Practica 1 - Fundamentos de Bases de Datos

Tommaso Genovese

10 octubre 2024

Contents

1 Introducción

Este documento corresponde a la primera práctica del curso *Fundamentos de Base de Datos* del año académico 2024/2025.

En esta práctica, he trabajado con consultas SQL sobre una base de datos de modelos de coches clásicos, aplicando diferentes técnicas para extraer y analizando las informaciones almacenadas. El esquema de la base de datos y las consultas realizadas se presentan a lo largo del documento.

2 Esquemas de la base de datos

2.1 Tablas en linea

.

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

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

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

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

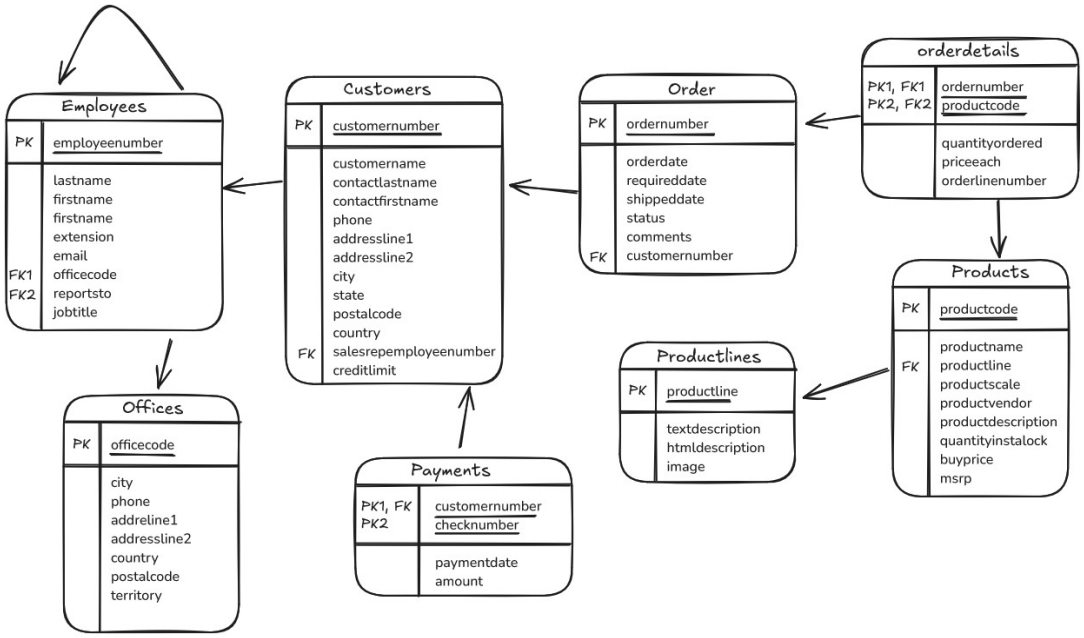
ORDERS (**ordernumber**, orderdate, requireddate, shippeddate, status, comments, customernumber → customers.customernumber)

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

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

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

2.2 Diagrama relacional



3 Consultas SQL

3.1 Query1

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 abona- da de mayor a menor cantidad. Cada linea debe mostrar: “customernumber”, “customername” y la cantidad total de dinero pagada.

```
SELECT c.customernumber,
       c.customername,
       Sum(p.amount) AS pago_totale
FROM   customers c,
       payments p,
       orders o,
       orderdetails od,
       products pr
WHERE  pr.productcode = od.productcode
       AND od.ordernumber = o.ordernumber
       AND o.customernumber = c.customernumber
       AND pr.productname = '1940 Ford Pickup Truck'
GROUP  BY c.customernumber,
          c.customername
ORDER  BY pago_totale DESC
```

3.1.1 Descripción de la implementación query1

Para hacer esta consulta necesitamos de conectar la tabla *Customers* con la tabla *Products* y verificar si el cliente has comprado un ”1940 Ford Pickup Truck”. Si sí imprimir la suma de los pagos de el cliente.

3.2 Query2

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.

```
SELECT p.productline,
       Avg(o.shippeddate - o.orderdate) AS Tiempo_medio
FROM   products p,
       orderdetails od,
       orders o
WHERE  p.productcode = od.productcode
       AND o.ordernumber = od.ordernumber
GROUP  BY p.productline
ORDER  BY tiempo_medio
```

3.2.1 Descripción de la implementación query2

Para hacer esta consulta necesitamos de calcular el tiempo medio, entres *shippeddate* y *orderdate*. Conectar *Products* y *Orders* para imprimir los datos requerido.

3.3 Query3

Empleados que reportan a otros empleados que reportan al director. El di- rector es aquella persona que no reporta a nadie. El listado debe mostrar el “employeenumber” y el “lastname”.

```
SELECT e.employeenumber,
       e.lastname
FROM   employees e
WHERE  e.reportsto IN (SELECT e.employeenumber
                       FROM   employees e
                       WHERE  e.reportsto IN (SELECT e.employeenumber
                                              FROM   employees e
                                              WHERE  e.reportsto IS NULL))
```

3.3.1 Descripción de la implementación query3

Para hacer esta consulta es necesario utilizar las subqueries así que buscamos el empleado director con la consulta mas interna, después buscamos los empleados quien reportan al director y al final los empleados quien reportan a esos empleados

3.4 Query4

Oficina que ha vendido el mayor numero de objetos. Nota: en un pedido ("order") se puede vender mas de una unidad de cada producto, cada unidad se considerar´a un objeto. La salida debe mostrar el "officecode" y el numero de productos vendidos.

```
SELECT o.officecode,
       Count(*)
FROM   offices o,
       employees e,
       customers c,
       orders ord,
       orderdetails od,
       products p
WHERE  od.ordernumber = ord.ordernumber
       AND p.productcode = od.productcode
       AND ord.customernumber = c.customernumber
       AND c.salesrepemployeenumber = e.employeenumber
       AND e.officecode = o.officecode
GROUP  BY o.officecode
ORDER  BY Count(*) DESC
LIMIT  1
```

3.4.1 Descripción de la implementación query4

Para hacer esta consulta es necesario contar el numero de ocurrencias de cada oficina conectada con los productos en los ordenes. Después yo he ordenado y limitado a 1, para visualizar solamente el primer dato.

3.5 Query5

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 numero de oficinas que no han realizado ninguna venta. Ordena las salidas por el numero de oficinas de forma que la primera línea muestre el país con mas oficinas que no han realizado ninguna venta.

```
SELECT o.country,
       Count(o.officecode) AS offices_without_sales
FROM   offices o
       LEFT JOIN employees e
              ON o.officecode = e.officecode
       LEFT JOIN customers c
              ON e.employeenumber = c.salesrepemployeenumber
       LEFT JOIN orders ord
              ON c.customernumber = ord.customernumber
                 AND ord.orderdate >= '2003-01-01'
                 AND ord.orderdate <= '2003-12-31'
WHERE  ord.ordernumber IS NULL
GROUP  BY o.country
HAVING Count(o.officecode) > 1
ORDER  BY offices_without_sales DESC;
```

3.5.1 Descripción de la implementación query5

Lo que he hecho aquí es utilizar los LEFT JOIN para obtener una tabla con un campo *orders*, donde si hay el valor NULL significa que esta oficina no tiene orden en el 2003. Cuenta cuantas oficinas de cada país no tienen no tienen orden y imprime el país y el numero.

3.6 Query6

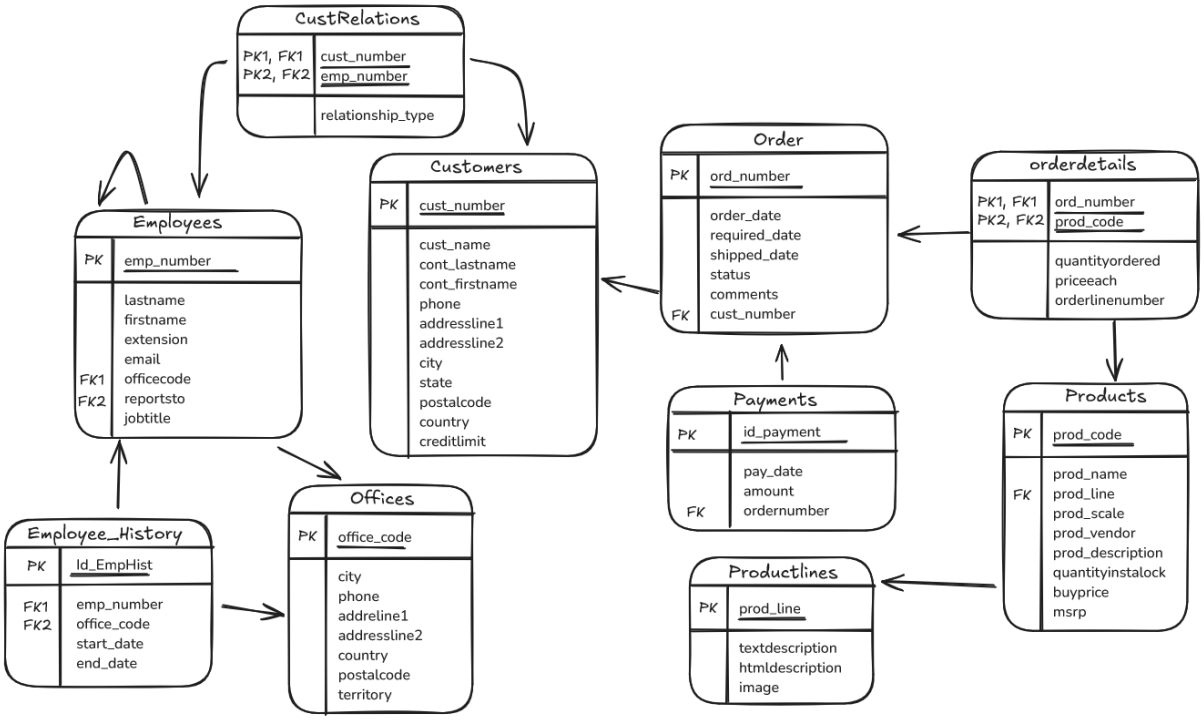
Definimos el carro de la compra como el conjunto de todos los productos comprados usando la misma "order". Se desea un listado de todas las parejas de productos que aparezcan en mas de un carro de la compra. La salida debe mostrar tres líneas conteniendo el identificador de ambos productos y el numero 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.

```
SELECT od1.productcode AS product1,
       od2.productcode AS product2,
       Count(*)        AS num_orders
FROM   orderdetails od1
       JOIN orderdetails od2
         ON od1.ordernumber = od2.ordernumber
            AND od1.productcode < od2.productcode
GROUP  BY od1.productcode,
          od2.productcode
HAVING Count(*) > 1
ORDER  BY product1 DESC;
```

3.6.1 Descripción de la implementación query5

Utilizando *od1* y *od2* podemos visualizar productos de el mismo orden (od1.ordernumber = od2.ordernumber), contarlos y filtrar también para evitar redundancia (od1.productcode < od2.productcode)

4 Rediseño de la Base de Dato



Esta es la mejor idea que tengo para rediseñar esta base de datos.

El primer cambio que he hecho es **modificar la legibilidad de los campos**. Por ejemplo, un campo llamado *"salesrepemployeenumber"* es demasiado largo, no es fácil de leer y entender; una opción mejor podría ser *"sales_rep_emp"* (en este rediseño pero no utilizo este campo).

El segundo cambio es **crear una tabla para guardar la historia de un empleado**: la tabla *Employee_History* tiene una clave primaria *id_EmpHis* y dos claves foráneas *emp_number* y *officecode*. En primer lugar yo no tenia id, solamente tenia *emp_numb* y *officecode* como claves primaria y extrañas; pero, de esta manera, si un empleado volvía a trabajar en una oficina donde ya había trabajado, habría un valor repetido.

El tercer cambio es **crear una tabla para relacionar los clientes con los empleados**, así que un cliente se puede relacionar con mas de 1 empleado.

El cuarto cambio es modificar la tabla *Payments*. Creo que es mejor si los pagos están relacionados directamente con los pedidos y no con el cliente; por lo tanto, he añadido la clave foránea *order_number* a la tabla *Payments*.

Un cambio adicional que se podría hacer es crear una tabla para el historial de precios de los productos, pero no veo su utilidad en esta base de datos.

Importante: *En el rediseño no he hecho el fichero nuevabase.sql y el comando make nuevabase es incompleto*