Solución. Las consultas simples

Ejercicio 1

```
SELECT
idfab,idproducto,descripcion,precio,
(precio * 1.16) AS iva_incluido
FROM productos
```

Los paréntesis son opcionales, también se puede poner como fórmula de cálculo: precio + precio * 16 /100.

Ejercicio 2

```
SELECT numpedido, fab, producto,
cant, importe / cant AS
precio_unitario, importe
FROM pedidos
```

Ejercicio 3

```
SELECT nombre, date() - contrato
AS dias_trabajados, year(date()) -
edad AS año_nacimiento
FROM empleados
```

Aquí hemos utilizado la función date() que devuelve el día actual y hemos utilizado la diferencia de fechas para saber cuántos días han transcurrido entre las dos fechas. Para saber el año de nacimiento restamos al año actual la edad del empleado. Para obtener el año actual aplicamos la función year() (que devuelve el año de una fecha) sobre la fecha actual (date())

Ejercicio 4

```
SELECT *
FROM clientes
ORDER BY repclie
```

Ejercicio 5

```
SELECT *
FROM oficinas
ORDER BY region, ciudad, oficina
DESC
```

Ejercicio 6

```
SELECT *
FROM pedidos
ORDER BY fechapedido
```

Ejercicio 7

```
SELECT TOP 4 *
FROM pedidos
ORDER BY importe DESC
```

Para obtener las más caras tenemos que ordenar por importe y en orden descendente para que aparezca las más caras primero. Además como sólo queremos las cuatro primeras utilizamos la cláusula TOP 4.

Ejercicio 8

```
SELECT TOP 5 numpedido, fab,
producto, cant, importe / cant AS
precio_unitario, importe
FROM pedidos
ORDER BY 5
```

Ordenamos los pedidos por precio unitario utilizando el nº de columna, el precio unitario es la quinta columna dentro de la lista de selección. En este caso la ordenación debe ser ascendente.

Ojo!:
Obtiene **6 registros** con **top 5**, pero **no son los 5 valores diferentes de precio unitario**…, sino las 5 líneas de pedido con menor precio unitario. Consulta los apuntes: *"Si pido los 25 primeros valores pero,* ***el que hace la posición 26 es el mismo valor que el de la 25, entonces devolverá 26 registros en vez de 25"*** (en nuestro caso los registros se ordenan por precio unitario; una vez ordenados se toman las 5 primeras filas además, como el valor de la fila 6 es el mismo que el de la 5 se incluye dicha fila en el resultado y devuelve 6 filas en lugar de 5).

Esto es así …, independientemente de que en la lista resultado existan 2, 3, 4 o 5 valores diferentes del campo de ordenación! ***(Es un comportamiento extraño de access respecto a otros sgbd…)***

Ejercicio 9

```
SELECT *
FROM pedidos
WHERE MONTH(fechapedido) = 3
```

MONTH(fecha) devuelve el número de mes de la fecha.

Ejercicio 10

```
SELECT numemp
FROM empleados
WHERE oficina IS NOT NULL
```

Los empleados que tienen asignada una oficina son los que tienen un valor en el campo oficina.

Ejercicio 11

```
SELECT oficina
FROM oficinas
WHERE dir IS NULL
```

El campo dir es el que nos dice quien es el director de la oficina.

Ejercicio 12

```
SELECT *
FROM oficinas
WHERE region IN ('norte','este')
ORDER BY region DESC
```

Los valores se ponen entre comillas simples o dobles ya que son valores alfanuméricos. También se puede poner WHERE region = 'norte' OR region = 'este'.
Ordenamos desc para que primero aparezcan las del norte.

Ejercicio 13

```
SELECT *
FROM empleados
WHERE nombre LIKE 'Julia *'
```

Los empleados cuyo nombre empiece por Julia, observar que antes del * hay un espacio en blanco para forzar a que el siguiente carácter después de la a sea un blanco y no coja por ejemplo Julian.

Ejercicio 14

```
SELECT *
FROM productos
WHERE idproducto LIKE '*x'
```

Solución. Las consultas multitabla

Ejercicio 1

```
SELECT oficinas.oficina, ciudad, numemp,
nombre
FROM oficinas INNER JOIN empleados ON
oficinas.oficina = empleados.oficina
WHERE region = 'este'
```

Como la columna de emparejamiento oficinas.oficina es clave principal en la tabla oficinas, es mejor utilizar el JOIN que un producto cartesiano. Emparejamos las dos tablas por el campo oficina. Las oficinas que no tengan

empleados no salen (es un INNER).

Como queremos sólo las oficinas del *este* añadimos la cláusula WHERE con la condición. El valor *este* debe ir entre comillas (es un valor alfanumérico).

Observar que en la lista de selección la columna oficina está *cualificada* (su nombre está precedido del nombre de la tabla), este detalle es necesario porque en las dos tablas existe una columna llamada *oficina* y el sistema no sabría cuál de las dos escoger.

```
SELECT oficinas.oficina, ciudad,
numemp, nombre
FROM oficinas LEFT JOIN empleados
ON oficinas.oficina =
empleados.oficina
WHERE region = 'este'
```

Si queremos que también aparezcan las oficinas que **no** tienen empleados cambiamos INNER por LEFT. Esto añade al resultado del inner, las filas de la tabla oficinas que no tienen correspondencia en la tabla empleados. *(Ejem. La oficina 28 es del este y no tiene correspondencia en empleados)*. Observar que la tabla oficinas está a la izquierda de la palabra JOIN.

Ojo!, si en la lista de selección ponemos empleados.oficina en vez de oficinas.oficina, en las filas de oficinas que no tienen empleados el número de oficina aparece nulo.

```
SELECT oficinas.oficina, ciudad, numemp,
nombre
FROM empleados RIGHT JOIN oficinas ON
oficinas.oficina = empleados.oficina
WHERE region = 'este'
```

Esta SELECT es equivalente a la anterior pero hemos cambiado *LEFT por RIGHT* porque ahora la tabla oficinas está a la derecha de la palabra JOIN.

```
SELECT oficinas.Oficina, oficinas.Ciudad,
empleados.Numemp, empleados.Nombre

FROM oficinas, empleados

WHERE
(((oficinas.Oficina)=[empleados].[oficina])
AND ((oficinas.Region)='este'));
```

Ejercicio 2

```
SELECT numpedido, importe,
clientes.nombre AS cliente,
limitecredito
FROM pedidos INNER JOIN clientes
ON pedidos.clie = clientes.numclie
```

En este ejercicio no pueden haber pedidos sin cliente, y lo que nos interesa son los pedidos, luego tampoco tienen que aparecer los clientes que no tienen pedidos, por lo tanto utilizamos un INNER JOIN.

Ejercicio 3

```
SELECT empleados.*, ciudad, region
FROM empleados LEFT JOIN oficinas
ON empleados.oficina =
oficinas.oficina
```

Aquí hemos utilizado LEFT JOIN para que también salgan los empleados que no tienen oficina asignada (el 110). Si lo hacemos con inner no saldrá dicho empleado.
Como queremos todos los datos del empleado utilizamos *empleados.** para abreviar.

Ejercicio 4

```
SELECT oficinas.*, nombre AS
director
FROM empleados RIGHT JOIN oficinas
ON empleados.numemp = oficinas.dir
WHERE objetivo > 600000
```

Nos interesan las oficinas con objetivo superior a 600.000pts., luego nos tenemos que asegurar que salgan todas incluso si no tienen director asignado por eso utilizamos RIGHT JOIN.
Ojo!: en los valores numéricos no utilizar el punto para separar los miles (lo consideraría coma decimal y entendería 600 en vez de 600000).

```
SELECT oficinas.*, nombre AS
director

FROM oficinas LEFT JOIN empleados
ON empleados.numemp = oficinas.dir

WHERE objetivo > 600000;
```

Ejercicio 5

```
SELECT numpedido, importe,
empleados.nombre AS representante,
clientes.nombre AS cliente
FROM (pedidos INNER JOIN clientes
ON pedidos.clie = clientes.numclie)
INNER JOIN empleados ON
pedidos.rep = empleados.numemp
WHERE importe > 25000
```

En este ejercicio no pueden haber pedidos sin representante ni cliente, y lo que nos interesa son los pedidos, luego tampoco tienen que aparecer los representantes que no tienen pedidos ni los clientes que no tienen pedidos, por lo tanto utilizamos un INNER JOIN.

Primero añadimos a cada línea de pedido los datos del cliente corespondiente (con el primer INNER) y a cada fila resultante añadimos los datos del representante correspondiente.

Nota: el representante que nos interesa es el que ha realizado el pedido y ese dato lo tenemos en el campo rep de pedidos por eso la condición de emparejamiento es pedidos.rep = empleados.rep.

Si hubiésemos querido el nombre del representante asignado al cliente, la condición hubiera sido clientes.repclie =

empleados.numemp.

Ejercicio 6

```
SELECT empleados.*
FROM empleados INNER JOIN
pedidos ON pedidos.rep =
empleados.numemp
WHERE fechapedido = contrato
```

Los representantes que buscamos tienen un pedido con la misma fecha que la de su contrato, tenemos que añadir a los pedidos los datos del representante correspondiente para poder comparar los dos campos.

Ejercicio 7

```
SELECT empleados.*, jefes.numemp
AS num_jefe, jefes.nombre AS
nombre_jefe, jefes.cuota AS
cuota_jefe
FROM empleados INNER JOIN
empleados jefes ON empleados.jefe =
jefes.numemp
WHERE empleados.cuota >
jefes.cuota
```

En una misma línea necesito los datos del empleado y los datos de su jefe, luego tengo que combinar empleados con empleados.

No interesan los empleados que no tienen jefe, por eso utilizo INNER.

El alias de tabla es obligatorio ya que combino la tabla empleados con ella misma.

```
SELECT empleados.*, jefes.numemp ,
jefes.nombre , jefes.cuota

FROM empleados INNER JOIN
empleados jefes ON empleados.jefe =
jefes.numemp

WHERE empleados.cuota > jefes.cuota
```

```
SELECT *

FROM empleados

where cuota> (select  cuota

                from empleados jefes

                  where
empleados.jefe = jefes.numemp)
```

Ejercicio 8

```
SELECT numemp
FROM empleados LEFT JOIN pedidos
ON pedidos.rep = empleados.numemp
WHERE importe > 10000 OR cuota <
10000
```

Una posible solución es combinar pedidos con empleados para poder seleccionar las líneas de importe > 10000 o cuota < 10000.

Hay que utilizar LEFT para que puedan aparecer empleados con cuota < 10000 que no tengan pedidos.

```
SELECT rep
FROM pedidos
WHERE importe > 10000
UNION
SELECT numemp
FROM empleados
WHERE cuota < 10000
```

Esta es otra solución, obtener por una parte los códigos de los empleados con una línea de pedido > 10000, por otra parte los códigos de los empleados con cuota < 10000 y finalmente unir las dos listas con una UNION.

Las consultas de resumen

Ejercicio 1

```
SELECT AVG(cuota) AS cuota_media,
AVG(ventas) AS ventas_media
FROM empleados
```

Sale una única fila con el resultado deseado. Siempre que se utilicen expresiones o funciones en la lista de selección, queda mejor utilizar un alias de columna para que ese aparezca en el encabezado del resultado.

Ejercicio 2

```
SELECT AVG(importe) AS
importe_medio, SUM(importe) AS
importe_total, AVG(importe/cant) AS
precio_venta_medio
FROM pedidos
```

El precio medio de venta es la media aritmética de los precios unitarios de cada pedido. El precio unitario se calcula dividiendo el importe del pedido por la cantidad del pedido: importe/cant, por lo que ponemos AVG(importe/cant).

Ejercicio 3

```
SELECT AVG(precio) AS p_medio_ACI
FROM productos
WHERE idfab = 'ACI'
```

Ahora no nos interesan todos los productos sino únicamente los del fabricante ACI, por lo que añadimos la cláusula WHERE para que antes de calcular la media, elimine del origen de datos los registros que no cumplan la condición.

Ejercicio 4

```
SELECT SUM(importe) AS
total_pedidos_V_Pantalla
FROM empleados INNER JOIN
pedidos ON empleados.numemp =
pedidos.rep
WHERE nombre = 'Vicente Pantalla'
```

El importe total lo tenemos que sacar de la tabla de pedidos, y además sólo nos interesan los de Vicente Pantalla. Como nos dan el nombre del representante en vez de su número y en el pedido sólo tenemos el número de representante tenemos que añadir a las líneas de cada pedido, los datos del representante correspondiente, por lo que el origen de datos debe ser el que aparece en la FROM.

Ejercicio 5

```
SELECT MIN(fechapedido) AS
primer_pedido
FROM pedidos
```

La fecha del primer pedido es la fecha más antigua de la tabla de pedidos.

Ejercicio 6

```
SELECT COUNT(*) AS
cuantos_pedidos_mayores
FROM pedidos
WHERE importe > 25000
```

Se podía haber utilizado también COUNT(numpedido) o cualquier nombre de columna que no pueda contener valores nulos, pero COUNT(*) es mejor por ser más rápido (la diferencia se nota con tablas muy voluminosas).

Ejercicio 7

Listar cuántos empleados están asignados a cada oficina, indicar el número de oficina y cuántos hay asignados.

```
SELECT oficina, COUNT(*) AS
cuantos_empleados
FROM empleados
GROUP BY oficina
```

Con esta solución obtenemos el listado pedido pero no aparecen las oficinas que no tienen empleados asignados ya que sacamos la información de la tabla empleados y aparece una fila con valor nulo en oficina que contiene el número de empleados que no tienen oficina. Si quisiéramos listar incluso las que no tengan empleados habría que recurrir a la solución 2

Solución 2

```
SELECT oficinas.oficina,
COUNT(numemp) AS
cuantos_empleados
FROM empleados RIGHT JOIN oficinas
ON empleados.oficina =
oficinas.oficina
GROUP BY oficinas.oficina
```

Utilizamos un RIGHT JOIN para que el origen de datos incluya también una fila por cada oficina que no tenga empleados.
En el GROUP BY y en la lista de selección hay que indicar el campo oficina de la tabla oficinas, si ponemos el de la tabla empleados, agrupará todas las oficinas que no tienen empleados en una fila (la columna empleados.oficina contiene valor nulo para esas filas).
Aquí no podemos utilizar COUNT(*) por que las oficinas sin empleados aparecerían con 1 en la columna cuantos_empleados ya que para esa oficina hay una fila.

Ejercicio 8

Para cada empleado, obtener su número, nombre, e importe vendido por ese empleado a cada cliente indicando el número de cliente.

Solución1

```
SELECT numemp, nombre, clie AS
cliente, SUM(importe) AS
total_vendido
FROM empleados LEFT JOIN pedidos
ON pedidos.rep = empleados.numemp
GROUP BY numemp, nombre, clie
```

Si queremos que salgan todos los empleados incluso los que no aparezcan en los pedidos habría que sustituir el INNER por un LEFT.

Solución2

```
SELECT numemp, nombre, clie AS
cliente, SUM(importe) AS
total_vendido
FROM empleados INNER JOIN
pedidos ON pedidos.rep =
empleados.numemp
GROUP BY numemp, nombre, clie
```

Necesitamos la tabla de pedidos para el importe vendido a cada cliente, necesitamos la tabla empleados para el nombre del representante, la de clientes no la necesitamos ya que nos piden el número de cliente y éste está en pedidos.
La agrupación básica que debemos realizar es por numemp y después por clie, pero como aparece el nombre del empleado en la lista de selección, hay que incluirlo también en el GROUP BY.
Después de determinar la agrupación básica que nos hace falta, siempre que se incluye una columna adicional en el GROUP BY hay que comprobar que esa nueva columna no cambia la agrupación básica.
Por ejemplo no podríamos añadir al GROUP BY la columna fechapedido ya que se formarían más grupos.

Ejercicio 9

Para cada empleado cuyos pedidos suman más de 30.000 ptas, hallar su importe medio de pedidos. En el resultado indicar el número de empleado y su importe medio de pedidos.

```
SELECT rep, AVG(importe) AS
importe_medio
FROM pedidos
GROUP BY rep
HAVING SUM(importe) > 30000
```

No queremos todos los empleados, únicamente los que tengan un importe total pedido superior a 30.000, luego tenemos que poner la condición SUM(importe) > 30000. Como esta condición contiene una función de columna (SUM()) se tiene que poner en la cláusula HAVING ya que selecciona filas de la tabla resultante no del origen de datos.

Ejercicio 10

Listar de cada producto, su descripción, precio y cantidad total pedida, incluyendo sólo los productos cuya cantidad total pedida sea superior al 75% del stock; y ordenado por cantidad total pedida.

```
SELECT productos.descripcion, precio, sum(pedidos.cant) AS cantidadtotalpedida, existencias
FROM pedidos INNER JOIN productos ON (productos.idproducto=pedidos.producto and
productos.idfab=pedidos.fab)
GROUP BY descripcion, precio, existencias
HAVING sum(cant)>existencias*0.75
ORDER BY 3;
```

```
SELECT productos.descripcion,
precio, sum(pedidos.cant) AS
cantidadtotalpedida
FROM pedidos INNER JOIN productos
ON productos.idproducto =
pedidos.producto
GROUP BY descripcion, precio,
existencias
HAVING sum(cant)>existencias*0.75
ORDER BY 3;
```

El uso de descripción y precio en la lista de selección, acompañando a una función de agregado, obliga a incluir dichos campos en la cláusula group by. Asimismo, la inclusión de existencias en la cláusula having nos obliga a incluir dicho campo también en el order by.

Para calcular el 75% de las existencias multiplicamos existencias por 0,75; observar que en la sentencia SQL hay que utilizar el punto para indicar los decimales.

Para indicar la columna de ordenación no podemos utilizar el alias, utilizamos el número de orden de la columna dentro de la lista de selección. En este caso la suma de importes es la tercera columna.

Ejercicio 11

Saber cuántas oficinas tienen empleados con ventas superiores a su cuota, no queremos saber cuáles sino cuántas hay.

```
SELECT COUNT(oficina)
FROM oficinas
WHERE oficina IN (
SELECT oficina
FROM empleados
where ventas > cuota
GROUP BY oficina
HAVING COUNT(oficina) > 0);
```

Otra opción, menos aconsejable es hacerlo mediante dos consultas:

Consulta1: distintas_oficinas

```
SELECT DISTINCT oficina
FROM empleados
WHERE ventas > cuota
```

Consulta11:

```
SELECT COUNT(oficina) AS
cuantas_oficinas
FROM distintas_oficinas
```

Si contamos las oficinas directamente de la tabla empleados nos salen 9 oficinas ya que la función COUNT (nb columna) cuenta los valores no nulos pero los valores repetidos los cuenta tantas veces como se repiten, como tenemos oficinas que se repiten en la columna oficina de la tabla oficinas, esas oficinas son contadas varias veces, y lo que nos interesa es contar los valores distintos.

En otros SQL la función COUNT puede llevar delante del nombre de la columna la cláusula DISTINCT que indica que sólo se tienen que tener en cuenta valores distintos (no cuenta los repetidos), por ejemplo COUNT(DISTINCT oficina), es una opción muy útil que desgraciadamente no incluye el SQL de Microsoft JET. Para solucionar el problema, en este SGBD debemos resolver dos consultas: una con la que obtenemos los valores distintos (en la solución la consulta se llama distintas_oficinas), y la otra que nos cuenta esos valores.

Como en estos ejercicios vamos a modificar los valores almacenados en la **base de datos EmpleadosOficinas**, es conveniente guardar antes una copia de las tablas, así que los ejercicios de este tema **los realizaremos sobre una**

copia de la base de datos anterior. A la copia le llamaremos **CopiaParaActualización**.

1. Subir un 5% el precio de todos los productos del fabricante ACI.

   UPDATE productos
   SET precio = precio * 1.05
   WHERE idfab = 'ACI';

   También se puede poner precio = precio + precio*0.05

2. Añadir una nueva oficina para la ciudad de Madrid, con el número de oficina 30, con un objetivo de 100000 y región Centro.

   INSERT INTO oficinas (oficina, region, ciudad, objetivo)
   VALUES (30, 'centro','Madrid', 100000);
   Como no asignamos valor a todos los campos, no hace falta poner todas las columnas en la lista de columnas. Los campos objetivo y ventas se rellenarán con el valor predeterminado, ya que estos campos en el diseño de tabla aparecen con valor predeterminado.

   INSERT INTO oficinas (oficina,region,ciudad,dir,objetivo,ventas)
   VALUES (30, 'centro', 'Madrid', null, 100000,0) ;
   Con esta solución nos aseguramos que el valor de dir sea nulo independientemente de su valor predeterminado y nos aseguramos que ventas sea igual a cero.

   INSERT INTO oficinas
   VALUES (30, 'Madrid', 'centro', null, 100000,0) ;
   En este caso, como no especificamos una lista de columnas tenemos que poner los valores en el mismo orden que las columnas en vista diseño de la tabla.

3. Cambiar los empleados de la oficina 11 a la oficina 30.
   UPDATE empleados SET oficina = 30 WHERE oficina = 21;
   Si ejecutamos esta sentencia antes de haber creado la oficina 30, el sistema nos devuelve un error.

4. Eliminar los pedidos del empleado 105.
   DELETE FROM pedidos WHERE rep = 105;

5. Eliminar las oficinas que no tengan empleados.

Solución 1

```
DELETE FROM oficinas
WHERE NOT EXISTS
        (SELECT *
        FROM empleados
        WHERE empleados.oficina = oficinas.oficina and
        empleados.oficina is not null);
```

Si la oficina no tiene empleados asignados, no existirá ningún empleado con el número de esa oficina (de ahí el uso de NOT EXISTS)

Solución 2

```
DELETE FROM oficinas
WHERE oficina NOT IN
        (SELECT oficina
        FROM empleados where oficina is not null) ;
```

Así mostramos las oficinas cuyo número no se encuentra entre las oficinas asignados a empleados.

6. A los empleados de la oficina *30* asignarles la oficina 11

```
UPDATE empleados
SET oficina = 11 WHERE oficina = 30;
```