



Informe de Prácticas de Fundamentos de Bases de Datos

Rodríguez Marrero, David
Larriba Moreno, Sergio

Índice

• Objetivos	2
• PK's y FK's	3
• Modelo E/R de la BDD	4
• Diagrama relacional de la BDD	5
• Consultas	
– Consulta 1	6
– Consulta 2	7
– Consulta 3	8
– Consulta 4	9
– Consulta 5	10
– Consulta 6	11-12
• Rediseño de la BDD	13-14
• Conclusiones	15

Objetivos

Los objetivos que hemos tenido en esta práctica han sido muy variados.

En primer lugar, aprender a manejar una base de datos con `psql` desde la consola ya que sólo habíamos visto bases de datos en aplicaciones que las hacen visibles gráficamente.

En segundo lugar, mejorar nuestros conocimientos en programación de sentencias SQL utilizando `joins`, `counts`, `sums`, subgrupos con la sintaxis `with` y muchas más funciones que hemos aprendido en clases teóricas y hemos podido poner en práctica.

Por último, aprender a base de ensayo y error, poco a poco conociendo la base de datos y al final poder predecir las salidas a nuestras consultas.

En esta práctica, hemos utilizado un Makefile que ejecuta las queries e imprime en pantalla con una estructura de tabla el resultado esperado. También hemos utilizado la base de datos `classicmodels.sql` en la que hemos hecho todas las consultas.

Al final de la práctica, entregaremos todos los códigos de las consultas y el Makefile que hemos utilizado prácticamente igual al dado en Moodle. También entregaremos este informe en el cual hemos redactado nuestros pensamientos durante toda la práctica.

PK's y FK's

1,2

PK

productline
- productscale
- productvendor
- productdescription
- quantityinstock
- buyprice
- msrcp

- customers (customer number, salesrepemployeenumber → employees. employeenumber, customername, contactlastname, contactfirstname, phone, addressline1, addressline2, city, state, postalcode, country, creditlimit).
- employees (employeenumber, reports to → employees. employeenumber, officecode → offices. officecode, lastname, firstname, extension, email, jobtitle)
- offices (officecode, city, phone, addressline1, addressline2, state, country, postalcode, territory)
- orderdetails (order number, productcode, order number → orders. order number, productcode → products. productcode, quantityordered, priceeach, orderline number)
- orders (order number, customer number → customers. customer number, orderdate, requireddate, shippeddate, status, comments)
- payments (customer number, check number, customer number → customers. customer number, paymentdate, amount)
- productlines (productline, textdescription, htmldescription, image)
- products (productcode, productline → productlines. productline, productname, productscale, productvendor, productdescription, quantityinstock, buyprice, msrcp)

Modelo E/R de la BDD

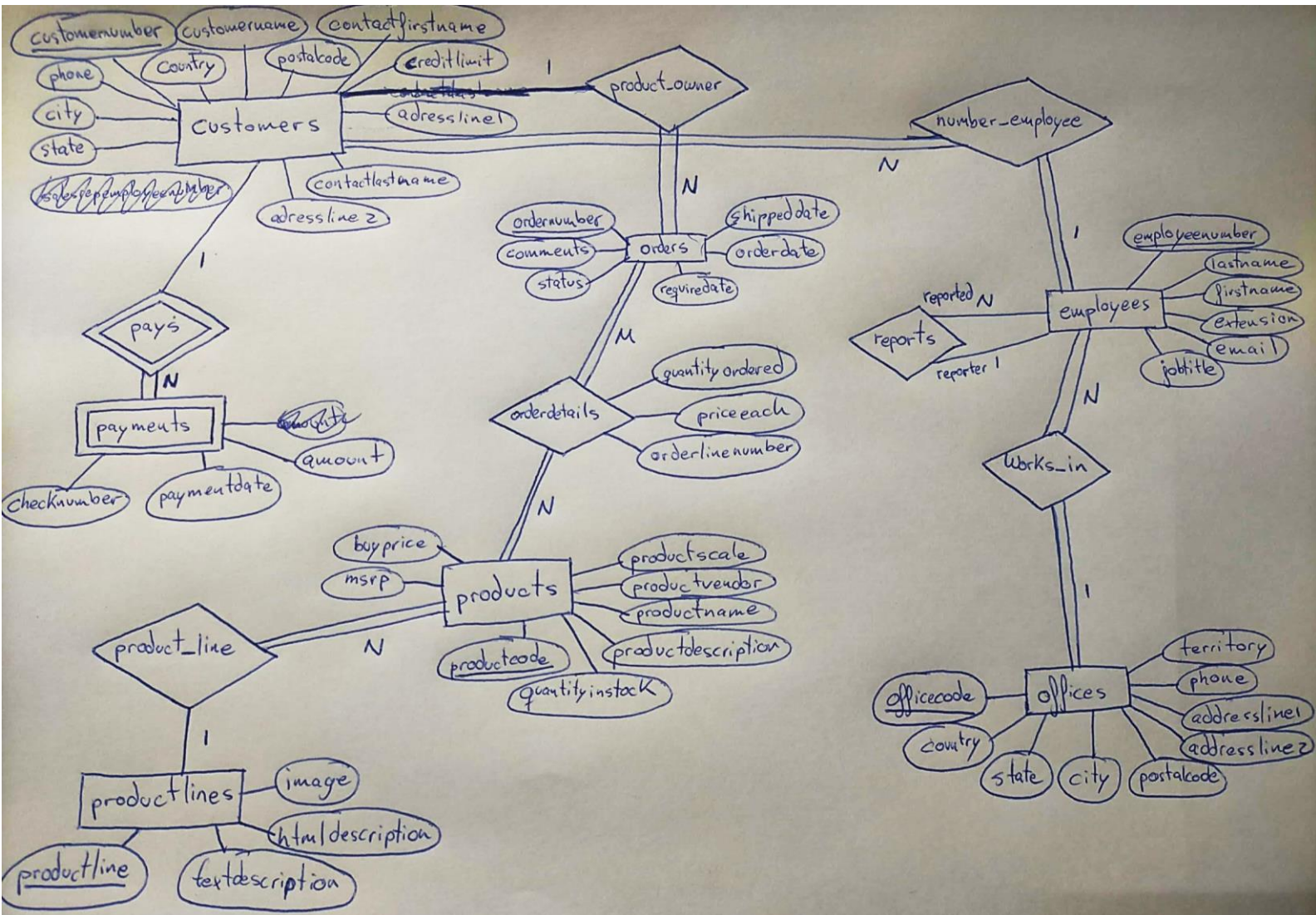
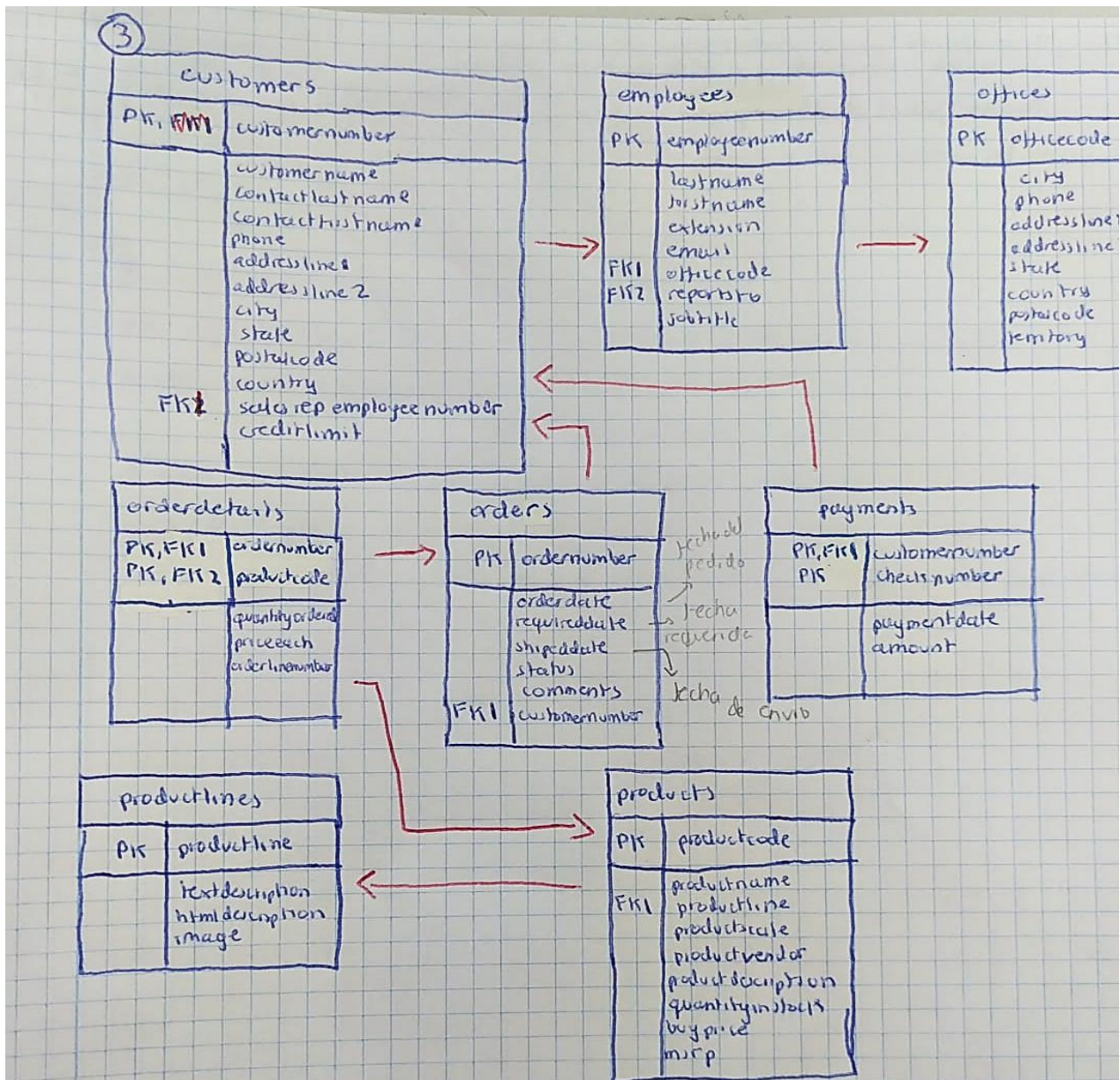


Diagrama relacional de la BDD



1º Hago una tabla con los productos y cuantos he vendido

Problemas de esta base de datos 1 - si un empleado se mueve de una oficina a otra se pierde las oficinas en las que había trabajado antes

2 - un cliente solo puede relacionarse con un unico empleado

3 - los pagos no estan asociados a una compra en concreto

Consulta 1

En este ejercicio tenemos que mostrar la cantidad total de dinero abonado por los clientes que hayan comprado el producto “1940 Ford Pickup Truck”, ordenar el resultado de mayor a menor por la cantidad de dinero abonado y dividir la tabla en “customernumber”, “customername” y la cantidad de dinero. Esto es nuestro código y salida resultante:

```
SELECT
    customers.customernumber,customers.customername,sum(payments.amount)
FROM
    products natural join orderdetails natural join orders natural join customers
    natural join payments
WHERE
    products.productname = '1940 Ford Pickup Truck'
GROUP BY
    customers.customernumber
ORDER BY
    sum(payments.amount) desc
```

```
eps@vmlabsdocentes:~/Escritorio/PracticasFundBB-main/Practicas$ make query1
query-1: Muestra la cantidad total de dinero abonado por los clientes que han adquirido el "1940 Ford Pickup Truck"
customernumber |      customername      |      sum
-----+-----+-----
124 | Mini Gifts Distributors Ltd. | 1752564.72
141 | Euro+ Shopping Channel      | 715738.98
276 | Annas Decorations, Ltd      | 274068.44
114 | Australian Collectors, Co.   | 180585.07
148 | Dragon Souvenirs, Ltd.       | 156251.03
321 | Corporate Gift Ideas Co.     | 132340.78
146 | Saveley & Henriot, Co.       | 130305.35
363 | Online Diecast Creations Co. | 116449.29
458 | Corrida Auto Replicas, Ltd   | 112440.09
398 | Tokyo Collectables, Ltd      | 105548.73
161 | Technics Stores Inc.         | 104545.22
121 | Baane Mini Imports           | 104224.79
282 | Souvenirs And Things Co.     | 91655.61
249 | Amica Models & Co.            | 82223.23
448 | Scandinavian Gift Ideas     | 76776.44
455 | Super Scale Inc.             | 70378.65
202 | Canadian Gift Exchange Network | 70122.19
233 | Quebec Home Shopping Network | 68977.67
129 | Mini Wheels Co.              | 66710.56
256 | Auto Associates & Cie.         | 58876.41
339 | Classic Gift Ideas, Inc       | 57939.34
333 | Australian Gift Network, Co   | 55190.16
452 | Mini Auto Werke              | 51059.99
475 | West Coast Collectables Co.   | 43748.72
173 | Cambridge Collectables Co.    | 32198.69
(25 rows)
```

Lo primero en lo que hemos pensado ha sido que teníamos que poner una condición en el nombre del producto para localizar a sus compradores, después los hemos ordenado de mayor a menor. Sin embargo, hemos tenido problemas con este ejercicio ya que al principio entendimos que había que enseñar la cantidad de dinero por la que compraron el producto por un fallo de lectura, sin embargo, después de leer la práctica con más detenimiento y comparar algunos resultados con otros compañeros, nos dimos cuenta de nuestro error y pudimos modificar la sentencia fácilmente para que enseñase la suma del dinero gastado en todos los productos, es decir, la cantidad de dinero abonada de cada cliente. Con esta sentencia pudimos ver que había un nivel medio-elevado de dificultad en base a nuestros conocimientos en esa etapa.

Consulta 2

En este ejercicio tenemos que mostrar el tiempo medio transcurrido entre que se realiza un pedido y se envía y agrupar las filas por la línea de producto. Hay que tener cuidado con el tiempo medio ya que es el average de la diferencia entre el día de envío y el día de pedido y probablemente dará un resultado decimal.

```
SELECT
    p.productline, Avg(o.shippeddate - o.orderdate) as dif
FROM
    orders o join orderdetails orderdi on o.ordernumber=orderdi.ordernumber
    join products p on p.productcode=orderdi.productcode
GROUP BY
    p.productline
ORDER BY
    dif desc
```

```
eps@vmlabsdocentes:~/Escritorio/PracticasFundBB-main/Practicas1$ make query2
query-2: Tiempo medio transcurrido entre que se realiza un pedido (orderdate) y se envía el pedido (shippeddate) agrupado por tipo de producto
 productline |      dif
-----+-----
Trains       | 5.7500000000000000
Trucks and Buses | 4.4470989761092150
Classic Cars  | 3.9611054247697032
Motorcycles   | 3.9078212290502793
Planes        | 3.8761904761904762
Vintage Cars  | 3.5177993527508091
Ships         | 3.4816513761467890
(7 rows)
```

Lo primero que hemos hecho son los “join” entre las distintas tablas para poder relacionar una línea de producto con los días de pedido y envío. Después hemos usado la función avg que calcula la media de lo que tenga en el argumento, en este caso es la diferencia entre el día de envío y el día de pedido teniendo así el tiempo que tarda en enviarse y lo llamamos dif para que aparezca así en la tabla resultante. Por otra parte, como nos devolverá una tabla gigante con valores repetidos, agrupamos por la línea de producto y ordenamos toda la tabla de mayor a menor en función al tiempo medio que tarde.

Vemos unos resultados muy parejos, pero sacamos como conclusión que los trenes son los productos que más tardan en enviarse.

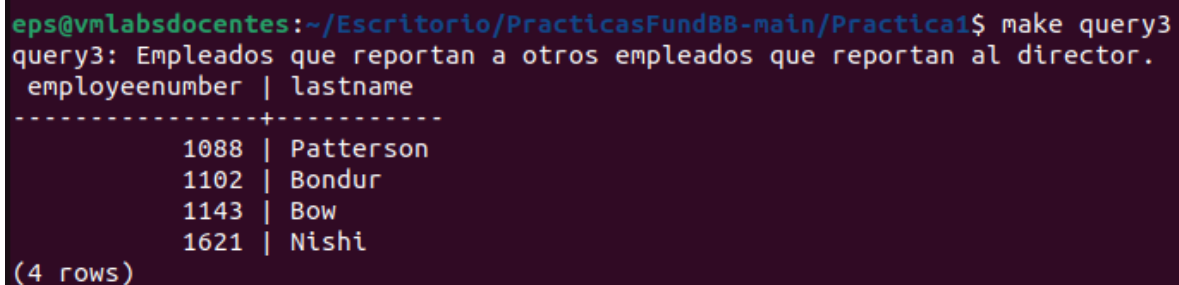
Por otra parte, hemos tenido problemas con esta consulta ya que hacíamos una simple suma entre todos los tiempos de cada producto y nos daba un valor muy elevado que no tenía sentido con lo que nos pedían. Nuestro fallo era que no sabíamos la existencia de la función “avg” que aplicándoselo al tiempo nos devuelve el tiempo medio de todos los productos.

Consulta 3

En este ejercicio nos piden enseñar a los empleados que reportan a otros empleados que a su vez reportan al director. Podremos diferenciar a este último porque no reporta a nadie por lo tanto ese campo en sus datos será NULL. Por último, nos pide que imprimamos el número de empleado y el apellido de las personas que pertenezcan a ese subconjunto.

```
with director as (select employees.employeenumber from employees where
employees.reportsto IS NULL),
    report_group as (select employees.employeenumber from employees where
employees.reportsto in (select * from director))

SELECT
    employees.employeenumber, employees.lastname
FROM
    employees
WHERE
    employees.reportsto IN (select * from report_group)
```



```
eps@vmlabsdocentes:~/Escritorio/Practicafund88-main/Practica1$ make query3
query3: Empleados que reportan a otros empleados que reportan al director.
employeenumber | lastname
-----+-----
          1088 | Patterson
          1102 | Bondur
          1143 | Bow
          1621 | Nishi
(4 rows)
```

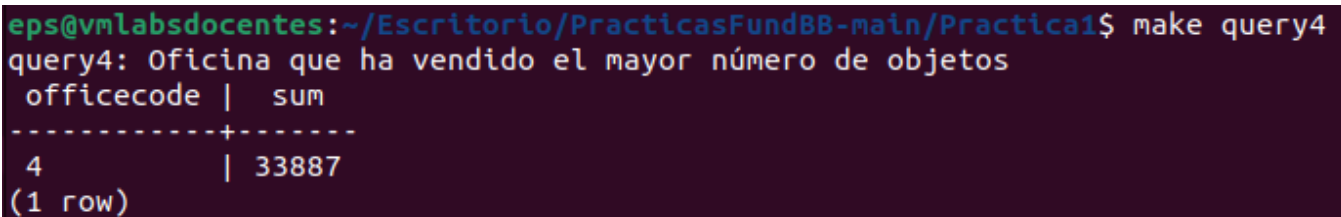
Hemos empezado creando dos subgrupos, el primero denota quién es el director buscando al empleado que no reporta a nadie, por lo tanto, que ese campo sea NULL. El segundo subgrupo contiene a los empleados que reportan al director (usando el primer subgrupo). Finalmente ponemos una condición para que muestre el número de empleado y el apellido de los empleados que reportan a cualquier persona del segundo subgrupo (que eran los empleados que reportaban al director).

Este ejercicio lo hemos ejecutado a la primera ya que el uso de los subgrupos ha facilitado bastante el ejercicio usando una estructura de código muy organizada parecida a la usada en algunos otros lenguajes de programación como C, Python, Js,

Consulta 4

En este apartado nos piden enseñar la oficina que ha vendido el mayor número de objetos, teniendo en cuenta que cada objeto se puede pedir varias veces dentro de un pedido. Por último, nos piden estructurar esta tabla resultante en una columna que enseñe el código de la oficina y el número total de productos vendidos. El resultado suponemos que dará el código de una sola oficina ya que sería la que más ha vendido.

```
SELECT
    o.officecode, sum(orderdi.quantityordered)
FROM
    orderdetails orderdi natural join orders natural join customers c join employees e natural
    join offices o on c.salesrepemployeenumber=e.employeenumber
GROUP BY
    o.officecode
ORDER BY
    sum(orderdi.quantityordered) desc limit 1
```



```
eps@vmlabsdocentes:~/Escritorio/PracticasFundBB-main/Practica1$ make query4
query4: Oficina que ha vendido el mayor número de objetos
officecode | sum
-----+-----
4          | 33887
(1 row)
```

Como podemos ver, nuestra salida muestra la oficina número 4, y que es la que más unidades ha vendido de todas las oficinas. El primer paso que hemos seguido en la ejecución de este apartado ha sido fijarnos en nuestro diagrama relacional para ver como podíamos relacionar la cantidad de productos pedidos con las oficinas. Por lo tanto, tuvimos que conectar varias tablas con “natural join”, la cual crea una tabla grande con datos de las dos relacionadas, y “join” de una columna que tiene los mismos datos en las dos tablas por lo que se sobrescribe en la nueva tabla. Después de esto, debíamos tener claro qué vamos a imprimir y si queríamos ver la cantidad de unidades vendidas primero teníamos que unir todas las cantidades de cada producto por cada oficina y sumar todas esas cantidades diferentes para agruparlas en solamente un número con su número de oficina correspondiente. Finalmente ordenamos de mayor a menor la cantidad de unidades y como nos piden la oficina que más ha vendido ponemos un “limit 1” que hace que muestre la primera fila ya que al ordenar de mayor a menor ahí es donde va a estar la oficina que más productos ha vendido. Hemos tenido algunos problemas con este apartado ya que las tablas de oficinas y “orderdetails”, la cual contiene la cantidad del pedido, están muy poco comunicadas y no sabíamos como relacionarlas hasta que nos fijamos bien en las claves extranjeras relacionando así con joins.

Consulta 5

En esta consulta nos piden que imprimamos una tabla que contenga el nombre del país y el número de oficinas en él las cuales no han vendido nada durante el año 2003 y ordenar la tabla según el número de oficinas que coincidan con la búsqueda. Pensamos que tendremos que hacer una restricción la cual solo analice los pedidos entre el 1 de Enero de 2003 y el 31 de Diciembre de 2003.

```
with no_sale_offices as
(select o.officecode from offices o natural join employees natural join customers natural join
orders ord where ord.shippeddate is NULL and ord.orderdate>='2003-01-01' and
ord.orderdate<='2003-12-31')
SELECT
    o.country, count(o.officecode) as offices_with_no_sales
FROM
    offices o
WHERE
    o.officecode in (select * from no_sale_offices)
GROUP BY
    o.country
HAVING
    count(o.officecode)>=1
ORDER BY
    count(o.officecode) desc
```

```
eps@vmlabsdocentes:~/Escritorio/PracticasFundBB-main/Practica1$ make query5
query5: Países que tienen al menos una oficina que no ha vendido nada durante el año 2003
country | offices_with_no_sales
-----+-----
(0 rows)
```

Lo primero que hemos hecho ha sido un subgrupo “no_sale_offices” en el que analizamos y metemos a las oficinas las cuales no han vendido productos desde el 1 de Enero de 2003 hasta el 31 de Diciembre de ese mismo año. Esto lo conseguimos restringiendo el día de pedido en esas fechas y obligando a que no haya día de envío, es decir, que este sea NULL. A partir de ahí es fácil imprimir a los países que contengan a esas oficinas, pero hay que indicar el número de oficinas que no han vendido nada, no su código, por lo tanto, utilizamos la función “count” que cuenta el número de oficinas que aparecen por cada país y devuelve ese número. Por último, nos pedían que indiquemos los países que tienen al menos una oficina que cumpla los requisitos anteriormente nombrados, por lo tanto, usamos el campo having para poner una restricción del “count” y es que sea mayor o igual que 1, que es el significado de al menos 1, y ordenamos esos números impresos de mayor a menor. Finalmente vemos que la salida en consola es vacía por lo tanto no hay ninguna oficina de ningún país en nuestra base de datos que no haya vendido nada durante el 2003. Esta sentencia nos ha resultado sencilla ayudándonos del subgrupo que nos permite realizar las conexiones de tablas con “join” fuera del código y lo hace más ordenado.

Consulta 6

En este último ejercicio nos piden un listado de parejas de productos que aparezcan en más de un pedido a la vez, enseñar los códigos de cada producto y el número de pedidos en los que coinciden. También nos dicen que tendremos que considerar que el orden aparición de los productos en el pedido no influye y considerar iguales el caso en el que 1 salga primero y 2 después con el caso de que 2 salga primero y 1 después. Pensamos que también habrá que tener en cuenta los productos repetidos, por lo que habrá muchas restricciones en el código.

```
SELECT
    id1.productcode as id1, id2.productcode as id2, count(id1.ordernumber) as carros
FROM
    orderdetails id1 join orderdetails id2 on id1.ordernumber=id2.ordernumber
WHERE
    id1.productcode <> id2.productcode and id1.productcode < id2.productcode
GROUP BY
    id1.productcode, id2.productcode
HAVING
    count(id1.ordernumber) > 1
ORDER BY
    id1.productcode, id2.productcode
```

```
eps@vmlabsdocentes:~/Escritorio/PracticasFundBB-main/Practica1$ make query6
query6: Se desea un listado de todas las parejas de productos que aparezcan en más de un carro de la compra
```

id1	id2	carros
S10_1678	S10_2016	20
S10_1678	S10_4698	22
S10_1678	S12_1099	6
S10_1678	S12_2823	22
S10_1678	S12_3380	3
S10_1678	S12_3990	5
S10_1678	S12_4675	3
S10_1678	S18_1889	2
S10_1678	S18_2581	6
S10_1678	S18_2625	18
S10_1678	S18_3278	4
S10_1678	S18_3482	2
S10_1678	S18_3782	6
S10_1678	S18_4721	4
S10_1678	S24_1578	23
S10_1678	S24_1785	4
S10_1678	S24_2000	16
S10_1678	S24_2360	12
S10_1678	S24_3371	2
S10_1678	S24_3949	3
S10_1678	S24_4278	7
S10_1678	S24_4620	4
S10_1678	S32_1374	12
S10_1678	S32_2206	8
S10_1678	S32_4289	6
S10_1678	S32_4485	12
S10_1678	S50_1341	6
S10_1678	S50_4713	12
S10_1678	S700_1691	6
S10_1678	S700_2466	4
S10_1678	S700_2834	11
S10_1678	S700_3167	6
S10_1678	S700_4002	3
S10_1678	S72_1253	2
S10_1949	S10_4962	15
S10_1949	S12_1666	21
S10_1949	S18_1097	25
S10_1949	S18_1342	7
S10_1949	S18_1367	10
S10_1949	S18_1749	2
S10_1949	S18_2248	2
S10_1949	S18_2325	4
S10_1949	S18_2432	10
S10_1949	S18_2795	5
S10_1949	S18_2949	22

S10_1949	S18_3136	18
S10_1949	S18_3320	12
S10_1949	S18_4600	12
S10_1949	S18_4668	18
S10_1949	S24_1937	5
S10_1949	S24_2022	5
S10_1949	S24_2300	5
S10_1949	S24_3969	2
S10_1949	S24_4258	12
S10_1949	S32_1268	8
S10_1949	S32_3522	15
S10_1949	S700_2824	14
S10_2016	S10_4698	24
S10_2016	S12_1099	3
S10_2016	S12_2823	14
S10_2016	S12_3380	2
S10_2016	S12_3990	3
S10_2016	S12_4675	2
S10_2016	S18_1889	2
S10_2016	S18_2581	11
S10_2016	S18_2625	26
S10_2016	S18_3278	3
S10_2016	S18_3782	2
S10_2016	S18_4721	2
S10_2016	S24_1578	21
S10_2016	S24_1785	9
S10_2016	S24_2000	20
S10_2016	S24_2360	4
S10_2016	S24_3420	2
S10_2016	S24_3949	6
S10_2016	S24_4278	9
S10_2016	S24_4620	2
S10_2016	S32_1374	14
S10_2016	S32_2206	4
S10_2016	S32_4289	8
S10_2016	S32_4485	4
S10_2016	S50_1341	8
S10_2016	S50_4713	4
S10_2016	S700_1691	8
S10_2016	S700_2047	2
S10_2016	S700_2466	6
S10_2016	S700_2834	13
S10_2016	S700_3167	7
S10_2016	S700_4002	6
S10_2016	S72_1253	3
S10_4698	S12_1099	3
S10_4698	S12_2823	18
S10_4698	S12_3380	2
S10_4698	S12_3990	3

Hemos cortado mitad de la salida ya que era muy larga, pero podemos ver claramente las dos columnas que en cada fila contienen las parejas de productos analizadas.

En primer lugar, hemos creado dos índices distintos que apuntan a la tabla “orderdetails” para distinguir cada elemento de la pareja. En segundo lugar, hemos empezado con las restricciones con el símbolo “< >” el cual nos ayuda con el problema comentado al principio y que el comparador “!=” no soluciona, por otro lado, hemos puesto que el producto 1 sea menor que el 2 para evitar repeticiones y relacionar los productos con más orden. Después, hemos agrupado los elementos de la tabla en función de las parejas para agrupar finalmente también los pedidos en los que aparecen contándolos con un “count” y poniendo una restricción de “having mayor que 1” para que muestre solo parejas que aparezcan en más de un pedido.

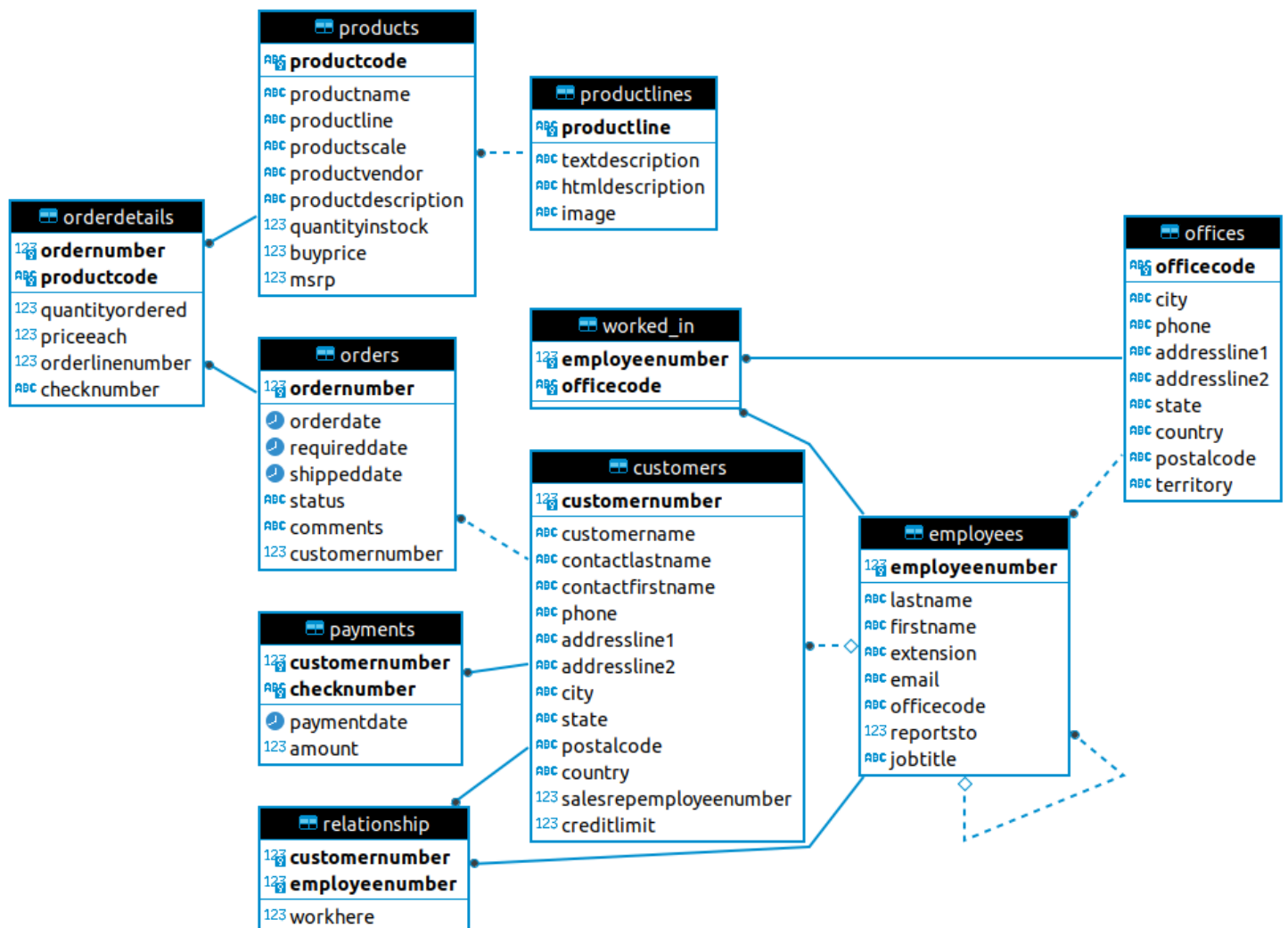
El problema que hemos tenido con este ejercicio fue la restricción de elementos distintos en la pareja ya que no sabíamos como no tener en cuenta el orden para determinar si las parejas eran distintas o no. Primero usamos solo la restricción “!=” pero no hacía esa función ya que veía distintos los dos casos.

Rediseño de la BDD

Casi al final de la práctica, nos piden que rediseñemos esta base de datos para mejorarla y nos nombra tres problemas que principalmente tendremos que solucionar:

1. Si un empleado se mueve de una oficina a otra se pierde la información de la oficina donde había trabajado antiguamente.
2. Un cliente solamente puede relacionarse con un único empleado.
3. Los pagos no están relacionados con una compra.

Analizando estos problemas hemos podido crear una nueva base de datos llamada “nuevabase.sql” en la que todos estos problemas están solucionados. Por otra parte, también hemos modificado el Makefile para que borre la antigua base de datos y cree nuestra nueva base. Seguidamente añado una imagen del diagrama relacional de la nueva base hecho con la aplicación Dbeaver y otra del makefile editado.



```
#remove logs: rm -rf *.log

nuevabase: dropdb2 createddb2 restore2 shell12

createdb2:
    @echo Creando BBDD
    @$(CREATEDB)

dropdb2:
    @echo Creando BBDD
    @$(DROPDB) $(DBNAME)
    @$(DROPDB) $(DBNAME2)
    rm -f *.log

restore2:
    @echo restore data base
    @cat $(DBNAME2).sql | $(PSQL)

dump2:
    @echo creando dumpfile
    @$(PG_DUMP) > $(DBNAME).sql

shell:
    @echo create psql shell
    @$(PSQL)
```

Como podemos ver en el diagrama, con nuestra nueva base de datos tenemos más relacionadas todas las tablas evitando hacer joins, factor que nos ha complicado el entendimiento de algunas consultas encomendadas.

Desde nuestro punto de vista ahora la base de datos está más organizada y a parte de poder tener un entendimiento mayor sobre ella, podemos hacer sentencias más cortas para enseñar el mismo resultado que en la antigua base de datos, por lo tanto, ganamos rapidez y sencillez en nuestro diseño.

Fijándonos ahora en el makefile, hemos añadido esas líneas al anterior makefile las cuales tienen la misma función que con la primera base solo que en este caso el dropdb2 también borra la primera base si existe y el dump2 es referido a la antigua base ya que es lo que queremos actualizar.

Conclusiones

Las conclusiones que hemos sacado de esta práctica son las siguientes:

- Los “join” son una herramienta imprescindible a la hora de mezclar salidas de distintas tablas y puede llegar a ser muy difícil y enrevesado conectarlas por su poca y lejana relación teniendo que pasar entre distintas tablas de la base de datos para poder llegar a la deseada.
- Ponerle índices como “apodos” refiriéndonos a una tabla es muy útil para acortar código, mejorar la estructura de este y crear sentencias más rápido. También, como hemos visto en la consulta 6 nos ayudan a extraer dos elementos de una misma tabla y referirnos a ellos con un nombre identificativo para nosotros que podemos modificar a nuestro gusto.
- Los subgrupos son muy útiles a la hora de organizar una consulta y poder tener las ideas más claras con lo que queremos imprimir finalmente. También nos permita “desahogar” el código principal y hacer uno mucho más estético y sobre todo más entendible por alguien que no tenga mucho conocimiento sobre SQL. Estos subgrupos también nos permiten crear una especie de tabla en nuestra cabeza y poder predecir la salida y entender mejor el funcionamiento de nuestra sentencia imaginándonos esa tabla de la que podemos sacar la información que vamos a imprimir.
- Las funciones “count”, “sum”, “avg” son herramientas que complementándose con el “group by” nos ayudan, a parte de a contar, sumar, y calcular la media, a agrupar esos campos contables y poder rebajar el tamaño de la tabla final sin tener elementos repetidos. Con esto podemos indicar fácilmente qué es lo importante para nosotros en la tabla resultante y dar una respuesta concisa y procesable.
- Las restricciones son lo más visto en las sentencias SQL y es muy importante crear una restricción útil y rápidamente procesable por la máquina. Estas nos permiten analizar directamente la sección que queramos de nuestros datos, como si se tratase de un subgrupo, pero mucho más rápido y con sentencias mas cortas y entendibles. Hemos podido ver las distintas restricciones que hay y cómo algunas, a parte de crear un subgrupo, nos cambia la tabla de salida desechando algunas opciones que no cumplen la condición como hemos usado en la consulta 4 el “limit 1” restricción la cual ya creada la tabla final sólo se queda con la primera fila, da igual su valor.

En conclusión, nos hemos adentrado en SQL mucho más pudiendo poner en práctica lo aprendido en clases de teoría y adquiriendo más conocimiento y experiencia a base de éxito y error. También hemos comprendido la similitud que tiene este lenguaje con otros de programación a la hora de tener que organizar el código para poder tener una idea clara e incluso predecir el resultado final. A la vez, hemos entendido la lejanía de este lenguaje con otros debido a su manera de crear subgrupos imaginarios y usar sentencias que no se usan en ningún otro lenguaje como “join” y las restricciones de tablas finales como “limit” o “having”. Conjuntamente nos ha encantado esta práctica y esperamos poder hacer más.