Universidad Autónoma de Madrid
Departamento de Informática

Estructura de Datos Memorias Práctica - 1

Hecho por: Diego Rodríguez
y Alejandro García

Estructura de la base de datos:

Productline(Productline, Textdescription, Htmldescription, Image)

Products(Productcode, Productname, Productline →Productline.productline, Productscale, Productvendor, Productdescription, Quantitystock, Buyprice, MSRP)

Orderdetails(Ordernumber → orders.ordernumber, Productcode → Products.Productcode, Quantityordered, Priceeach,orderlinenumber)

Orders(Ordernumber, customernumber→customer.customernumber, orderdate, Requireddate, shippeddate, Status, Comments)

Payments(Customernumber→customer.customernumber,Checknumber, Paymentdate, amount)

Customers(Customernumber, Salesremployeenumber →employees.employeenumber, customernumber, contactlastname, contactfirstname, Phone, Addressline1, Addressline2, city, State, Postalcode, Country, creditlimit)

Employees(Employeenumber, Officecode →Offices.officecode, Reportsto→Employees.employeenumber, lastname, firstname, extension, email, jobtitle)

Offices(Officecode, City, Phone, Addressline1, Addressline2, State, Country, Postalcode, Territory)

Esquema de la base de datos:

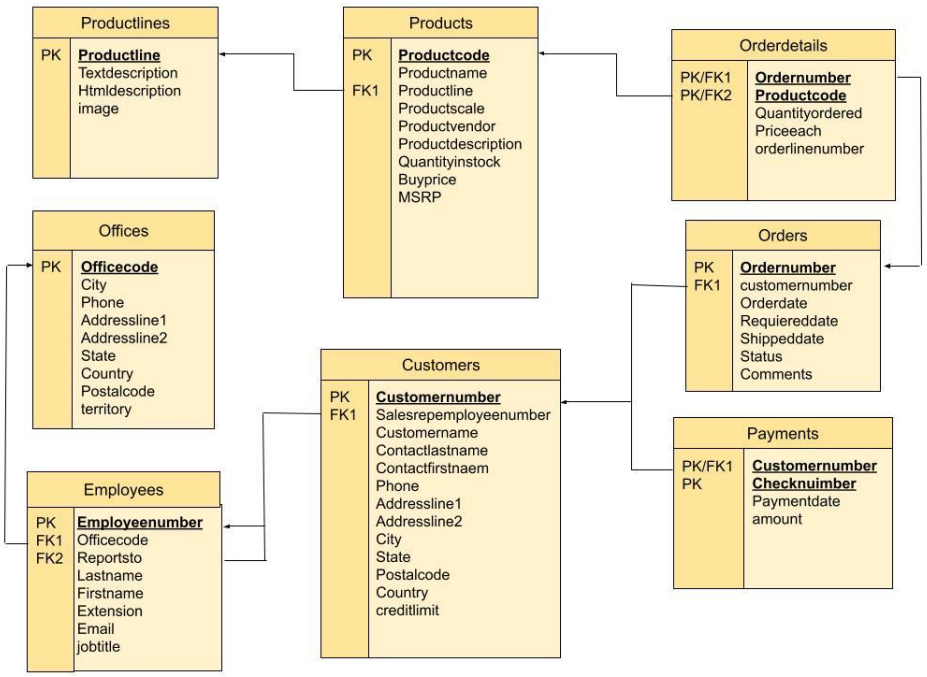
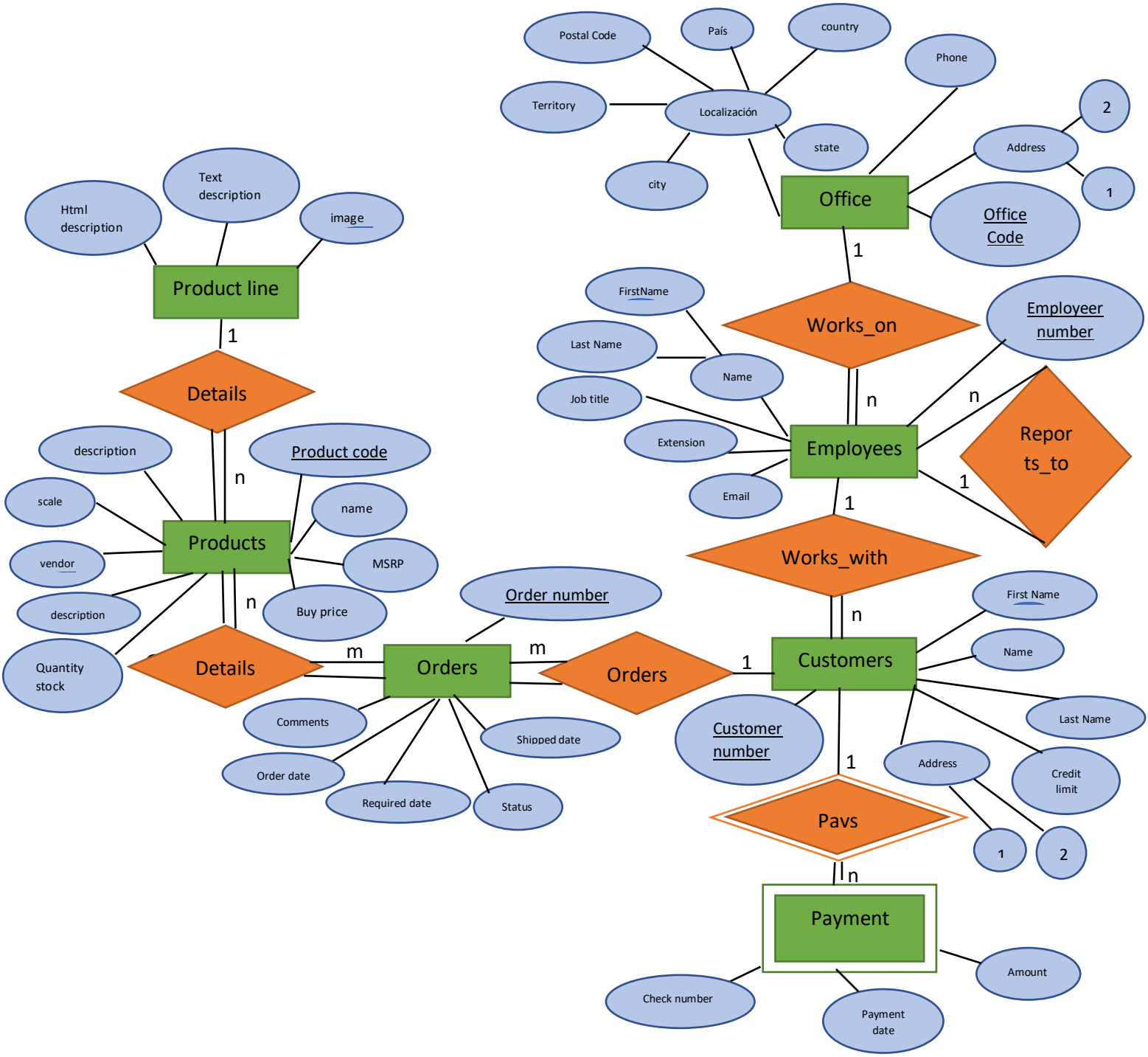


Diagrama relacional de la base de datos:



SQL y una breve descripción de la implementación:

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

```
WITH ordernum
     AS (SELECT orderdetails. ordernumber
         FROM   orderdetails
                JOIN products
                  ON products.productcode = orderdetails.productcode
         WHERE  products.productname = '1940 Ford Pickup Truck'),
     custnum
     AS (SELECT orders.customernumber
         FROM   orders
                INNER JOIN ordernum
                        ON orders.ordernumber = ordernum.ordernumber)
SELECT customers.customernumber,
       customers.customername,
       Sum(payments.amount)
FROM   customers
       JOIN custnum
         ON customers.customernumber = custnum.customernumber
       JOIN payments
         ON payments.customernumber = custnum.customernumber
GROUP  BY customers.customernumber
ORDER  BY sum DESC;
```

En primer lugar hemos definido dos tablas temporales con la función “WITH”: una llamada "ordernum" que guarda el "ordernumber" de aquellas órdenes cuyo "productname" fuese '1940 Ford Pickup Truck'; y una segunda con nombre "custnum" con el "customernumber" de aquellas personas asociadas a los "ordernumbers" de la tabla "ordernum". Con todo esto hemos seleccionado el "customernumber", el "customername" y la suma del "paymentsamount" de cada cliente cuyo "customernumber" apareciera en la tabla "custnum", previamente definida.

Para comprobar el correcto funcionamiento de la query hemos realizado varias pruebas. En primer lugar, hemos cambiado todas las tuplas donde “productname” era '1940 Ford Pickup Truck' y la hemos ejecutado comprobando que devuelve una tabla vacía. En segundo lugar, en el estado de la relación original hemos añadido un payment de 1000 a nombre de un cliente que ha comprado una '1940 Ford Pickup Truck' y hemos visto como dicha suma subía en 1000. Por último, hemos añadido un gran pago a una orden de uno de estos productos, y hemos visto como el cliente asociado a esta compra se situaba el primero de la tabla devuelta por la consulta.

- **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.

```
WITH orderandtime
     AS (SELECT orders.ordernumber,
                orders.shippeddate - orders.orderdate AS DiffDates
         FROM   orders where orders.status='Shipped')
SELECT products.productline,
       Avg(orderandtime.diffdates) AS PrCodAvg
FROM   orderdetails
       JOIN orderandtime
         ON orderdetails.ordernumber = orderandtime.ordernumber
       JOIN products
         ON orderdetails.productcode = products.productcode
GROUP  BY products.productline;
```

Para resolver esta cuestión hemos empezado almacenando en la tabla "orderandtime" el número de orden y la diferencia entre las fechas siempre que el pedido hubiera sido enviado. Con esto hemos seleccionado "productline" y la media de la diferencia de las fechas almacenada en la tabla definida con anterioridad (llamando a esta columna “PrCodAvg”).

Nos hemos asegurado de que es correcta adelantado varios años la fecha de compra de uno de los pedidos, haciendo que al ejecutar la consulta el resultado obtenido sea significativamente mayor. Además, esta misma prueba ha sido realizada con distintas líneas de producción y hemos comprobado que únicamente cambiaba el valor de la que habíamos modificado.

- Query3: 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”.

```
WITH numdir
     AS (SELECT employees.employeenumber
         FROM   employees
         WHERE  employees.reportsto IS NULL),
     subdir
     AS (SELECT employees.employeenumber
         FROM   employees
                JOIN numdir
                  ON employees.reportsto = numdir.employeenumber)
SELECT employees.employeenumber,
       employees.lastname
FROM   employees
       JOIN subdir
         ON employees.reportsto = subdir.employeenumber;
```

Con un “WITH” hemos creado las tablas temporales "numdir" con el "employeenumber" de aquellos empleados que no reporten a nadie, es decir los jefes, y también la tabla de nombre "subdir" que guarda el "employeenumber" de aquellos que reporten al director (cuyo ID se almacena en la tabla anterior). Después, tan solo hemos tenido que seleccionar aquellos empleados cuyo valor almacenado en el apartado "reportsto" está en la columna "employeenumber" de "subdir".

Como método de comprobación hemos añadido dos empleados cuyo supervisor reporta al director, añadiendo al resultado de la consulta 2 tuplas. También, hemos hecho que nadie reportara al director, provocando que la consulta devolviese una tabla vacía.

- **Query4:** 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.

```
WITH objpercust AS
(
        SELECT   orders.customernumber,
                 Sum(orderdetails.quantityordered) AS qpercust
        FROM     orders
        JOIN     orderdetails
        ON       orders.ordernumber=orderdetails.ordernumber
        GROUP BY customernumber ), objperempnum AS
(
        SELECT   customers.salesrepemployeenumber,
                 Sum(qpercust) AS qpere
        FROM     customers
        JOIN     objpercust
        ON       customers.customernumber=objpercust.customernumber
        GROUP BY customers.salesrepemployeenumber )
SELECT   employees.officecode ,
        Sum(qpere) AS qperof
FROM     employees
JOIN     objperempnum
ON       employees.employeenumber=objperempnum.salesrepemployeenumber
GROUP BY officecode
ORDER BY qperof DESC limit 1;
```

Primero hemos utilizado la función "WITH" para definir "objpercust", que guarda el número de objetos que ha sido comprado por cada cliente (utilizando su "customernumber" para identificarlo). Con esta misma función, hemos creado "objperempnum" donde se almacena cuántos productos ha vendido cada empleado (utilizando para ello la tabla anterior y "customers" para relacionar las compras de cada individuo con un empleado, quién es identificado por su número de comercial). Tras esto, nos valemos de las nuevas tablas creadas y de "employees" para asociar el número de ventas de cada empleado a la oficina para la que trabaja cada uno de ellos, y mostramos el código de la oficina que más productos ha vendido.

Para ver que la consulta era correcta hemos multiplicado la cantidad de productos de un pedido asociado a una oficina minoritaria, que tras la modificación aparecía como resultado de ejecutar la query, pues estaba registrado como la que más productos había vendido. También hemos quitado el límite de oficinas que se muestran para comprobar que al sumar elementos a pedidos de distintas oficinas se sumaban y ordenaban todas correctamente.

- 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 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.

```
WITH fechin2003
     AS (SELECT orders.customernumber
         FROM   orders
         WHERE  orders.orderdate <= '2003-12-31'
                AND orders.orderdate >= '2003-01-01'),
     empsalein2003
     AS (SELECT customers.salesrepemployeenumber
         FROM   customers
                JOIN fechin2003
                  ON customers.customernumber = fechin2003.customernumber),
     ofcnotin2003
     AS ((SELECT employees.officecode
          FROM   employees)
         EXCEPT
         (SELECT employees.officecode
          FROM   employees
                 JOIN empsalein2003
                   ON
         employees.employeenumber = empsalein2003.salesrepemployeenumber))
SELECT offices.country,
       Count(*)
FROM   offices
       JOIN ofcnotin2003
         ON offices.officecode = ofcnotin2003.officecode
GROUP  BY offices.country;
```

Para obtener los datos deseados primero hay que definir la tabla temporal "fechin2003" que almacena el número de cliente de aquellos que hayan realizado un pedido en 2003. Por otro lado, en "empsalein2003" el "salesrepemployeenumber" de los empleados asociados a los clientes que compraron en 2003 (cuyo número está almacenado en la tabla anterior); y "ofcnotin2003" tiene el código de la oficina que no tenga ningún empleado en la tabla "empsalein2003", es decir, nadie que haya realizado una venta en dicho año. Por último, lo único que nos queda es contar todas estas oficinas cuyo código se encuentra en "ofcnotin2003" y agruparlas por país.

Esta consulta nos ha resultado difícil de comprobar ya que se deben comprobar muchos detalles. Para empezar, hemos reducido el intervalo de tiempo a un día para que no todas las oficinas hayan recibido un pedido en ese periodo. Otra comprobación que probamos fue actualizar la fecha de las compras realizadas en 2003 y asignarlas al 2004, luego la consulta devuelve todos los países. Alterando un poco la consulta y haciendo que devuelva los “officode” en vez del país, comprobamos que devuelve todas las oficinas y por tanto podemos conjeturar que funciona correctamente.

- **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 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.

```
with resultado AS
(
        SELECT   o1.productcode,
                 o2.productcode,
                 count(DISTINCT o1.ordernumber) AS cont
        FROM     orderdetails                   AS o1
        JOIN     orderdetails                   AS o2
        ON       o1.ordernumber = o2.ordernumber
        WHERE    o1.productcode < o2.productcode
        GROUP BY o1.productcode,
                 o2.productcode)
SELECT   *
FROM     resultado
WHERE    cont >1
ORDER BY cont;
```

Comenzamos creando la tabla temporal "resultado", la cual nos muestra los "productcode" de una pareja de productos que han sido comprado juntos en al menos un carro de la compra y en cuántos carros distintos aparece cada una de ellas. Para que cada pareja aparezca una única vez hemos utilizado la condición de que el código del primer producto debe ser estrictamente menor que el del segundo. Posteriormente, tan solo tuvimos que seleccionar aquellas parejas que hubieran sido compradas juntas en más de una ocasión, es decir, aquellas tuplas cuyo valor almacenado en la columna "cont" fuese mayor que 1.

Para cerciorarnos de su validez hemos añadido y eliminado productos de forma controlada y visto cómo variaba la columna "cont" de aquellas parejas en las que aparecía. Para forzar un poco los extremos, hemos buscado aquellas parejas que aparecían sólo 2 veces y eliminado uno de los productos de uno de los carros, de este modo se registra que la pareja se ha comprado junta una única vez y no aparece en el listado.

Rediseño de la base de datos:

Con el objetivo de solucionar los siguientes tres problemas que presentaba la base de datos original:

- La pérdida de la información sobre en qué oficinas ha trabajado un empleado.
- El hecho de que un cliente sólo puede ser atendido por un único empleado.
- La falta de relación entre un pago y una compra.

Para ello, hemos propuesto la siguiente implementación de la base de datos. En primer lugar, hemos creado una nueva tabla "EmployeeOffice" que tiene por columnas: id (un número que actúa como clave primaria), employeenumber (clave externa a employees.employeenumber), Officecode (clave externa a Offices.Officecode), datein y dateout (fechas que indican en qué intervalo de tiempo ha estado trabajando un empleado en una oficina). Esta nueva relación soluciona el primero de los problemas ya que la información que antes se perdía ahora se guarda en esta tabla.

Tras ello, para resolver el segundo inconveniente, se nos ocurrieron dos métodos: añadir la columna employeenumber (como clave externa a employees.employeenumber) a la tabla "orders" o crear una tabla con dos columnas, employeenumber y customernumber ambas claves externas con referencia a employees.employeenumber y customers.customernumber respectivamente. Finalmente hemos optado por la primera opción ya que, la segunda no permitía conocer en qué pedidos de un cliente ha trabajado cada empleado, que es información relevante para la nueva base de datos.

Por último, el tercer problema lo hemos resuelto cambiando la columna customernumber de la tabla "payments" por ordernumber (como clave foránea referenciando a orders.ordernumber), que además actúa como clave primaria. Esto permite relacionar un pago con un pedido y conociendo el pedido somos capaces de extraer el cliente, luego no se pierda información, pues se conecta un pedido con un pago. Con este rediseño hemos intentado mejorar la representación de nuestro minimundo por parte de la base de datos, creando relaciones y almacenando información que consideramos relevantes.

El esquema de la nueva base de datos: