PRÁCTICA 1 EDAT 1262

Pareja nº : 6

David Teofilo Garitagoitia Romero
Diego Grande Camarero

2.1.+2.2. Claves primarias y extranjeras de cada tabla.

customers(**customernumber**, customername, contactlastname, contactfirstname, phone, addresline1, addresline2, city, state, postalcode, country, salesrepemployeenumber->employees.employeenumber, creditlimit)

employees(**employeenumber**, lastname, firstname, extension, email, officecode->offices.officode, reportsto->employe.employeenumber, jobtitle)

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

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

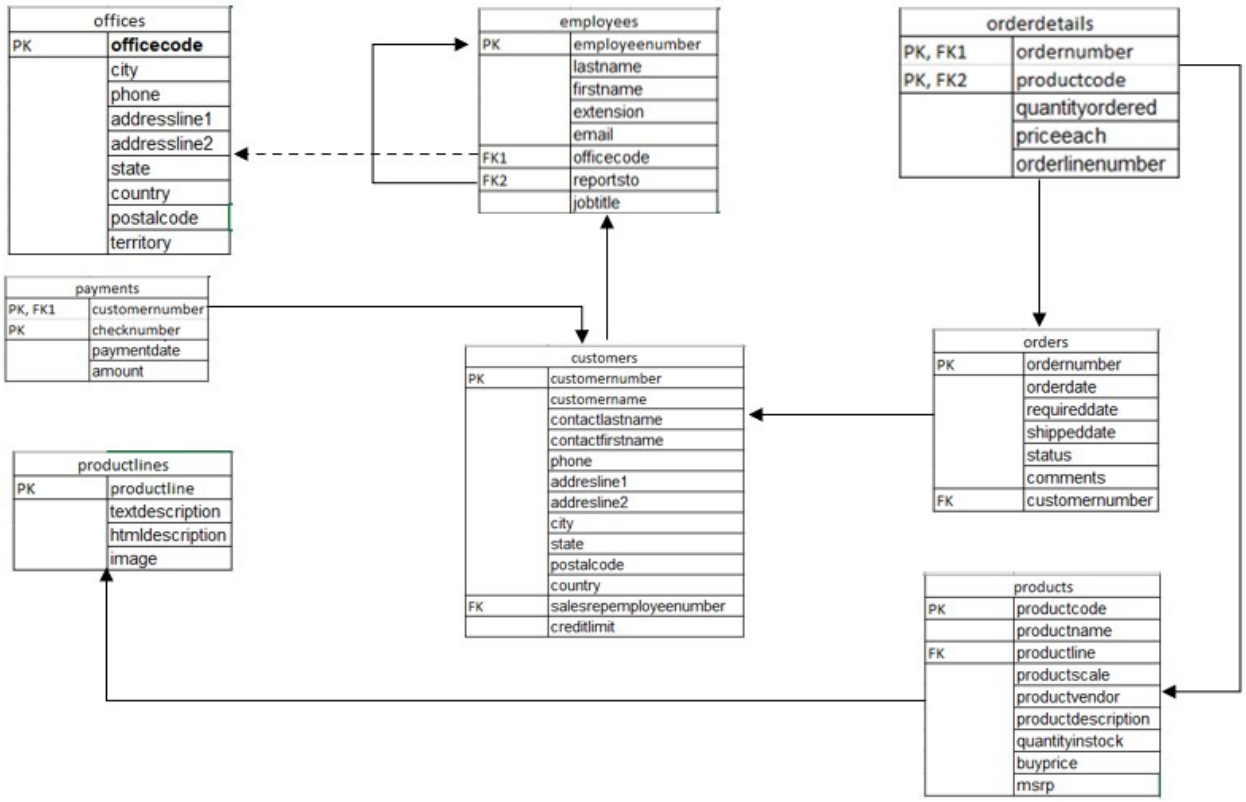
orders(**ordernumber**, orderdate, requireddate, shippeddate, status, comments, customernumber->customers.customernumber)

payments(**customernumber->customers.customernumber, checknumber**, paymentdate, amount)

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

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

2.3. Diagrama del modelo relacional de la base de datos implementada.



3. Consultas:

3.1. Muestra la cantidad total de dinero abonado por los clientes que han adquirido el “1940 Ford Pickup Truck” (el dinero puede haber sido abonado para comprar otros modelos). Ordena el resultado por la cantidad de dinero abonada de mayor a menor cantidad. Cada línea debe mostrar: “customernumber”, “customername” y la cantidad total [“total_paid”] de dinero pagada.

Tras un primer intento sin anidamineto (concatenando el nombre del producto al primer WHERE) y comparar consultar otros .log, vimos que algo fallaba.

Después de estudiar el código, se decidió anidarlo. Comprobándolo por separado desde la consulta más interna, sumándola luego a la primera consulta.

```
SELECT CS.customernumber, CS.customername, Sum(D.quantityordered * D.priceeach)
AS total_paid
FROM customers CS, orders O, orderdetails D, products PR
WHERE PR.productcode = D.productcode
      AND D.ordernumber = O.ordernumber
      AND O.customernumber = CS.customernumber
      AND CS.customernumber IN(

      SELECT CS.customernumber
      FROM customers CS, orders O, orderdetails D, products PR
      WHERE PR.productname = '1940 Ford Pickup Truck'
            AND PR.productcode = D.productcode
            AND D.ordernumber = O.ordernumber
            AND O.customernumber = CS.customernumber)

GROUP BY CS.customernumber
ORDER BY total_paid DESC;
```

3.2. Tiempo medio transcurrido entre que se realiza un pedido (orderdate) y se envía el pedido (shippeddate) agrupado por tipo de producto ("productline"). Cada línea debe mostrar el "productline" y el tiempo medio correspondiente.

Salvo el incidente con la pérdida de precisión decimal debida a no usar la funcón AVG, si no a calcular manualmente la media (SUM("diferencias")/ COUNT ("total")), esta consulta se hizo prácticamente sola.

```
SELECT L.productline, AVG(O.shippeddate - O.orderdate) AS delivering_time
FROM ORDERS O, ORDERDETAILS D, PRODUCTS PR, PRODUCTLINES L
WHERE O.ordernumber = D.ordernumber
      AND D.productcode = PR.productcode
      AND PR.productline = L.productline

GROUP BY L.productline
ORDER BY delivering_time ASC;
```

3.3. Empleados que reportan a otros empleados que reportan al director. El director es aquella persona que no reporta a nadie. El listado debe mostrar el "employeenumber" y el "lastname".

Esta consulta se conceptualizó inicalmente en base a la jerarquía de la propia empresa, pero al contrario de como se presenta en el enunciado: Desde el director hacia abajo.

A la hora de redactarla se siguió el orden normal.

Hizo falta crear duplicados de la tabla employees para poder relacionarla consigo misma.

```
SELECT e.employeenumber, e.lastname
FROM employees e, employees e_1, employees e_2
--e reporta a e_1 que reporta a e_2 que es el jefe
WHERE e.reportsto=e_1.employeenumber AND e_1.reportsto=e_2.employeenumber AND
e_2.reportsto IS NULL;

WHERE e.reportsto IS NULL AND e.employeenumber=e_1.reportsto AND
e_1.employeenumber=e_2.reportsto;
```

3.4. Oficina que ha vendido el mayor número de objetos. Nota: en un pedido ("order") se puede vender más de una unidad de cada producto, cada unidad se considerará un objeto. La salida debe mostrar el "officecode" y el número de productos vendidos.

En esta consulta se hizo la comparación de claves de manera normal, siguiendo el diagrama del MR sin mayores complicaciones.

```
SELECT OFC.officecode, SUM(OD.quantityordered) AS SUM_ORDER
FROM offices OFC, employees E, customers CS, orders O, orderdetails OD
WHERE OD.ordernumber=O.ordernumber
      AND O.customernumber=CS.customernumber
      AND CS.salesrepemployeenumber = E.employeenumber
      AND E.officecode=OFC.officecode

GROUP BY OFC.officecode
order BY SUM(OD.quantityordered) DESC LIMIT 1;
```

3.5. Países que tienen al menos una oficina que no ha vendido nada durante el año 2003. La salida debe mostrar dos columnas conteniendo el nombre del país y el número de oficinas que no han realizado ninguna venta. Ordena las salidas por el número de oficinas de forma que la primera línea muestre el país con más oficinas que no han realizado ninguna venta.

En esta consulta se tuvieron que realizar múltiples selecciones. Para comprobar errores, trabajamos de dentro a fuera, como en la primera.

Hubo un primer planteamiento que idealmente realizaría un bucle menos (mayor eficiencia computacional) que consistía en seleccionar las officinas (OFFICECODE, de manera similar al 2º bucle actual) cuyos productos vendidos en el intervalo 1-1-2003 ,31-12-2003 hubiera sido 0, pero no nos dio ninguna, así que lo tomamos por un error.

Más adelante se utilizó la lógica inversa: coger las que sí habían vendido para posteriormente coger TODAS LAS QUE NO estuvieran en este conjunto (para lo que se necista un 3er bucle).

En este punto se utilizó el finalmente el infame método UAM para corroborar lo que ya sabíamos: todas las oficinas habían vendido algo en 2003, por lo que el resultado final era Ø.

```
SELECT OFC.country, COUNT(OFC.officecode) AS N_OFFICES
FROM OFFICES OFC
WHERE OFC.officecode IN
      SELECT OFC.officecode
      FROM OFFICES OFC, EMPLOYEES E, CUSTOMERS CS, ORDERS O
      EXCEPT

      (SELECT OFC.officecode
      FROM OFFICES OFC, EMPLOYEES E, CUSTOMERS CS, ORDERS O
      WHERE OFC.officecode = E.officecode
      AND E.employeenumber = CS.salesrepemployeenumber
      AND CS.customernumber = O.customernumber
      AND O.customernumber IN

            (SELECT O.customernumber
             FROM ORDERS O
             WHERE O.ordernumber IN(

                  SELECT O.ordernumber
                  FROM ORDERS O
                  WHERE 2003 = extract(year from O.orderdate)))))

GROUP BY OFC.COUNTRY;
```

3.6. Definimos el carro de la compra como el conjunto de todos los productos comprados usando la misma "order". Se desea un listado de todas las parejas de productos que aparezcan en más de un carro de la compra. La salida debe mostrar tres líneas conteniendo el identificador de ambos productos y el número de carros de la compra en el cual aparecen. Nota, cada pareja de productos debe aparecer una única vez, en particular la pareja de productos con identificadores (id1, id2) no es distinta de la pareja (Id2, Id1) que tiene los mismos identificadores pero en otro orden.

Por la complejidad de esta última consulta, el código ha sido comentado línea a línea.

Estamos convencidos de que hay una manera más fácil de realizarla, con menos redundancia, pero no hemos encontrado manera de asegurar que no se repitieran los id's en distinto orden.

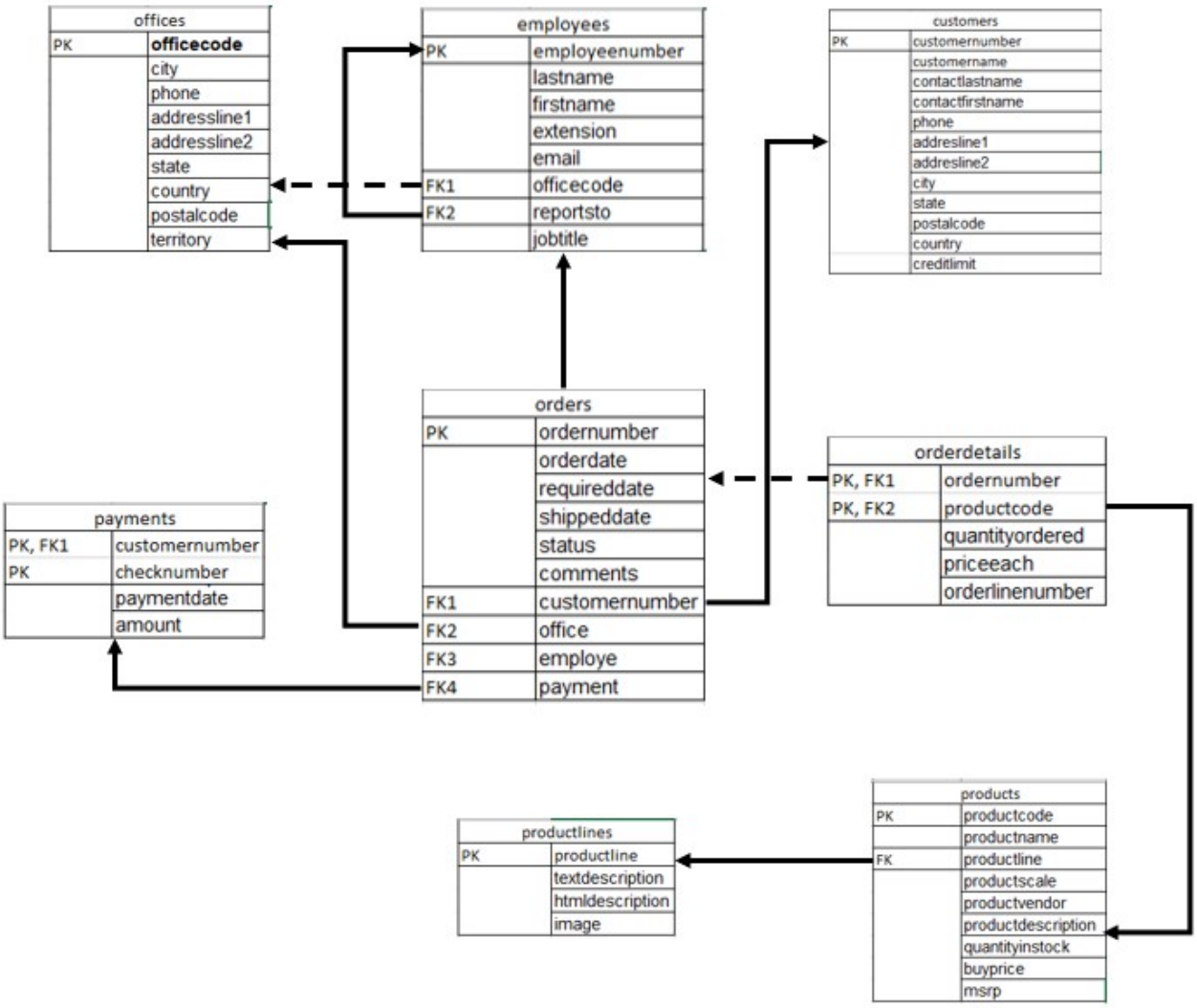
```
CREATE VIEW PAREJAS_MAS_DE_UN_CARRO AS
SELECT OD.productcode AS PR_1, OD_2.productcode AS PR_2, count(*) AS CARROS
--Para ello buscamos dos productos que esten en un mismo carro (order)
--Es decir un producto en un order, otro producto en ese mismo order y
FROM orders O JOIN orderdetails OD ON OD.ordernumber=O.ordernumber
JOIN orderdetails OD_2 ON OD_2.ordernumber=O.ordernumber
--asegurarse que no se trate del mismo producto
WHERE OD.productcode<>OD_2.productcode
--Los agrupamos por las veces que aparecen esos productos
--Es decir, los carros diferentes que los contienen
GROUP BY OD.productcode, OD_2.productcode
--filtramos para que solo muestre aquellos que aparecen en al menos un carro
--HAVING COUNT(*) > 1
ORDER BY OD.productcode DESC;

--Una vez con esta tabla ya podemos buscar los que aparecen repetidos cambiando
su orden
SELECT PM.pr_1, PM.pr_2, PM.carros+PM_2.carros AS SUM_CARROS
FROM PAREJAS_MAS_DE_UN_CARRO PM, PAREJAS_MAS_DE_UN_CARRO PM_2
--sumamos los carros de los que aparecen x-y e y-x
--mostramos los productos que aparecen en un orden y otro
--al sumar las apariciones de ambos ordenes hallamos el total
--Para que solo cuente las de la forma x-y
WHERE PM.pr_1=PM_2.pr_2 and PM_2.pr_1=PM.pr_2 and PM.pr_1>PM_2.pr_1
--y los unimos con aquellos que solo aparecen en un orden
UNION

(SELECT PMx.*
FROM PAREJAS_MAS_DE_UN_CARRO PMx
--los que solo aparecen en un orden son aquellos que solo aparecen de la forma
x-y
--y no existe la forma y-x
WHERE NOT EXISTS(
SELECT PMx_2.*
FROM PAREJAS_MAS_DE_UN_CARRO PMx_2
WHERE PMx_2.pr_2=PMx.pr_1 and PMx_2.pr_1=PMx.pr_2) and PMx.carros>1)
ORDER BY pr_1, pr_2;
```

4. Rediseño de la Base de Datos

El diseño de la base de datos es en general bastante pobre y muestra múltiples deficiencias. Por ejemplo: si un empleado se mueve de una oficina a otra se pierde la información de la/s oficina/s donde había trabajado en el pasado, un cliente solo puede relacionarse con un único empleado o los pagos no están asociados a una compra en concreto. Diseña una base de datos que evite los inconvenientes citados. Incluye en la memoria: (1) el nuevo diagrama relacional y comenta como tus cambios solucionan los problemas planteados y (2) añade un nuevo comando en el fichero makefile que se ejecute con makefile nuevabase y que borre todas las tablas y datos de la base de datos y cree las tablas de tu nuevo diseño (los comandos SQL necesarios para crear las tablas se almacenarán en un fichero llamado nuevabase.sql).



Añadimos a orders los campos customernumber (FK que hace referencia a customers.customernumber) y employee (FK que referencia a employees.employeenumber),

pudiendo de esta forma relacionar a cada usuario con más de un empleado, al estar unidos entre sí por compras en concreto.

También se añade el campo office (FK que hace referencia a offices.officecode) y de esta forma, si el empleado que realizó la venta cambia de oficina, el pedido está ligado con la oficina en la que se hizo: ya no necesitamos “rastrear” al empleado para conocer la oficina.

Por último se añade la clave payment, que referencia a payments.checknumber, consiguiendo así que los pagos se asocien con una compra en concreto.