Assignment 1 report

Pablo Cuesta Sierra and Álvaro Zamanillo Sáez. Group 1251.

Section 2. Primary keys and Foreign keys of each table of the database:

offices(**officecode**, city, phone, addressline1, addressline2, state, country, postalcode, territory)

employees(**employeenumber**, lastname, firstname, extension, email, officecode→offices.officecode, reportsto→employeenumber, jobtitle)

customers(**customernumber**, customername, contactlastname, contactfirstname, phone, addressline1, addressline2, city, state, postalcode, country, salesrepemployeenumber→employees.employeenumber, creditlimit)

payments(**customernumber**→customers.customernumber, **checknumber**, paymentdate, amount)

orders(**ordernumber**, orderdate, requireddate, shippeddate, status, comments, customernumber→customers.customernumber)

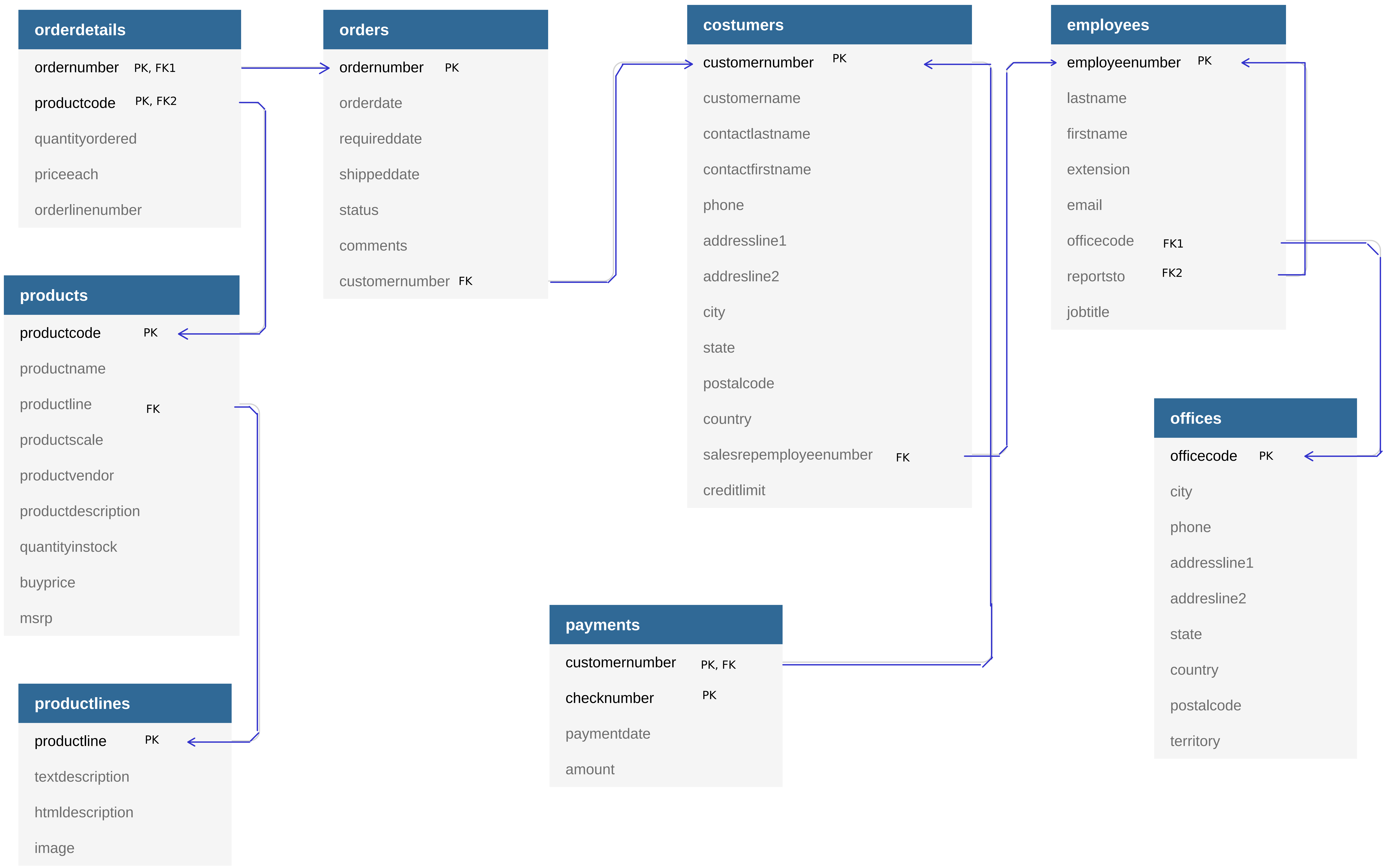
orderdetails(**ordernumber**→orders.ordernumber, **productcode**→products.productcode, quantityordered, priceeach, orderlinenumber)

products(**productcode**, productname, productline→productlines.productline, productscale, productvendor, productdescription, quantityinstock, buyprice, msrp)

productlines(**productline**, textdescription, htmldescription, image)

> Note: as requested, primary keys are in bold and foreign keys are shown as: FK→column they reference

In the following page is the requested database relational schema.



orderdetails	
ordernumber	PK, FK1
productcode	PK, FK2
quantityordered	
priceeach	
orderlinenumber	

orders	
ordernumber	PK
orderdate	
requireddate	
shippeddate	
status	
comments	
customernumber	FK

costumers	
customernumber	PK
customername	
contactlastname	
contactfirstname	
phone	
addressline1	
addresline2	
city	
state	
postalcode	
country	
salesrepemployeenumber	FK
creditlimit	

employees	
employeenumber	PK
lastname	
firstname	
extension	
email	
officecode	FK1
reportsto	FK2
jobtitle	

products	
productcode	PK
productname	
productline	FK
productscale	
productvendor	
productdescription	
quantityinstock	
buyprice	
msrp	

offices	
officecode	PK
city	
phone	
addressline1	
addresline2	
state	
country	
postalcode	
territory	

payments	
customernumber	PK, FK
checknumber	PK
paymentdate	
amount	

productlines	
productline	PK
textdescription	
htmldescription	
image	

Section 3. SQL queries:

1. Here the idea is to join (using the foreign and primary keys of each table) the tables products, orderdetails, oders, customers and payments; so that we can sum the total amount of all of the payments made by each customer (hence the GROUP BY statement) that has, at some point puchased the product “1940 Ford Pickup Truck”.

```
SELECT customernumber,
       customername,
       Sum(amount) AS total
FROM   ((((products
           natural JOIN orderdetails)
          natural JOIN orders)
         natural JOIN customers)
        natural JOIN payments) AS x
WHERE  productname = '1940 Ford Pickup Truck'
GROUP  BY customernumber,
          customername
ORDER  BY total DESC;
```

2. We use the tables productlines, products, orderdetails and orders, in order to link the shippeddate and orderdate (stored in the orders table), with the orders’ productline (by which we have to order). Finally, the average of the difference between order date and shipping date is made for each productline.

```
SELECT productline,
       Avg(shippeddate - orderdate)
FROM   (((productlines
          natural JOIN products)
         natural JOIN orderdetails)
        natural JOIN orders) AS x
GROUP  BY productline;
```

3. First of all we find the director, which is the employee who does not report to any other employee, in this case, in table e3. Then we find all of those who report to him, in table e2; and then display the number and lastname of the employees that report to those who report to the director, in table e1.

```
SELECT e1.employeenumber,
       e1.lastname
FROM   ((employees AS e1
         JOIN employees AS e2
           ON e1.reportsto = e2.employeenumber)
        JOIN employees AS e3
          ON e2.reportsto = e3.employeenumber)
```

```
WHERE  e3.reportsto IS NULL ;
```

4. Here we have to join the offices with the orderdetails (using the tables in between), so that the quantity of every product of every order can be summed for each office, to get how many items have been sold by each office; then we sort the results in descending order and display only the first one.

```
SELECT officecode,
       Sum(quantityordered) AS itemssold
FROM   ((((offices
           NATURAL JOIN employees AS e)
          JOIN customers
            ON salesrepemployeenumber = e.employeenumber)
         NATURAL JOIN orders)
        NATURAL JOIN orderdetails)
GROUP  BY officecode
ORDER  BY itemssold DESC
LIMIT  1;
```

5. Here a nested query was used. We first search for those offices that have sold an item in 2003, by looking for orders with date in 2003, then sum, for each country, the offices that do not appear in the first query (those offices that sold an item in 2003). The results are sorted by descending order.

```
SELECT country,
       Count(officecode) AS noffices
FROM   offices
WHERE  officecode NOT IN (SELECT DISTINCT officecode
                          FROM   (((offices
                                    natural JOIN employees AS e)
                                   JOIN customers
                                     ON salesrepemployeenumber =
                                        e.employeenumber)
                                  natural JOIN orders)
                          WHERE  orderdate > '2002-12-31'
                                 AND orderdate < '2004-01-01')
GROUP  BY country
ORDER  BY noffices DESC;
```

6. We take the cartesian product of the table orderdetails with itself and select all of the pairs that have been purchased in the same order (with same ordernumber); in order to sort out repetitions-(id1, id2) and (id2, id1),-we use the condition that the code of the first product is lower than that of the second. Then we count the amount of times they appear together (each pair).

```
SELECT o1.productcode,
       o2.productcode,
       Count(*)
FROM   orderdetails AS o1,
       orderdetails AS o2
WHERE  o1.ordernumber = o2.ordernumber
       AND o1.productcode < o2.productcode
GROUP  BY o1.productcode,
          o2.productcode;
```

Section 4

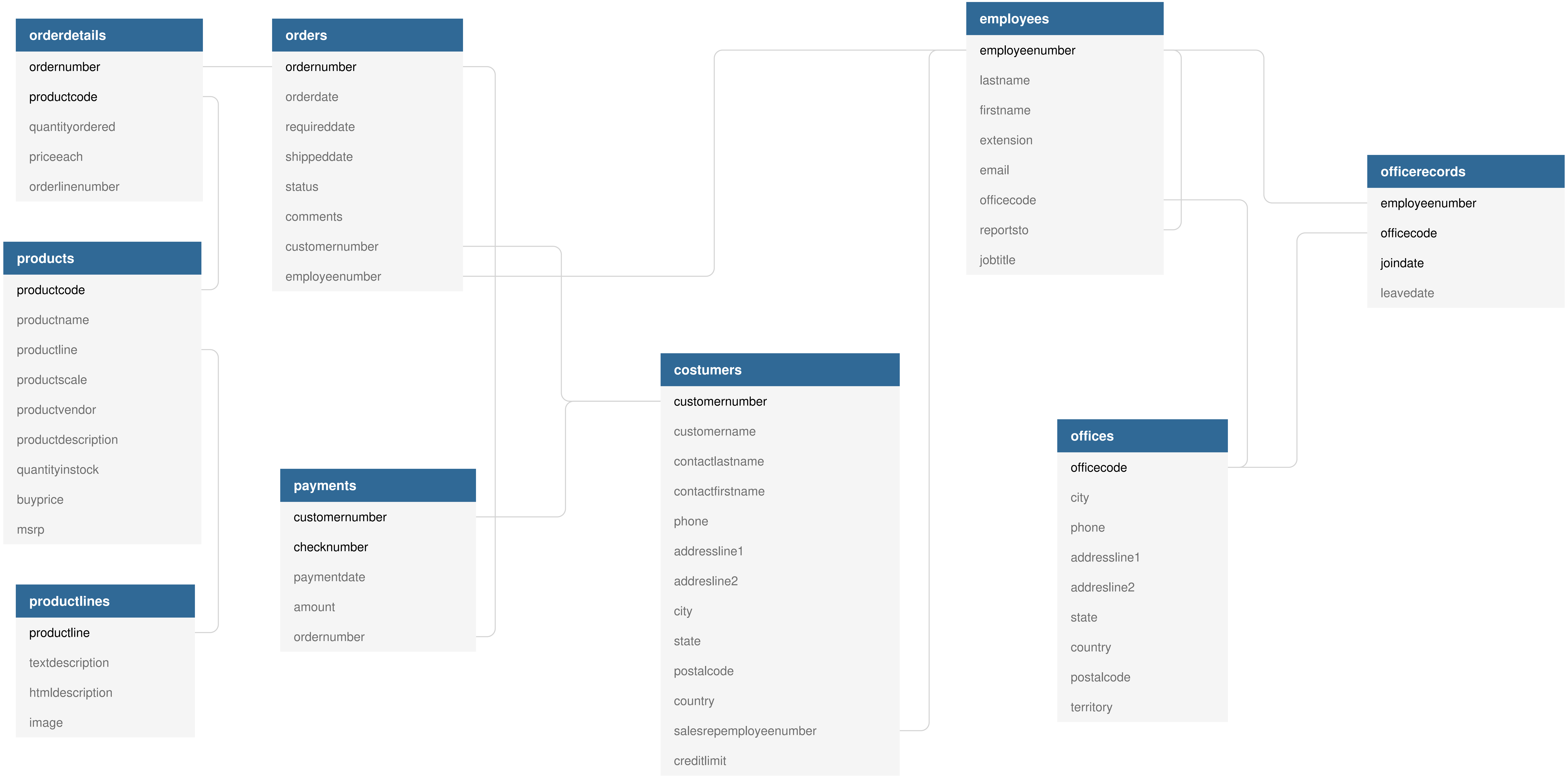
In this section we redesign the database so that the limitations explained in the assignment can be solved. The relational schema of this new design is presented in the next page.

The first issue is that if an employee moves from one office to another, we lose track of the office in which he previously worked. To solve this problem, we have decided it is best to create a new table, called *officerecords*, with four columns: *employeenumber* (FK to employee.employeenumber), *officecode* (FK to office.officode), *joindate* (date in which the employee joins the office) and *leavedate* (date in which the employee leaves the office, this field may be null). As an employee could work in an office, then leave, and come back again; the primary key of this table would be *(employeenumber, officecode, joindate)*. In this new database we would not remove the row *officenumber*, which is the office in which the employee is currently working.

The second problem is that payments are not associated to order. We have decided to include a new field in the *payments* table: *ordernumber*, which is a foreign key to *orders.ordernumber*. We have also thought about the problems that this could carry: perhaps we would have to worry about inconsistencies regarding the amount stored in the payment and the amount in the order; and, about the fact that the tables *orders*, *payments* and *customers* would have three links among each other, which could be a bit redundant. We still have decided not to delete any of this, mainly because it would alter too much the original database, as the FK *payments.customer* is already a PK of this table (payments).

Finally, the third issue is that a customer cannot contact more than one employee. To fix this, we have left untouched the table customer, where the FK *salesrepemployeenumber* refers to the employee originally assigned to the customer, which is the one that the customer has to contact for anything not directly related to an order; but we have added to the table *orders* a FK (*employeenumber*) referencing the table *employees*, where the employee that the customer contacted for that particular order is stored.

We have encountered a couple of problems when executing the newdatabase.sql file. We have not altered the data and therefore, we had to ommit the NOT NULL constraints in the new columns that we have created in order not to get errors because the data that was inserted into the database had null values for those fields.



orderdetails
ordernumber
productcode
quantityordered
priceeach
orderlinenumber

products
productcode
productname
productline
productscale
productvendor
productdescription
quantityinstock
buyprice
msrp

productlines
productline
textdescription
htmldescription
image

orders
ordernumber
orderdate
requireddate
shippeddate
status
comments
customernumber
employeenumber

payments
customernumber
checknumber
paymentdate
amount
ordernumber

costumers
customernumber
customername
contactlastname
contactfirstname
phone
addressline1
addresline2
city
state
postalcode
country
salesrepemployeenumber
creditlimit

employees
employeenumber
lastname
firstname
extension
email
officecode
reportsto
jobtitle

offices
officecode
city
phone
addressline1
addresline2
state
country
postalcode
territory

officerecords
employeenumber
officecode
joindate
leavedate