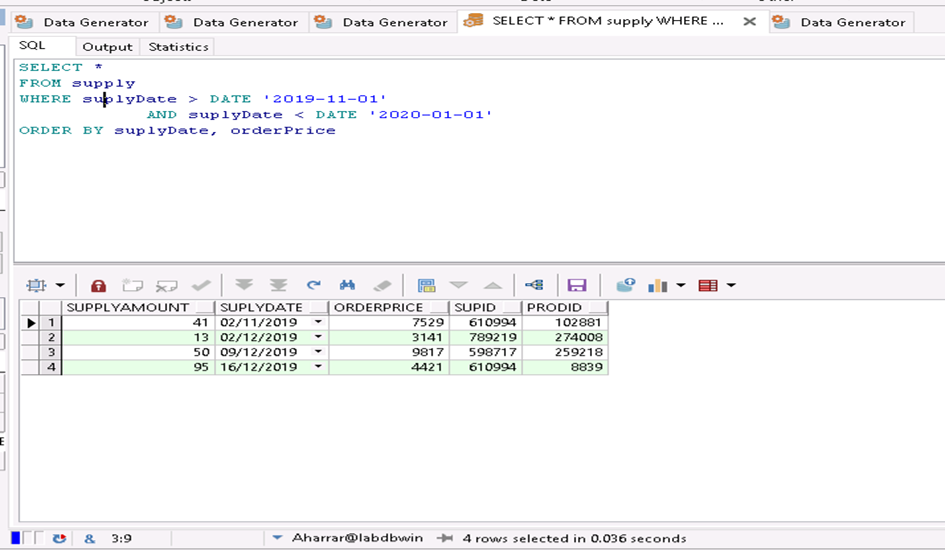
SELECT \*

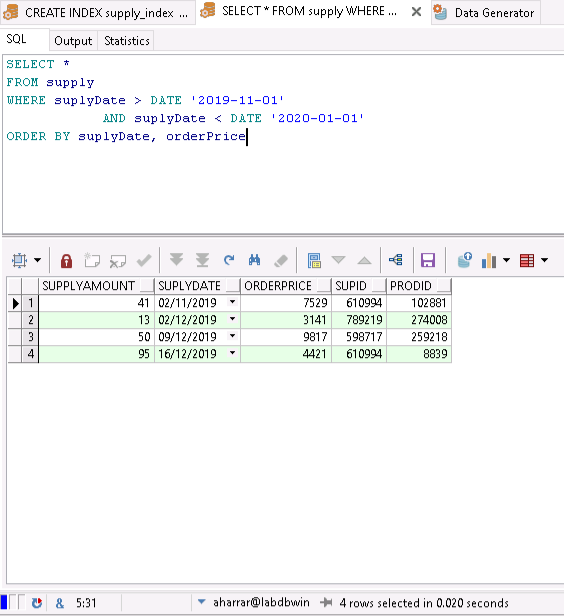
FROM supply

WHERE suplyDate > DATE '2019-11-01'

AND suplyDate < DATE '2020-01-01'

ORDER BY suplyDate, orderPrice





The first time we ran the query(without the index) it was done in 0.036 seconds, now with the index it is done in 0.020 seconds

3)

CREATE INDEX customer\_index  
ON customer(id, birthday, address, phone, email, name)  
COMPUTE STATISTICS;

select \*

from customer

where id = (

select id from (select count(id) as c, id

from sales

group by id)

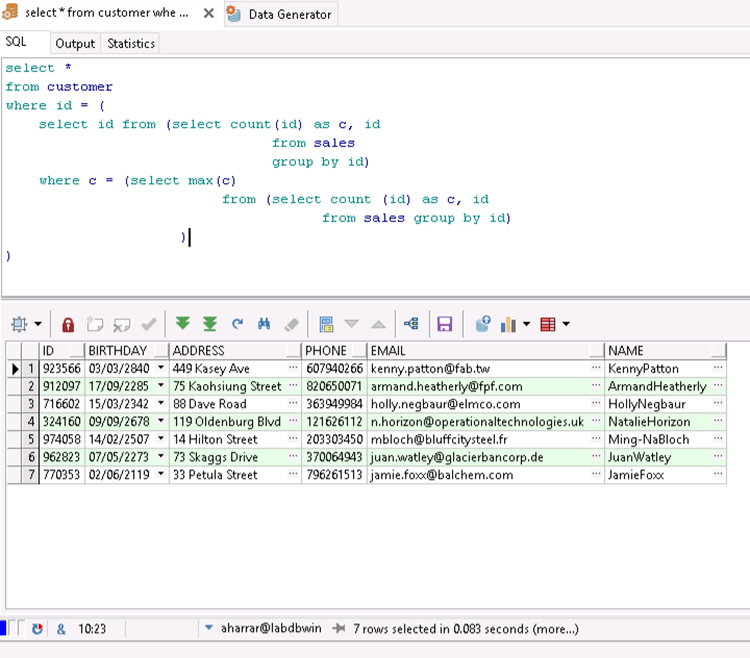
where c = (select max(c)

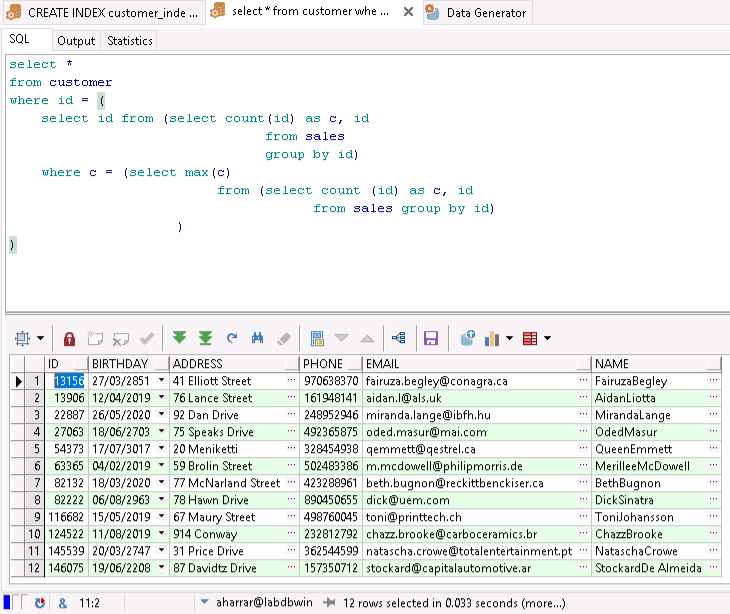
from (select count (id) as c, id

from sales group by id)

)

)

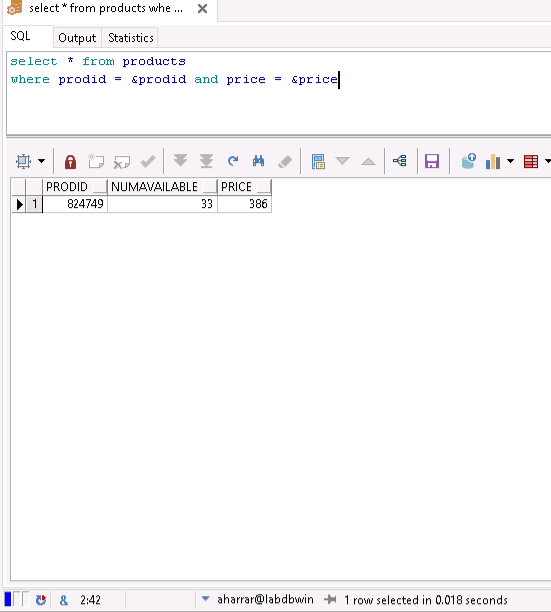


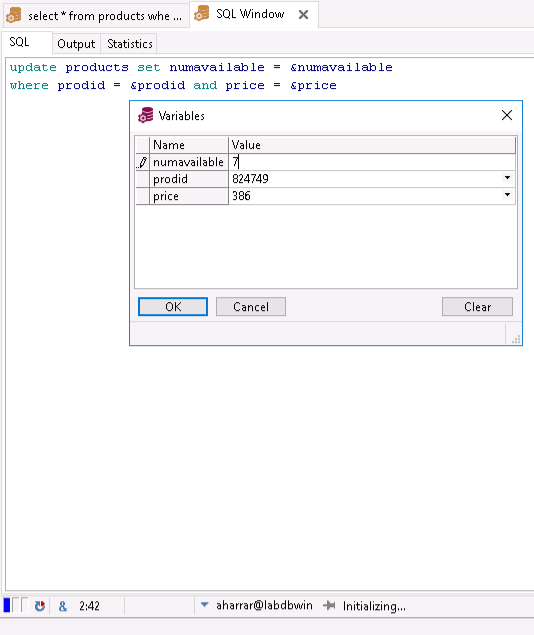


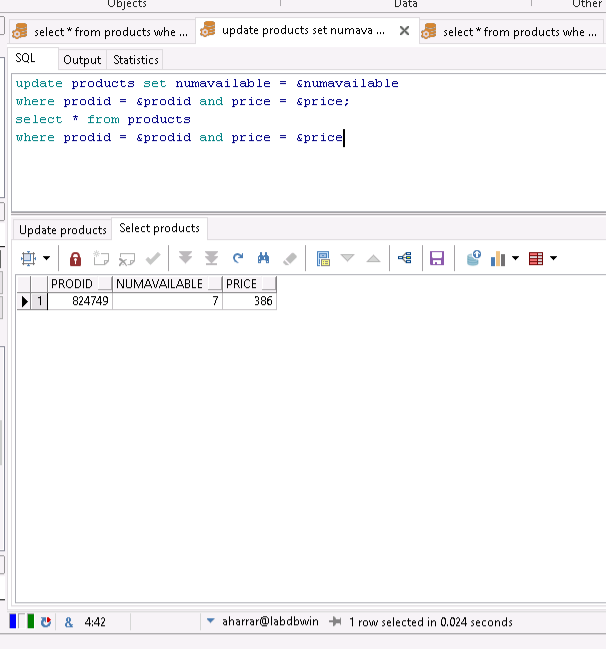
The first time we ran the query(without the index, and there were less elements in the database) it was done in 0.083 seconds, now with the index it is done in 0.033 seconds

2.3) Updates + commit and rollback

1)Update:

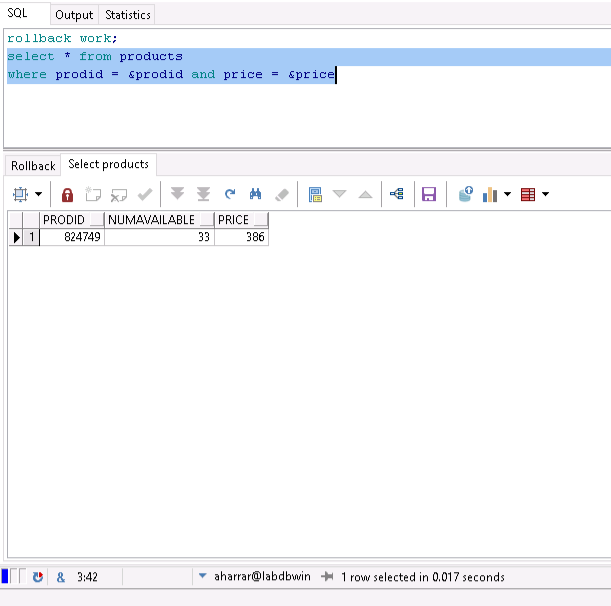




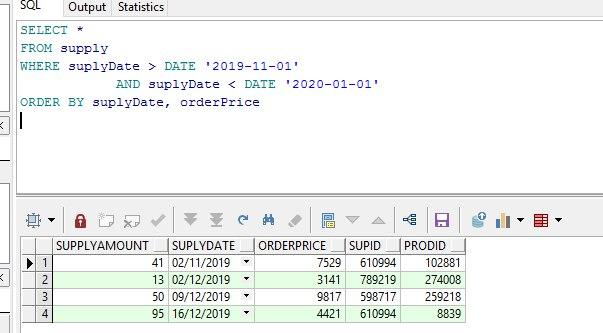


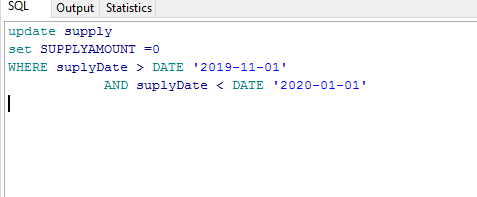
We updated the number of that specific item from 33 to 7

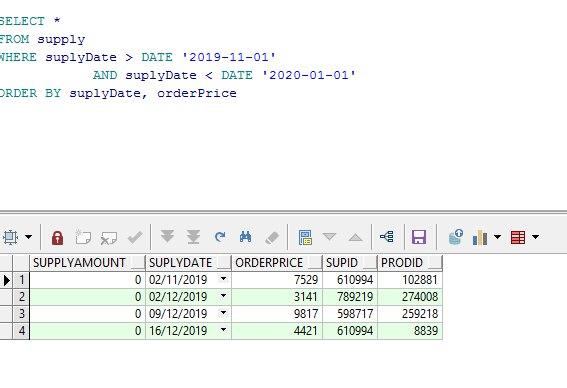
And now we make a rollback command



The second update query set the amount of a product to 0 if the supply occurred between the 2019/11/01 and the 2020/01/01 (not a good season?):



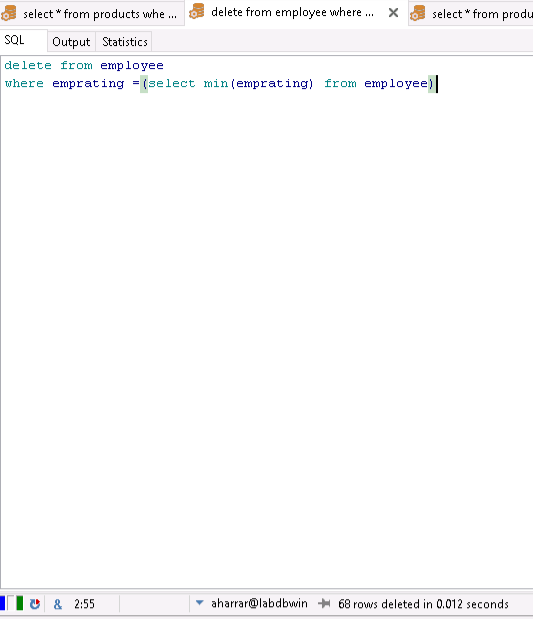


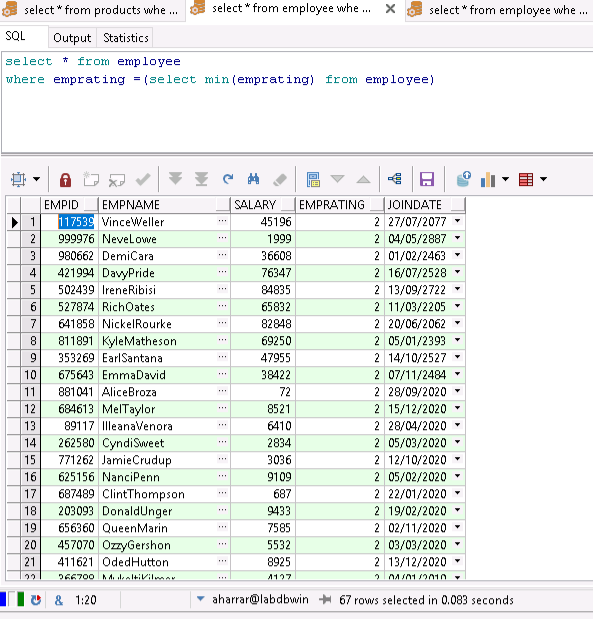
And 

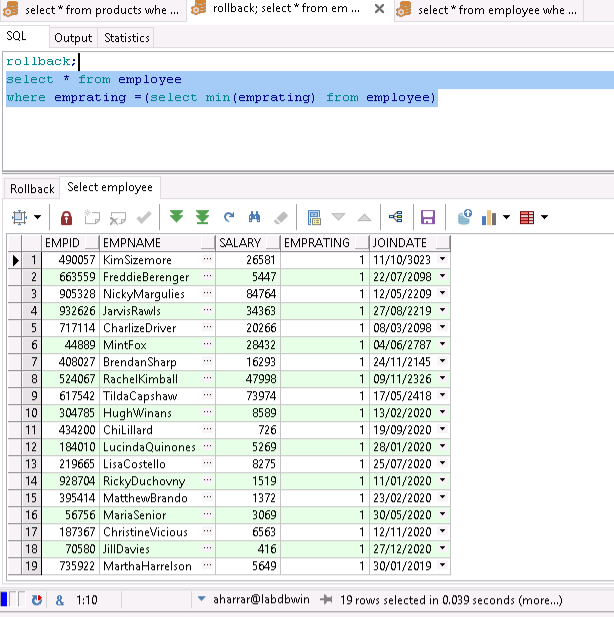
We can see that after the update command those article’s amount has been set to zero

Rollback:

In this example we decided to delete the worsts worker then rollback



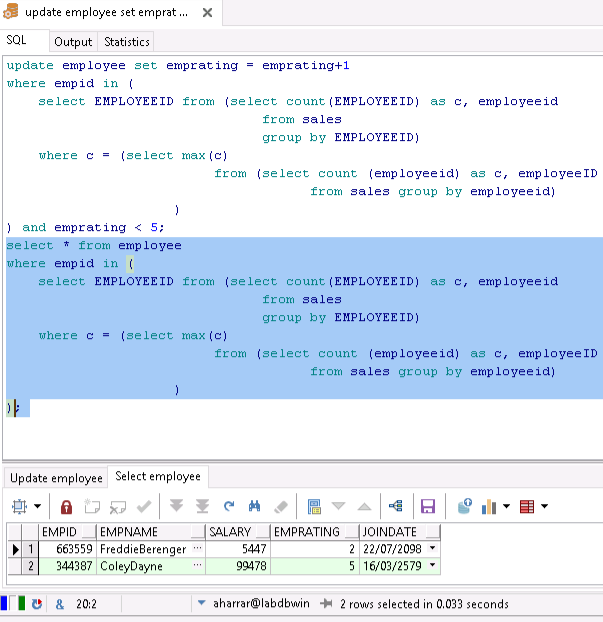




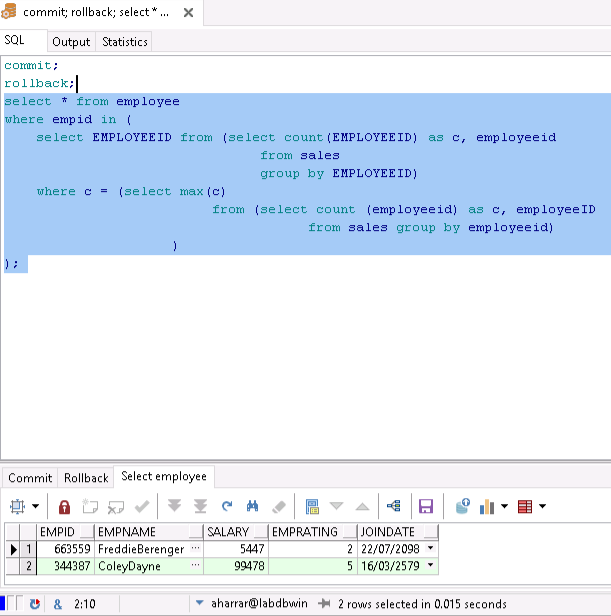
We first deleted the employee with rating = 1 (so in the second you can see only employee with rating = 2) then rollback

Commit:

In this example we will increase the rating of the best employee, run the command “commit” and show that rollback doesn’t work



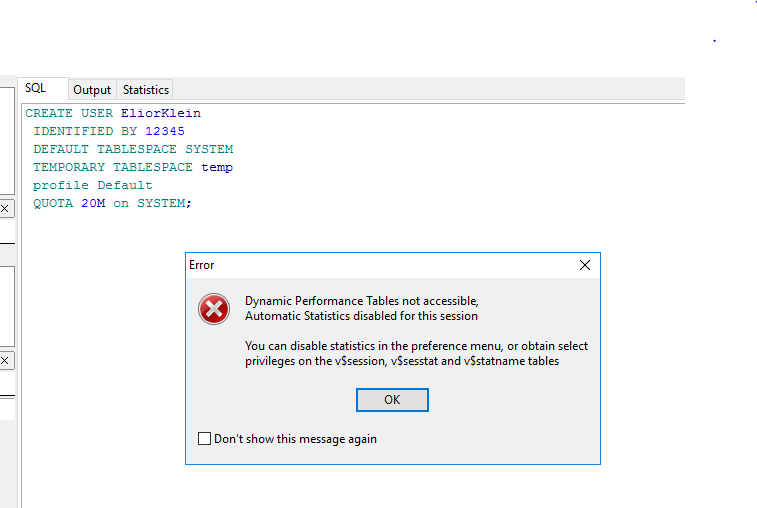
In the previous chapter (the one about the queries) we ran the same query only that time those employee has the rating 1 and 4, now the update worked and they have 2 and 5



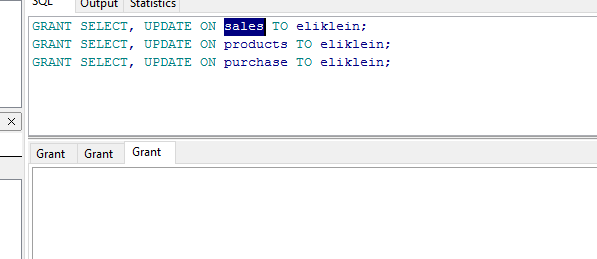
And this picture you can see we run the “commit” command and then the “rollback” command but the ratings stay the same

2.4) Revoke and grant

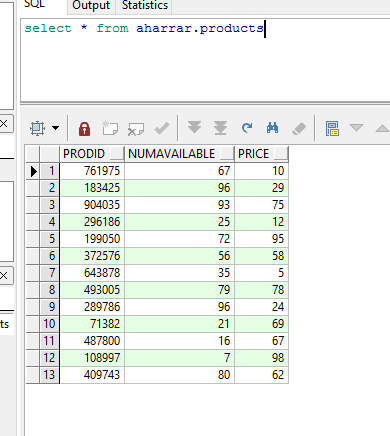
I try to give a permission to a new user but that doesn’t work because we doesn’t have the permission

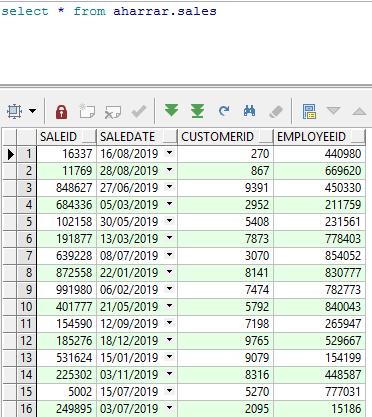


We will use an existing user

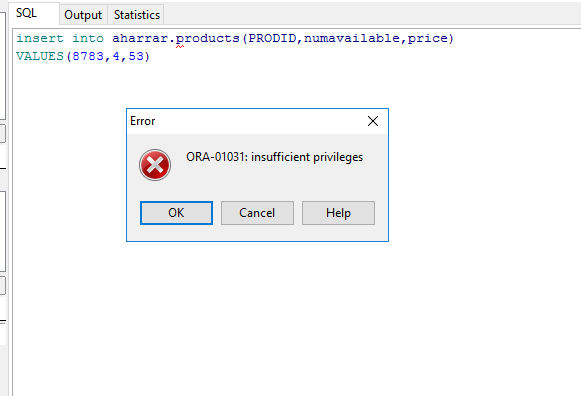


We will now get the tables from the other user





Now we will try to insert



2.5) Conditions

ALTER TABLE EMPLOYEE

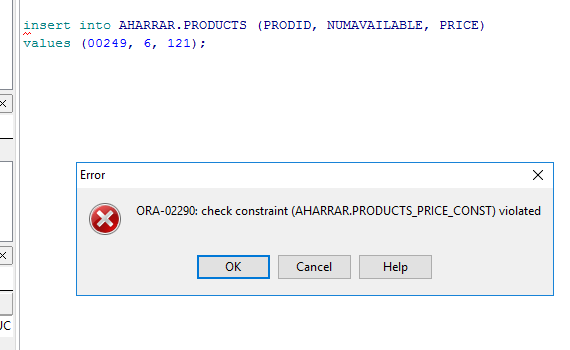
ADD CONSTRAINT EMPLOYEE\_SALARY CHECK (SALARY >5400 )

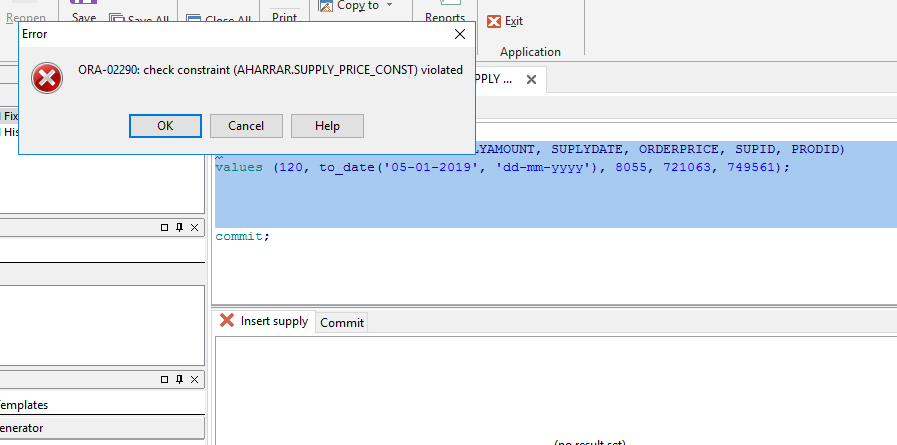
ALTER TABLE SUPPLY

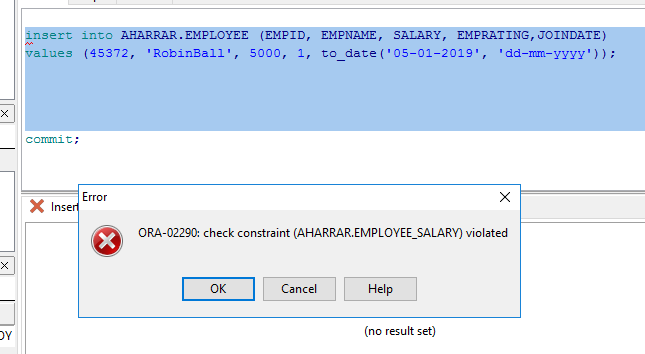
ADD CONSTRAINT SUPPLY\_PRICE\_CONST CHECK (SUPPLYAMOUNT < 100 );

ALTER TABLE products

ADD CONSTRAINT PRODUCTS\_PRICE\_CONST CHECK (PRICE < 100 );







2.6) Views

The following view shows the sales on a specific product on a specific year:

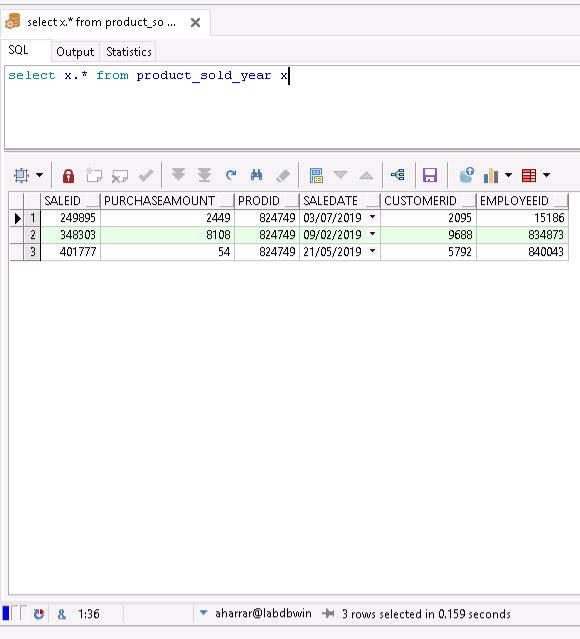
create view product\_sold\_year AS

SELECT \*

From purchase natural join sales

WHERE EXTRACT(year FROM saleDate) = &year1 AND prodID = &prodId

We get that result with the following parameters: prodID = 824749, year=2019



The following view shows the supply of different company on a specific day:

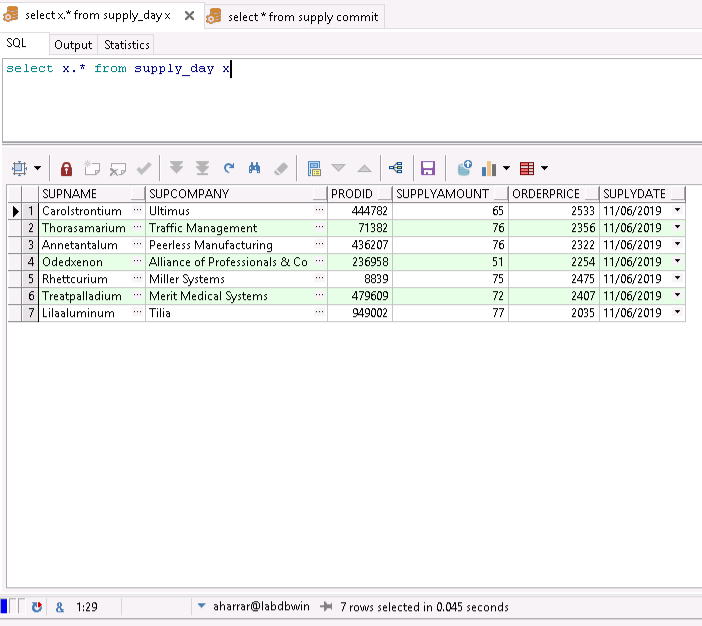
create view supply\_day AS

SELECT ser.supname, ser.supcompany, sy.prodid, sy.supplyamount, sy.orderprice, sy.suplydate

FROM supply sy natural join supplier ser

WHERE EXTRACT(day FROM sy.suplydate) = &day1 AND EXTRACT(MONTH FROM sy.suplydate) = &month1 AND EXTRACT(YEAR FROM sy.suplydate) = &year1

We get that result with the following parameters: day = 11, month = 6, year = 2019



Step 3)

3.1) Functions and procedures

That function is an event that give prizes to the best customers. In order to do that, the function checks every sales over the past years that the price is more than 150nis then chose someone randomaly to give him a prize.

create or replace procedure Lottery

is

idSales number;

priceSale number;

i number;

rand number;

cursor salesIterator

is select \*

from sales natural join customer

where saledate >= to\_date('01/01/2000', 'DD/MM/YYYY')

and saledate < to\_date('01/01/2020', 'DD/MM/YYYY')

order by saleid;

begin

i := 0;

for sales\_rec in salesIterator

loop

idSales := sales\_rec.saleid;

select (select sum(price)

from purchase natural join products

where saleid = idSales

)

into priceSale

from dual;

if priceSale > 150 then

i := i + 1;

dbms\_output.put\_line( i || ') ' || idSales || ' of customr: ' ||

sales\_rec.name || ' ' ||

sales\_rec.phone || ' ' ||

sales\_rec.email );

end if;

end loop;

select round(dbms\_random.value(1,i)) rnum

into rand

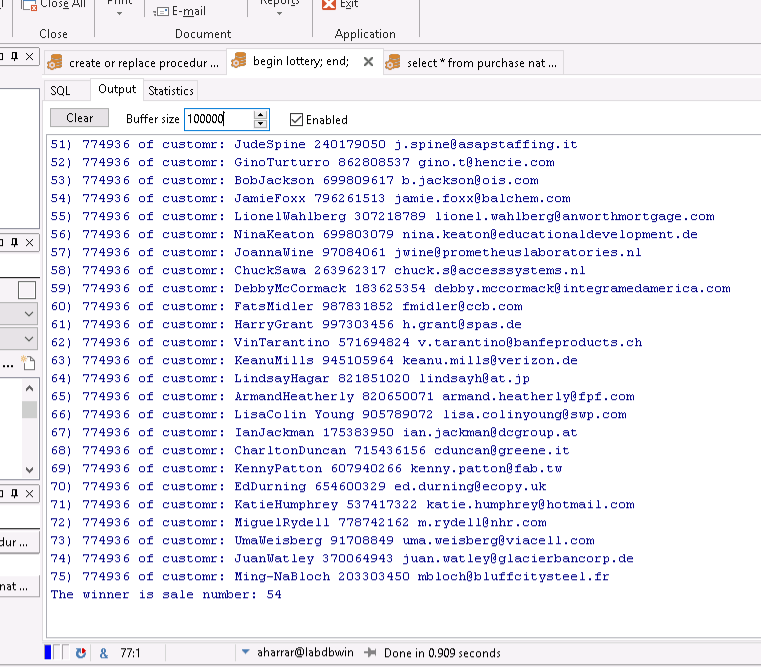
from dual;

dbms\_output.put\_line( 'The winner is sale number: ' || rand);

end Lottery;

The command to execute the function:

Begin lottery; end;



That function check every employee who worked on every month over the last year and calculate their salary:

create or replace procedure EmployeeSalary

is idEmp number;

mounthCounter number;

salaryEmp number;

M number;

totalSalaries number;

cursor employeeIterator

is select EXTRACT(MONTH FROM saledate) as m,empid, salary

from employee natural join sales

where saleid in ( select saleid from sales

where saledate >= to\_date('01/01/2019', 'DD/MM/YYYY')

and saledate < to\_date('01/01/2020', 'DD/MM/YYYY'))

order by empid, saledate;

begin

mounthCounter := 0;

idEmp := 0;

totalSalaries := 0;

for emp\_rec in employeeIterator

loop

if idEmp = emp\_rec.empid and M != emp\_rec.m then

mounthCounter := mounthCounter + 1;

else

if idEmp != 0 then

dbms\_output.put\_line('empid: ' || idEmp || ' salary for this year: ' || salaryEmp \* mounthCounter );

end if;

mounthCounter := 1;

idEmp := emp\_rec.empid;

salaryEmp := emp\_rec.salary;

M := emp\_rec.m;

end if;

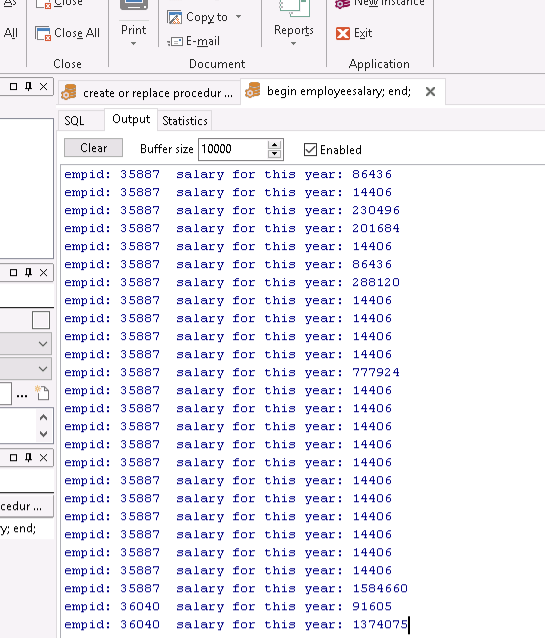
end loop;

dbms\_output.put\_line('Total Salaries for all employees this year is:' || totalSalaries);

end EmployeeSalary;

the command to call the function:

begin employeesalary; end;



3.2) Trigers

1) That triger will raise an exception if a purchase occure and there is not enough available products.

create or replace trigger on\_sale

before insert

on purchase

for each row

declare

c\_amount INTEGER;

begin

select p.numavailable into c\_amount

from products p

where p.prod = :new.prodid;

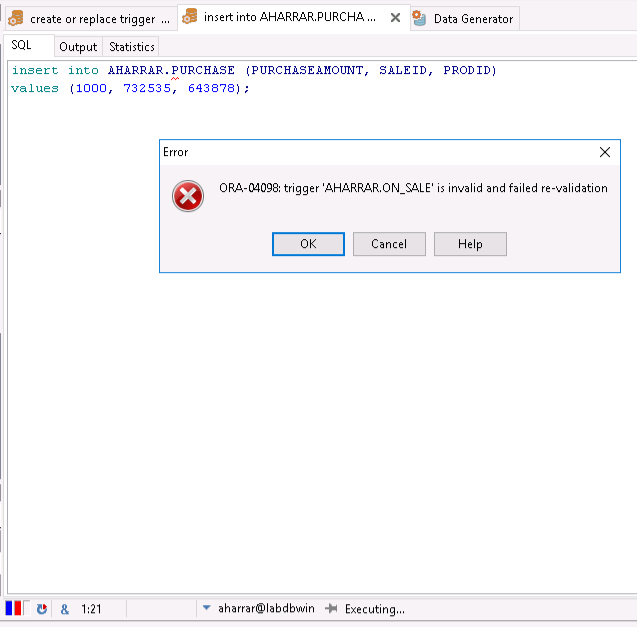
if :new.purchaseamount > c\_amount then

raise\_application\_error(-20001, 'More than available');

end if;

end on\_sale;

Now if we wants to buy 1000 time the same product it will raise an error:



2) That trigger will update the attribute “numavailable” of a product when a supply occure.

create or replace trigger on\_supply

after insert

on supply

for each row

declare

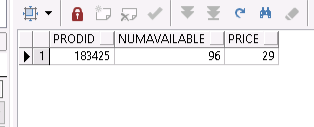
begin

update products p

set p.numavailable = p.numavailable + :new.supplyamount

where p.prodid = :new.prodid;

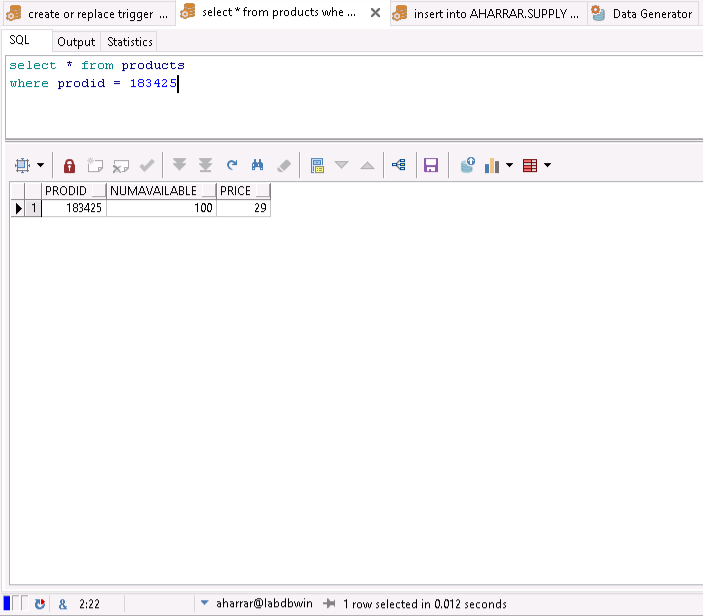
end on\_supply;



insert into AHARRAR.SUPPLY (SUPPLYAMOUNT, SUPLYDATE, ORDERPRICE, SUPID, PRODID)

values (4, to\_date('06-04-2019', 'dd-mm-yyyy'), 200, 894826, 183425);

commit;



And here we can see that the counter changed from ‘96’ to ‘100’

3.3) Queries

1)Return every employee who work at the variable “year”:

SELECT EMPID,EMPNAME,salary,EMPRATING,JOINDATE

FROM employee NATURAL JOIN sales WHERE EXTRACT(YEAR FROM SALEDATE) = &year

2)select SALEID, sum(price)

from purchase natural join products natural join sales

where (case when &date <> '0' then to\_date(&date, 'dd-mm-yyyy') end) in (select DIARYDATE

from diary) And

(case when &date <> '0' then to\_date(&date, 'dd-mm-yyyy') end) = SALEDATE

group by SALEID

order by SALEID

3)

The following view shows the sales on a specific product on a specific year:

create view product\_sold\_year AS

SELECT \*

From purchase natural join sales

WHERE EXTRACT(year FROM saleDate) = &year1 AND prodID = &prodId

4) The following view shows the supplyof different company on a specific day:

create view supply\_day AS

SELECT ser.supname, ser.supcompany, sy.prodid, sy.supplyamount, sy.orderprice, sy.suplydate

FROM supply sy natural join supplier ser

WHERE EXTRACT(day FROM sy.suplydate) = &day1 AND EXTRACT(MONTH FROM sy.suplydate) = &month1 AND EXTRACT(YEAR FROM sy.suplydate) = &year1

3.4 (Report

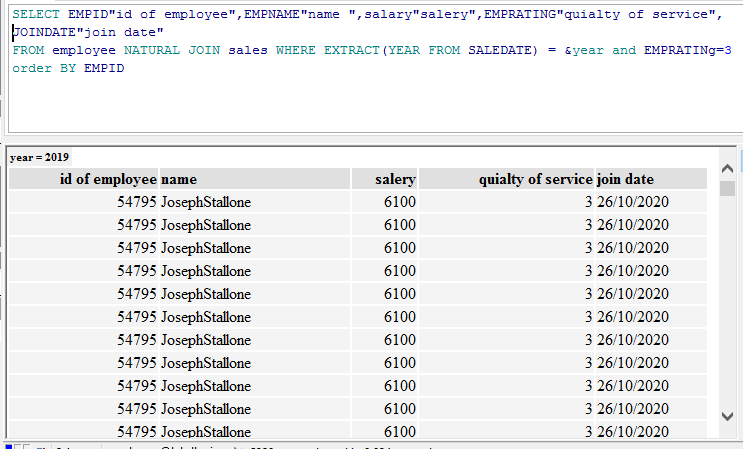
3.4.1)That report show us every employee who work at the variable “year” and with "rating":

SELECT EMPID"id of employee",EMPNAME"name ",salary"salary",EMPRATING"quialty of service",

JOINDATE"join date"

FROM employee NATURAL JOIN sales WHERE EXTRACT(YEAR FROM SALEDATE) = &year and EMPRATINg=3

order BY EMPID



3.4.2)

The report show us every products that doesn’t left much(numAvailable<10) , products that we need to make an order and their price(calculated by the price of the order divided by the amount):

SELECT p.prodid "product id"

,p.NUMAVAILABLE "available in stock",

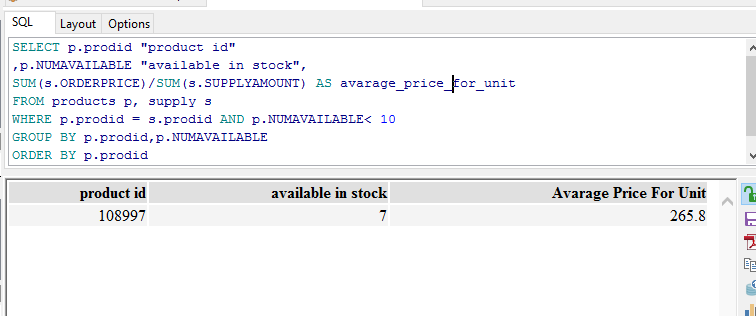
SUM(s.ORDERPRICE)/SUM(s.SUPPLYAMOUNT) AS avarage\_price\_for\_unit

FROM products p, supply s

WHERE p.prodid = s.prodid AND p.NUMAVAILABLE< 10

GROUP BY p.prodid,p.NUMAVAILABLE

ORDER BY p.prodid



3.5)graph

We will now present a graph of revenue distribution for a particular year by query(התפלגות ההכנסות לשנה):

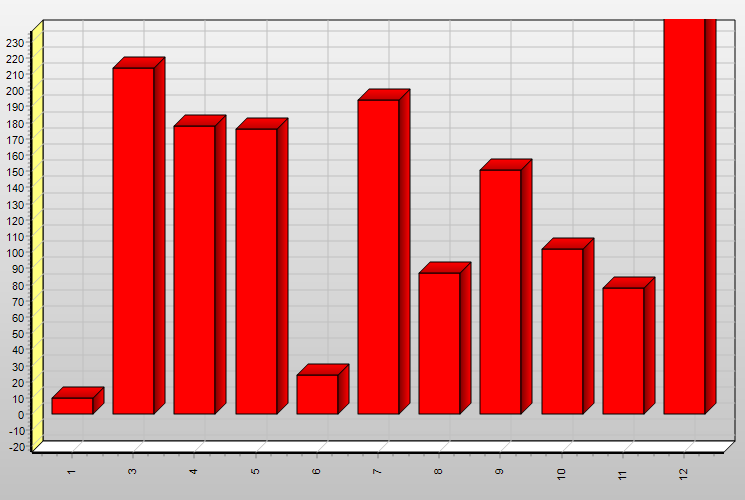
SELECT sum(price), EXTRACT(MONTH FROM SALEDATE) AS "Month"

from purchase natural join products natural join sales

WHERE EXTRACT(YEAR FROM SALEDATE) = &year

GROUP BY EXTRACT(MONTH FROM saledate)

ORDER BY "Month"



3.5.2)

Step 4)

The supermarket has now a website where client can purchase products.

4.1)

Entity 1) Esales:

The sale that occurred on the website.

Has the following attributes:

EsaleId => the number of the sale that identify it from the others

customerAcc => the number account of the customer

Esalesdate => when the sale occurred

The entity Esales is bound to the entity products with the relationship “Echase” and has the attribute ”Amount of” to know how many of a product the customer bought

Entity 2) Ecustomer :

That entity contain the information of the customer.

Has the following attributes:

CustomerAcc => the number account of the customer

Email => email of the customer

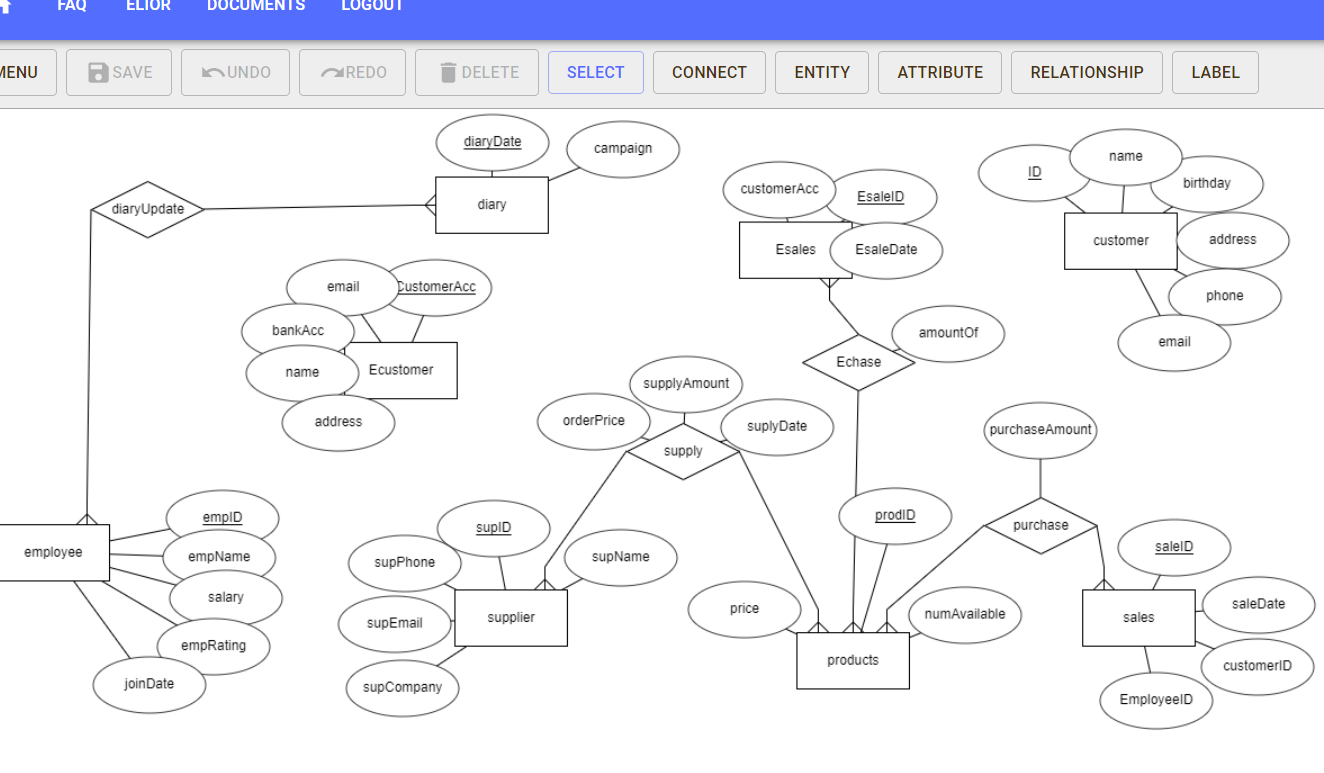
bankAcc => bank coordinates

name => name of the customer

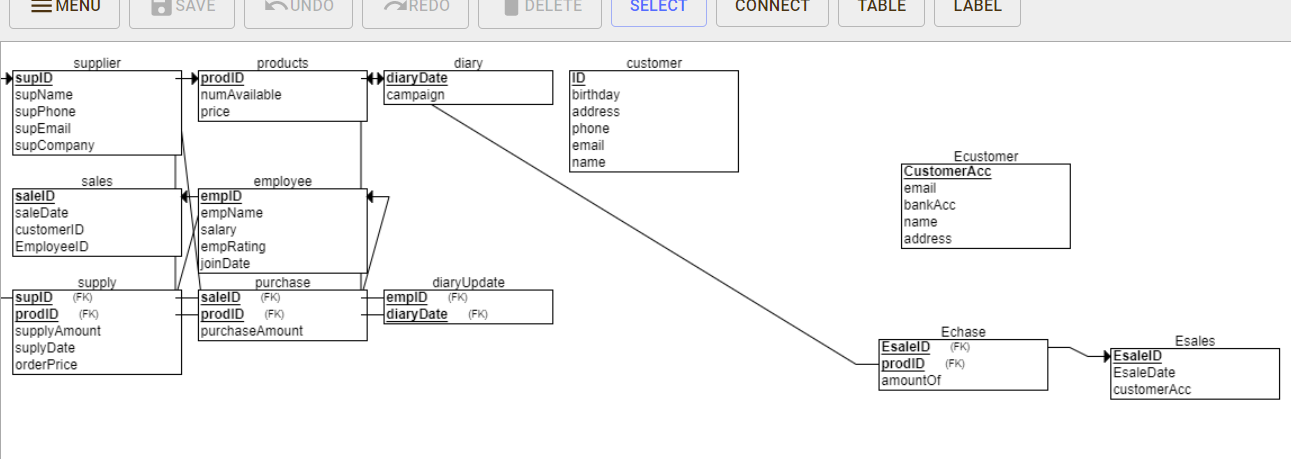
address => for the delivery (coming soon…)

4.2)

Extended ERD:

D

Extended DSD:



4.3)

CREATE TABLE Esales

(

EsaleID INT NOT NULL,

EsaleDate DATE NOT NULL,

customerAcc INT NOT NULL,

PRIMARY KEY (EsaleID)

);

CREATE TABLE Ecustomer

(

CustomerAcc INT NOT NULL,

email VARCHAR(50) NULL,

bankAcc INT NOT NULL,

name VARCHAR(50) NULL,

address VARCHAR(50) NULL,

PRIMARY KEY (CustomerAcc)

);

CREATE TABLE Echase

(

amountOf INT NOT NULL,

EsaleID INT NOT NULL,

prodID INT NOT NULL,

PRIMARY KEY (EsaleID, prodID),

FOREIGN KEY (EsaleID) REFERENCES Esales(EsaleID),

FOREIGN KEY (prodID) REFERENCES products(prodID)

);

4.4) Done…

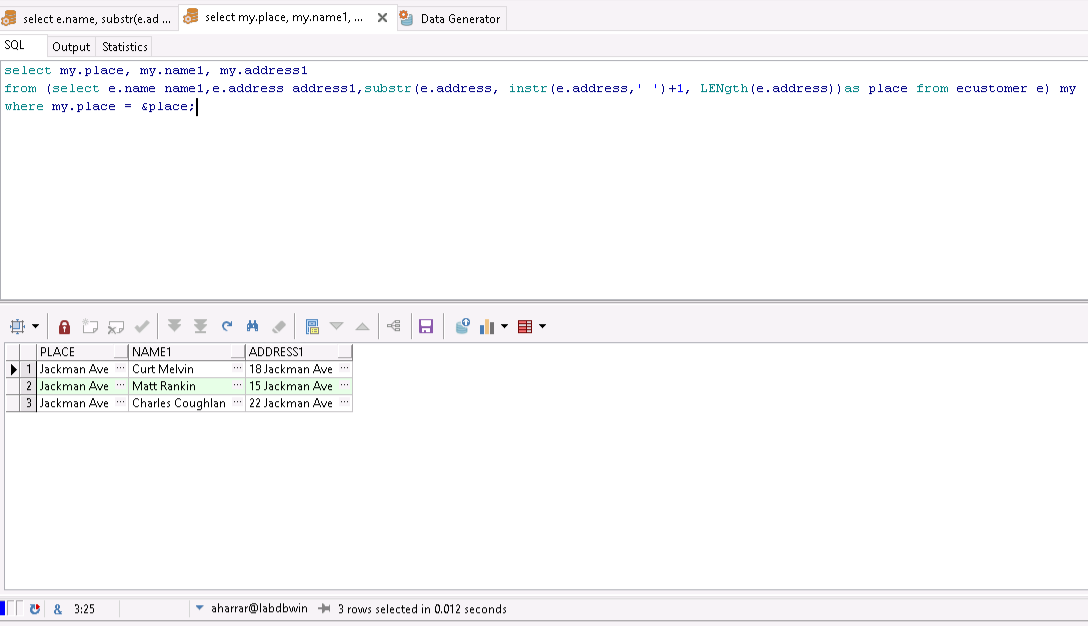
4.5)

1) Query that return every customer that lives in the same neighborhood to facilitate the deliveries:

select my.place, my.name1, my.address1

from (select e.name name1,e.address address1,substr(e.address, instr(e.address,' ')+1, LENgth(e.address))as place from ecustomer e) my

where my.place = &place;



2) The following query return the benefits of every sales(online included):

select distinct (select sum(purchaseamount\*price)

from purchase natural join sales natural join products

where saledate < to\_date('01-01-2020', 'dd/mm/yyyy') and saledate > to\_date('01-01-2019', 'dd/mm/yyyy')

)

+

(select sum(amountof\*price)

from echase natural join esales natural join products

where esaledate < to\_date('01-01-2020', 'dd/mm/yyyy') and esaledate > to\_date('01-01-2019', 'dd/mm/yyyy')

)

-

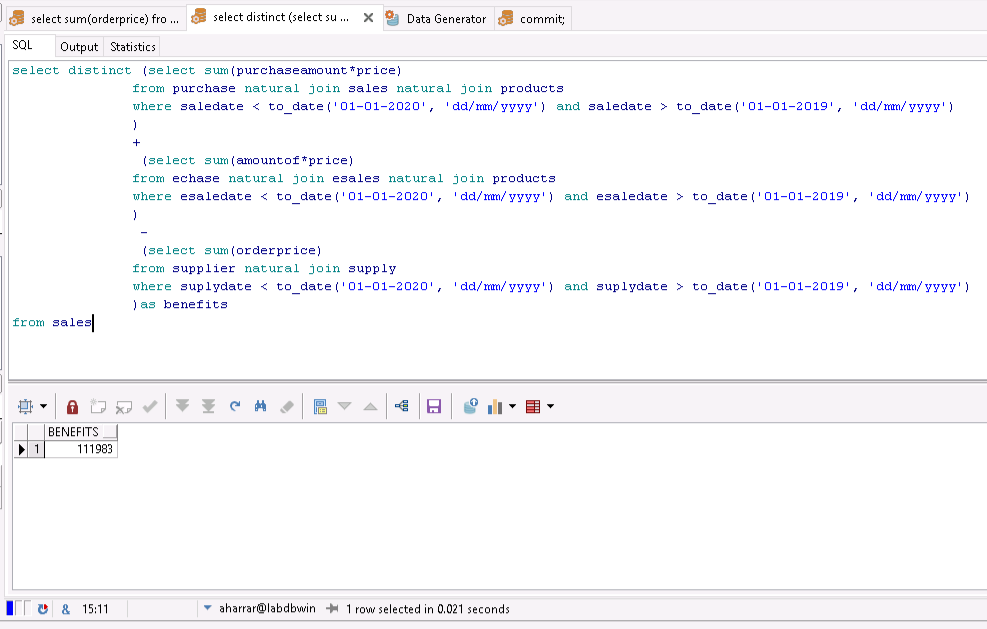
(select sum(orderprice)

from supplier natural join supply

where suplydate < to\_date('01-01-2020', 'dd/mm/yyyy') and suplydate > to\_date('01-01-2019', 'dd/mm/yyyy')

)as benefits

from sales



4.6)

First view:

Create view neighbors as

select my.place, my.name1, my.address1

from (select e.name name1,e.address address1,substr(e.address, instr(e.address,' ')+1, LENgth(e.address))as place from ecustomer e) my

where my.place = &place;

and the command to run it:

select x.neighbor x;

Second view:

create view Benefits as

select distinct (select sum(purchaseamount\*price)

from purchase natural join sales natural join products

where saledate < to\_date('01-01-2020', 'dd/mm/yyyy') and saledate > to\_date('01-01-2019', 'dd/mm/yyyy')

)

+

(select sum(amountof\*price)

from echase natural join esales natural join products

where esaledate < to\_date('01-01-2020', 'dd/mm/yyyy') and esaledate > to\_date('01-01-2019', 'dd/mm/yyyy')

)

-

(select sum(orderprice)

from supplier natural join supply

where suplydate < to\_date('01-01-2020', 'dd/mm/yyyy') and suplydate > to\_date('01-01-2019', 'dd/mm/yyyy')

)as benefits

from sales;

And the command to run it:

Select x.\* benefits x;

THE END