UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

REPORT ABOUT PRACTICE 1

DATA STRUCTURES

GROUP 1291

Degree in Bilingual Computer Engineering

Lucía Gil Maroto and Paula Beltrán Marianini

Contenido

1. <u>GOALS</u>

In this assignment we will design, develop, and query a database containing scale models of classic cars. This assignment will make us gain dexterity in SQL Language, which is new for us.

2. <u>EXPLORE THE DATA BASE</u>

Once we have installed the virtual machine with the DBeaver and the PostgreSQL following the tutorial uploaded to model, it is time to start with the assignment.

We have created the database called classicmodels using the "createdb" command from the Makefile. Then, we have explored the database from DBeaver.

The first tasks from this assignment are identify the primary keys and foreign keys of each table and to create the database relational schema.

Therefore, using the nomenclature we were asked, the primary and foreign keys from each table are:

orders(**ordernumber**, costumernumber→costumers.costumernumber)

costumers(**costumernumber**, salesrepemployeenumber→employees.eployeenumber)

employees(**employeenumber**, officecode→offices.officecode)

offices(**officecode**)

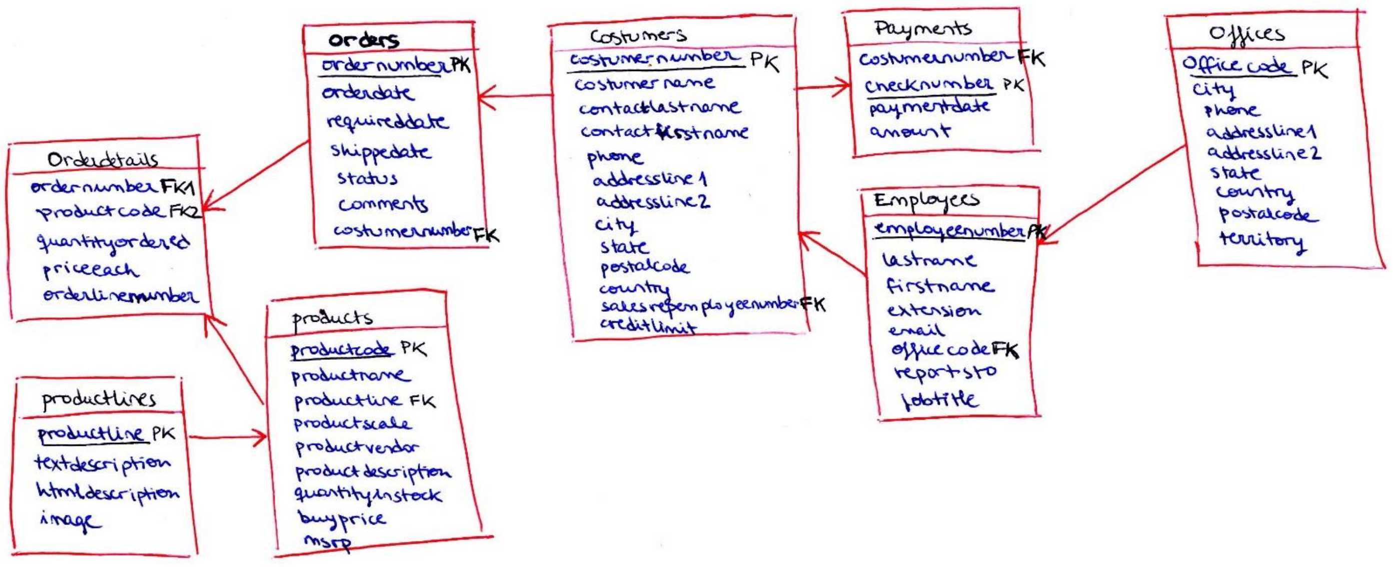
orderdetails(ordernumber→orders.ordernumber, productcode→products.productcode)

products(**productcode**, productline→productlines.productline)

productlines(**productlines**)

payments(**checknumber**, costumernumber→costumers.costumernumber)

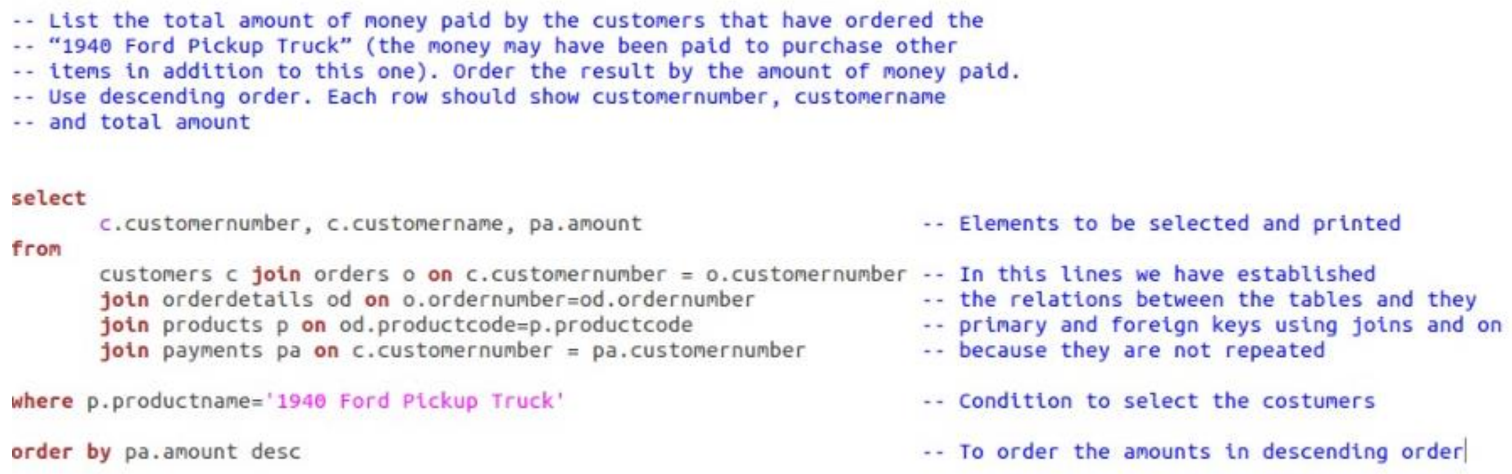
Then, the database relational schema is (done in a paper and already scanned):



3. QUERIES

We have explained all the queries in each .sql archive but we have added here some images of our queries with the brief comments regarding implementation:

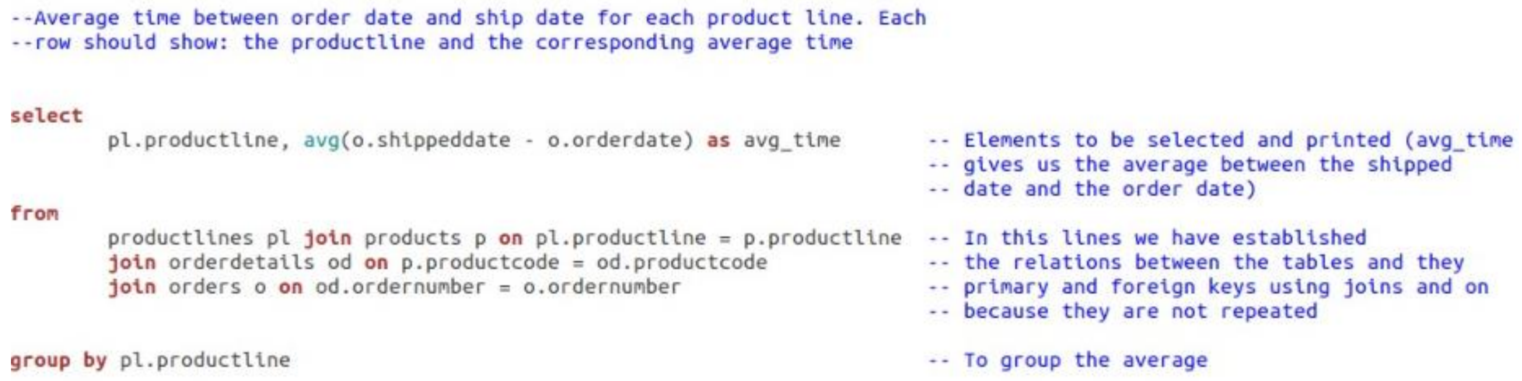
Query 1.



```
-- List the total amount of money paid by the customers that have ordered the
-- "1940 Ford Pickup Truck" (the money may have been paid to purchase other
-- items in addition to this one). Order the result by the amount of money paid.
-- Use descending order. Each row should show customernumber, customername
-- and total amount


select
       c.customernumber, c.customername, pa.amount                    -- Elements to be selected and printed
from
       customers c join orders o on c.customernumber = o.customernumber -- In this lines we have established
       join orderdetails od on o.ordernumber=od.ordernumber             -- the relations between the tables and they
       join products p on od.productcode=p.productcode                  -- primary and foreign keys using joins and on
       join payments pa on c.customernumber = pa.customernumber         -- because they are not repeated

where p.productname='1940 Ford Pickup Truck'                            -- Condition to select the costumers

order by pa.amount desc                                                 -- To order the amounts in descending order
```

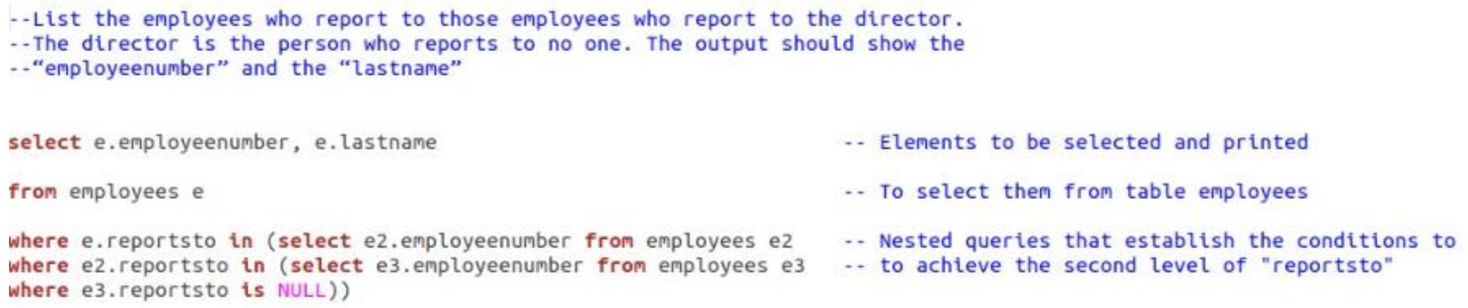
In query 1 we haven't been able to achieve the output desire because we get the results of all the orders made from each costumer that have purchased that item once, we don't know if this is another limitation in the database or that we haven't been able to implement it.

Query 2:



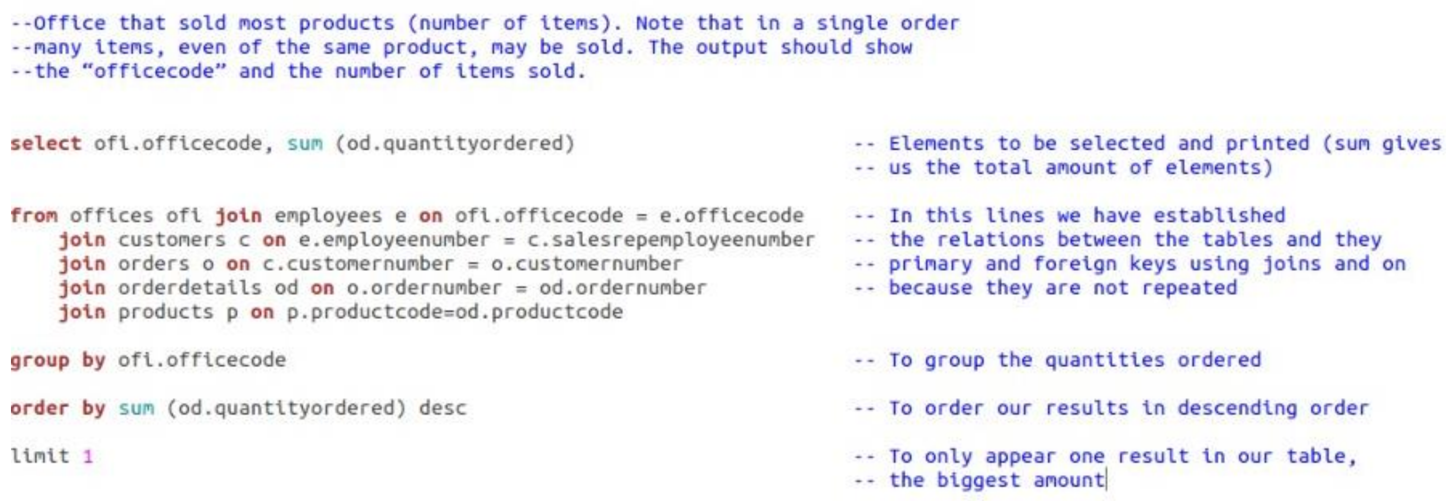
```
--Average time between order date and ship date for each product line. Each
--row should show: the productline and the corresponding average time


select
       pl.productline, avg(o.shippeddate - o.orderdate) as avg_time       -- Elements to be selected and printed (avg_time
                                                                          -- gives us the average between the shipped
                                                                          -- date and the order date)
from
       productlines pl join products p on pl.productline = p.productline  -- In this lines we have established
       join orderdetails od on p.productcode = od.productcode             -- the relations between the tables and they
       join orders o on od.ordernumber = o.ordernumber                    -- primary and foreign keys using joins and on
                                                                          -- because they are not repeated

group by pl.productline                                                   -- To group the average
```

Query 3:



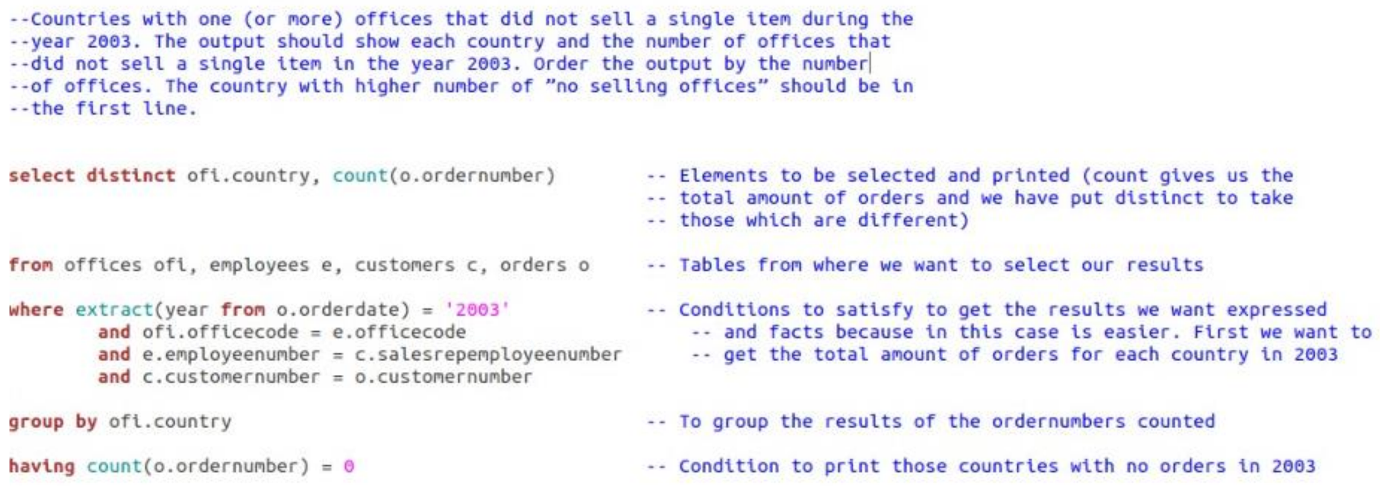
```
--List the employees who report to those employees who report to the director.
--The director is the person who reports to no one. The output should show the
--"employeenumber" and the "lastname"


select e.employeenumber, e.lastname                                -- Elements to be selected and printed

from employees e                                                   -- To select them from table employees

where e.reportsto in (select e2.employeenumber from employees e2   -- Nested queries that establish the conditions to
where e2.reportsto in (select e3.employeenumber from employees e3  -- to achieve the second level of "reportsto"
where e3.reportsto is NULL))
```

Query 4:



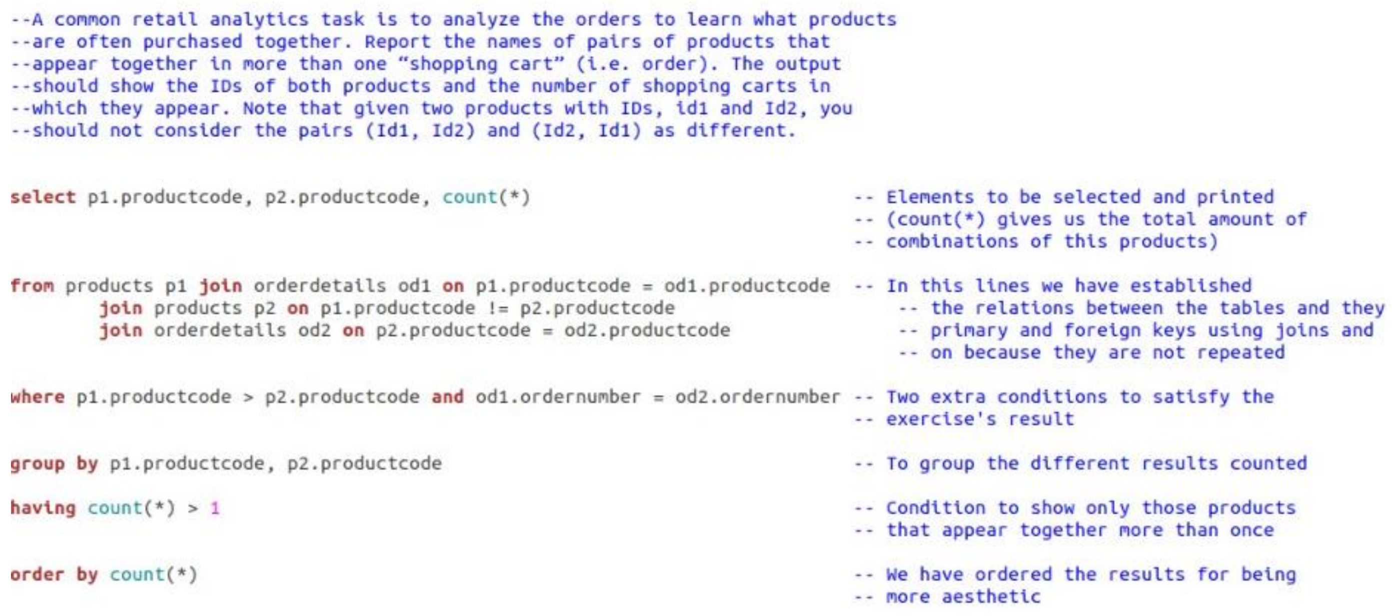
```sql
--Office that sold most products (number of items). Note that in a single order
--many items, even of the same product, may be sold. The output should show
--the “officecode” and the number of items sold.


select ofi.officecode, sum (od.quantityordered)                       -- Elements to be selected and printed (sum gives
                                                                      -- us the total amount of elements)

from offices ofi join employees e on ofi.officecode = e.officecode    -- In this lines we have established
    join customers c on e.employeenumber = c.salesrepemployeenumber   -- the relations between the tables and they
    join orders o on c.customernumber = o.customernumber              -- primary and foreign keys using joins and on
    join orderdetails od on o.ordernumber = od.ordernumber            -- because they are not repeated
    join products p on p.productcode=od.productcode

group by ofi.officecode                                               -- To group the quantities ordered

order by sum (od.quantityordered) desc                                -- To order our results in descending order

limit 1                                                               -- To only appear one result in our table,
                                                                      -- the biggest amount
```

Query 5:



```sql
--Countries with one (or more) offices that did not sell a single item during the
--year 2003. The output should show each country and the number of offices that
--did not sell a single item in the year 2003. Order the output by the number
--of offices. The country with higher number of ”no selling offices” should be in
--the first line.


select distinct ofi.country, count(o.ordernumber)          -- Elements to be selected and printed (count gives us the
                                                           -- total amount of orders and we have put distinct to take
                                                           -- those which are different)

from offices ofi, employees e, customers c, orders o       -- Tables from where we want to select our results

where extract(year from o.orderdate) = '2003'              -- Conditions to satisfy to get the results we want expressed
        and ofi.officecode = e.officecode                     -- and facts because in this case is easier. First we want to
        and e.employeenumber = c.salesrepemployeenumber       -- get the total amount of orders for each country in 2003
        and c.customernumber = o.customernumber

group by ofi.country                                       -- To group the results of the ordernumbers counted

having count(o.ordernumber) = 0                            -- Condition to print those countries with no orders in 2003
```

Query 6:



```sql
--A common retail analytics task is to analyze the orders to learn what products
--are often purchased together. Report the names of pairs of products that
--appear together in more than one “shopping cart” (i.e. order). The output
--should show the IDs of both products and the number of shopping carts in
--which they appear. Note that given two products with IDs, id1 and Id2, you
--should not consider the pairs (Id1, Id2) and (Id2, Id1) as different.


select p1.productcode, p2.productcode, count(*)                              -- Elements to be selected and printed
                                                                             -- (count(*) gives us the total amount of
                                                                             -- combinations of this products)

from products p1 join orderdetails od1 on p1.productcode = od1.productcode  -- In this lines we have established
        join products p2 on p1.productcode != p2.productcode                   -- the relations between the tables and they
        join orderdetails od2 on p2.productcode = od2.productcode              -- primary and foreign keys using joins and
                                                                               -- on because they are not repeated

where p1.productcode > p2.productcode and od1.ordernumber = od2.ordernumber -- Two extra conditions to satisfy the
                                                                             -- exercise's result

group by p1.productcode, p2.productcode                                      -- To group the different results counted

having count(*) > 1                                                          -- Condition to show only those products
                                                                             -- that appear together more than once

order by count(*)                                                            -- We have ordered the results for being
                                                                             -- more aesthetic
```

4. DATABASE REDISIGN

Due to lack of time, we are not going to submit this part of the assignment.