Unidad Didáctica 6:

REALIZACIÓN DE CONSULTAS

1. Consulta de datos

El proceso más importante que podemos llevar a cabo en una base de datos es la consulta de los datos. De nada serviría una base de datos si no pudiéramos consultarla. Es además la operación que efectuaremos con mayor frecuencia.

Para consultar la información SQL pone a nuestra disposición la sentencia **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***).
- Realizar cálculos sobre los datos.
- Agrupar datos.

2. La sentencia SELECT

El formato de la sentencia SELECT es:

```
SELECT [* | ALL | DISTINCT ] <nombre_campo> [{,<nombre_campo>}]
FROM <nombre_tabla>|<nombre_vista> [{,<nombre_tabla>|<nombre_vista>}]
[WHERE <condicion> [{ AND|OR <condicion>}]]
[GROUP BY <nombre_campo> [{,<nombre_campo >}]]
[HAVING <condicion>[{ AND|OR <condicion>}]]
[ORDER BY <nombre_campo>|<indice_campo> [ASC | DESC]
      [{,<nombre_campo>|<indice_campo>[ASC | DESC ]}]];
```

Veamos por partes que quiere decir cada una de las partes que conforman la sentencia.

- **SELECT,** palabra clave que indica que la sentencia de SQL que queremos ejecutar es de selección.
- ***,** el asterisco significa que se seleccionan todas las columnas.
- **ALL,** indica que queremos seleccionar todas las filas. No se utiliza puesto que es la opción por defecto.
- **DISTINCT,** suprime las filas duplicadas.

- **FROM,** indica la tabla (o tablas) desde la que queremos recuperar los datos. En el caso de que exista más de una tabla se denomina a la consulta "consulta combinada" o "join". En las consultas combinadas es necesario aplicar una condición de combinación a través de una cláusula `WHERE`.
- **WHERE,** especifica una condición que debe cumplirse para que los datos sean devueltos por la consulta. Admite los operadores lógicos `AND` y `OR`.
- **GROUP BY,** especifica la agrupación que se da a los datos. Se usa siempre en combinación con funciones agregadas.
- **HAVING,** especifica una condición que debe cumplirse para que los datos sean devueltos por la consulta. Su funcionamiento es similar al de `WHERE` pero aplicado al conjunto de resultados devueltos por la consulta. Debe aplicarse siempre junto a `GROUP BY` y la condición debe estar referida a los campos contenidos en ella.
- **ORDER BY,** presenta el resultado ordenado por las columnas indicadas. El orden puede expresarse con `ASC` (orden ascendente) y `DESC` (orden descendente). El valor predeterminado es `ASC`.

Para poder ver reflejado el resultado de algunas consultas que iremos haciendo a partir de este momento, vamos a suponer que disponemos de una base de datos con las siguientes tablas y la siguiente información almacenada en cada una de ellas:

departamentos

clientes

codigo_cli	nombre_cli	nif	direccion	ciudad	telefono
10	ECIGSA	38.567.893-C	Aragón 11	Barcelona	NULL
20	CME	38.123.898-E	Valencia 22	Girona	972.23.57.21
30	ACME	36.432.127-A	Mallorca 33	Lleida	973.23.45.67
40	JGM	38.782.345-B	Rosellón 44	Tarragona	977.33.71.43

empleados						
codigo_empleado	nombre_empl	apellido_empl	sueldo	nombre_dep	ciudad_dep	num_proyec
1	María	Puig	1,0E+5	DIR	Girona	1
2	Pedro	Mas	9,0E+4	DIR	Barcelona	4
3	Ana	Ros	7,0E+4	DIS	Lleida	3
4	Jorge	Roca	7,0E+4	DIS	Barcelona	4
5	Clara	Blanc	4,0E+4	PROG	Tarragona	1
6	Laura	Tort	3,0E+4	PROG	Tarragona	3
7	Rogelio	Salt	4,0E+4	NULL	NULL	4
8	Sergio	Grau	3,0E+4	PROG	Tarragona	NULL

Figura 3. Tabla EMPLEADOS

proyectos						
codigo_proyec	nombre_proyec	precio	fecha_inicio	fecha_prev_fin	fecha_fin	codigo_cliente
1	GESCOM	1,0E+6	1-1-98	1-1-99	NULL	10
2	PESCI	2,0E+6	1-10-96	31-3-98	1-5-98	10
3	SALSA	1,0E+6	10-2-98	1-2-99	NULL	20
4	TINELL	4,0E+6	1-1-97	1-12-99	NULL	30

Figura 4. Tabla PROYECTOS

3. Seleccionar columnas en una consulta

Podemos visualizar todas las columnas de datos en una tabla si se escribe el asterisco (*) detrás de la palabra clave `SELECT`. También se puede visualizar todas las columnas de una tabla si se enumeran todas las columnas después de la palabra clave `SELECT`.

Ejemplo: Visualizar todas las columnas de la tabla CLIENTES.

```
SELECT *
FROM clientes;
```

```
SELECT codigo_cli,nombre_cli, nif, direccion, ciudad,
telefono
FROM clientes;
```

La respuesta a esta consulta sería:

codigo_cli	nombre_cli	nif	direccion	ciudad	telefono
10	ECIGSA	38.567.893-C	Aragón 11	Barcelona	NULL
20	CME	38.123.898-E	Valencia 22	Girona	972.23.57.21
30	ACME	36.432.127-A	Mallorca 33	Lleida	973.23.45.67
40	JGM	38.782.345-B	Rosellón 44	Tarragona	977.33.71.43

Podemos utilizar la sentencia `SELECT` para visualizar columnas específicas de la tabla si se indican los nombres de las columnas separadas por comas. Es necesario especificar el orden en que quiero que aparezcan las columnas en el resultado.

Si hubiésemos querido ver sólo el código, el nombre, la dirección y la ciudad, habríamos hecho:

```
SELECT codigo_cli, nombre_cli, direccion, ciudad
FROM clientes;
```

Y habríamos obtenido la respuesta siguiente:

codigo_cli	nombre_cli	direccion	ciudad
10	ECIGSA	Aragón 11	Barcelona
20	CME	Valencia 22	Girona
30	ACME	Mallorca 33	Lleida
40	JGM	Rosellón 44	Tarragona

Cuando se muestra el resultado de una consulta, normalmente se utiliza el nombre de la columna seleccionada como cabecera de columna. Es posible que esta cabecera no sea descriptiva y, por tanto, sea difícil de comprender. Podemos cambiar la cabecera de una columna mediante un **alias de columna**. Para ello, tenemos que especificar el alias tras la columna en la sentencia `SELECT` utilizando un espacio como separador o la palabra reservada `AS`. Por defecto, las cabeceras de alias aparecen en mayúsculas. Si el alias contiene espacios o caracteres especiales (como # o $), o es sensible a mayúsculas/minúsculas, escribiremos el alis entre comillas dobles (" ").

```
SELECT codigo_cli AS codigo, nombre_cli "NOMBRE DEL
CLIENTE"
FROM clientes;
```

4. Expresiones aritméticas

Una expresión aritmética puede contener nombres de columna, valores numéricos constantes y operadores 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 un nueva columna.

Los operadores aritméticos se pueden utilizar en cualquier cláusula de una sentencia SQL excepto en la cláusula `FROM`.

Ejemplo:

```
SELECT nombre, precio,precio*1.16
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*1.16  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.

```
SELECT apellido_empl, 12 * sueldo+100
FROM empleados;

es diferente de :

SELECT apellido_empl, 12 * (sueldo+100)
FROM empleados;
```

Cuando una expresión aritmética se calcula sobre valores NULL, el resultado es el propio valor NULL. Por ejemplo, cuando intentamos dividir entre cero, recibiremos un mensaje de error. Sin embargo, si dividimos un numero entre un valor nulo, el resultado será nulo o desconocido.

5. Operador de concatenación

Todas las bases de datos incluyen algún operador para encadenar textos. En Oracle son los signos **||**. Las columnas a cada lado del operador se combinan para hace una única columna se resultados.

Ejemplo :

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

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

6. Cláusula WHERE.

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.

Veamos un ejemplo en el que pedimos "los códigos de los empleados que trabajan en el proyecto número 4":

```
SELECT codigo_empl
FROM empleados
WHERE num_proyec = 4;
```

La respuesta a esta consulta sería la que podéis ver a continuación:

codigo_empl
2
4
7

En la clausula `WHERE` las cadenas de caracteres y las fechas deben escribirse entre comillas simples (' '), pero no las constantes numéricas. Las búsquedas de caracteres son sensibles a mayúsculas/minúsculas.

Para definir las condiciones en la cláusula `WHERE`, podemos utilizar alguno de los operadores de los que dispone el SQL, que son los siguientes:

6.1. Operadores de comparación:

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

Los operadores de comparación se utilizan en condiciones que comparan una expresión con otro valor o expresión. No se puede utilizar un alias en la cláusula `WHERE`.

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. Sólo que es un orden alfabético estricto, es decir, el orden de los caracteres en la tabla de códigos.

6.2. Operadores lógicos:

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.

Es posible construir condiciones complejas uniendo dos o más condiciones simples a través de los operadores lógicos **AND** y **OR**.

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;
```

6.3. Operador BETWEEN:

El operador **BETWEEN** nos permite obtener datos que se encuentren en un rango de valores. Debemos especificar un límite inferior y otro límite superior. Ambos límites están incluidos en el rango.

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE columna [NOT] BETWEEN límite1 AND límite2;
```

Un ejemplo en el que se pide "Los códigos de los empleados que ganan entre 20.000 y 50.000 euros anuales" sería:

```
SELECT codigo_empl
FROM empleados
WHERE sueldo BETWEEN 2.0E+4 AND 5.0E+4;
```

La respuesta a esta consulta sería

codigo_empl
5
6
7
8

6.4. Operador IN:

Para comprobar si un valor coincide con los elementos de una lista utilizaremos el operador **IN**.

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE columna [NOT] IN (valor1, ..., valorN);
```

Ejemplo:

Queremos saber el nombre de todos los departamentos que se encuentran en las ciudades de Lleida o Tarragona":

```
SELECT nombre_dep, ciudad_dep
FROM departamentos
WHERE ciudad_dep IN ('Lleida', 'Tarragona');
```

6.5. Operador LIKE:

Para comprobar si una columna de tipo carácter cumple alguna propiedad determinada, podemos usar **LIKE.**

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE columna [NOT] LIKE  cadena;
```

LIKE utiliza una cadena que puede contener estos símbolos:

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

Ejemplos:

Nombres de empleados que empiezan por la letra J:

```
SELECT codigo_empl, nombre_empl
FROM empleados
WHERE nombre_empl LIKE 'J%';
```

codigo_empl	nombre_empl
4	Jorge

Proyectos que empiezan por S y tienen cinco letras:

```
SELECT codigo_proyec
FROM proyectos
WHERE nombre_proyec LIKE 'S_ _ _ _';
```

codigo_proyec
3

6.6. Operador IS NULL:

Para comprobar si un valor es nulo utilizaremos **IS NULL**, y para averiguar si no lo es, **IS NOT NULL**. El formato es:

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE columna IS [NOT] NULL;
```

Ejemplo:

Queremos saber el código y el nombre de todos los empleados que no están asignados a ningún proyecto:

```
SELECT codigo_empl, nombre_empl
FROM empleados
WHERE num_proyec IS NULL;
```

codigo_empl	nombre_empl
8	Sergio

6.7. Precedencia de operadores:

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

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

7. Cláusula DISTINCT

Si queremos que en una consulta nos aparezcan las filas resultantes sin repeticiones, es preciso poner la palabra clave `DISTINCT` inmediatamente después de `SELECT`. El formato de `DISTINCT` es:

```
SELECT DISTINCT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
[WHERE condiciones];
```

Ejemplo:

Si quisiéramos ver qué sueldos se están pagando en nuestra empresa, podríamos hacer:

```
SELECT DISTINCT sueldo
FROM empleados;
```

sueldo
3,0E+4
4,0E+4
7,0E+4
9,0E+4
1,0E+5

8. Cláusula ORDER BY

Si se desea que, al hacer una consulta, los datos aparezcan en un orden determinado, es preciso utilizar la cláusula `ORDER BY` en la sentencia `SELECT`, que presenta el siguiente formato:

```
SELECT nombre_columnas_a seleccionar
FROM tabla_a_consultar
[WHERE condiciones]
ORDER BY columna_según_la_cual_se_quiere_ordenar [DESC]
[, col_ordenación [DESC]...];
```

Ejemplo:
Imaginemos que queremos consultar los nombres de los empleados ordenados según el sueldo que ganan, y si ganan el mismo sueldo, ordenados alfabéticamente por el nombre:

```
SELECT codigo_empl, nombre_empl, apellido_empl, sueldo
FROM empleados
ORDER BY sueldo, nombre_empl;
```

Esta consulta daría la respuesta siguiente:

codigo_empl	nombre_empl	apellido_empl	sueldo
6	Laura	Tort	3,0E+4
8	Sergio	Grau	3,0E+4
5	Clara	Blanc	4,0E+4
7	Rogelio	Salt	4,0E+4
3	Ana	Ros	7,0E+4
4	Jorge	Roca	7,0E+4
2	Pedro	Mas	9,0E+4
1	María	Puig	1,0E+5

Si no se especifica nada más, se seguirá un orden ascendente, pero si se desea seguir un orden descendente es necesario añadir DESC detrás de cada factor de ordenación expresado en la cláusula ORDER BY.

También se puede explicitar un orden ascendente poniendo la palabra clave ASC (opción por defecto).
El orden por defecto es ascendente:

- Los valores numéricos se muestran con los valores más pequeños primero.
- Los valores de fecha se muestran con la fecha anterior primero.
- Los valores de caracteres se muestran en orden alfabético.
- Los valores nulos se muestran los últimos cuando el orden es ascendente y los primeros para los descendentes.

También podemos utilizar un alias de columna en la cláusula `ORDER BY`.

Ejemplo:

```
SELECT codigo_empl, sueldo * 12 AS salario_anual
FROM empleados
ORDER BY salario_anual;
```

9. Funciones

Todos los SGBD implementan funciones para facilitar la creación de consultas complejas. Esas funciones dependen del SGBD que utilicemos, las que aquí se comentan son algunas de las que se utilizan con Oracle.

Todas las funciones devuelven un resultado que procede de un determinado cálculo. La mayoría de funciones precisan que se les envíe datos de entrada (**parámetros** o **argumentos**) que son necesarios para realizar el cálculo de la función. Este resultado, lógicamente depende de los parámetros enviados. Dichos parámetros se pasan entre paréntesis. De tal manera que la forma de invocar a una función es:

```
nombreFunción[(parámetro1[, parámetro2,...])]
```

Si una función no precisa parámetros (como **SYSDATE**) no hace falta colocar los paréntesis.

En realidad hay dos tipos de funciones:

- Funciones que operan con datos de la misma fila.
- Funciones que operan con datos de varias filas diferentes (funciones de agrupación).

En este apartado se tratan las funciones del primer tipo (más adelante se comentan las de agrupación).

Nota: tabla DUAL (Oracle)

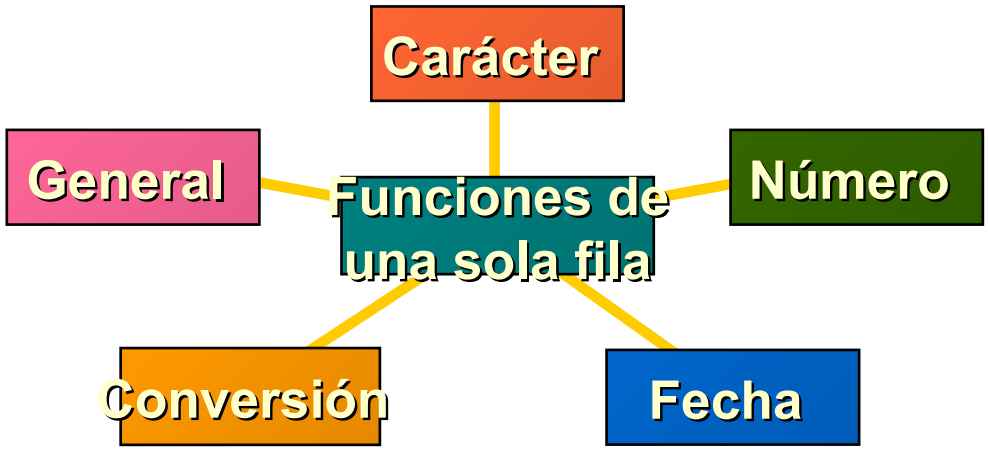
Oracle proporciona una tabla llamada dual con la que se permiten hacer pruebas. Esa tabla tiene un solo campo (llamado **DUMMY**) y una sola fila de modo que es posible hacer pruebas.

Por **ejemplo** la consulta:

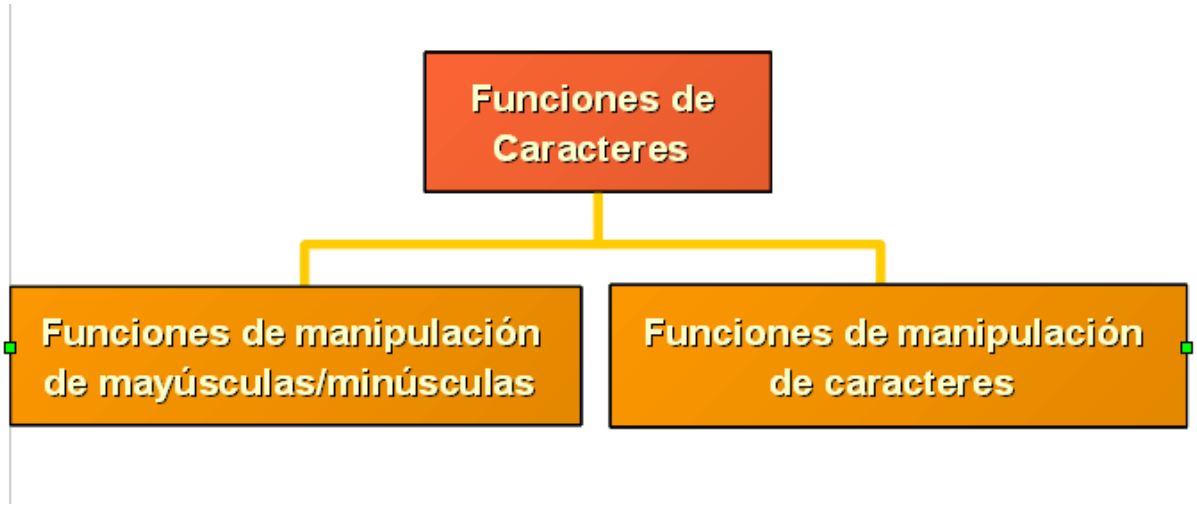
```
SELECT SQRT(5)
FROM DUAL;
```

Muestra una tabla con el contenido de ese cálculo (la raíz cuadrada de 5). DUAL es una tabla interesante para hacer pruebas.

En los siguientes apartados se describen algunas de las funciones más interesantes.



9.1. Funciones de caracteres



LOWER
UPPER
INITCAP

CONCAT
LENGTH
TRIM

SUBSTR
REPLACE
LPAD/RPAD ...

Funciones de conversión del texto a mayúsculas y minúsculas:

Función	Descripción
LOWER(texto)	Convierte el texto a minúsculas (funciona con los caracteres españoles)
UPPER(texto)	Convierte el texto a mayúsculas
INITCAP(texto)	Coloca la primera letra de cada palabra en mayúsculas

Funciones de manipulación de caracteres:

Función	Descripción
RTRIM(texto)	Elimina los espaciosa la derecha del texto
LTRIM(texto)	Elimina los espacios a la izquierda que posea el texto
TRIM(texto)	Elimina los espacios en blanco a la izquierda y la derecha del texto y los espacios dobles del interior.
TRIM(caracteres FROM texto)	Elimina del texto los caracteres indicados. Por ejemplo TRIM('h' FROM nombre) elimina las haches de la columna *nombre* que estén a la izquierda y a la derecha
SUBSTR(texto,n[,m])	Obtiene los *m* siguientes caracteres del texto a partir de la posición *n* (si *m* no se indica se cogen desde *n* hasta el final).
LENGTH(texto)	Obtiene el tamaño del texto
INSTR(texto, textoBuscado [,posInicial [, nAparición]])	Obtiene la posición en la que se encuentra el texto buscado en el texto inicial. Se puede empezar a buscar a partir de una posición inicial concreta e incluso indicar el número de aparición del texto buscado. Ejemplo, si buscamos la letra *a* y ponemos 2 en
TRANSLATE(texto, caracteresACambiar, caracteresSustitutivos)	Potentísima función que permite transformar caracteres. Los *caracteresACambiar* son los caracteres que se van a cambiar, los *caracteresSustitutivos* son los caracteres que
LPAD(texto, anchuraMáxima, [caracterDeRelleno]) RPAD(texto, anchuraMáxima, [caracterDeRelleno])	Rellena el texto a la izquierda (LPAD) o a la derecha (RPAD) con el carácter indicado para ocupar la anchura indicada. Si el texto es más grande que la anchura indicada, el texto se recorta. Si no se indica carácter de relleno se rellenaré el espacio marcado con espacios en blanco. Ejemplo: LPAD('Hola',10,'-') da como resultado ------*Hola*
REVERSE(texto)	Invierte el texto (le da la vuelta)

Algunos ejemplos:

Función	Resultado
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD(salary,10,'*')	*****24000
RPAD(salary, 10, '*')	24000*****
TRIM('H' FROM 'HelloWorld')	elloWorld

9.2. Funciones numéricas

Funciones de redondeo:

Función	Descripción
ROUND(n,decimales)	Redondea el número al siguiente número con el número de decimales indicado más cercano. ROUND(8.239,2) devuelve 8.24
TRUNC(n,decimales)	Los decimales del número se cortan para que sólo aparezca el número de decimales indicado

Funciones matemáticas:

MOD(n1,n2)	Devuelve el resto resultado de dividir *n1* entre *n2*
POWER(valor,exponente)	Eleva el valor al exponente indicado
SQRT(n)	Calcula la raíz cuadrada de n
SIGN(n)	Devuelve 1 si *n* es positivo, cero si vale cero y -1 si es negativo
ABS(n)	Calcula el valor absoluto de *n*
EXP(n)	Calcula e^n, es decir el exponente en base *e* del número n
LN(n)	Logaritmo neperiano de *n*
LOG(n)	Logaritmo en base 10 de *n*
SIN(n)	Calcula el seno de *n* (*n* tiene que estar en radianes)
COS(n)	Calcula el coseno de *n* (*n* tiene que estar en radianes)
TAN(n)	Calcula la tangente de *n* (*n* tiene que estar en radianes)
ACOS(n)	Devuelve en radianes el arco coseno de *n*
ASIN(n)	Devuelve en radianes el arco seno de *n*
ATAN(n)	Devuelve en radianes el arco tangente de *n*
SINH(n)	Devuelve el seno hiperbólico de *n*
COSH(n)	Devuelve el coseno hiperbólico de *n*
TANH(n)	Devuelve la tangente hiperbólica de *n*

9.3. Funciones de fecha

Obtener la fecha y la hora actual:

Función	Descripción
SYSDATE	Obtiene la fecha y hora actuales
SYSTIMESTAMP	Obtiene la fecha y hora actuales en formato TIMESTAMP

Calcular fechas:

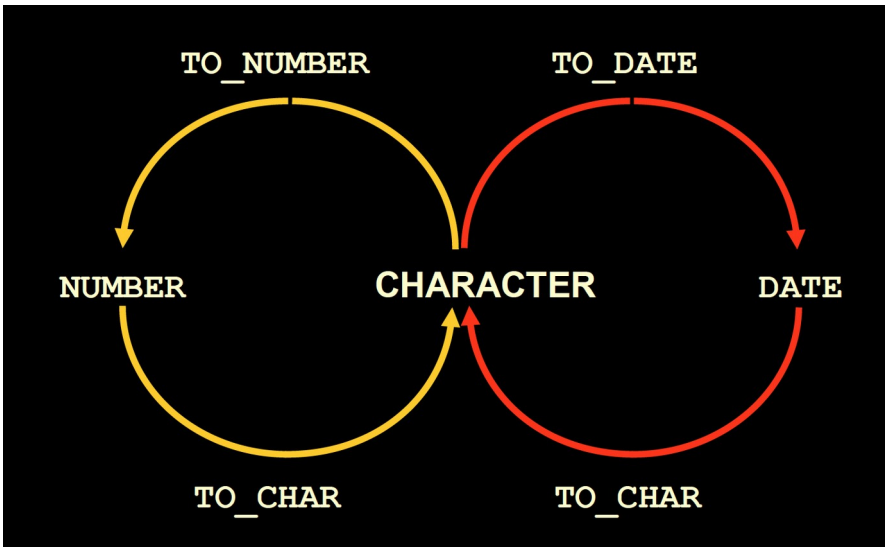
Función	Descripción
ADD_MONTHS(fecha,n)	Añade a la fecha el número de meses indicado por *n*
MONTHS_BETWEEN(fecha1, fecha2)	Obtiene la diferencia en meses entre las dos fechas (puede ser decimal)
NEXT_DAY(fecha,día)	Indica cual es el día que corresponde a añadir a la fecha el día indicado. El día puede ser el texto '*Lunes*', '*Martes*', '*Miércoles*',... (si la configuración está en español) o el número de día de la semana (1=lunes, 2=martes,...)
LAST_DAY(fecha)	Obtiene el último día del mes al que pertenece la fecha. Devuelve un valor DATE
EXTRACT(valor FROM fecha)	Extrae un valor de una fecha concreta. El valor puede ser day (día), month (mes), year (año), etc.
GREATEST(fecha1, fecha2,..)	Devuelve la fecha más moderna la lista
LEAST(fecha1, fecha2,..)	Devuelve la fecha más antigua la lista
ROUND(fecha [,'formato']	Redondea la fecha al valor de aplicar el formato a la fecha. El formato puede ser: 'YEAR' Hace que la fecha refleje el año completo 'MONTH' Hace que la fecha refleje el mes completo más cercano a la fecha 'HH24' Redondea la hora a las 00:00 más cercanas 'DAY' Redondea al día más cercano
TRUNC(fecha [formato])	Igual que el anterior pero trunca la fecha en lugar de redondearla.

9.4. Funciones de conversión

Para las asignaciones, Oracle Server puede convertir automáticamente lo siguiente:

De	A
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

Hay veces que necesitamos hacer conversiones explicitas de tipos de datos:



La función TO_CHAR nos permite convertir una fecha o un número a formato cadena de caracteres:
TO_CHAR(date, 'format_model')

TO_CHAR(number, 'format_model')

Para convertir una cadena de caracteres a formato numérico se usa la función TO_NUMBER:
TO_NUMBER(char [, 'format_model'])
con el 'format_model' le indicamos la estructura del char.

Si se quiere convertir una cadena de caracteres en formato de fecha se utilizará la función:
TO_DATE(char [, 'format_model'])
con el 'format_model' le indicamos la estructura del char.

Estos son distintos elementos que se pueden usar para construir el 'format_model' de fecha:

YYYY	**Año completo en números**
YEAR	**Años en letra**
MM	**Valor de dos dígitos para el mes**
MONTH	**Nombre completo del mes**
MON	**Abreviatura de tres letras del mes**
DY	**Abreviatura de tres letras del día de la semana**
DAY	**Nombre completo del día de la semana**
DD	**Día del mes en número**

Ejemplo: Si tenemos la columna denominada hire_date, cuyo dominio es de tipo fecha y queremos obtener una cadena de caracteres formada por el día en número, el mes en letra y el año con 4 cifras, todos ellos separados por un espacio en blanca, usaremos:

```
TO_CHAR (hire_date, 'fmDD Month YYYY')
```

se obtendría:

17 June 1987
21 September 1989
13 January 1993
3 January 1990
21 May 1991
7 February 1999
16 November 1999

Estos son algunos de los elementos de formato que puede utilizar con la función TO_CHAR para mostrar un valor numérico como carácter::

9	**Representa un número.**
0	**Obliga a mostrar un cero.**
$	**Coloca un signo de dólar flotante.**
L	**Utiliza el símbolo de divisa local flotante.**
.	**Imprime una coma decimal.**
,	**Imprime un indicador de miles.**

9.5. Funciones generales

Permiten definir valores a utilizar en el caso de que las expresiones tomen el valor nulo.

Función	Descripción
NVL(valor,sustituto)	Si el *valor* es NULL, devuelve el valor *sustituto*; de otro modo, devuelve valor
NVL2(valor,sustituto1, sustituto2)	Variante de la anterior, devuelve el valor *sustituto1* si *valor* no es nulo. Si *valor* es nulo devuelve el *sustituto2*

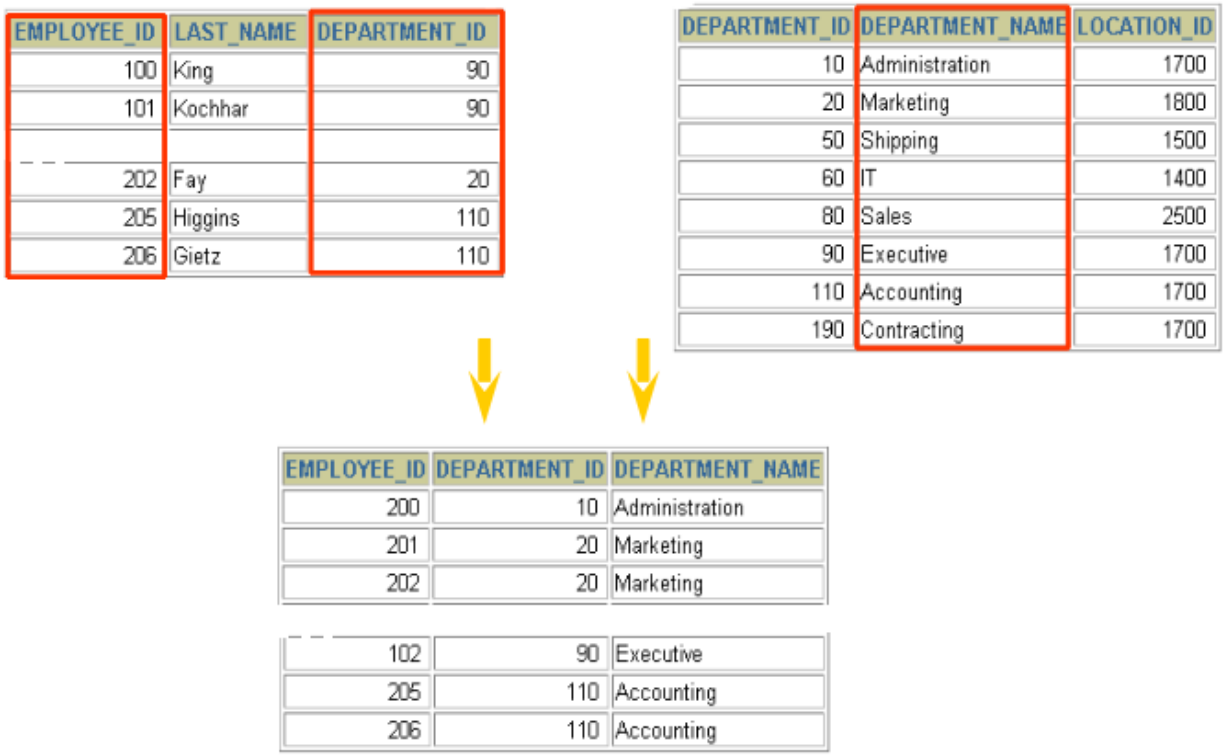
COALESCE(listaExpresiones)	Devuelve la primera de las expresiones que no es nula. Ejemplo[2]:
	
	`CREATE TABLE test (`
	`col1 VARCHAR2(1),`
	`col2 VARCHAR2(1),`
	`col3 VARCHAR2(1));`
	
	`INSERT INTO test VALUES (NULL, 'B', 'C');`
	`INSERT INTO test VALUES ('A', NULL, 'C');`
	`INSERT INTO test VALUES (NULL, NULL, 'C');`
	`INSERT INTO test VALUES ('A', 'B', 'C');`
	
	`SELECT COALESCE(col1, col2, col3) FROM test;`
	
	El resultado es:
	B
	A
	C
	A

NULLIF(valor1,valor2)	Devuelve nulo si *valor1* es igual a *valor2*. De otro modo devuelve *valor1*

10. Obtener datos de múltiples tablas

Es más que habitual necesitar en una consulta 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.

En el SQL es posible listar más de una tabla que se quiere consultar especificándolo en la cláusula `FROM`.



EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
202	Fay	20
205	Higgins	110
206	Gietz	110

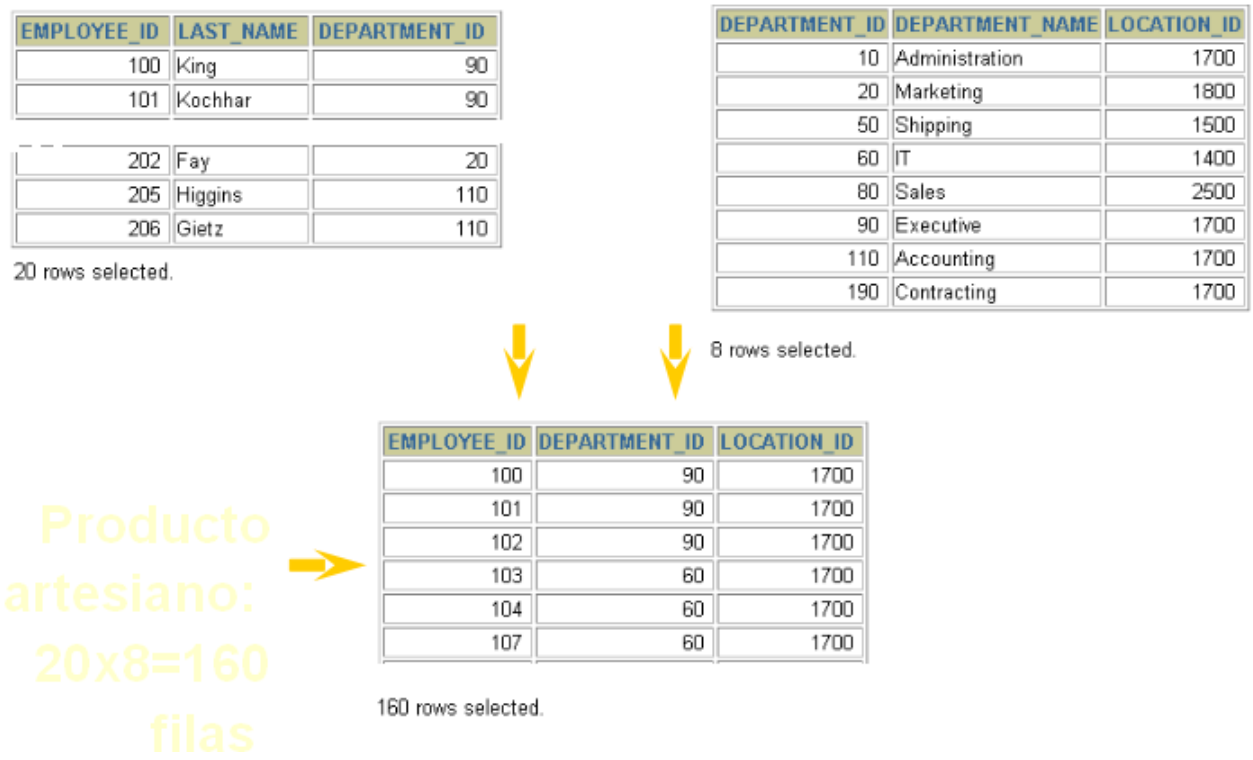
DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700

EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
200	10	Administration
201	20	Marketing
202	20	Marketing
102	90	Executive
205	110	Accounting
206	110	Accounting

10.1. Producto cartesiano

Cuando una condición de unión no es válida o está completamente omitida, el resultado es un producto cartesiano, en el que se muestran todas las combinaciones de filas. Todas las filas de la primera tabla se unen con todas las filas de la segunda tabla.

Los productos cartesianos tienden a generar un gran número de filas y es poco frecuente que el resultado sea útil. Debemos incluir siempre una condición de unión válida en una cláusula `WHERE`, a menos que se tenga la necesidad específica de combinar todas las filas de todas las tablas.




```
SELECT EMPLOYEE_ID, DEPARTMENT_ID, LOCATION_ID
FROM   EMPLOYEES, DEPARTMENTS;
```

La sintaxis es correcta ya que, efectivamente, en el apartado **FROM** se pueden indicar varias tablas separadas por comas. Pero eso produce un producto cruzado: aparecerán todos los registros de los empleados relacionados con todos los registros de departamentos,.

El producto cartesiano a veces es útil para realizar consultas complejas, pero en el caso normal no lo es. Necesitamos discriminar ese producto para que sólo aparezcan los registros de los empleados relacionados con sus departamentos correspondientes. A eso se le llama asociar o combinar (***join***) tablas.

10.2. Combinaciones internas o de igualdad: *equijoin*

La combinación consigue crear una sola tabla a partir de las tablas especificadas en la cláusula FROM, haciendo coincidir los valores de las columnas relacionadas de estas tablas.

Ejemplo:

Queremos saber el NIF del cliente y el código y el precio del proyecto que desarrollamos para el cliente número 20:

```
SELECT proyectos.codigo_proyecto, proyectos.precio, clientes.nif

FROM clientes, proyectos
WHERE clientes.codigo_cli = proyectos.codigo_cliente
AND clientes.codigo_cli = 20;
```

El resultado sería:

proyectos.codigo_proyecto	proyectos.precio	clientes.nif
3	1,0E+6	38.123.898-E

Si trabajamos con más de una tabla, puede ocurrir que la tabla resultante tenga dos columnas con el mismo nombre. Por ello es obligatorio especificar a qué tabla corresponden las columnas a las que nos estamos refiriendo, denominando la tabla a la que pertenecen antes de ponerlas (por ejemplo, *clientes.codigo_cli*). Para simplificarlo, se utilizan los **alias** que, en este caso, se definen en la cláusula `FROM`.

Ejemplo:

'c' podría ser el alias de la tabla clientes. De este modo, para indicar a qué tabla pertenece codigo_cli, sólo haría falta poner: c.codigo_cli.

Veamos cómo quedaría la consulta anterior expresada mediante alias, aunque en este ejemplo no serían necesarios, porque todas las columnas de las dos tablas tienen nombres diferentes.

Pediremos, además, las columnas c.codigo_cli y p.codigo_cliente.

```
SELECT p.codigo_proyecto, p.precio, c.nif,
p.codigo_cliente, c.codigo_cli
FROM clientes c, proyectos p
WHERE c.codigo_cli = p.codigo_cliente AND c.codigo_cli =
20;
```

Entonces obtendríamos este resultado:

p.codigo_proyec	p.precio	c.nif	p.codigo_cliente	c.codigo_cli
3	1,0E+6	38.123.898-E	20	20

Notemos que en `WHERE` necesitamos expresar el vínculo que se establece entre las dos tablas, en este caso *codigo_cli* de **clientes** y *codigo_cliente* de **proyectos**. Expresado en operaciones del álgebra relacional, esto significa que hacemos una combinación en lugar de un producto cartesiano.

La operación que acabamos de hacer es una equicombinación (***equi-join***); por lo tanto, nos aparecen dos columnas idénticas: c.codigo_cli y p.codigo_cliente.

10.3. Combinaciones de no igualdad

A las relaciones descritas anteriormente se las llama relaciones en igualdad (***equijoins***), 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, pero estas tablas poseen una relación que ya no es de igualdad.

La forma sería:

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

10.4. Combinaciones externas: *outer join*

En las combinaciones internas, si una fila no satisface una condición de unión, no aparecerá en el resultado de la consulta.

Las filas que faltan se pueden devolver si se utiliza un operador de combinación externa en la condición de la combinación. El operador es un signo más entre paréntesis **(+)** y se coloca a continuación del nombre de la columna en la tabla sin filas coincidentes.

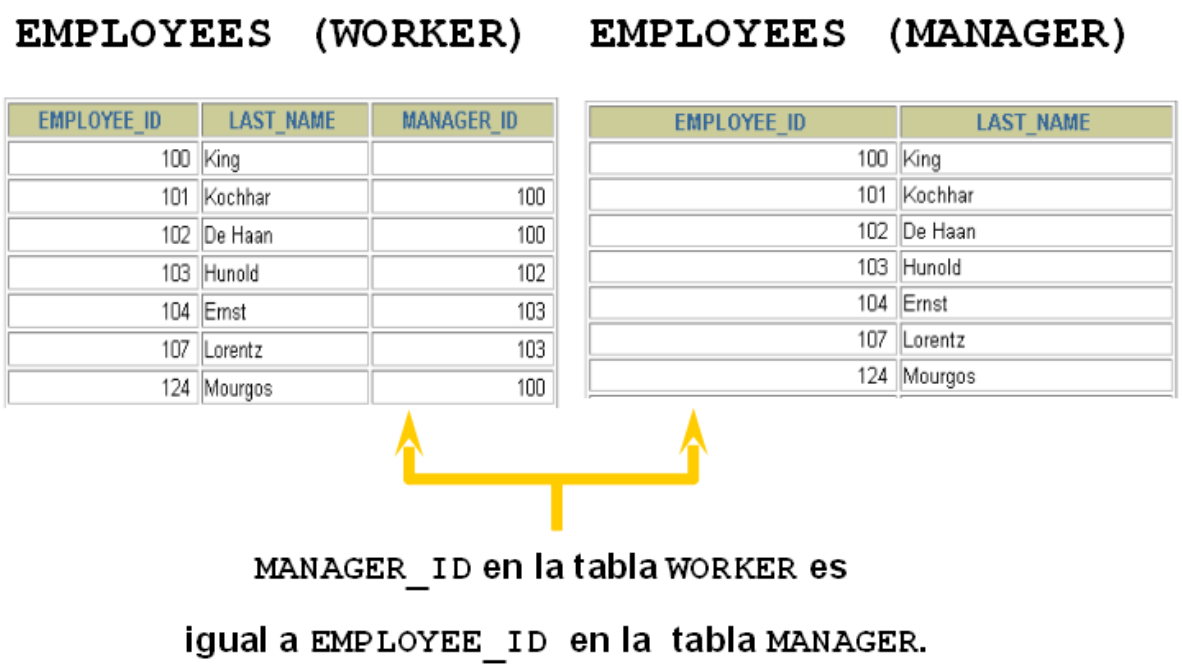
Si utilizamos OUTER JOIN:

```
SELECT  e.code_no, e.experimentos, r.res1, r.res2
FROM    experimentos e, resultados r
WHERE   e.code_no = r.code_no(+);
```

Restricciones del operador (+):

- El operador de unión externa solamente puede aparecer en un lado de la expresión, en el que falta la información.
- Una condición que implique una combinación externa no puede utilizar el operador IN ni estar enlazada a otra condición con el operador OR.

10.5. Combinaciones de una tabla consigo misma: *self join*



A veces es necesario unir una tabla consigo misma. Para buscar el nombre del director de cada empleado, necesitamos unir la tabla EMPLEADOS consigo misma.

Por ejemplo, para buscar el nombre del director de *Mourgos*, necesito:

- Buscar a *Mourgos* en la tabla `EMPLEADOS` mirando la columna `LAST_NAME`.
- Buscar el número de su director mirando la columna `MANAGER_ID`. Su número es 100.
- Buscar el nombre del director en `EMPLOYEE_ID` igual a 100 mirando en la columna `LAST_NAME`. El numero del empleado 100 es *King* por lo que *King* es el director de *Mourgos*.

En este proceso tenemos que mirar dos veces la tabla. Para simular dos tablas en la cláusula `FROM`, pondremos dos alias diferentes para la misma tabla.

```
SELECT worker.last_name ||' trabaja para '||
manager.last_name
FROM employees worker, employees manager
WHERE worker.manager_id = manager.employee_id;
```

La cláusula `WHERE` contiene la unión que significa *"dónde el número de director de un trabajador coincide con el número de empleado para el director"*.

10.6. Sintaxis SQL 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)]
[WHERE condiciones];
```

Se describen sus posibilidades en los siguientes apartados.

10.6.1. 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.

Ejemplo:

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

10.6.2. Natural Join

La cláusula **NATURAL JOIN** se basa en todas las columnas de dos tablas que tienen el mismo nombre. Selecciona filas de las dos tablas que tienen los mismos valores en todas las columnas coincidentes.

La unión sólo puede ocurrir en columnas que tengan los mismos nombres y tipos de datos en las dos tablas. Si las columnas tienen el mismo nombre pero distinto tipo de dato, se produce error.

Ejemplo:

Veamos a continuación un ejemplo en el que las columnas para las que se haría la combinación natural se denominan igual en las dos tablas. Ahora queremos saber el código y el nombre de los empleados que están asignados al departamento cuyo teléfono es 977.33.38.52:

```
SELECT codigo_empl, nombre_empl
FROM empleados NATURAL JOIN departamentos
WHERE telefono = '977.333.852';
```

empleados.codigo_empl	empleados.nombre_empl
5	Clara
6	Laura
8	Sergio

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

10.6.3. Join Using

Las uniones naturales utilizan todas las columnas con nombres y tipos de datos coincidentes para unir las tablas. La cláusula **USING** se puede utilizar para especificar solamente las columnas que se deben utilizar para una unión de igualdad.

Las columnas de referencia en la cláusula **USING** no deben tener un alias de tabla en ningún lugar de la sentencia SQL.

Ejemplo:

```
SELECT codigo_empl, nombre_empl

FROM empleados JOIN departamentos USING (nombre_dep, ciudad_dep)

WHERE telefono = '977.333.852';
```

empleados.codigo_empl	empleados.nombre_empl
5	Clara
6	Laura
8	Sergio

10.6.4. 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:

Ejemplo:

```
SELECT p.codigo_proyecto, p.precio, c.nif,
p.codigo_cliente, c.codigo_cli

FROM clientes c JOIN proyectos p ON c.codigo_cli = p.codigo_cliente

WHERE c.codigo_cli = 20;
```

Y obtendríamos el resultado :

p.codigo_proyec	p.precio	c.nif	p.codigo_cliente	c.codigo_cli
3	1,0E+6	38.123.898-E	20	20

Ejemplo:

```
SELECT E.APELLIDO AS "Nombre empleado", J.APELLIDO AS "Nombre jefe"

FROM EMPLE E JOIN EMPLE J
ON (E.DIR=J.EMP_NO);
```

10.6.5. Combinaciones externas

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

Ejemplo:

Si quisiéramos vincular con una combinación natural interna las tablas empleados y departamentos para saber el código y el nombre de todos los empleados y el nombre,la ciudad y el teléfono de todos los departamentos, haríamos:

```
SELECT e.codigo_empl, e.nombre_empl, e.nombre_dep, e.ciudad_dep,
d.telefono

FROM empleados e NATURAL JOIN departamentos d;
```

Y obtendríamos el siguiente resultado:

e.codigo_empl	e.nombre_empl	e.nombre_dep	e.ciudad_dep	d.telefono
1	María	DIR	Girona	972.23.89.70
2	Pedro	DIR	Barcelona	93.422.60.70
3	Ana	DIS	Lleida	973.23.50.40
4	Jorge	DIS	Barcelona	93.224.85.23
5	Clara	PROG	Tarragona	977.33.38.52
6	Laura	PROG	Tarragona	977.33.38.52
8	Sergio	PROG	Tarragona	977.33.38.52

Fijémonos en que en el resultado no aparece el empleado número 7, que no está asignado a ningún departamento, ni el

departamento de programación de Girona, que no tiene ningún empleado asignado.

En los ejemplos siguientes veremos cómo varían los resultados que iremos obteniendo según los tipos de combinación externa:

a) Combinación externa izquierda

```
SELECT e.codigo_empl, e.nombre_empl, e.nombre_dep,
e.ciudad_dep, d.telefono

FROM empleados e NATURAL LEFT OUTER JOIN departamentos d;
```

(la tabla LEFT sería empleados por que es la que está a la izquierda del JOIN)

El resultado sería el que podemos ver a continuación:

e.codigo_empl	e.nombre_empl	e.nombre_dep	e.ciudad_dep	d.telefono
1	María	DIR	Girona	972.23.89.70
2	Pedro	DIR	Barcelona	93.422.60.70
3	Ana	DIS	Lleida	973.23.50.40
4	Jorge	DIS	Barcelona	93.224.85.23
5	Clara	PROG	Tarragona	977.33.38.52
6	Laura	PROG	Tarragona	977.33.38.52
7	Rogelio	NULL	NULL	NULL
8	Sergio	PROG	Tarragona	977.33.38.52

Aquí figura el empleado 7.

b) Combinación externa derecha

```
SELECT e.codigo_empl, e.nombre_empl, e.nombre_dep,
e.ciudad_dep, d.telefono
FROM empleados e NATURAL RIGHT OUTER JOIN departamentos d;
```

Obtendríamos este resultado

e.codigo_empl	e.nombre_empl	e.nombre_dep	e.ciudad_dep	d.telefono
1	María	DIR	Girona	972.23.89.70
2	Pedro	DIR	Barcelona	93.422.60.70
3	Ana	DIS	Lleida	973.23.50.40
4	Jorge	DIS	Barcelona	93.224.85.23
5	Clara	PROG	Tarragona	977.33.38.52
6	Laura	PROG	Tarragona	977.33.38.52
8	Sergio	PROG	Tarragona	977.33.38.52
NULL	NULL	PROG	Girona	972.23.50.91

Aquí figura el departamento de programación de Girona.

c) Combinación externa plena

```
SELECT e.codigo_empl, e.nombre_empl, e.nombre_dep,
e.ciudad_dep, d.telefono
FROM empleados e NATURAL FULL OUTER JOIN departamentos d;
```

Y obtendríamos el siguiente resultado:

e.codigo_empl	e.nombre_empl	e.nombre_dep	e.ciudad_dep	d.telefono
1	María	DIR	Girona	972.23.89.70
2	Pedro	DIR	Barcelona	93.422.60.70
3	Ana	DIS	Lleida	973.23.50.40
4	Jorge	DIS	Barcelona	93.224.85.23
5	Clara	PROG	Tarragona	977.33.38.52
6	Laura	PROG	Tarragona	977.33.38.52
7	Rogelio	NULL	NULL	NULL
8	Sergio	PROG	Tarragona	977.33.38.52
NULL	NULL	PROG	Girona	972.23.50.91

```
Aquí figura el empleado 7 y el departamento de programación
de Girona.
```

10.6.6. Combinaciones con más de dos tablas

Si queremos combinar tres tablas o más con el SQL92 introductorio, sólo tenemos que añadir todas las tablas en el FROM y los vínculos necesarios en el `WHERE`. Si queremos combinarlas con el SQL99 , tenemos que ir haciendo combinaciones de tablas por pares, y la tabla resultante se convertirá en el primer componente del siguiente par.

```
Veamos ejemplos de los dos casos, suponiendo que queremos
combinar las tablas empleados, proyectos y clientes:

SELECT *
FROM empleados, proyectos, clientes

WHERE num_proyec = codigo_proyec AND codigo_cliente = codigo_cli;

o bien:

SELECT *
FROM (empleados JOIN proyectos ON num_proyec =
codigo_proyec)
JOIN clientes ON codigo_cliente = codigo_cli;
```

11. Agrupaciones: GROUP BY y HAVING

Es muy común utilizar consultas en las que se desee agrupar los datos a fin de realizar cálculos en vertical, es decir calculados a partir de datos de distintos registros.

Antes de ver la cláusula que me permite trabajar con grupos, vamos a conocer funciones de agregación que me permitirán realizar cálculos sobre grupos.

11.1. Funciones de agregación

El SQL nos ofrece las siguientes funciones de agregación para efectuar varias operaciones sobre los datos de una base de datos:

Función	Significado
COUNT(*)	Cuenta los elementos de un grupo. Se utiliza el asterisco para no tener que indicar un nombre de columna concreto, el resultado es el mismo para cualquier columna
SUM(*expresión*)	Suma los valores de la expresión
AVG(*expresión*)	Calcula la media aritmética sobre la expresión indicada
MIN(*expresión*)	Mínimo valor que toma la expresión indicada
MAX(*expresión*)	Máximo valor que toma la expresión indicada
STDDEV(*expresión*)	Calcula la desviación estándar
VARIANCE(*expresión*)	Calcula la varianza

A diferencia de las funciones de una sola fila, las funciones de grupo operan sobre juegos de filas para proporcionar un resultado por grupo. Estos juegos pueden ser la tabla completa o la tabla dividida en grupos.

La sintaxis de las funciones de grupo:

```
SELECT [column,] group_function(column), ...
FROM table
[WHERE condition]
[GROUP BY column]
[ORDER BY column];
```

Los argumentos que pueden tener estas funciones son:

```
COUNT({* | [DISTINCT | ALL] expr} )
AVG( [DISTINCT | ALL] n)
MAX( [DISTINCT | ALL] expr )
MIN( [DISTINCT | ALL] expr )
SUM( [DISTINCT | ALL] n )
```

- DISTINCT hace que la función solo considere valores no duplicados; ALL hace que se considere todos los valores incluyendo duplicados. El valor por defecto es ALL y, por lo tanto, no es necesario especificarlo.

- Todas las funciones de grupo ignoran los valores nulos. Para sustituir un valor nulo por otro valor usar las funciones NVL, NVL2.

En general, las funciones de agregación se aplican a una columna, excepto la función de agregación COUNT, que normalmente se aplica a todas las columnas de la tabla o tablas seleccionadas. Por lo tanto, **COUNT (*)** contará todas las filas de la tabla o las tablas que cumplan las condiciones, incluidas las filas duplicadas y las que contengan valores nulos en alguna de sus columnas. Si se utilizase **COUNT(distinct columna)**, sólo contaría los valores que no fuesen nulos ni repetidos, y si se utilizase **COUNT (columna)**, sólo contaría los valores que no fuesen nulos de la columna especificada.

Ejemplo de utilización de la función COUNT (*)

Veamos un ejemplo de uso de la función COUNT, que aparece en la cláusula SELECT, para hacer la consulta "¿Cuántos departamentos están ubicados en la ciudad de Lleida?":

```
SELECT COUNT(*) AS numero_dep
FROM departamentos
WHERE ciudad_dep = 'Lleida';
```

La respuesta a esta consulta sería:

numero_dep
1

Ejemplo:

Veamos la diferencia entre los dos ejemplos siguientes:

```
SELECT AVG(COMISION)
FROM EMPLE;
```

En este caso, la media se calcula como la comisión total pagada a todos los empleados dividida entre el número de empleados que perciben comisión (4 empleados). Esto es debido a que las funciones de grupo ignoran los valores nulos de la columna.

```
SELECT AVG(NVL(COMISION, 0))
FROM EMPLE;
```

En este caso la media se calcula como la comisión total pagada a todos los empleados dividida por el número total de empleados de la empresa (14 empleados). La función NVL fuerza a las funciones de grupo a que incluyan valores nulos.

11.2. Cláusula GROUP BY

Hasta ahora, todas las funciones de grupo han tratado la tabla como un gran grupo de información. A veces, necesitamos dividir la tabla de información en grupos más pequeños. Esto se puede realizar utilizando la cláusula **GROUP BY**.

La sintaxis es la siguiente:

```
SELECT nombre_columnas_a seleccionar
FROM tabla_a_consultar
[WHERE condiciones]
GROUP BY columnas_según_las_cuales_se_quiere_agrupar
[HAVING condiciones_por_grupos]
[ORDER BY columna_ordenación [DESC] [, columna [DESC]...]];
```

Tenemos que tener en cuenta:

- Si incluimos una función de grupo en una cláusula `SELECT`, no podemos seleccionar también resultados individuales, a menos que la columna individual aparezca en la cláusula `GROUP BY`.
- Podemos utilizar la cláusula `WHERE` para excluir filas antes de dividirla en grupos.
- No podemos utilizar un alias de columna en la cláusula `GROUP BY`.
- Por defecto, las filas se ordena en orden ascendente de las columnas incluidas en la lista `GROUP BY`. Podemos sustituir este orden por defecto utilizando la cláusula `ORDER BY`.
- Al utilizar la cláusula `GROUP BY`, tenemos que asegurarnos de que todas las columnas de la lista `SELECT` que no son funciones de grupo están incluidas en la cláusula `GROUP BY`.

Ejemplo:

Imaginemos que queremos saber el sueldo medio que ganan los empleados de cada departamento:

```
SELECT nombre_dep, ciudad_dep, AVG(sueldo) AS sueldo_medio
```

```
FROM empleados
GROUP BY nombre_dep, ciudad_dep;

El resultado de esta consulta sería:
```

nombre_dep	ciudad_dep	sueldo_medio
DIR	Barcelona	9,0E + 4
DIR	Girona	1,0E + 5
DIS	Lleida	7,0E + 4
DIS	Barcelona	7,0E + 4
PROG	Tarragona	3,3E + 4
NULL	NULL	4,0E + 4

11.3. Cláusula HAVING

La cláusula **`HAVING`** especifica condiciones de búsqueda para grupos de filas; lleva a cabo la misma función que antes cumplía la cláusula `WHERE` para las filas de toda la tabla, pero ahora las condiciones se aplican a los grupos obtenidos.

Oracle Server realiza los siguientes pasos cuando se utiliza la cláusula HAVING:

- Las filas se agrupan.
- Se aplica al grupo la función de grupo.
- Se muestran los grupos que coinciden con los criterios de la cláusula HAVING.

Ejemplo:

Veamos un ejemplo de uso de una función de agregación SUM del SQL que aparece en la cláusula HAVING de GROUP BY:

Queremos saber los códigos de los proyectos en los que la suma de los sueldos de los empleados es mayor que 180.000 euros:

```
SELECT num_proyec
FROM empleados
GROUP BY num_proyec
HAVING SUM (sueldo) >1.8E+5;
```

El resultado de esta consulta sería

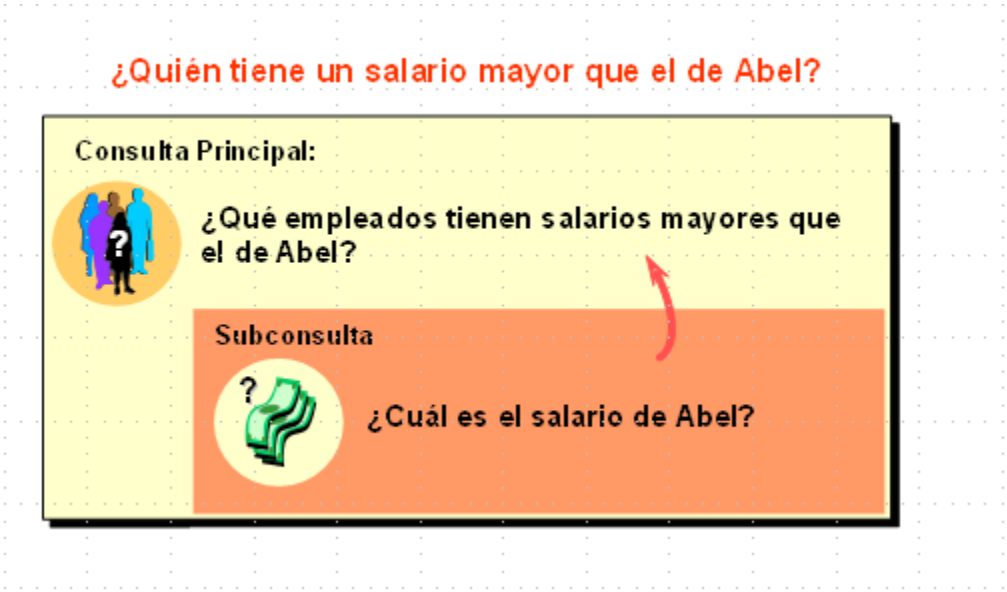
num_proyec
4

El orden de ejecución de una sentencia completa:

1. Seleccionar las filas deseadas utilizando WHERE. Esta cláusula eliminará columnas en base a la condición indicada.
2. Se establecen los grupos indicados en la cláusula GROUP BY.
3. Se calculan los valores de las funciones de totales (COUNT, SUM, AVG,...).
4. Se filtran los registros que cumplen la cláusula HAVING.
5. El resultado se ordena en base al apartado ORDER BY.

12. Subconsultas

Una **subconsulta** es una consulta incluida dentro de una cláusula `WHERE` o `HAVING` de otra consulta. En ocasiones, para expresar ciertas condiciones no hay más remedio que obtener el valor que buscamos como resultado de una consulta.



Supongamos que deseamos escribir una consulta para averiguar quién gana un salario mayor que Abel. Para resolver este problema necesitamos dos consultas: una para buscar lo que gana Abel y una segunda para buscar quién gana más de esa cantidad.

Podemos resolver este problema combinando las dos consultas, colocando una consulta dentro de otra.

La consulta interna o subconsulta devuelve un valor que utiliza la consulta externa o principal. La utilización de una subconsulta es equivalente a realizar dos consultas secuenciales y usar el resultado de la primera como valor de búsqueda en la segunda.

```
SELECT APELLIDO
FROM EMPLE
WHERE SALARIO > (SELECT SALARIO
                 FROM EMPLE
                 WHERE APELLIDO='Abel');
```

Sintaxis:

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

Podemos colocar la subconsulta en diversas cláusulas SQL como, por ejemplo:

- La cláusula `WHERE`.
- La cláusula `HAVING`.
- La cláusula `FROM`.

En la sintaxis *`OPERADOR`* es un operador de comparación, que pueden ser de dos clases:

- Operadores de una sola fila (<, >, <>, <=,...).
- Operadores de varias filas (`IN, ANY, ALL`, …).

Además debemos tener en cuenta:

- La subconsulta se debe escribir entre paréntesis.
- Tenemos que colocar la subconsulta a la derecha de la condición de comparación para una mayor legibilidad.
- Hay dos tipos de subconsultas: subconsultas que devuelven una sola fila y subconsultas que devuelven más de una fila.

12.1. Subconsultas de una sola fila

Son consultas que devuelven una sola fila a partir de la sentencia `SELECT` interna. Este tipo de subconsultas utilizan operadores de una sola fila:

Operador	Significado
=	Igual que
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
<>	No igual a

Ejemplo:
Mostrar los empleados que tienen el mismo oficio que el empleado 7900.

```
SELECT APELLIDO
FROM EMPLE
WHERE OFICIO = (SELECT OFICIO
                FROM EMPLE
                WHERE EMP_NO = 7900);
```

Podemos tener también **una sola consulta externa y varias subconsultas internas**. Además, las consultas externas e internas pueden obtener datos de tablas diferentes:

Ejemplo:

Mostrar los empleados cuyo oficio sea el mismo que el oficio del empleados 7900 y sus salarios sea superior al salario del empleado 7521.

```
SELECT APELLIDO
FROM EMPLE
WHERE OFICIO = (SELECT OFICIO
                FROM EMPLE
                WHERE EMP_NO = 7900)
AND   SALARIO >(SELECT SALARIO
                FROM EMPLE
                WHERE EMP_NO = 7521);
```

Podemos mostrar datos de una consulta principal **utilizando una función de grupo** en una subconsulta para devolver una sola fila.

Ejemplo:

Mostrar el apellido del empleado, el oficio y el salario de todos los empleados cuyos salarios sea igual al mínimo.

```
SELECT APELLIDO, OFICIO, SALARIO
FROM EMPLE
WHERE SALARIO = (SELECT MIN(SALARIO)
                 FROM EMPLE );
```

Ejemplo:

Si quisiéramos saber los códigos y los nombres de los proyectos de precio más elevado, en primer lugar tendríamos que encontrar los proyectos que tienen el precio más elevado.

Lo haríamos de la forma siguiente:

```
SELECT codigo_proyec, nombre_proyec
FROM proyectos
WHERE precio = (SELECT MAX(precio)
                FROM proyectos);
```

El resultado de la consulta anterior sería:

codigo_proyec	nombre_proyec
4	TINELL

También se pueden utilizar **subconsultas** no solo con la cláusula `WHERE` sino **con la cláusula `HAVING`.** En este caso, Oracle Server ejecuta la subconsulta y los resultados se devuelven a la cláusula `HAVING` de la consulta principal.

Ejemplo:

Mostrar los departamentos que tienen un salario mínimo mayor que el del departamento 30.

```
SELECT DEPT_NO AS "DEPARTAMENTO", MIN(SALARIO) AS "SALARIO
MINIMO"
FROM EMPLE
GROUP BY DEPT_NO
HAVING MIN(SALARIO) = ( SELECT MIN(SALARIO)
                        FROM EMPLE
                        WHERE DEPT_NO = 30);
```

12.2. Subconsultas de varias filas

Las subconsultas que devuelven más de una fila se denominan subconsultas de varias filas. En este caso se utiliza un operador de varias filas.

Operador	Significado
IN	Igual a cualquier miembro de la lista
ANY	Compara el valor con cada valor devuelto por la subconsulta
ALL	Compara el valor con todos los valores devueltos por la subconsulta

Ejemplo: USO OPERADOR IN EN SUBCONSULTAS DE VARIAS FILAS

Mostrar apellidos, salario y número de departamento de los empleados que ganan el mismo salario que el mínimo de cada departamento.

```
SELECT APELLIDO, SALARIO, DEPT_NO
FROM EMPLE
WHERE SALARIO IN (SELECT MIN(SALARIO)
                  FROM EMPLE
                  GROUP BY DEPT_NO);
```

Ejemplo: USO OPERADOR ANY EN SUBCONSULTAS DE VARIAS FILAS

Mostrar los empleados que no son analistas y cuyos salarios son menores que los de cualquier analista.

```
SELECT EMP_NO, APELLIDO, OFICIO, SALARIO
FROM EMPLE
WHERE SALARIO < ANY (SELECT SALARIO
                     FROM EMPLE
                     WHERE OFICIO = 'ANALISTA')
AND OFICIO <> 'ANALISTA';
```

- **< ANY** significa menos que el máximo.
- **> ANY** significa más que el mínimo.
- **= ANY** es equivalente a IN.

Ejemplo: USO OPERADOR ALL EN SUBCONSULTAS DE VARIAS FILAS

Mostrar los empleados que nos sean analistas y cuyos salarios son menores que el salario de todos los empleados con oficio analista .

```
SELECT EMP_NO, APELLIDO, OFICIO, SALARIO
FROM EMPLE
WHERE SALARIO < ALL (SELECT SALARIO
                     FROM EMPLE
                     WHERE OFICIO = 'ANALISTA')
AND OFICIO <> 'ANALISTA';
```

- **< ALL** significa menos que el mínimo.
- **> ALL** significa más que el máximo.

El operador **NOT** se puede utilizar con los operadores IN, ANY y ALL.

12.3. Subconsultas de varias columnas

Hasta ahora se han escrito subconsultas de una sola fila y subconsultas de varias filas en las que las que la sentencia `SELECT` interna solo devolvía una columna. Si deseamos comparar dos o más columnas , debemos escribir una cláusula `WHERE` compuesta utilizando operadores lógicos.

Las comparaciones de columnas de una subconsulta de varias columnas pueden ser comparaciones entre pares o comparaciones no entre pares.

Ejemplo:

Visualizar los detalles de los empleados dirigidos por el mismo director y que trabajen en el mismo departamento que los empleados de código 7788 o 7844.

```
SELECT EMP_NO, APELLIDOS, DIR, DEPT_NO
FROM EMPLE
WHERE (DIR, DEPT_NO) IN (SELECT DIR, DEPT_NO
                         FROM EMPLE
                         WHERE EMP_NO IN (7788, 7844))
AND EMP_NO NOT IN (7788, 7844);
```

En este caso se trata de una **subconsulta de comparación de pares.**

Ejemplo:

Visualizar los detalles de los empleados dirigidos por el mismo director que los empleados con código 7788 o 7844 y que trabajen en el mismo departamento que los empleados con código 7788 o 7844.

```
SELECT EMP_NO, APELLIDOS, DIR, DEPT_NO
FROM EMPLE
WHERE DIR IN    (SELECT DIR
                 FROM EMPLE
                 WHERE EMP_NO IN (7788, 7844))
AND DEPT_NO IN (SELECT DEPT_NO
                 FROM EMPLE
                 WHERE EMP_NO IN (7788, 7844))
AND EMP_NO NOT IN (7788, 7844);
```

En este caso se trata de una **subconsulta de comparación no entre pares.**

12.4. Uso de una subconsulta en la cláusula FROM

Podemos utilizar una subconsulta en la cláusula `FROM` de una sentencia `SELECT`. Una subconsulta en la cláusula `FROM` de una sentencia `SELECT` también se denomina *vista en línea* y define el origen de datos para dicha sentencia `SELECT` en particular y solo para esa sentencia.

Ejemplo:

Visualizar apellidos, salarios, números de departamento y salarios medios para todos los empleados que ganan más que el salario medio de su departamento.

```
SELECT APELLIDO, SALARIO, DEPT_NO, AVG(SALARIO)
FROM EMPLE P, (SELECT DEPT_NO, AVG(SALARIO) SALMEDIO
               FROM EMPLE
               GROUP BY DEPT_NO) S
WHERE P.DEPT_NO = S.DEPT_NO
AND P.SALARIO > S.SALMEDIO;
```

12.5. Uso del operador EXISTS

Con sentencias SELECT anidadas, todos los operadores lógicos son válidos. Además, se puede utilizar el operador **EXISTS**. Este operador se utiliza con frecuencia para probar si existe un valor que la consulta externa haya recuperado en el juego de resultados de los valores recuperados por la consulta interna. Si la subconsulta devuelve al menos 1 fila, el operador devuelve TRUE. Si el valor no existe, devuelve FALSE.

Ejemplo:

Buscar los empleados que tengan al menos una persona que les informe (que sean jefes).

```
SELECT EMP_NO, APELLIDO, OFICIO
FROM   EMPLE P
WHERE  EXISTS (SELECT 'X'
               FROM EMPLE
               WHERE DIR=P.EMP_NO);
```

El operador **EXISTS** asegura que la búsqueda en la consulta interna no continua si al menos una coincidencia para el director y el código de empleado se encuentra en la condición.

Observar que no es necesario que la consulta SELECT interna devuelva un valor especifico, por lo que se puede seleccionar una constante. Desde el punto de vista del rendimiento, es más rápido seleccionar una constante que una columna.

Se puede utilizar el operador **IN** como alternativa para el operador **EXISTS**, como se muestra en el siguiente ejemplo:

Ejemplo:

```
SELECT EMP_NO, APELLIDO, OFICIO
FROM   EMPLE
WHERE  EMP_NO IN (SELECT DIR
                  FROM EMPLE
                  WHERE DIR IS NOT NULL);
```

13. Recuperación jerárquica

Utilizando consultas jerárquicas podemos recuperar datos basándonos en una relación jerárquica natural entre las filas de una tabla. Un proceso denominado *desplazamiento por el árbol* permite construir la jerarquía.

Imaginemos una tabla de empleados definida por un código de empleado, nombre del mismo y el código del jefe. Este último código está relacionado con el código de empleado que posee el jefe en cuestión. Así definido, una consulta que muestre el nombre de un empleado y el nombre de su jefe directo, sería:

```
SELECT e.nombre AS empleado, j.nombre AS jefe
FROM emple e
JOIN emple j ON (e.dir = j.emp_no);
```

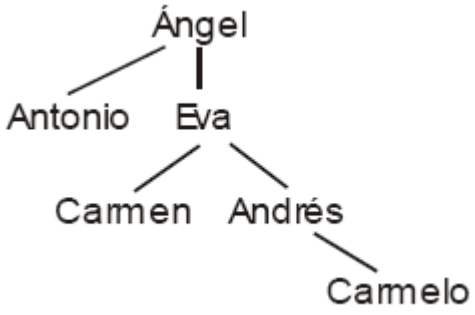
o

```
SELECT e.nombre AS empleado, j.nombre AS jefe
FROM emple e, emple j
WHERE e.dir = j.emp_no;
```

Saldría por ejemplo:

EMPLEADO	JEFE
Antonio	Ángel
Ángel	
Eva	Ángel
Carmen	Eva
Andrés	Eva
Carmelo	Andrés

En el ejemplo se observa como un jefe puede tener otro jefe, generando una estructura jerárquica:



En este tipo de estructuras, a veces se requieren consultas que muestren todos los empleados de un jefe, mostrando los mandos intermedios. Se trata de una consulta que recorre ese árbol. Este tipo de consultas posee esta sintaxis:

```
SELECT [LEVEL,] listaDeColumnasYExpresiones
FROM tabla(s)...
[WHERE condiciones...]
[START WITH condiciones]
CONNECT BY [PRIOR] expresion1=[PRIOR] expresion2
```

En la sintaxis:

- **LEVEL** Para cada fila devuelta por una consulta jerárquica, la pseudocolumna LEVEL devuelve 1 para la fila raíz, 2 para un secundario de la raíz, …
- **START WITH** Especifica las filas raíz de la jerarquía (donde empezar). Esta cláusula es necesaria para una consulta jerárquica verdadera.
- **CONNECT BY PRIOR** Especifica las columnas en las que existe la relación entre las filas principal y secundaria. Esta cláusula es necesaria para una consulta jerárquica.

Ejemplo:

```
SELECT nombre FROM emple
START WITH nombre='Andrés'
CONNECT BY PRIOR dir = emp_no;
```

Resultado:

NOMBRE
Andrés
Eva
Ángel

Sin embargo:

```
SELECT nombre FROM emple
START WITH nombre='Andrés'
CONNECT BY dir= PRIOR emp_no;
```

Devuelve:

NOMBRE
Andrés
Carmelo

Si en lugar de Andrés en esa consulta buscáramos desde Ángel, saldría:

NOMBRE
Ángel
Antonio
Eva
Carmen
Andrés
Carmelo

El modificador LEVEL permite mostrar el nivel en el árbol jerárquico de cada elemento:

Ejemplo:

```
SELECT LEVEL, nombre FROM emple
START WITH nombre='Ángel'
CONNECT BY dir= PRIOR emp_no;
```

Resultado:

LEVEL	NOMBRE
1	Ángel
2	Antonio
2	Eva
3	Carmen
3	Andrés
4	Carmelo

Para eliminar recorridos, se utilizan condiciones en `WHERE` o en el propio `CONNECT`. De modo que :

Ejemplos:

```
SELECT LEVEL, nombre FROM emple
WHERE nombre!='Eva'
START WITH nombre='Ángel'
CONNECT BY dir= PRIOR emp_no;
```

En este caso se eliminaría la fila del empleado Eva.

```
SELECT LEVEL, nombre FROM emple
START WITH nombre='Ángel'
CONNECT BY dir = PRIOR emp_no AND nombre!='Eva';
```

No sale ni Eva ni sus empleados (se elimina al empleado y todas sus filas secundarias).

14. Los operadores SET

Los operadores SET combinan los resultados de dos o más consultas componentes en un resultado. Las consultas que contienen operadores SET se llaman *consultas compuestas*.

Operador	**Devuelve**
UNION	Todas las filas seleccionadas por cada una de las consultas.
UNION ALL	Todas las filas seleccionadas por cualquiera de las consultas, incluidos duplicados.
INTERSECT	Todas las filas distintas seleccionadas por ambas consultas.
MINUS	Todas las filas distintas seleccionadas por la primera sentencia SELECT y no seleccionadas por la segunda sentencia SELECT.

Todos los operadores SET tienen la misma prioridad. Si una sentencia SQL contiene varios operadores SET, Oracle Server los evalúa de izquierda (arriba) a derecha (abajo) si no hay paréntesis que especifiquen explícitamente otro orden.

14.1. El operador UNION

El operador **UNION** permite unir consultas de dos o más sentencias `SELECT ...FROM`. Su formato es:

```
SELECT columnas
FROM tabla
[WHERE condiciones]
UNION [ALL]
SELECT columnas
FROM tabla
[WHERE condiciones];
```

Si ponemos la opción **ALL**, aparecerán todas las filas obtenidas a causa de la unión. No la pondremos si queremos eliminar las filas repetidas. Lo más importante de la unión es que somos nosotros quienes tenemos que procurar que se efectúe entre columnas definidas sobre dominios compatibles; es decir, que tengan la misma interpretación semántica.

Ejemplo:

Si queremos saber todas las ciudades que hay en nuestra base de datos, podríamos hacer:

```
SELECT ciudad
```

```
FROM clientes
UNION
SELECT ciudad_dep
FROM departamentos;
```

El resultado de esta consulta sería:

ciudad
Barcelona
Girona
Lleida
Tarragona

A tener en cuenta:

- El número de columnas y los tipos de datos de las columnas seleccionadas deben ser idénticos en todas las sentencias `SELECT` utilizadas en la consulta. Los nombres de las columnas no tienen que ser idénticos.
- Los valores `NULL` no se ignoran durante la comprobación de duplicados.
- El operador `IN` tiene una prioridad más alta que el operador `UNION`.
- Por defecto, el resultado se ordena en orden ascendente por la primera columna de la cláusula `SELECT`.
- Si utilizamos el operador **`UNION ALL`**, las filas duplicadas no se eliminan y el resultado no se ordena por defecto.
- No se puede utilizar la palabra clave `DISTINCT`.

14.2. El operador INTERSECT

Para hacer la intersección entre dos o más sentencias `SELECT … FROM`, podemos utilizar el operador **`INTERSECT`**, cuyo formato es:

```
SELECT columnas
FROM tabla
[WHERE condiciones]
INTERSECT
SELECT columnas
FROM tabla
[WHERE condiciones];
```

Lo más importante de la intersección es que somos nosotros quienes tenemos que vigilar que se haga entre columnas definidas sobre dominios compatibles, es decir, que tengan la misma interpretación semántica.

Ejemplo:

Si queremos saber todas las ciudades donde tenemos departamentos en los que podamos encontrar algún cliente, podríamos hacer:

```
SELECT ciudad
FROM clientes
INTERSECT
SELECT ciudad_dep
FROM departamentos;
```

El resultado de esta consulta sería:

ciudad
Barcelona
Girona
Lleida
Tarragona

INTERSECT no ignora los valores NULL.

14.3. El operador MINUS

Para encontrar la diferencia (filas devueltas por la primera consulta que no estén presentes en la segunda) entre dos o más sentencias `SELECT... FROM` podemos utilizar el operador **`MINUS`**, que tiene este formato:

```
SELECT columnas
FROM tabla
[WHERE condiciones]
MINUS
SELECT columnas
FROM tabla
[WHERE condiciones];
```

Lo más importante de la diferencia es que somos nosotros quienes tenemos que vigilar que se haga entre columnas definidas sobre dominios compatibles.

Ejemplo:

Si queremos saber los clientes que no nos han contratado ningún proyecto, podríamos hacer:

```
SELECT codigo_cli
FROM clientes
MINUS
SELECT codigo_cliente
FROM proyectos;
```

El resultado de esta consulta sería

codigo_cli
40

Todas las columnas de la cláusula `WHERE` deben estar en la cláusula `SELECT` para que el operador `MINUS` funcione.