

Realización de Consultas

Contenidos

- ☞ La sentencia SELECT
- ☞ Consultas básicas, filtros y ordenación
- ☞ Consultas resumen
- ☞ Subconsultas
- ☞ Consultas multitabla. Composiciones internas y externas
- ☞ Consultas reflexivas
- ☞ Consultas con tablas derivadas

Objetivos

- ☞ Identificar herramientas y sentencias para realizar consultas
- ☞ Identificar y crear consultas simples sobre una tabla
- ☞ Identificar y crear consultas que generan valores resumen
- ☞ Identificar y crear consultas con composiciones internas y externas
- ☞ Identificar y crear subconsultas
- ☞ Valorar las ventajas e inconvenientes de las distintas opciones válidas para realizar una consulta

Con este tema, se detalla la sintaxis completa de la sentencia SELECT en toda su extensión. Se proporciona al estudiante métodos para construir consultas simples y complejas de forma estructurada, con filtros, agrupaciones y ordenaciones.

4.1. El lenguaje DML

Las sentencias DML del lenguaje SQL son las siguientes:

- La sentencia **SELECT**, que se utiliza para extraer información de la base de datos, ya sea de una tabla o de varias.
- La sentencia **INSERT**, cuyo cometido es insertar uno o varios registros en alguna tabla.
- La sentencia **DELETE**, que borra registros de una tabla.
- La sentencia **UPDATE**, que modifica registros de una tabla.

Cualquier ejecución de un comando en un SGBD se denomina **CONSULTA**, término derivado del anglosajón **QUERY**. Este término debe ser entendido más que como una *consulta* de información, como una *orden*, es decir, las **QUERYS** o **CONSULTAS** no son solo **SELECT**, sino también cualquier sentencia de tipo **UPDATE, INSERT, CREATE, DROP, etc.** entendidas todas ellas como peticiones al SGBD para realizar una operación determinada.

4.2. La sentencia **SELECT**

La sentencia **SELECT** es la sentencia más versátil de todo SQL, y por tanto la más compleja de todas. Como se ha expuesto anteriormente, se utiliza para consultar información de determinadas tablas. Es posible ejecutar sentencias muy sencillas que muestran todos los registros de una tabla:

```
#esta consulta selecciona todos los campos y muestra todos los
#registros de la tabla empleados
SELECT * FROM empleados;
```

O incluso consultas que obtienen información filtrada de múltiples tablas, usando relaciones entre tablas e incluso tablas virtuales creadas a partir de una consulta.

```
#esta consulta obtiene el total de los pedidos
#de los clientes de una tienda
SELECT NombreCliente,tot.Cantidad
FROM Clientes,Pedidos,
     (SELECT sum(Cantidad*PrecioUnidad) as Cantidad,NumeroPedido
      FROM DetallePedidos GROUP BY NumeroPedido) tot
WHERE Clientes.NumeroCliente=Pedidos.NumeroCliente
AND Pedidos.numeroPedido=tot.NumeroPedido ORDER BY Cantidad;
```

4.3. Consultas básicas

El formato básico para hacer una consulta es el siguiente:

```
SELECT [DISTINCT] select_expr [,select_expr] ... [FROM tabla]
```

select_expr:

nombre_columna [AS alias]

| * (El carácter '*' sirve para mostrar TODO)

| expresión

nombre_columna indica un nombre de columna, es decir, se puede seleccionar de una tabla una serie de columnas, o todas si se usa *, o una expresión algebraica compuesta por operadores, operandos y *funciones*.

El parámetro opcional **DISTINCT** fuerza que solo se muestren los registros con valores distintos, o, dicho de otro modo, que suprima las repeticiones.

En la página siguiente, se muestran algunos ejemplos del uso de la sentencia **SELECT**. Hay que prestar atención a algunas curiosidades¹:

- En la consulta 4 se selecciona una columna calculada ($1+5$), con la selección de los registros de una tabla, incorporándose esa nueva columna creada al conjunto de filas devueltas por el gestor.
- En la consulta número 5 se hace uso de una expresión algebraica para crear una columna cuyo resultado será el de sumar 1 y 6. Para hacer esto, en MySQL no es necesario indicar la cláusula **FROM**, pero en Oracle, hay que poner la cláusula **FROM** con la tabla dual. Al no haber una tabla real seleccionada, el resultado será una única fila con el resultado.
- En la consulta 3 se hace uso de la función *concat* cuyo cometido es concatenar dos columnas creando una única columna. En Oracle, la función *concat* solo admite dos parámetros, mientras que en MySQL se puede concatenar múltiples parámetros. Para concatenar en oracle múltiples columnas se puede hacer uso del operador **||**. Véase Sección 6.4.
- En las consultas 6 y 7 se muestran las marcas de los vehículos. La diferencia entre ambas consultas está en el uso del **DISTINCT**, que elimina las repeticiones. Hay dos vehículos seat en la consulta 6 y uno en la 7.

¹La ejecución de las consultas que se muestra a continuación está realizada en MySQL, pero son perfectamente compatibles con Oracle, excepto la consulta 5, que hay que añadir "FROM dual".

```
#consulta 1
SELECT * FROM vehiculos;
+-----+
| matricula | modelo       | marca   |
+-----+
| 1129FGT  | ibiza gt      | seat    |
| 1132GHT  | leon tdi 105cv | seat    |
| M6836YX  | corolla g6     | toyota  |
| 7423FZY  | coupe          | hyundai |
| 3447BYD  | a3 tdi 130cv   | audi   |
+-----+
```

```
#consulta 2
SELECT matricula, modelo
FROM vehiculos;
```

```
+-----+
| matricula | modelo       |
+-----+
| 1129FGT  | ibiza gt      |
| 1132GHT  | leon tdi 105cv |
| M6836YX  | corolla g6     |
| 7423FZY  | coupe          |
| 3447BYD  | a3 tdi 130cv   |
+-----+
```

as "matricula coche"
para poder llamarlo
así con un espacio en
medio

```
#consulta 3
SELECT matricula,
       concat(marca,modelo)* as coche
FROM vehiculos;
```

```
+-----+
| matricula | coche        |
+-----+
| 1129FGT  | seatibiza gt   |
| 1132GHT  | seatleon tdi 105cv |
| M6836YX  | toyotacorolla g6 |
| 7423FZY  | hyundaicoupe   |
| 3447BYD  | audia3 tdi 130cv |
+-----+
```

* concat(marca," ",modelo) para dejar
un espacio en medio)

```
#consulta 4
SELECT matricula, modelo, 1+5
FROM vehiculos;
```

```
+-----+
| matricula | modelo       | 1+5   |
+-----+
| 1129FGT  | ibiza gt      | 6    |
| 1132GHT  | leon tdi 105cv | 6    |
| M6836YX  | corolla g6     | 6    |
| 7423FZY  | coupe          | 6    |
| 3447BYD  | a3 tdi 130cv   | 6    |
+-----+
```

#consulta 5

```
SELECT 1+6;
```

```
+-----+
| 1+6 |
+-----+
| 7 |
+-----+
```

#consulta 6

```
SELECT marca FROM vehiculos;
```

marca
seat
seat
toyota
hyundai
audi

#consulta 7

```
SELECT DISTINCT marca
FROM vehiculos;
```

marca
seat
toyota
hyundai
audi

◊ **Actividad 4.1:** Crea una tabla en MySQL con la siguiente estructura:
MASCOTAS(Nombre, especie, raza, edad, sexo).

Introduce 6 registros² y, a continuación, codifica las siguientes querys:

- Muestra el nombre y la especie de todas las mascotas.
- Muestra el nombre y el sexo de las mascotas poniendo un alias a los campos.
- Muestra el nombre y la fecha de nacimiento aproximada de las mascotas (consulta la documentación de MySQL y usa la función date_sub y now).

Realiza el mismo procedimiento creando la tabla en Oracle.

4.4. Filtros

Los filtros son condiciones que cualquier gestor de base de datos interpreta para seleccionar registros y mostrarlos como resultado de la consulta. En SQL la palabra clave para realizar filtros es la cláusula **WHERE**.

A continuación se añade a la sintaxis de la cláusula **SELECT** la sintaxis de los filtros:

```
SELECT [DISTINCT] select_expr [,select_expr] ...
[FROM tabla] [WHERE filtro]
```

filtro es una expresión que indica la condición o condiciones que deben satisfacer los registros para ser seleccionados.

Un ejemplo de funcionamiento sería el siguiente:

```
#selecciona los vehículos de la marca seat
SELECT * FROM vehiculos
WHERE marca='seat';
+-----+-----+-----+
| matricula | modelo | marca |
+-----+-----+-----+
| 1129FGT | ibiza gt | seat |
| 1132GHT | leon tdi 105cv | seat |
+-----+-----+-----+
```

² Véase Capítulo 5 si es necesario.

4.4.1. Expresiones para filtros

Los filtros se construyen mediante *expresiones*. Una expresión, es una combinación de operadores, operandos y funciones que producen un resultado. Por ejemplo, una expresión puede ser:

```
#expresión 1 (oracle): (2+3)*7
SELECT (2+3)*7 from dual;

(2+3)*7
-----
35

#expresión 2 (mysql): (2+3)>(6*2)
SELECT (2+3)>(6*2);
+-----+
| (2+3)>(6*2) |
+-----+
|          0 | #0 = falso, es falso que 5>12
+-----+

#expresión 3 (mysql): la fecha de hoy -31 años;
SELECT date_sub(now(), interval 31 year);
+-----+
| date_sub(now(), interval 31 year) |
+-----+
| 1977-10-30 13:41:40               |
+-----+
```

Se detalla a continuación los elementos que pueden formar parte de las expresiones:

- **Operando**: Los operandos pueden ser constantes, por ejemplo el número entero 3, el número real 2.3, la cadena de caracteres 'España' o la fecha '2010-01-02'; también pueden ser variables, por ejemplo el campo *edad* o el campo *NombreMascota*; y pueden ser también otras expresiones³.
- **Operadores aritméticos**: +, -, *, /, %. El operador + y el operador - se utilizan para sumar o restar dos operandos (binario) o para poner el signo positivo o negativo a un operando (unario). El operador * es la multiplicación de dos operandos y el operador / es para dividir. El operador % o resto de la división entera a %b devuelve el resto de dividir a entre b.

³Todas los operandos numéricos ya sean reales o enteros van sin comilla simple, y cualquier otra cosa que no sea número, por ejemplo, cadenas de caracteres o fechas, van entre comillas simples.

!=

- Operadores relacionales:** $>$, $<$, $<>$, $>=$, $<=$, $=$. Los operadores relacionales sirven para comparar dos operandos. Así, es posible preguntar si un campo es mayor que un valor, o si un valor es distinto de otro. Estos operadores devuelven un número entero, de tal manera que si el resultado de la expresión es *cierto* el resultado será 1, y si el resultado es *falso* el resultado será 0. Por ejemplo, la expresión $a > b$ devuelve 1 si a es estrictamente mayor que b y 0 en caso contrario. La expresión $d <> e$ devuelve 1 si d y e son valores distintos.
- Operadores lógicos:** AND, OR, NOT. Los operadores lógicos toman como operandos valores lógicos, esto es, cierto o falso, en caso de SQL, 1 o 0. Los operadores lógicos se comportan según las siguientes tablas de verdad:

Operando 1	Operando 2	Op1 AND Op2	Op1 OR Op2	NOT Op1
falso	falso	falso	falso	cierto
falso	cierto	falso	cierto	cierto
cierto	falso	falso	cierto	falso
cierto	cierto	cierto	cierto	falso

Cuadro 4.1: Tabla de verdad de los operadores lógicos.

Por otro lado, se necesita un tratamiento de los valores nulos; hay que incluir como un posible operando el valor nulo:

Operando 1	Operando 2	Op1 AND Op2	Op1 OR Op2	NOT Op1
falso	falso	falso	falso	cierto
falso	cierto	falso	cierto	cierto
cierto	falso	falso	cierto	falso
cierto	cierto	cierto	cierto	falso
nulo	X	nulo	nulo	nulo
X	nulo	nulo	nulo	no X

Cuadro 4.2: Tabla de verdad de los operadores lógicos con valores nulos.

- Paréntesis:** $()$. Los operadores tienen una prioridad, por ejemplo, en la expresión $3+4*2$, la multiplicación se aplica antes que la suma, se dice que el operador $*$ tiene más prioridad que el operador $+$. Para alterar esta prioridad, se puede usar el operador paréntesis, cuyo cometido es precisamente dar máxima prioridad a una parte de una expresión. Así, $(3+4)*2$, no es lo mismo que $3+4*2$.

- **Funciones:** date_add, concat, left, right.... Cada SGBD incorpora su propio repertorio de funciones que en pocas ocasiones coincide con el de otros SGBD.

En la tabla que se muestra a continuación aparecen los resultados que provoca la ejecución de algunos de los operadores descritos:

Operación	Resultado
$7+2*3$	13
$(7-2)*3$	15
$7>2$	1
$9<2$	0
$7>2 \text{ AND } 4<3$	0
$7>2 \text{ OR } 4<3$	1
$(10>=10 \text{ AND } 0<=1)+2$	3

Cuadro 4.3: Tabla resumen de los operadores usados en expresiones.

4.4.2. Construcción de filtros

A continuación, se muestran ejemplos de construcción de filtros para una base de datos de jugadores de la liga americana de baloncesto (NBA), todos ellos compatibles para Oracle y MySQL:

```
#la tabla jugadores contiene todos los jugadores de la nba
describe jugadores;
+-----+-----+-----+-----+-----+
| Field      | Type       | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| codigo     | int(11)    | NO   | PRI | NULL    |       |
| Nombre     | varchar(30) | YES  |     | NULL    |       |
| Procedencia | varchar(20) | YES  |     | NULL    |       |
| Altura     | varchar(4)  | YES  |     | NULL    |       |
| Peso        | int(11)    | YES  |     | NULL    |       |
| Posicion    | varchar(5)  | YES  |     | NULL    |       |
| Nombre_equipo | varchar(20) | YES  | MUL | NULL    |       |
+-----+-----+-----+-----+-----+
#Consulta que selecciona los nombres de los jugadores de los Lakers
SELECT Nombre FROM jugadores WHERE Nombre_equipo='Lakers';
+-----+
| Nombre          |
+-----+
| Ron Artest     |
+-----+
```

Kobe Bryant	
...	
Pau Gasol	
+-----+	

#Consulta que selecciona los jugadores españoles de los Lakers

```
SELECT codigo,Nombre,Altura
FROM jugadores WHERE Nombre_equipo='Lakers'
    and Procedencia='Spain';
```

codigo Nombre Altura
+-----+-----+-----+
66 Pau Gasol 7-0
+-----+-----+-----+

#Consulta que selecciona los jugadores españoles y eslovenos de los lakers

```
SELECT Nombre, Altura,Procedencia FROM jugadores
WHERE Nombre_equipo='Lakers'
    AND (Procedencia='Spain' OR Procedencia='Slovenia');
```

Nombre Altura Procedencia
+-----+-----+-----+
Pau Gasol 7-0 Spain
Sasha Vujacic 6-7 Slovenia
+-----+-----+-----+

4.4.3. Filtros con operador de pertenencia a conjuntos

Además de los operadores presentados anteriormente (aritméticos, lógicos, etc.) se puede hacer uso del operador de pertenencia a conjuntos IN, cuya sintaxis es la siguiente:

nombre_columna IN (Value1, Value2, ...)

Este operador permite comprobar si una columna tiene un valor igual que cualquier de los que están incluidos dentro del paréntesis, así por ejemplo, si se desea seleccionar los jugadores españoles, eslovenos o serbios de los Lakers, se codificaría así:

```
# versión larga
SELECT Nombre, Altura,Procedencia
FROM jugadores WHERE Nombre_equipo='Lakers' AND
    (Procedencia='Spain'
        OR Procedencia='Slovenia',
        OR Procedencia='Serbia & Montenegro');
```

```
+-----+-----+
| Nombre          | Altura | Procedencia      |
+-----+-----+
| Pau Gasol       | 7-0    | Spain             |
| Vladimir Radmanovic | 6-10   | Serbia & Montenegro |
| Sasha Vujacic  | 6-7    | Slovenia          |
+-----+-----+-----+
```

#versión corta (con el operador IN)

```
SELECT Nombre, Altura,Procedencia FROM jugadores
WHERE Nombre_equipo='Lakers' AND
Procedencia IN ('Spain','Slovenia','Serbia & Montenegro');
```

```
+-----+-----+
| Nombre          | Altura | Procedencia      |
+-----+-----+
| Pau Gasol       | 7-0    | Spain             |
| Vladimir Radmanovic | 6-10   | Serbia & Montenegro |
| Sasha Vujacic  | 6-7    | Slovenia          |
+-----+-----+-----+
```

4.4.4. Filtros con operador de rango

El operador de rango BETWEEN permite seleccionar los registros que estén incluidos en un rango. Su sintaxis es:

```
nombre_columna BETWEEN Value1 AND Value2
```

Por ejemplo, para seleccionar los jugadores de la nba cuyo peso esté entre 270 y 300 libras se codificaría la siguiente query:

```
SELECT Nombre,Nombre_equipo,Peso FROM jugadores
WHERE Peso BETWEEN 270 AND 300;
```

```
+-----+-----+
| Nombre          | Nombre_equipo | Peso |
+-----+-----+
| Chris Richard   | Timberwolves  | 270 |
| Paul Davis      | Clippers      | 275 |
| ....            | ....          | ... |
| David Harrison  | Pacers        | 280 |
+-----+-----+
```

#sería equivalente a

```
SELECT Nombre,Nombre_equipo FROM jugadores
WHERE Peso >= 270 AND Peso <=300;
```

- ◊ **Actividad 4.2:** Saca el peso en kilogramos de los jugadores de la NBA que pesen entre 120 y 150 kilos. Una libra equivale a 0.4535 kilos.

4.4.5. Filtros con test de valor nulo

Los operadores IS e IS NOT permiten verificar si un campo es o no es nulo respectivamente. De esta manera, es posible comprobar, por ejemplo, los jugadores cuya procedencia es desconocida:

```
SELECT nombre,Nombre_equipo
FROM jugadores WHERE Procedencia IS null;
+-----+-----+
| nombre | Nombre_equipo |
+-----+-----+
| Anthony Carter | Nuggets |
+-----+-----+

#la query contraria saca el resto de jugadores
SELECT nombre,Nombre_equipo
FROM jugadores WHERE Procedencia IS NOT null;
+-----+-----+
| nombre | Nombre_equipo |
+-----+-----+
| Corey Brever | Timberwolves |
| Greg Buckner | Timberwolves |
| Michael Doleac | Timberwolves |
.....
| C.J. Watson | Warriors |
| Brandan Wright | Warriors |
+-----+-----+
```

4.4.6. Filtros con test de patrón

Los filtros con test patrón seleccionan los registros que cumplan una serie de características. Se pueden usar los caracteres comodines % y _ para buscar una cadena de caracteres. Por ejemplo, seleccionar de la tabla de vehículos aquellos vehículos cuyo modelo sea 'tdi':

```
SELECT * FROM vehiculos where modelo like '%tdi%';
+-----+-----+
```

matricula	modelo	marca
1132GHT	leon tdi 105cv	seat
3447BYD	a3 tdi 130cv	audi

El carácter comodín **%** busca coincidencias de cualquier número de caracteres, incluso cero caracteres. El carácter comodín **_** busca coincidencias de exactamente un carácter.

Para ilustrar el funcionamiento del carácter **_**, se consultan aquellos equipos que empiecen por R, que terminen por S y que tengan 7 caracteres.

```
SELECT Nombre,Conferencia
FROM equipos WHERE Nombre like 'R_____s';
+-----+
| Nombre | Conferencia |
+-----+
| Raptors | East      |
| Rockets | West      |
+-----+
```

Ambos comodines se pueden usar en un mismo filtro, por ejemplo, para sacar aquellos equipos que en su nombre como segunda letra la o, se usaría el patrón:

```
SELECT Nombre,Conferencia
FROM equipos WHERE Nombre like '_o%';
+-----+
| Nombre | Conferencia |
+-----+
| Bobcats | East      |
| Hornets | West      |
| Rockets | West      |
+-----+
```

Se puede poner "NOT LIKE loquesea" para negar la codición también!!!!

4.4.7. Filtros por límite de número de registros

Este tipo de filtros no es estándar y su funcionamiento varía con el SGBD. Consiste en limitar el número de registros devuelto por una consulta. En MySQL, la sintaxis es:

[LIMIT [desplazamiento,] nfilas]

nfilas especifica el número de filas a devolver y *desplazamiento* especifica a partir de qué fila se empieza a contar (desplazamiento).

```
#devuelve las 4 primeras filas
SELECT nombre,Nombre_equipo
FROM jugadores limit 4;
+-----+
| nombre | Nombre_equipo |
+-----+
| Corey Brever | Timberwolves |
| Greg Buckner | Timberwolves |
| Michael Doleac | Timberwolves |
| Randy Foye | Timberwolves |
+-----+
```

"LIMIT" se escribe
SIEMPRE al final!!!

```
#devuelve 3 filas a partir de la sexta
SELECT nombre,Nombre_equipo
FROM jugadores LIMIT 5,3;
+-----+
| nombre | Nombre_equipo |
+-----+
| Marko Jaric | Timberwolves |
| Al Jefferson | Timberwolves |
| Mark Madsen | Timberwolves |
+-----+
```

Oracle limita el número de filas apoyándose en una pseudo columna, de nombre *rownum*:

```
--Saca los 25 primeros jugadores
SELECT *
FROM jugadores
WHERE rownum <= 25;
```

4.5. Ordenación

Para mostrar ordenados un conjunto de registros se utiliza la cláusula *ORDER BY* de la sentencia SELECT.

```
SELECT [DISTINCT] select_expr [,select_expr] ...
[FROM tabla]
[WHERE filtro]
[ORDER BY {nombre_columna | expr | posición} [ASC | DESC] , ...]
```

Esta cláusula permite ordenar el conjunto de resultados de forma ascendente (ASC) o descendente (DESC) por una o varias columnas. Si no se indica ASC o DESC por defecto es ASC. La columna por la que se quiere ordenar se puede expresar por el nombre de la columna, una expresión o bien la posición numérica del campo que se quiere ordenar. Por ejemplo:

```
#estructura de la tabla equipos
DESCRIBE equipos;
+-----+-----+-----+-----+-----+
| Field      | Type       | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| Nombre      | varchar(20) | NO   | PRI  | NULL    |        |
| Ciudad      | varchar(20)  | YES  |       | NULL    |        |
| Conferencia | varchar(4)   | YES  |       | NULL    |        |
| Division    | varchar(9)   | YES  |       | NULL    |        |
+-----+-----+-----+-----+-----+
#obtener los equipos de la conferencia oeste de la nba ordenados por división
SELECT Nombre,Division
FROM equipos WHERE Conferencia='West',
ORDER BY Division ASC;
+-----+-----+
| Nombre      | Division   |
+-----+-----+
| Jazz        | NorthWest  |
| Nuggets     | NorthWest  |
| Trail Blazers | NorthWest |
| Timberwolves | NorthWest |
| Supersonics | NorthWest |
| Clippers    | Pacific    |
| Kings       | Pacific    |
| Warriors    | Pacific    |
| Suns        | Pacific    |
| Lakers      | Pacific    |
| Hornets     | SouthWest  |
| Spurs        | SouthWest  |
| Rockets      | SouthWest  |
| Mavericks   | SouthWest  |
| Grizzlies   | SouthWest  |
+-----+-----+
```

Si no se especifica,
por defecto es ASC!

```
#se puede ordenar por varios campos, p.ej: además de que cada
#división esté ordenada ascendente se ordene por nombre
#de equipo
SELECT Division,Nombre FROM equipos
WHERE Conferencia='West'
ORDER BY Division ASC,Nombre DESC;
+-----+-----+
| Division | Nombre   |
+-----+-----+
| NorthWest | Trail Blazers |
| NorthWest | Timberwolves |
| NorthWest | Supersonics |
| NorthWest | Nuggets |
| NorthWest | Jazz |
| Pacific   | Warriors |
| Pacific   | Suns |
| Pacific   | Lakers |
| Pacific   | Kings |
| Pacific   | Clippers |
| SouthWest  | Spurs |
| SouthWest  | Rockets |
| SouthWest  | Mavericks |
| SouthWest  | Hornets |
| SouthWest  | Grizzlies |
+-----+-----+
```

4.6. Consultas de resumen

En SQL se pueden generar consultas más complejas que resuman cierta información, extrayendo información calculada de varios conjuntos de registros. Un ejemplo de consulta resumen sería la siguiente:

```
SELECT count(*) FROM vehiculos;
+-----+ 
| count(*) |
+-----+
|      5 |
+-----+
```

Esta consulta devuelve el número de registros de la tabla vehículos, es decir, se genera un resumen de la información contenida en la tabla vehículos. La expresión `count(*)` es una función que toma como entrada los registros de la tabla consultada

y cuenta cuántos registros hay. El resultado de la función count es un único valor (1 fila, 1 columna) con el número 5 (número de registros de la tabla vehículos).

Para poder generar información resumida hay que hacer uso de las *funciones de columna*. Estas funciones de columna convierten un conjunto de registros en una información simple cuyo resultado es un cálculo. A continuación, se expone una lista de las funciones de columna disponibles en SQL:

SUM (Expresión)	#Suma los valores indicados en el argumento
AVG (Expresión)	#Calcula la media de los valores
MIN (Expresión)	#Calcula el mínimo
MAX (Expresión)	#Calcula el máximo
COUNT (nbColumna)	#Cuenta el número de valores de una columna #(excepto los nulos)
COUNT (*)	#Cuenta el número de valores de una fila #Incluyendo los nulos.

A modo de ejemplo, se muestran algunas consultas resúmenes:

```
#consulta 1
#¿Cuánto pesa el jugador más pesado de la nba?
SELECT max(peso) FROM jugadores;

#consulta 2
#¿Cuánto mide el jugador más bajito de la nba?
SELECT min(altura) FROM jugadores;

#consulta 3
#¿Cuántos jugadores tienen los Lakers?
SELECT count(*) FROM jugadores WHERE Nombre_equipo='Lakers';

#consulta 4
#¿Cuánto pesan de media los jugadores de los Blazers?
SELECT avg(peso) FROM jugadores WHERE Nombre_equipo='Blazers';
```

Con las consultas de resumen se pueden realizar *agrupaciones* de registros. Se denomina agrupación de registros a un conjunto de registros que cumplen que tienen una o varias columnas con el mismo valor. Por ejemplo, en la tabla vehículos:

```
SELECT * FROM vehiculos;
```

matricula	modelo	marca
1129FGT	ibiza gt	seat
1132GHT	leon tdi 105cv	seat
M6836YX	corolla g6	toyota
7423FZY	coupe	hyundai
3447BYD	a3 tdi 130cv	audi

En esta consulta hay dos registros cuya marca='seat'. Se puede agrupar estos dos registros formando un único grupo, de tal manera que el grupo 'seat' tiene los modelos ibiza gt (1129FGT) y leon tdi 105cv (1132GHT). A este grupo de registros se le puede aplicar una función de columna para realizar determinados cálculos, por ejemplo, contarlos:

```
SELECT marca, count(*) FROM
vehiculos GROUP BY marca;
+-----+
| marca | count(*) |
+-----+
| audi  |      1 |
| hyundai |     1 |
| seat   |      2 |
| toyota |      1 |
+-----+
```

En este caso, si se agrupa (GROUP BY) por el campo marca, salen 4 grupos (audi, hyundai, seat y toyota). La función de columna, cuando se agrupa por un campo, actúa para cada grupo. En este caso, para cada grupo se ha contado el número de registros que tiene. En el caso de seat, cuenta los 2 antes mencionados.

La sintaxis para la sentencia SELECT con GROUP BY queda como sigue:

```
SELECT [DISTINCT] select_expr [,select_expr] ...
[FROM tabla]
[WHERE filtro]
[GROUP BY expr [, expr].... ]
[ORDER BY {nombre_columna | expr | posición} [ASC | DESC] , ...]
```

Se observa que GROUP BY va justo antes de la cláusula ORDER BY. A continuación, a modo de ejemplo, se muestran algunas consultas con grupos y funciones de columna.

```
#consulta 1
#¿Cuánto pesa el jugador más pesado de cada equipo?
SELECT Nombre_equipo, max(peso)
FROM jugadores GROUP BY Nombre_equipo;
+-----+
| Nombre_equipo | max(peso) |
+-----+
| 76ers         |      250 |
| Bobcats       |      266 |
| Bucks          |      260 |
.....
| Trail Blazers |      255 |
| Warriors       |      250 |
| Wizards         |      263 |
+-----+



#consulta 2
#¿Cuántos equipos tiene cada conferencia en la nba?
SELECT count(*),conferencia FROM equipos GROUP BY conferencia;
+-----+
| count(*) | conferencia |
+-----+
|      15 | East        |
|      15 | West        |
+-----+



#query 3
#¿Cuánto pesan de media los jugadores de españa, francia e italia?
SELECT avg(peso),procedencia FROM jugadores
WHERE procedencia IN ('Spain','Italy','France') GROUP BY procedencia;
+-----+
| avg(peso) | procedencia |
+-----+
| 218.4000 | France      |
| 221.0000 | Italy       |
| 208.6000 | Spain       |
+-----+
```

IMPORTANTE: Se observa que para cada agrupación, se ha seleccionado también el nombre de la columna por la cual se agrupa. Esto no es posible si no se incluye el GROUP BY. Por ejemplo:

Algunas consultas devuelven resultados incorrectos si no se incluye el GROUP BY. Por ejemplo, la consulta siguiente devolvería todos los jugadores de la NBA con un peso de 250 kg.

```
mysql> SELECT count(*),conferencia FROM equipos;
ERROR 1140 (42000): Mixing of GROUP columns
(MIN(),MAX(),COUNT(),...) with no GROUP columns is illegal
if there is no GROUP BY clause
```

Precisamente, el SGBD advierte de que para mezclar funciones de columna y columnas de una tabla hay que escribir una cláusula GROUP BY.

4.6.1. Filtros de Grupos

Los filtros de grupos deben realizarse mediante el uso de la cláusula HAVING puesto que WHERE actúa antes de agrupar los registros. Es decir, si se desea filtrar resultados calculados mediante agrupaciones se debe usar la siguiente sintaxis:

```
SELECT [DISTINCT] select_expr [,select_expr] ...
[FROM tabla]
[WHERE filtro]
[GROUP BY expr [, expr].... ]
[HAVING filtro_grupos]
[ORDER BY {nombre_columna | expr | posición} [ASC | DESC] , ...]
```

HAVING aplica los mismos filtros que la cláusula WHERE. A continuación se ilustran algunos ejemplos:

```
#query 1:
#Seleccionar los equipos de la nba cuyos jugadores
#pesan de media más de 228 libras
SELECT Nombre_equipo,avg(peso)
FROM jugadores
GROUP BY Nombre_equipo
HAVING avg(peso)>228 ORDER BY avg(peso);
```

Nombre_equipo	avg(peso)
Suns	228.8462
Wizards	229.6923
Lakers	230.0000
Jazz	230.0714
Knicks	235.4667

```
#query 2
#seleccionar qué equipos de la nba tienen más de 1 jugador español
SELECT Nombre_equipo,count(*)
    FROM jugadores
    WHERE procedencia='Spain'
    GROUP BY Nombre_equipo
    HAVING count(*)>1;
+-----+-----+
| Nombre_equipo | count(*) |
+-----+-----+
| Raptors       |      2 |
+-----+-----+
```

4.7. Subconsultas

Las subconsultas se utilizan para realizar filtrados con los datos de otra consulta. Estos filtros pueden ser aplicados tanto en la cláusula WHERE para filtrar registros como en la cláusula HAVING para filtrar grupos. Por ejemplo, con la base de datos de la NBA, es posible codificar una consulta para pedir los nombres de los jugadores de la división 'SouthEast':

```
SELECT nombre FROM jugadores
WHERE Nombre_equipo IN
(SELECT Nombre FROM equipos WHERE division='SouthWest');
+-----+
| nombre          |
+-----+
| Andre Brown    |
| Kwame Brown    |
| Brian Cardinal |
| Jason Collins   |
| ...             |
+-----+
```

Se observa que la subconsulta es precisamente la sentencia SELECT encerrada entre paréntesis. De esta forma, se hace uso del operador *in* para tomar los equipos de la división 'SouthEast'. Si se ejecuta la subconsulta por separado se obtiene:

```
SELECT Nombre FROM equipos
WHERE division='SouthWest';
```

Nombre
Hornets
Spurs
Rockets
Mavericks
Grizzlies

La subconsulta se convierte en algo equivalente a:

```
SELECT nombre FROM jugadores
WHERE Nombre_equipo IN
('Hornets','Spurs','Rockets',
'Mavericks','Grizzlies')
```

En las siguientes secciones se detallan los posibles operadores que se pueden usar con las subconsultas.

4.7.1. Test de Comparación

Consiste en usar los operadores de comparación =, >=, <=, <>, >y < para comparar el valor producido con un valor único generado por una subconsulta. Por ejemplo, para consultar el nombre del jugador de mayor altura de la nba, es posible hacer algo como esto:

```
SELECT nombre FROM jugadores
WHERE altura =
  (SELECT max(altura) FROM jugadores);
+-----+
| nombre |
+-----+
| Yao Ming |
```

Se puede comprobar que la subconsulta produce un único resultado, utilizándolo para filtrar.

Nótese que con este tipo de filtro la subconsulta solo debe producir un único valor (una fila y una columna), por tanto, si se codifica algo del tipo:

```
SELECT nombre FROM jugadores
WHERE altura = (SELECT max(altura),max(peso) FROM jugadores);
ERROR 1241 (21000): Operand should contain 1 column(s)
```

También fallaría que la subconsulta devolviera más de una fila:

```
SELECT nombre FROM jugadores  
WHERE altura = (SELECT max(altura)  
FROM jugadores GROUP BY Nombre_Equipo);  
ERROR 1242 (21000): Subquery returns more than 1 row
```

Una restricción importante es que la subconsulta debe estar siempre al lado derecho del operador de comparación. Es decir:

Campo <= subconsulta

siendo inválida la expresión:

subconsulta >= Campo

4.7.2. Test de pertenencia a conjunto

Este test consiste en una variante del usado para consultas simples, y es el que se ha utilizado para ilustrar el primer ejemplo de la sección. Consiste en usar el operador IN para filtrar los registros cuya expresión coincida con algún valor producido por la subconsulta.

Por ejemplo, para extraer las divisiones de la nba donde juegan jugadores españoles:

```
SELECT division FROM equipos WHERE nombre in  
(SELECT Nombre_equipo FROM jugadores WHERE procedencia='Spain');  
+-----+  
| division |  
+-----+  
| Atlantic |  
| NorthWest |  
| Pacific |  
| SouthWest |  
+-----+
```

4.7.3. Test de existencia

El test de existencia permite filtrar los resultados de una consulta si existen filas en la subconsulta asociada, esto es, si la subconsulta genera un número de filas distinto de 0.

Para usar el test de existencia se utiliza el operador EXISTS:

```
SELECT columnas FROM tabla
WHERE EXISTS (subconsulta)
```

El operador **EXISTS** también puede ser precedido de la negación (**NOT**) para filtrar si no existen resultados en la subconsulta:

```
SELECT columnas FROM tabla
WHERE NOT EXISTS (subconsulta)
```

Para seleccionar los equipos que no tengan jugadores españoles se podría usar la siguiente consulta:

```
SELECT Nombre FROM equipos WHERE NOT EXISTS
    (SELECT Nombre FROM jugadores
     WHERE equipos.Nombre = jugadores.Nombre_Equipo
     AND procedencia='Spain');

+-----+
| Nombre      |
+-----+
| 76ers       |
| Bobcats    |
| Bucks       |
| ...         |
+-----+
```

Para comprender la lógica de esta query, se puede asumir que cada registro devuelto por la consulta principal provoca la ejecución de la subconsulta, así, si la consulta principal (SELECT Nombre FROM Equipos) devuelve 30 registros, se entenderá que se ejecutan 30 subconsultas, una por cada nombre de equipo que retorne la consulta principal. Esto en realidad no es así, puesto que el SGBD optimiza la consulta para hacer tan solo dos consultas y una operación *join* que se estudiará más adelante, pero sirve de ejemplo ilustrativo del funcionamiento de esta consulta:

```
SELECT Nombre from equipos;
+-----+
| Nombre      |
+-----+
| 76ers       | -> subconsulta ejecutada #1
| Bobcats    | -> subconsulta ejecutada #2
| ...         | ...
| Raptors    | -> subconsulta ejecutada #22
| ...         | -> ....
+-----+
```

Cada subconsulta ejecutada sería como sigue:

```
#subconsulta ejecutada #1
SELECT Nombre FROM jugadores
WHERE '76ers' = jugadores.Nombre_Equipo
AND procedencia='Spain';
```

Esta subconsulta no retorna resultados, por tanto, el equipo '76ers' es seleccionado para ser devuelto en la consulta principal, puesto que no existen (NOT EXISTS) jugadores españoles.

Sin embargo para el registro 22 de la consulta principal, aquel cuyo Nombre es 'Raptors', la consulta:

```
#subconsulta ejecutada #22
SELECT Nombre FROM jugadores
WHERE 'Raptors' = jugadores.Nombre_Equipo
AND procedencia='Spain';
```

Nombre
Jose Calderon
Jorge Garbajosa

devuelve 2 jugadores, por tanto, existen (EXISTS) registros de la subconsulta y por tanto el equipo 'Raptors' NO es seleccionado por la consulta principal.

En conclusión, se puede decir que la consulta principal *enlaza* los registros con los devueltos por las subconsultas.

4.7.4. Test cuantificados ALL y ANY

Los test cuantificados sirven para calcular la relación entre una expresión y todos los registros de la subconsulta (ALL) o algunos de los registros de la subconsulta (ANY).

De esta manera se podría saber los jugadores de la nba que pesan más que todos los jugadores españoles:

```

SELECT nombre,peso from jugadores
WHERE peso > ALL
(SELECT peso FROM jugadores WHERE procedencia='Spain');
+-----+-----+
| nombre | peso |
+-----+-----+
| Michael Doleac | 262 |
| Al Jefferson | 265 |
| Chris Richard | 270 |
| ... | ... |
+-----+-----+

```

Al igual que en el caso del operador `exists`, se puede asumir que por cada registro de la consulta principal se ejecuta una subconsulta. En tal caso, para el jugador 'Michael Doleac' cuyo peso es 262 libras, se comprobaría si 262 es mayor que los pesos de todos los jugadores españoles, que son devueltos por la subconsulta (`SELECT peso FROM jugadores WHERE procedencia='Spain'`).

También se podría consultar los bases (en inglés Guard '`G`') `posicion='G'`, que pesan más que cualquier (ANY) pivot (en inglés Center '`C`') `posicion='C'` de la nba:

```

SELECT nombre,peso from jugadores
WHERE posicion='G' AND
peso > ANY
(SELECT peso FROM jugadores where posicion='C');
+-----+-----+
| nombre | peso |
+-----+-----+
| Joe Johnson | 235 |
+-----+-----+

```

Se comprueba que en el caso de 'Joe Johnson', su peso (235 libras), es mayor que algún peso de algún pivot de la nba, dándose así el peculiar caso de un base más pesado que algún pivot.

4.7.5. Subconsultas anidadas

Se puede usar una subconsulta para filtrar los resultados de otra subconsulta. De esta manera se *anidan* subconsultas. Por ejemplo, si se desea obtener el nombre de la ciudad donde juega el jugador más alto de la nba, habría que pensar cómo hacerlo de forma estructurada:

1. Obtener la altura del jugador más alto:

```
X <- (SELECT max(altura) from jugadores)
```

2. Obtener el nombre del jugador, a través de la altura se localiza al jugador y por tanto, su equipo:

```
Y <- SELECT Nombre_equipo from jugadores WHERE Altura = X
```

3. Obtener la ciudad:

```
SELECT ciudad FROM equipos WHERE nombre= Y
```

Ordenando todo esto, se puede construir la consulta de abajo a arriba:

```
SELECT ciudad FROM equipos WHERE nombre =
  (SELECT Nombre_equipo FROM jugadores WHERE altura =
    (SELECT MAX(altura) FROM jugadores));
+-----+
| ciudad |
+-----+
| Houston |
+-----+
```

Esta manera de generar consultas es muy sencilla, y a la vez permite explorar la información de la base de datos de forma *estructurada*. En opinión de muchos autores esta forma estructurada de generar consultas es la que dio a SQL su 'S' de *Structured*.

4.8. Consultas multitabla

Una consulta multitabla es aquella en la que se puede consultar información de más de una tabla. Se aprovechan los campos relacionados de las tablas para *unirlas* (join). Para poder realizar este tipo de consultas hay que utilizar la siguiente sintaxis:

```
SELECT [DISTINCT] select_expr [,select_expr] ...
[FROM referencias_tablas]
[WHERE filtro]
[GROUP BY expr [, expr].... ]
[HAVING filtro_grupos]
[ORDER BY {nombre_columnas | expr | posición} [ASC | DESC] , ...]
```

La diferencia con las consultas sencillas se halla en la cláusula FROM. Esta vez en lugar de una tabla se puede desarrollar el token referencias_tablas:

```
referencias_tablas:  
referencia_tabla[, referencia_tabla] ...  
| referencia_tabla [INNER | CROSS] JOIN referencia_tabla [ON condición]  
| referencia_tabla LEFT [OUTER] JOIN referencia_tabla ON condición  
| referencia_tabla RIGHT [OUTER] JOIN referencia_tabla ON condición  
referencia_tabla:  
nombre_tabla [[AS] alias]
```



La primera opción, (referencia_tabla[, referencia_tabla] ...) es típica de SQL 1 (SQL-86) para las **uniones**, que consisten en un producto cartesiano más un filtro por las columnas relacionadas, y el resto de opciones son propias de SQL 2 (SQL-92 y SQL-2003).

4.8.1. Consultas multitabla SQL 1



El producto cartesiano de dos tablas son todas las combinaciones de las filas de una tabla unidas a las filas de la otra tabla. Por ejemplo, una base de datos de mascotas con dos tablas animales y propietarios:

```
SELECT * FROM propietarios;
```

dni	nombre
51993482Y	José Pérez
2883477X	Matías Fernández
37276317Z	Francisco Martínez

```
SELECT * FROM animales;
```

codigo	nombre	tipo	propietario
1	Cloncho	gato	51993482Y
2	Yoda	gato	51993482Y
3	Sprocket	perro	37276317Z

Un producto cartesiano de las dos tablas se realiza con la siguiente sentencia:

```
SELECT * FROM animales,propietarios;
```

codigo	nombre	tipo	propietario	dni	nombre
1	Cloncho	gato	51993482Y	51993482Y	José Pérez
2	Yoda	gato	51993482Y	51993482Y	José Pérez
3	Sprocket	perro	37276317Z	51993482Y	José Pérez
1	Cloncho	gato	51993482Y	2883477X	Matías Fernández
2	Yoda	gato	51993482Y	2883477X	Matías Fernández
3	Sprocket	perro	37276317Z	2883477X	Matías Fernández
1	Cloncho	gato	51993482Y	37276317Z	Francisco Martínez
2	Yoda	gato	51993482Y	37276317Z	Francisco Martínez
3	Sprocket	perro	37276317Z	37276317Z	Francisco Martínez

La operación genera un conjunto de resultados con todas las combinaciones posibles entre las filas de las dos tablas, y con todas las columnas. Aparentemente esto no tiene mucha utilidad, sin embargo, si se aplica un filtro al producto cartesiano, es decir, una condición WHERE que escoja solo aquellas filas en las que el campo dni (del propietario) coincide con el propietario (de la mascota), se obtienen los siguientes interesantes resultados:

```
SELECT * FROM animales,propietarios
WHERE propietarios.dni=animales.propietario;
```



codigo	nombre	tipo	propietario	dni	nombre
1	Cloncho	gato	51993482Y	51993482Y	José Pérez
2	Yoda	gato	51993482Y	51993482Y	José Pérez
3	Sprocket	perro	37276317Z	37276317Z	Francisco Martínez

Mediante esta consulta se ha obtenido información relacionada entre las dos tablas. Se aprecia como los dos gatos (Cloncho y Yoda) aparecen con su dueño (José Pérez), y que, Sprocket el perro, aparece con su dueño Francisco Martínez. Esta operación se llama **JOIN** de las tablas Propietarios y Animales, y consiste en realizar un producto cartesiano de ambas y un filtro por el campo relacionado (Clave Foránea vs Clave Primaria).

Por tanto, **JOIN = PRODUCTO CARTESIANO + FILTRO**. En el apartado siguiente se estudiará que existen varios tipos de join y que SQL 2 incluye en su sintaxis formas de parametrizar estos tipos de join.

Este mismo procedimiento se puede aplicar con N tablas, por ejemplo, con la base de datos *jardineria*", cuyo gráfico de relaciones es el siguiente:

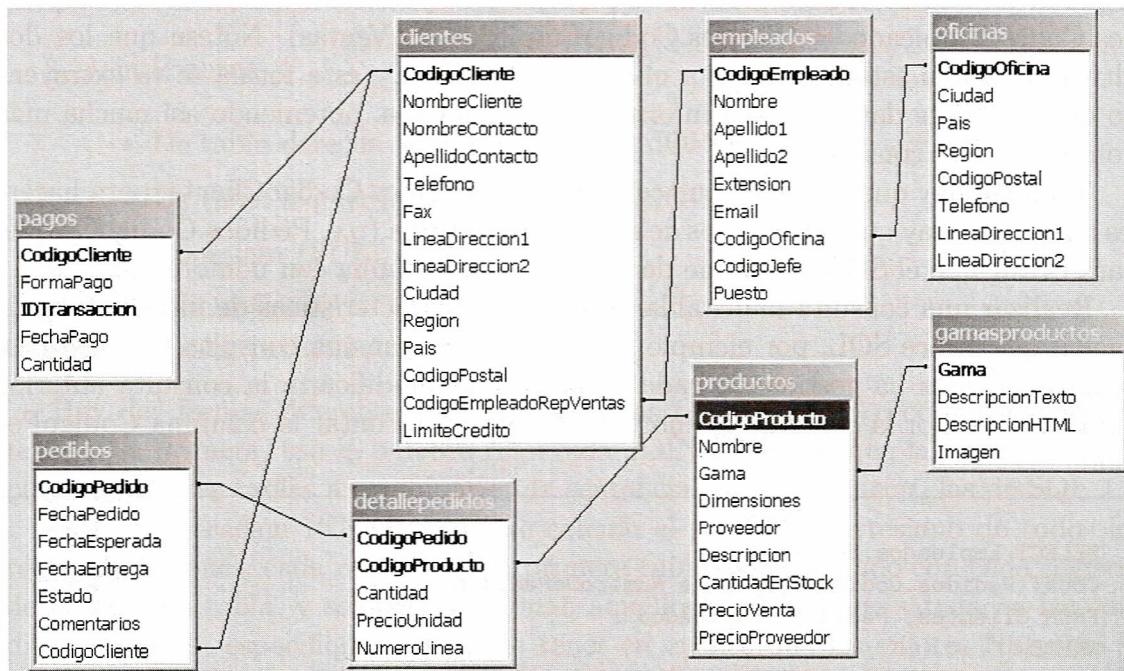


Figura 4.1: Relaciones de la bbdd "jardineria".

se puede generar una consulta para obtener un listado de pedidos gestionados por cada empleado:

```

mysql> SELECT Empleados.Nombre, Clientes.NombreCliente, Pedidos.CodigoPedido
  FROM Clientes, Pedidos, Empleados
 WHERE Clientes.CodigoCliente=Pedidos.CodigoCliente
       AND
         Empleados.CodigoEmpleado = Clientes.CodigoEmpleadoRepVentas
 ORDER BY Empleados.Nombre;
 +-----+-----+-----+
 | Nombre | NombreCliente | CodigoPedido |
 +-----+-----+-----+
 | Emmanuel | Naturagua | 18 |
 | Emmanuel | Beragua | 16 |
 .....
 | Walter Santiago | DGPRODUCTIONS GARDEN | 65 |
 | Walter Santiago | Gardening & Associates | 58 |
 +-----+-----+-----+
  
```

Se observa que en este caso hay dos JOIN, el primer join entre la tabla Clientes y Pedidos, con la condición Clientes.CodigoCliente=Pedidos.CodigoCliente y la segunda join entre el resultado de la primera join y la tabla Empleados (Empleados.CodigoEmpleado = Clientes.CodigoEmpleadoRepVentas). Nótese que los dos filtros de la join están unidas por el operador AND. De esta forma se va extrayendo de la base de datos toda la información relacionada, obteniendo así mucha más potencia en las consultas.

También hay que fijarse en que como hay dos campos CodigoCliente, para hacerles referencia, hay que precederlos de su nombre de tabla (p.e. Pedidos.CodigoCliente) para evitar que el SGBD informe de que hay una columna con nombre ambiguo.

Realizar una consulta multitabla no limita las características de filtrado y agrupación que ofrece SQL, por ejemplo, si se desea realizar una consulta para obtener cuántos pedidos ha gestionado cada empleado, se modificaría la consulta anterior para agrupar por la columna Nombre de Empleado y contar la columna CodigoPedido:

```
SELECT Empleados.Nombre,
       COUNT(Pedidos.CodigoPedido) as NumeroDePedidos
  FROM Clientes, Pedidos, Empleados
 WHERE
       Clientes.CodigoCliente=Pedidos.CodigoCliente
      AND
       Empleados.CodigoEmpleado = Clientes.CodigoEmpleadoRepVentas
 GROUP BY Empleados.Nombre
 ORDER BY NumeroDePedidos;
```

Nombre	NumeroDePedidos
Michael	5
Lorena	10
...	..
Walter Santiago	20

4.8.2. Consultas multitabla SQL 2

SQL 2 introduce otra sintaxis para los siguientes tipos de consultas multitablas: las joins (o composiciones) internas, externas y productos cartesianos (también llamadas composiciones cruzadas):

1. Join Interna:

- De equivalencia (INNER JOIN)

- Natural (NATURAL JOIN)
2. Producto Cartesiano (CROSS JOIN)
 3. Join Externa
 - De tabla derecha (RIGHT OUTER JOIN)
 - De tabla izquierda (LEFT OUTER JOIN)
 - Completa (FULL OUTER JOIN)

Composiciones internas. INNER JOIN

Hay dos formas diferentes para expresar las INNER JOIN o composiciones internas. La primera, usa la palabra reservada JOIN, mientras que la segunda usa ',' para separar las tablas a combinar en la sentencia FROM, es decir, las de SQL 1.

Con la operación INNER JOIN se calcula el producto cartesiano de todos los registros, después, cada registro en la primera tabla es combinado con cada registro de la segunda tabla, y solo se seleccionan aquellos registros que satisfacen las condiciones que se especifiquen. Hay que tener en cuenta que los valores Nulos no se combinan.

Como ejemplo, la siguiente consulta toma todos los registros de la tabla animales y encuentra todas las combinaciones en la tabla propietarios. La JOIN compara los valores de las columnas dni y propietario. Cuando no existe esta correspondencia entre algunas combinaciones, estas no se muestran; es decir, que si el dni de un propietario de una mascota no coincide con algún dni de la tabla de propietarios, no se mostrará el animal con su respectivo propietario en los resultados.

```

 SELECT * FROM animales INNER JOIN propietarios 
  ON animales.propietario = propietarios.dni;
+-----+-----+-----+-----+-----+
| codigo | nombre | tipo   | propietario | dni        | nombre       |
+-----+-----+-----+-----+-----+
|     1  | Cloncho | gato  | 51993482Y  | 51993482Y | José Pérez  |
|     2  | Yoda    | gato  | 51993482Y  | 51993482Y | José Pérez  |
|     3  | Sprocket | perro | 37276317Z  | 37276317Z | Francisco Martínez |
+-----+-----+-----+-----+-----+
# Nótese que es una consulta equivalente a la vista en el apartado anterior
# select * from animales,propietarios
# where animales.propietario=propietarios.dni;

```

Además, debe tenerse en cuenta que si hay un animal sin propietario no saldrá en el conjunto de resultados puesto que no tiene coincidencia en el filtro:

```
INSERT INTO animales VALUES (null,'Arco','perro',null);
SELECT * FROM animales;
+-----+-----+-----+-----+
| codigo | nombre | tipo | propietario |
+-----+-----+-----+-----+
| 1 | Cloncho | gato | 51993482Y |
| 2 | Yoda | gato | 51993482Y |
| 3 | Sprocket | perro | 37276317Z |
| 4 | Arco | perro | NULL | #nueva mascota sin propietario
+-----+-----+-----+-----+
SELECT * FROM animales INNER JOIN propietarios
    ON animales.propietario = propietarios.dni;
+-----+-----+-----+-----+-----+-----+
| codigo | nombre | tipo | propietario | dni | nombre |
+-----+-----+-----+-----+-----+-----+
| 1 | Cloncho | gato | 51993482Y | 51993482Y | José Pérez |
| 2 | Yoda | gato | 51993482Y | 51993482Y | José Pérez |
| 3 | Sprocket | perro | 37276317Z | 37276317Z | Francisco Martínez |
+-----+-----+-----+-----+-----+-----+
#LA NUEVA MASCOTA NO APARECE!
```

Puede hacerse variantes de la inner join cambiando el operador del filtro, por ejemplo:

```
SELECT * FROM animales INNER JOIN propietarios
    ON propietarios.dni >= animales.propietario;
+-----+-----+-----+-----+-----+-----+
| codigo | nombre | tipo | propietario | dni | nombre |
+-----+-----+-----+-----+-----+-----+
| 1 | Cloncho | gato | 51993482Y | 51993482Y | José Pérez |
| 2 | Yoda | gato | 51993482Y | 51993482Y | José Pérez |
| 3 | Sprocket | perro | 37276317Z | 51993482Y | José Pérez |
| 3 | Sprocket | perro | 37276317Z | 37276317Z | Francisco Martínez |
+-----+-----+-----+-----+-----+-----+
```

Composiciones naturales. NATURAL JOIN

Es una especialización de la INNER JOIN. En este caso se comparan todas las columnas que tengan el mismo nombre en ambas tablas. La tabla resultante contiene solo una columna por cada par de columnas con el mismo nombre.

```
DESCRIBE Empleados;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| CodigoEmpleado | int(11) | NO | PRI | NULL | 
| Nombre | varchar(50) | NO | | NULL | 
| Apellido1 | varchar(50) | NO | | NULL | 
| Apellido2 | varchar(50) | YES | | NULL | 
| Extension | varchar(10) | NO | | NULL | 
| Email | varchar(100) | NO | | NULL | 
| CodigoOficina | varchar(10) | NO | | NULL | #relación
| CodigoJefe | int(11) | YES | | NULL | 
| Puesto | varchar(50) | YES | | NULL | 
+-----+-----+-----+-----+-----+
```

```
DESCRIBE Oficinas;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| CodigoOficina | varchar(10) | NO | PRI | NULL | #relación
| Ciudad | varchar(30) | NO | | NULL | 
| Pais | varchar(50) | NO | | NULL | 
| Region | varchar(50) | YES | | NULL | 
| CodigoPostal | varchar(10) | NO | | NULL | 
| Telefono | varchar(20) | NO | | NULL | 
| LineaDireccion1 | varchar(50) | NO | | NULL | 
| LineaDireccion2 | varchar(50) | YES | | NULL | 
+-----+-----+-----+-----+-----+
```

#NATURAL JOIN coge los **mismos nombres de campo**, en este caso **CodigoOficina**

```
SELECT CodigoEmpleado,Empleados.Nombre,
Oficinas.CodigoOficina,Oficinas.Ciudad
FROM Empleados NATURAL JOIN Oficinas; 
+-----+-----+-----+
| CodigoEmpleado | Nombre | CodigoOficina | Ciudad |
+-----+-----+-----+
| 1 | Marcos | TAL-ES | Talavera de la Reina |
| 2 | Ruben | TAL-ES | Talavera de la Reina |
| ... |
| 31 | Mariko | SYD-AU | Sydney |
+-----+-----+-----+
```

Hay que fijarse en que, aunque CodigoEmpleado es un campo que está en dos tablas, esta vez no es necesario precederlo del nombre de tabla puesto que NATURAL JOIN devuelve un único campo por cada pareja de campos con el mismo nombre.

Producto cartesiano.CROSS JOIN

Este tipo de sintaxis devuelve el producto cartesiano de dos tablas:

```
#equivalente a SELECT * FROM animales,propietarios;
SELECT * FROM animales CROSS JOIN propietarios;
```

codigo	nombre	tipo	proprietario	dni	nombre
1	Cloncho	gato	51993482Y	51993482Y	José Pérez
1	Cloncho	gato	51993482Y	2883477X	Matías Fernández
1	Cloncho	gato	51993482Y	37276317Z	Francisco Martínez
2	Yoda	gato	51993482Y	51993482Y	José Pérez
2	Yoda	gato	51993482Y	2883477X	Matías Fernández
2	Yoda	gato	51993482Y	37276317Z	Francisco Martínez
3	Sprocket	perro	37276317Z	51993482Y	José Pérez
3	Sprocket	perro	37276317Z	2883477X	Matías Fernández
3	Sprocket	perro	37276317Z	37276317Z	Francisco Martínez
4	Arco	perro	NULL	51993482Y	José Pérez
4	Arco	perro	NULL	2883477X	Matías Fernández
4	Arco	perro	NULL	37276317Z	Francisco Martínez

Nótese que aparece también el nuevo animal insertado sin propietario (Arco).

Composiciones externas.OUTER JOIN

En este tipo de operación, las tablas relacionadas no requieren que haya una equivalencia. El registro es seleccionado para ser mostrado aunque no haya otro registro que le corresponda.

OUTER JOIN se subdivide dependiendo de la tabla a la cual se le admitirán los registros que no tienen correspondencia, ya sean de tabla izquierda, de tabla derecha, o combinación completa.

Si los registros que admiten no tener correspondencia son los que aparecen en la tabla de la izquierda se llama composición de tabla izquierda o LEFT JOIN (o LEFT OUTER JOIN):

```
#ejemplo de LEFT OUTER JOIN
#animales LEFT OUTER JOIN propietarios
#animales está a la izquierda
#propietarios está a la derecha
```

```
SELECT * FROM animales LEFT OUTER JOIN propietarios
  ON animales.propietario = propietarios.dni;
```

codigo	nombre	tipo	propietario	dni	nombre
1	Cloncho	gato	51993482Y	51993482Y	José Pérez
2	Yoda	gato	51993482Y	51993482Y	José Pérez
3	Sprocket	perro	37276317Z	37276317Z	Francisco Martínez
4	Arco	perro	NULL	NULL	NULL

Se observa que se incluye el perro Arco que no tiene propietario, por tanto, sus campos relacionados aparecen con valor NULL. El sentido de esta query podría ser, sacar todos los animales y si tienen relación, sacar sus propietarios, y si no tiene propietario, indicarlo con un valor NULO o con VACÍO.

¿Sabías que ... ? Oracle implementaba las consultas externas antes de la aparición de las OUTER JOIN, utilizando el operador (+)= en lugar del operador = en la cláusula WHERE. Esta sintaxis aún está disponible en las nuevas versiones de este SGBD.

Si los registros que admiten no tener correspondencia son los que aparecen en la tabla de la derecha, se llama composición de tabla derecha RIGHT JOIN (o RIGHT OUTER JOIN):

```
#ejemplo de RIGHT OUTER JOIN
#animales RIGHT OUTER JOIN propietarios
#animales está a la izquierda
#propietarios está a la derecha
SELECT * FROM animales RIGHT OUTER JOIN propietarios
  ON animales.propietario = propietarios.dni;
```

codigo	nombre	tipo	propietario	dni	nombre
1	Cloncho	gato	51993482Y	51993482Y	José Pérez
2	Yoda	gato	51993482Y	51993482Y	José Pérez
NULL	NULL	NULL	NULL	2883477X	Matías Fernández
3	Sprocket	perro	37276317Z	37276317Z	Francisco Martínez

En este caso, los que aparecen son todos los propietarios, incluido Matías Fernández que no tiene una mascota. Se ve que el perro Arco no aparece, pues esta vez los registros que se desean mostrar son todos los de la tabla derecha (es decir, propietarios).

La operación que admite registros sin correspondencia tanto para la tabla izquierda como para la derecha, por ejemplo, animales sin propietario y propietarios sin animales, se llama composición externa completa o **FULL JOIN (FULL OUTER JOIN)**. Esta operación presenta los resultados de tabla izquierda y tabla derecha aunque no tengan correspondencia en la otra tabla. La tabla combinada contendrá, entonces, todos los registros de ambas tablas y presentará valores nulos para registros sin pareja.

```
#ejemplo de FULL OUTER JOIN
#animales FULL OUTER JOIN propietarios
#animales está a la izquierda
#propietarios está a la derecha
SELECT * FROM animales FULL OUTER JOIN propietarios
    ON animales.propietario = propietarios.dni;
+-----+-----+-----+-----+-----+
| codigo | nombre | tipo   | propietario | dni        | nombre      |
+-----+-----+-----+-----+-----+
|     1  | Cloncho | gato  | 51993482Y  | 51993482Y  | José Pérez  |
|     2  | Yoda    | gato  | 51993482Y  | 51993482Y  | José Pérez  |
|     3  | Sprocket | perro | 37276317Z  | 37276317Z  | Francisco Martínez |
|     4  | Arco    | perro | NULL       | NULL       | NULL        |
|    NULL | NULL    | NULL   | NULL       | 2883477X  | Matías Fernández |
+-----+-----+-----+-----+-----+
```

¿Sabías que . . . ? En SQL existe el operador UNION, que añade al conjunto de resultados producidos por una SELECT, los resultados de otra SELECT.

La sintaxis es:

```
SELECT .... FROM ....
UNION [ALL]
SELECT .... FROM ....
```

El parámetro ALL incluye todos los registros de las dos SELECT, incluyendo los que son iguales. Si no se indica ALL, se excluyen los duplicados.

Aunque MySQL no implementa la característica FULL OUTER JOIN, sí que se puede simular haciendo una unión de los resultados de un LEFT OUTER JOIN y

los resultados de un RIGHT OUTER JOIN, puesto que UNION, sin la opción ALL, elimina los registros duplicados, por tanto, se podría codificar la FULL OUTER JOIN anterior de la siguiente forma:

```
mysql> SELECT * FROM animales LEFT OUTER JOIN propietarios
    ->     ON animales.propietario = propietarios.dni
-> UNION
-> SELECT * FROM animales RIGHT OUTER JOIN propietarios
->     ON animales.propietario = propietarios.dni;
+-----+-----+-----+-----+-----+
| codigo | nombre | tipo | propietario | dni      | nombre   |
+-----+-----+-----+-----+-----+
| 1      | Cloncho | gato | 51993482Y | 51993482Y | José Pérez |
| 2      | Yoda     | gato | 51993482Y | 51993482Y | José Pérez |
| 3      | Sprocket | perro | 37276317Z | 37276317Z | Francisco Martínez |
| 4      | Arco     | perro | NULL       | NULL      | NULL      |
| NULL   | NULL     | NULL | NULL       | 2883477X | Matías Fernández |
+-----+-----+-----+-----+-----+
```

4.9. Consultas reflexivas

A veces, es necesario obtener información de relaciones reflexivas, por ejemplo, un informe de empleados donde junto a su nombre y apellidos apareciera el nombre y apellidos de su jefe. Para ello, es necesario hacer una JOIN entre registros de la misma tabla:

```
mysql> desc Empleados;
+-----+-----+-----+-----+-----+
| Field        | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| CódigoEmpleado | int(11) | NO  | PRI | NULL    |        |
| Nombre        | varchar(50)| NO  |     | NULL    |        |
| Apellido1     | varchar(50)| NO  |     | NULL    |        |
| Apellido2     | varchar(50)| YES |     | NULL    |        |
| Extension     | varchar(10) | NO  |     | NULL    |        |
| Email         | varchar(100) | NO  |     | NULL    |        |
| CódigoOficina | varchar(10) | NO  |     | NULL    |        |
| CódigoJefe    | int(11)    | YES |     | NULL    | #autorelación |
| Puesto        | varchar(50) | YES |     | NULL    |        |
+-----+-----+-----+-----+-----+
```

```
SELECT concat(emp.Nombre, ' ', emp.Apellido1) as Empleado,
       concat(jefe.Nombre, ' ', jefe.Apellido1) as jefe
```

```

    FROM Empleados emp INNER JOIN Empleados jefe
    ON emp.CodigoEmpleado=jefe.CodigoJefe;
+-----+-----+
| Empleado | jefe      |
+-----+-----+
| Marcos Magaña | Ruben López |
| Ruben López   | Alberto Soria |
| ...          |            |
| Alberto Soria | Kevin Fallmer |
| Kevin Fallmer | Julian Bellinelli |
| Kevin Fallmer | Mariko Kishi  |
+-----+-----+

```

Analizando la query anterior, primero se observa el uso de la tabla empleados dos veces, una con un alias emp que representa los empleados como subordinados y otra con alias jefe que representa los empleados como jefes. Ambas tablas (aunque en realidad son la misma) se unen en una JOIN a través de la relación CódigoEmpleado y CódigoJefe.

Por otro lado, el primer campo que se selecciona es la concatenación del nombre y apellido del empleado (concat(emp.Nombre, ' ', emp.Apellido1)) al que a su vez le damos un alias (empleado) y el segundo campo que es la concatenación de los empleados jefes, al que le se le da el alias jefe.

Se puede observar que en esta query no aparecen los empleados sin jefe, puesto que se ha utilizado un INNER JOIN. Para mostrarlos, habría que usar un LEFT o RIGHT OUTER JOIN.

4.10. Consultas con tablas derivadas

Las consultas con tablas derivadas, o *inline views*, son aquellas que utilizan sentencias SELECT en la cláusula FROM en lugar de nombres de tablas, por ejemplo:

```

SELECT * FROM
  (SELECT CódigoEmpleado, Nombre FROM Empleados
   WHERE CódigoOficina='TAL-ES') as tabla_derivada;

```

En este caso se ha de distinguir, por un lado la tabla derivada, (SELECT CódigoEmpleado, Nombre FROM Empleados) que tiene un alias tabla_derivada, es decir, una especie de tabla temporal cuyo contenido es el resultado de ejecutar la consulta, su nombre es tabla_derivada y tiene dos columnas, una CódigoEmpleado y otra Nombre. Este tipo de consultas ayudará a obtener información relacionada de forma mucho más avanzada.

Por ejemplo, en la base de datos *jardineria*, si se desea sacar el importe del pedido de menor coste de todos los pedidos, hay que pensar primero como sacar el total de todos los pedidos y de ahí, el pedido con menor coste con la función de columna MIN:

```
#1: Para calcular el total de cada pedido, hay que codificar esta query
SELECT SUM(Cantidad*PrecioUnidad) as total,CodigoPedido
    FROM DetallePedidos
    GROUP BY CodigoPedido;
+-----+-----+
| total | CodigoPedido |
+-----+-----+
| 1567 |           1 |
| 7113 |           2 |
| 10850|           3 |
.....
|   154 |         117 |
|    51 |         128 |
+-----+-----+
```



```
#2: Para calcular el menor pedido, se puede hacer una tabla
# derivada de la consulta anterior y con la función MIN
# obtener el menor de ellos:
SELECT MIN(total) FROM (
    SELECT SUM(Cantidad*PrecioUnidad) as total,CodigoPedido
        FROM DetallePedidos
        GROUP BY CodigoPedido
    ) AS TotalPedidos;
+-----+
| MIN(total) |
+-----+
|      4 |
+-----+
```

#TotalPedidos es la tabla derivada formada
#por el resultado de la consulta entre paréntesis

Las tablas derivadas no tienen limitación, es decir, se pueden unir a otras tablas, filtrar, agrupar, etc.

4.11. Prácticas Resueltas

Práctica 4.1: Consultas simples en MS-Access

Con la BBDD Automóviles, genera sentencias SQL para obtener:

1. El nombre de las marcas y modelos de los vehículos.
2. El nombre de los modelos cuyas emisiones estén entre 150 y 165.
3. El nombre de los modelos cuyas emisiones estén entre 150 y 165 o que su consumo esté entre 5 y 6 ordenado por consumo descendente.
4. Un listado de todas las Marcas que hay (sin repeticiones).

Para crear una consulta en modo SQL en Access, se pulsa en la pestaña “Crear”, opción “Diseño de Consulta”, y a continuación, se pulsa en el botón “SQL”. Finalmente, se escribe la sentencia SELECT y, para ejecutarla, se pulsa en la admiración de la pestaña “Diseño”.

```
#1
SELECT Marca,Modelo FROM Automóviles;
#2
SELECT modelo FROM Automóviles WHERE
    Emisiones >= 150 AND Emisiones <=165;
#3
SELECT modelo,consumo,emisiones FROM Automóviles WHERE
    (Emisiones >= 150 AND Emisiones <=165) OR (Consumo>=5 AND Consumo<=6)
        ORDER BY consumo DESC;
#4
SELECT DISTINCT Marca FROM Automóviles;
```



Práctica 4.2: Consultas simples con la BBDD jardinería

Codifica en MySQL y Oracle sentencias para obtener la siguiente información:

1. Sacar el código de oficina y la ciudad donde hay oficinas.
2. Sacar cuántos empleados hay en la compañía.
3. Sacar cuántos clientes tiene cada país.
4. Sacar cuál fue el pago medio en 2005 (pista: Usar la función YEAR de mysql o la función to_char(fecha,'yyyy') de Oracle).

5. Sacar cuántos pedidos están en cada estado ordenado descendente por el número de pedidos.
6. Sacar el precio del producto más caro y del más barato.

```

#1
SELECT CodigoOficina,ciudad FROM Oficinas;
#2
SELECT Count(*) FROM Empleados;
#3
SELECT Count(*),Pais FROM Clientes GROUP BY Pais;
#4
SELECT AVG(Cantidad) FROM Pagos WHERE YEAR(FechaPago)=2005; #(mysql) 6
#5
SELECT AVG(Cantidad) FROM Pagos WHERE TO_CHAR(FechaPago,'YYYY')='2005';
#6
SELECT Count(*),Estado FROM Pedidos GROUP BY Estado
ORDER BY Count(*) DESC;
#6
SELECT Max(PrecioVenta),Min(PrecioVenta) FROM Productos;

```



Práctica 4.3: Subconsultas con la BBDD jardinería

Codifica en SQL sentencias para obtener la siguiente información:

1. Obtener el nombre del cliente con mayor límite de crédito.
2. Obtener el nombre, apellido1 y cargo de los empleados que no representen a ningún cliente.

```

#1
SELECT NombreCliente FROM Clientes WHERE
    LimiteCredito = (SELECT Max(LimiteCredito) FROM Clientes);
#2
SELECT Nombre, Apellido1, Puesto FROM Empleados WHERE CodigoEmpleado
    NOT IN (SELECT CodigoEmpleadoRepVentas FROM Clientes );

```



Práctica 4.4: Consultas multitabla con la BBDD jardinería

Codifica en SQL consultas para obtener:

1. Sacar un listado con el nombre de cada cliente y el nombre y apellido de su representante de ventas.
2. Mostrar el nombre de los clientes que no hayan realizado pagos junto con el nombre de sus representantes de ventas.
3. Listar las ventas totales de los productos que hayan facturado más de 3000 euros. Se mostrará el nombre, unidades vendidas, total facturado y total facturado con impuestos (18% IVA).
4. Listar la dirección de las oficinas que tengan clientes en Fuenlabrada.

```
#1
SELECT NombreCliente, Nombre as NombreEmp, Apellido1 as ApeEmp
      FROM Clientes INNER JOIN Empleados ON
            Clientes.CodigoEmpleadoRepVentas=Empleados.CodigoEmpleado;

#2
SELECT NombreCliente,Nombre as NombreEmp, Apellido1 as ApeEmp
      FROM Clientes INNER JOIN Empleados ON
            Clientes.CodigoEmpleadoRepVentas=Empleados.CodigoEmpleado
      where CodigoCliente not in (SELECT CodigoCliente FROM Pagos);

#3
SELECT Nombre, SUM(Cantidad) As TotalUnidades,
       SUM(Cantidad*PrecioUnidad) as TotalFacturado,
       SUM(Cantidad*PrecioUnidad)*1.18 as TotalConImpuestos
      FROM DetallePedidos NATURAL JOIN Productos
      GROUP BY Nombre
      HAVING Sum(Cantidad*PrecioUnidad)>3000;

#4
SELECT CONCAT(Oficinas.LineaDireccion1,Oficinas.LineaDireccion2),
       Oficinas.Ciudad
      FROM Oficinas, Empleados,Clientes
     WHERE Oficinas.CodigoOficina=Empleados.CodigoOficina AND
           Empleados.CodigoEmpleado=Clientes.CodigoEmpleadoRepVentas AND
           Clientes.Ciudad='Fuenlabrada';
```

Práctica 4.5: Consulta con tablas derivadas

Sacar el cliente que hizo el pedido de mayor cuantía:

Esta consulta es mejor codificarla en un archivo de texto para no tener que escribirla múltiples veces si da errores. La estrategia para resolverla es hacer pequeñas consultas (queries A,B y C) para luego unirlas y generar la definitiva:

```
#query A: Sacar la cuantía de los pedidos:  
(select CódigoPedido, CódigoCliente,  
sum(Cantidad*PrecioUnidad) as total  
from Pedidos natural join DetallePedidos  
group by CódigoPedido,CódigoCliente) TotalPedidos;  
  
#query B: Sacar el pedido más caro:  
select max(total) from  
(select CódigoPedido, CódigoCliente,  
sum(Cantidad*PrecioUnidad) as total  
from Pedidos natural join DetallePedidos  
group by CódigoPedido,CódigoCliente) TotalPedidos;  
  
#query C: Sacar el código de cliente correspondiente  
al pedido más caro (querydefinitiva.sql)
```

querydefinitiva.sql

```
Select TotalPedidos.CódigoCliente,NOMBRECLIENTE from  
(select CódigoPedido, CódigoCliente,  
sum(Cantidad*PrecioUnidad) as total  
from Pedidos natural join DetallePedidos  
group by CódigoPedido,CódigoCliente) TotalPedidos  
inner join Clientes on  
Clientes.CódigoCliente=TotalPedidos.CódigoCliente  
where total=  
( select max(total) from  
(select CódigoPedido, CódigoCliente,  
sum(Cantidad*PrecioUnidad) as total  
from Pedidos natural join DetallePedidos  
group by CódigoPedido,CódigoCliente) TotalPedidos  
);
```



4.12. Prácticas Propuestas

Práctica 4.6: Consultas simples en MS-Access

Con la BBDD Automóviles, genera sentencias SELECT para obtener esta información:

1. Modelos de vehículos TDI.⁴ 
2. Modelos de la marca 'Audi' y de la marca 'Seat' ordenado por Marca y Modelo. 
3. Marcas de Vehículos que empiecen por T y terminen en a 
4. Vehículos que tengan foto. 
5. El consumo de los vehículos está expresado en litros/100km. Listar el consumo de los vehículos 'Seat' en MPG, Millas por galón (10 MPG=23.49 l/100km).



Práctica 4.7: Consultas simples con la BBDD jardinería

Codifica en SQL (Oracle y MySQL) sentencias para obtener la siguiente información:

1. Sacar la ciudad y el teléfono de las oficinas de Estados Unidos. 
2. Sacar el nombre, los apellidos y el email de los empleados a cargo de Alberto Soria. 
3. Sacar el cargo, nombre, apellidos y email del jefe de la empresa. 
4. Sacar el nombre, apellidos y cargo de aquellos que no sean representantes de ventas. 
5. Sacar el número de clientes que tiene la empresa. 
6. Sacar el nombre de los clientes españoles. 
7. Sacar cuántos clientes tiene cada país. 
8. Sacar cuántos clientes tiene la ciudad de Madrid. 
9. Sacar cuántos clientes tienen las ciudades que empiezan por M. 

⁴Hay que tener en cuenta que en Access, el comodín % es un *

10. Sacar el código de empleado y el número de clientes al que atiende cada representante de ventas. 
11. Sacar el número de clientes que no tiene asignado representante de ventas. 
12. Sacar cuál fue el primer y último pago que hizo algún cliente. 
13. Sacar el código de cliente de aquellos clientes que hicieron pagos en 2008. 
14. Sacar los distintos estados por los que puede pasar un pedido. 
15. Sacar el número de pedido, código de cliente, fecha requerida y fecha de entrega de los pedidos que no han sido entregados a tiempo. 
16. Sacar cuántos productos existen en cada línea de pedido. 
17. Sacar un listado de los 20 códigos de productos más pedidos ordenado por cantidad pedida. (pista: Usar el filtro LIMIT de MySQL o el filtro rownum de Oracle). 
18. Sacar el número de pedido, código de cliente, fecha requerida y fecha de entrega de los pedidos cuya fecha de entrega ha sido al menos dos días antes de la fecha requerida. (pista: Usar la función addDate de MySQL o el operador + de Oracle). 
19. Sacar la facturación que ha tenido la empresa en toda la historia, indicando la base imponible, el IVA y el total facturado. NOTA: La base imponible se calcula sumando el coste del producto por el número de unidades vendidas. El IVA, es el 18 % de la base imponible, y el total, la suma de los dos campos anteriores. 
20. Sacar la misma información que en la pregunta anterior, pero agrupada por código de producto filtrada por los códigos que empiecen por FR. 

◊

Práctica 4.8: Subconsultas con la BBDD jardinería

Codifica en SQL sentencias para obtener la siguiente información:

1. Obtener el nombre del producto más caro. 
2. Obtener el nombre del producto del que más unidades se hayan vendido en un mismo pedido. 

3. Obtener los clientes cuya línea de crédito sea mayor que los pagos que haya realizado.
4. Sacar el producto que más unidades tiene en stock y el que menos unidades tiene en stock.



◊

Práctica 4.9: Consultas multitabla con la BBDD jardinería

Codifica en SQL las siguientes consultas:

1. Sacar el nombre de los clientes y el nombre de sus representantes junto con la ciudad de la oficina a la que pertenece el representante.
2. Sacar la misma información que en la pregunta anterior pero solo los clientes que no hayan echo pagos.
3. Obtener un listado con el nombre de los empleados junto con el nombre de sus jefes.
4. Obtener el nombre de los clientes a los que no se les ha entregado a tiempo un pedido (FechaEntrega>FechaEsperada).



◊

Práctica 4.10: Consultas variadas con la BBDD jardinería

Codifica en SQL las siguientes consultas:

1. Sacar un listado de clientes indicando el nombre del cliente y cuántos pedidos ha realizado.
2. Sacar un listado con los nombres de los clientes y el total pagado por cada uno de ellos.
3. Sacar el nombre de los clientes que hayan hecho pedidos en 2008.
4. Listar el nombre del cliente y el nombre y apellido de sus representantes de aquellos clientes que no hayan realizado pagos.



5. Sacar un listado de clientes donde aparezca el nombre de su comercial y la ciudad donde está su oficina. 
 6. Sacar el nombre, apellidos, oficina y cargo de aquellos que no sean representantes de ventas. 
 7. Sacar cuántos empleados tiene cada oficina, mostrando el nombre de la ciudad donde está la oficina. 
 8. Sacar un listado con el nombre de los empleados, y el nombre de sus respectivos jefes. 
 9. Sacar el nombre, apellido, oficina (ciudad) y cargo del empleado que no representa a ningún cliente. 
 10. Sacar la media de unidades en stock de los productos agrupados por gama. 
 11. Sacar los clientes que residan en la misma ciudad donde hay una oficina, indicando dónde está la oficina. 
 12. Sacar los clientes que residan en ciudades donde no hay oficinas ordenado por la ciudad donde residen. 
 13. Sacar el número de clientes que tiene asignado cada representante de ventas. 
 14. Sacar cuál fue el cliente que hizo el pago con mayor cuantía y el que hizo el pago con menor cuantía. 
 15. Sacar un listado con el precio total de cada pedido. 
 16. Sacar los clientes que hayan hecho pedidos en el 2008 por una cuantía superior a 2000 euros. 
 17. Sacar cuántos pedidos tiene cada cliente en cada estado. 
 18. Sacar los clientes que han pedido más de 200 unidades de cualquier producto. 
- ◊

Práctica 4.11: Más consultas variadas

Con la base de datos NBA codifica las siguientes consultas:

1. Equipo y ciudad de los jugadores españoles de la NBA. 

2. Equipos que comiencen por H y terminen en S. 
3. Puntos por partido de 'Pau Gasol' en toda su carrera. 
4. Equipos que hay en la conferencia oeste ('west'). 
5. Jugadores de Arizona que pesen más de 100 kilos y midan más de 1.82m (6 pies). 
6. Puntos por partido de los jugadores de los 'cavaliers'. 
7. Jugadores cuya tercera letra de su nombre sea la v. 
8. Número de jugadores que tiene cada equipo de la conferencia oeste 'West'. 
9. Número de jugadores Argentinos en la NBA. 
10. Máxima media de puntos de 'Lebron James' en su carrera. 
11. Asistencias por partido de 'Jose Calderon' en la temporada '07/08'. 
12. Puntos por partido de 'Lebron James' en las temporadas del 03/04 al 05/06. 
13. Número de jugadores que tiene cada equipo de la conferencia este 'East'. 
14. Tapones por partido de los jugadores de los 'Blazers'. 
15. Media de rebotes de los jugadores de la conferencia Este 'East'. 
16. Rebotes por partido de los jugadores de los equipos de Los Angeles. 
17. Número de jugadores que tiene cada equipo de la división NorthWest. 
18. Número de jugadores de España y Francia en la NBA. 
19. Número de pivots 'C' que tiene cada equipo. 
20. ¿Cuánto mide el pívot más alto de la nba? 
21. ¿Cuánto pesa (en libras y en kilos) el pívot más alto de la NBA? 
22. Número de jugadores que empiezan por 'Y'. 
23. Jugadores que no metieron ningún punto en alguna temporada. 
24. Número total de jugadores de cada división. 
25. Peso medio en kilos y en libras de los jugadores de los 'Raptors'. 

26. Mostrar un listado de jugadores con el formato Nombre(Equipo) en una sola columna. 
27. Puntuación más baja de un partido de la NBA. 
28. Primeros 10 jugadores por orden alfabético. 
29. Temporada con más puntos por partido de 'Kobe Bryant'. 
30. Número de bases 'G' que tiene cada equipo de la conferencia este 'East'. 
31. Número de equipos que tiene cada conferencia. 
32. Nombre de las divisiones de la conferencia Este. 
33. Máximo reboteador de los 'Suns'. 
34. Máximo anotador de la toda base de datos en una temporada. 
35. Sacar cuántas letras tiene el nombre de cada jugador de los 'grizzlies' (Usar función LENGTH). 
36. ¿Cuántas letras tiene el equipo con nombre más largo de la NBA (Ciudad y Nombre)? 

◊

Práctica 4.12: Consultas con tablas derivadas

Realizar las siguientes consultas con tablas derivadas con las BBDD NBA y jardinería:

- Sacar el importe medio de los pedidos de la BBDD jardinería.
- Sacar un listado con el número de partidos ganados por los equipos de la NBA.
- Sacar la media de partidos ganados por los equipos del oeste.
- ¿Cuál es el pedido más caro del empleado que más clientes tiene?

◊

4.13. Resumen

Los conceptos clave de este capítulo son los siguientes:

- La sentencia SELECT devuelve un conjunto de resultados en forma de tabla compuesto por filas (o registros) y columnas (o campos).
- La cláusula FROM de la sentencia SELECT especifica las tablas de las que se extrae la información, y permite, a través de operaciones como el producto cartesiano y las JOIN, construir conjuntos de datos de información relacionada.
- Cuando se especifica más de una tabla en la cláusula FROM se denomina consulta multitabla. Pueden ser escritas en dos tipos de sintaxis, SQL1 y SQL2. SQL1 solo permite composiciones internas (INNER JOIN), mientras que SQL2 permite, además, composiciones externas (OUTER JOIN). Las NATURAL JOIN, son un tipo de INNER JOIN que hace coincidir la información de dos campos con el mismo nombre.
- Los registros de una SELECT se pueden FILTRAR mediante el uso de la cláusula WHERE. Los filtros se pueden construir mediante expresiones, el operador de pertenencia a conjuntos (IN), operador de rango (BETWEEN), test de valor nulo (IS, IS NOT), test de patrón (LIKE), y límitación de registros (LIMIT y numrows).
- Para ordenar un conjunto de resultados se utiliza la palabra clave ORDER BY. Se puede ordenar ascendentemente (ASC) o descendentemente (DESC).
- Las consultas de resumen se usan para calcular información en base a conjuntos o grupos de datos. Los grupos se construyen mediante la cláusula GROUP BY.
- Los filtros sobre grupos se generan mediante la cláusula HAVING.
- Las subconsultas son SELECT usadas para filtrar información mediante test de comparación, Test de Existencia (EXISTS), Test cuantificados (ANY y ALL). Pueden ser anidadas.
- Las consultas reflexivas son las que forman en su cláusula FROM la misma tabla dos o más veces.
- Las tablas derivadas son tablas virtuales generadas a través de consultas en tiempo de ejecución. Tienen un alias que las identifica.

4.14. Test de repaso

1. ¿Para qué sirve DISTINCT en una SELECT?

- a) Para mostrar las filas idénticas
- b) Para no mostrar filas idénticas
- c) Para mostrar, aparte, las filas distintas
- d) Ninguna de las anteriores

2. Un filtro WHERE puede incorporar expresiones con

- a) Operadores numéricos
- b) Operadores relacionales
- c) Llamadas a funciones
- d) Todas las anteriores

3. El operador IN no se puede usar para

- a) Escribir en un filtro una lista de valores
- b) Escribir en un filtro una subconsulta
- c) Una ordenación
- d) Encadenar varios condiciones de tipo AND

4. El test de valor nulo

- a) Sirve para comprobar si un conjunto de resultados es vacío
- b) Sirve para comprobar si el valor de un campo es desconocido
- c) Sirve para comprobar si el valor de un campo es o no es desconocido
- d) Todas las anteriores son correctas

5. El patrón %AX_ - seleccionaría el valor

- a) XXXAX11
- b) AX1111
- c) XXXX1F
- d) XXXAX1

6. Filtrar por el número de resultados

- a) No se puede de ninguna manera en Oracle
- b) Se puede mediante la cláusula LIMIT en Oracle
- c) Se puede mediante la cláusula numrows en Oracle
- d) No se puede hacer de ninguna manera en MySQL

7. En SQL se puede ordenar por

- a) El número de columna (1,2,3...)
- b) El nombre de la columna
- c) Una expresión
- d) Todas las anteriores

8. Si junto a una función de columna, se selecciona un campo

- a) Se debe agrupar por el campo
- b) No se debe agrupar por el campo
- c) No se puede seleccionar el campo
- d) No se puede seleccionar la función de columna

9. Una subconsulta

- a) Es un tipo especial de tabla derivada
- b) Se puede anidar con otras subconsultas
- c) Sus resultados se pueden ordenar bajo determinadas circunstancias
- d) Todas las opciones anteriores son posibles

SOLUCIONES: 1.b; 2.d; 3.c; 4.c; 5.a; 6.c; 7.d; 8.a; 9.b.

4.15. Comprueba tu aprendizaje

1. Realiza una lista con los operadores que puedes escribir en un filtro, clasificándolos según su tipo.
2. La tabla alumno tiene un campo llamado Nacionalidad. Escribe una consulta para sacar los alumnos de España, Italia y Dinamarca de dos formas, una con el operador IN, y otra con operadores AND.
3. Escribe la sintaxis de la sentencia SELECT con todas sus cláusulas correspondientes (ORDER BY, HAVING, GROUP BY, etc.) y describe para qué sirve cada una.
4. Haz un esquema clasificando los tipos de JOIN que existen. Incorpora un ejemplo de cada una de ellas.
5. ¿En qué se diferencia LEFT OUTER JOIN de RIGHT OUTER JOIN? Pon un ejemplo de una consulta que afecte a dos tablas, indicando la diferencia.
6. ¿En qué se diferencia FULL OUTER JOIN de INNER JOIN? Pon un ejemplo de una consulta que afecte a dos tablas, indicando la diferencia.
7. ¿Qué es una tabla derivada? ¿Debe llevar alias una tabla derivada? ¿Por qué?
8. Escribe un ejemplo de la ejecución de una subconsulta con el operador EXISTS y otra con el operador NOT EXISTS.
9. ¿Para qué sirve el operador UNION? Pon un ejemplo de su uso.
10. Define para qué sirven los siguientes operadores de filtros, y pon un ejemplo de cada uno de ellos:
 - BETWEEN ... AND
 - ANY
 - ALL
 - IS
 - IS NOT
 - LIKE
11. ¿Qué diferencia hay entre HAVING $x>y$ y WHERE $x>y$?
12. ¿Para qué sirve un CROSS JOIN?
13. ¿Qué diferencia hay entre NATURAL JOIN e INNER JOIN? Pon un ejemplo usando ambas de dos consultas que hagan produzcan los mismos resultados.
14. ¿Qué es una consulta reflexiva? Pon un ejemplo de una sentencia SQL con una consulta reflexiva.
15. ¿Se podría utilizar una tabla derivada dentro de una subconsulta?