Tema 7. Consultas Básicas de Selección

Contenido

1 La sentencia SELECT 2
1.1 Sintaxis sencilla del comando SELECT 2
2 Consultas calculadas 3
2.1 Cálculos aritméticos 3
2.2 Concatenación de textos 3
3 Condiciones 4
3.1 Operadores de comparación 4
3.2 Valores lógicos 4
3.3 BETWEEN 5
3.4 IN 5
3.5 LIKE 5
3.6 IS NULL 6
3.7 Precedencia de operadores 6
4 Ordenación 6
5 Nulos 7
6 Subconsultas 8
6.1 Subconsultas simples 8
6.2 Subconsultas de múltiples filas 9
6.3 Consultas EXISTS 10
7 Consultas multitabla 11
7.1 Producto cartesiano 11
7.2 Asociar tablas 11
7.3 Relaciones sin igualdad 12
7.4 Sintaxis 1999 13
7.4.1 CROSS JOIN 13
7.4.2 NATURAL JOIN 14
7.4.3 JOIN USING 14
7.4.4 JOIN ON (JOIN ON = INNER JOIN ON) 14
7.4.5 Relaciones externas 14
7.4.6 Consultas de no coincidentes 15

1 La sentencia SELECT

El único comando que pertenece al lenguaje de consultas de SQL es el versátil comando **SELECT**.

Este comando permite:

- Obtener datos de ciertas columnas de una tabla (proyección).
- Obtener registros (filas) de una tabla de acuerdo con ciertos criterios (selección).
- Mezclar datos de tablas diferentes (asociación, join, cruce de tablas).
- Realizar cálculos sobre los datos.
- Agrupar datos.

1.1 Sintaxis sencilla del comando SELECT

```
SELECT { * | [DISTINCT] columna | expresión [[AS] alias], ...}
FROM tabla;
```

Donde:

- *****. El asterisco significa que se seleccionan todas las columnas
- **DISTINCT**. Hace que no se muestren los valores duplicados.
- **columna**. Es el nombre de una columna de la tabla que se desea mostrar
- **expresión**. Una expresión válida SQL
- **alias**. Es un nombre que se le da a la cabecera de la columna en el resultado de esta instrucción.

Ejemplos:

```
/* Selección de todos los registros de la tabla clientes */
SELECT * FROM Clientes;

/* Selección de algunos campos*/
SELECT nombre, apellido1, apellido2 FROM Clientes;
```

2 Consultas calculadas

2.1 Cálculos aritméticos

Los operadores + (suma), - (resta), * (multiplicación) y / (división), se pueden utilizar para hacer cálculos en las consultas. Cuando se utilizan como expresión en una consulta SELECT, no modifican los datos originales sino que como resultado de la vista generada por SELECT, aparece una nueva columna. Ejemplo:

```
SELECT nombre, precio,precio*1.21 FROM articulos;
```

Esa consulta obtiene tres columnas. La tercera tendrá como nombre la expresión utilizada, para poner un alias basta utilizar dicho alias tras la expresión:

```
SELECT nombre, precio, precio*1.21 AS precio_con_iva
FROM articulos;
```

La prioridad de esos operadores es la normal: tienen más prioridad la multiplicación y división, después la suma y la resta. En caso de igualdad de prioridad, se realiza primero la operación que esté más a la izquierda. Como es lógico se puede evitar cumplir esa prioridad usando paréntesis; el interior de los paréntesis es lo que se ejecuta primero.

Cuando una expresión aritmética se calcula sobre valores NULL, el resultado es el propio valor NULL.

2.2 Concatenación de textos

Todas las bases de datos incluyen algún operador para encadenar textos. En SQLSERVER es el signo + en Oracle son los signos ||. Ejemplo (Oracle):

```
SELECT tipo, modelo, tipo || '-' || modelo "Clave Pieza"
FROM piezas;
```

El resultado sería:

TIPO	MODELO	Clave Pieza
AR	6	AR-6
AR	7	AR-7
AR	8	AR-8
AR	9	AR-9
AR	12	AR-12
AR	15	AR-15
AR	20	AR-20
AR	21	AR-21
BI	10	BI-10
BI	20	BI-20
BI	22	BI-22
BI	24	BI-24

En la mayoría de bases de datos, la función CONCAT (se describe más adelante) realiza la misma función.

3 Condiciones

Se pueden realizar consultas que restrinjan los datos de salida de las tablas. Para ello se utiliza la cláusula **WHERE**. Esta cláusula permite colocar una condición que han de cumplir todos los registros, los que no la cumplan no aparecen en el resultado.

Ejemplo:

```
SELECT Tipo, Modelo FROM Piezas WHERE Precio>3;
```

3.1 Operadores de comparación

Se pueden utilizar en la cláusula **WHERE**, son:

Operador	Significado
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
=	Igual
<>	Distinto
!=	Distinto

Se pueden utilizar tanto para comparar números como para comparar textos y fechas. En el caso de los textos, las comparaciones se hacen en orden alfabético. Solo que es un orden alfabético estricto. Es decir el orden de los caracteres en la tabla de códigos.

En muchas bases de datos hay problemas con la ñ y otros símbolos nacionales (en especial al ordenar o comparar con el signo de mayor o menor, ya que el orden ASCII no respeta el orden de cada alfabeto nacional). No obstante es un problema que tiende a arreglarse en la actualidad en todos los SGBD (en Oracle no existe problema alguno) especialmente si son compatibles con Unicode.

3.2 Valores lógicos

Son:

Operador	Significado
AND	Devuelve verdadero si las expresiones a su izquierda y derecha son ambas verdaderas
OR	Devuelve verdadero si cualquiera de las dos expresiones a izquierda y derecha del OR, son verdaderas
NOT	Invierte la lógica de la expresión que está a su derecha. Si era verdadera, mediante NOT pasa a ser falso.

Ejemplos:

```
/* Obtiene a las personas de entre 25 y 50 años*/
SELECT nombre, apellido1,apellido2 FROM personas
WHERE edad>=25 AND edad<=50;

/*Obtiene a la gente de más de 60 años o de menos de 20*/
SELECT nombre, apellido1,apellido2 FROM personas
WHERE edad>60 OR edad<20;
```

3.3 BETWEEN

El operador BETWEEN nos permite obtener datos que se encuentren en un rango. Uso:

```
SELECT tipo,modelo,precio FROM piezas
 WHERE precio BETWEEN 3 AND 8;
```

Saca piezas cuyos precios estén entre 3 y 8 (ambos incluidos).

3.4 IN

Permite obtener registros cuyos valores estén en una lista de valores:

```
SELECT tipo,modelo,precio FROM piezas
 WHERE precio IN (3, 5, 8);
```

Obtiene piezas cuyos precios sean 3, 5 u 8 (no valen ni el precio 4 ni el 6, por ejemplo).

3.5 LIKE

Se usa sobre todo con textos, permite obtener registros cuyo valor en un campo cumpla una condición textual. LIKE utiliza una cadena que puede contener estos símbolos:

Símbolo	Significado
%	Una serie cualquiera de caracteres
_	Un carácter cualquiera

Ejemplos:

```
/* Selecciona nombres que empiecen por S */
SELECT nombre FROM personas WHERE nombre LIKE 'S%';

/*Selecciona las personas cuyo apellido sea Sanchez, Senchez, Stnchez,...*/
SELECT apellido1 FROM Personas WHERE apellido1
 LIKE 'S_nchez';
```

3.6 IS NULL

Devuelve verdadero si el valor que examina es nulo:

```
SELECT nombre,apellidos FROM personas
WHERE telefono IS NULL
```

Esa instrucción selecciona a la gente que no tiene teléfono. Se puede usar la expresión IS NOT NULL que devuelve verdadero en el caso contrario, cuando la expresión no es nula.

3.7 Precedencia de operadores

A veces las expresiones que se producen en los SELECT son muy extensas y es difícil saber qué parte de la expresión se evalúa primero, por ello se indica la siguiente tabla de precedencia (tomada de Oracle):

Orden de precedencia	Operador
1	*(Multiplicar) / (dividir)
2	+ (Suma) - (Resta)
3	|| (Concatenación)
4	Comparaciones (>, <, !=, ...)
5	IS [NOT] NULL, [NOT]LIKE, IN
6	NOT
7	AND
8	OR

4 Ordenación

El orden inicial de los registros obtenidos por una **SELECT** no guarda más que una relación respecto al orden en el que fueron introducidos. Para ordenar basándose en criterios más interesantes, se utiliza la cláusula **ORDER BY**.

En esa cláusula se coloca una lista de campos que indica la forma de ordenar. Se ordena primero por el primer campo de la lista, si hay coincidencias por el segundo, si ahí también las hay por el tercero, y así sucesivamente.

Se puede colocar la palabra **ASC** O **DESC** (por defecto se toma **ASC**). Esas palabras significan en ascendente (de la A a la z, de los números pequeños a los grandes) o en descendente (de la z a la A, de los números grandes a los pequeños) respectivamente.

Sintaxis completa de **SELECT** (para una sola tabla):

SELECT {* | [**DISTINCT**] {columna | expresión} [[**AS**] alias], ... }
FROM tabla
[**WHERE** condición]
[**ORDER BY** expresión1 [,expresión2,…][{**ASC**|**DESC**}]];

5 Nulos

En los lenguajes de programación se utiliza el valor nulo para reflejar que un identificador (una variable, un objeto,..) no tiene ningún contenido. Por ejemplo cuando un puntero en lenguaje C señala a null se dice que no está señalando a nada. Al programar en esos lenguajes no se permite utilizar un valor nulo en operaciones aritméticas o lógicas.

Las bases de datos relacionales permiten más posibilidades para el valor nulo (null), aunque su significado no cambia: valor vacío. No obstante en las bases de datos se utiliza para diversos fines.

En claves ajenas o secundarias indican que el registro actual no está relacionado con ninguno. En otros atributos indica que la tupla en cuestión carece de dicho atributo: por ejemplo en una tabla de personas un valor nulo en el atributo teléfono indicaría que dicha persona no tiene teléfono.

Es importante indicar que el texto vacío ‘’, no significa lo mismo en un texto que el nulo; como tampoco el valor cero significa nulo.

Puesto que ese valor se utiliza continuamente, resulta imprescindible saber cómo actúa cuando se emplean operaciones lógicas sobre ese valor. Eso significa definir un tercer valor en la lógica booleana, además de los clásicos verdadero y falso. Un valor nulo no es ni verdadero ni falso (se suele interpretar como un quizás, o usando la aritmética clásica en valores lógicos, el 1 o un valor mayor que 1 es verdadero, el 0 falso y el 0,5 nulo).

El uso de operadores lógicos con el nulo da lugar a que:

- **verdadero Y (AND) nulo** da como resultado, nulo (siguiendo la aritmética planteada antes: 1·0,5=0,5)
- **falso Y (AND) nulo** da como resultado, falso (0·0,5=0)
- **verdadero O (OR) nulo** da como resultado, verdadero (1+0,5>1)
- **falso O nulo** da como resultado nulo (0+0,5=0,5)
- **la negación de nulo**, da como resultado nulo

6 Subconsultas

6.1 Subconsultas simples

Se trata de una técnica que permite utilizar el resultado de una consulta SELECT en otra consulta SELECT. Permite solucionar consultas que requieren para funcionar el resultado previo de otra consulta.

La sintaxis es:

```
SELECT listaExpresiones
FROM tabla
WHERE expresión OPERADOR
 (SELECT listaExpresiones
 FROM tabla);
```

Se puede colocar el SELECT de la subconsulta dentro de las cláusulas WHERE, FROM o HAVING (Se verá más adelante). El operador puede ser >, <,>=, <=, !=, = o IN.

Ejemplo:

```
SELECT nombre_empleado, paga
FROM empleados
WHERE paga <
 (SELECT paga FROM empleados
 WHERE nombre_empleado='Martina');
```

Lógicamente el resultado de la subconsulta debe incluir el campo que estamos analizando. Se pueden realizar esas subconsultas las veces que haga falta:

```
SELECT nombre_empleado, paga
FROM empleados
WHERE paga <
 (SELECT paga FROM empleados
 WHERE nombre_empleado='Martina')
AND paga >
 (SELECT paga FROM empleados WHERE nombre_empleado='Luis');
```

En realidad lo primero que hace la base de datos es calcular el resultado de la subconsulta:

```
SELECT nombre_empleado, paga
FROM empleados
WHERE paga < (SELECT paga FROM empleados
              WHERE nombre_empleado='Martina')
AND paga > (SELECT paga FROM empleados
            WHERE nombre_empleado='Luis');
```

La última consulta obtiene los empleados cuyas pagas estén entre lo que gana Luis (1870 euros) y lo que gana Martina (2500).

Las subconsultas siempre se deben encerrar entre paréntesis y se debería colocar a la derecha del operador relacional. Una **subconsulta** que utilice los operadores de comparación >, <,>=,... **tiene que devolver un único valor**, de otro modo ocurre un error. Además tienen que tener el mismo tipo de columna para relacionar la subconsulta con la consulta que la utiliza (no puede ocurrir que la subconsulta tenga dos columnas y ese resultado se compare usando una sola columna en la consulta general).

6.2 Subconsultas de múltiples filas

En el apartado anterior se comentaba que las subconsultas sólo pueden devolver una fila. Pero a veces se necesitan consultas del tipo: mostrar el sueldo y nombre de los empleados cuyo sueldo supera al de cualquier empleado del departamento de ventas.

La subconsulta necesaria para ese resultado mostraría todos los sueldos del departamento de ventas. Pero no podremos utilizar un operador de comparación directamente ya que esa subconsulta devuelve más de una fila. La solución a esto es utilizar operadores lógicos especiales, que permiten el uso de subconsultas de varias filas.

Estos operadores son:

Instrucción	Significado
ANY	Compara con cualquier registro de la subconsulta. La instrucción es válida si hay un registro en la subconsulta que permite que la comparación sea cierta
ALL	Compara con todos los registros de la consulta. La instrucción resulta cierta si es cierta toda comparación con los registros de la subconsulta
IN	No usa comparador, ya que sirve para comprobar si un valor se encuentra en el resultado de la subconsulta
NOT IN	Comprueba si un valor no se encuentra en una subconsulta

Ejemplo:

```
SELECT nombre, sueldo
FROM empleados
WHERE sueldo >= ALL (SELECT sueldo FROM empleados);
```

La consulta anterior obtiene el empleado que más cobra.

```
SELECT nombre FROM empleados
WHERE dni IN (SELECT dni FROM directivos);
```

En ese caso se obtienen los nombres de los empleados cuyos dni están en la tabla de directivos.

Si se necesita comprobar dos columnas en una consulta IN, se hace:

```
SELECT nombre FROM empleados
WHERE (cod1,cod2) IN (SELECT cod1,cod2 FROM directivos);
```

6.3 Consultas EXISTS

Este operador devuelve verdadero si la consulta que le sigue devuelve algún valor. Si no, devuelve falso. Se utiliza sobre todo en consultas correlacionadas(*). Ejemplo:

```
SELECT tipo, modelo, precio_venta
FROM piezas p
WHERE EXISTS (
 SELECT tipo, modelo FROM existencias
 WHERE tipo=p.tipo AND modelo=p.modelo);
```

La tabla **`piezas`** necesita llevar un alias para poder evaluar la condición de selección de la subconsulta ya que tanto en **`piezas`** como en **`existencias`** hay columnas con el mismo nombre (**`tipo`** y **`modelo`**)

Esta consulta devuelve las piezas que se encuentran en la tabla de existencias (es igual al ejemplo comentado en el apartado subconsultas de múltiples filas). La consulta contraria es:

```
SELECT tipo, modelo, precio_venta
FROM piezas p
WHERE NOT EXISTS (
 SELECT tipo, modelo FROM existencias
 WHERE tipo=p.tipo AND modelo=p.modelo);
```

Normalmente las consultas EXISTS se pueden realizar de alguna otra forma con los operadores ya comentados.

(*)Subconsultas correlacionadas

Cuando los nombres de columnas que aparecen en una subconsulta son nombres de columnas de la consulta principal o de otra subconsulta más externa, caso de las anidadas, se dice que son **referencias externas** y la **subconsulta** que es **correlacionada**.

En las subconsultas correlacionadas, cada vez que se selecciona una nueva fila, en la consulta principal o en otra subconsulta más externa, que contenga la columna referenciada, se repetirá el proceso de selección.

Si en una subconsulta correlacionada coincide el nombre de una referencia externa con el nombre de alguna columna de la tabla que está siendo seleccionada en la subconsulta, se deberá asignar un alias a cada tabla y se utilizará para identificar cada columna.

7 Consultas multitabla

Lo normal en una consulta es necesitar datos que se encuentran distribuidos en varias tablas. Las bases de datos relacionales se basan en que los datos se distribuyen en tablas que se pueden relacionar mediante un campo. Ese campo es el que permite integrar los datos de las tablas.

Por ejemplo, en una tabla de empleados cuya clave es el **dni** y otra tabla de tareas que se refiere a tareas realizadas por los empleados, es seguro (si el diseño está bien hecho) que en la tabla de tareas aparecerá el **dni** del empleado para saber qué empleado realizó la tarea.

7.1 Producto cartesiano

```
SELECT {* | [DISTINCT] {columna | expresión} [[AS] alias], ... }
FROM tabla1, tabla2, tabla3,…
[WHERE condición]
[ORDER BY expresión1 [,expresión2,…][{ASC|DESC}]];
```

En el apartado FROM se pueden indicar varias tablas separadas por comas. Pero eso produce un producto cartesiano, aparecerán todos los registros de cada tabla relacionados con todos los registros de las otras tablas.

Ejemplo:

```
SELECT cod_tarea, descripcion_tarea, dni_empleado, nombre_empleado
FROM tareas,empleados;
```

El producto cartesiano a veces es útil para realizar consultas complejas, pero en el caso normal no lo es, se hace necesario discriminar ese producto para que sólo aparezcan los registros de las tareas relacionadas con sus empleados correspondientes. A eso se le llama asociar (join) tablas.

7.2 Asociar tablas

Se escribe en la cláusula **WHERE** la condición de asociación.

Ejemplo:

Si tenemos las tablas:

empleados

dni	nombre_empleado	sueldo

tareas

cod_tarea	descripcion_tarea	dni_empleado

utensilios_utilizados

cod_tarea	nombre_utensilio

La consulta anterior se escribiría del a siguiente manera:

```
SELECT cod_tarea, descripcion_tarea, dni_empleado, nombre_empleado
FROM tareas,empleados
WHERE tareas.dni_empleado = empleados.dni;
```

Se utiliza la notación **tabla.columna** para evitar la ambigüedad, ya que el mismo nombre de campo se puede repetir en varias tablas. Para evitar repetir continuamente el nombre de la tabla, se puede utilizar un alias de tabla:

```
SELECT t.cod_tarea, t.descripcion_tarea, e.dni, e.nombre_empleado
FROM tareas t,empleados e
WHERE t.dni_empleado = e.dni;
```

Al apartado **WHERE** se le pueden añadir condiciones encadenándolas con el operador **AND**.

Ejemplo:

```
SELECT t.cod_tarea, t.descripcion_tarea
FROM tareas t,empleados e
WHERE t.dni_empleado = e.dni AND e.nombre_empleado='Javier';
```

Es posible enlazar más de dos tablas a través de sus campos relacionados.

Ejemplo:

```
SELECT t.cod_tarea, t.descripcion_tarea, e.nombre_empleado, u.nombre_utensilio
FROM tareas t,empleados e, utensilios_utilizados u
WHERE t.dni_empleado = e.dni AND t.cod_tarea=u.cod_tarea;
```

7.3 Relaciones sin igualdad

A las relaciones descritas anteriormente se las llama relaciones en igualdad, ya que las tablas se relacionan a través de campos que contienen valores iguales en dos tablas.

Sin embargo no siempre las tablas tienen ese tipo de relación, por ejemplo:

EMPLEADOS	
Empleado	Sueldo
Antonio	18000
Marta	21000
Sonia	15000

CATEGORIAS		
categoría	Sueldo mínimo	Sueldo máximo
D	6000	11999
C	12000	17999
B	18000	20999
A	20999	80000

En el ejemplo anterior podríamos averiguar la categoría a la que pertenece cada empleado de la siguiente manera:

```
SELECT e.empleado, e.sueldo, c.categoria
FROM empleados e, categorias c
WHERE e.sueldo BETWEEN c.sueldo_minimo AND c.sueldo_maximo;
```

7.4 Sintaxis 1999

En la versión SQL de 1999 se ideó una nueva sintaxis para consultar varias tablas. La razón fue separar las condiciones de asociación respecto de las condiciones de selección de registros. Oracle incorpora totalmente esta normativa.

La sintaxis completa es:

```
SELECT tabla1.columna1, tabl1.columna2,... tabla2.columna1, tabla2.columna2,...
FROM tabla1 [CROSS JOIN tabla2]| [NATURAL JOIN tabla2]|
    [JOIN tabla2 USING(columna)]|
    [JOIN tabla2 ON (tabla1.columa=tabla2.columna)]|
    [LEFT|RIGHT|FULL OUTER JOIN tabla2 ON (tabla1.columa=tabla2.columna)]
```

7.4.1 CROSS JOIN

Utilizando la opción CROSS JOIN se realiza un producto cruzado entre las tablas indicadas. Eso significa que cada tupla de la primera tabla se combina con cada tupla de la segunda tabla. Es decir si la primera tabla tiene 10 filas y la segunda otras 10, como resultado se obtienen 100 filas, resultado de combinar todas entre sí.

```
SELECT * FROM piezas CROSS JOIN existencias;
```

No es una operación muy utilizada.

7.4.2 NATURAL JOIN

Establece una relación de igualdad entre las tablas a través de los campos que tengan el mismo nombre en ambas tablas:

```
SELECT * FROM piezas
NATURAL JOIN existencias;
```

En ese ejemplo se obtienen los registros de piezas relacionados en existencias a través de los campos que tengan el mismo nombre en ambas tablas.

Hay que asegurarse de que sólo son las claves principales y ajenas de las tablas relacionadas, las columnas en las que el nombre coincide, de otro modo fallaría la asociación y la consulta no funcionaría.

7.4.3 JOIN USING

Permite establecer relaciones indicando qué columna (o columnas) común a las dos tablas hay que utilizar:

```
SELECT * FROM piezas
JOIN existencias USING(tipo, modelo);
```

Las columnas deben tener exactamente el mismo nombre en ambas tablas.

7.4.4 JOIN ON (JOIN ON = INNER JOIN ON)

Permite establecer relaciones cuya condición se establece manualmente, lo que permite realizar asociaciones más complejas o bien asociaciones cuyos campos en las tablas no tienen el mismo nombre:

```
SELECT * FROM piezas
JOIN existencias ON (piezas.tipo=existencias.tipo AND
                     piezas.modelo=existencias.modelo);
```

7.4.5 Relaciones externas

Utilizando las formas anteriores de relacionar, sólo aparecen en el resultado de la consulta filas presentes en las tablas relacionadas. Es decir en esta consulta:

```
SELECT * FROM piezas
JOIN existencias
ON (piezas.tipo=existencias.tipo AND piezas.modelo=existencias.modelo);
```

Sólo aparecen piezas presentes en la tabla de existencias. Si hay piezas que no están en existencias, éstas no aparecen (y si hay existencias que no están en la tabla de piezas, tampoco salen).

Por ello se permite utilizar relaciones laterales o externas (outer join). Su sintaxis es:

```
{LEFT | RIGHT | FULL} OUTER JOIN tabla
{ON(condición) | USING (expresion)}
```

Así esta consulta:

```
SELECT * FROM piezas
LEFT OUTER JOIN existencias
ON(piezas.tipo=existencias.tipo AND piezas.modelo=existencias.modelo);
```

Obtiene los datos de las piezas estén o no relacionadas con datos de la tabla de existencias (la tabla **LEFT** sería piezas porque es la que está a la izquierda del **JOIN**).

En la consulta anterior, si el **LEFT** lo cambiamos por un **RIGHT**, aparecerán las existencias no presentes en la tabla piezas (además de las relacionadas en ambas tablas).

La condición **FULL OUTER JOIN** produciría un resultado en el que aparecen los registros no relacionados de ambas tablas (piezas sin existencias relacionadas y viceversa).

7.4.6 Consultas de no coincidentes

Un caso típico de uso de las relaciones externas es el uso de consultas de no coincidentes. Estas consultas sustituyen de forma más eficiente al operador NOT IN ya que permiten comprobar filas de una tabla que no están presentes en una segunda tabla que se relaciona con la primera a través de alguna clave secundaria.

Por ejemplo supongamos que tenemos una base de datos que posee una tabla de empresas y otra de trabajadores de las empresas. Ambas tablas supongamos que se relacionan a través del CIF de la empresa. Si existen empresas de las que no tenemos ningún trabajador podremos saberlo mediante esta consulta:

```
SELECT e.nombre FROM empresas e
LEFT OUTER JOIN trabajadores t ON (t.cif=e.cif)
WHERE t.dni IS NULL;
```

En la consulta anterior gracias a utilizar el operador **LEFT OUTER**, obtenemos la lista completa de empresas (tengan o no trabajadores). Si eliminamos aquellas filas con el dni de los trabajadores no nulo, entonces salen las empresas sin trabajadores relacionados (sólo las empresas sin trabajadores mostrarán valores nulos en los datos de los trabajadores).