UT.4 REALIZACIÓN DE CONSULTAS

1. INTRODUCCIÓN.... 2
2. CONSULTAS SQL.... 5
2.1 PROYECCIÓN.... 7
2.1.1 EXPRESIONES ARITMÉTICAS.... 8
2.1.2 EL OPERADOR CONCATENACIÓN.... 9
2.1.3 ALIAS DE COLUMNAS.... 10
2.2 LA CLÁUSULA WHERE (SELECCIÓN).... 11
2.2.1 PRECEDENCIA DE OPERADORES.... 14
2.3 LA CLÁUSULA ORDER BY. ORDENACIÓN DE REGISTROS.... 15
3. FUNCIONES.... 17
3.1 FUNCIONES A NIVEL DE FILA.... 17
3.1.1. FUNCIONES NUMÉRICAS.... 18
3.1.2 FUNCIONES DE CADENA DE CARACTERES.... 21
3.1.3 FUNCIONES DE CONVERSIÓN.... 24
3.1.4 FUNCIONES DE MANEJO DE FECHAS.... 25
3.1.5 OTRAS FUNCIONES: NVL Y DECODE.... 28
3.1.6. FUNCIONES ANIDADAS.... 29
4. CONSULTAS DE RESUMEN.... 30
4.1. FUNCIONES DE AGREGADO: SUM Y COUNT.... 31
4.2 FUNCIONES DE AGREGADO: MIN Y MAX.... 31
4.3 FUNCIONES DE AGREGADO: AVG, VAR, STDEV Y STDEVP.... 32
5. AGRUPAMIENTO DE REGISTROS.... 33
6. CONSULTAS MULTITABLAS.... 34
6.1 PRODUCTO CARTESIANO.... 35
6.2 COMPOSICIONES INTERNAS.... 36
6.2.1 JOIN IMPLÍCITO.... 36
6.2.2 JOIN EXPLÍCITO.... 37
6.3 COMPOSICIONES EXTERNAS.... 39
7. SUBCONSULTAS.... 40
7.1 SUBCONSULTAS MONO REGISTRO.... 41
7.1.1 USO DE FUNCIONES DE GRUPO EN SUBCONSULTAS.... 42
7.2 SUBCONSULTAS MULTIREGISTRO.... 43
7.3 SUBCONSULTAS MULTICOLUMNA.... 45
7.3.1 COMPARACIÓN ENTRE COLUMNAS.... 47
7.4 SUBCONSULTAS EN LA CLÁUSULA FROM.... 48
7.5 SUBCONSULTAS EN CLÁUSULA SELECT.... 49
8. UNION, INTERSECT Y MINUS.... 49
9. CREACIÓN DE TABLAS A PARTIR DE UNA CONSULTA.... 52
10. OTRAS INSTRUCCIONES ÚTILES.... 52
10.1 COALESCE.... 53
10.2 CASE.... 53
10.3 FETCH FIRST n ROWS ONLY.... 54

1. INTRODUCCIÓN

En unidades anteriores has aprendido que **SQL** es un conjunto de sentencias u órdenes que se necesitan para acceder a los datos. Este lenguaje es utilizado por la mayoría de las aplicaciones donde se trabaja con datos para acceder a ellos. Es decir, es la vía de comunicación entre el usuario y la base de datos.

SQL nació a partir de la publicación "A relational model of data for large shared data banks" de Edgar Frank Codd. IBM aprovechó el modelo que planteaba Codd para desarrollar un lenguaje acorde con el recién nacido MODelo relacional, a este primer lenguaje se le llamó **SEQUEL** (Structured English QUEry Language). Con el tiempo SEQUEL se convirtió en **SQL** (Structured Query Language). En 1979, la empresa Relational Software sacó al mercado la primera implementación comercial de SQL. Esa empresa es la que hoy conocemos como Oracle.

Actualmente SQL sigue siendo el estándar en lenguajes de acceso a base de datos relacionales.

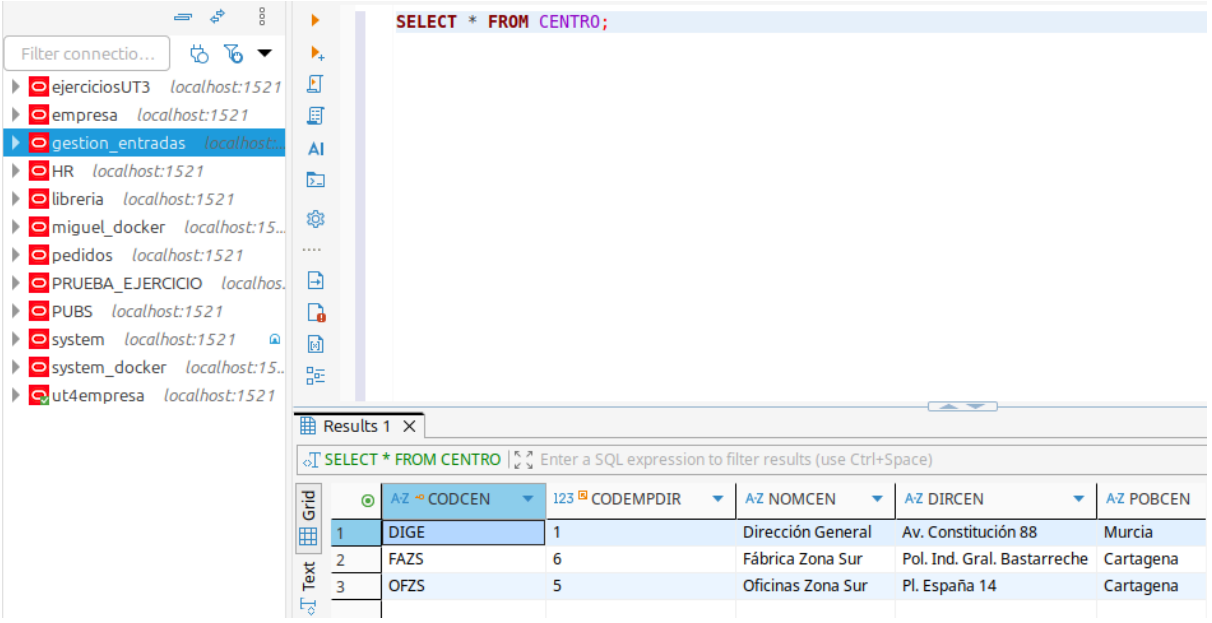
En 1992, ANSI e ISO completaron la **estandarización** de SQL y se definieron las sentencias básicas que debía contemplar SQL para que fuera estándar. A este SQL se le denominó ANSI-SQL o SQL92.

Hoy en día todas las bases de datos comerciales cumplen con este estándar aunque, eso sí, cada fabricante añade sus mejoras al lenguaje SQL.

La **primera fase** del trabajo con cualquier base de datos comienza con sentencias **DDL** (en español Lenguaje de Definición de Datos), puesto que antes de poder almacenar y recuperar información debimos definir las estructuras donde agrupar la información: **las tablas.**

La **siguiente fase** será **manipular los datos**, es decir, trabajar con sentencias **DML** (en español Lenguaje de Manipulación de Datos). Este conjunto de sentencias está orientado a consultas y manejo de datos de los objetos creados. Básicamente consta de cuatro sentencias: **`SELECT, INSERT, DELETE y UPDATE`**. En esta unidad nos centraremos en una de ellas, que es la sentencia para consultas: **`SELECT`**.

Las sentencias SQL que se verán a continuación pueden ser ejecutadas desde un cliente con entorno gráfico como SQL Developer o DBeaver. Necesitaremos estar conectados al esquema sobre el que queramos lanzar las consultas, las cuales serán escritas en el área de edición del entorno:



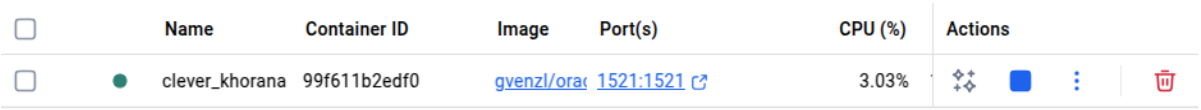
El resultado de la consulta aparece en la parte inferior de la pantalla en forma de tabla.

También se pueden indicar las sentencias SQL desde el entorno de SQL*Plus que ofrece Oracle. Dicho entorno está instalado en la máquina en la que tengamos la instancia de Oracle Express Edition. En el caso de tener la máquina virtual en Oracle Virtual Box ya tienes las instrucciones de conexión en la guía de instalación del entorno.

En el caso de utilizan un contenedor Docker, para poder ejecutar instrucciones en SQL*Plus lo primero es conectar con la máquina en la que está instalado Oracle XE. Para ello abrimos un terminal y ejecutamos:

```
docker exec -it -u oracle contenedor_docker bash
```

donde `oracle` es el usuario con el que accedemos a la máquina (por tanto es usuario de Linux, no de Oracle), y "contenedor_docker" es el nombre del contenedor. Puedes consultarlo aquí (columna Name. en este caso sería clever_khorana)



Se usa el parámetro `-u oracle` porque el entorno de la base de datos, incluyendo el comando `sqlplus`, está configurado para ese usuario, y no para `root`".

Una vez conectados aparece el prompt de la shell bash. Ahora toca conectarse con la base de datos. Sintaxis:

```
sqlplus usuario/password@//localhost:1521/XEPDB1
```

Si, por ejemplo, nos quisiéramos conectar con el usuario "libreria" para acceder a su esquema escribiríamos:

```
miguel@miguel-Zenbook-UX3402ZA-UX3402ZA:~$ docker exec -it -u oracle clever_khorana bash
bash-4.4$ sqlplus libreria/oracle123@//localhost:1521/XEPDB1
```

Una vez hecho esto deberíamos de tener acceso a los objetos de "libreria".

En el caso de utilizar SQL*Plus hay algunos comandos básicos que debes conocer para hacer más legibles los resultados de las consultas:

- **SET LINESIZE**: Este comando establece el **ancho total de la línea** que SQL*Plus mostrará antes de saltar a la siguiente. Si tus columnas de datos aparecen cortadas o en varias líneas, aumentar este valor es la solución.

  Ejemplo:
  `SET LINESIZE 200;`

- **SET PAGESIZE**: Define el **número de filas** que se mostrarán en una "página" **antes de que se repitan los encabezados de las columnas**. Un valor más alto evita que los encabezados aparezcan constantemente en medio de tus resultados. Para eliminar los encabezados por completo (excepto al principio), puedes usar `SET PAGESIZE 0`, aunque un valor como `50` o `100` suele ser más práctico.

  Ejemplo:
  `SET PAGESIZE 50;`

- **COLUMN...FORMAT**: Este es uno de los comandos más útiles para controlar el ancho y la apariencia de columnas específicas. Es especialmente útil para acortar columnas de texto largas (VARCHAR2) o para dar formato a números.
  - Para una columna de **texto**, `A` seguido de un número define el ancho.
    Ejemplo
    `COLUMN nombre_columna FORMAT A20; -- La columna 'nombre_columna' ahora tendrá un ancho de 20 caracteres.`
  - Para una columna **numérica**, 9 representa un dígito.
    Ejemplo
    `COLUMN salario FORMAT 999G999D99; -- Formatea un número como, por ejemplo, 1,234.56`

Para ejecutar cualquiera de las sentencias SQL que aprenderás en los siguientes puntos, simplemente debes escribirla completa y pulsar Intro para que se inicie su ejecución.

VISUALIZACIÓN DE LA ESTRUCTURA DE UNA TABLA

Se realiza mediante el comando SQL*Plus DESCRIBE (DESC), y muestra la descripción de las columnas de una tabla, pudiendo existir los siguientes tipos de campos:

NUMBER(p, s): donde 'p' es la precisión (el número total de dígitos) y 's' es la escala (el número de dígitos en la parte decimal).
VARCHAR2(n): hace referencia a cadenas de caracteres de longitud variable de tamaño n.
DATE: fecha/hora
CHAR(s): hace referencia a cadenas de caracteres de longitud fija de tamaño s.

Sintaxis (ambas son equivalentes):

```
DESC Nombre_tabla
DESCRIBE Nombre_tabla
```

Desde clientes de escritorio, como DBeaver o SQL Developer, utilizaremos el entorno visual para consultar la estructura de una tabla.

2. CONSULTAS SQL

La sentencia **SELECT** se utiliza para recuperar la información de la base de datos. Con ella podemos hacer lo siguiente:

- **Selección:** Podemos utilizar la capacidad de **selección** de SQL para **seleccionar** los registros de la tabla que queramos, **devueltos** por una **consulta**. Podemos utilizar varios criterios para restringir de manera selectiva los registros que queremos ver.
- **Proyección:** Podemos utilizar la capacidad de proyección para **seleccionar las columnas** en la tabla que se deseen, **devueltas** por nuestra **consulta**. Podemos seleccionar **tantas tablas como necesitemos.**
- **Unión:** Permite extraer información de **distintas tablas** mediante la creación de **enlaces** entre una columna de las dos tablas compartidas.

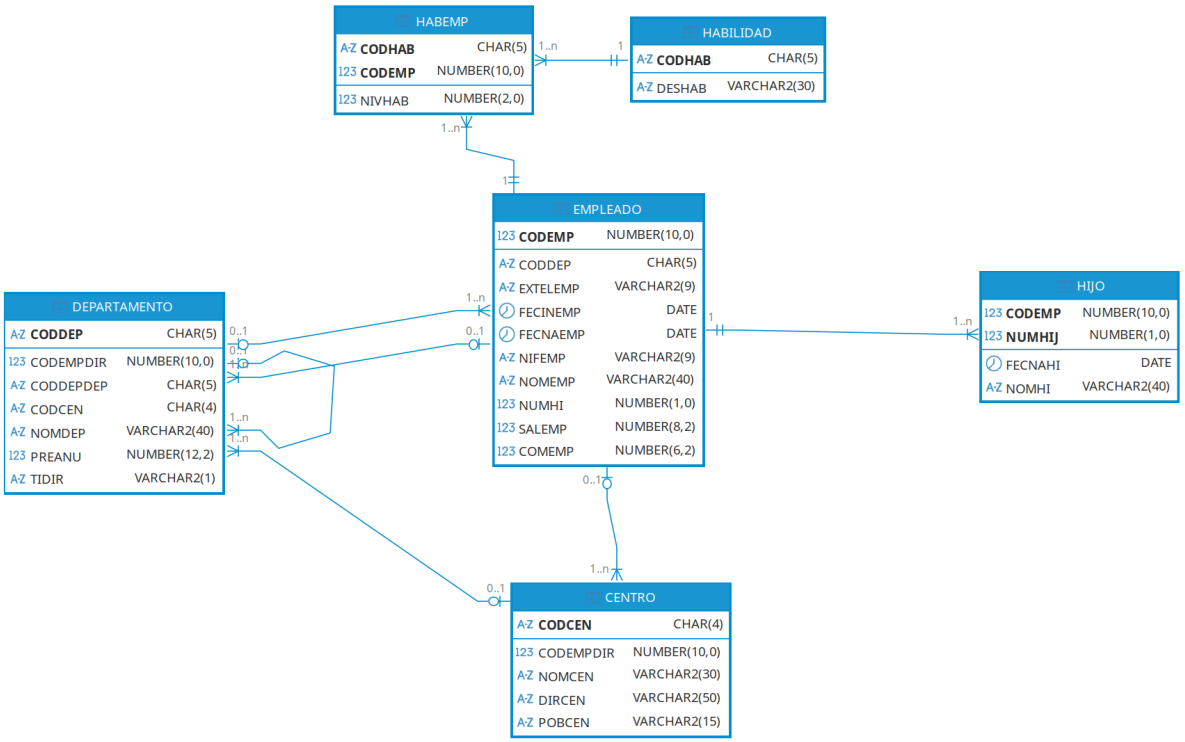
Su sintaxis es la siguiente:

```
SELECT [ALL | DISTINCT]
[expre_colum1 [[AS] alias1],expre_colum2 [[AS]
alias2],...,expre_column [[AS] aliasn] | *]
FROM   [nombre_tabla1, nombre_tabla2,......,nombre_tablan]
[WHERE condición]
[ORDER BY expr_colum [DESC|ASC]  [,expre_colum
[DESC|ASC]....];;
```

Si incluyes la cláusula ALL después de SELECT, indicas que quieres seleccionar todas las filas estén o no repetidas. Es el valor por defecto y no se suele especificar.

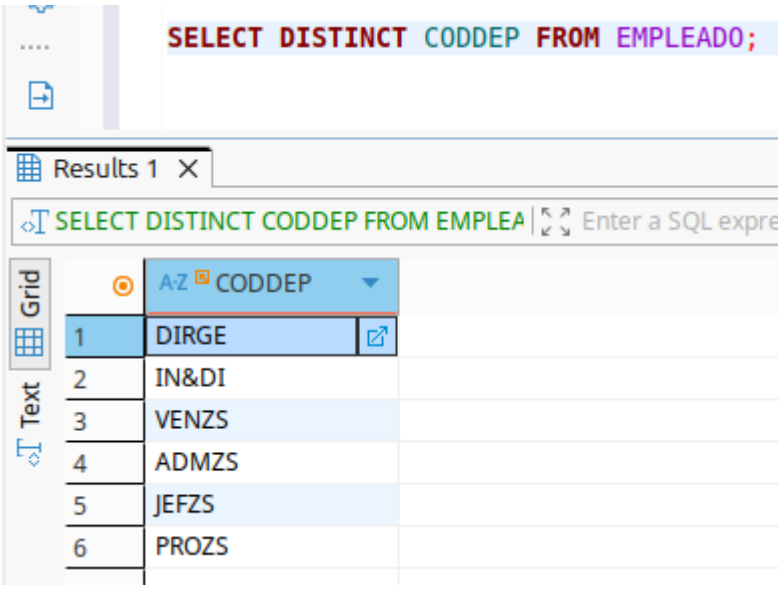
Las filas duplicadas se eliminan usando la cláusula `DISTINCT` en la instrucción `SELECT`.

Para los ejemplos utilizaremos la BBDD del script 06bd-empresa.sql. Para su creación generamos un esquema propio y posteriormente ejecutamos el script que encontramos en el aula virtual. El diagrama de tablas de esta base de datos es:



Ejemplo de consulta:

```
SELECT DISTINCT coddep
FROM empleado;
```



2.1 PROYECCIÓN

Para seleccionar **campos devueltos** por la consulta utilizaremos bien * para indicar que queremos **todas las columnas**, o indicaremos de uno en uno el nombre de los campos que queremos recuperar;

A tener en cuenta:

- **Se pueden nombrar** a las columnas anteponiendo el nombre de la tabla de la que proceden, pero esto es opcional (cuando trabajamos con una sola tabla), y quedaría: NombreTabla.NombreColumna
- Si queremos incluir todas las columnas de una tabla podemos utilizar el **comodín** asterisco ("*"). Quedaría así: `SELECT * FROM NombreTabla;`
- **expre_column** puede ser **el nombre** de las columnas, **constantes**, **expresiones** o **funciones** SQL.

La cláusula **`FROM`** especifica la **tabla o lista de tablas** de donde se obtendrán los datos. Si se utiliza más de una, éstas deben aparecer separadas por comas. A este tipo de consulta se denomina consulta combinada o join.

También puedes añadir el nombre del usuario que es **propietario** de esas tablas, o nombre del esquema al que pertenece la tabla, indicándolo de la siguiente manera:

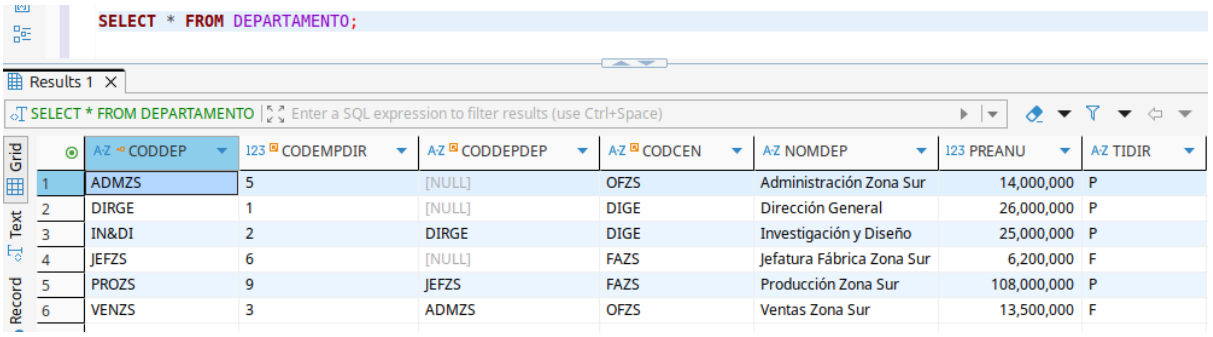
USUARIO.TABLA

De este modo podemos distinguir entre las tablas de un usuario y otro cuando trabajamos con diferentes esquemas.

Ejemplos:

```
SELECT *
FROM DEPARTAMENTO;
```

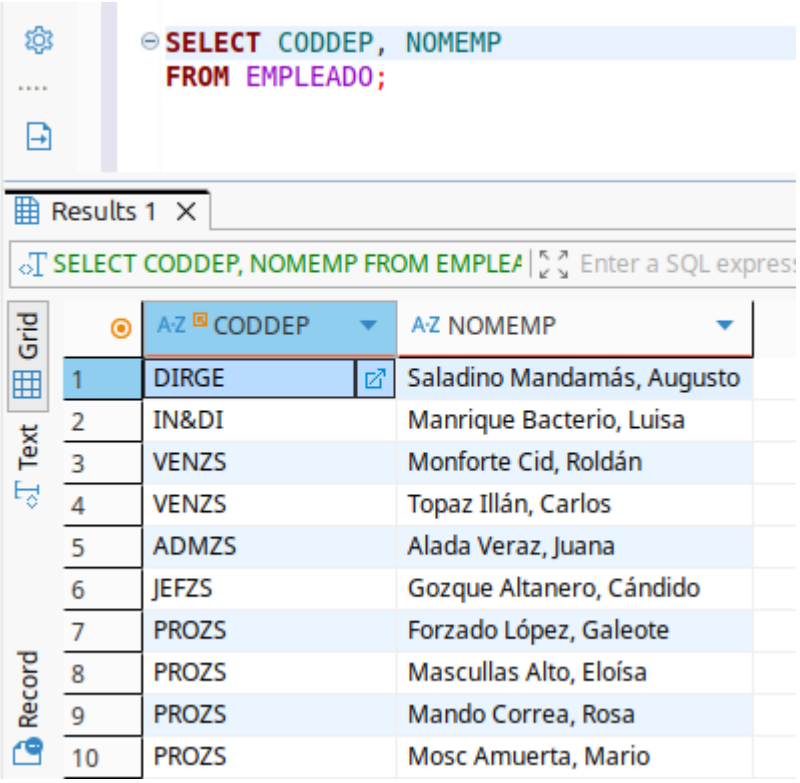
Extrae **todos** los elementos de la tabla DEPARTAMENTO.



	CODDEP	CODEMPDIR	CODDEPDEP	CODCEN	NOMDEP	PREANU	TIDIR
1	ADMZS	5	[NULL]	OFZS	Administración Zona Sur	14,000,000	P
2	DIRGE	1	[NULL]	DIGE	Dirección General	26,000,000	P
3	IN&DI	2	DIRGE	DIGE	Investigación y Diseño	25,000,000	P
4	JEFZS	6	[NULL]	FAZS	Jefatura Fábrica Zona Sur	6,200,000	F
5	PROZS	9	JEFZS	FAZS	Producción Zona Sur	108,000,000	P
6	VENZS	3	ADMZS	OFZS	Ventas Zona Sur	13,500,000	F

```
SELECT CODDEP, NOMEMP
FROM EMPLEADO;
```

Muestra las **columnas CODEMP y NOMEMP** de la tabla EMPLEADO.



Escritura de una sentencia SQL:

- Los comandos pueden constar de una o varias líneas.
- Las tabulaciones pueden ser usadas por comodidad.
- Las abreviaturas y separaciones de palabras no están permitidas.
- Los comandos no son sensibles a las mayúsculas y a las minúsculas.
- Colocar punto y coma al final de la última cláusula.

2.1.1 EXPRESIONES ARITMÉTICAS

Los **operadores básicos son: + , - , * , /** , y permiten realizar operaciones aritméticas sobre el valor de los campos.

Ejemplo:

```
SELECT NOMEMP, SALEMP, SALEMP + SALEMP * 0.05
FROM EMPLEADO;
```

```
SELECT NOMEMP, SALEMP, SALEMP + SALEMP * 0.05
FROM EMPLEADO;
```

Results 1

SELECT NOMEMP, SALEMP, SALEMP + SAL | Enter a SQL expression to filter results (use Ctrl+Spa

	NOMEMP	SALEMP	SALEMP+SALEMP*0.05
1	Saladino Mandamás, Augusto	7,200	7,560
2	Manrique Bacterio, Luisa	4,500	4,725
3	Monforte Cid, Roldán	5,200	5,460
4	Topaz Illán, Carlos	3,200	3,360
5	Alada Veraz, Juana	6,200	6,510
6	Gozque Altanero, Cándido	5,000	5,250
7	Forzado López, Galeote	1,600	1,680
8	Mascullas Alto, Eloísa	1,600	1,680
9	Mando Correa, Rosa	3,100	3,255
10	Mosc Amuerta, Mario	1,300	1,365

El orden general de **prioridad** es el siguiente:
* / + -: Los operadores * y / tienen prioridad sobre los operadores + y - . Los operadores de la **misma prioridad** se evalúan de **izquierda a derecha**.

Los **paréntesis** () se utilizan para **modificar las prioridades**.

VALOR NULL: Un valor **NULL** es un valor que es inaccesible, sin valor conocido o inaplicable. No representa ni un cero ni un espacio en banco, ya que el cero es un número y blanco un carácter.

Si operamos con un NULL la expresión tomará el valor nulo.

2.1.2 EL OPERADOR CONCATENACIÓN

El operador de concatenación:

- Está representado por dos barras verticales (||)
- Vincula columnas o cadenas de caracteres con otras columnas.
- Crea una columna resultado que es una expresión de tipo carácter.
- Se pueden utilizar alias de columnas para la concatenación.

Ejemplo:

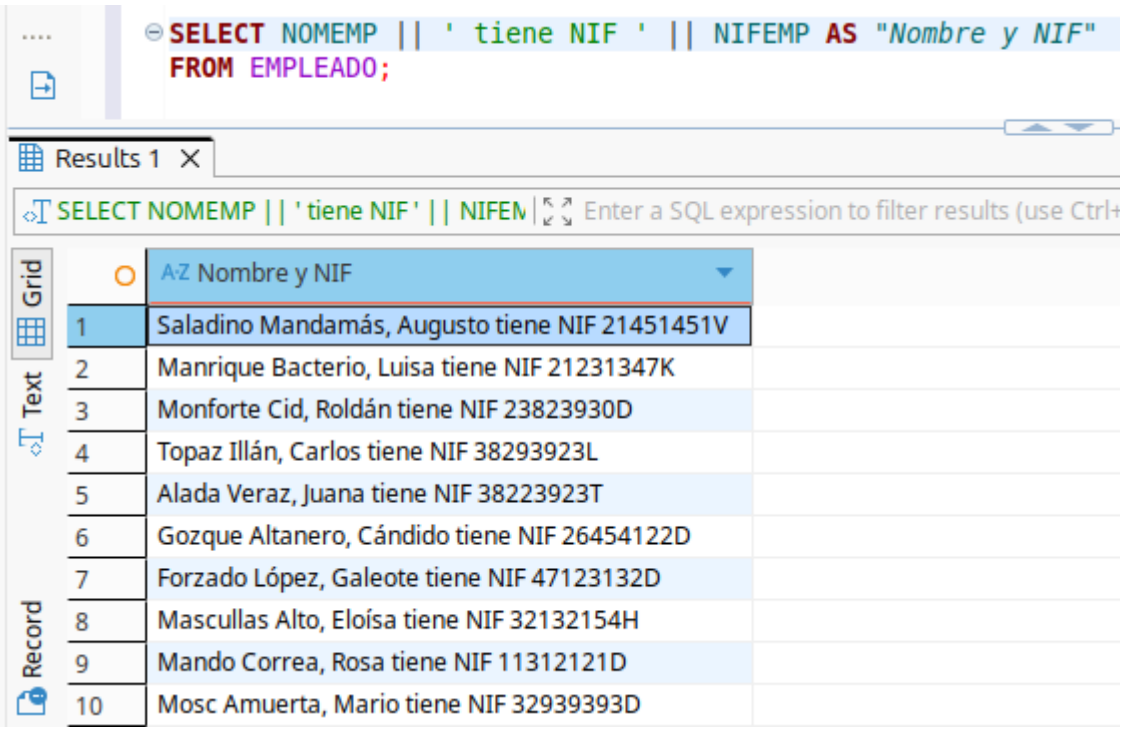
```
SELECT NOMEMP || CODDEP
FROM EMPLEADO;
```

Cadenas de caracteres:

- Un literal es un carácter, expresión o número incluido en la lista de la cláusula SELECT.
- Los valores literales de tipo fecha y carácter deben ir entre comillas simples.
- Por cada fila devuelta se genera una cadena de caracteres.

Ejemplo:

```
SELECT NOMEMP ||' tiene NIF ' || NIFEMP AS "Nombre y NIF"
FROM EMPLEADO;
```



2.1.3 ALIAS DE COLUMNAS

- Un alias de columna renombra un encabezamiento de columna.
- Es útil especialmente en cálculos.
- Sigue inmediatamente al nombre de la columna. Se puede utilizar la palabra clave opcional AS entre el nombre de la columna y el alias.
- Se requiere encerrar el alias entre comillas dobles si contienen espacios en blanco, caracteres especiales o palabras clave.

Ejemplo.

```
SELECT NOMEMP, SALEMP, (SALEMP + SALEMP * 0.05) NUEVO_SALARIO
FROM EMPLEADO;
```

```
SELECT NOMEMP, SALEMP, (SALEMP + SALEMP * 0.05) NUEVO_SALARIO
FROM EMPLEADO;
```

Results 1

SELECT NOMEMP, SALEMP, (SALEMP + SA | Enter a SQL expression to filter results (use Ctrl+Spa

	NOMEMP	SALEMP	NUEVO_SALARIO
1	Saladino Mandamás, Augusto	7,200	7,560
2	Manrique Bacterio, Luisa	4,500	4,725
3	Monforte Cid, Roldán	5,200	5,460
4	Topaz Illán, Carlos	3,200	3,360
5	Alada Veraz, Juana	6,200	6,510
6	Gozque Altanero, Cándido	5,000	5,250
7	Forzado López, Galeote	1,600	1,680
8	Mascullas Alto, Eloísa	1,600	1,680
9	Mando Correa, Rosa	3,100	3,255
10	Mosc Amuerta, Mario	1,300	1,365

2.2 LA CLÁUSULA WHERE (SELECCIÓN)

Podemos restringir las filas recuperadas usando la cláusula **WHERE**. Una cláusula WHERE contiene una **condición** que se deben cumplir los registros devueltos durante la selección, y se escribe a continuación de la cláusula FROM.

El **criterio de búsqueda o condición** puede ser más o menos sencillo y para crearlo se pueden conjugar operadores de diversos tipos, funciones o expresiones más o menos complejas. Su formato es: *expresión operador expresión*

Expresión puede ser una **constante**, una **expresión aritmética**, un valor **NULL** o un **nombre de columna**. Se pueden utilizar operadores de **comparación** y **operadores lógicos.**

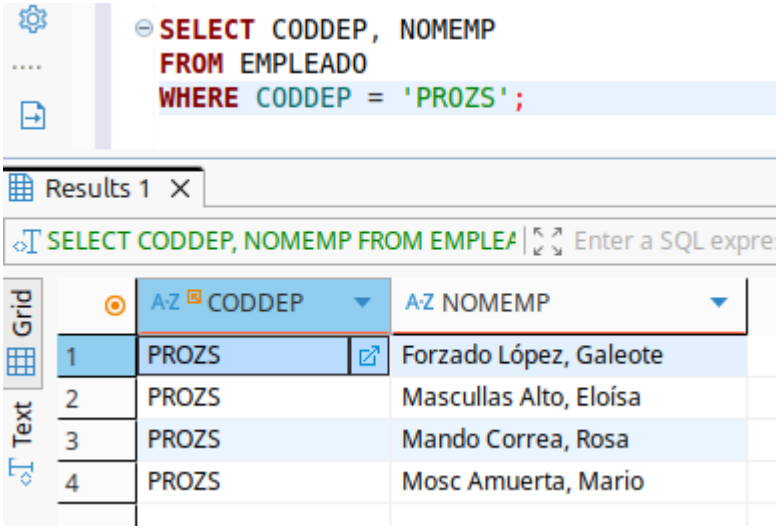
- Operadores de **comparación**: = > < >= <= != <>
- Operadores **lógicos**: **AND** (devuelve verdadero si sus expresiones a derecha e izquierda son ambas verdaderas), **OR** (devuelve verdadero si alguna de sus expresiones a derecha o izquierda son verdaderas) y **NOT** (invierte el resultado de la condición que le **sigue**, si la expresión es verdadera devuelve falsa y si es falsa devuelve verdadera)
- Operadores de **conjunto de valores: IN** (Igual que cualquiera de los miembros entre paréntesis), **BETWEEN…AND** (Entre. Contenido dentro del rango). Los valores especificados por BETWEEN se incluyen en la operación. BETWEEN se puede usar con fechas y cadenas, siempre entre comillas simples.

- Operadores de **comparación de cadena de caracteres: `LIKE`** `'_abc%'`. 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.
- Para filtrar una columna por valores NULL escribiremos `IS NULL`. Por el contrario, si queremos que los valores NULL sean excluidos escribiremos la condición `IS NOT NULL`.

Ejemplos:

Extrae código de departamento y nombre para los empleados cuyo código de departamento sea 'PROZS':

```
SELECT CODDEP, NOMEMP
FROM EMPLEADO
WHERE CODDEP = 'PROZS';
```



Extrae nombre y salario para los empleados cuyo salario mensual es superior a 5000€:

```
SELECT NOMEMP, SALEMP FROM EMPLEADO
WHERE SALEMP > 5000;
```

```
SELECT NOMEMP, SALEMP FROM EMPLEADO
WHERE SALEMP > 5000;
```

Results 1

SELECT NOMEMP, SALEMP FROM EMPLEA | Enter a SQL exp

	NOMEMP	SALEMP
1	Saladino Mandamás, Augusto	7,200
2	Monforte Cid, Roldán	5,200
3	Alada Veraz, Juana	6,200

```
SELECT *
FROM HIJO
WHERE NOMHI LIKE '%Mando%';
```

Results 1

SELECT * FROM HIJO WHERE NOMHI LIKE | Enter a SQL expression to filter results (use Ctrl+Space)

	CODEMP	NUMHIJ	FECNAHI	NOMHI
1	9	1	1988-02-28 00:00:00.000	León Mando, Elvira
2	9	2	1990-07-18 00:00:00.000	León Mando, Plácido

```
SELECT *
FROM DEPARTAMENTO
WHERE PREANU BETWEEN 14000000 AND 26000000;
```

ts 1

CT * FROM DEPARTAMENTO WHERE I | Enter a SQL expression to filter results (use Ctrl+Space)

CODDEP	CODEMPDIR	CODDEPDEP	CODCEN	NOMDEP	PREANU	TIDIR
ADMZS	5	[NULL]	OFZS	Administración Zona Sur	14,000,000	P
DIRGE	1	[NULL]	DIGE	Dirección General	26,000,000	P
IN&DI	2	DIRGE	DIGE	Investigación y Diseño	25,000,000	P

EJERCICIO:

Extraer de la tabla EMPLEADO:

- Columnas CODDEP y NOMEMP para aquellos empleados cuyo CODDEP no sea PROZS ni VENZS.
- Todas las columnas de la tabla EMPLEADO cuyo CODDEP sea PROZS y su NIF acabe en la letra 'D'.
- Nombre completo, código de departamento y número de hijos para los empleados que pertenezcan al departamento PROZS o tengan 1 o más hijos.
- El nombre y extensión telefónica de todos los empleados. Excluir de la consulta los registros que no tengan extensión asignada.

```
SELECT CODDEP, NOMEMP
FROM EMPLEADO
WHERE CODDEP NOT IN ('PROZS','VENZS');

SELECT *
FROM EMPLEADO
WHERE CODDEP = 'PROZS' AND NIFEMP LIKE '%D';

SELECT NOMEMP, CODDEP, NUMHI
FROM EMPLEADO
WHERE CODDEP = 'PROZS' OR NUMHI >= 1;

SELECT NOMEMP, EXTELEMP
FROM EMPLEADO
WHERE EXTELEMP IS NOT NULL;
```

2.2.1 PRECEDENCIA DE OPERADORES

Con frecuencia utilizaremos la sentencia `SELECT` acompañada de expresiones muy extensas, y resultará difícil saber qué parte de dicha expresión se evaluará primero, por ello es conveniente conocer el orden de precedencia que tiene Oracle:

1. Se evalúa la multiplicación (*) y la división (/) al mismo nivel.
2. A continuación sumas (+) y restas (-).
3. Concatenación (| |).
4. Todas las comparaciones (<, >, …).
5. Después evaluaremos los operadores IS NULL, IN, LIKE, BETWEEN.
6. NOT.
7. AND.
8. OR.

Si quisiéramos variar este orden necesitaríamos utilizar paréntesis.

2.3 LA CLÁUSULA ORDER BY. ORDENACIÓN DE REGISTROS

`ORDER BY` especifica el criterio de **clasificación u ordenación** de la respuesta de la consulta. Prevalece el de la izquierda cuando hay varios atributos.

```
ORDER BY expre_columna [DESC|ASC] [,expre_columna [DESC|ASC]...]
```

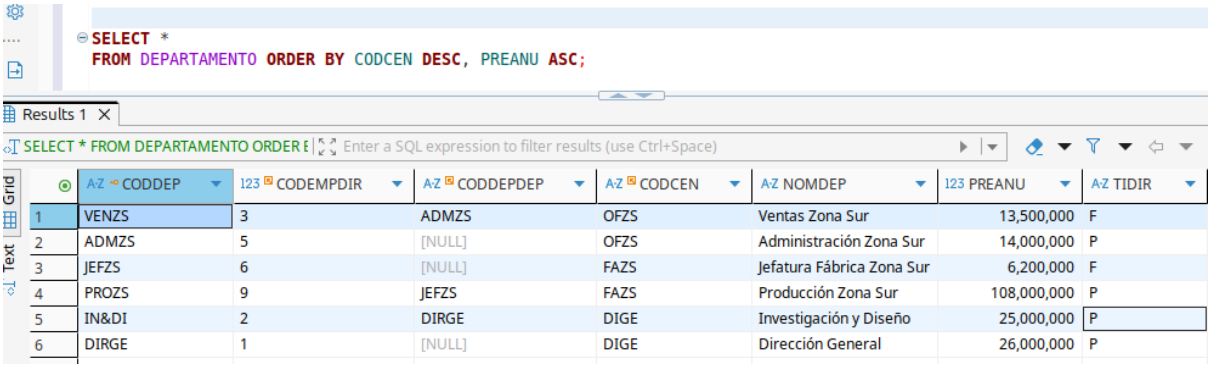
Después de cada columna de ordenación se puede incluir el tipo de ordenación (ascendente o descendente) utilizando las palabras reservadas **`ASC`** **o** **`DESC`**. Por **defecto**, y si no se pone nada, la ordenación es **ascendente**.

Debes saber que es posible ordenar por **más de una columna**. Es más, puedes ordenar no solo por columnas sino a través de una expresión creada con columnas, una constante (aunque no tendría mucho sentido) o funciones SQL.

No todos los tipos de campos te servirán para ordenar, únicamente aquellos de tipo **carácter, número o fecha.**

La siguiente consulta devuelve una lista de departamentos ordenados de forma descendente en función del código del centro al que pertenecen. En caso de haber dos en el mismo centro, ordenamos los resultados por presupuesto de forma ascendente.

```
SELECT *
FROM DEPARTAMENTO
ORDER BY CODCEN DESC, PREANU ASC;
```

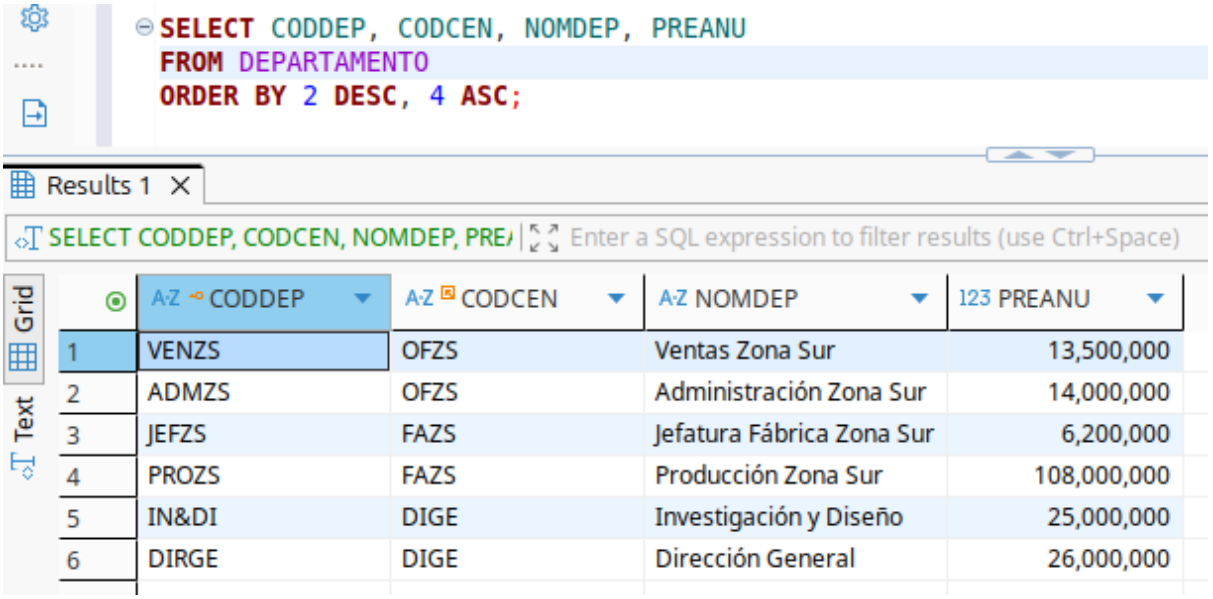


	CODDEP	CODEMPDIR	CODDEPDEP	CODCEN	NOMDEP	PREANU	TIDIR
1	VENZS	3	ADMZS	OFZS	Ventas Zona Sur	13,500,000	F
2	ADMZS	5	[NULL]	OFZS	Administración Zona Sur	14,000,000	P
3	JEFZS	6	[NULL]	FAZS	Jefatura Fábrica Zona Sur	6,200,000	F
4	PROZS	9	JEFZS	FAZS	Producción Zona Sur	108,000,000	P
5	IN&DI	2	DIRGE	DIGE	Investigación y Diseño	25,000,000	P
6	DIRGE	1	[NULL]	DIGE	Dirección General	26,000,000	P

Puedes colocar el número de orden del campo por el que quieres que se ordene en lugar de su nombre, es decir, referenciar a los campos por su posición en la lista de selección. Por ejemplo, para la consulta anterior si únicamente solicitamos el código de departamento, el código de centro, el nombre de departamento y el presupuesto anual:

```
SELECT CODDEP, CODCEN, NOMDEP, PREANU
```

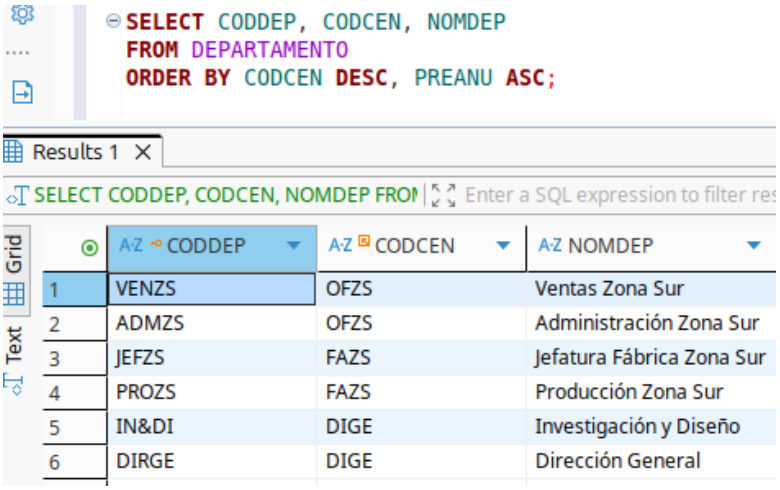
```
FROM DEPARTAMENTO
ORDER BY 2 DESC, 4 ASC;
```



Si colocamos un número mayor a la cantidad de campos de la lista de selección, aparece un mensaje de error y la sentencia no se ejecuta.

También es posible seleccionar por una columna no extraída en la selección:

```
SELECT CODDEP, CODCEN, NOMDEP
FROM DEPARTAMENTO
ORDER BY CODCEN DESC, PREANU ASC;
```



3. FUNCIONES

En casi todos los Sistemas Gestores de Base de Datos existen funciones ya creadas que facilitan la creación de consultas más complejas. Dichas funciones varían según el SGBD, veremos aquí las que utiliza Oracle.

Las funciones son realmente operaciones que se realizan sobre los datos y que realizan un determinado cálculo. Para ello necesitan unos datos de entrada llamados parámetros o argumentos y en función de éstos, se realizará el cálculo de la función que se esté utilizando. Normalmente los parámetros se especifican entre paréntesis. Las funciones se pueden utilizar para:

- Realizar cálculos en general sobre datos.
- Modificar ítem de datos individuales.
- Manipular la salida de grupos de fila.
- Modificar formatos de los datos para su presentación.
- Convertir los tipos de datos de las columnas.

Se pueden incluir en las cláusulas **`SELECT, WHERE y ORDER BY`**, y se especifican de la siguiente manera:

```
NombreFunción [(parámetro1, [parámetro2, …)]
```

Existen dos tipos de funciones en SQL:

1. Funciones a nivel fila: Devuelven un valor por cada fila seleccionada en la tabla, pueden ser de carácter, numéricas, de fecha, de conversión y generales.
2. Funciones a nivel de grupo: Devuelven un valor por un grupo de filas.

Oracle proporciona una tabla con la que podemos hacer pruebas, esta tabla se llama `Dual` y contiene un único campo llamado `DUMMY` y una sola fila. Utilizaremos la tabla `Dual` en algunos de los ejemplos que vamos a ver en los siguientes apartados.

3.1 FUNCIONES A NIVEL DE FILA

Las **funciones a nivel fila** pueden ser:

- **Funciones numéricas:** Aceptan números como datos de entrada y devuelven valores numéricos.
- **Funciones de caracteres:** Aceptan caracteres como datos de entrada y pueden devolver caracteres o números.
- **Funciones fecha:** Operan con valores de dato tipo fecha.
- **Funciones de conversión:** Convierten un valor de un tipo de dato.
- **Funciones generales.**

3.1.1. FUNCIONES NUMÉRICAS

Estas funciones actúan sobre un número, unas variables o una columna de una tabla.

Para probar estas funciones vamos a utilizar la tabla DUAL (tabla de trabajo de Oracle, que sirve para probar funciones o hacer cálculos rápidos).

Las funciones de valores simples son las siguientes:

Para trabajar con campos de tipo número tenemos las siguientes funciones:

- **`ABS(n):`** Calcula el valor ABSoluto de un número n.

Ejemplo: `SELECT ABS(-17) FROM DUAL; -- Resultado: 17`

- **`EXP(n):`** Calcula e^n, es decir, el exponente en base e del número n.

Ejemplo: `SELECT EXP(2) FROM DUAL; -- Resultado: 7,38`

- **`CEIL(n):`** Calcula el valor entero inmediatamente superior o igual al argumento n.

Ejemplo: `SELECT CEIL(17.4) FROM DUAL; -- Resultado: 18`

- **`FLOOR(n):`** Calcula el valor entero inmediatamente inferior o igual al parámetro n.

Ejemplo: `SELECT FLOOR(17.4) FROM DUAL; -- Resultado: 17`

- **`MOD(m,n):`** Calcula el resto resultante de dividir m entre n.

Ejemplo: `SELECT MOD(15, 2) FROM DUAL; --Resultado: 1`

- **`POWER(valor, exponente):`** Eleva el valor al exponente indicado.

Ejemplo: `SELECT POWER(4, 5) FROM DUAL; -- Resultado: 1024`

- **`ROUND(n, decimales):`** Redondea el número n al siguiente número con el número de decimales que se indican.

Ejemplo: `SELECT ROUND(12.5874, 2) FROM DUAL; -- Resultado: 12.59`

- **`SQRT(n):`** Calcula la raíz cuadrada de n.

Ejemplo: `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.

Ejemplos:

```
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 un valor positivo, retorna 1, si es negativo, devuelve -1 y 0 si es 0.

Ejemplo: `SELECT SIGN(-23) FROM DUAL; – Resultado: -1`

EJERCICIOS DE FUNCIONES ARITMÉTICAS

1) Obtener el valor absoluto de –20

```
SELECT ABS(-20) "ABSOLUTO" FROM DUAL;
```

2) Obtener el valor absoluto de SALARIO - 10.000 para todas las filas de la tabla EMPLE.

```
SELECT ABS(SALEMP - 10000) "ABSOLUTO" FROM EMPLEADO;
```

3) Obtener el valor entero inmediatamente superior a 20.7, 20.2, 16, -20.7, -20.2, -16

```
SELECT CEIL(20.7)"SUPERIOR" FROM DUAL; -- 21
SELECT CEIL(20.2)"SUPERIOR" FROM DUAL; -- 21
SELECT CEIL(16)"SUPERIOR" FROM DUAL; -- 16
SELECT CEIL(-20.7)"SUPERIOR" FROM DUAL; -- -20
SELECT CEIL(-20.2)"SUPERIOR" FROM DUAL; -- -20
SELECT CEIL(-16)"SUPERIOR" FROM DUAL; -- -16
SELECT CEIL(20.7), CEIL(20.2)FROM DUAL;
```

4) Los mismos para el inmediatamente inferior.

```
SELECT FLOOR(20.7)"INFERIOR" FROM DUAL; -- 20
SELECT FLOOR(20.2)"INFERIOR" FROM DUAL; -- 20
SELECT FLOOR(16)"INFERIOR" FROM DUAL;-- 16
SELECT FLOOR(-20.7)"INFERIOR" FROM DUAL;-- -21
SELECT FLOOR(-20.2)"INFERIOR" FROM DUAL;-- -21
```

```
SELECT FLOOR(-16)"INFERIOR" FROM DUAL;-- -16
```

5) Obtener el resto de dividir 11 entre 4, entre 0 y entre 15.

```
SELECT MOD(11,4) "RESTO" FROM DUAL; -- 3
SELECT MOD(11,0) "RESTO" FROM DUAL; -- 11
SELECT MOD(11,15) "RESTO" FROM DUAL; -- 11
```

6) Obtener el resto de dividir -10/3, 10/-3, 10.4/4.5

```
SELECT MOD(-10,3) "RESTO" FROM DUAL; -- -1
SELECT MOD(10,-3) "RESTO" FROM DUAL; -- 1
SELECT MOD(10.4,4.5)"RESTO" FROM DUAL; -- 1.4
```

7) Calcular 3 elevado a 4, 3 elevado a –4, -3 elevado a 4 y 4.5 elevado a 2.4

```
SELECT POWER (3,4) "ELEVADO" FROM DUAL;-- 81
SELECT POWER (3,-4) "ELEVADO" FROM DUAL;-- 0,012345679
SELECT POWER (-3,4) "ELEVADO" FROM DUAL; -- 81
SELECT POWER (4.5,2.4) "ELEVADO" FROM DUAL; -- 36.95
```

8) Redondear 1.5634 con un decimal y sin decimales.

```
SELECT ROUND (1.5634,1) "REDONDEO" FROM DUAL; -- 1.6
SELECT ROUND (1.5634) "REDONDEO" FROM DUAL; -- 2
```

9) Redondear 1.2324 con dos decimales y sin decimal

```
SELECT ROUND (1.2324,2) "REDONDEO" FROM DUAL; -- 1.23
SELECT ROUND (1.2324,0) "REDONDEO" FROM DUAL;-- 1
```

10) Redondear 145.5 un lugar a la izq. de la coma, dos y tres.

```
SELECT ROUND (145.5,-1) "REDONDEO"FROM DUAL; -- 150
SELECT ROUND (145.5,-2) "REDONDEO"FROM DUAL;-- 100
SELECT ROUND (145.5,-3) "REDONDEO"FROM DUAL; -- 0
```

11) Obtener el SIGNo de –10 y 10

```
SELECT SIGN(-10) "SIGNO" FROM DUAL; -- -1
SELECT SIGN(10) "SIGNO" FROM DUAL; -- 1
```

12) Obtener la raíz cuadrada de 25 y 25.6.

```
SELECT SQRT(25) "RAIZ CUADRADA" FROM DUAL; -- 5
SELECT SQRT(25.6) "RAIZ CUADRADA" FROM DUAL; -- 5.05
```

13) Truncar el número 1.5634 a un decimal, 2 y cero.

```
SELECT TRUNC (1.5634,1) "TRUNCADO" FROM DUAL;-- 1.5
SELECT TRUNC (1.5634,2) "TRUNCADO" FROM DUAL;-- 1.56
SELECT TRUNC (1.5634,0) "TRUNCADO" FROM DUAL;-- 1
```

14) Truncar el número 187.98 a la izquierda de la coma a 1 posición, 2 y 3.

```
SELECT TRUNC (187.98,-1) "TRUNCADO" FROM DUAL;-- 180
SELECT TRUNC (187.98,-2) "TRUNCADO" FROM DUAL;-- 100
SELECT TRUNC (187.98,-3) "TRUNCADO" FROM DUAL;-- 0
```

3.1.2 FUNCIONES DE CADENA DE CARACTERES

Son las funciones disponibles para manipular **campos de tipo carácter o cadena de caracteres.** Como **resultado** podremos obtener **caracteres o números**. Estas son las funciones más habituales:

- **`CHR(n):`** Devuelve el carácter cuyo valor codificado es n.

Ejemplo: `SELECT CHR(81) FROM DUAL; --Resultado: Q`

- **`ASCII(n):`** Devuelve el valor ASCII de n.

Ejemplo: `SELECT ASCII('O') FROM DUAL; --Resultado: 79`

- **`CONCAT(cad1, cad2):`** Devuelve las dos cadenas unidas. Es equivalente al operador ||

Ejemplo: `SELECT CONCAT('Hola', 'Mundo') FROM DUAL; --Resultado: HolaMundo`

- **`LOWER(cad):`** Devuelve la cadena cad con todos sus caracteres en minúsculas.

Ejemplo: `SELECT LOWER('En MInúsculAS') FROM DUAL; --Resultado: en minúsculas`

- **`UPPER(cad):`** Devuelve la cadena cad con todos sus caracteres en mayúsculas.

Ejemplo: `SELECT UPPER('En MAyúsculAS') FROM DUAL; --Resultado: EN MAYÚSCULAS`

- **`INITCAP(cad):`** Devuelve la cadena cad con el primer carácter de cada palabra en mayúscula.

Ejemplo: `SELECT INITCAP('ana martínez') FROM DUAL; --Resultado: Ana Martínez`

- **`LPAD(cad1, n, cad2):`** Devuelve cad1 con longitud n, ajustada a la derecha, rellenando por la izquierda con cad2.

Ejemplo: `SELECT LPAD('M', 5, '*') FROM DUAL; --Resultado: ****M`

- **`RPAD(cad1, n, cad2):`** Devuelve cad1 con longitud n, ajustada a la izquierda, rellenando por la derecha con cad2.

Ejemplo: `SELECT RPAD('M', 5, '*') FROM DUAL; --Resultado: M****`

- **`REPLACE(cad, ant, nue):`** Devuelve cad en la que cada ocurrencia de la cadena ant ha sido sustituida por la cadena nue.

Ejemplo: `SELECT REPLACE('correo@gmail.es', 'es', 'com') FROM DUAL; --Resultado: correo@gmail.com`

- **`TRANSLATE(cad, 'abc', 'xyz'):`** Funciona como un **diccionario de sustitución letra por letra**. Reemplaza cada carácter individualmente basándose en su posición en dos cadenas de mapeo.

Ejemplo: `SELECT TRANSLATE('c*l*r+d*', '*+', 'oa') FROM DUAL; --Resultado: colorado`

- **`SUBSTR(cad, m, n):`** Devuelve la cadena cad compuesta por n caracteres a partir de la posición m.

Ejemplo: `SELECT SUBSTR('1234567', 3, 2) FROM DUAL; --Resultado: 34`

- **`LENGTH(cad):`** Devuelve la longitud de cad.

Ejemplo: `SELECT LENGTH('hola') FROM DUAL; --Resultado: 4`

- **`TRIM(cad):`** Elimina los espacios en blanco a la izquierda y la derecha de cad.

Ejemplo: `SELECT TRIM(' Hola de nuevo ') FROM DUAL; --Resultado: Hola de nuevo`

- **`LTRIM(cad):`** Elimina los espacios a la izquierda que posea cad.

Ejemplo: `SELECT LTRIM(' Hola') FROM DUAL; --Resultado: Hola`

- **`RTRIM(cad):`** Elimina los espacios a la derecha que posea cad.

Ejemplo: `SELECT RTRIM('Hola ') FROM DUAL; --Resultado: Hola`

- **`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.

Ejemplo:

```
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
```

EJERCICIOS DE CADENAS

- Devolver la letra cuyo valor ASCII es 65

SELECT CHR (65) FROM DUAL;

- Obtener en una columna el apellido y el departamento de cada uno de los empleados de la tabla EMPLEADO como en este ejemplo: Saladino Mandamás, Augusto trabaja en DIRGE.

```
SELECT CONCAT(CONCAT(NOMEMP,' trabaja en '),CODDEP) FROM EMPLEADO;
SELECT NOMEMP||' trabaja en '||CODDEP FROM EMPLEADO;
```

- Obtener el nombre completo de los empleados de EMPLEADO en minúsculas

```
SELECT LOWER(NOMEMP) FROM EMPLEADO;
```

- Visualizar en mayúscula, minúscula y tipo título la cadena 'oRaCle Y sql'

```
SELECT UPPER('oRaCLe Y sql') FROM DUAL;
SELECT LOWER('oRaCLe Y sql') FROM DUAL;
SELECT INITCAP('oRaCLe Y sql') FROM DUAL;
```

- Para cada fila de la tabla DEPARTAMENTO obtener el nombre del departamento. Todos los resultados tendrán exactamente 30 caracteres, rellenando por la izquierda con puntos en caso necesario si la longitud es menor.

```
SELECT LPAD(NOMDEP, 30, '.') FROM DEPARTAMENTO;
```

- Concatenar las cadenas ' HOLA', 'ADIOS ' y '*' suprimiendo espacios en blanco.

```
SELECT LTRIM('         HOLA') || RTRIM('ADIOS        ') || '*' FROM
DUAL;
```

- Visualizar el nombre y su primera letra para la tabla EMPLEADO

```
SELECT NOMEMP, SUBSTR(NOMEMP,1,1)FROM EMPLEADO;
```

- Partiendo de la cadena 'ABCDEF' obtenemos 2 caracteres a partir de la 3ª posición. 2 caracteres a partir de la 3ª posición x el final, y después a partir de la 4ª posición

```
SELECT SUBSTR('ABCDEF',3,2)FROM DUAL;
SELECT SUBSTR('ABCDEF',-3,2)FROM DUAL;
SELECT SUBSTR('ABCDEF',4)FROM DUAL;
```

- Visualizar el apellido y la primera letra del apellido de cada empleado en minúsculas.

```
SELECT NOMEMP, LOWER(SUBSTR(NOMEMP,1,1)) FROM EMPLEADO;
```

- A partir de la cadena 'LOS PILARES DE LA TIERRA' sustituir A por a, E por e, I por i, O por o y U por u.

```
SELECT TRANSLATE ('LOS PILARES DE LA TIERRA', 'AEIOU','aeiou') FROM DUAL;
```

- Sustituir 'O' por 'A' en la cadena 'BLANCO Y NEGRO'

```
SELECT TRANSLATE ('BLANCO Y NEGRO','O','A') FROM DUAL;
```

- Sustituir 'O' por 'AS' en la cadena 'BLANCO Y NEGRO'

  ```
  SELECT REPLACE ('BLANCO Y NEGRO','O','AS') FROM DUAL;
  ```

- Obtener el valor ASCII de Ñ y ñ.

```
SELECT ASCII('Ñ')FROM DUAL;
SELECT ASCII('ñ')FROM DUAL;
```

- Encontrar la posición de la primera ocurrencia de la letra 'A' en la columna Apellido de EMPLE.

```
SELECT NOMEMP, INSTR(NOMEMP,'A') FROM EMPLEADO;
```

- Calcula el número de caracteres de la columna NOMHI para todas las filas de HIJO.

```
SELECT NOMHI, LENGTH(NOMHI) FROM HIJO;
```

3.1.3 FUNCIONES DE CONVERSIÓN

Los SGBD tienen funciones que pueden **pasar de un tipo de dato a otro.** Oracle **convierte automáticamente** datos de manera que el resultado de una expresión tenga sentido. Por tanto, de manera automática se pasa de texto a número y al revés. Ocurre lo mismo para pasar de tipo texto a fecha y viceversa. Pero existen ocasiones en que queremos realizar esas conversiones de modo explícito, para lo que contamos con funciones de conversión.

`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: Formatos para números y su significado.

Símbolo	Significado
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 número o fecha d a cadena de caracteres, se utiliza normalmente para fechas 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.

Para las funciones `TO_CHAR` y `TO_DATE`, en el caso de fechas, indicamos el formato incluyendo los siguientes símbolos:

Formatos para fechas y su SIGNificado.

Símbolo	Significado
YY	Año en formato de dos cifras.
YYYY	Año en formato de cuatro cifras.
MM	Mes en formato de dos cifras.
MON	Las tres primeras letras del mes.
MONTH	Nombre completo del mes.
DY	Día de la semana en tres letras.
DAY	Día completo de la semana.
DD	Día en formato de dos cifras.
D	Día de la semana (del 1 al 7).
Q	Trimestre del año.
WW	Semana del año.
AM / PM	Indicador a.m. o p.m.
HH12 HH24	Hora en formato de 1 a 12. Hora en formato de 0 a 23.
MI	Minutos (de 0 a 59).
SS SSSS	Segundos dentro del minuto. Segundos transcurridos desde las 00:00 horas.

3.1.4 FUNCIONES DE MANEJO DE FECHAS

En los SGBD los **campos de tipo fecha** son muy comunes. Oracle tiene dos tipos de datos para manejar fechas, son DATE y TIMESTAMP.

`DATE` almacena fechas concretas incluyendo a veces la hora, mientras que **`TIMESTAMP`** almacena un instante de tiempo más concreto que puede incluir hasta fracciones de segundo.

Algo **importante** a tener en cuenta cuando trabajamos con fechas en Oracle es **no depender del formato por defecto**, ya que esto es una de las **principales fuentes de errores** en el código SQL de Oracle. A esto se le llama **conversión implícita**. El **formato de la fecha** que maneja tu base SGBD **no es una configuración global fija**. Puede cambiar por:

- La configuración del servidor de la base de datos.
- La configuración del sistema operativo del cliente.
- Un comando `ALTER SESSION` ejecutado por un usuario.

Esto significa que una consulta que funciona perfectamente para ti puede **fallar para otro usuario** o en otro entorno. Para escribir código robusto y portable, **siempre debes usar `TO_DATE`** para convertir una cadena de texto a una fecha, asegurando que la conversión funcione siempre, sin importar la configuración de la sesión.

Podemos realizar **operaciones aritméticas (+, -)** con 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.

Por ejemplo:

```
SELECT SYSDATE – 5;
```

Devolvería la fecha correspondiente a 5 días antes de la fecha actual.

En Oracle tenemos las siguientes funciones más comunes:

`SYSDATE:` Devuelve la fecha y hora actuales.
Ejemplo: `SELECT SYSDATE FROM DUAL; --Resultado: 26/07/11`

`SYSTIMESTAMP:` Devuelve la fecha y hora actuales en formato TIMESTAMP.
Ejemplo: `SELECT SYSTIMESTAMP FROM DUAL; --Resultado: 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.
Ejemplo: `SELECT ADD_MONTHS(TO_DATE('27/07/11', 'DD/MM/YY'), 5) FROM DUAL; --Resultado: 27/12/11`

`MONTHS_BETWEEN(fecha1, fecha2):` Devuelve el número de meses que hay entre fecha1 y fecha2.
Ejemplo: `SELECT MONTHS_BETWEEN(TO_DATE('12/07/11', 'DD/MM/YY'),TO_DATE('12/03/11',  'DD/MM/YY')) FROM DUAL; --Resultado: 4`

`LAST_DAY(fecha):` Devuelve el último día del mes al que pertenece la fecha. El valor devuelto es tipo DATE.
Ejemplo: `SELECT LAST_DAY(TO_DATE('27/07/11', 'DD/MM/YY')) 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.
Ejemplo: `SELECT NEXT_DAY(TO_DATE('31/12/11', 'DD/MM/YY'),'MONDAY') 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.
Ejemplo: `SELECT EXTRACT(MONTH FROM SYSDATE) FROM DUAL; --Resultado: 7`

Se pueden emplear estas funciones enviando como argumento el nombre de un campo de tipo fecha.

EJERCICIOS DE FUNCIONES FECHA

1) Obtener la fecha del sistema

SELECT SYSDATE FROM DUAL;

2) Dada la tabla EMPLEADO, mostrar la fecha de alta (FECINEMP), sumando y restando dos meses. Mostrar las tres fechas con una cabecera especificada por tí.

```
SELECT  FECINEMP "FECHA", ADD_MONTHS(FECINEMP,2)"FECHA+2",
ADD_MONTHS(FECINEMP,-2) "FECHA-2" FROM EMPLEADO;
```

3) Obtener en la tabla EMPLEADO el último día del mes para cada mes de las fechas de alta.

```
SELECT FECINEMP, LAST_DAY(FECINEMP) FROM EMPLEADO;
```

4) Obtener nuestra edad utilizando diferentes funciones de fecha.

```
SELECT TRUNC(MONTHS_BETWEEN(SYSDATE, TO_DATE('21/06/1987',
'DD/MM/YYYY'))/12) FROM DUAL;
```

5) Partiendo de la fecha del sistema ¿En qué fecha caerá el próximo martes?

```
SELECT NEXT_DAY(SYSDATE,'TUESDAY') FROM DUAL;
```

CAMBIO DEL FORMATO DE FECHA EN EL SISTEMA

Por defecto el formato de fecha viene definido por el parámetro NLS_TERRITORY que especifica el idioma para fecha, punto decimal, separador de miles y símbolo de moneda.

En España `NLS_TERRITORY = SPAIN`

El valor por omisión para la fecha lo determina NLS_DATE_FORMAT.

Usando la orden `ALTER SESSION` esto se puede modificar:
Por ejemplo cambiar el formato de la fecha a día/nombre del mes/año hora:minutos:segundos

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD/MONTH/YYYY HH24:MI:SS';
```

También podemos cambiar el lenguaje usado para nombres de días y meses

```
ALTER SESSION SET NLS_DATE_LANGUAGE='ENGLISH';  -- OTRO - FRENCH
```

No obstante, como hemos comentado la mejor práctica es utilizar la función TO_DATE siempre que trabajemos con fechas para evitar problemas por diferentes configuraciones.

Ejercicio:

Ejecuta la siguiente instrucción:

```
ALTER SESSION SET NLS_DATE_LANGUAGE='SPANISH';
```

y ahora vuelve a ejecutar el ejemplo anterior:

```
SELECT NEXT_DAY(SYSDATE,'TUESDAY') FROM DUAL;
```

¿Qué ha ocurrido? ¿Cómo lo solucionamos?

3.1.5 OTRAS FUNCIONES: NVL Y DECODE

Cualquier columna opcional de una tabla puede contener un **valor nulo** independientemente al tipo de datos que tenga definido. Un valor **NULL** no será posible nunca en los casos en que definimos esa columna como no nula (**NOT NULL**), o que sea clave primaria (**PRIMARY KEY**).

Cualquier operación que se haga **con un valor NULL devuelve un NULL**. Por ejemplo, si se intenta dividir por NULL, no nos aparecerá ningún error sino que como resultado obtendremos un NULL (no se producirá ningún error tal y como puede suceder si intentáramos dividir por cero).

También es posible que el resultado de una función nos de un valor nulo.

Por tanto, es habitual encontrarnos con estos valores y es entonces cuando aparece la necesidad de poder hacer algo con ellos. Las funciones con nulos nos permitirán hacer algo en caso de que aparezca un valor nulo.

- **`NVL(valor, expr1):`** Si valor es NULL, entonces devuelve expr1. Ten en cuenta que expr1 debe ser del mismo tipo que valor.

- **`DECODE(expr1, cond1, valor1 [, cond2, valor2, ...], default ):`** Esta función evalúa una expresión expr1, si se cumple la primera condición (cond1) devuelve el valor1, en caso contrario evalúa la siguiente condición y así hasta que una de las condiciones se cumpla. Si no se cumple ninguna condición se devuelve el valor por defecto que hemos llamado default. Si no establecemos un valor por defecto y tenemos una valor en la tabla que no cumple con ninguna condición, se devolverá un NULL.

Si en la tabla EMPLEADO queremos un listado de su salario total (salario más comisiones) obtendremos NULL en el caso de que el trabajador no tenga comisión. Para evitarlo podemos escribir:

```
SELECT NOMEMP, SALEMP, COMEMP, SALEMP + NVL(COMEMP, 0) AS "SALARIO
TOTAL"
FROM EMPLEADO;
```

En la tabla HABEMP queremos asociar un literal a los distintos niveles de habilidad que muestran los empleados:

```
SELECT CODHAB, DECODE(NIVHAB, 10, 'MASTER', 9 ,'EXPERTO', 8,
'AVANZADO', 'MEDIO') NIVEL FROM HABEMP;

SELECT CODHAB, DECODE(NIVHAB, 10, 'MASTER', 9 ,'EXPERTO', 8,
'AVANZADO') NIVEL FROM HABEMP; -- Devuelve NULL a los valores
distintos de 8, 9 y 10
```

3.1.6. FUNCIONES ANIDADAS

Las funciones a nivel fila pueden anidarse hasta cualquier nivel y son evaluadas desde el nivel más profundo al nivel menos profundo.

Sintaxis:

```
... F3 (F2 (F1 (column, arg1), arg2 ), arg3) ...
```

Ejemplo:

Visualizar la fecha del próximo viernes que se lleva 6 meses con la fecha de contratación. La fecha resultante debería tener el formato “viernes , 13 noviembre , 1993”, ordenando los resultados por fecha de contratación, también se creará un alias en el que figure “REVISIÓN PRÓXIMOS 6 MESES”

```
SELECT NOMEMP, FECINEMP,
TO_CHAR (NEXT_DAY (ADD_MONTHS (FECINEMP, 6), 'FRIDAY'), 'day, dd
month, yyyy') "REVISIÓN PRÓXIMOS 6 MESES"
FROM EMPLEADO
ORDER BY 2;
```

4. CONSULTAS DE RESUMEN

La sentencia **`SELECT`** nos va a permitir **obtener resúmenes de los datos de modo vertical**. Para ello consta de una serie de cláusulas específicas (**`GROUP BY`**, **`HAVING`**) y tenemos también unas **funciones** llamadas **de agrupamiento o de agregado** que son las que nos dirán qué cálculos queremos realizar sobre los datos (sobre la columna).

Hasta ahora las consultas que hemos visto daban como resultado un subconjunto de filas de la tabla de la que extraíamos la información. Sin embargo, este tipo de consultas que vamos a ver no corresponde con ningún valor de la tabla sino un **total calculado** sobre los datos de la tabla. Esto hará que las consultas de resumen tengan limitaciones que iremos viendo.

Las funciones que podemos utilizar se llaman de agrupamiento (de agregado). Éstas **toman un grupo de datos** (una columna) y **producen un único dato que resume el grupo**. Por ejemplo, la función **`SUM()`** acepta una columna de datos numéricos y devuelve la suma de estos.

> 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. Es el valor 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.

4.1. FUNCIONES DE AGREGADO: SUM Y COUNT

Sumar y contar filas o datos contenidos en los campos es algo bastante común. Imagina que para nuestra tabla EMPLEADOS necesitamos sumar las comisiones totales que cobran nuestros trabajadores.

- La función `SUM`:
  - `SUM([ALL|DISTINCT] expresión)`
  - Devuelve la suma de los valores de la expresión.
  - Sólo puede utilizarse con columnas cuyo tipo de dato sea número. El resultado será del mismo tipo aunque puede tener una precisión mayor.
  - Por ejemplo,
    - `SELECT SUM(COMEMP) FROM EMPLEADO;`
- La función `COUNT`:
  - `COUNT([ALL|DISTINCT] expresión)`
  - Cuenta los elementos de un campo. `Expresión` contiene el nombre del campo que deseamos 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** (*).
  - Si utilizamos `COUNT(*)`, calcularemos el total de filas, incluyendo aquellas que contienen valores `NULL`.
  - Por ejemplo,
    - `SELECT COUNT(COMEMP) FROM EMPLEADO;`
    - `SELECT COUNT(*) FROM EMPLEADO;`

Evidentemente estas funciones pueden combinarse con cualquier cláusula WHERE. Por ejemplo, obtener el número de empleados que tiene un hijo:

```
SELECT COUNT(*) FROM EMPLEADO WHERE NUMHI = 1;
```

4.2 FUNCIONES DE AGREGADO: MIN Y MAX

Las siguientes funciones permiten encontrar el valor máximo y mínimo de una columna o expresión:

- **Función** `MIN`**:**
  - `MIN ([ALL| DISTINCT] expresión)`

    - 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).
    - Un ejemplo sería:
        - `SELECT MIN(SALEMP) FROM EMPLEADO;`
- **Función** `MAX`**:**
    - `MAX ([ALL| DISTINCT] expresión)`
    - 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).
    - Un ejemplo,
        - `SELECT MAX(SALEMP) FROM EMPLEADO;`

De nuevo pueden combinarse con cualquier cláusula WHERE o anidarse con cualquier otra función de las ya vistas.

4.3 FUNCIONES DE AGREGADO: AVG, VAR, STDEV Y STDEVP

Para cálculos estadísticos de los datos guardados en nuestra base de datos disponemos de funciones que calculan el promedio, la varianza y la desviación típica.

- **Función** `AVG`
    - `AVG ([ALL| DISTINCT] expresión)`
    - Devuelve el promedio de los valores de un grupo, para ello se omiten los valores 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.
- **Función** `VAR`
    - `VAR ([ALL| DISTINCT] expresión)`
    - Devuelve la varianza estadística de todos los valores de la expresión.
    - Como tipo de dato admite únicamente columnas numéricas. Los valores nulos (NULL) se omiten.
- **Función** `STDEV`
    - `STDEV ([ALL| DISTINCT] expresión)`
    - Devuelve la desviación típica estadística de todos los valores de la expresión.
    - Como tipo de dato admite únicamente columnas numéricas. Los valores nulos (NULL) se omiten.

Ejemplo: Utilizando la tabla empleado calcula el salario medio, mínimo y máximo:

```
SELECT AVG(SALEMP) AS "SALARIO MEDIO", MIN(SALEMP) AS "SALARIO
MÍNIMO", MAX(SALEMP) AS "SALARIO MÁXIMO" FROM EMPLEADO;
```

Si calculamos la comisión media, ¿A qué resultado se llega?. Como puedes observar no se tienen en cuenta valores NULL ni en numerador ni en denominador.

5. AGRUPAMIENTO DE REGISTROS

Hasta aquí las consultas de resumen que hemos visto obtienen totales de todas las filas de un campo o una expresión calculada sobre uno o varios campos. Lo que hemos obtenido ha sido una única fila con un único dato.

Ya verás como en muchas ocasiones en las que utilizamos consultas de resumen nos va a interesar calcular **totales parciales**, es decir, **agrupados según un determinado campo**.

De este modo podríamos obtener de una tabla EMPLEADOS, en la que se guarda su sueldo y su actividad dentro de la empresa, el valor medio del sueldo en función de la actividad realizada en la empresa. También podríamos tener una tabla clientes y obtener el número de veces que ha realizado un pedido, etc.

En todos estos casos en lugar de una única fila de resultados necesitaremos una fila por cada actividad, cada cliente, etc.

Podemos obtener estos **subtotales** utilizando la cláusula **`GROUP BY`**.

La sintaxis es la siguiente:

```
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 la cláusula **`HAVING`** se especifica la condición que han de cumplir los grupos para que se realice la consulta.

Es muy importante que te fijes bien en el orden en el que se ejecutan las cláusulas:

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`**.

Veamos un par de ejemplos:

Partiendo de la tabla DEPARTAMENTO calcular el presupuesto total anual de cada centro de trabajo:

```
SELECT CODCEN, SUM(PREANU) FROM DEPARTAMENTO GROUP BY CODCEN;
```

Calcular el salario medio por departamento (con dos cifras decimales) para los empleados que tienen 1 hijo ordenado por departamento. Filtrar para que solamente aparezcan los departamentos cuyo código incluye una letra ‘E’:

```
SELECT CODDEP, ROUND(AVG(SALEMP), 2)
FROM EMPLEADO
WHERE NUMHI = 1
GROUP BY CODDEP
HAVING CODDEP LIKE '%E%'
ORDER BY CODDEP ASC;
```

6. CONSULTAS MULTITABLAS

Recuerda que una de las propiedades de las bases de datos relacionales era que distribuíamos la información en varias tablas que a su vez estaban relacionadas por algún campo común (claves ajenas o foráneas). Así evitábamos repetir datos. Por tanto, también será frecuente que tengamos que consultar datos que se encuentren distribuidos por distintas tablas.

Disponemos de una tabla EMPLEADO cuya clave principal es CODEMP, que está a su vez está relacionada con la tabla DEPARTAMENTO a través del campo **CODDEP**. Si quisiéramos obtener el nombre de los empleados y el nombre del departamento en el que trabajan necesitaríamos coger datos de ambas tablas.

Imagina también que en lugar de tener una tabla EMPLEADO, dispusiéramos de dos por tenerlas en servidores distintos. Lo lógico es que en algún momento tendríamos que unirlas.

Hasta ahora las consultas que hemos usado se referían a una sola tabla, pero también es posible hacer consultas usando varias tablas en la misma sentencia **`SELECT`**. Esto permitirá realizar distintas operaciones como son:

- La composición interna.
- La composición externa.

En la versión SQL de 1999 se especifica una nueva sintaxis para consultar varias tablas que Oracle incorpora, así que también la veremos. La razón de esta nueva sintaxis era separar las condiciones de asociación respecto a las condiciones de selección de registros.

La sintaxis es la siguiente:

```
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 | RIGHT | FULL OUTER JOIN tabla2 ON

(tabla1.columna=tabla2.columna)]
```

6.1 PRODUCTO CARTESIANO

Si combinamos dos tablas sin aplicar ninguna restricción obtendremos un producto cartesiano de ambas, esto es, obtendremos todas las posibles combinaciones de filas entre esas dos tablas.

Esta operación no es muy común (de hecho en muchas ocasiones es producto de un error) ya que el número de resultados puede ser muy elevado en cuanto trabajemos con tablas medianamente grandes.

En SQL99 la cláusula para crear un producto cartesiano será `CROSS JOIN`. Por ejemplo:

```
SELECT *
FROM EMPLEADO
CROSS JOIN DEPARTAMENTO;
```

Con esto obtendrás todas las combinaciones posibles sin ninguna condición.

Lo habitual será por tanto **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 en las composiciones internas 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 la cláusula 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.

Puedes combinar una tabla consigo misma pero debes poner de manera obligatoria un alias a uno de los nombres de la tabla que vas a repetir. Esta será la técnica para relaciones reflexivas.

6.2 COMPOSICIONES INTERNAS

Son aquellas en las que solo se conservan las filas que cumplen la condición de unión entre las tablas.
En otras palabras, solo aparecen en el resultado los registros que tienen correspondencia en ambas tablas.

6.2.1 JOIN IMPLÍCITO

El JOIN implícito fue establecido en SQL-89. Ocurre cuando a continuación de la cláusula FROM escribimos las tablas a relacionar separadas por una coma. En la cláusula WHERE establecemos la relación entre las columnas de emparejamiento en la forma **`NombreTabla1.Camporelacionado1 = NombreTabla2.Camporelacionado2`**. La sintaxis es la siguiente:

```
SELECT tabla1.columna1, tabla1.columna2, …, tabla2.columna1,
tabla2.columna2, …
FROM tabla1, tabla2
WHERE condición;
```

Veamos un ejemplo, si queremos obtener el nombre de los empleados junto al nombre de sus hijos, así como su fecha de nacimiento en formato DD-MM-AAAA, tendremos:

```
SELECT E.NOMEMP, H.NOMHI,
TO_CHAR(H.FECNAHI, 'DD-MM-YYYY') AS FechaNacimientoHijo
FROM EMPLEADO E, HIJO H
WHERE E.CODEMP = H.CODEMP;
```

Vamos a obtener una lista de todos los empleados junto al nombre de las habilidades o competencias que dominan:

```
SELECT E.NOMEMP, H.DESHAB
FROM EMPLEADO E, HABILIDAD H, HABEMP HE
```

```
WHERE E.CODEMP = HE.CODEMP AND
HE.CODHAB = H.CODHAB;
```

Vamos a obtener un listado de empleados (nombre y NIF) junto con el nombre del departamento en el que trabajan, el nombre de su centro de trabajo y el director de dicho centro, ordenados por código de empleado:

```
SELECT E1.NOMEMP AS "EMPLEADO", E1.NIFEMP, D.NOMDEP, C.NOMCEN,
E2.NOMEMP AS "DIRECTOR"
FROM EMPLEADO E1, EMPLEADO E2, CENTRO C, DEPARTAMENTO D
WHERE E1.CODDEP = D.CODDEP AND D.CODCEN = C.CODCEN
AND E2.CODEMP = C.CODEMPDIR
ORDER BY E1.CODEMP;
```

6.2.2 JOIN EXPLÍCITO

El operador JOIN nació con la versión SQL 1999 con el objetivo de separar las condiciones de selección de registros que se utilizaba en la cláusula WHERE de las que se emplean para las condiciones de relación.

Sintaxis:

```
SELECT tabla1.columna1, tabla1.columna2, …, tabla2.columna1,
tabla2.columna2, …
FROM tabla1
INNER JOIN tabla2
ON condición;
```

Los ejemplos anteriores escritos con esta sintaxis quedarían de esta forma:

```
SELECT E.NOMEMP, H.NOMHI,
TO_CHAR(H.FECNAHI, 'DD-MM-YYYY')  AS FechaNacimientoHijo
FROM EMPLEADO E
INNER JOIN HIJO H
ON E.CODEMP = H.CODEMP;

SELECT
    E.NOMEMP,
    H.DESHAB
FROM
    EMPLEADO E
    INNER JOIN HABEMP HE ON E.CODEMP = HE.CODEMP
    INNER JOIN HABILIDAD H ON HE.CODHAB = H.CODHAB;
```

```
SELECT
    E1.NOMEMP AS "EMPLEADO",
    E1.NIFEMP,
    D.NOMDEP,
    C.NOMCEN,
    E2.NOMEMP AS "DIRECTOR"
FROM
    EMPLEADO E1
    INNER JOIN DEPARTAMENTO D ON E1.CODDEP = D.CODDEP
    INNER JOIN CENTRO C ON D.CODCEN = C.CODCEN
    INNER JOIN EMPLEADO E2 ON C.CODEMPDIR = E2.CODEMP
ORDER BY
    E1.CODEMP;
```

Cuando no se especifica el tipo de *join*, la palabra clave `JOIN` **por defecto significa** `INNER JOIN`.

Por tanto:

`JOIN` = `INNER JOIN`

en cualquier sistema de gestión de bases de datos relacionales (MySQL, PostgreSQL, SQL Server, Oracle, etc.).

Otra forma de JOIN interno explícito es el **`NATURAL JOIN.`** En él se detectan automáticamente las claves de unión, basándose en el nombre de la columna que coincide en ambas tablas. Por supuesto, se requerirá que las columnas de unión tengan el mismo nombre en cada tabla. Además, esta característica funcionará incluso si no están definidas las claves primarias o ajenas.

```
SELECT
    E.NOMEMP,
    H.DESHAB
FROM
    EMPLEADO E
    NATURAL JOIN HABEMP HE
    NATURAL JOIN HABILIDAD H;
```

Utiliza `NATURAL JOIN` con prudencia. Si en las tablas hay campos con el mismo nombre por cualquier motivo que no forman parte de los campos de combinación, `NATURAL JOIN` los incluirá en la condición de unión, lo que casi con total seguridad producirá resultados no deseados.

6.3 COMPOSICIONES EXTERNAS

Hay ocasiones en las que tendremos que obtener algunas filas de una tabla que no tienen correspondencia con ninguna fila de la otra. Imagina, por ejemplo, que tenemos en nuestra base de datos departamentos que en este momento no tienen ningún empleado. Si tuviéramos que obtener un informe con los datos de los empleados por departamento, seguro que deben aparecer esos departamentos aunque no tengan empleados. Para poder hacer esta combinación usaremos las composiciones externas.

Para realizar este tipo de consultas utilizaremos LEFT OUTER JOIN o RIGHT OUTER JOIN en función de la tabla en la que se generarán los valores nulos. Por ejemplo, si queremos sacar una lista de todos los empleados junto con las habilidades que dominan:

```
SELECT E.NOMEMP, H.DESHAB
FROM EMPLEADO E
LEFT OUTER JOIN HABEMP EH ON E.CODEMP = EH.CODEMP
LEFT JOIN HABILIDAD H ON EH.CODHAB = H.CODHAB
ORDER BY E.NOMEMP;
```

A-Z NOMEMP	A-Z DESHAB
Forzado López, Galeote	Fontanería
Forzado López, Galeote	Mecanografía
García Comisión, Elena	Telefonista
García Comisión, Elena	Técnicas de Venta
Gil Cadena, Francisco	[NULL]
Gozque Altanero, Cándic	[NULL]
Gómez Redes, Sara	Gestión Contable
Gómez Redes, Sara	Marketing
Hernández Nómina, Ped	Gerencia

Podemos ver que obtenemos los empleados sin ninguna habilidad especial, que de otro modo no aparecerían. Un RIGHT OUTER JOIN es una imagen especular de un LEFT OUTER JOIN, asegurando que todos los registros de la tabla derecha se incluyan en el resultado.

Por ejemplo, queremos mostrar **todas** las habilidades y, si algún empleado la tiene, su nombre:

```
SELECT H.DESHAB,E.NOMEMP
FROM
  EMPLEADO E
  INNER JOIN HABEMP EH ON E.CODEMP = EH.CODEMP
  RIGHT OUTER JOIN HABILIDAD H ON EH.CODHAB = H.CODHAB
ORDER BY H.DESHAB;
```

Un `FULL OUTER JOIN` es la combinación de un `LEFT JOIN` y un `RIGHT JOIN`. Asegura que **todos los registros de *ambas* tablas** se incluyan en el resultado.

- Si una fila de la tabla A no tiene match en la B, se incluye (con `NULL` para las columnas de B).
- Si una fila de la tabla B no tiene match en la A, también se incluye (con `NULL` para las columnas de A).

Mostrar una lista completa de **todos los empleados** y **todas las habilidades**, relacionándolos si es posible.

```
SELECT E.NOMEMP, H.DESHAB
FROM EMPLEADO E
FULL OUTER JOIN HABEMP HE ON E.CODEMP = HE.CODEMP
FULL OUTER JOIN HABILIDAD H ON HE.CODHAB = H.CODHAB
ORDER BY E.NOMEMP, H.DESHAB;
```

De `nuevo` cuando no se especifica el tipo de *join*, la palabra clave `LEFT|JOIN|FULL` **por defecto significa** `LEFT|JOIN|FULL OUTER JOIN`.

Por tanto:

`LEFT JOIN` = LEFT OUTER `JOIN`
`RIGHT JOIN` = RIGHT OUTER `JOIN`
`FULL JOIN` = FULL OUTER `JOIN`

7. SUBCONSULTAS

Una subconsulta es una consulta que se realiza previamente dentro de otra consulta, esta se ejecuta una vez y antes de que se ejecute la consulta principal, que utilizará el resultado de la subconsulta.

Las subconsultas pueden ser incluidas dentro de la cláusula WHERE, HAVING o FROM.

Sintaxis:

```
SELECT lista_columnas
FROM tabla
WHERE condición operador (SELECT lista_columnas
FROM tabla);
```

Ejemplo:

Mostrar el nombre de los empleados cuyo salario es superior al del empleado con código 6:

```
SELECT NOMEMP
FROM EMPLEADO
WHERE SALEMP >
(SELECT SALEMP
FROM EMPLEADO
WHERE CODEMP = 6);
```

El **OPERADOR** puede ser **>**, **<**, **>=**, **<=**, **!=**, **=** o **IN**.

Como puedes ver en la sintaxis, las subconsultas **deben ir entre paréntesis y a la derecha** del operador.

7.1 SUBCONSULTAS MONO REGISTRO

La sentencia SELECT anidada devuelve un **único registro.**

Se utilizan con operadores de **comparación**:

=	Igual a
>	Mayor que
>=	Mayor que o igual a
<	Menor que
<=	Menor que o igual a
<>	No igual a (o distinto de)

Ejemplo 1:

Visualizar el nombre de los empleados y el departamento para aquellos trabajadores que tienen el mismo departamento que el empleado 8.

```
SELECT NOMEMP, CODDEP
FROM EMPLEADO
WHERE CODDEP = (SELECT CODDEP
FROM EMPLEADO
WHERE CODEMP = 8);
```

Ejemplo 2:

Visualizar los datos de aquellos trabajadores que tienen el mismo departamento que el empleado 8 cuyo salario sea superior al del empleado 22.

```
SELECT *
FROM EMPLEADO
WHERE CODDEP = (SELECT CODDEP
FROM EMPLEADO
WHERE CODEMP = 8) AND
SALEMP > (
SELECT SALEMP
FROM EMPLEADO
WHERE CODEMP = 22);
```

7.1.1 USO DE FUNCIONES DE GRUPO EN SUBCONSULTAS

Se pueden visualizar los datos de la consulta principal por medio de una función de grupo en una subconsulta, siempre que se devuelva un único registro.

Ejemplo:

Visualizar el nombre, el departamento y el salario de todos los empleados cuyo salario es igual al mínimo salario de la empresa.

```
SELECT NOMEMP, CODDEP, SALEMP
FROM EMPLEADO
WHERE SALEMP =  (SELECT MIN(SALEMP) FROM EMPLEADO);
```

La cláusula HAVING en las subconsultas

Si una subconsulta va ligada a una función de grupo, se puede incluir en la cláusula HAVING, siempre que la condición de la cláusula HAVING sea una función de grupo.

Ejemplo:

Visualizar todos los departamentos que tienen el salario mínimo mayor que el mínimo salario del departamento VENZS.

```
SELECT CODDEP
FROM EMPLEADO
GROUP BY CODDEP
HAVING MIN(SALEMP) >
(SELECT MIN(SALEMP)
FROM EMPLEADO
WHERE CODDEP = 'VENZS');
```

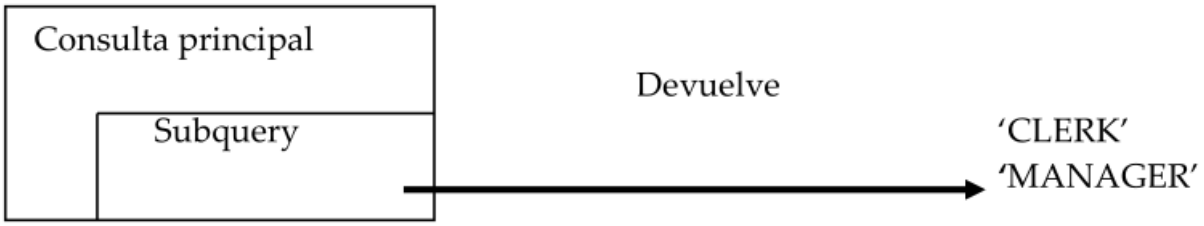
Errores en las subconsultas: Se produce un cuando la subquery devuelve más de un valor (registro). En este caso sustituiremos el operador = por IN convirtiéndose por tanto una consulta multi-registro. En general este error se produce cuando utilizamos un operador mono-registro en vez de uno multi-registro.

Ejemplo:

```
SELECT NOMEMP
FROM EMPLEADO
WHERE SALEMP =
(SELECT MIN(SALEMP) FROM EMPLEADO
GROUP BY CODDEP);
```

7.2 SUBCONSULTAS MULTIREGISTRO

Son aquellas en las que la sentencia SELECT anidada devuelve más de un registro.



Los operadores válidos en este caso son:

- **`ANY|SOME.`** 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, lo que hace es comprobar si el valor se encuentra en el resultado de la subconsulta.
- **`NOT IN|ANY|ALL.`** Niega la condición.
- **`EXISTS.`** Se ejecuta la consulta si la subconsulta devuelve algún valor.

`ANY` y `ALL` pueden ir acompañados de los operadores de comparación: =,>,<. <>,<= y >=

Ejemplo 1:

Averiguar qué empleados ganan un salario igual a cualquiera de los mínimos por departamento.

```
SELECT NOMEMP, CODDEP, SALEMP
FROM EMPLEADO
```

```
WHERE SALEMP IN
(SELECT MIN (SALEMP)
FROM EMPLEADO
GROUP BY CODDEP);
```

Ejemplo 2:

Muestra los empleados que NO pertenecen al departamento 'PROZS', pero que ganan más dinero que AL MENOS UNO de los empleados que sí están en 'PROZS'.

```
SELECT NOMEMP, CODDEP, SALEMP, CODEMP
FROM EMPLEADO
WHERE SALEMP > ANY
(SELECT SALEMP
FROM EMPLEADO
WHERE CODDEP = 'PROZS')
AND CODDEP <> 'PROZS';
```

Esta consulta sería equivalente a:

```
SELECT NOMEMP, CODDEP, SALEMP, CODEMP
FROM EMPLEADO
WHERE SALEMP >
(SELECT MIN(SALEMP)
FROM EMPLEADO
WHERE CODDEP = 'PROZS')
AND CODDEP <> 'PROZS';
```

En general:

Operador ANY/ALL	Equivalente con Función de Grupo	Explicación
> ANY	**> MIN**	Mayor que el mínimo (con ganar a uno me vale).
< ANY	**< MAX**	Menor que el máximo (con perder contra uno me vale).
> ALL	**> MAX**	Mayor que TODOS (tengo que ganar al mejor).
< ALL	**< MIN**	Menor que TODOS (tengo que perder contra el peor).

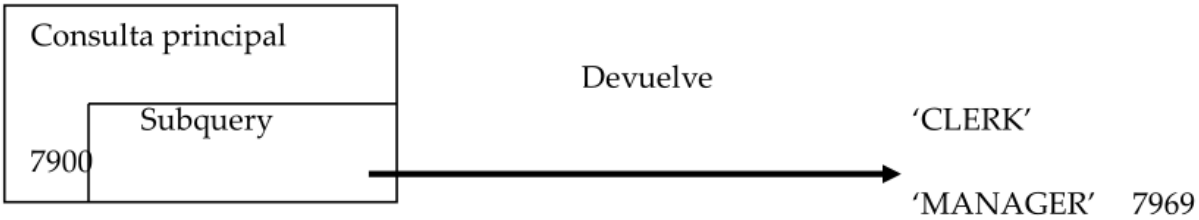
Ejemplo 3:

Visualizar aquellos empleados cuyo salario sea superior a la media de los salarios de todos los departamentos.

```
SELECT NOMEMP, SALEMP
FROM EMPLEADO
WHERE SALEMP > ALL
(SELECT AVG (SALEMP)
FROM EMPLEADO
GROUP BY CODDEP);
```

7.3 SUBCONSULTAS MULTICOLUMNA

En ellas la sentencia SELECT anidada devuelve más de una columna. Hasta ahora se han creado subconsultas mono_registro y multi_registro, en las que sólo una columna era comparada en la cláusula WHERE o cláusula HAVING de la sentencia SELECT. Si se quiere comparar dos o más columnas habrá que escribir en la cláusula WHERE compuesta, la cual utiliza operadores lógicos. Las subconsultas multi_columna, permitirán transformar diferentes condiciones WHERE en una única cláusula WHERE.



Sintaxis:

```
SELECT column1, column2,...
FROM tabla
WHERE (column1, column2 ... )
IN ( SELECT column1, column2,...
FROM tabla
WHERE condición);
```

Ejemplo 1:

Queremos encontrar al empleado que gana el salario máximo de cada departamento. En la consulta sacamos el nombre del empleado, el nombre del departamento al que pertenece y su salario, ordenado de forma ascendente por nombre de departamento.

```
SELECT NOMEMP, NOMDEP, SALEMP
FROM EMPLEADO
JOIN DEPARTAMENTO ON EMPLEADO.CODDEP = DEPARTAMENTO.CODDEP
WHERE (EMPLEADO.CODDEP, SALEMP) IN (
    SELECT CODDEP, MAX(SALEMP)
    FROM EMPLEADO
    GROUP BY CODDEP
)
ORDER BY NOMDEP;
```

Ejemplo 2:

Obtener empleados que tengan **la misma habilidad y el mismo nivel** que alguna de las que tiene el empleado 12, excluyendo al propio empleado 12.

```
SELECT E.NOMEMP, H.DESHAB, HE.NIVHAB
FROM EMPLEADO E
JOIN HABEMP HE ON E.CODEMP = HE.CODEMP
JOIN HABILIDAD H ON HE.CODHAB = H.CODHAB
WHERE (HE.CODHAB, HE.NIVHAB) IN (
    SELECT CODHAB, NIVHAB
    FROM HABEMP
    WHERE CODEMP = 12
)
AND E.CODEMP != 12;
```

Ejemplo 3:

Buscar empleados que ganen exactamente lo mismo (tanto en Salario Base como en Comisión) que el empleado 7.

```
SELECT NOMEMP, SALEMP, COMEMP
FROM EMPLEADO
WHERE (SALEMP, NVL(COMEMP, 0)) IN (
    SELECT SALEMP, NVL(COMEMP, 0)
    FROM EMPLEADO
    WHERE CODEMP = 7
)
AND CODEMP != 7;
```

7.3.1 COMPARACIÓN ENTRE COLUMNAS

- **PAIRWISE:** consiste en comparar simultáneamente en la WHERE las columnas que están antes del operador con las que están en las subconsulta.

Ejemplo:

Busca empleados cuyo **conjunto exacto** de (Salario, Hijos) exista en el departamento PROZS. Trata ambas columnas como una unidad indivisible.

```
SELECT CODEMP, NOMEMP, SALEMP, NUMHI, CODDEP
FROM EMPLEADO
WHERE (SALEMP, NUMHI) IN (
    SELECT SALEMP, NUMHI
    FROM EMPLEADO
    WHERE CODDEP = 'PROZS'
);
```

- **NON-PAIRWISE:** compara una columna que está antes del operador con otra que está dentro de la SUBQUERY y así sucesivamente.

Ejemplo:

Esta consulta busca empleados que tengan un salario que exista en PROZS **Y** un número de hijos que exista en PROZS, pero **no necesariamente provenientes de la misma persona**.

```
SELECT CODEMP, NOMEMP, SALEMP, NUMHI, CODDEP
FROM EMPLEADO
WHERE SALEMP IN (
    SELECT SALEMP
    FROM EMPLEADO
    WHERE CODDEP = 'PROZS'
)
AND NUMHI IN (
    SELECT NUMHI
    FROM EMPLEADO
    WHERE CODDEP = 'PROZS'
);
```

7.4 SUBCONSULTAS EN LA CLÁUSULA FROM

Una subconsulta en la cláusula `FROM` actúa como si fuera una **tabla temporal** creada al vuelo. Oracle ejecuta primero la subconsulta, guarda el resultado en memoria (o disco si es muy grande) y la consulta principal trata ese resultado como una tabla más a la que podemos hacer `JOIN`, poner alias, etc.

Sintaxis:

```
SELECT ...
FROM Tabla A
JOIN (SELECT ... FROM Tabla B) Alias_Temporal
ON A.col = Alias_Temporal.col;
```

Ejemplo 1: Datos agregados junto a datos de detalle

Queremos listar el nombre del empleado, su salario, y **el salario medio de su departamento**. *Nota: Esto requiere mezclar un dato de detalle (empleado) con un dato agrupado (media del departamento). Una subconsulta en el FROM es ideal para esto.*

```
SELECT E.NOMEMP, E.SALEMP, DATOS_DEP.MEDIA_SALARIO
FROM EMPLEADO E
JOIN (
    -- Esta subconsulta crea una "tabla" con el código de dpto y su
media
    SELECT CODDEP, ROUND(AVG(SALEMP), 2) as MEDIA_SALARIO
    FROM EMPLEADO
    GROUP BY CODDEP
) DATOS_DEP ON E.CODDEP = DATOS_DEP.CODDEP
WHERE E.SALEMP > DATOS_DEP.MEDIA_SALARIO; -- (Opcional: ver solo los
que cobran más que la media)
```

Ejemplo 2: El ranking top-N

Aunque en Oracle moderno (12c+) existe `FETCH FIRST`, entender cómo funciona esto es vital. Queremos los 3 empleados que más ganan. No podemos usar `ROWNUM` directamente sobre la tabla sin ordenar, porque `ROWNUM` se asigna antes de ordenar.

```
SELECT * FROM (
    -- Primero ordenamos en la "tabla temporal"
    SELECT NOMEMP, SALEMP
    FROM EMPLEADO
    ORDER BY SALEMP DESC
```

```
)
WHERE ROWNUM <= 3;
```

7.5 SUBCONSULTAS EN CLÁUSULA SELECT

Las subconsultas en cláusulas SELECT, o subconsultas escalares, se usan para obtener un valor calculado por cada fila de la consulta principal. **Regla de oro:** La subconsulta debe devolver **exactamente 1 fila y 1 columna**. Si devuelve más, fallará.

Ejemplo:

Listado de Departamentos y el número de empleados que tiene cada uno (sin hacer un `GROUP BY` en la consulta principal).

```
SELECT D.NOMDEP,
       (SELECT COUNT(*)
        FROM EMPLEADO E
        WHERE E.CODDEP = D.CODDEP) AS NUM_EMPLEADOS
FROM DEPARTAMENTO D;
```

8. UNION, INTERSECT Y MINUS

La sintaxis es la siguiente:

```
[Instrucción SQL 1]
UNION [ALL] | INTERSECT | MINUS
[Instrucción SQL 2];
```

• **`UNION:`** El propósito del comando SQL UNION es combinar los resultados de dos consultas juntas.

Ambas consultas deben tener el **mismo número** de columnas, y las columnas correspondientes (la 1ª con la 1ª, la 2ª con la 2ª, etc.) deben tener **tipos de datos compatibles**.

También tened en cuenta que cuando utilizamos UNION, sólo se seleccionan valores distintos (similar a SELECT DISTINCT). Si se quieren unir dos SELECT independientes y con estructura homogénea se utilizan las cláusulas UNION y UNION ALL. Si especificamos la cláusula UNION no aparecerán los registros que estén repetidos, mientras que, con la cláusula UNION ALL, si aparecerán los registros repetidos.

Ejemplo: Obtener una lista única de todos los empleados que están en el departamento de 'Ventas Zona Levante' **O** en el departamento de 'Ventas Zona Sur'.

```
-- Primer conjunto: Empleados de Ventas Zona Levante
```

```
SELECT CODEMP, NOMEMP, CODDEP
FROM EMPLEADO
WHERE CODDEP = 'VENZL'

UNION
-- Segundo conjunto: Empleados de Ventas Zona Sur
SELECT CODEMP, NOMEMP, CODDEP
FROM EMPLEADO
WHERE CODDEP = 'VENZS';
```

Ejemplo UNION ALL: Crear un informe que liste a todos los empleados de 'Ventas Zona Sur' y, *además*, a todos los empleados (de cualquier departamento) que ganen más de 5.000.

```
-- Primer conjunto: Empleados de Ventas Zona Sur
SELECT CODEMP, NOMEMP, CODDEP
FROM EMPLEADO
WHERE CODDEP = 'VENZS'

UNION ALL

-- Segundo conjunto: Empleados con salario mayor a 5000
SELECT CODEMP, NOMEMP, SALEMP
FROM SALEMP
WHERE SALARIO > 5.000;
```

• **INTERSECT:** Parecido al comando UNION, INTERSECT también opera en dos instrucciones SQL. La diferencia es que, mientras UNION actúa fundamentalmente como un operador OR (O) (el valor se selecciona si aparece en la primera o la segunda instrucción), el comando INTERSECT actúa como un operador AND (Y) (el valor se selecciona si aparece en ambas instrucciones).

Ejemplo: Encontrar a los empleados que son buenos gestores. Es decir, aquellos que están en el departamento de 'Ventas' **Y ADEMÁS** tienen la habilidad 'Gerencia'.

```
-- Primer conjunto: TODOS los empleados de Ventas
SELECT CODEMP, NOMEMP
FROM EMPLEADO
WHERE CODDEP IN ('VENZS', 'VENZL')

INTERSECT

-- Segundo conjunto: TODOS los empleados con habilidad 'Liderazgo'
```

```
SELECT E.CODEMP, E.NOMEMP
FROM EMPLEADO E
INNER JOIN HABEMP EH ON E.CODEMP = EH.CODEMP
INNER JOIN HABILIDAD H ON EH.CODHAB = H.CODHAB
WHERE H.DESHAB = 'Gerencia';
```

• **`MINUS`:** MINUS opera en dos instrucciones SQL. Toma todos los resultados de la primera instrucción SQL, y luego sustrae aquellos que se encuentran presentes en la segunda instrucción SQL para obtener una respuesta final. Si la segunda instrucción SQL incluye resultados que no están presentes en la primera instrucción SQL, dichos resultados se ignoran.

Ejemplo: Encontrar a los empleados de 'Ventas' que necesitan formación. Es decir, aquellos que están en 'Ventas' **PERO QUE NO** tienen la habilidad 'Negociación'.

```
-- Conjunto A: TODOS los empleados de Ventas
SELECT CODEMP, NOMEMP
FROM EMPLEADO
WHERE CODDEP IN ('VENZS', 'VENZL')

MINUS

-- Conjunto B (los que restamos): Empleados que SÍ tienen 'Negociación'
SELECT E.CODEMP, E.NOMEMP
FROM EMPLEADO E
INNER JOIN HABEMP EH ON E.CODEMP = EH.CODEMP
INNER JOIN HABILIDAD H ON EH.CODHAB = H.CODHAB
WHERE H.DESHAB = 'Negociación';
```

¿Cuándo es conveniente el uso de estos operadores de conjunto?

1. Cuando las fuentes de datos son diferentes (La razón principal). Por ejemplo, quieres una sola lista con los nombres y teléfonos de todos tus **Clientes** y todos tus **Proveedores**.
    - `JOIN` no sirve: No hay una relación directa para "unir" un cliente con un proveedor.
    - `WHERE` no sirve: No puedes poner `FROM Clientes, Proveedores` sin generar un producto cartesiano (un caos).
    - La única solución es `UNION`
2. Cuando la lógica de las consultas es muy compleja. A veces, las dos listas que quieres unir *vienen* de la misma tabla, pero los criterios para obtenerlas son tan complejos y diferentes que intentar unirlos con `AND` y `OR` sería una pesadilla.
3. Para comparar conjuntos.

9. CREACIÓN DE TABLAS A PARTIR DE UNA CONSULTA

Aunque corresponde a una instrucción DDL, implica conocer la sintaxis de una consulta, por lo que la incluimos aquí. Esta sentencia permite crear una tabla a partir de la consulta de otra tabla ya existente. La nueva tabla contiene los datos obtenidos de la consulta. Sintaxis:

```
CREATE TABLE nb_tabla [(
Col1 [, col2] ……….....)]
AS consulta;
```

Ejemplo 1:

Crear una tabla llamada EMPLEADO con los empleados del departamento de ventas (aquellos cuyo código empieza por VEN).

```
CREATE TABLE EMPLEADO_VENTAS
AS SELECT *
FROM EMPLEADO
WHERE CODDEP LIKE 'VEN%';
```

Ejemplo 2:

Crear una tabla EMPLEADO_DEPARTAMENTO a partir de las tablas EMPLEADO y DEPARTAMENTO. Esta tabla contendrá el nombre completo y el nombre del departamento de cada empleado.

```
CREATE TABLE EMPLEADO_DEPARTAMENTO
AS SELECT E.NOMEMP, D.NOMDEP
FROM EMPLEADO E
JOIN DEPARTAMENTO D ON E.CODDEP = D.CODDEP;
```

10. OTRAS INSTRUCCIONES ÚTILES

Aquí veremos otras instrucciones y funciones útiles a la hora de crear consultas:

10.1 COALESCE

Es una función estándar ANSI (funciona en Oracle, SQL Server, PostgreSQL, MySQL, etc.) que acepta una lista de *n* expresiones y **devuelve la primera que NO sea NULL**. Si todas son NULL, devuelve NULL.

Su sintaxis formal es:

```
COALESCE(expr_1, expr_2, ..., expr_n)
```

Ejemplo práctico con la base de datos DULCES

Imagina que en la tabla `PEDIDOS` queremos mostrar la fecha de envío. Si no se ha enviado (fecha_envio es `NULL`), queremos ver la fecha del pedido. Y si por algún error la fecha del pedido también fuera `NULL` (hipotéticamente), queremos mostrar la fecha actual (`SYSDATE`).

```
SELECT idpedido,
       COALESCE(fecha_envio, fecha_pedido, SYSDATE) as
fecha_efectiva
FROM PEDIDOS;
```

- **Paso 1:** Mira fecha_envio. ¿Tiene valor? Úsalo.
- **Paso 2:** Si es NULL, salta a fecha_pedido. ¿Tiene valor? Úsalo.
- **Paso 3:** Si ambos son NULL, usa SYSDATE.

Comparativa: COALESCE vs. NVL

Característica	NVL (Oracle Nativo)	COALESCE (Estándar ANSI)
Argumentos	Acepta exactamente **2** argumentos.	Acepta **2 o más** argumentos.
Portabilidad	Solo funciona en Oracle.	Funciona en cualquier BBDD moderna.
Tipos de datos	Intenta convertir implícitamente el 2º tipo al 1º.	Es más estricto: todos los argumentos deben ser convertibles al mismo tipo común.
Evaluación	**Evalúa ambos argumentos siempre.**	**Short-circuit (Cortocircuito)**. Se detiene al encontrar el primer valor.

10.2 CASE

`CASE` es una **expresión** (devuelve un valor), no una sentencia de control de flujo (como en PL/SQL). Por tanto, pueden usarse en el `SELECT`, en el `ORDER BY`, e incluso dentro de funciones de agregación. Permite condiciones complejas con operadores (`<`, `>`, `AND`, `OR`).

Sintaxis básica:

```
CASE
    WHEN condicion_1 THEN resultado_1
    WHEN condicion_2 THEN resultado_2
    ...
    ELSE resultado_por_defecto
END
```

Ejemplo práctico con BBDD "DULCES"

Vamos a usar la tabla **CAJAS**. Supongamos que queremos clasificar las cajas en tres categorías comerciales ("Económica", "Estándar", "Premium") basándonos en su precio, para un informe de marketing.

```
SELECT nombre,
       precio,
       CASE
           WHEN precio < 1500 THEN 'Gama Económica'
           WHEN precio BETWEEN 1500 AND 3000 THEN 'Gama Estándar'
           WHEN precio > 3000 THEN 'Gama Premium'
           ELSE 'Precio no catalogado' -- Buena práctica: siempre
cubrir el ELSE
       END AS categoria_comercial
FROM CAJAS
ORDER BY precio DESC;
```

Comparativa rápida: CASE vs DECODE

- **DECODE:** Es exclusivo de Oracle, más antiguo y sintácticamente más corto, pero solo permite comparaciones de igualdad (=).
- **CASE:** Es estándar ANSI (portable), más legible y permite rangos (<, >).

10.3 FETCH FIRST n ROWS ONLY

Sintaxis Básica (El "top N")

Imagina que quieres mostrar las **3 cajas más caras** del catálogo.

Sintaxis:

```
SQL
SELECT nombre, precio
FROM CAJAS
```

```
ORDER BY precio DESC
FETCH FIRST 3 ROWS ONLY;
```

- **Ventaja clave:** Oracle ordena *primero* y corta *después*. En este ejemplo ordena las cajas por precio y, una vez ordenadas se queda solo con las 3 primeras.

2. El problema de los empates (`WITH TIES`)

¿Qué pasa si la caja nº 3 y la nº 4 cuestan lo mismo?

- `ONLY`: Corta estrictamente en 3 filas (la nº 4 se queda fuera arbitrariamente).
- `WITH TIES`: Si hay empate en la última posición del corte, **trae también a los empatados**.

Ejemplo (Precios repetidos): Si buscamos cajas alrededor de 2775€ (tenemos `BITT` y `SWEE` con ese precio).

```
SELECT nombre, precio
FROM CAJAS
ORDER BY precio DESC
FETCH FIRST 5 ROWS WITH TIES;
```

Si la 5ª caja tiene el mismo precio que la 6ª, Oracle devolverá 6 filas (o más). Esto es vital para garantizar la equidad en informes de "los mejores clientes" o "notas más altas".

3. Paginación (OFFSET + FETCH)

Ideal para entender cómo funcionan las webs que muestran resultados "Página 1 de 10". Permite saltarse un número de filas antes de empezar a coger.

Ejemplo: "Muéstrame la **segunda** página de resultados, asumiendo que mostramos 4 cajas por página".

```
SELECT nombre, precio
FROM CAJAS
ORDER BY precio DESC
OFFSET 4 ROWS FETCH NEXT 4 ROWS ONLY;
```

- `OFFSET 4 ROWS`: Sáltate las 4 primeras (las más caras).
- `FETCH NEXT 4`: Coge las 4 siguientes.