

UD5. SQL II : SELECT

Índice

Introducción.....	3
El SELECT BÁSICO.....	3
Select de expresiones.....	5
WHERE.....	7
Operadores de comparación.....	8
Uso de Funciones.....	9
Funciones numéricas.....	10
Funciones de cadenas de caracteres.....	12
Funciones de fechas.....	17
JOINS.....	20
JOIN ON, JOIN USING.....	26
NATURAL JOIN.....	28
OUTER JOIN.....	28
GROUP BY.....	29
Funciones de Agregación.....	30
HAVING.....	30
SUBCONSULTAS.....	35
IN.....	35
ANY y ALL.....	38
Subconsultas correlacionadas.....	38
EXISTS.....	39
El problema de los nulos.....	39
SELECTS ANIDADAS.....	41
Limitando el número de resultados.....	41
Rownum.....	41
Paginar.....	41
Offset... fetch first.....	43
ROW_NUMBER() y SELECT OVER.....	43
COSULTAS CON PK COMPUESTA.....	44
JOIN de varias veces la misma tabla.....	47
OPERACIONES CON CONJUNTOS.....	49
INSERT, UPDATE, DELETE USANDO SELECTS.....	51

Introducción

Las bases de datos no tendrían ninguna utilidad si no se pudiera acceder a ellas, hacer operaciones y analizarlas para tomar decisiones. Las consultas son las operaciones que permiten mostrar los registros de las tablas, así como hacer operaciones sobre ellos. Las consultas son las operaciones que más se efectúan en la vida de la base de datos. Los desarrolladores de software que almacenan los datos en bases de datos y que acceden a ellos a través de sistemas gestores de bases de datos tienen que embeber en los programas que desarrollan infinidad de consultas asociadas a los diferentes elementos que se pueden encontrar en sus formularios, botones, listas desplegables, cuadros de textos, cuadriculas y cualquier tipo de proceso en la programación de sus aplicaciones.

El SELECT BÁSICO

El comando que se usa para la consulta de datos en la mayoría de las bases de datos relacionales es SELECT. La sintaxis del comando es compleja, pero se podría comenzar reduciéndola como sigue:

```
SELECT [DISTINCT | ALL] {*} | columna[,columnas] | expresión}
FROM tabla
[WHERE condicion-where]
[GROUP BY columna[,columnas] ]
[HAVING condición-having]
[ORDER BY columna {DESC | ASC} [,columnas [{DESC | ASC}]]];
```

En cualquier consulta, las cláusulas SELECT y FROM son obligatorias. La primera se usa para indicar las columnas que se necesitan proyectar y la segunda, para indicar cuáles son las tablas que deben participar para proyectar dichas columnas y hacer operaciones con dichos campos. Cualquier tabla será incluida, inevitablemente, si posee alguna columna que se requiere proyectar a través del SELECT. Las tablas incluidas en la cláusula FROM deben ser las suficientes y necesarias, ni una más ni una menos, ya que cada una implica aumentar el número de registros en la selección.

La cláusula WHERE también es muy usada, en pocas consultas no aparecerá. Esta cláusula permite seleccionar, de todos los registros que se combinan con las tablas declaradas en FROM, aquellos que interesan para la operación que se persigue. Tendrá siempre una operación lógica que filtra los registros que retornar.

La cláusula ORDER BY permite ordenar los resultados a través de una o más columnas y de forma ascendente o descendente.

Si vemos el esquema de la página siguiente, una consulta que puede obtener el nombre y apellido de los clientes sería:

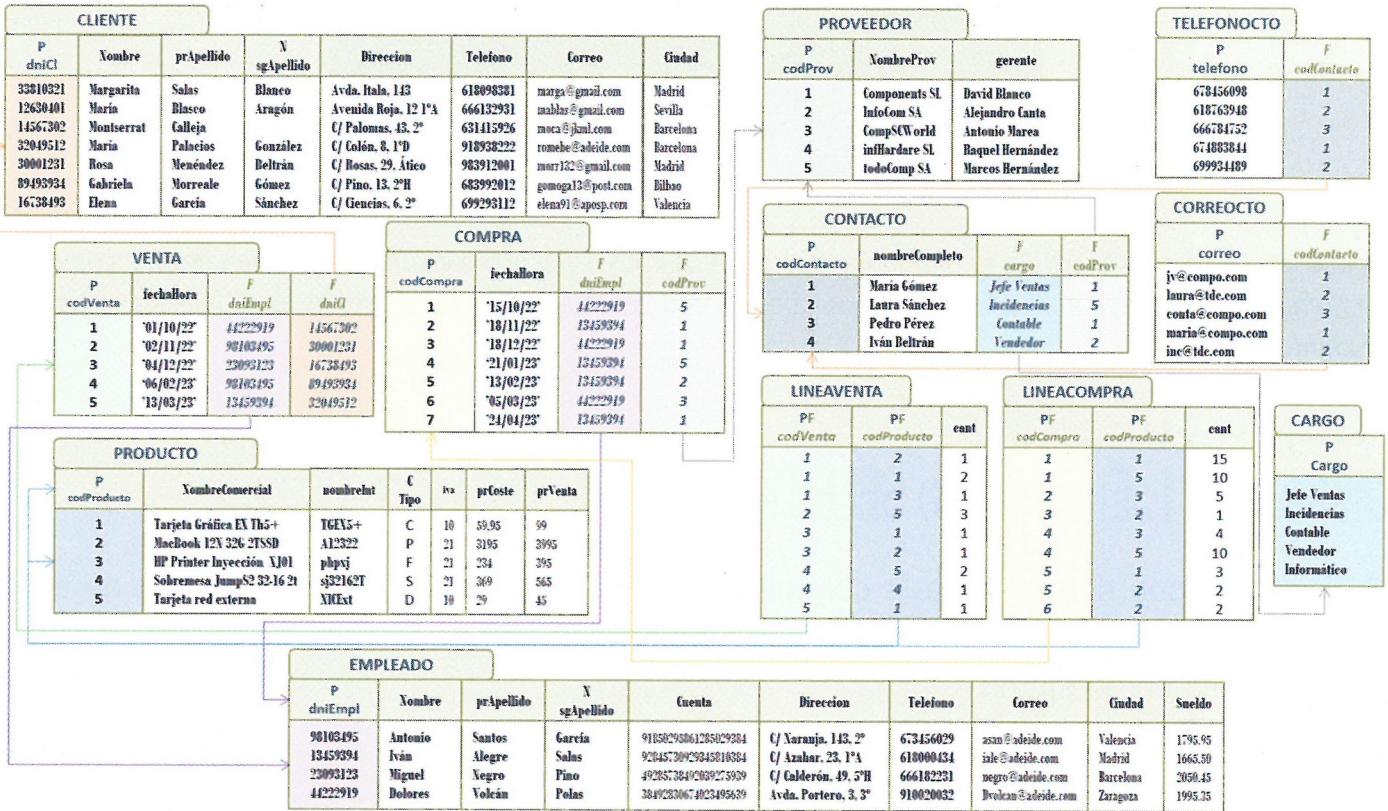
```
SELECT nombre, prApellido, sgApellido FROM Cliente;
```

Para mostrar todas las columnas de las tablas que se indican en el FROM, se usa el carácter *. De este modo, no se necesita indicar cada una de las columnas. La siguiente consulta muestra todos los registros de la tabla Cliente, mostrando todas sus columnas:

select * from cliente;

Esta consulta equivale a esta otra, donde se indica el nombre de cada una de las columnas de la tabla Cliente:

SELECT dniCl, Nombre, prApellido, sgApellido, Direccion, Telefono, Correo FROM Cliente;



El resultado de una consulta se puede ordenar según una columna u otra. Para ello, se usa la cláusula ORDER BY. Si la columna por la que se ordena es numérica, se ordenará del valor menor al mayor y, si es alfabética, se ordenará por orden alfabético, teniendo en cuenta la tabla de caracteres ASCII, donde las letras en mayúsculas están antes que las letras en minúsculas.

La siguiente consulta muestra el nombre y los apellidos de los clientes ordenados alfabéticamente por el primer apellido:

SELECT Nombre, prApellido, sgApellido FROM Cliente ORDER BY prApellido;

No es necesario que se seleccione la columna por la que se ordena. Por ejemplo, la siguiente consulta muestra el nombre y apellidos de los clientes, pero ordenados por el DNI, aunque el DNI de los clientes no se selecciona:

SELECT Nombre, prApellido, sgApellido FROM Cliente ORDER BY dni;

También se puede ordenar usando más de una columna. De esta manera, la ordenación se hace primero por la primera columna; a continuación, por la segunda, y así sucesivamente. La siguiente consulta muestra todas las columnas de la tabla Cliente, ordenados por primer apellido y luego por segundo apellido:

```
SELECT Nombre, prApellido, sgApellido FROM Cliente ORDER BY prApellido, sgApellido;
```

Para facilitar la declaración de la cláusula ORDER BY, esta permite indicar la columna por la que se quiere ordenar de las indicadas en el predicado SELECT. Por ejemplo, la siguiente consulta muestra el DNI, el nombre, los apellidos y la dirección de los clientes y se ordena por la quinta columna que se está mostrando, es decir, dirección:

```
SELECT dni, Nombre, prApellido, sgApellido, direccion FROM Cliente ORDER BY 5;
```

El orden puede ser alterado si se usa la palabra DESC en la cláusula ORDER BY, así, los valores numéricos se mostrarán de mayor a menor y los alfabéticos de orden descendiente, es decir, desde la z a la A. La siguiente consulta muestra el nombre y apellidos ordenados por el primer apellido descendientemente:

```
SELECT Nombre, prApellido, sgApellido FROM Cliente ORDER BY prApellido DESC;
```

PRÁCTICA PROPUESTA

CONSULTAS CON SALIDAS ORDENADAS

Para la base de datos que se muestra en la Figura Y que corresponde con la última versión del caso «Tienda de informática», realiza las siguientes consultas:

- Mostrar el nombre comercial e interno de los productos ordenados por el precio de coste.
- Mostrar el código del producto y el nombre interno ordenados del más caro al más barato.
- Mostrar los datos de los clientes ordenados por el teléfono.
- Mostrar el nombre y apellidos de los empleados ordenados por DNI.
- Mostrar el código y el nombre del proveedor ordenados por el nombre del gerente.

Select de expresiones

Hemos visto que en el SELECT, se pueden seleccionar expresiones, por ejemplo cálculos aritméticos con los valores de columnas de las tablas declaradas en la cláusula FROM, pero no se pueden hacer operaciones lógicas ni relacionales. Los operadores aritméticos son +, -, *, % y /, donde el simbolo * es el que se usa para multiplicar dos valores numéricos y % para obtener el resto de una división entera.

La siguiente consulta muestra lo que se pretende ganar de cada uno de los productos, resultado que se obtiene de restar al precio de venta el de coste:

```
SELECT NombreComercial, prVenta-prCoste FROM Producto;
```

Como se puede comprobar en la consulta anterior, la columna segunda se seleccionará con la cabecera prVenta-prCoste. Existe la posibilidad de cambiar el nombre de la cabecera de esa columna para la salida de una consulta. Esto no quiere decir que se cambie el nombre de la columna, sino que, para la consulta en cuestión, el nombre de dicha columna es otro. Para ello, se usa la palabra **as** detrás de la columna. Esto se denomina **alias de columna**. La palabra as puede omitirse en Oracle. Si el alias tiene espacios en blanco habrá que usar comillas dobles.

Por ejemplo, la columna de la operación prVenta-prCoste de la consulta anterior podría quedar con el alias Ganancias. La consulta sería:

```
SELECT NombreComercial, prVenta-prCoste as Ganancias FROM Producto;
```

Cuando se hace una operación aritmética sobre una columna que contiene un valor NULL, el resultado es NULL. Podemos usar la función **NVL(expresion,valorinull)** que devuelve el valor de la expresión si no es nula o valorinull si la expresión es nula.

Si se usa el simbolo * para proyectar todas las columnas, no se pueden declarar cálculos ni poner ninguna otra columna de la tabla. Por ejemplo, el siguiente comando seria erróneo:

```
SELECT *, prVenta-prCoste FROM Producto;
```

Cuando en la salida de una consulta se repiten los mismos valores, se pueden resumir para que no se repitan. Por ejemplo, si se observa la columna codProv de la tabla Compra, se puede deducir que, en dicha columna, se tiene el código de los proveedores a los que se les ha hecho alguna compra. De este modo, para conocer los diferentes proveedores a los que se les ha comprado alguna vez algún producto, basta con mostrar la columna codProv de la tabla Compra. La consulta seria:

```
SELECT codProv FROM Compra;
```

La salida de esta consulta muestra la columna completa codProv de la tabla Compra. Si se ha realizado más de una compra al mismo proveedor, este código sale repetido. Pues bien, con el predicado DISTINCT, dicho código solo se muestra una vez, obteniéndose así los diferentes proveedores sin repetir a los que se les ha comprado algún producto.

La consulta seria:

```
SELECT DISTINCT codProv FROM Compra;
```

Ojo al DISTINCT, no elimina la repetición de la columna que está a su lado sió de toda la fila. Si seleccionamos varios campos, el distinct elimina las tuplas repetidas y no solo un campo.

PRÁCTICA PROPUESTA

CONSULTAS CON ALIAS

- a) Mostrar sin repetir el DNI de los clientes que han hecho alguna compra.
- b) Mostrar el precio de coste y el precio de venta más el 21 % de IVA, usando los alias "precio de coste más IVA" y "Precio de venta más IVA"
- c) Mostrar el DNI, nombre, apellidos y cuenta de los empleados, usando los alias DNI del Empleado, Primer apellido Empleado, Segundo apellido Empleado y Cuenta bancaria.

También podemos usar expresiones sobre cadenas de caracteres. El operador || concatena cadenas de caracteres.

Así por ejemplo, listar los nombres y primeros apellidos de los clientes en un campo llamado "nombre completo" sería

```
SELECT Nombre || prApellido as "nombre completo" FROM Cliente;
```

o incluso con cadenas literales

```
SELECT 'el nombre es : ' || Nombre || prApellido as "nombre completo" FROM Cliente;
```

WHERE

La cláusula Where se usa para seleccionar de toda una tabla aquellos que cumplen una condición. Esta condición estará compuesta por una o más expresiones lógicas o relacionales. Cuando se ejecuta una sentencia SELECT, se recorre toda la tabla registro a registro y, para cada uno de ellos, se verifica si se cumple la condición. Aquellos que la cumplan serán seleccionados. Es por eso que, en la condición, solo pueden usarse columnas de las tablas que se declaran en la cláusula FROM.

Por ejemplo, en el caso de que se quisieran mostrar aquellos productos que se venden a más de 200 euros, se ejecutaría la consulta:

```
SELECT * FROM Producto WHERE pVenta>299;
```

No es necesario que las columnas que participen en la condición estén también en el SELECT. Lo que se debe tener en cuenta es que, de cualquier columna que se usa en una consulta, su tabla debe estar en la cláusula FROM.

Atendiendo a las columnas que se quieren seleccionar y los filtros que se quieren realizar, lo más difícil es saber cuál o cuáles son las tablas que consultar. Por ejemplo, si se quisiera mostrar el código de aquellos productos que se han comprado en alguna compra más de 10 unidades, la pregunta que habría que hacerse es: ¿en qué tabla se almacena la cantidad comprada de un producto determinado? La respuesta es fácil, la columna cant de la tabla lineaCompra, almacena esta información. Debido a que se busca es el código del producto y lo que dicho campo está en la misma tabla, no se hace necesario usar más tablas. La consulta quedaría como sigue:

```
SELECT codProducto, cant FROM LineaCompra WHERE cant>193
```

Sin embargo, si la consulta requiere el nombre del producto, se hace necesario incluir en la consulta la tabla Producto y usar técnicas para cruzar las tablas que se verán más adelante.

PRÁCTICA RESUELTA

CONSULTAS SIMPLES

- a) Mostrar el nombre y apellidos de los clientes que residen en Barcelona.
- b) Mostrar los teléfonos del contacto con código 1.
- c) Mostrar los correos del contacto con código 1.
- d) Mostrar el nombre y apellidos de los empleados que han nacido en Madrid.
- e) Mostrar el nombre completo de los contactos que son contables.
- f) Mostrar el nombre completo de los contactos que trabajan en el proveedor con código 1.
- g) Mostrar el código y la fecha de las compras hechas al proveedor con código 5.
- h) Mostrar el nombre comercial de los productos cuyo precio de coste se encuentre en el rango 20—50.
- i) Mostrar el nombre comercial y el código de los productos cuyo precio de venta sea mayor que el doble del de coste.

Operadores de comparación

Para evaluar las condiciones del Where hay que usar operadores de comparación que produzcan expresiones evaluables a verdadero o falso.

<	el operando de la izquierda es menor que el de la derecha
<=	el operando de la izquierda es menor o igual que el de la derecha
>	el operando de la izquierda es mayor que el de la derecha
>=	el operando de la izquierda es mayor o igual que el de la derecha
==	el operando de la izquierda es igual que el de la derecha
!= o <>	el operando de la izquierda es distinto que el de la derecha
Is null	El operando de la izquierda es nulo
Is not null	El operando de la izquierda es no nulo (tiene un valor)
IN(...)	El operando de la izquierda contiene algún valor de los listados entre comas dentro del IN
NOT IN(...)	El operando de la izquierda NO contiene NINGÚN valor de los listados entre comas dentro del IN
Between... and ...	El operando de la izquierda tiene un valor entre el valor a la izquierda y derecha del and
Like	El operando de la izquierda se compara con una expresión regular de la derecha

PRÁCTICA RESUELTA

CONSULTAS CON OPERADORES

- Mostrar el DNI de los clientes que no son de Málaga, Córdoba o Santander, que no se llaman ni Rosa ni Elena y cuyo primer apellido es Salas. La salida se mostrará en orden descendente en función del segundo apellido.
- Mostrar el DNI de los empleados que son de Sevilla, de Barcelona o de Madrid y que no tengan segundo apellido. La salida se mostrará en orden al DNI.
- Mostrar el código del producto de aquellos que, en una línea de venta, se hayan vendido con una cantidad superior o igual a 3. La salida se mostrará en orden descendente según el código del producto.
- Mostrar nombre y apellidos con los alias Nombre, Primer apellido y Segundo apellido de aquellos clientes que son de Barcelona, que se llaman María, Rosa o Gabriela y cuyo primer apellido es Blasco, Palacios o García o su segundo apellido es Beltran o Gómez. La salida se mostrará en orden descendente según el segundo apellido

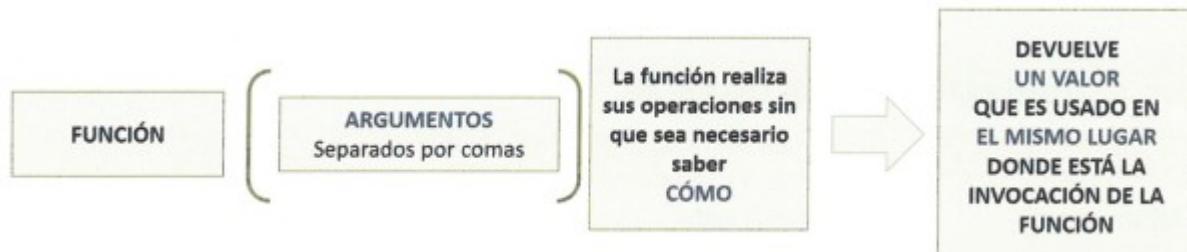
Uso de Funciones

Una función en un lenguaje informático es un mecanismo que acepta un conjunto de valores y devuelve un resultado. Los valores de entrada de una función se llaman argumentos. Una función puede no tener ningún argumento de entrada, uno o más de uno.

Todos los lenguajes de programación disponen de un conjunto muy amplio de funciones que permiten realizar operaciones con un conjunto de datos. Estas funciones se denominan funciones del lenguaje. Por ejemplo, la función ABS calcula el valor absoluto de un número, la función LENGTH calcula la longitud de una cadena y la función SUM calcula la suma de una columna. Estas funciones pueden tomar n valores de entrada, llamados argumentos, y devuelven un valor.

Dependiendo del tipo de datos que admite, las funciones pueden ser de una sola fila, funciones agregadas, funciones analíticas, funciones de referencia de objetos, funciones del modelo y otras llamadas funciones definidas por el usuario. Estas últimas son las que el programador desarrolla para sus aplicaciones y son complementarias a las funciones del sistema.

El valor que devuelve una función puede almacenarse en una variable o pertenecer a una expresión completa.



Por ejemplo, la función UPPER admite un argumento de tipo cadena. La función devuelve un resultado, la cadena convertida en mayúsculas. Este resultado se puede almacenar en una variable o comportar parte de una expresión, por ejemplo, UPPER(Ciudad)='MADRID', pero el valor original de la ciudad no ha sido modificado. El operador = en esta expresión comprueba si la mayúscula del argumento Ciudad es igual a MADRID. Así, UPPER(nombre) devuelve el valor de nombre convertido en mayúsculas, pero nombre sigue manteniendo su valor. La siguiente consulta muestra el nombre y apellidos de los clientes escritos en mayúsculas:

```
SELECT UPPER(Nombre), UPPER(prApellido), UPPER(sgApellido) FROM Cliente;
```

Si la función no requiere ningún argumento, no hace falta usar los paréntesis, por ejemplo, la función SYSDATE, que devuelve la fecha actual del sistema.

Como todas las consultas necesitan obligatoriamente usar la cláusula FROM, a veces, para consultar valores de algunas variables del sistema o de los usuarios, o usar una función que no necesita indicar ninguna tabla, se hace uso de una tabla que registro y que está creada para hacer estas acciones denominada DUAL. De esta manera, para comprobar la fecha actual del sistema se puede ejecutar la consulta

```
SELECT SYSDATE FROM Dual;
```

Funciones numéricas

Las funciones numéricas tienen como argumentos de entrada valores que son numéricos y devuelven otro número tras realizar cálculos con los datos de entrada. Para el lenguaje SQL estas funciones de una sola fila devuelven una sola fila por cada fila que se evalúa en una consulta. Estas funciones pueden estar tanto en la cláusula SELECT como en la cláusula WHERE. Estas funciones son ABS, ACOS, ASIN, ATAN, ATAN2, BITAND, CEL,COS, COSH, EXP, FLOOR, LN, MOD, NANVL, POWER, ROUND, SIGN, SINH, SQRT, TANH, TRUNC y WIDTH_BUCKET. Para los valores decimales, se usa el punto . y no la coma ,.

Función	Descripción	Ejemplo	Observación
ABS(numero)	Devuelve el valor absoluto de <i>numero</i>	<code>SELECT ABS(10) FROM DUAL;</code>	Devuelve 10
ASIN(numero), ACOS(numero), ATAN(numero)	Calcula el valor arco del seno, coseno o tangente	<code>SELECT ASIN(-0.86) FROM DUAL;</code>	Devuelve menos 59,3165
CEIL(numero)	Devuelve el entero más pequeño que es mayor o igual que <i>numero</i>	<code>SELECT CEIL(9.19) FROM DUAL;</code>	Devuelve 10
EXP(numero)	Devuelve 2,718281 elevado a <i>numero</i>	<code>SELECT EXP(2) FROM DUAL;</code>	Devuelve 7,389056
FLOOR(numero)	Devuelve el defecto entero de <i>numero</i>	<code>SELECT FLOOR(9.95) FROM DUAL;</code>	Devuelve 9
MOD(numero1, numero2)	Devuelve el resto de la división de <i>numero1</i> entre <i>numero2</i>	<code>SELECT MOD(29.5) FROM DUAL;</code>	Devuelve 4
POWER(base, exponente)	Devuelve el número <i>base</i> elevado al número <i>exponente</i>	<code>SELECT POWER(2.4) FROM DUAL;</code>	Devuelve 16
ROUND(numero)	Devuelve el número redondeado	<code>SELECT ROUND(1.8949) FROM DUAL;</code>	Devuelve 1
ROUND(numero[,dec])	Devuelve el número redondeado usando tantos dígitos como el número <i>dec</i>	<code>SELECT ROUND(1.8949,3) FROM DUAL;</code>	Devuelve 1,895
TRUNC(numero)	Devuelve el número quitando sus decimales	<code>SELECT TRUNC(1.99939) FROM DUAL;</code>	Devuelve 1
TRUNC(numero[,dec])	Devuelve el número redondeado por defecto usando <i>dec</i> decimales	<code>SELECT TRUNC(1.99939,2) FROM DUAL;</code>	Devuelve 1,99
SIGN(numero)	Muestra el signo de <i>numero</i> , con 1 si es positivo, con -1 si es negativo y con 0 si es cero	<code>SELECT SIGN(-2.03) FROM DUAL;</code>	Devuelve -1
SQRT(numero)	Devuelve la raíz cuadrada de <i>numero</i>	<code>SELECT SQRT(81) FROM DUAL;</code>	Devuelve 9

Existen otras funciones numéricas llamadas de agregación, que usan los valores de una columna como un conjunto, devolviendo un único valor tras aplicar sus operaciones. Este valor puede ser parte de otra expresión que contenga funciones simples u operaciones aritméticas. Estas funciones se usan mucho en consultas de agrupamiento, ya que se puede realizar pequeños subgrupos atendiendo a valores repetidos de las diferentes columnas que participan en la consulta. Las funciones más usadas son AVG para la media aritmética, COUNT para contar el número de valores, MIN y MAX para el mínimo y máximo y SUM para sumar los valores de una expresión. A continuación, en la Tabla 5.2, se detalla, con ejemplos, junto a otras funciones de especial interés, el uso de estas funciones.

Función	Descripción	Ejemplos
AVG	Calcula la media aritmética de un conjunto de valores numéricos	<code>SELECT AVG(sueldo) FROM empleado;</code> <code>SELECT AVG(sueldo) FROM empleado WHERE sueldo>1000;</code>
COUNT	Cuenta el número de valores	<code>SELECT COUNT(*) FROM empleado WHERE sueldo>1900;</code>
MAX	Calcula el valor máximo de un conjunto de valores	<code>SELECT MAX(sueldo) FROM empleado;</code>
MIN	Calcula el valor mínimo de un conjunto de valores	<code>SELECT MIN(sueldo) FROM empleado;</code>
STDDEV	Calcula la desviación típica estándar de un conjunto de valores	<code>SELECT STDDEV(sueldo) FROM empleado;</code>
SUM	Suma un conjunto de valores	<code>SELECT SUM(prVenta-prCoste) FROM Producto;</code>
VARIANCE	Calcula la varianza de un conjunto de valores	<code>SELECT VARIANCE(sueldo) FROM empleado;</code>

PRÁCTICA RESUELTA

CONSULTAS CON FUNCIONES NUMÉRICAS

- Mostrar cuántos productos tienen que su precio de coste es menor que la mitad del precio de venta.
- Mostrar el precio más caro de los productos con IVA del 10 %.
- Mostrar el precio más barato de los productos.
- Contar cuantos productos de tipo portátil (tipo='P') tienen un precio de coste menor que la tercera parte del precio de venta.
- Mostrar la suma de la diferencia del precio de venta con respecto al de coste de todos los productos de tipo periférico (Tipo='F').
- Mostrar la media de la diferencia de los precios de venta con respecto al de coste de todos los productos cuyo tipo es Sobremesa (Tipo='S') o Portátil (Tipo='P').
- Mostrar los precios de coste y de venta con un solo decimal de aquellos productos cuyo tipo sea Componentes (Tipo='C').
- Mostrar los precios de coste redondeados con 1 decimal de aquellos productos cuyo precio de venta es solo un 20 % más que el de coste y su tipo es C, F o D.
- Mostrar el resto de la división del precio de venta con el de coste para aquellos productos con IVA del 10 % o del 21 %.
- Mostrar cuántas ventas se han realizado. Como cada fila que hay en la tabla Venta representa una venta, basta con contar el número de filas que tiene esa tabla.

Funciones de cadenas de caracteres

Un carácter es un simbolo que se trata como texto, ya sea una letra, un número o un carácter especial. Para almacenar un texto, los sistemas informáticos usan una tabla. Dicha tabla tiene un juego de caracteres y la más usada se denomina tabla ASCII (American Standard Code Interchange Information) o tabla de códigos estándar americano para el intercambio de información. Esta tabla está compuesta por 256 caracteres codificados, un número que ocupa un byte, es decir, un número comprendido entre 0 y 255. Para almacenar los caracteres, se usan su número. Cuando se representan en pantalla o en papel, se traduce dicho número por su carácter correspondiente según la tabla ASCII. Los caracteres que van desde 32 hasta 127 se llaman imprimibles y son iguales para todos los sistemas. Dependiendo de la lengua, la sección de la tabla que va desde la posición 128 hasta las 255 cambia según los caracteres que usa dicho juego de caracteres.

De este modo, el mismo número puede corresponder a diferentes símbolos, cuando estos son superiores a 127. Este problema aparece, por ejemplo, con la letra Ñ y los caracteres con tildes del idioma español. Estos pueden aparecer con otros simbolos dependiendo del juego de caracteres seleccionado.

El conjunto de caracteres imprimibles de la tabla ASCII pueden ser un carácter de tipo letra, un carácter de tipo número o un carácter de tipo simbolo especial. Los caracteres de tipo numérico se codifican con un número que está comprendido entre 48 y 57, las letras mayúsculas entre 65 y 90 y las minúsculas entre 97 y 122. Es importante tener en cuenta que, cuando se combinan letras y números en una cadena, los números en realidad son caracteres.

Caracteres ASCII imprimibles																														
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
!	"	#	\$	%	&	/	()	*	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>		
63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93
?	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[\]
94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124
^	_	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	}	

Una cadena, es decir, un conjunto de caracteres, se puede transformar a mayúsculas usando la función UPPER. Para ello, la función recorre cada uno de los caracteres con el fin de comprobar si estos se encuentran en el rango 97-122 de la tabla ASCII. Para cada coincidencia, lo único que tiene que hacer es cambiar dicho carácter por su equivalente en mayúsculas, que se obtiene restando 32 al número ASCII que ocupa el carácter en cuestión. De lo único que hay que preocuparse es de usar la función, invocándola y pasando como argumento la cadena que transformar. El siguiente ejemplo muestra todos los empleados cuya ciudad es 'MADRID'. Independientemente de cómo este almacenado en la base de datos, esta consulta convierte el contenido del campo Ciudad a su equivalente en mayúsculas comparándola con la cadena MADRID, escrita en su totalidad en mayúsculas.

```
SELECT nombre,prApellido, sgApellido FROM Empleado WHERE UPPER(Ciudad)='MADRID';
```

De este modo, si en la inserción se introdujo Madrid, madrid o MAdrid, la comparación funciona para todas ellas. Los valores en la tabla no cambian.

La conversión a minúsculas es exactamente igual. La función que se usa es LOWER y suma 32 a cada carácter que no está en el rango 97-122, obteniendo así una cadena compuesta en su totalidad con caracteres en minúsculas.

La función INITCAP devuelve la cadena con su primer carácter escrito en mayúsculas y el resto en minúsculas, lo que se conoce como formato Titulo. El siguiente es un ejemplo equivalente al anterior, pero esta vez se hace necesario compararlo con la cadena Madrid.

El problema que existe para usar esta técnica son los caracteres con tildes, ya que no hay una distancia de 32 posiciones entre la letra á y Á, ya que estos ocupan posiciones superiores a 127 de la tabla ASCII. Se puede diseñar una función de usuario que transforme una cadena a mayúsculas teniendo en cuenta la posición en la que se encuentra en la tabla ASCII.

Las funciones para manipular cadenas son muchas, y muy útiles si se decide evaluar los valores que se introducen y se manejan en las tablas. Por ejemplo, eliminando los espacios no útiles de las cadenas, los caracteres especiales que no deberían estar, controlar las cadenas en mayúsculas y minúsculas, la conversión entre cadenas de tipo numéricas y números, y viceversa, y un largo etcétera. Las funciones de tratamiento de cadenas más importantes son CHR, CONCAT, INITCAP, LOWER, LPAD, LTRIM, NCHR, NLS_INITCAP, NLS_LOWER, NLS_UPPER, NLSSORT, REGEXP_REPLACE, REGEXP_SUBSTR, REPLACE, RPAD, RTRIM, SUBSTR, TRANSLATE, TRANSLATE USING, TRIM y UPPER.

La conversión de un número a cadena también es importante, sobre todo, cuando se quieren usar funciones cuyos argumentos tienen que ser de tipo cadena. Las funciones más usadas relacionadas con este uso son ASCII, INSTR, LENGTH, REGEXP_COUNT, REGEXP_INSTR.

Para conocer el juego de caracteres activo, se usan las funciones NLS_CHARSET_DECL_LEN, NLS_CHARSET_ID y NLS_CHARSET_NAME.

El operador **LIKE** se usa para comparar una cadena de texto con una expresión regular. Si la cadena está dentro del rango de cadenas expresadas por la expresión regular, la evaluación de la cadena es cierta. Una expresión regular está formada por cualquier carácter imprimible y una serie de caracteres con significados especiales. Los más usados son % y _, donde % indica todas las cadenas posibles y _ todos los caracteres posibles. Por ejemplo, A% indica que la primera letra sea una A y el resto cualquier cosa, incluso la cadena vacía, es decir, solo la A. La expresión A%A indicaría que la primera letra y la última debe ser una A y en el medio, cualquier cadena. En la Tabla se muestran diferentes ejemplos usando % y _.

Expresión regular	Descripción	Cadenas válidas
%O	La cadena debe terminar por O mayúscula	OSO, camellO, oviedO, O
S%a	La cadena debe empezar por S mayúscula y acabar por a minúscula	Sevilla, Sosa, Salada, Segovia, Soria, Salamanca
E%	La cadena debe comenzar con la letra E mayúscula	España, E, Español, Elena, Eduardo, Elche
A__	La cadena tendrá tres caracteres y comenzará con la letra A	Ana, Año, Alá, Ant
S_%_A	La cadena comienza por S, termina por A y tiene como mínimo cuatro caracteres	Sevilla, Sosa, Salamandra, Suecia
%__s	La cadena tendrá terminará por s y tendrá como mínimo tres caracteres	Color123s, 1875s, rosas, canTOs

PRÁCTICA RESUELTA

CONSULTAS CON LIKE

- a) Mostrar el nombre comercial de los productos cuyo nombre interno contenga el simbolo +.
- b) Mostrar el código del producto de tipo P, que tengan en su nombre comercial la palabra Mac.
- c) Mostrar el DNI de los empleados cuyo sueldo es menor que 1200 o mayor que 1900 y su primer apellido empieza por A.
- d) Mostrar el DNI de aquellos clientes cuyo nombre contenga la letra a y su primer apellido empieza por la letra g o por la letra n y termine por la letra a o por la z.

Existe un conjunto de funciones muy útiles para manipular las cadenas y convertirlas según interese. Con la función **CONCAT(cadena1, cadena2)**, se obtiene una cadena resultado de la concatenación de las dos cadenas que se han indicado entre paréntesis. Otra función muy útil es **CHR(n)**, que permite obtener el carácter cuya posición en la tabla ASCII es n. La función **ASCII(car)** hace lo contrario, devolver la posición en la tabla ASCII que tiene el carácter que se le pasa entre paréntesis. Si es una cadena, devuelve el del primer carácter de dicha cadena. La función **LENGTH(cadena)** devuelve el número de caracteres que posee la cadena entre paréntesis.

Para eliminar espacios e incluso para comparar cadenas que pudieran tener por error más de un espacio donde no debieran, se puede usar las funciones **TRIM, RTRIM y LTRIM**. Estas funciones tienen en cuenta el carácter que es diferente al espacio, carácter 32 de la tabla ASCII. Los caracteres diferentes al espacio se denominan carácter no espacio.

La función TRIM elimina tanto los espacios en blanco a la izquierda del primer carácter no espacio de la cadena como los de la derecha del último carácter no espacio y también los dobles espacios.

La función LTRIM(cadena) devuelve una cadena que se obtiene de eliminar los espacios a la izquierda del primer carácter no espacio de la cadena que está entre paréntesis. La función RTRIM(cadena) hace el equivalente, pero con la parte derecha.

Estas tres funciones son aún más potentes si se usa su segundo parámetro con el formato **[R|L]TRIM(cadena,subcadena)**. Esta opción permite eliminar, en vez de espacios, la subcadena pasada como segundo parámetro.

Con las funciones SUBSTR, REPLACE y TRANSLATE se pueden transformar cadenas de una manera mucho más potente. Con **SUBSTR(cadena, pos1), longitud]**, se obtiene la subcadena de la cadena entre paréntesis que va desde la posición pos1 hasta el final, o cogiendo la longitud de caracteres a partir de pos1 cuando se ha indicado el argumento opcional longitud. Con **REPLACE(cadena, subcad, cadsust)**, se obtiene una nueva cadena formada de sustituir en la cadena la subcadena subcad por la cadena cadsust. Si no se indica la cadena de sustitución, simplemente, se elimina la subcadena de la cadena. Con **TRANSLATE(cadena1,cadena2, cadena3)**, se obtiene una nueva cadena donde las subcadenas que coincidan con cadena2 se sustituyen por la cadena3. Esta opción equivale al uso de los tres parámetros en REPLACE

PRÁCTICA RESUELTA

CONSULTAS CON CADENAS

- a) Mostrar el nombre en iniciales y apellidos en mayúsculas de aquellos empleados cuya dirección comienza con los caracteres C/.
- b) Mostrar los apellidos en mayúsculas y la dirección eliminando los espacios de la izquierda y de la derecha de aquellos empleados cuyos DNI terminan en O.
- c) Mostrar en mayúsculas el nombre y apellidos de los clientes cuya tercera letra en su nombre sea una r o una s y la tercera letra del segundo apellido sea una a sin tilde o una l. Usar la función SUBSTR. Ordenar la salida en orden al primer apellido.

Funciones de fechas

Las fechas no se almacenan como una cadena, sino que está compuesto por un conjunto de bytes donde se almacenan valores relacionados con las fechas en cuanto a su siglo, año, mes, día, hora, minuto, segundo, etcétera. Para manipular las fechas y tratarlas como cadenas, se tiene un conjunto de funciones que permiten manipular las fechas obteniendo resultados muy útiles. La fecha actual del sistema se obtiene con la función **SYSDATE** y no es necesario indicar ningún parámetro, por lo que no se usa paréntesis. Es una función muy útil y se usa en las inserciones/actualizaciones o para comprobar aspectos sobre las fechas.

Las funciones para el tratamiento de las fechas son, principalmente, ADD_MONTHS, CURRENT_DATE, CURRENT_TIMESTAMP, DBTIMEZONE, LAST_DAY, FROM_TZ, LOCALTIMESTAMP, MONTHS_BETWEEN, NEW_TIME, NEXT_DAY, NUMTODSINTERVAL, NUMTOYMINTERVAL, ROUND(fecha), SYSDATE, SYSTIMESTAMP, TO_CHAR(fecha), y TRUNC(fecha).

Funciones de fechas		
Función	Descripción	Ejemplos
ADD_MONTHS(fecha,m)	Devuelve la fecha incrementada en <i>m</i> meses	<code>SELECT ADD_MONTHS(SYSDATE, 1) FROM DUAL;</code>
LAST_DAY(fecha)	Devuelve la fecha del último día del mes de la fecha que se indica entre paréntesis	<code>SELECT LAST_DAY(SYSDATE) FROM DUAL;</code>
MONTHS_BETWEEN(fecha1, fecha2)	Devuelve cuántos meses hay entre <i>fecha1</i> y <i>fecha2</i>	<code>SELECT MONTHS_BETWEEN(ADD_MONTHS(SYSDATE,1), LAST_ DAY(SYSDATE)) FROM DUAL;</code>
NEXT_DAY(fecha, diasemana)	Devuelve la fecha del siguiente día de la semana indicado por <i>diasemana</i> y posterior a la <i>fecha</i> que hay entre paréntesis.	<code>SELECT NEXT_DAY(SYSDATE, 'Domingo') FROM DUAL;</code>

Para transformar cadenas a fecha, y viceversa, se usan las funciones TO_CHAR, TO_DATE y TO_NUMBER. Estas funciones tienen como segundo argumento el formato de fecha que se desea obtener. Este formato tiene una serie de caracteres compuestos por las letras cc o scc para el valor del siglo; y, yy, yyy o wyy para el valor del año; q para el número del trimestre; w o ww para el número de la semana del mes o del año respectivamente; mm para el número del mes; d, dd o ddd para el número del dia; h, hh o hh12 para las horas; mi para los minutos, y ss para los segundos.

Para transformar un valor que es de tipo DATE o NUMBER a una cadena de caracteres para su tratamiento como texto, se usa **TO_CHAR(valor, formato)**. En el formato, se puede combinar texto con estas letras, por ejemplo:

```
SELECT TO_CHAR(SYSDATE,"Hoy es el dia " dd " en el mes " month " del año " yyyy') FROM DUAL;
```

En la consulta anterior, month devuelve el texto del mes de la fecha. Para transformar una cadena a número o para formatear números según nos convenga se usa **TO_NUMBER(varlor,formato)**.

Los formatos serán:

9 Posiciones numéricas. Si el número que se quiere visualizar contiene menos dígitos de los que se especifican en el formato, se rellena con blancos.

0 Visualiza ceros por la izquierda hasta completar la longitud del formato especificado.

\$ Antepone el signo de dólar al número.

L Coloca en la posición donde se incluya, el símbolo de la moneda local (se puede configurar en la base de datos mediante el parámetro NSL_CURRENCY)

S Aparecerá el símbolo del signo.

D Posición del símbolo decimal.

G Posición del separador de grupo

Para transformar un tipo que es NUMBER o cadena a tipo DATE, se usa **TO_DATE(valor,formato)**.

Formatos:

YY	Año en formato de dos cifras
YYYY	Año en formato de cuatro cifras
MM	Mes en formato de dos cifras
MON	Las tres primeras letras del mes
MONTH	Nombre completo del mes
DY	Día de la semana en tres letras
DAY	Día completo de la semana
DD	Día en formato de dos cifras
D	Día de la semana del 1 al 7
Q	Semestre
WW	Semana del año
MI	Minutos de 0 a 59
AM	Indicador de a.m.
PM	Indicador de p.m.
SS	Segundos dentro del minuto
SSSS	Segundos desde las 0 horas
HH12	Hora de 1 a 12
HH24	Hora de 0 a 24

EXTRACT(parte from fecha): retorna la parte (especificada por el primer argumento) de una fecha. Puede extraer el año (year), mes (month), día (day), hora (hour), minuto (minute), segundo (second), etc.

Ejemplo: select extract(month from sysdate) from dual;

retorna el número de mes de la fecha actual.

Todas las operaciones con campos de tipo date se hacen en días.

Por ejemplo esta consulta devuelve la fecha del día de ayer.

select sysdate-1 from dual;

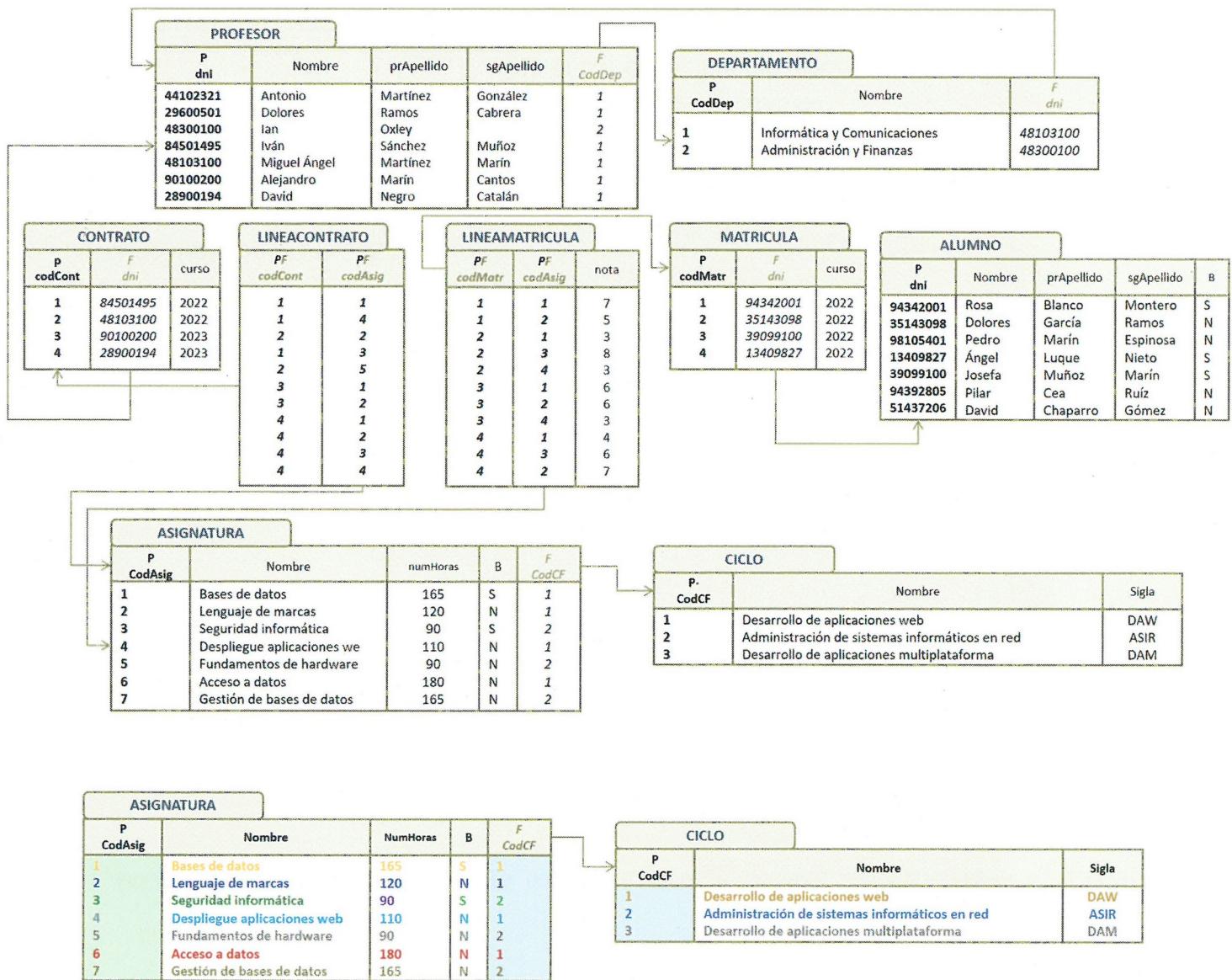
Y ésta devuelve el número de días que pasaron entre hoy y el 1 de enero de 2023

select sysdate-to_date('01/01/23','dd/mm/yy') from dual;

JOINS

La mayoría de las consultas requieren más de una tabla para su construcción. Si se indica más de una tabla en la cláusula FROM, aparece un nuevo problema. La operación resultante de una consulta en la que se incluyen dos tablas en la sección FROM es la combinación de cada uno de los registros de la primera tabla con cada uno de los registros de la segunda. Esta operación se denomina producto cartesiano.

Supóngase la base de datos de la Figura. Esta base de datos es de un centro de formación (**usuario academia**) donde se imparten asignaturas de diferentes ciclos de formación profesional. La tabla Ciclo almacena los ciclos de los que se imparte algún módulo y, en la tabla Asignatura, se almacena información sobre dichos módulos. El sentido de las flechas es el que hace referencia a cada una de las claves ajenas.



Ahora, se construye la siguiente consulta, donde se han indicado dos tablas en el FROM:

SELECT * FROM Asignatura, Ciclo;

Si se ejecuta esta consulta, se comprobará que muestra 18 registros, resultado de combinar cada registro de la tabla asignatura con cada uno de los registros de la tabla Ciclo. Esto es lo que ocurre cuando se indica más de una tabla en la cláusula FROM.

ASIGNATURA x CICLO							
P CodAsig	Nombre	numHoras	B	F CodCF	P CodCF	Nombre	Sigla
1	Bases de datos	165	S	1	1	Desarrollo de aplicaciones web	DAW
1	Bases de datos	165	S	1	2	Administración de sistemas informáticos en red	ASIR
1	Bases de datos	165	S	1	3	Desarrollo de aplicaciones multiplataforma	DAM
2	Lenguaje de marcas	120	N	1	1	Desarrollo de aplicaciones web	DAW
2	Lenguaje de marcas	120	N	1	2	Administración de sistemas informáticos en red	ASIR
2	Lenguaje de marcas	120	N	1	3	Desarrollo de aplicaciones multiplataforma	DAM
3	Seguridad informática	90	S	2	1	Desarrollo de aplicaciones web	DAW
3	Seguridad informática	90	S	2	2	Administración de sistemas informáticos en red	ASIR
3	Seguridad informática	90	S	2	3	Desarrollo de aplicaciones multiplataforma	DAM
4	Despliegue aplicaciones web	110	N	1	1	Desarrollo de aplicaciones web	DAW
4	Despliegue aplicaciones web	110	N	1	2	Administración de sistemas informáticos en red	ASIR
4	Despliegue aplicaciones web	110	N	1	3	Desarrollo de aplicaciones multiplataforma	DAM
5	Fundamentos de hardware	90	N	2	1	Desarrollo de aplicaciones web	DAW
5	Fundamentos de hardware	90	N	2	2	Administración de sistemas informáticos en red	ASIR
5	Fundamentos de hardware	90	N	2	3	Desarrollo de aplicaciones multiplataforma	DAM
6	Acceso a datos	180	N	1	1	Desarrollo de aplicaciones web	DAW
6	Acceso a datos	180	N	1	2	Administración de sistemas informáticos en red	ASIR
6	Acceso a datos	180	N	1	3	Desarrollo de aplicaciones multiplataforma	DAM
7	Gestión de bases de datos	165	N	2	1	Desarrollo de aplicaciones web	DAW
7	Gestión de bases de datos	165	N	2	2	Administración de sistemas informáticos en red	ASIR
7	Gestión de bases de datos	165	N	2	3	Desarrollo de aplicaciones multiplataforma	DAM

Se puede comprobar que el producto cartesiano de dos tablas proyecta las columnas de la primera tablas unidas con las columnas de la segunda tabla y existen dos columnas que se relacionan, CodCF de Asignatura y codCF de Ciclo, que también se podrían escribir con la notación Asignatura.codCF y Ciclo.codCF.

Si se analiza el resultado, cada asignatura se ha combinado con todos los ciclos posibles, obteniendo todas las posibilidades de combinación. Se combinan Asignatura y Ciclo porque se desea obtener el ciclo al que pertenece cada asignatura. Ahora bien, esta combinación relaciona todos los registros de una con todos los de otra. ¿Cuáles serían los registros que interesa obtener? Obvio, los campos claves de Ciclo que coinciden con las claves ajenas en Asignatura. Para ello, lo único que hay que hacer es obligar a que solo se seleccionen aquellos registros que sus CodCF sean iguales. Para acceder al campo de una tabla, se puede hacer por el nombre de la columna o indicando, además, de qué tabla se trata. Así, para indicar la columna CodCF de Asignatura, se debe escribir Asignatura.CodCF. Cuando en la combinación se tienen campos que se llaman iguales, se hace necesario indicar a qué tabla hace referencia dicho campo, de lo contrario, existe ambigüedad.

La consulta anterior hubiera quedado como sigue:

```
SELECT Asignatura.nombre, numHoras, Ciclo.nombre, sigla
FROM Asignatura, Ciclo
WHERE Asignatura.CodCF=Ciclo.CodCF;
```

Primero, se hace el producto cartesiano y, luego, se rechazan todos los registros que no cumplen la condición del WHERE, es decir, solo selecciona aquellos registros que los campos Asignatura.CodCF y Ciclo.CodCF son iguales. En la Figura 5.10, se muestra el filtrado de cada una de las filas que no coinciden el campo común que las relaciona, es decir, codCF.

ASIGNATURA x CICLO							Se seleccionan los valores que coinciden
P CodAsig	Nombre	numHoras	B	F codCF	P CodCF	Nombre	Sigla
1	Bases de datos	165	S	1	1	Desarrollo de aplicaciones web	DAW
1	Bases de datos	165	S	1	2	Administración de sistemas informáticos en red	ASIR
1	Bases de datos	165	S	1	3	Desarrollo de aplicaciones multiplataforma	DAM
2	Lenguaje de marcas	120	N	1	1	Desarrollo de aplicaciones web	DAW
2	Lenguaje de marcas	120	N	1	2	Administración de sistemas informáticos en red	ASIR
2	Lenguaje de marcas	120	N	1	3	Desarrollo de aplicaciones multiplataforma	DAM
3	Seguridad informática	90	S	2	1	Desarrollo de aplicaciones web	DAW
3	Seguridad informática	90	S	2	2	Administración de sistemas informáticos en red	ASIR
3	Seguridad informática	90	S	2	3	Desarrollo de aplicaciones multiplataforma	DAM
4	Despliegue aplicaciones web	110	N	1	1	Desarrollo de aplicaciones web	DAW
4	Despliegue aplicaciones web	110	N	1	2	Administración de sistemas informáticos en red	ASIR
4	Despliegue aplicaciones web	110	N	1	3	Desarrollo de aplicaciones multiplataforma	DAM
5	Fundamentos de hardware	90	N	2	1	Desarrollo de aplicaciones web	DAW
5	Fundamentos de hardware	90	N	2	2	Administración de sistemas informáticos en red	ASIR
5	Fundamentos de hardware	90	N	2	3	Desarrollo de aplicaciones multiplataforma	DAM
6	Acceso a datos	180	N	1	1	Desarrollo de aplicaciones web	DAW
6	Acceso a datos	180	N	1	2	Administración de sistemas informáticos en red	ASIR
6	Acceso a datos	180	N	1	3	Desarrollo de aplicaciones multiplataforma	DAM
7	Gestión de bases de datos	165	N	2	1	Desarrollo de aplicaciones web	DAW
7	Gestión de bases de datos	165	N	2	2	Administración de sistemas informáticos en red	ASIR
7	Gestión de bases de datos	165	N	2	3	Desarrollo de aplicaciones multiplataforma	DAM

El resultado, después de eliminar todos los registros que no cumplen la condición **Asignatura.CodCF = Ciclo.CodCF** será

ASIGNATURA x CICLO							
P CodAsig	Nombre	numHoras	B	F codCF	P CodCF	Nombre	Sigla
1	Bases de datos	165	S	1	1	Desarrollo de aplicaciones web	DAW
2	Lenguaje de marcas	120	N	1	1	Desarrollo de aplicaciones web	DAW
3	Seguridad informática	90	S	2	2	Administración de sistemas informáticos en red	ASIR
4	Despliegue aplicaciones web	110	N	1	1	Desarrollo de aplicaciones web	DAW
5	Fundamentos de hardware	90	N	2	2	Administración de sistemas informáticos en red	ASIR
6	Acceso a datos	180	N	1	1	Desarrollo de aplicaciones web	DAW
7	Gestión de bases de datos	165	N	2	2	Administración de sistemas informáticos en red	ASIR

Si se analiza la tabla resultante, lo que se ha obtenido es cada una de las asignaturas con el ciclo al que pertenece. Es decir, se ha relacionado la tabla Asignatura con sus claves ajenas en la tabla Ciclo. A continuación, la consulta solo muestra las columnas seleccionadas en la cláusula SELECT. El resultado final es

Nombre	numHoras	Nombre	Sigla
Bases de datos	165	Desarrollo de aplicaciones web	DAW
Lenguaje de marcas	120	Desarrollo de aplicaciones web	DAW
Seguridad informática	90	Administración de sistemas informáticos en red	ASIR
Despliegue aplicaciones web	110	Desarrollo de aplicaciones web	DAW
Fundamentos de hardware	90	Administración de sistemas informáticos en red	ASIR
Acceso a datos	180	Desarrollo de aplicaciones web	DAW
Gestión de bases de datos	165	Administración de sistemas informáticos en red	ASIR

Cuando se combinan dos tablas, seguramente ambas comparten un campo común, y es este el que se debe igualar en la cláusula WHERE del modo tabla1.campoComun=tabla2.campoComun.

Todas las relaciones 1:1 y 1:N generan columnas con claves ajenas que hacen referencia a la tabla correspondiente. Cuando se quieren unir, se usa el producto cartesiano.

Todas las relaciones N:M generan una nueva tabla. Entonces, cuando se quieren relacionar las claves ajenas con sus correspondientes claves primarias, se hace necesario utilizar tres tablas. Estas tres tablas irán en la cláusula FROM y en el WHERE las condiciones de igualdad.

La tabla LineaMatricula une la tabla Matricula, que almacena la información de una matricula de un alumno, con la tabla Asignatura para almacenar las asignaturas de dicha matricula. Para obtener información relacionada, se requiere la asociación de las tres tablas: Matricula, LineaMatricula y Asignatura. La combinación correcta de los registros de estas tres tablas da como resultado la información de las asignaturas en las que se ha matriculado un alumno en particular. Ahora, el producto cartesiano es más complejo, combina cada combinación con cada registro de la tercera tabla. Si la tabla Asignatura tiene 6 registros, la tabla LineaMatricula 11 y la tabla Matricula 4, la consulta siguiente devuelve 264 registros, es decir, la combinación de cada asignatura con cada registro de lineaamatricula y con cada registro de la tabla Matricula. La consulta que mostraria el producto cartesiano de tres tablas es la siguiente:

```
SELECT * FROM Matricula, LineaMatricula, Asignatura;
```

cualquier operación sobre las tablas Matricula, LineaMatricula y Asignatura, se deben cruzar el campo que relaciona Matricula y LineaMatricula, es decir, Matricula.codMatr=LineaMatricula.codMatr, y el campo que relaciona LineaMatricula con Asignatura, es decir, LineaMatricula.codAsig=Asignatura.codAsig. De esta forma, se obtiene el producto cartesiano con el filtro para eliminar los registros no vinculantes del producto cartesiano.

La consulta resultante seria la siguiente:

```
SELECT * FROM Matricula, LineaMatricula, Asignatura  
WHERE Matricula.codMatr=LineaMatricula.codMatr AND LineaMatricula.codAsig=Asignatura.codAsig;
```

De la consulta anterior, si se quisiera seleccionar el nombre y apellidos del alumno cuyo DNI es el que está en la tabla Matricula, se deberia añadir la tabla Alumno. El cartesiano es ahora de cuatro tablas y la consulta seria:

```
SELECT * FROM Matricula, LineaMatricula, Asignatura, Alumno  
WHERE Matricula.codMatr=LineaMatricula.codMatr AND  
LineaMatricula.codAsig=Asignatura.codAsig AND Matricula.dni=Alumno.dni;
```

Cuando se hace el producto cartesiano de muchas tablas, interesa usar lo que se conoce como alias de tablas. Es una forma de nombrar para dicha consulta las tablas que en ella

intervienen. No es un cambio de nombre de la tabla, sino que, para dicha consulta, la tabla se llama con ese otro nombre. La consulta anterior usando alias podría ser la siguiente:

```
SELECT * FROM Matricula M, LineaMatricula L, Asignatura Ag, Alumno Al  
WHERE M.CodMatr=L.CodMatr AND L.codAsig=Ag.codAsig AND M.dni=Al.dni;
```

PRÁCTICA RESUELTA

CONSULTAS CON JOINS

- a) Mostrar el DNI de los profesores que pertenecen al departamento de Informática y Comunicaciones.
- b) Mostrar el DNI de los alumnos bilingües que se han matriculado en el año 2029.
- c) Mostrar el nombre y el primer apellido de los alumnos bilingües que se han obtenido una calificación superior a 7.
- d) Mostrar el nombre y los apellidos de los alumnos que se encuentran matriculados en las asignaturas Gestión de Base de Datos y Acceso a Datos.
- e) Mostrar el nombre y apellidos de los alumnos que han aprobado asignaturas del ciclo con sigla DAM.

JOIN ON, JOIN USING

El operador JOIN se usa para separar las condiciones de filtrado con las condiciones para el producto cartesiano. En la siguiente sentencia, se puede comprobar cuáles son las condiciones de selección de registros y cuáles de asociación:

```
SELECT Al.Nombre,prApellido, sgApellido FROM Alumno Al,Matricula M, LineaMatricula L, Asignatura Ag, Ciclo C WHERE Al.dni=M.dni AND M.codMatr=L.codMatr AND L.CodAsig=Ag.CodAsig AND Sigla='DAM' AND nota>=5 AND Ag.CodCF=C.CodCF;
```

Aunque toda la expresión lógica de la cláusula WHERE es una condición de selección de registros, se pueden distinguir, por una parte, las condiciones de asociación y, por otra, Sigla='DAM' AND nota>=5.

El operador JOIN se usa en la cláusula FROM y tiene la sintaxis siguiente

Tabla1 [LEFT | RIGHT | FULL OUTER] JOIN tabla2 USING(columnas) o también

Tabla1 [LEFT | RIGHT | FULL OUTER] JOIN tabla2 ON condicion.

El objetivo es fácil: combinar las tablas que se indican en el FROM a través de JOIN. Se pueden añadir tantas como tablas participen en la asociación. Si se usa la opción USING, entre paréntesis, se indican las columnas que se deben cruzar (que deben llamarse igual en las dos tablas) y, si se usa la opción ON, la condición de asociación. También se debe tener en cuenta si existe ambigüedad con algún campo, ya que puede existir el mismo nombre para diferentes tablas.

Por ejemplo, la consulta:

```
SELECT Asignatura.nombre, numHoras, Ciclo.nombre, sigla  
FROM Asignatura, Ciclo  
WHERE Asignatura.CodCF=Ciclo.CodCF;
```

queda, usando el operador JOIN USING, de la manera:

```
SELECT Asignatura.nombre, numHoras, Ciclo.nombre, sigla  
FROM Asignatura JOIN Ciclo USING (CodCF);
```

y, usando ON, es:

```
SELECT Asignatura.nombre, numHoras, Ciclo.nombre, sigla  
FROM Asignatura JOIN Ciclo ON Asignatura.CodCF=Ciclo.CodCF;
```

Cuando se tiene más de dos tablas, simplemente, se separa con JOIN. La consulta:

```
SELECT Asignatura.nombre, Alumno.nombre, prApellido, sgApellido  
FROM Alumno, Matricula, LineaMatricula, Asignatura  
WHERE Alumno.dni=Matricula.dni  
AND Matricula.codMatr=LineaMatricula.codMatr  
AND Asignatura.codAsig=LineaMatricula.codAsig  
AND Asignatura.Nombre IN ('Gestión de bases de datos', 'Acceso a datos')  
AND nota>7
```

Sería usando JOIN USING de la siguiente forma:

```
SELECT Asignatura.nombre, Alumno.nombre, prApellido, sgApellido  
FROM Alumno JOIN Matricula USING(dni)  
JOIN LineaMatricula USING(codMatr)  
JOIN Asignatura USING(codAsig)  
WHERE Asignatura.Nombre IN ('Gestión de bases de datos', 'Acceso a datos') AND nota>7;
```

y si se usa JOIN ON

```
SELECT Asignatura.nombre, Alumno.nombre, prApellido, sgApellido  
FROM Alumno JOIN Matricula ON Alumno.dni=Matricula.dni  
JOIN LineaMatricula ON Matricula.codMatr=LineaMatricula.codMatr  
JOIN Asignatura ON LineaMatricula.codAsig=Asignatura.codAsig  
WHERE Asignatura.Nombre IN ('Gestión de bases de datos', 'Acceso a datos') AND nota>7;
```

NATURAL JOIN

Cuando entre dos tablas solo existe un campo que se llama igual en ambas tablas, siendo uno clave primaria y el otro ajena, se puede hacer la operación producto natural. El producto natural es el producto cartesiano y la condición de asociación a la vez. En la escritura de la sentencia, no se declara la condición de asociación. La siguiente consulta no es válida, ya que hay más de un campo que se llama igual, CodCFy nombre.

```
SELECT * FROM Ciclo NATURAL JOIN Asignatura;
```

Se podría cambiar el campo nombre de la tabla asignatura y comprobar que sí funciona el producto natural. Para ello, se ejecutan los siguientes dos comandos:

```
ALTER TABLE Asignatura RENAME COLUMN Nombre TO NombreAsig;
```

```
SELECT * FROM Ciclo NATURAL JOIN Asignatura;
```

OUTER JOIN

El operador JOIN, por defecto, solo cruza los valores iguales entre claves ajenas y claves primarias. Esta operación se denomina asociación interna. La opción para indicar que una operación JOIN es interna es a través de INNER. Pues bien, una asociación puede ser externa. En las combinaciones externas, se combinan todos los registros de una columna, aunque no estén relacionados. Para indicar cuál de las dos columnas se expande, se usa LEFT para la primera tabla, RIGHT para la segunda y FULL para ambas columnas. La siguiente consulta muestra todas las asignaturas para cada Ciclo, tenga o no asignaturas asociadas.

```
SELECT * FROM Ciclo LEFT OUTER JOIN Asignatura USING(CodCF);
```

Si se quiere expandir la tabla de la derecha, que en el ejemplo anterior es Asignatura, se indica RIGHT en vez de LEFT. La consulta quedaría como sigue:

```
SELECT * FROM Ciclo RIGHT OUTER JOIN Asignatura USING(CodCF);
```

También se puede indicar que se expandan ambas tablas. Para ello, se usa FULL OUTER como indica el siguiente ejemplo:

```
SELECT * FROM Ciclo FULL OUTER JOIN Asignatura USING(CodCF);
```

Podemos usar join... using o join... on

PRÁCTICA RESUELTA

CONSULTAS CON JOIN ON Y USING

- Mostrar el DNI de los profesores que pertenecen al departamento de informática y comunicaciones cuyo DNI es mayor que 44200200.
- Mostrar el DNI de los profesores que pertenecen al departamento de administración y finanzas cuyo primer apellido empieza por P y el segundo apellido termina por z y fueron contratados en el curso 2022.
- Mostrar el nombre y el primer apellido de los profesores que han impartido la asignatura Acceso a datos.
- Mostrar el nombre y los apellidos de los alumnos que se encuentran matriculados en la asignatura de Gestión de Base de Datos y de Acceso a Datos y que han aprobado la asignatura.
- Mostrar el nombre y apellidos de los alumnos que han aprobado asignaturas del ciclo con sigla ASIR, cuyo segundo apellido tiene una z. Mostrar los registros ordenados por el primer apellido y luego por el segundo

GROUP BY

En la agrupación por columnas, lo que se intenta es crear una única fila por cada valor distinto de las columnas que forman el grupo. Una vez agrupados los registros, se usan funciones de grupos de valores según lo que se pretenda acometer. Estas funciones son AVG, COUNT, MAX, MIN y SUM, entre otras.

La cláusula para agrupar por diferentes valores atendiendo a ciertas columnas es GROUP BY. La sintaxis de esta cláusula es GROUP BY columna1[,columna2...]. Cuando se usa junto a una cláusula WHERE, la cláusula de agrupación siempre se escribe después. Por ejemplo, la siguiente consulta trata de agrupar y contar las asignaturas por ciclos:

```
SELECT COUNT(*) FROM Asignatura group by CodCF;
```

Las columnas que se pueden seleccionar sólo son las que intervienen en la agrupación. Así, la única columna que se podría seleccionar sería CodCF. Por tanto, la siguiente consulta si es válida:

```
SELECT CodCF, COUNT(*) FROM Asignatura GROUP BY CodCF;
```

Pero esta no lo sería, puesto que nombre no participa en el GROUP BY.

```
SELECT Nombre, COUNT(*) FROM Asignatura GROUP BY CodCF;
```

Funciones de Agregación

Las funciones de agregación o agrupación permiten realizar operaciones por cada conjunto agrupado con la orden GROUP BY. Cada uno de esos cálculos que se hacen con un conjunto de valores agrupados se denomina cálculo resumen. Generalmente, se usan en la cláusula SELECT.

Las funciones más usadas son AVG, COUNT, MAX, MIN, SUM, STDEV y VAR.

La función AVG permite calcular la media aritmética. Por ejemplo, la siguiente consulta calcula la media de las calificaciones por cada asignatura:

```
SELECT CodASig, AVG(notas) FROM LineaMatricula GROUP BY CodASig;
```

La función COUNT permite contar el número de filas por cada agrupación. Generalmente, no es importante indicar la columna con la que contar, por lo que se usa COUNT(*). Cuando se quieren contar las filas que tienen valor en una columna, esta se indica entre paréntesis, por ejemplo, COUNT(dni). Se puede usar el predicado DISTINCT para no contar los valores que se repiten de dicha columna COUNT(DISTINCT dni). El siguiente ejemplo cuenta cuántos alumnos se han matriculado por asignatura en el año 2022:

```
SELECT CodASig, COUNT(*) FROM Matricula WHERE curso=2922 GROUP BY CodASig;
```

Para calcular el número mayor de un conjunto de valores se usa la función MAX, indicando entre paréntesis la columna que formará dicho conjunto. Esta función también se puede usar con cadenas, sabiendo que están ordenados según la tabla de caracteres ASCII. Para calcular el valor mínimo, se usa MIN con la misma operatividad que MAX.

Otra función muy útil es SUM. Esta función suma los valores de una columna y también se puede usar para sumar cada uno de los grupos de valores agrupados por la columna o

columnas que se han indicado en la cláusula GROUP BY. La siguiente consulta suma la cantidad de productos comprados por cada tipo:

```
SELECT codProducto, SUM(cantidad) FROM LineaCompra NATURAL JOIN Producto  
GROUP BY CodProducto;
```

El valor que devuelve una función puede ser el valor de entrada de otra, de tal forma que se obtienen expresiones con funciones anidadas. Así, se pueden tener expresiones del estilo `AVG(SUM(columna))` con el fin de calcular la media de las sumas de los valores numéricos agrupados.

HAVING

La cláusula HAVING permite filtrar los resultados de una agrupación. La cláusula WHERE se aplica a cada registro y la cláusula HAVING se aplica a cada fila agrupada. Es importante advertir que esta cláusula se declara después de WHERE, así, una vez evaluado cada registro, se agrupa según la columna o las columnas que se indican en GROUP BY y, a continuación, se evalúa para cada fila agrupada la condición definida en la cláusula HAVING. En el uso del having hay que tener las siguientes consideraciones:

Solo se usa HAVING si se ha usado GROUP BY, es decir, si no hay GROUP BY, no hay HAVING.

El uso de WHERE es totalmente independiente con HAVING, puede existir WHERE con HAVING o no, ya que la cláusula WHERE afecta a cada una de las filas antes de agrupar y HAVING, a cada una de las filas después de agrupar.

En la condición del HAVING, solo pueden intervenir los campos seleccionados en el SELECT, ya que esta condición se efectúa una vez que se obtiene la agrupación de las filas y, en esta agrupación, solo existen las columnas del SELECT.

Supóngase que se parte de la base de datos correspondiente a la base de datos de un centro de formación.

Se quiere mostrar el número de profesores que trabajan para cada departamento. En esta consulta, se hace necesario agrupar en la tabla Profesor aquellos profesores que tengan el mismo departamento, es decir, el mismo código de departamento. Para ello, se usa la cláusula GROUP BY CodDep. Una vez que las filas han sido agrupadas por cada CodDep, se procede a contar el número de filas con la función COUNT(*). Como cada fila de la tabla profesor representa un único profesor, no se hace necesario usar DISTINCT.

```
Select codDep, count(*) from profesor group by codDep;
```

PROFESOR				
P dni	Nombre	prApellido	sgApellido	F CodDep
44102321	Antonio	Martínez	González	1
29600501	Dolores	Ramos	Cabrera	1
84501495	Iván	Sánchez	Muñoz	1
48103100	Miguel Ángel	Martínez	Marín	1
90100200	Alejandro	Marín	Cantos	1
28900194	David	Negro	Catalán	1
48300100	Ian	Oxley		2

Diagrama explicativo:

- Los primeros 5 profesores pertenecen al Grupo 1, porque tienen el mismo valor en la columna CodDep (1).
- El profesor Ian pertenece al Grupo 2, porque tiene el valor 2 en la columna CodDep.
- El resultado de la consulta es:
 - Para el Grupo 1 (CodDep=1): COUNT(*)=6
 - Para el Grupo 2 (CodDep=2): COUNT(*)=1

Supóngase ahora que se quiere mostrar para cada asignatura el número de alumnos matriculados. Como lo que se quiere obtener son los alumnos matriculados, las tablas con las que se requiere trabajar son Matricula y LineaMatricula. Se obtiene el producto natural de estas dos tablas por el campo codMatr y se agrupa por el campo codAsig. Solo queda la operación de agrupación que, en este caso, es contar cada fila, es decir, COUNT(*) . La consulta queda como sigue:

```
SELECT CodAsig, COUNT(*) FROM Matricula NATURAL JOIN LineaMatricula GROUP BY CodAsig,
```

Supóngase ahora que se quiere mostrar para cada asignatura y curso el número de alumnos matriculados.

Esta consulta es parecida a la anterior, con la salvedad de que ahora se hace necesario agrupar, además por cada curso. Por tanto, una vez que se agrupa por el código de la asignatura, luego se agrupa por cada curso.

La cláusula de agrupación es CodAsig, Curso. La consulta queda como sigue:

```
SELECT CodAsig, Curso, COUNT(*) FROM Matricula NATURAL JOIN LineaMatricula  
GROUP BY CodAsig, Curso;
```

Ahora, supóngase que se quiere mostrar la media de cada alumno. La información que hay que consultar se encuentra en la tabla LineaMatricula, con el fin de calcular la media de la columna Nota. La agrupación se hace por el campo dni, ya que interesa agrupar por cada alumno. Se necesita entonces las tablas Matricula y LineaMatricula y su producto natural. La consulta queda:

```
SELECT dni, AVG(Nota) FROM Matricula NATURAL JOIN LineaMatricula GROUP BY dni;
```

Ahora, se procede a hacerla misma consulta, pero añadiendo la media de cada alumno por cada curso. Lo único que hay que hacer con respecto a la consulta anterior es agrupar en un segundo nivel por el campo curso. Una vez que se ha agrupado por cada alumno usando el campo dni, luego se agrupa usando el campo Curso. La consulta queda como sigue:

```
SELECT dni, curso, AVG(Nota) FROM Matricula NATURAL JOIN LineaMatricula GROUP  
BY dni, curso;
```

La siguiente consulta pretende mostrar para los alumnos bilingües su nota más alta. El campo que se debe usar de la tabla Alumno es B con el fin de filtrar aquellos que lo son, B='S'. Para calcular la nota más alta, se usa la función de agrupación MAX(Nota) en la tabla LineaMatricula. El campo que se utilizará para agrupar es dni, que se encuentra en la tabla Matricula. Por tanto, para esta consulta, se requieren las tablas Alumno, Matricula y LineaMatricula. La siguiente consulta se implementa usando JOIN USING:

```
SELECT dni, MAX(Nota) FROM Alumno JOIN Matricula USING(dni) JOIN LineaMatricula  
USING(CodMatr) WHERE B='S' GROUP BY dni;
```

La siguiente consulta quiere mostrar la nota más alta en cada curso de aquellas asignaturas del ciclo con sigla ASIR. En esta consulta, interesa agrupar por cada curso de la tabla Matricula. Para filtrar por la sigla ASIR, se debe partir de la tabla Ciclo, uniéndola luego con Asignatura a través de campo común CodCF. Una vez que se han obtenido las asignaturas del ciclo ASIR, se hará la unión con la tabla LineaMatricula usando el campo

común CodASig. Además, en esta tabla, está la calificación obtenida por cada alumno en cada curso en cada asignatura. La consulta usando JOIN ON queda como sigue:

```
SELECT curso, MAX(Nota) FROM Ciclo C JOIN Asignatura ON C.codCF=Ag.codCF  
JOIN LineaMatricula ON Ag.codAsig=L.codASig WHERE Sigla='ASIR' GROUP BY curso;
```

En la siguiente consulta, se quiere mostrar la media de aquellos alumnos cuyo primer apellido empieza por B o por C, que cursa asignaturas cuyo profesor que las imparte pertenece al departamento de administración y finanzas.

Para mostrar la media en la cláusula SELECT, se añade la función AVG(Nota). A continuación, se quieren filtrar aquellos alumnos cuyo primer apellido empieza por B o por C. Esta información se encuentra en Alumno y se debe unir con la tabla Matricula, que es la que se está usando para agrupar, es decir Alumno JOIN Matricula on Alumno.dni=Matricula.dni

Ahora, para filtrar los profesores que pertenecen al departamento de administración y finanza, se parte de la tabla Departamento, uniéndola con Profesor usando Departamento D JOIN Profesor P USING (codDep) y luego se une con la tabla contrato con JOIN Contrato ON profesor.dni=Contrato.dni

Ahora, solo queda combinar las asignaturas del contrato usando LineaContrato y las asignaturas de las matrículas usando LineaMatricula empleando Matricula JOIN LineaMatricula USING(CodMatr) y Contrato JOIN LineaContrato USING (CodCont). Por último, para que se trate de la misma asignatura, se añade LineaContrato JOIN LineaMatricula USING (codAsig).

La consulta final se agrupa por alumnos quedando como sigue:

```
SELECT dni, AVG(Nota)  
FROM Alumno JOIN Matricula ON Alumno.dni=Matricula.dni  
JOIN LineaMatricula USING(codMatr)  
JOIN LineaContrato USING (codAsig)  
JOIN Contrato USING(CodCont)  
JOIN Profesor ON Profesor.dni=Contrato.dni  
JOIN Departamento ON Profesor.codDep=Departamento.codDep  
WHERE Departamento.Nombre='Administración y Finanzas'  
AND (prApellido LIKE 'B%' OR prApellido LIKE 'C%')  
GROUP BY Alumno.dni;
```

El uso de ON está justificado porque existe más de un campo dni de diferentes tablas y en la misma consulta, el del profesor y el del alumno. Para anular la ambigüedad, se usa mejor la palabra ON en vez de USING.

A continuación, se pretende hacer uso de la cláusula HAVING. Para ello, supóngase que se desea mostrar el DNI de los profesores que están impartiendo cuatro o más asignaturas en el curso actual. Para conocer el curso actual, se obtiene el año con cuatro dígitos de SYSDATE y se compara con el campo numérico curso. Para ello, se usaría Curso=TO_NUMBER(TO_CHAR(SYSDATE, 'YYYY')). Para contar el número de asignaturas que imparte un profesor, se usa la tabla LineaContrato, haciendo la operación

`COUNT(codAsig)>=4.` Esta condición es la que se usa en la cláusula HAVING. Se hace el producto natural por el campo común CodCont de las tablas Contrato y LineaContrato. La consulta queda como sigue:

```
SELECT dni, COUNT(CodAsig) FROM Contrato NATURAL JOIN LineaContrato
WHERE Curso=TO_NUMBER(TO_CHAR(SYSDATE,'YYYY'))
GROUP BY dni
HAVING COUNT(CodAsig)>=4;
```

continuación, se quiere confeccionar una consulta para mostrar el DNI de los profesores que están impartiendo más asignaturas que la media en el curso actual. Para ello, se cuenta las asignaturas que imparte cada profesor y, luego, se filtra usando la cláusula HAVING. En esta cláusula, será necesario volver a contar las asignaturas por profesor y calcular la media. El resultado es el siguiente:

```
SELECT dni, COUNT(CodAsig) FROM Contrato NATURAL JOIN LineaContrato
WHERE Curso=TO_NUMBER(TO_CHAR(SYSDATE,'YYYY'))
GROUP BY dni
HAVING COUNT(CodAsig)>=
    (SELECT AVG(COUNT(CodAsig)) FROM Contrato NATURAL JOIN LineaContrato
     WHERE Curso=TO_NUMBER(TO_CHAR(SYSDATE,'YYYY'))
     GROUP BY dni);
```

Por último, se quiere mostrar el DNI de los alumnos que tienen más de un 9 como media de las asignaturas en las que se ha matriculado. La consulta requiere de la cláusula HAVING como se muestra a continuación:

```
Select dni, AVG(notas) FROM Matricula
GROUP BY dni
HAVING AVG(Notas)>9;
```

SUBCONSULTAS

Una subconsulta está compuesta de una consulta principal y otra subordinada a esta. La consulta segunda devuelve un conjunto de valores a la principal haciendo uso de sus filtros con el fin de seleccionar solo aquellos registros que son de interés. La lista de valores que devuelve la consulta interior se convierte en la lista de valores de entrada de la consulta principal, que se encuentra en un nivel superior y por ello se llama subconsulta. El objetivo final es minimizar el número de combinaciones posibles, ya que, antes de combinar, se filtran los registros de interés. Una vez filtrado, se devuelve a la consulta superior los registros de interés, generalmente los campos identificativos. De este modo, se reduce enormemente el coste del proceso del producto cartesiano. Si en una consulta se filtra en su cláusula WHERE aquellos registros innecesarios de una tabla, la consulta que está por encima procesa menos registros y no realiza el producto cartesiano con toda la tabla, minimizando el número de evaluaciones.

La estructura puede tener una consulta anidada con otra a través de unos operadores, y pueden existir más de dos consultas anidadas. Se puede resumir esta estructura con el siguiente esquema:

```
SELECT columnas FROM tablas  
WHERE condiciones CONECTOR  
(SELECT columnas FROM tablas WHERE condiciones...)
```

Los conectores más usados son <, >, =, <=, >=, IN y NOT IN

IN

El operador IN busca si la expresión de la izquierda se encuentra en la lista de la derecha. Por ejemplo, la siguiente expresión comprueba si el DNI está en la lista de valores que se indican entre paréntesis. Si está devuelve el valor lógico true:

```
dni IN (44200500, 20493495, 33900543, 98102392)
```

Usando la potencia de este operador, se podría sustituir dicha lista de valores explícita por los valores que devuelve una consulta. Por ejemplo, si se quisiera listar el nombre y apellidos de aquellos profesores que pertenecen al departamento de informática y comunicaciones, en vez de hacer el producto cartesiano de las tablas Profesor y Departamento, se podría buscar el código del departamento con nombre informática y comunicaciones, y este valor pasarlo a una consulta superior que selecciona aquellos profesores cuyo código de departamento es justo este valor. La consulta quedaría de la forma siguiente:

```
SELECT nombre, prApellido, sgApellido  
FROM Profesor  
WHERE codDep IN  
(SELECT codDep FROM Departamento  
WHERE nombre='Informática y Comunicaciones');
```

Obsérvese que la consulta segunda está escrita entre paréntesis y comporta una lista con el código del departamento cuyo nombre es informática y comunicaciones. Despues, la consulta principal muestra el nombre y apellidos de aquellos registros cuyo codDep está en dicha lista.

El funcionamiento es exactamente análogo al producto natural de ambas tablas, pero sin combinar todos los registros, con lo que aumenta la eficiencia tanto espacial como temporal.

Supóngase ahora que se quiere mostrar el DNI de aquellos alumnos que han obtenido una calificación mayor o igual que siete. Esta calificación se encuentra en la tabla LineaMatricula en la columna nota. Por lo tanto, esta tabla se usará para filtrar con la condición nota \geq 7. La consulta principal debe partir de Matricula, que es la tabla que contiene la columna que se quiere proyectar, es decir, el DNI del alumno.

La consulta principal partirá de la siguiente manera:

```
select dni FROM Matricula WHERE
```

A continuación, se quiere llegar hasta LineaMatricula, por lo que será necesario buscar los campos claves que hacen llegar hasta ella. Desde Matricula se llega hasta LineaMatricula a traves del campo común codMatr, por lo tanto, estos son los campos que se incluyen en el operador IN. La consulta queda como sigue:

```
SELECT dni FROM Matricula  
WHERE codMatr IN  
    (SELECT codMatr FROM LineaMatricula  
    WHERE nota $\geq$ 7);
```

La consulta haciendo uso del operador JOIN que es equivalente a la anterior seria la siguiente:

```
SELECT dni FROM Matricula JOIN LineaMatricula ON  
Matricula.codMatr=LineaMatricula.codMatr WHERE nota $\geq$ 7;
```

su equivalente usando la palabra USING:

```
SELECT dni FROM Matricula JOIN LineaMatricula USING(codMatr) WHERE nota $\geq$ 7;
```

Supóngase ahora que se quiere mostrar el código de las asignaturas en las que se ha matriculado el alumno David Chaparro Gómez. Se comienza desde las columnas que se quieren proyectar. Para este caso, seria codAsig. A continuación, se debe localizar qué tabla contiene dicha columna que llega hasta la tabla que contiene los datos que filtrar.

El nombre y apellidos del alumno se encuentran en la tabla Alumno, por tanto, se debe llegar hasta ella. El recorrido más corto proyectando la columna CodAsig es desde LineaMatricula hasta la tabla Alumno. Por tanto, las tablas que recorrer son LineaMatricula, Matricula y Alumno buscando el código clave de la tabla en la lista de códigos que devuelve la subconsulta. Esta subconsulta está compuesta de tres consultas anidadas:

```

SELECT codAsig FROM LineaMatricula
WHERE codMatr IN
    (SELECT codMatr FROM Matricula
     WHERE dni IN
         (SELECT dni FROM Alumno
          WHERE nombre='David' AND prApellido='Chaparro' AND sgApellido='Gómez'));
```

El uso de alias puede ser que ya no sea necesario en cada una de las consultas anidadas, ya que, en cada una de ellas, solo se encuentra la tabla que se requiere y, por tanto, sus columnas son únicas. Se debe tener en cuenta que no siempre se cumple esto, es decir, una consulta anidada puede contener más de una tabla en la cláusula FROM, dependiendo de los requisitos de filtrado.

Supóngase que se quiere hacer la misma consulta, pero filtrando las asignaturas en las que está matriculado en el curso actual. Este filtrado estará en la subconsulta que contenga la tabla que posee la columna curso, es decir, Matricula. La consulta final queda como sigue:

```

SELECT codAsig FROM LineaMatricula
WHERE codMatr IN
    (SELECT codMatr FROM Matricula
     WHERE curso=TO_NUMBER(TO_CHAR(sysdate,'YYYY'))
     AND dni IN
         (SELECT dni FROM Alumno
          WHERE nombre='David' AND prApellido='Chaparro' AND sgApellido='Gómez'));
```

La consulta equivalente a esta haciendo uso del operador JOIN es la siguiente:

```

SELECT codAsig
FROM LineaMatricula JOIN Matricula USING(codMatr)
JOIN Alumno USING(dni)
WHERE curso=TO_NUMBER(TO_CHAR(sysdate,'YYYY'))
AND nombre='David' AND prApellido='Chaparro' AND sgApellido='Gómez');
```

PRÁCTICA PROPUESTA

SUBCONSULTAS

- Mostrar las asignaturas impartidas por los profesores del departamento de informática y comunicaciones ordenados de mayor a menor número de horas.
- Mostrar la calificación más alta que tienen los alumnos bilingües en asignaturas no bilingües del ciclo formativo con sigla ASIR.
- Mostrar el nombre y apellidos de los alumnos que tienen una calificación mayor o igual que 9 ordenados por el primer apellido.
- Mostrar el nombre y apellidos de los profesores que imparten clases de las asignaturas Gestión de Bases de Datos y Acceso de Datos.
- Mostrar el nombre de ciclo que tiene la asignatura de máxima duración.
- Mostrar el nombre y apellidos de los alumnos bilingües que están matriculados en asignaturas bilingües de más de 150 horas y son del ciclo con sigla DAM o DAW.
- Mostrar el nombre de las asignaturas que son bilingües y pertenecen al ciclo con sigla DAW o DAM impartidas por el jefe de los departamentos cuyo nombre empieza por I.

ANY y ALL

Estos operadores se usan en subconsultas que devuelven más de un valor y se requiere el uso de los operadores de comparación **>**, **>=**, **=**, **<=** y **<**. De esto modo, se indicará si es mayor, menor o igual que todos o cualquiera del conjunto de valores que devuelve la subconsulta.

Cuando la subconsulta devuelve varios valores, se usa ANY para comparar si la expresión de la izquierda es mayor, mayor o igual, igual, menor o igual o menor que algún valor de la lista devuelta por la subconsulta. Si se quiere comparar que la expresión de la izquierda es mayor, mayor o igual, igual, menor o igual o menor que todos los valores de la derecha, se usa ALL. Los operadores ANY y SOME son sinónimos.

Por ejemplo, la siguiente consulta devuelve el nombre y apellidos de los alumnos que tienen una nota en cualquier asignatura mayor que todos los demás en gestión de bases de datos:

```
SELECT nombre, prApellido, sgApellido
FROM Alumno
WHERE dni IN
    (SELECT dni FROM Matricula
     WHERE codMatr IN
         (SELECT codMatr FROM LineaMatricula
          WHERE nota>ALL
              (SELECT nota FROM LineaMatricula
               WHERE codASig IN
                   (SELECT codASig FROM Asignatura
                    WHERE nombre='Gestión de bases de datos'))));
```

PRÁCTICA PROPUESTA

ANY Y ALL

realiza las siguientes consultas en las que se hace necesario usar operadores ANY y ALL:

- Mostrar el nombre y los apellidos de los alumnos que tienen más nota en Gestión de Bases de Datos que el que tiene menos nota en Acceso a Datos.
- Mostrar el nombre y apellidos de los profesores del departamento de Informática y Comunicaciones que han aprobado a todos los alumnos matriculados en el curso actual.
- Mostrar el nombre y apellidos de los alumnos del curso actual que tienen más nota que el curso pasado en Gestión de bases de datos.

Subconsultas correlacionadas

Existe un tipo de subqueries llamada correlada o correlacionada. Estas consultas consisten en referir en el where de subconsulta a un atributo de la consulta principal.

Veamos un ejemplo.

Sea la tabla trabajadores(dni, nombre, salario, departamento);

Cual es el trabajador que más gana por departamento?

Esta consulta no es más que seleccionar aquel trabajador cuyo salario es igual al salario máximo DE SU departamento. Para poner ese 'DE SU' habrá que restringir la subconsulta a sólo su departamento:

```
select * from trabajadores t where salario = (select max(salario) from trabajadores where departamento=t.departamento);
```

Se podría haber hecho esta consulta sin correlar pero sería más compleja, en este caso:

```
select      *      from      trabajadores      where      (departamento,salario)      in      (select      departamento,max(salario)      from      trabajadores      group      by      departamento);
```

EXISTS

Un caso de consulta correlada es la que usa la cláusula EXISTS. Esta subconsulta devuelve verdadero si existe alguna fila que cumpla la query, es decir, si la subconsulta devuelve alguna fila.

Por ejemplo. Sean las tablas :

```
departamento(coddep, nombre);
trabajador(dni, nombre, coddep);
```

Hay que seleccionar los trabajadores del departamento de 'ventas', esto se puede pensar de dos formas:

1. Seleccionamos el trabajador cuyo coddep sea igual al coddep del departamento cuyo nombre es ventas:

```
select nombre from trabajador where coddep in (select coddep from departamento where nombre='ventas');
```

2. Seleccionamos el trabajador si existe una fila en la tabla departamento cuyo nombre es ventas y el código coincide con el código del departamento del trabajador:

```
select nombre from trabajador t where exists (select * from departamento where nombre='ventas' and coddep=t.coddep);
```

Fijémonos como la condición de asociación es la misma pero puesta en distintos sitios. Aunque las dos select devuelven los mismos valores muchas veces es más fácil hacerlas con EXISTS que con IN

El problema de los nulos

Cuando la subconsulta devuelve nulos en su lista de valores, la consulta con IN y con EXISTS devuelve lo mismo. Sin embargo con NOT IN y NOT EXISTS la cosa cambia.

En la base de datos anterior supón que hay trabajadores sin departamento, es decir, que hay trabajadores cuyo coddep es nulo.

Vamos a obtener la lista de departamentos que no tienen trabajadores, podríamos pensar que es equivalente a listar los departamentos cuyo coddep no está en la lista de los coddep de los trabajadores:

```
select * from departamentos where coddep NOT IN (select coddep from trabajadores)
```

La consulta anterior está mal y no devolvería resultados pues la subconsulta devuelve nulos así que hay que filtrarlos. La siguiente consulta sí es correcta:

```
SELECT * FROM departamentos WHERE coddep NOT IN (select coddep from trabajadores where coddep is not null)
```

Cuando usamos NOT IN y la subconsulta selecciona la FK que puede admitir nulos, hay que filtrar con is not null

Sin embargo, con not exists no tenemos ese problema y la haríamos normalmente. La siguiente consulta es totalmente correcta:

```
SELECT * FROM departamentos t where NOT EXISTS (select * from trabajadores where coddep=t.coddep);
```

SELECTS ANIDADAS

Las subconsultas no solo se usan en las cláusulas WHERE; también se pueden utilizar consultas anidadas en cláusulas FROM. En el siguiente ejemplo, nuestra subconsulta no devuelve solo un valor, sino una tabla.

```
SELECT department_id, AVG(salary) AS media FROM employees GROUP BY department_id;
```

Ahora, vamos a averiguar cual es el salario medio por departamento más alto. Primero, tendremos que calcular el salario medio por departamento y, luego, utilizar esa tabla para encontrar el salario medio más alto. Lo podemos hacer mediante la siguiente consulta:

```
SELECT MAX(media) AS maximo
FROM (
  SELECT department_id, AVG(salary) AS media FROM employees GROUP BY department_id
);
```

Observa que la consulta interna devuelve una tabla con varias filas y columnas. Para ser más específicos, la tabla tiene dos columnas, department_id y media, que muestran el salario medio de los trabajadores en función de su departamento. El número de filas corresponde al número de departamentos (es decir, los que aparecen en la tabla employees).

A continuación, la consulta externa solo calcula el salario medio más alto basándose en la tabla que hay en el from, y devuelve ese valor.

Limitando el número de resultados.

Rownum

ROWNUM es una pseudocolumna (no una columna real) que está disponible en una consulta. A ROWNUM se le asignarán los números 1, 2, 3, 4, ... N , donde N es el número de filas en el conjunto con el que se usa ROWNUM. Un valor ROWNUM no se asigna permanentemente a una fila (este es un error común). Una fila de una tabla no tiene un número; no se puede pedir la fila 5 de una tabla; no existe tal cosa.

También resulta confuso para muchas personas cuándo se asigna realmente un valor ROWNUM. Se asigna un valor ROWNUM a una fila después de que pasa la fase de predicado de la consulta, pero antes de que la consulta realice alguna clasificación o agregación. Además, un valor ROWNUM se incrementa solo después de asignarlo, razón por la cual where rownum>valor nunca devolverá una fila pues nunca es verdadero ya para la primera fila.

Paginar

Supongamos que tienes una consulta que devuelve una gran cantidad de datos: miles, cientos de miles o más filas. Sin embargo, a ti sólo te interesan los N primeros, digamos los 10 primeros o los 100 primeros. Hay dos formas de abordar esto:

Hacer que la aplicación cliente ejecute esa consulta y obtenga solo las primeras N filas.

Utilizar esa consulta como una vista en línea y utilizar ROWNUM para limitar los resultados, como en `SELECT * FROM (la consulta) WHERE ROWNUM <= N`.

El segundo enfoque es muy superior al primero, por dos razones. La menor de las dos razones es que requiere menos trabajo por parte del programador, porque la base de datos se encarga de limitar el conjunto de resultados. La razón más importante es el procesamiento especial que puede realizar la base de datos para brindarte solo las N primeras filas. Usar la consulta top-N significa que le has proporcionado información adicional a la base de datos. Tú le has dicho: "Sólo me interesa obtener N filas; nunca consideraré el resto". Esto es mucho más eficiente.

Es importante recalcar que a la hora de ordenar, será más eficiente si ordenamos por algo único, así pues si por ejemplo ordenamos por un valor (por ejemplo salario o precio) que puede tener muchos valores repetidos podemos introducir dentro de la ordenación por ejemplo el ROWID

El rownum se usa en el where de esta manera: `rownum {< | <= | valor} | {=1}` siendo *valor* un entero que indica el número máximo de tuplas que queremos. Hay que tener en cuenta que ya que rownum se evalúa en el where podemos llevar sorpresas. Para evitarlas, debemos tener claro el orden en que Oracle realiza las consultas y en caso de duda realizar una metaquery siempre.

Por ejemplo la siguiente consulta parece que devuelve el producto más caro:

```
select * from produtos where rownum=1 order by prezo desc;
```

Cuando realmente lo que estamos haciendo es seleccionando el primer producto y ordenándolo.

Esta otra consulta parece que se queda con el precio medio de las dos primeras categorías:

```
select categoria, avg(prezo) from produtos where rownum<3 group by categoria;
```

Sabrías decir lo que hace la consulta anterior?

La forma correcta de hacer estas dos consultas sería:

```
select a.* from (select * from produto order by prezo desc) a where rownum<2;
```

```
select a.* from (select avg(prezo) from produtos group by categoria) a where rownum<3;
```

Ya veremos más adelante como el uso de vistas puede simplificarnos estos problemas.

Si necesitamos un conjunto de filas entre dos valores:

```
select * from
( select a.* , ROWNUM rnum from
  ( consulta con el order by ) a
  where ROWNUM <= :MAX_ROW_TO_FETCH )
where rnum >= :MIN_ROW_TO_FETCH;
```

Offset... fetch first

La manera moderna de hacer la consulta TOP-N es usando

FETCH FIRST n ROWS ONLY que sí es usable en la consulta real por lo que no tenemos que hacer selects anidadas. En el resto de gestores existe una cláusula parecida llamada en unos casos limit, offset, etc.

OFFSET n ROWS sirve para indicar el desplazamiento en la ventana de paginación. Por ejemplo esta consulta devuelve los 5 primeros empleados que más ganan.

```
select * from employees order by salary desc fetch first 5 rows only;
```

Esta otra devuelve los 5 siguientes:

```
select * from employees order by salary desc offset 5 rows fetch first 5 rows only;
```

vemos como este método es mucho más simple que el antiguo con rownum:

```
select * from (
  select a.* ,rownum as r from
    (select * from employees order by salary) a
  where rownum<=10) where r>5;
```

ROW_NUMBER() y SELECT OVER...

Otra solución interesante es usar la función row_number(). Esta función asigna un número consecutivo a cada fila devuelta por la select según la ordenación que se especifica en over...

Ejemplo de select que devuleve los 5 primeros trabajadores que más ganan:

```
SELECT *
FROM (
  SELECT employees.* ,row_number() OVER (ORDER BY salary desc) rn
  FROM employees
) WHERE rn <=5;
```

Es parecida a usar rownum, pero tiene una ventaja respecto a usar offset y fetch first n rows: permite devolver las n primeras filas de una lista ordenada agrupada por un valor.

Es decir, agrupamos sin usar funciones de agregación. Esto se hace con la cláusula

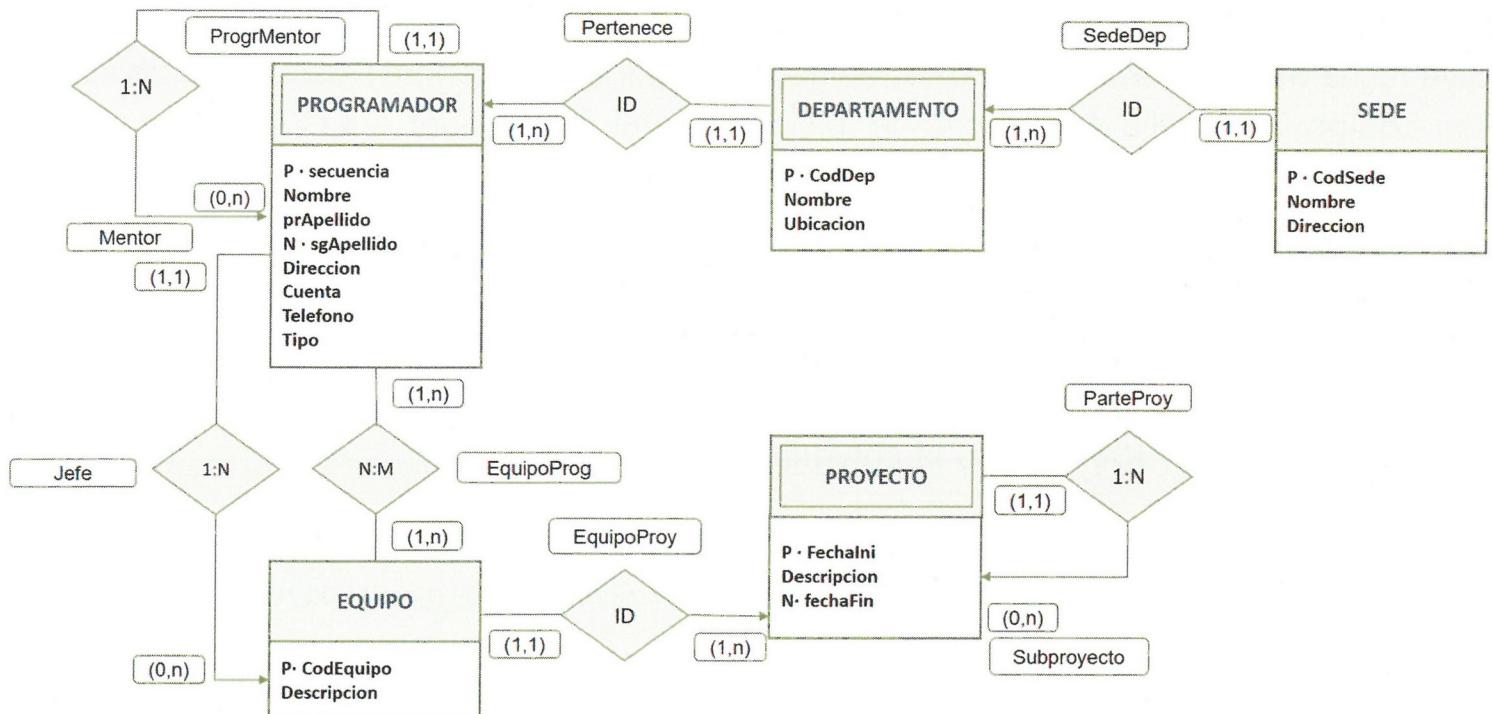
PARTITION BY

Por ejemplo en este caso podemos seleccionar los 3 trabajadores que más ganan de cada departamento:

```
SELECT *
FROM (
  SELECT employees.* ,row_number() OVER (partition by department_id ORDER BY salary desc) rn
  FROM employees
) WHERE rn<=3
order by department_id;
```

COSULTAS CON PK COMPUESTA

Cuando aparecen claves primarias que están compuestas por más de una columna afectan a que así sean también las claves ajenas. La ventaja es que, en la misma tabla, se tiene el código de la entidad de la que depende, y esta, cuando se relaciona, también se propaga. La desventaja es que, en el JOIN, se deben tener en cuenta todas las columnas que comportan las claves.



Sede(P.CodSede, Nombre, Direccion)

Departamento(PF.CodSede → Sede, P.CodDep, Nombre, Ubicacion)

Programador(PF.CodSede, PF.CodDep → Departamento, P.Secuencia, Nombre, prApellido, N.sgApellido, Direccion, Cuenta, Telefono, Tipo, FN.CodSedeMentor, FN.CodDepMentor, FN.SecuenciaMentor → Programador)

Equipo(P.CodEquipo, Descripcion, F.CSede, F.CDep, F.Sec → Programador)

Proyecto(PF.CodEquipo → Equipo, P.Fechalni, Descripcion, N.FechaFin, FN.CSub, FN.FISub → Proyecto)

EquipoProg(PF.CodSede, PF.CodDep, PF.secuencia → Programador, PF.CodEquipo → Equipo)

Para mostrar los subproyectos del proyecto Tecnisa que se han iniciado en el año actual, se procede comprobando si los campos CSub y FISub valen NULL, lo que quiere decir que se trata de un proyecto. Si no, se trata de un subproyecto que pertenece al proyecto con Código CSub,FISub. La consulta quedaría:

```
SELECT P.Descripcion FROM Proyecto P, Proyecto S
WHERE P.Descripcion='Tecnisa'
AND TO_CHAR(FechaIni,'YYYY')=TO_CHAR(SYSDATE,'YYYY')
AND P.CodEquipo=S.CSub AND P.FechaIni=S.FISub;
```

Usando el operador JOIN ON sería:

```
SELECT P.Descripcion  
FROM Proyecto P JOIN Proyecto S ON (P.CodEquipo=S.CSub AND P.FechaIni=S.FISub)  
WHERE P.Descripcion='Tecnisa'  
AND TO_CHAR(FechaIni,'YYYY')=TO_CHAR(SYSDATE,'YYYY');
```

Supóngase ahora que se quieren mostrar los proyectos que aún no se han acabado. Para ello, la fecha fin aún está a NULL. La consulta queda como sigue:

Select descripcion from proyecto where fechaFin is null;

Supóngase que se quieren mostrar los proyectos con Descripcion Equipo Desarrollo 4 cuya fecha es anterior al 1/1/2023. Se debe cumplir que el campo Descripcion de equipo sea Equipo Desarrollo 4, la fecha de inicio, fechalni sea inferior a 1/1/23. La consulta es la siguiente:

```
SELECT P.Descripcion FROM Proyecto P, Equipo E  
WHERE E.Descripcion='Equipo Desarrollo 4' AND FechaIni<TO_DATE('01/01/23')  
AND P.CodEquipo=E.CodEquipo;
```

Usando el operador JOIN USING la consulta seria:

```
SELECT P.Descripcion FROM Proyecto P JOIN Equipo E USING (codEquipo)  
WHERE E.Descripcion='Equipo Desarrollo 4' AND FechaIni<TO_DATE('01/01/23');
```

Supóngase ahora que se quiere mostrar el nombre de los proyectos raíces, es decir, aquellos que no son subproyectos, que han comenzado en el año 2023. Para ello, la clave de proyecto que apunta al proyecto al que pertenece debe estar puesto a NULL, es decir, los campos CSub y FISub:

```
SELECT Descripcion FROM Proyecto P  
WHERE TO_CHAR(FechaIni,'YYYY')='2023'  
AND CodES IS NULL AND FIni IS NULL;
```

Ahora, por ejemplo, se quiere mostrar el nombre y los apellidos de los programadores del departamento de producción. Basta con hacer el producto natural de la tabla Programador con Departamento.

```
SELECT P.nombre, prApellido, sgApellido FROM Programador P, Departamento D  
WHERE D.Nombre='Producción'  
AND P.CodDep=D.CodDep;
```

Supóngase que se quieren mostrar todos los programadores mentores que trabajan en la sede Servicios Centrales. Aquellos programadores mentores tienen como clave ajena mentor como ajena. La clave ajena se encuentra compuesta por CodSedeMentor, CodDepMentor y SecuenciaMentor. La consulta queda como sigue:

```

SELECT P.nombre, prApellido, sgApellido
FROM Programador P, Sede S
WHERE S.Nombre='Servicios Centrales'
AND P.CodSedeMentor IS NULL AND P.CodDepMentor IS NULL AND secuenciaMentor IS
NULL
AND P.CodSede=S.CodSede;

```

La consulta anterior también se puede implementar usando JOIN ON. Esta consulta seria:

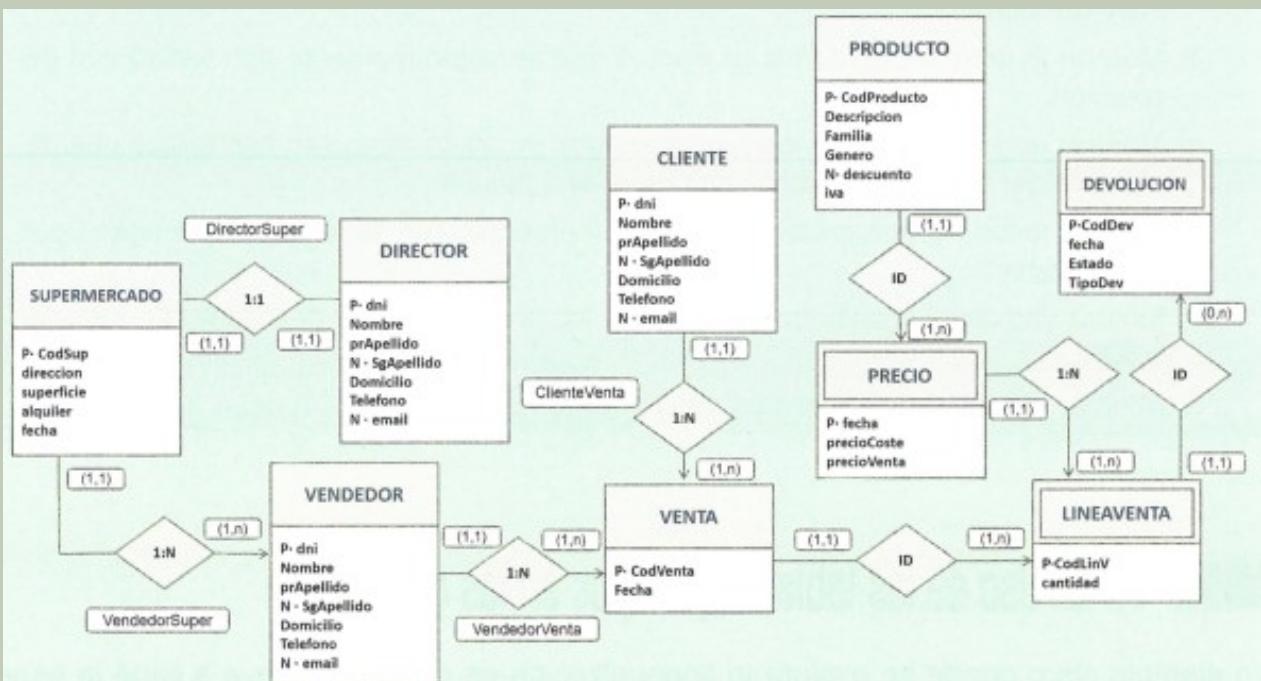
```

SELECT P.nombre, prApellido, sgApellido
FROM Programador P JOIN Sede S ON (P.codSede=S.codSede)
WHERE S.Nombre='Servicios Centrales'
AND P.CodSedeMentor IS NULL AND P.CodDepMentor IS NULL AND secuenciaMentor IS NULL;

```

PRÁCTICA PROPUESTA

CLAVES COMPUSTAS



- Mostrar los códigos de los productos cuyo precio está comprendido en el rango 100 €—200 €.
- Mostrar el nombre de los productos cuyo precio está comprendido en el rango 100 €—200 €.
- Mostrar el código de los productos que son muy baratos (menos de 10 €) o muy caros (más de 500 €).
- Mostrar los vendedores que trabajan en el supermercado con código 105.
- Mostrar los vendedores que trabajan en el supermercado con dirección C/ San Luis, 1234, 41003.
- Mostrar todos los clientes de los que se conoce su correo electrónico.
- ¿Se podrían consultar todos los empleados que trabajan en el supermercado con código 102 usando una única consulta?
- Mostrar los clientes que nunca han comprado nada.
- Mostrar todas las líneas de ventas de la venta con código 302.
- Mostrar el código de los clientes que han hecho alguna compra durante el mes de febrero del año pasado.
- Mostrar todas las líneas de ventas de las ventas efectuadas durante el mes de enero del presente año.
- Mostrar el nombre y apellidos de los clientes que han hecho alguna compra durante el mes de marzo de hace dos años

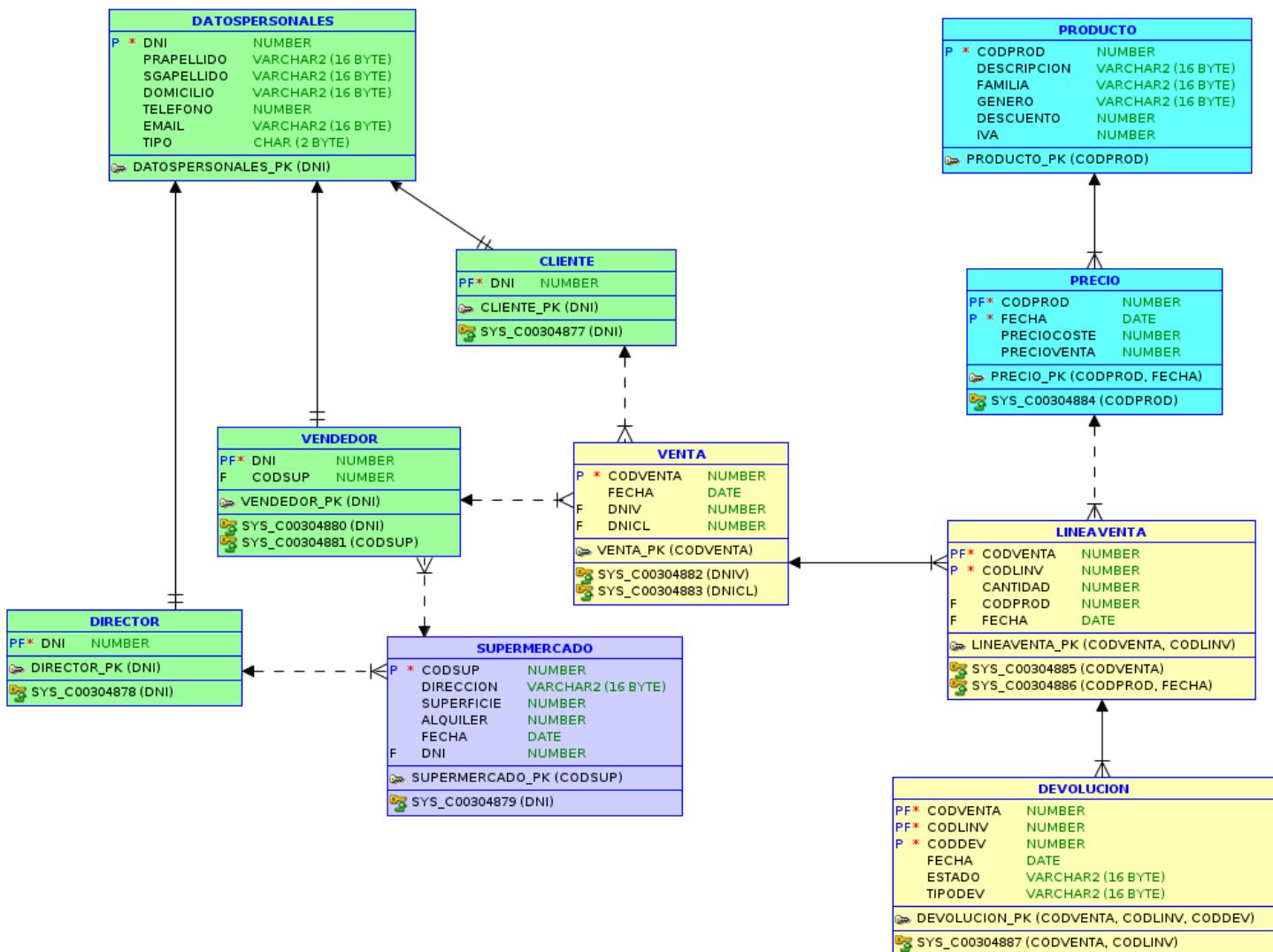
PRÁCTICA PROPUESTA

CLAVES COMPUSTAS II

- m) Mostrar el nombre y apellidos de los directores que trabajan en supermercados con más de 1000 metros cuadrados de superficie.
- n) Mostrar el código de los clientes que han sido atendidos por el vendedor con nombre y apellidos David Blanco Murcia.
- o) Mostrar la descripción de todos los productos que ha vendido la vendedora Raquel Palacios Gómez.
- p) Mostrar la descripción de los productos que en algún momento han tenido una devolución.
- q) Mostrar la descripción de los productos que en algún momento han tenido una devolución del tipo «Mal estado» durante el año pasado.
- r) Mostrar todos los precios que ha tenido el producto con descripción «Lentejas Gourmet adeide».
- s) Mostrar las devoluciones que han tenido los productos cuyo descuento es superior a 0,5.
- t) Mostrar el código de las ventas que tienen un producto con IVA del 4 %.

JOIN de varias veces la misma tabla

Supongamos la base de datos de la figura:



Supóngase que se quieren mostrar el nombre y apellidos de los clientes que han comprado algún producto y ha sido atendido por un vendedor con nombre Antonio, Alejandro o David. Se hace necesario tener la tabla Cliente y la tabla Vendedor para acceder al campo nombre, pero ocurre que es la misma tabla. Se debe tener en cuenta que es necesario combinar DatosPersonales con Venta a través de su campo dni y, mas tarde, se debe combinar DatosPersonales con la condición venta.dniV=datospersonales.dni. No puede usarse la misma versión de DatosPersonales porque si no lo que significaría es que tanto el vendedor como el cliente se llaman igual.

Para distinguir el primer JOIN que usa la tabla DatosPersonales del segundo producto natural que también usa DatosPersonales, se deben usar alias para cada tabla DatosPersonales, ya que se hace necesario incluir la misma tabla dos veces. Se usa el alias d2 para la segunda tabla DatosPersonales. La consulta queda como sigue:

```
SELECT nombre, prApellido, sgApellido
FROM DatosPersonales d JOIN Venta v ON d.dni=v.dni
JOIN DatosPersonales d2 ON v.dniV=d2.dni
WHERE d2.Nombre IN ('Antonio','Alejandro','David');
```

Usando subconsultas, este problema no existe, ya que la última subconsulta tiene en el FROM la tabla DatosPersonales, que equivale a d2 de la consulta anterior y ya “sin querer” incluimos otra versión de la tabla. La consulta es la siguiente:

```
SELECT nombre, prApellido, sgApellido
FROM DatosPersonales
WHERE dni IN
  (SELECT dniCI FROM venta
   WHERE dniV IN
     (SELECT dni FROM DatosPersonales
      WHERE Nombre IN ('Antonio','Alejandro','David')));
```

A veces es obligatorio hacer JOIN ya que queremos mostrar campos de diversas tablas. Supóngase ahora que se quieren mostrar el nombre y apellidos de los directores que dirigen supermercados donde hay algún vendedor cuyo apellido empieza por A que ha vendido algún producto que empieza por B a un cliente cuyo nombre empieza por C. En esta consulta, se necesita tener la tabla DatosPersonales como director, como vendedor y como cliente. En esta consulta, se hace necesario incluir DatosPersonales tres veces. Se usan los alias d1, d2 y d3 para distinguirlas. La consulta resultante queda como sigue:

```
SELECT d.Nombre, d.prApellido, d.sgApellido
FROM DatosPersonales d JOIN Supermercado ON d.dni=Supermercado.dni
JOIN Vendedor ON Supermercado.codsup=Vendedor.codsup
JOIN DatosPersonales d2 ON Vendedor.dni=d2.dni
JOIN Venta ON venta.dniV=dp2.dni
JOIN LineaVenta ON Venta.codventa=lineaventa.codventa
JOIN Producto ON LineaVenta.codprod=Producto.codprod
JOIN DatosPersonales d3 ON venta.dniC=d3.dni
WHERE d2.nombre LIKE 'A%' AND producto.nombre LIKE 'B%' AND d3.nombre LIKE
```

```
SELECT nombre, prApellido, sgApellido
FROM DatosPersonales WHERE dni IN
  (SELECT dni FROM Supermercado WHERE codsup IN
    (SELECT codsup FROM Vendedor WHERE dni IN
      (SELECT dni FROM DatosPersonales
       WHERE prApellido LIKE 'A%' AND dni IN
         (SELECT dniV FROM venta
          WHERE CodVenta IN
            (SELECT CodVenta FROM LineaVenta WHERE CodProd IN
              (SELECT CodProd FROM Producto
               WHERE Nombre LIKE 'B%'))))
      AND dniC IN (SELECT dni FROM DatosPersonales
       WHERE Nombre LIKE 'C%'))));
```

OPERACIONES CON CONJUNTOS

Entre dos conjuntos, que pueden ser los valores obtenidos de dos consultas diferentes, puede resultar un nuevo conjunto tras las operaciones diferencia, unión e intersección.

Los esquemas de las tablas que intervienen en la Operación deben ser idénticos en número, tipo y orden de las columnas. Los operadores para hacer estas operaciones son UNION, MINUS e INTERSECT.

UNION permite añadir el resultado del primer SELECT al segundo SELECT.

```
SELECT nombre, prApellido, sgApellido FROM Profesor  
UNION  
SELECT nombre, prApellido, sgApellido FROM Alumno;
```

Con esta consulta, se obtendrán los profesores y alumnos almacenados en la base de datos. Si hubiera algún alumno que también fuese profesor, se seleccionaría una sola vez. En caso de querer obtener las filas que se repiten, se usará el operador UNION ALL.

Con **INTERSECT** se unen dos consultas cuyo resultado será las filas que estén presentes en ambas. Por ejemplo, con la siguiente consulta, se obtiene las asignaturas que tienen más de 100 horas y las que pertenecen al ciclo con sigla ASIR:

```
SELECT nombre FROM Asignatura WHERE numHoras>100  
INTERSECT  
SELECT nombre FROM Asignatura JOIN Ciclo USING(CodCF) WHERE UPPER(sigla)=’ASIR’;
```

Con el operador **MINUS**, se obtienen las filas que estén en la primera consulta, pero que no estén en la segunda. Por ejemplo, la siguiente consulta muestra los códigos de las asignaturas que no están en la tabla LineaMatricula.

```
SELECT CodAsig FROM Asignatura  
MINUS  
SELECT CodAsig FROM LineaMatricula;
```

Obviamente, las consultas se pueden combinar, siendo la salida de entre dos de ellas la de la izquierda del segundo operador. Se usarán los paréntesis necesarios para agrupar los operadores como si se tratases de operadores aritméticos. Por ejemplo:

```
(SELECT CodAsig FROM Asignatura INTERSECT SELECT CodAsig FROM LineaMatricula)  
MINUS  
SELECT CodAsig FROM LineaContrato WHERE dni=29874102;
```


INSERT, UPDATE, DELETE USANDO SELECTS

Las operaciones de inserción, actualización y eliminación también pueden incluir consultas. En las inserciones de datos, la sintaxis es

INSERT INTO Tabla SELECT consulta.

En las actualizaciones a través del comando UPDATE, la sintaxis que hay que seguir es **UPDATE Tabla SET campo=consulta [WHERE subconsulta].**

Otro caso bastante útil es la eliminación de registros dependiendo de los resultados de una consulta. El comando que se debe usar es DELETE con la sintaxis

DELETE [FROM] Tabla WHERE subconsulta.

Ejemplos:

```
INSERT INTO superEmpleado(id, apellidos) SELECT id, apellidos FROM Empleado  
WHERE sueldo>4999;
```

```
UPDATE Empleado SET sueldo=(SELECT avg(sueldo) from empleado) WHERE  
sueldo<(SELECT AVG(sueldo) FROM Empleado);
```

```
DELETE Empleado WHERE dni IN (SELECT dni FROM Venta WHERE CodProd IN  
(SELECT CodProd FROM Producto WHERE CodProd IN (SELECT CodProd FROM  
Devolucion)));
```

Sabrías decir un fallo de la consulta anterior?

Solucionario

consultas simples

- a) SELECT nombre, prApellido, sgApellido FROM Cliente
WHERE ciudad='Barcelona';
- b) SELECT telefono FROM TelefonoCto
WHERE codContacto=1;
- c) SELECT correo FROM CorreoCto
WHERE codContacto=1;
- d) SELECT Nombre, prApellido, sgApellido FROM Empleado
WHERE ciudad='Madrid';
- e) SELECT nombreCompleto FROM Contacto
WHERE cargo='Contable';
- f) SELECT nombreCompleto FROM Contacto
WHERE codProv=1;
- g) SELECT codCompra, fechaHora FROM Compra
WHERE codProv=5;
- h) SELECT nombreComercial FROM Producto
WHERE prCoste>=20 AND prCoste<=50;
- i) SELECT nombreComercial FROM Producto
WHERE prVenta>prCoste*2;

Consultas con operadores

- a) SELECT dni FROM Empleado
WHERE Ciudad NOT IN ('Málaga', 'Córdoba', 'Santander')
AND Nombre NOT IN ('Rosa', 'Elena') AND prApellido='Salas'
ORDER BY sgApellido DESC;
- b) SELECT dni FROM Empleado
WHERE Ciudad IN ('Sevilla', 'Barcelona', 'Madrid') AND sgApellido IS NULL
ORDER BY dni;
- c) SELECT codProducto FROM LineaVenta WHERE cant>=3
ORDER BY codProducto DESC;
- d) SELECT nombre "Nombre", prApellido "Primer apellido", sgApellido "Segundo apellido"
FROM Cliente
WHERE Ciudad='Barcelona' AND Nombre IN ('María', 'Rosa', 'Gabriela')
AND (prApellido IN ('Blasco', 'Palacios', 'García')
OR sgApellido IN ('Beltrán', 'Gómez'))
ORDER BY sgApellido;

Consultas con funciones numéricas

- a) SELECT COUNT(*) FROM Producto WHERE prCoste<prVenta/2;
- b) SELECT MAX(prVenta) FROM Producto WHERE iva=10;
- c) SELECT MIN(prVenta) FROM Producto;
- d) SELECT COUNT(*) FROM Producto WHERE tipo='P' AND prCoste<prVenta/3;
- e) SELECT SUM(prVenta-prCoste) FROM Producto WHERE tipo='F';
- f) SELECT AVG(prVenta-prCoste) FROM Producto WHERE tipo IN ('S','P');
- g) SELECT TRUNC(prCoste,1), TRUNC(prVenta,1) FROM Producto WHERE Tipo='C';
- h) SELECT ROUND(prCoste,1) FROM Producto
WHERE prCoste>=prVenta/1.2 AND tipo IN ('C','F','D');
- i) SELECT MOD(prVenta,prCoste) FROM Producto WHERE iva IN (10,21);
- j) SELECT COUNT(*) FROM Venta;

Consultas con LIKE

- a) SELECT nombreComercial FROM Producto WHERE nombreInt LIKE '%+%' ;
- b) SELECT CodProducto FROM Producto WHERE tipo='P' AND nombreComercial LIKE '%Mac%';
- c) SELECT dni FROM Empleado
WHERE (sueldo >1200 OR sueldo>1900) AND prApellido LIKE 'A%' ;
- d) SELECT dni FROM Cliente WHERE nombre LIKE '%a%'
AND (prApellido LIKE 'g%' OR prApellido LIKE 'n%')
AND (prApellido LIKE '%a' OR sgApellido LIKE '%z'));

Consultas con cadenas

- a) SELECT INITCAP(nombre), UPPER(prApellido), UPPER(sgApellido)
FROM Empleado WHERE Direccion LIKE 'C/%';
- b) SELECT UPPER(prApellido), UPPER(sgApellido), TRIM(direccion)
FROM Empleado WHERE Direccion LIKE '%0';
- c) SELECT UPPER(Nombre), UPPER(prApellido), UPPER(sgApellido) FROM Cliente
WHERE UPPER(SUBSTR(Nombre, 3,1)) IN ('R','S')
AND UPPER(SUBSTR(sgApellido,3,1)) IN ('A','L')
ORDER BY prApellido;

Consultas con JOINS

- a) Se parte de la tabla que contiene los campos que se quieren proyectar, es decir, Profesor. Lo siguiente es ver qué se debe filtrar y qué tablas se requieren para dicho filtrado. Como se requiere filtrar el nombre del departamento, la condición que hay que aplicar es:

`departamento.nombre='Informática y Comunicaciones'`

Por tanto, se necesitan dos tablas, Profesor y Departamento. El campo que relaciona departamento con profesor es CodDep, por lo que se debe exigir que se cumpla la condición:

`Departamento.CodDep=Profesor.CodDep`

La consulta queda como sigue:

```
SELECT dni FROM Profesor, Departamento  
WHERE Departamento.nombre='Informática y Comunicaciones'  
AND Departamento.CodDep=Profesor.CodDep;
```

Se pueden utilizar alias de tablas para hacer más cómoda la escritura. La consulta anterior quedaría:

```
SELECT dni FROM Profesor P, Departamento D  
WHERE D.nombre='Informática y Comunicaciones'  
AND D.CodDep=P.CodDep;
```

- b) Esta vez, se necesita la tabla Alumno para comprobar si es bilingüe, con el campo B puesto a S. Luego, se requiere la tabla Matricula para filtrar el año del curso. La consulta quedaría:

```
SELECT dni FROM Alumno, Matricula  
WHERE B='S' AND curso=2029  
AND Matricula.dni=Alumno.dni;
```

- c) La calificación obtenida en una asignatura se encuentra en la tabla LineaMatricula. Por esta razón, la tabla Matricula se debe cruzar con dicha tabla.

```
SELECT Nombre, prApellido  
FROM Alumno, Matricula, LineaMatricula  
WHERE Nota>7  
AND Alumno.dni=Matricula.dni  
AND Matricula.codMatr=LineaMatricula.codMatr;
```

Usando alias de tablas, la misma consulta sería como sigue:

```
SELECT Nombre, prApellido  
FROM Alumno A, Matricula M, LineaMatricula L  
WHERE Nota>7  
AND A.dni=M.dni AND M.codMatr=L.codMatr;
```

- d) Se parte de la tabla que contiene los campos que se quieren proyectar, es decir, Alumno. Para cruzarla con la información que se quiere filtrar, que es que el nombre de la asignatura sea Base de Datos o Acceso a Datos, se deben usar las tablas que las unen, que son Matrícula, para cruzar el DNI, y LineaMatricula, para cruzar el código de la asignatura. A continuación, se debe añadir la tabla Asignatura para filtrar por el nombre de la asignatura.

En esta consulta, se debe tener en cuenta que existe más de una columna con el mismo nombre. Existen dos columnas Nombre, una que hace referencia al nombre del alumno y la otra, al nombre de la asignatura, por lo que se tiene que usar el nombre de la tabla en dichos campos. El resultado final quedaría como sigue:

```
SELECT Al.Nombre, Al.prApellido, Al.sgApellido  
FROM Alumno Al, Matricula M, LineaMatricula L, Asignatura Ag  
WHERE Al.dni=M.dni  
AND M.codMatr=L.codMatr  
AND L.codAsig=Ag.codAsig  
AND UPPER(Ag.Nombre) IN ('GESTIÓN DE BASES DE DATOS', 'ACCESO A DATOS');
```

- e) Los campos que se quieren proyectar están en Alumno. Desde Alumno, se debe llegar hasta la tabla Ciclo para filtrar la sigla. En el camino, se debe filtrar que nota sea mayor o igual que cinco usando la tabla LineaMatricula. Obviamente, se hace necesario pasar por la tabla Matricula para cruzar el DNI. Los filtros son:

```
Sigla='DAM' AND nota>=5
```

Ahora, para cruzar las tablas, se comienza en Alumno y se pasa a Matricula:

Alumno.dni=Matricula.dni

A continuación, se cruza Matricula con LineaMatricula y con Asignatura:

```
Matricula.codMatr=LineaMatricula.codMatr  
AND LineaMatricula.CodAsig=Asignatura.CodAsig
```

Por último, se une Asignatura con Ciclo con la condición:

Asignatura.CodCF=Ciclo.CodCF

La consulta completa sería la siguiente:

```
SELECT Alumno.Nombre, prApellido, sgApellido  
FROM Alumno, Matricula, LineaMatricula, Asignatura, Ciclo  
WHERE Sigla='DAM' AND nota>=5  
AND Alumno.dni=Matricula.dni  
AND Matricula.codMatr=LineaMatricula.codMatr  
AND LineaMatricula.CodAsig=Asignatura.CodAsig  
AND Asignatura.CodCF=Ciclo.CodCF;
```

Los campos prApellido, sgApellido, Sigla y nota no necesitan su tabla origen, ya que no son campos ambiguos para esta consulta.

Usando alias, la consulta quedaría así:

```
SELECT Al.Nombre, prApellido, sgApellido  
FROM Alumno Al, Matricula M, LineaMatricula L, Asignatura Ag, Ciclo C  
WHERE Sigla='DAM' AND nota>=5  
AND Al.dni=M.dni  
AND M.codMatr=L.codMatr  
AND L.CodAsig=Ag.CodAsig  
AND Ag.CodCF=C.CodCF;
```

Consultas con JOIN ON y USING

a) SELECT dni FROM Profesor P JOIN Departamento D ON D.CodDep=P.CodDep
WHERE D.nombre='Informática y Comunicaciones' AND dni>44200200;

```
SELECT dni FROM Profesor P JOIN Departamento D USING(CodDep)  
WHERE D.nombre='Informática y Comunicaciones' AND dni>44200200;
```

b) SELECT dni
FROM Profesor P JOIN Departamento D ON D.CodDep=P.CodDep
JOIN Contrato C ON C.dni=P.dni

```
WHERE prApellido LIKE 'P%' AND sgApellido LIKE '%z'  
AND D.nombre='Administración y Finanzas' AND curso=2022;
```

```
SELECT dni  
FROM Profesor P JOIN Departamento D USING(CodDep)  
    JOIN Contrato C USING(dni)  
WHERE prApellido LIKE 'P%' AND sgApellido LIKE '%z'  
AND D.nombre='Administración y Finanzas' AND curso=2022;
```

c) SELECT Nombre, prApellido
FROM Profesor P JOIN Contrato C ON P.dni=C.dni
 JOIN LineaContrato L ON C.codCont=L.codCont
 JOIN Asignatura Ag ON L.codAsig=Ag.codAsig
WHERE Ag.Nombre='Acceso a datos';

```
SELECT Nombre, prApellido  
FROM Profesor P JOIN Contrato C USING(dni)  
    JOIN LineaContrato L USING(codCont)  
    JOIN Asignatura Ag USING(codAsig)  
WHERE Ag.Nombre='Acceso a datos';
```

d) SELECT Al.Nombre, prApellido, sgApellido
FROM Alumno Al JOIN Matricula M ON Al.dni=M.dni
 JOIN LineaMatricula L ON M.codMatr=L.codMatr
 JOIN Asignatura Ag ON L.codAsig=Ag.codAsig
WHERE Ag.Nombre IN ('Gestión de bases de datos', 'Acceso a datos') AND nota>=5;

```
SELECT Al.Nombre, prApellido, sgApellido  
FROM Alumno Al JOIN Matricula M USING(dni)  
    JOIN LineaMatricula L USING(codMatr)  
    JOIN Asignatura Ag USING(codAsig)  
WHERE Ag.Nombre IN ('Gestión de bases de datos', 'Acceso a datos') AND nota>=5;
```

e) SELECT Al.Nombre, prApellido, sgApellido
FROM Alumno Al JOIN Matricula M ON Al.dni=M.dni
 JOIN LineaMatricula L ON M.codMatr=L.codMatr
 JOIN Asignatura Ag ON L.codAsig=Ag.codAsig
 JOIN Ciclo C ON Ag.codCF=C.codCF
WHERE sigla='ASIR' AND nota>=5 AND sgApellido LIKE '%z%'
ORDER BY prApellido, sgApellido;

```
SELECT Al.Nombre, prApellido, sgApellido  
FROM Alumno Al JOIN Matricula M USING(dni)  
    JOIN LineaMatricula L USING(codMatr)  
    JOIN Asignatura Ag USING(codAsig)  
    JOIN Ciclo C USING(codCF)  
WHERE Sigla='ASIR' AND nota>=5 AND sgApellido LIKE '%z%'  
ORDER BY prApellido, sgApellido;
```

