Fundamentos de bases de datos

Práctica 1

David Brenchley Uriol

Javier San Andrés de Pedro

Índice

1. Esquema de la base de datos

En esta sección, vamos a enunciar los atributos de cada tabla y a ver la clave primaria y foránea de cada tabla. En la siguiente sección, todo lo expuesto aquí quedará reflejado de manera más visual.

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

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

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

products (**productcode**, productname, productscale, productvendor, productdescription, quantiyinstock, buyprice, msrp, porductline ⟶ productlines.productline);

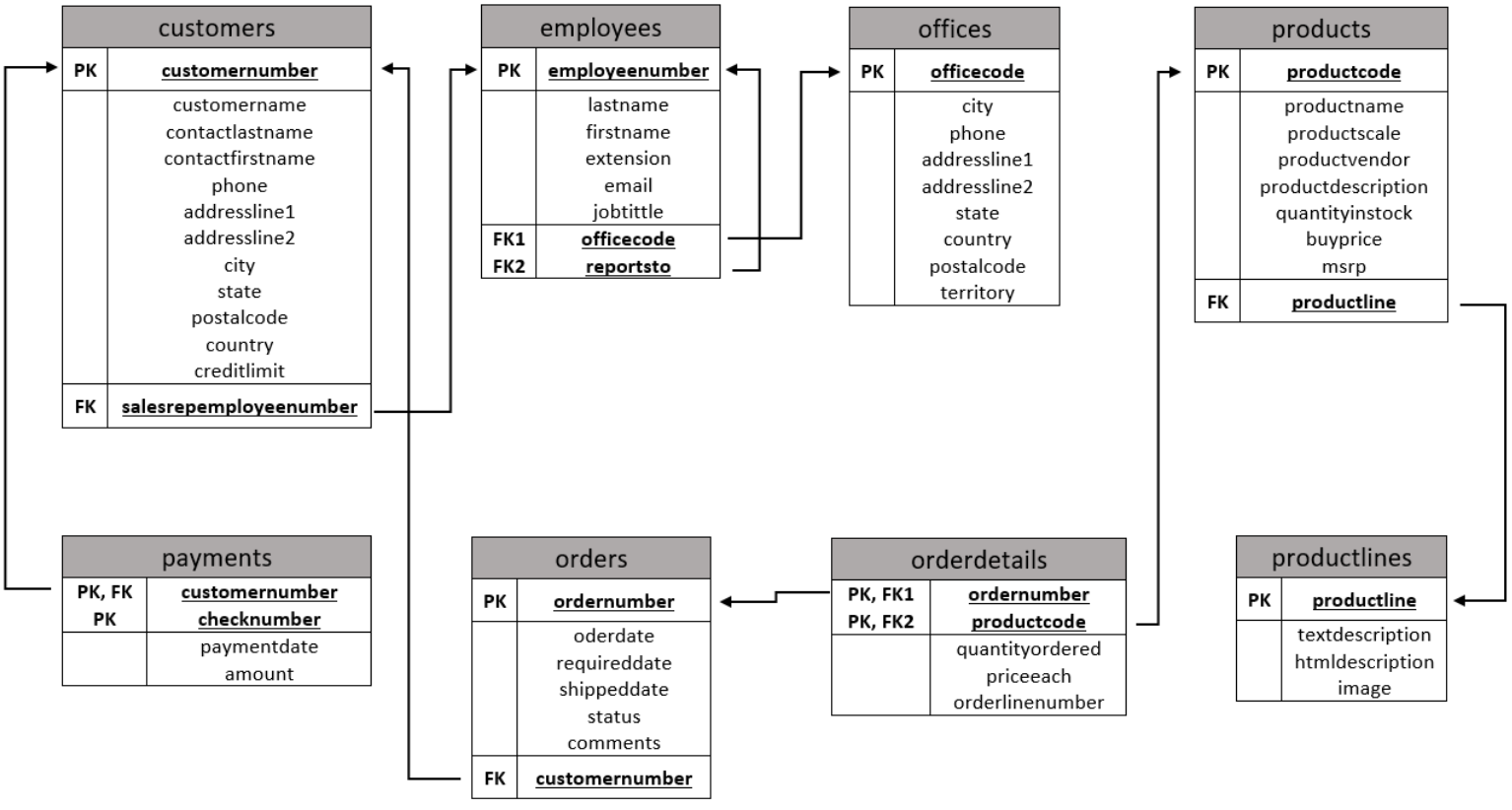
payments (**customernumber** ⟶ customers.customernumber, **checknumber**, amount, paymentdate);

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

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

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

2. Diagrama relacional



3. Consultas

Consulta 1:

En esta consulta, hay que mostrar la cantidad total de dinero abonado por cada cliente que se ha llevado un artículo determinado. En nuestro caso, el 1940 Ford Pickup Truck.

Este es el código que hemos elaborado:

```
WITH Pickup_customers
     AS (SELECT DISTINCT c.customernumber,
                         c.customername
         FROM   customers c,
                orders o1,
                orderdetails o2,
                products p
         WHERE  p.productname = '1940 Ford Pickup Truck'
                AND o2.productcode = p.productcode
                AND o2.ordernumber = o1.ordernumber
                AND o1.customernumber = c.customernumber)
SELECT p.customernumber,
       p.customername,
       Sum(pa.amount) AS "Total Pagado"
FROM   Pickup_customers p,
       payments pa
WHERE  p.customernumber = pa.customernumber
GROUP  BY p.customernumber,
          p.customername
ORDER  BY "Total Pagado";
```

Nuestra estrategia se basa en crear primero una tabla auxiliar (Pickup_customers) donde guardemos las tuplas de nombre del cliente y número de cliente de todos los clientes diferentes que hayan comprado el artículo en cuestión. Al tener cada cliente un único y propio customernumber asociado a un determinado nombre, el select distinct nos sirve para poder discernir entre clientes y quedarnos solo con uno de cada, evitando así que pueda haber repeticiones que repercutan en el resultado final.

Una vez creada esta tabla, simplemente hay que tomar las tuplas de la auxiliar e ir sumando los diferentes pagos que se han ido haciendo asociados a cada único customernumber. De esa forma, con solo agregar y ordenar por el número total pagado tendríamos lo que se nos pide.

El problema de no usar primero un select que te seleccione una única vez cada customernumber es que, en caso contrario, el segundo select a la hora de sumar sumaría de manera repetida todo lo que ha pagado un determinado cliente tantas veces como compras diferentes del artículo haya realizado. Por ejemplo, si Juan ha comprado en dos compras diferentes ese artículo y Juan ha pagado en total de todas las compras 30; de no hacer esa criba, nos saldría que Juan ha pagado 60, cuando eso es evidentemente falso.

Es importante, por tanto, tener en cuenta que un mismo cliente puede comprar ese artículo en dos compras diferentes (con diferentes ordernumber).

Para la comprobación de que hemos realizado bien el ejercicio, hemos alterado en 1€ uno de los pagos realizados por el customernumber 141 (el cliente que más ha pagado en total). Si todo está correcto, debería de incrementarse solo en ese euro su total de dinero pagado.

Mediante el comando:

```
update payments set amount='59831.55'
where customernumber='141' and checknumber ='HJ32686';
```

Podemos pasar de tener un dato al otro en una determinada compra tal y como se puede apreciar en las imágenes de debajo:

141	DB583216	2004-11-01	36,140.38
141	DL460618	2005-05-19	46,895.48
141	HJ32686	2004-01-30	59,830.55
141	ID10962	2004-12-31	116,208.4
141	IN446258	2005-03-25	65,071.26

141	MF629602	2004-08-16	20,009.53
141	NU627706	2004-05-17	26,155.91
141	HJ32686	2004-01-30	59,831.55
144	IR846303	2004-12-12	36,005.71
144	LA685678	2003-04-09	7,674.94

Veamos que el resultado difiere solo en la diferencia introducida:

	customernumber	customername	Total Pagado
1	141	Euro+ Shopping Channel	715,738.98
2	124	Mini Gifts Distributors Ltd.	584,188.24
3	114	Australian Collectors, Co.	180,585.07
4	148	Dragon Souveniers, Ltd.	156,251.03
5	276	Annas Decorations, Ltd	137,034.22
6	321	Corporate Gift Ideas Co.	132,340.78
7	146	Saveley & Henriot, Co.	130,305.35

	customernumber	customername	Total Pagado
1	141	Euro+ Shopping Channel	715,739.98
2	124	Mini Gifts Distributors Ltd.	584,188.24
3	114	Australian Collectors, Co.	180,585.07
4	148	Dragon Souveniers, Ltd.	156,251.03
5	276	Annas Decorations, Ltd	137,034.22
6	321	Corporate Gift Ideas Co.	132,340.78
7	146	Saveley & Henriot, Co.	130,305.35

Se puede apreciar claramente que ese el caso. Tras una revisión visual sobre la veracidad de los resultados (podemos habernos equivocado en alguna otra parte), concluimos dando como válida la consulta.

Consulta 2:

En esta consulta, tenemos que ver la media de tiempo transcurrido entre que se ordena un producto hasta que llega. Nuestro código es el siguiente:

```
SELECT p.productline,
       Cast(Avg(o1.shippeddate - o1.orderdate) AS DECIMAL(5, 3)) AS
       "Tiempo Medio Transcurrido"
FROM   orders o1,
       orderdetails o2,
       products p
WHERE  o1.ordernumber = o2.ordernumber
       AND o2.productcode = p.productcode
       AND o1.status = 'Shipped'
GROUP  BY p.productline;
```

Básicamente, lo único que hacemos es enlazar de alguna manera cada productline con la order que contenga un producto de ese mismo tipo y, ahí, obtener datos sobre cuál fue la diferencia de tiempo transcurrido. Además, hemos considerado para el cálculo solo los pedidos que tenían el estado de ‘Shipped’ (lo cual varía los resultados si no se tiene en cuenta el detalle). Por cada diferencia de tiempo, se inserta en un cálculo de una media asociada a cada productline. Luego solo quedaría agrupar por cada una.

Para la comprobación de cómo de correcta es la consulta, primero vamos a observar que en nuestra salida hay tantas productline como las hay en la propia tabla. Esto es lo que hay almacenado en la tabla de productline:

	productline	textdescription	htmldescription	image
1	Classic Cars	Attention car enthusiasts: Make your wildest car ownership dreams come true. Whether you are looking for classic muscl	[NULL]	[NULL]
2	Motorcycles	Our motorcycles are state of the art replicas of classic as well as contemporary motorcycle legends such as Harley Davids	[NULL]	[NULL]
3	Planes	Unique, diecast airplane and helicopter replicas suitable for collections, as well as home, office or classroom decoration:	[NULL]	[NULL]
4	Ships	The perfect holiday or anniversary gift for executives, clients, friends, and family. These handcrafted model ships are unic	[NULL]	[NULL]
5	Trains	Model trains are a rewarding hobby for enthusiasts of all ages. Whether you're looking for collectible wooden trains, elec	[NULL]	[NULL]
6	Trucks and Buses	The Truck and Bus models are realistic replicas of buses and specialized trucks produced from the early 1920s to present	[NULL]	[NULL]
7	Vintage Cars	Our Vintage Car models realistically portray automobiles produced from the early 1900s through the 1940s. Materials us	[NULL]	[NULL]

Y esto lo que nos da nuestra consulta:

	productline	Tiempo Medio Transcurrido
1	Classic Cars	4.002
2	Trains	5.75
3	Planes	3.888
4	Trucks and Buses	4.49
5	Motorcycles	3.877
6	Vintage Cars	3.517
7	Ships	3.454

Ya de primeras, por lo menos, hay el mismo número. A continuación, vamos a realizar una consulta que me cuente cuántas orders utilizan products de un determinado productline. Voy a utilizar como ejemplo Classic Cars. De esa forma, luego variaré de manera consciente los datos para esperar una variación determinada. Eso confirmará la validez de la consulta.

Realizando una búsqueda, llego a que el siguiente order contiene un producto cuyo productline es 'Classic Cars'.

	ordernumber	orderdate	requireddate	shippeddate	status
1	10,103	2003-01-29	2003-02-07	2003-02-02	Shipped

Ahora, mediante una consulta, obtengo cuántos productos de todos los diferentes pedidos están en el total para el cálculo de la media. En concreto me fijo en solo Classic Cars.

```
SELECT p.productline,
       count(*) AS
       "Número media"
FROM   orderdetails o2,
       products p
WHERE  o2.productcode = p.productcode
GROUP  BY p.productline;
```

	productline	Número media
1	Classic Cars	1,010
2	Trains	81
3	Planes	336
4	Trucks and Buses	308
5	Motorcycles	359
6	Vintage Cars	657
7	Ships	245

Mediante otra consulta, observo que existen 3 productos dentro del pedido 10103 cuyo tipo es Classic Cars. Después, cambio para esa order la shippeddate al 2003-02-22. Lo cual me debería dejar la media de tiempo en (1010*4'002+3*20) / 1010 = 4'06.

```
update orders set shippeddate='2003-02-22'
where ordernumber ='10103'
```

	productline	Tiempo Medio Transcurrido
1	Classic Cars	4.065
2	Trains	5.75
3	Planes	3.888
4	Trucks and Buses	4.976
5	Motorcycles	3.877
6	Vintage Cars	3.715
7	Ships	3.454

Consulta 3:

En esta consulta, hay que exponer los empleados cuyo superior tiene como superior al director. De una forma u otra, nos parece que es claro que la query lleva de forma implícita varios select anidados. Por tanto, nuestro código es el siguiente:

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

Explicaremos brevemente la lógica detrás de la consulta. Lo que hacemos en el select más interno es seleccionar al director (se entiende que solo hay uno, aunque si hubiera varios tampoco afectaría a la validez del programa). Una vez tenemos al director buscamos qué empleados tienen como supervisor directo al director; eso se efectúa en el select del intermedio. Por último, queremos a esos empleados que tienen como superiores los anteriores; cosa que se consigue con el select más externo.

No se hace necesario la utilización del select distinct ya que cada empleado tiene un único superior y no se podría dar el caso en apareciera por reportar a otros dos empleados diferentes. Por eso, de haber dos directores, tendrían que ser personas diferentes y no se vería comprometido el resultado.

La comprobación va a ser sencilla. Primero, vamos a comprobar con una consulta qué número tiene el director. Después, vamos a obtener de manera explícita quiénes le tienen de superior directo. Por último, repetimos el proceso, pero con los empleados de los anteriores. Literalmente, estamos desgranando el select anidado.

```
SELECT employeenumber
    FROM   employees
    WHERE  reportsto IS null
```

De aquí sacamos que el director es el empleado de número 1002

```
SELECT employeenumber
    FROM   employees
    WHERE  reportsto = '1002';
```

Los empleados cuyo superior directo es el director tienen números 1056 y 1076

```
SELECT employeenumber
FROM   employees
WHERE  reportsto = '1056'
        OR reportsto = '1076';
```

La salida muestra:

	employeenumber
1	1,088
2	1,102
3	1,143
4	1,621

Esta salida es idéntica a la de nuestro programa. Vamos a cambiar un dato para que haya otro empleado más que tenga como superior al 1056, otro al 1076 y vamos a cambiar algún empleado que reporte al 1056.

```
update employees set reportsto = '1088' where employeenumber = '1102';
update employees set reportsto = '1056' where employeenumber = '1619';
update employees set reportsto = '1076' where employeenumber = '1612';
```

1,102	Bondur	Gerard	x5408	gbondur@classicmodelcars.com	4	1,056	Sale Manager (EMEA)
1,612	Marsh	Peter	x102	pmarsh@classicmodelcars.com	6	1,088	Sales Rep
1,619	King	Tom	x103	tking@classicmodelcars.com	6	1,088	Sales Rep

Tras estos cambios, deberían aparecer los nuevos números y debe desaparecer el 1102. Efectivamente, se comprueba que la consulta es correcta:

	employeenumber	lastname
1	1,088	Patterson
2	1,143	Bow
3	1,621	Nishi
4	1,619	King
5	1,612	Marsh

Consulta 4:

En este caso, hay que dar con la oficina que más unidades ha conseguido vender. Este es nuestro código:

```
SELECT o.officecode,
       Sum(o2.quantityordered) AS "Unidades vendidas"
FROM   offices o,
       employees e,
       customers c,
       orders o1,
       orderdetails o2
WHERE  o.officecode = e.officecode
       AND c.salesrepemployeenumber = e.employeenumber
       AND o1.customernumber = c.customernumber
       AND o2.ordernumber = o1.ordernumber
GROUP  BY o.officecode
ORDER  BY "Unidades vendidas" DESC
LIMIT  1;
```

Lo que hacemos es ver qué empleados vinculados a según qué oficina atendieron a un cliente. De ese cliente, podemos extraer el pedido que hizo y, en función de él, determinar la cantidad de unidades que se solicitaron. Ese último dato tiene que pasar a sumar como unidades vendidas por esa oficina en concreto y luego ya solo quedaría agrupar y ordenar según el total. De igual forma, se podría conseguir el mismo resultado con varios JOIN.

Ahora, comprobemos que el resultado es correcto. Según nuestra consulta, la oficina 4 es la que más vendió con un total de 33887 unidades. Vamos a alterar una compra asociada a esta oficina en 200 unidades. Luego, vamos a añadirle un número significativo a otra oficina para quitarle el primer puesto a la oficina 4.

Tomamos primero un empleado de la oficina 4.

1,370	Hernandez	Gerard	x2028	ghernande@classicmodelcars.com	4

Buscamos mediante un filtro en el visualizador Dbeaver, clientes asociados a él.

103	Atelier graphique	Schmitt	Carine

Hacemos lo mismo pero para buscar un pedido asociado a ese cliente.

10,123	2003-05-20	2003-05-29	2003-05-22	Shipped	[NU	103

Buscamos ese pedido en orderdetails y variamos la cantidad de uno cualquiera:

ordernumber	productcode	quantityordered	priceeach	orderlinenumber
10,123	S18_1589	26	120.71	2

Ahora aplicamos el siguiente comando para sumar 200 unidades más y vemos que el resultado aumenta conforme a lo esperado:

```
UPDATE orderdetails
SET    quantityordered = '226'
WHERE  ordernumber = '10123'
       AND quantityordered = '26';
```

	officecode	Unidades vendidas
1	4	34,087

Si vemos quién es la segunda oficina que más ha vendido en segundo lugar, vemos que es la oficina 1 con 15.910. Repito toda la parafernalia anterior y le sumo a un determinado pedido 20 mil unidades. Compruebo que, efectivamente, la oficina con más ventas cambia:

```
UPDATE orderdetails
SET    quantityordered = '20021'
WHERE  ordernumber = '10124'
       AND quantityordered = '21';
```

	officecode	Unidades vendidas
1	1	35,910

Consulta 5:

Para esta consulta, tenemos que sacar los países que cuentan con alguna oficina que no ha venido nada en el año 2003. Este es nuestro código

```
SELECT country,
       Count(*) AS "Número de oficinas"
FROM   offices o
WHERE  officecode NOT IN (SELECT DISTINCT officecode
                          FROM   employees,
                                 customers c
                          WHERE  c.customernumber IN
                                 (SELECT DISTINCT customernumber
                                  FROM   orders
                                  WHERE  orderdate >= '2003-01-01'
                                         AND orderdate <=
                                             '2003-12-31')
                                 AND c.salesrepemployeenumber = employeenumber)
GROUP  BY country
ORDER  BY "Número de oficinas" DESC;
```

Tal y como se puede apreciar, hemos hecho uso de tres select anidados. En el select más interno, seleccionamos todos los clientes distintos que han hecho algún pedido en el año 2003. Después, seleccionamos dentro de esos aquellas oficinas en las que haya un empleado que haya atendido algún cliente de los anteriores. El select más externo sirve para seleccionar aquellos países donde haya una oficina que no aparezca en las anteriores y, por tanto, que no haya realizado ninguna venta en el año 2003.

Para comprobar el resultado, simplemente vamos a modificar el código de nuestra consulta para que el año sea por ejemplo el 2008. Una breve vista por los datos de la tabla de orders, señalará de manera inmediata que no existe ningún pedido más allá del año 2005. Por tanto, nos debiera de dar como resultado todas las oficinas agrupadas por país.

Salida de nuestra consulta modificado

	country	Número de oficinas
1	USA	3
2	Australia	1
3	France	1
4	Japan	1
5	UK	1

Datos de la tabla offices

	officecode	country
1	1	USA
2	2	USA
3	3	USA
4	4	France
5	5	Japan
6	6	Australia
7	7	UK

Una nota sobre la consulta anterior, hemos tenido en cuenta que vender implica recibir un pedido. Es decir, no hemos atendido al estado del pedido: si se ha entregado, si está en estado “disputed”, si está “in process” o en estado de “on hold” … De todos modos, proporcionamos una query5_1 donde se tenga en cuenta la venta si y solo sí el pedido está en modo shipped. El resto de estados, se considera que el pedido se ha realizado, pero no se ha llevado a cabo una venta.

```
SELECT country,
       Count(*) AS "Número de oficinas"
FROM   offices o
WHERE  officecode NOT IN (SELECT DISTINCT officecode
                          FROM   employees,
                                 customers c
                          WHERE  c.customernumber IN
                                 (SELECT DISTINCT customernumber
                                  FROM   orders
                                  WHERE  orderdate >= '2003-01-01'
                                         AND orderdate <=
                                             '2003-12-31'
                                         AND status = 'Shipped')
                                 AND c.salesrepemployeenumber = employeenumber)
GROUP  BY country
ORDER  BY "Número de oficinas" DESC;
```

Consulta 6:

En esta última consulta, tenemos que seleccionar los pares de productos que aparecen en más de un carro de la compra.

```
WITH lista
     AS (SELECT o.productcode  AS producto1,
                o2.productcode AS producto2,
                o.ordernumber
         FROM   orderdetails o,
                orderdetails o2
         WHERE  ( o.ordernumber = o2.ordernumber
                  AND o.productcode < o2.productcode ))
SELECT producto1,
       producto2,
       Count(*) AS "Carros de Compra"
FROM   lista
GROUP  BY producto1,
          producto2
HAVING Count(*) > 1
ORDER  BY "Carros de Compra" ASC;
```

Lo primero que hacemos es crear una tabla temporal que sea el producto cartesiano de orderdetails consigo misma. Pero ese producto cartesiano habrá que limitarlo por cada carro de la compra diferente. Es decir, lo que hacemos es hacer todas las parejas posibles de productos por cada compra y almacenamos las tuplas en una tabla. Para evitar las repeticiones por intercambio de argumentos, añadimos el signo menor. La lógica detrás de eso, se puede ver fácil con un ejemplo. Si para un determinado order, tenemos dos

productos A y B, queremos evitar tener AB y BA. Solo queremos quedarnos con uno, por lo que el signo menor o mayor en su defecto, nos dejaría AB en el primer caso y BA en el segundo.

La siguiente parte del código, se basa en coger de esa tabla aquellos pares que aparezcan más de una vez en un carro de compra diferente (para eso utilizamos el having count(*)).

Para la comprobación del ejercicio, vamos a ver que, efectivamente, no se hacen más parejas de las que se debiera por carro de compra. Queremos ver que ese Select funciona correctamente:

```
SELECT o.productcode  AS producto1,
                o2.productcode AS producto2,
                o.ordernumber
        FROM    orderdetails o,
                orderdetails o2
        WHERE   ( o.ordernumber = o2.ordernumber
                  AND o.productcode < o2.productcode )
```

Veamos que para el pedido 10100 genera seis parejas diferentes donde, según se puede apreciar, ya hemos evitado el problema de tener parejas intercambiadas.

	producto1	producto2	ordernumber
1	S18_1749	S24_3969	10,100
2	S18_1749	S18_4409	10,100
3	S18_1749	S18_2248	10,100
4	S18_2248	S24_3969	10,100
5	S18_2248	S18_4409	10,100
6	S18_4409	S24_3969	10,100
7	S18_2325	S24_2022	10,101
8	S18_2325	S24_1937	10,101

Si observamos los datos de la tabla de orderdetails, vemos que para el pedido 10100 hay cuatro productos diferentes.

	ordernumber	productcode	quantityordered	priceeach	orderlinenumber
1	10,100	S18_1749	30	136	3
2	10,100	S18_2248	50	55.09	2
3	10,100	S18_4409	22	75.46	4
4	10,100	S24_3969	49	35.29	1
5	10,101	S18_2325	25	108.06	4
6	10,101	S18_2795	26	167.06	1

No solo se puede comprobar que están todas las parejas, es que siendo n el número de producto asociados a un determinado ordernumber, C(n, 2) es el número de parejas que se esperan. En nuestro caso, C(4, 2) = 6 tal y como se puede apreciar.

El resto del funcionamiento del programa lo hemos comprobado cogiendo una pareja que solo apareciera una vez y añadiéndosela a otro carro. Efectivamente, el resultado nos la incluía. El proceso contrario, véase coger una de 2 y sacarlo de algún carro, también nos daba resultados correctos.

4. Rediseño de la base de datos

Estos son el nuevo esquema de la base de datos y diagrama relacional con los cambios realizados:

customers (**customernumber**, customername, contactlastname, contactfirstname, phone, addressline1, addressline2, city, state, postalcode, country, creditlimit);

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

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

empofficehistory (**employeenumber** ⟶ employees.employeenumber, transisitondate, oldofficecode ⟶ offices.officecode, newofficecode ⟶ offices.officecode);

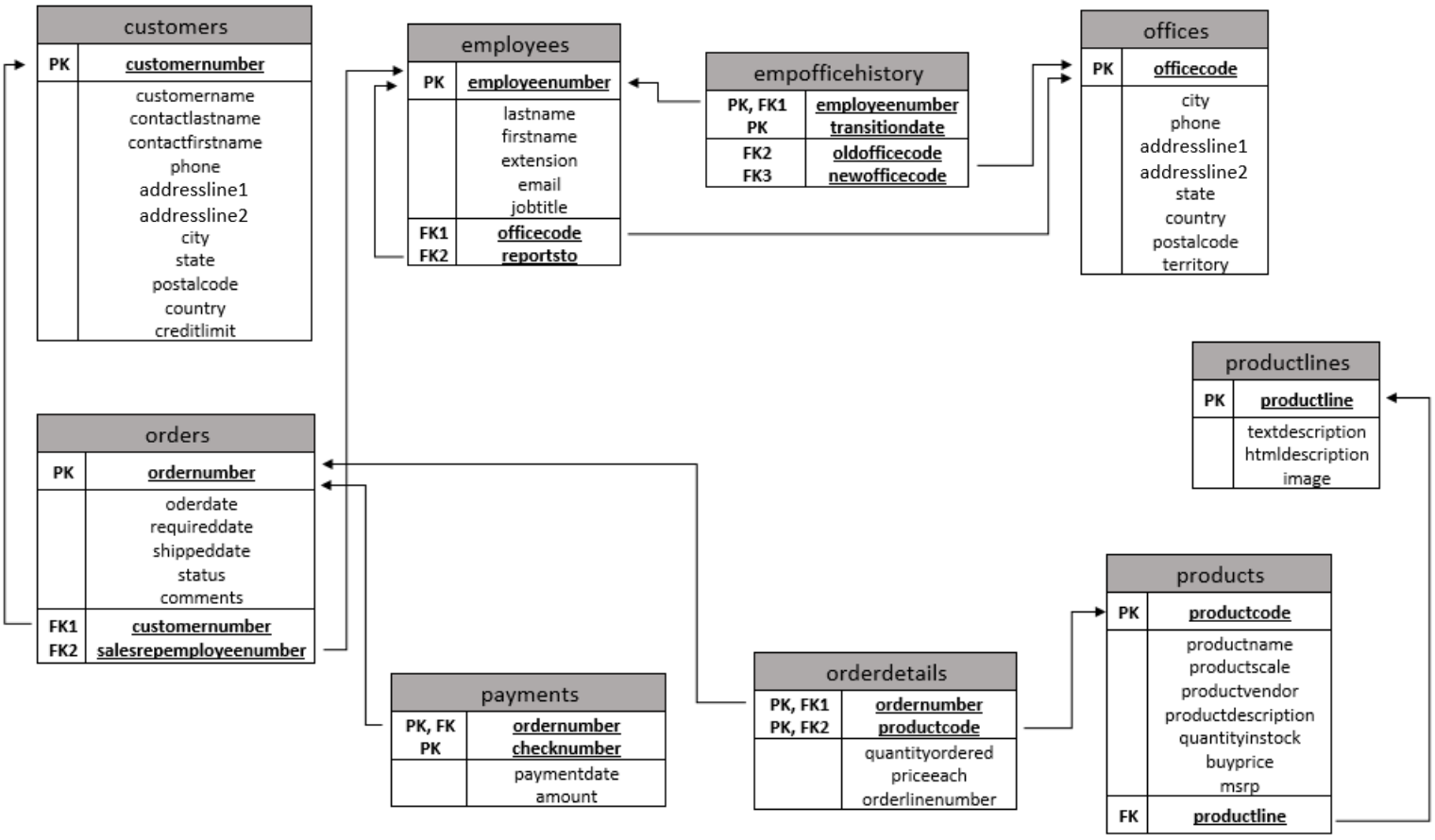
products (**productcode**, productname, productscale, productvendor, productdescription, quantiyinstock, buyprice, msrp, porductline ⟶ productlines.productline);

payments (**ordernumber** ⟶ orders.ordernumber, **checknumber**, amount, paymentdate);

orders (**ordernumber**, orderdate, requireddate, shippeddate, status, comments, customernumber ⟶ customers.customernumber, salesrepemployeenumber ⟶ employees.employeenumber);

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

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



Debido al diseño de la base de datos original (classicmodels), inevitablemente se pierde la posibilidad de guardar el historial oficinesco, es decir, las oficinas en la que habían trabajado en el pasado, y las transiciones de una a otra. Por otro lado, un cliente solo no podía relacionarse con diferentes empleados según el pedido y los pagos no estaban asociados a una compra en concreto.

Para resolver dichas deficiencias, hemos rediseñado la base de datos (nuevabase), realizando los siguientes cambios:

1) Los "customers" ya no están asociados a un "employee", sino a los orders. Para ello, customers ha perdido su clave foránea de "salesrepemployeenumber" y ha sido asignado a orders. En otras palabras, la manera que tienen de relacionarse un empleado y un cliente es a través de un pedido. A cada pedido se le asigna un empleado y, como un cliente puede realizar varios pedidos, entonces un cliente se puede relacionar con varios empleados dependiendo del pedido que vaya a realizar.

2) Análogamente, cada "payment" deja de estar asociado a un "customers", para estarlo a un order. La clave foránea de payments, "customernumber", que además formaba parte de su clave primaria, se convierte ahora en "ordernumber", la cual hace referencia a "orders", y también es clave primaria junto a "checknumber". De esta forma, cada compra se relaciona directamente con su correspondiente pago.

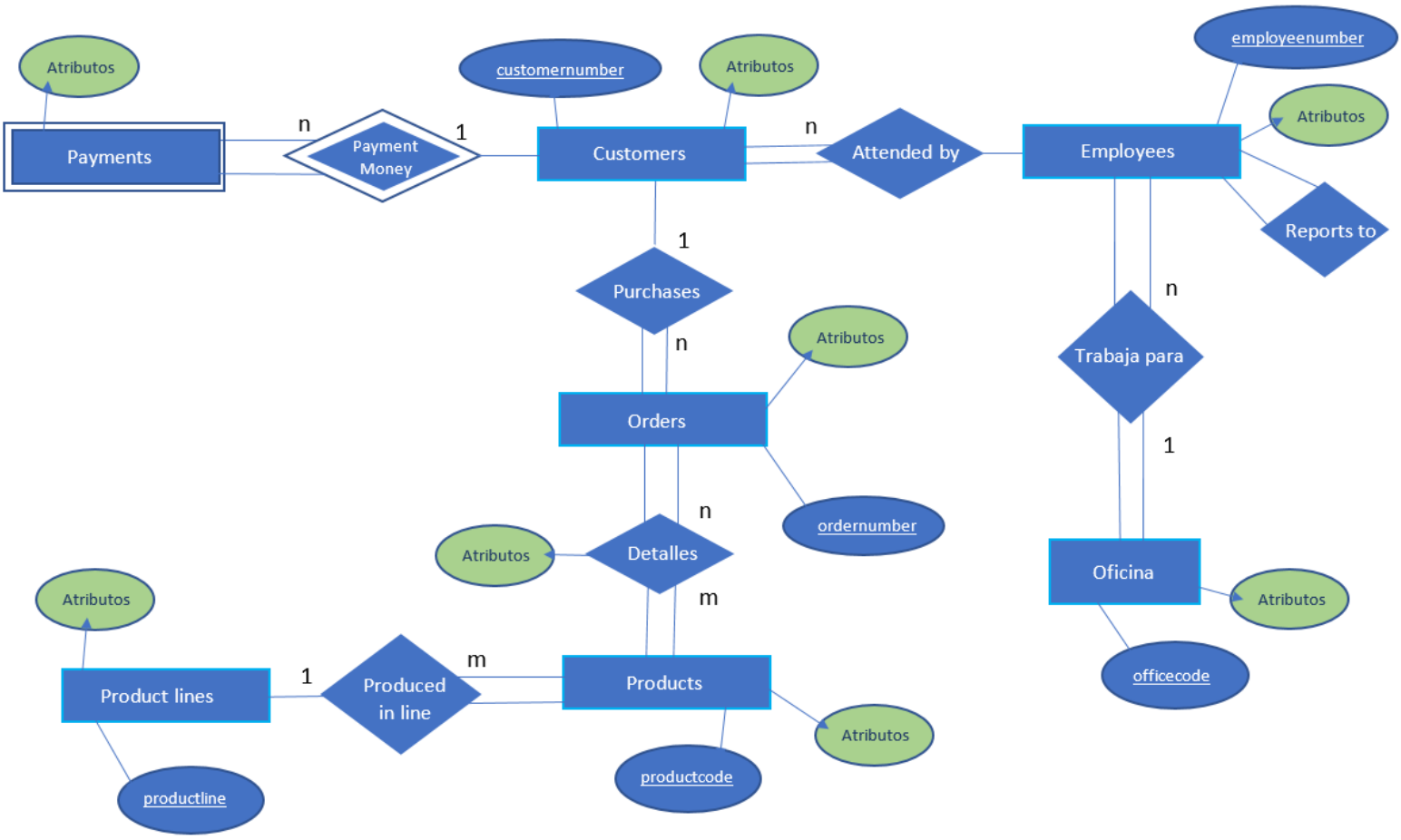
3) Finalmente, hemos creado una nueva relación entre los "employees" y las "offices" denominada "empofficehistory", la cual tiene como clave primaria a "employeenumber" y "transitiondate" (fecha de cambio de oficina), además de dos atributos "oldofficecode" y "newofficecode" que además son claves foráneas que hacen referencia a "offices". En las restricciones en la creación de la tabla en SQL, se permite que el campo oldofficecode sea NULL porque no todos los empleados tienen por qué haber cambiado de oficina. Sin embargo, todos tienen que trabajar en alguna oficina.

Dichos cambios solucionan las deficiencias anteriores y crean una nueva base completa y eficiente.

5. Modelos E/R

A continuación, se disponen los modelos Entidad-Relación de la base datos primera y luego del rediseño de la base de datos.

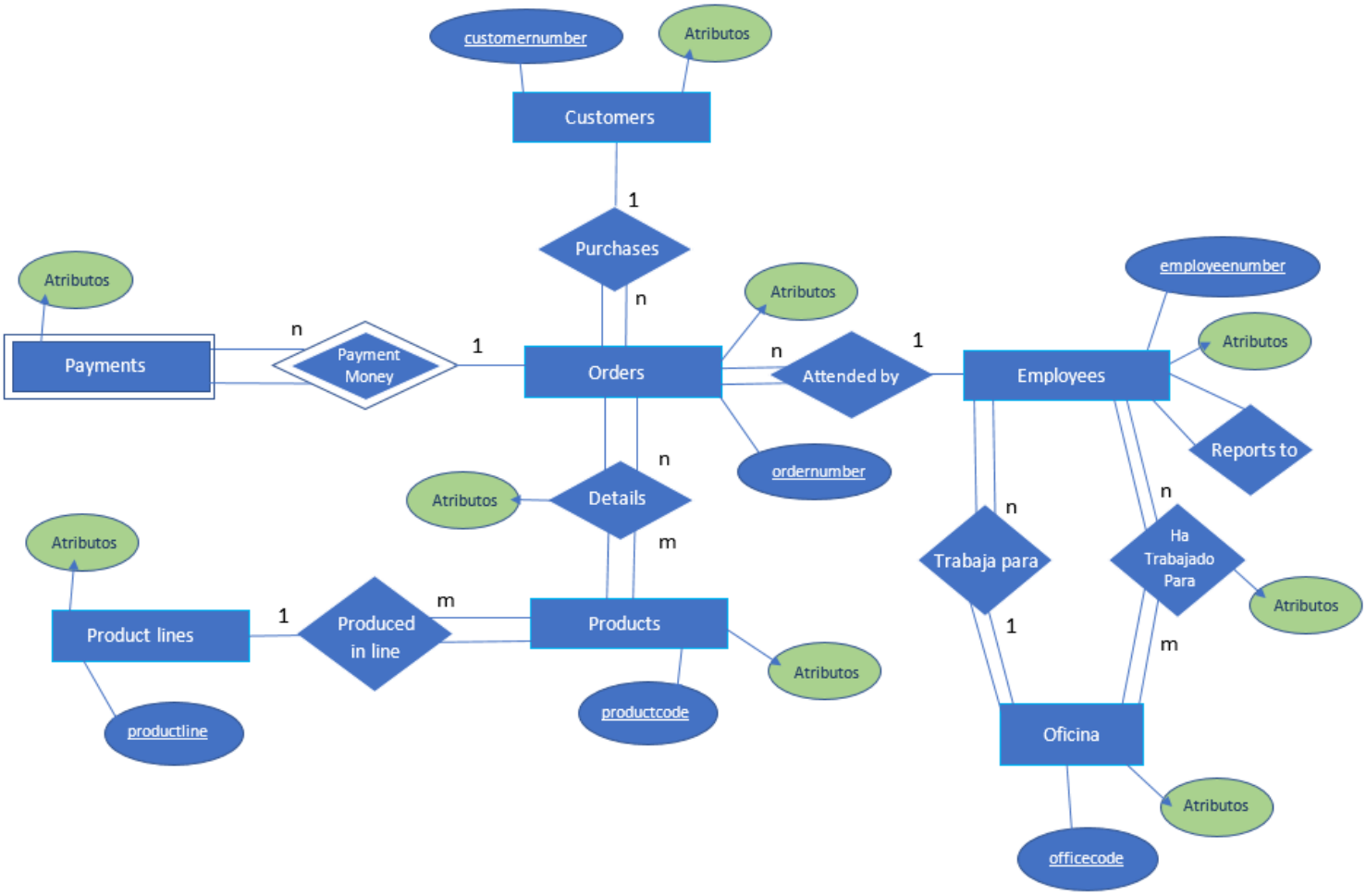
Base de datos antigua:



En los campos donde pone atributos, vamos a rellenarlos de manera literal a continuación:

- **Entidad payments:** checknumber, paymentdate, amount.
- **Entidad customer:** customername, contact (firstname, lastname), phone, address (addressline1, addressline2), location (city, state, postalcode, country), creditlimit.
- **Entidad employees:** name (lastname, firstname), extension, email, jobtittle.
- **Entidad offices:** location (city, state, country, postalcode, territory), phone, address (addressline1, addressline2).
- **Entidad orders:** deadline (orderdate, requiredate, shippeddate), status, comments.
- **Relación orderdetails (detalles):** quantityordered, priceeach, orderlinenumber.
- **Entidad products:** product (name, scale, vendor, description), quantityinstock, buyprice, msrp.
- **Entidad productlines:** description (text, html), image.

Base de datos rediseñada:



En los campos donde pone atributos, vamos a rellenarlos de manera literal a continuación:

- **Entidad payments:** checknumber, paymentdate, amount.
- **Entidad customer:** customername, contact (firstname, lastname), phone, address (addressline1, addressline2), location (city, state, postalcode, country), creditlimit.
- **Entidad employees:** name (lastname, firstname), extension, email, jobtittle.
- **Entidad offices:** location (city, state, country, postalcode, territory), phone, address (addressline1, addressline2).
- **Relación empofficehistory (ha trabajado para):** transitiondate.
- **Entidad orders:** deadline (orderdate, requiredate, shippeddate), status, comments.
- **Relación orderdetails (detalles):** quantityordered, priceeach, orderlinenumber.
- **Entidad products:** product (name, scale, vendor, description), quantityinstock, buyprice, msrp.
- **Entidad productlines:** description (text, html), image.

Notas:

Se ha modificado el makefile para poder generar la base de datos rediseñada de classicmodels mediante el comando "nuevabase". También se ha creado el comando "dumpnew" para poder guardar la base de datos en nuevabase.sql y así luego recuperarla mediante "restorenew".

Finalmente, se ha añadido el comando query5_1, que ejecuta la consulta 5 considerando una venta solo si el estado del producto es 'Shipped'.