CONSULTA

SELECT

```
SELECT [ALL | DISTINCT] columna1 "columna primera", columna2  AS
"columna segunda", ...
FROM tabla1 T1, usuario.tabla2 T2, ...
WHERE condición1, condición2, ...
ORDER BY columna1 [ASC | DESC], 2 [ASC | DESC], …, columnaN [ASC |
DESC];
```

T1 y T2 son alias de las tablas.
2 en ORDER BY indica orden por la columna segunda indicada en SELECT
Las cadenas en las funciones y operadores van entre comillas simples.

Comparación (en WHERE)
(=, !=, < >, ^=, <,>, <=, >=, IN (), NOT IN (), BETWEEN, NOT BETWEEN, LIKE '_abc%', IS NULL)

LIKE se utiliza sobre todo con textos y permite obtener columnas cuyo valor en un campo cumpla una condición textual. Utiliza una cadena que puede contener los símbolos "%" que sustituye a un conjunto de caracteres o "_" que sustituye a un carácter. Ej.

```
SELECT nombre FROM EMPLEADOS WHERE APELLIDO1 LIKE 'R %';

SELECT UNIV_COD, NOMBRE_UNIV FROM UNIVERSIDADES WHERE CIUDAD IN
('SEVILLA', 'CÁDIZ');
```

Operadores

Operadores aritméticos (+, -, *, /)

```
SELECT SALARIO*1,05 FROM EMPLEADOS WHERE SALARIO<=1000;
```
salario+5%

Esto es una consulta calculada (SELECT y WHERE):

```
SELECT Nombre, Credito, Credito+25 AS CreditoNuevo FROM USUARIOS;
```

Operador de concatenación (||)

```
SELECT Nombre, Apellido1 || ' ' || Apellido2 FROM EMPLEADOS;
```

Operadores lógicos (AND, OR, NOT)

```
SELECT empl_dni FROM HIS_SALAR WHERE salario<=800 OR salario>2000;
SELECT NOMBRE_TRAB FROM TRABAJOS WHERE NOMBRE_TRAB NOT IN
     ('CONTABLE');
```
Lo siguiente también funciona:

```
SELECT NOMBRE_TRAB FROM TRABAJOS WHERE NOT NOMBRE_TRAB='CONTABLE';
```

Precedencias o qué expresión se evalúa primero

(* y /, + y -, | |, comparaciones (<, >, …), operadores IS NULL, IN NOT NULL, LIKE, BETWEEN, NOT, AND, OR)

Funciones *(próx. página)*

Funciones numéricas

ABS (n) valor absoluto SELECT ABS(-17) FROM DUAL; – Resultado: 17

EXP(n) e^n SELECT EXP(2) FROM DUAL; – Resultado: 7,38

CEIL(n) Calcula el valor entero inmediatamente superior o igual al argumento n.
SELECT CEIL(17.4) FROM DUAL; – Resultado: 18

FLOOR(n) Calcula el valor entero inmediatamente inferior o igual al parámetro n.
SELECT FLOOR(17.4) FROM DUAL; – Resultado: 17

MOD(m,n) resto de dividir m/n SELECT MOD(15, 2) FROM DUAL; – Resultado: 1

POWER(valor, exponente) Potencia SELECT POWER(4, 5) FROM DUAL; Res: 1024

ROUND(n, decimales) SELECT ROUND(12.5874, 2) FROM DUAL; – Result: 12.59

SQRT(n) raíz cuadrada SELECT SQRT(25) FROM DUAL; – Resultado: 5

TRUNC(m,n) Trunca un número a la cantidad de decimales especificada por el segundo argumento. Si se omite el segundo argumento, se truncan todos los decimales. Si "n" es negativo, el número es truncado desde la parte entera.
SELECT TRUNC(127.4567, 2) FROM DUAL; – Resultado: 127.45
SELECT TRUNC(4572.5678, -2) FROM DUAL; – Resultado: 4500
SELECT TRUNC(4572.5678, -1) FROM DUAL; – Resultado: 4570
SELECT TRUNC(4572.5678) FROM DUAL; – Resultado: 4572

SIGN(n) Si el argumento "n"es valor positivo, retorna 1, si es negativo, devuelve -1 y 0 si es 0.
SELECT SIGN(-23) FROM DUAL; – Resultado: -1

Funciones de cadena de caracteres

CHR(n) carácter ASCII SELECT CHR(81) FROM DUAL; – Resultado: Q

ASCII(n) código ASCII SELECT ASCII('O') FROM DUAL; – Resultado: 79

CONCAT(cad1, cad2) equivale a || SELECT CONCAT('Hola', 'Mundo') FROM DUAL; – Resultado: HolaMundo

LOWER(cad) SELECT LOWER('En MInúsculAS') FROM DUAL; en minúsculas

UPPER(cad) SELECT UPPER('En MAyúSCulAS') FROM DUAL; EN MAYÚSCULAS

INITCAP(cad) 1a. en Mays. SELECT INITCAP('hola') FROM DUAL; – Hola

LPAD(cad1, n, cad2) SELECT LPAD('M', 5, '*') FROM DUAL; – ****M
Devuelve cad1 con longitud n, ajustada a la derecha, rellenando por la izquierda con cad2.

RPAD(cad1, n, cad2) SELECT RPAD('M', 5, '*') FROM DUAL; – M****
Devuelve cad1 con longitud n, ajustada a la izquierda, rellenando por la derecha con cad2.

REPLACE(cad, ant, nue) Devuelve la cad en la que cada ocurrencia de la cadena anterior ha sido sustituida por la cadena nue. SELECT REPLACE('correo@gmail.es', 'es', 'com') FROM DUAL; – Resultado: correo@gmail.com

SUBSTR(cad, m, n) SELECT SUBSTR('1234567', 3, 2) FROM DUAL; – 34
Devuelve la cadena cad compuesta por n caracteres a partir de la posición m.

LENGTH(cad) Devuelve la longitud de cad. SELECT LENGTH('hola') FROM DUAL; – 4

TRIM(cad) SELECT TRIM(' Hola__mundo ') FROM DUAL; – Resul.: 'Hola mundo'
Elimina los espacios a la izquierda y la derecha de cad y los espacios dobles del interior.

LTRIM(cad) SELECT LTRIM(' Hola') FROM DUAL; – Resultado: 'Hola'
Elimina los espacios a la izquierda que posea cad.

RTRIM(cad) SELECT RTRIM('Hola ') FROM DUAL; – Resultado: 'Hola'
Elimina los espacios a la derecha que posea cad.

INSTR(cad, cadBuscada [, posInicial [, nAparición]])
Obtiene la posición en la que se encuentra la cadena buscada en la cadena inicial cad. Se puede comenzar a buscar desde una posición inicial concreta e incluso indicar el número de aparición de la cadena buscada. Si no encuentra nada devuelve cero.
SELECT INSTR('usuarios', 'u') FROM DUAL; – Resultado: 1
SELECT INSTR('usuarios', 'u', 2) FROM DUAL; – Resultado: 3
SELECT INSTR('usuarios', 'u', 2, 2) FROM DUAL; – Resultado: 0

Funciones de manejo de fechas

Notas:A las fechas le podemos sumar números y esto se entiende como sumarles días, si ese número tiene decimales se suman días, horas, minutos y segundos. La diferencia entre dos fechas también nos dará un número de días.
En Oracle: Los operadores aritméticos "+" (más) y "-" (menos) pueden emplearse para las fechas. Por ejemplo: `SELECT SYSDATE – 5;` Devolvería la fecha correspondiente a 5 días antes de la fecha actual. Oracle tiene dos tipos de datos para manejar fechas:

`DATE` almacena fechas concretas incluyendo a veces la hora.
`TIMESTAMP` almacena un instante de tiempo +concreto que puede incluir fracciones de segundo.

Se pueden emplear estas funciones enviando como argumento el nombre de un campo de tipo fecha.
`SYSDATE` Devuelve la fecha y hora actuales. `SELECT SYSDATE FROM DUAL;` – 26/07/11
`SYSTIMESTAMP` Devuelve la fecha y hora actuales en formato TIMESTAMP.
`SELECT SYSTIMESTAMP FROM DUAL;` – 26-JUL-11 08.32.59,609000 PM +02:00
`ADD_MONTHS(fecha, n)` Añade a la fecha el número de meses indicado con n.
`SELECT ADD_MONTHS('27/07/11', 5) FROM DUAL;` – Resultado: 27/12/11
`MONTHS_BETWEEN(fecha1, fecha2)` Devuelve el núm. de meses entre fecha1 y fecha2.
`SELECT MONTHS_BETWEEN('12/07/11','12/03/11') FROM DUAL;` – 4
`LAST_DAY(fecha)` Último día del mes al que pertenece la fecha. El valor devuelto es tipo DATE
`SELECT LAST_DAY('27/07/11') FROM DUAL;` --Resultado: 31/07/11
`NEXT_DAY(fecha, d)` Indica el día que corresponde si añadimos a la fecha el día d. El día devuelto puede ser texto ('Lunes', Martes', ..) o el número del día de la semana (1=lunes, 2=martes, ..) dependiendo de la configuración.
`SELECT NEXT_DAY('31/12/11','LUNES') FROM DUAL;` – Resultado: 02/01/12
`EXTRACT(valor FROM fecha)` Extrae un valor de una fecha concreta. El valor puede ser day, month, year, hours, etc.
`SELECT EXTRACT(MONTH FROM SYSDATE) FROM DUAL;` – Resultado: 7

Funciones de conversión de un tipo de dato a otro

`TO_NUMBER(cad, formato)` Convierte textos en números. Se suele utilizar para dar un formato concreto a los números. Los formatos que podemos utilizar son los siguientes:

- `9` Posiciones numéricas. Si el número que se quiere visualizar contiene menos dígitos de los que se especifican en el formato, se rellena con blancos.
- `0` Visualiza ceros por la izquierda hasta completar la longitud del formato especificado.
- `$` Antepone el signo de dólar al número.
- `L` Coloca en la posición donde se incluya, el símbolo de la moneda local (se puede configurar en la base de datos mediante el parámetro NSL_CURRENCY)
- `S` Aparecerá el símbolo del signo.
- `D` Posición del símbolo decimal, que en español es la coma.
- `G` Posición del separador de grupo, que en español es el punto.

`TO_CHAR(d, formato)` Convierte un núm. o fecha d a cad. de caracteres, se utiliza para fechas normalmente ya que de número a texto se hace de forma implícita como hemos visto antes.
`TO_DATE(cad, formato)` Convierte textos a fechas. Podemos indicar el formato con el que queremos que aparezca. Parámetros para las funciones `TO_CHAR` y `TO_DATE`:

YY	Año en formato de dos cifras	YYYY	Año en formato de cuatro cifras
MM	Mes en formato de dos cifras	MON	Las tres primeras letras del mes
MONTH	Nombre completo del mes	Q	Semestre
DY	Día de la semana en tres letras	DAY	Día completo de la semana
DD	Día en formato de dos cifras	D	Día de la semana del 1 al 7
AM	Indicador a.m.	PM	Indicador p.m.
WW	Semana del año	MI	Minutos de 0 a 59
HH12	Hora de 1 a 12	HH24	Hora de 0 a 23

SS Segundos dentro del minuto SSSS Segundos dentro desde las 0 horas

Otras funciones: NVL y DECODE

Cualquier operación que se haga con un valor NULL devuelve un NULL. Por ejemplo, si se intenta dividir por NULL obtendremos NULL. También es posible que el resultado de una función nos de un valor nulo. Aparece entonces la necesidad de poder hacer algo con ellos.

`NVL(valor, expr1)` Si valor es NULL, devuelve expr1. expr1 debe ser d mismo tipo q valor.

```
SELECT NVL(DIRECC1, 'Sin dirección conocida') FROM EMPLEADOS;
```

`DECODE(expr1, cond1, valor1 [, cond2, valor2, ...], default )`

Esta función evalúa una expresión expr1, si se cumple la primera condición (cond1) devuelve el valor1, sino evalúa la siguiente condición y así hasta que una se cumpla. Si no se cumple ninguna condición se devuelve el valor por defecto que hemos llamado default.

Funciones de agregado (agrupamiento)

Este tipo consultas **toman un grupo de datos** (una columna) y **producen un único dato que resume el grupo**. No se corresponden con ningún valor de la tabla sino un **total calculado** sobre los datos de la tabla. Esto conlleva limitaciones.

El simple hecho de utilizar una función de agregado en una consulta la convierte en consulta de resumen. Todas las funciones de agregado tienen una estructura muy parecida: FUNCIÓN ([ALL| DISTINCT] Expresión) y debemos tener en cuenta que:

- La palabra ALL indica que se tienen que tomar todos los valores de la columna. Por defecto.
- La palabra DISTINCT indica que se considerarán todas las repeticiones del mismo valor como uno solo (considera valores distintos).
- El grupo de valores sobre el que actúa la función lo determina el resultado de la expresión que será el nombre de una columna o una expresión basada en una o varias columnas. Por tanto, en la expresión nunca puede aparecer ni una función de agregado ni una subconsulta.
- Todas las funciones se aplican a las filas del origen de datos una vez ejecutada la cláusula WHERE (si la tuviéramos).
- Todas las funciones (excepto COUNT) ignoran los valores NULL.
- Podemos encontrar una función de agrupamiento dentro de una lista de selección en cualquier sitio donde pudiera aparecer el nombre de una columna. Es por eso que puede formar parte de una expresión pero no se pueden anidar funciones de este tipo.
- No se pueden mezclar funciones de columna con nombres de columna ordinarios, aunque hay excepciones que veremos más adelante.

`SUM([ALL|DISTINCT] expresión)` `SELECT SUM(credito) FROM Usuarios;`

Devuelve la suma de los valores de la expresión. Sólo puede utilizarse con columnas cuyo tipo sea número. El resultado será del mismo tipo aunque puede tener una precisión mayor.

`COUNT([ALL|DISTINCT] expresión)` `SELECT COUNT(nombre) FROM Usuarios;`

Cuenta los elementos de un campo. Expresión contiene el nombre del campo a contar. Los operandos de expresión pueden incluir el nombre del campo, una constante o una función. Puede contar cualquier tipo de datos incluido texto. COUNT simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan. La función COUNT no cuenta los registros que tienen campos NULL a menos que **expresión** sea el carácter comodín **asterisco** (*): `SELECT COUNT(*) FROM Usuarios;`

```
SELECT COUNT(Nombre) FROM EMPLEADOS WHERE SEXO='M';
```

`MIN ([ALL| DISTINCT] expresión)` `SELECT MIN(credito) FROM Usuarios;`

Devuelve el valor mínimo de la expresión sin considerar los nulos (`NULL`).
En expresión podemos incluir el nombre de un campo de una tabla, una constante o una función (pero no otras funciones agregadas de SQL).

`MAX ([ALL| DISTINCT] expresión)` `SELECT MAX (credito) FROM Usuarios;`

Devuelve el valor máximo de la expresión sin considerar los nulos (`NULL`).
En expresión podemos incluir el nombre de un campo de una tabla, una constante o una función (pero no otras funciones agregadas de SQL).

AVG ([ALL| DISTINCT] expresión)

```
SELECT AVG(SALARIO_MIN), AVG(SALARIO_MAX) FROM TRABAJOS;
```

Devuelve el promedio de los valores de un grupo, para ello se omiten los nulos (NULL).
El grupo de valores será el que se obtenga como resultado de la expresión y ésta puede ser un nombre de columna o una expresión basada en una columna o varias de la tabla.
Se aplica a campos tipo número y el tipo de dato del resultado puede variar según las necesidades del sistema para representar el valor.

VAR ([ALL| DISTINCT] expresión)

Devuelve la varianza estadística de todos los valores de la expresión.
Admite únicamente columnas numéricas. Los valores nulos (NULL) se omiten.

STDEV ([ALL| DISTINCT] expresión)

Devuelve la desviación típica estadística de todos los valores de la expresión.
Admite únicamente columnas numéricas. Los valores nulos (NULL) se omiten.

Agrupamiento de registros

En muchas ocasiones en las que utilizamos consultas de resumen nos va a interesar calcular **totales parciales**, es decir, **agrupados según un determinado campo**. En lugar de una única fila de resultados necesitaremos una fila por cada campo.

```
SELECT columna1, columna2, …
FROM tabla1, tabla2, …
WHERE condición1, condición2, …
GROUP BY columna1, columna2, …
HAVING condición
ORDER BY ordenación;
```

En la cláusula GROUP BY se colocan las columnas por las que vamos a agrupar.
En HAVING la condición que han de cumplir los grupos para que se realice la consulta.

Orden en el que se ejecutan las cláusulas. Importante:

1.- WHERE que filtra las filas según las condiciones que pongamos.
2.- GROUP BY que crea una tabla de grupos nueva.
3.- HAVING filtra los grupos.
4.- ORDER BY que ordena o clasifica la salida.

Las columnas que aparecen en el SELECT y que no aparezcan en la cláusula GROUP BY deben tener una función de agrupamiento. Si esto no se hace así producirá un error. Otra opción es poner en la cláusula GROUP BY las mismas columnas que aparecen en SELECT.

```
SELECT provincia, SUM(credito) FROM Usuarios GROUP BY provincia;
```

Si estuviéramos interesados en la suma de créditos agrupados por provincia pero únicamente de las provincias de Cádiz y Badajoz nos quedaría:

```
SELECT provincia, SUM(credito) FROM Usuarios GROUP BY provincia
HAVING provincia='CÁDIZ' OR provincia='BADAJOZ';
```

Consultas multitablas

Si distribuíamos la información en varias tablas relacionadas por algún campo común, también es posible que tengamos que consultar datos que se encuentren distribuidos por distintas tablas.

Composiciones internas

Si combinamos dos o más tablas (en SELECT con FROM tablas) sin ninguna restricción, el resultado será un producto cartesiano (SQL coge una fila de una tabla y la asocia con todos y cada uno de las filas de la otra tabla, independientemente de que tengan relación o no), y esto puede ser costoso en términos de procesamiento de esos datos.
Necesitaremos **discriminar** de alguna forma para que únicamente aparezcan filas de una tabla que estén relacionadas con la otra tabla. A esto se le llama **asociar tablas** (JOIN).

Para hacer una composición interna se parte de un producto cartesiano y se eliminan aquellas filas que no cumplen la condición de composición. Lo importante es emparejar los campos que han de tener valores iguales. Las reglas para las composiciones son:

- Pueden combinarse tantas tablas como se desee.
- El criterio de combinación puede estar formado por más de una pareja de columnas.
- En SELECT pueden citarse columnas de ambas tablas, condicionen o no, la combinación.
- Si hay columnas con el mismo nombre en las distintas tablas, deben identificarse especificando la tabla de procedencia o utilizando un alias de tabla.

Las columnas que aparecen en la cláusula WHERE se denominan columnas de emparejamiento ya que son las que permiten emparejar las filas de las dos tablas. Éstas no tienen por qué estar incluidas en la lista de selección. Emparejaremos tablas que estén relacionadas entres sí y además, una de las columnas de emparejamiento será clave principal en su tabla. Cuando emparejamos campos debemos especificar de la siguiente forma: Tabla1.Camporelacion1 = Tabla2.Camporelacion2.

Podemos combinar una tabla consigo misma pero se debe poner de manera obligatoria un alias a uno de los nombres de la tabla que vas a repetir.

Ejemplo: si queremos obtener el historial laboral de los empleados con nombres y apellidos, la fecha en que entraron a trabajar y la fecha de fin de trabajo y si ya no continúan en la empresa:

```
SELECT Nombre, Apellido1, Apellido2, Fecha_inicio, Fecha_fin
FROM EMPLEADOS, HISOTRIAL_LABORAL
WHERE HISTORIAL_LABORAL.Empleado_DNI= EMPLEADOS.DNI;
```

Obtener el historial con nombre de departa., nombre y apellidos del empleado de todos los departa.:

```
SELECT Nombre_Dpto, Nombre, Apellido1, Apellido2
FROM DEPARTAMENTOS, EMPLEADOS, HISTORIAL_LABORAL
WHERE EMPLEADOS.DNI= HISTORIAL_LABORAL. EMPLEADO_DNI
AND HISTORIAL_LABORAL.DPTO_COD = DEPARTAMENTOS. DPTO_COD;
```

Obtener un listado con el histórico laboral de un empleador cuyo DNI sea '12345'. En dicho listado interesa conocer el nombre del puesto, así como el rango salarial.

```
SELECT T.NOMBRE_TRAB, T.SALARIO_MIN, T.SALARIO_MAX,
HL.EMPLEADO_DNI, HL.TRAB_COD, HL.FECHA_INICIO, HL.FECHA_FIN,
HL.DPTO_COD, HL.SUPERVISOR_DNI
FROM TRABAJOS T, HISTORIAL_LABORAL HL
WHERE T.TRABAJO_COD=HL.TRAB_COD AND EMPLEADO_DNI='12345';
```

Composiciones externas

Puede ser necesario seleccionar algunas filas de una tabla aunque éstas no tengan correspondencia con las filas de la otra tabla. Añadiremos un signo más entre paréntesis (+) en la igualdad entre campos que ponemos en la cláusula WHERE. El carácter (+) irá detrás del nombre de la tabla en la que deseamos aceptar valores nulos.
Ejemplo: Tenemos guardadas en dos tablas la información de los empleados de la empresa (**Cod_empleado, Nombre, Apellidos, salario y Cod_dpto**) por otro lado los departamentos (**Codigo_dep, Nombre**) de esa empresa. Recientemente se ha remodelado la empresa y se han creado un par de departamentos más pero no se les ha asignado los empleados. Si tuviéramos que obtener un informe con los datos de los empleados por departamento, seguro que deben aparecer esos departamentos aunque no tengan empleados. En este ejemplo, la cláusula WHERE es `Cod_dpto (+)= Codigo_dep` ya que es en la tabla empleados donde aparecerán valores nulos.

Obtener un listado con los nombres de todos los departamentos, -aunque no tengan asignado ningún jefe-, y sus jefes, con sus datos personales.

```
SELECT  D.NOMBRE_DPTO, D.JEFE, E.NOMBRE, E.APELLIDO1, E.APELLIDO2
     FROM DEPARTAMENTOS D, EMPLEADOS E
     WHERE D.JEFE(+) = E.DNI;
```

Composiciones en la versión SQL99

SQL incluye en esta versión mejoras de la sintaxis a la hora de crear composiciones en consultas:

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

`CROSS JOIN`: creará producto cartesiano de filas de las tablas, podemos olvidarnos de WHERE.
`NATURAL JOIN`: detecta automáticamente las claves de unión, basándose en el nombre de la columna que coincide en ambas tablas. Se requerirá que las columnas de unión tengan el mismo nombre en cada tabla. Esto funcionará incluso si no están definidas las claves primarias o ajenas.
`JOIN USING`: las tablas pueden tener más de un campo para relacionar y no siempre queremos que se relacionen por todos los campos. Esta cláusula permite establecer relaciones indicando qué campo o campos comunes se quieren utilizar para ello.
`JOIN ON`: se utiliza para unir tablas en la que los nombres de columna no coinciden en ambas tablas o se necesita establecer asociaciones más complicadas.
`OUTER JOIN`: se puede eliminar el uso del signo (+) para composiciones externas utilizándolo.
`LEFT OUTER JOIN`: es una composición externa izquierda, todas las filas de la tabla de la izquierda se devuelven, aunque no haya ninguna columna correspondiente en las tablas combinadas.
`RIGTH OUTER JOIN`: es una composición externa derecha, todas las filas de la tabla de la derecha se devuelven, aunque no haya ninguna columna correspondiente en las tablas combinadas.
`FULL OUTER JOIN`: es una composición externa en la que se devolverán todas las filas de los campos no relacionados de ambas tablas.

Ejemplos que ya hemos vistos adaptados:
Obtener el historial laboral de los empleados incluyendo nombres y apellidos de los empleados, la fecha en que entraron a trabajar y la fecha de fin de trabajo si ya no continúan en la empresa. Es una consulta de composición interna, luego utilizaremos `JOIN ON`:

```
SELECT E.Nombre, E.Apellido1, E.Apellido2, H.Fecha_inicio,
H.Fecha_fin FROM EMPLEADOS E JOIN HISTORIAL_LABORAL H ON
(H.Empleado_DNI = E.DNI);
```

Obtener un listado con los nombres de los distintos departamentos y sus jefes con sus datos personales. Ten en cuenta que deben aparecer todos los departamentos aunque no tengan asignado ningún jefe. Aquí estamos ante una composición externa, luego podemos utilizar `OUTER JOIN`:

```
SELECT  D.NOMBRE_DPTO, D.JEFE, E.NOMBRE, E.APELLIDO1,
E.APELLIDO2 FROM DEPARTAMENTOS D LEFT OUTER JOIN EMPLEADOS E
ON (D.JEFE = E.DNI);
```

Otras consultas multitablas: Unión, Intersección y diferencia de consultas

Podemos tener varias tablas con los mismos datos guardados para distintos registros y puede que queramos unirla en una única tabla.
`UNION`: combina filas del primer `SELECT` con las de otro `SELECT`, desapareciendo las duplicadas.
`INTERSECT`: examina las filas de dos `SELECT` y devolverá aquellas que aparezcan sólo en ambos conjuntos. Las filas duplicadas se eliminarán.
`MINUS`: devuelve aquellas filas que están en el primer `SELECT` pero no en el segundo. Las filas duplicadas del primer `SELECT` se reducirán a una antes de comenzar la comparación.
Estas operaciones se pueden combinar anidadas, pero es conveniente utilizar paréntesis para indicar que operación quieres que se haga primero.

Es **importante** usar en los dos SELECT el mismo número y tipo de columnas y en el mismo orden.

Obtener los nombres y ciudades de todos los proveedores y clientes de Alemania.

```
SELECT NombreCia, Ciudad FROM  PROVEEDORES WHERE Pais =
'Alemania' UNION SELECT NombreCia, Ciudad  FROM CLIENTES
WHERE Pais = 'Alemania';
```

Una academia de idiomas da clases de inglés, frances y portugues; almacena los datos de los alumnos en tres tablas distintas una llamada "ingles", en una tabla denominada "frances" y los que aprenden portugues en la tabla "portugues". La academia necesita el nombre y domicilio de todos los alumnos que cursan los tres idiomas para enviarles iformación sobre los exámene**s.**

```
SELECT nombre, domicilio FROM ingles INTERSECT
SELECT nombre, domicilio FROM frances INTERSECT
SELECT nombre, domicilio FROM portugues;
```

Ahora la academia necesita el nombre y domicilio solo de todos los alumnos que cursan inglés (no quiere a los que ya cursan portugués pues va a enviar publicidad referente al curso de portugués).

```
SELECT nombre, domicilio FROM INGLES MINUS
SELECT nombre,domicilio FROM PORTUGUES;
```

Subconsultas

A veces hay que utilizar en una consulta los resultados de otra que llamaremos subconsulta:

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

La subconsulta puede ir dentro de las cláusulas WHERE, HAVING o FROM. Deben ir entre paréntesis y a la derecha del operador. El OPERADOR puede ser >, <, >=, <=, !=, = o IN.
Las subconsultas con estos operadores devuelven un único valor. Si devolviera más generaría error.
Tipos de datos que devuelve la subconsulta y columna con la que se compara ha de ser el mismo.

Ejemplo: Nombre de los empleados y el sueldo de aquellos que cobran menos que Ana.

```
SELECT Nombre_empleado, sueldo FROM EMPLEADOS
WHERE SUELDO < (SELECT SUELDO FROM EMPLEADOS
WHERE Nombre_emple = 'Ana');
```

Podemos comparar un valor con varios, es decir, si queremos que la subconsulta devuelva más de un valor y comparar el campo que tenemos con dichos valores.
Imagina que queremos ver si el sueldo de un empleado que es administrativo es mayor o igual que el sueldo medio de otros puestos en la empresa. Para saberlo deberíamos calcular el sueldo medio de las demás ocupaciones que tiene la empresa y éstos compararlos con la de nuestro empleado.
Como ves, el resultado de la subconsulta es más de una fila. ¿Qué hacemos?

Cuando el resultado de la subconsulta es más de una fila, SQL utiliza **instrucciones especiales** entre el operador y la consulta. Estas instrucciones son:

- `ANY`. Compara con cualquier fila de la consulta. La instrucción es válida si hay un registro en la subconsulta que permite que la comparación sea cierta.
- `ALL`. Compara con todas las filas de la consulta. La instrucción resultará cierta si es cierta toda la comparación con los registros de la subconsulta.
- `IN`. No utiliza comparador, comprueba si el valor está en el resultado de la subconsulta.
- `NOT IN`. Comprueba si un valor no se encuentra en una subconsulta.

En la siguiente consulta obtenemos el empleado que menos cobra:

```
SELECT nombre, sueldo FROM EMPLEADOS
WHERE sueldo <= ALL (SELECT sueldo FROM EMPLEADOS);
```