PARTE I

SENTENCIA SELECT. FUNCIONES.

1. INTRODUCCIÓN.

SQL es un lenguaje con un conjunto de sentencias u órdenes que nos permiten trabajar con nuestras Bases de 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, cada fabricante añade sus mejoras al lenguaje SQL.

2. TIPOS DE SENTENCIAS SQL.

Tenemos diferentes tipos de sentencias SQL en función de las diferentes tareas que podemos realizar sobre una base de datos.

La primera fase del trabajo con cualquier base de datos comienza con sentencias para la definición de datos, DDL. Antes de poder almacenar y recuperar información debimos definir las estructuras **donde** agrupar la información:

CREATE TABLE	Añade una nueva tabla a la base de datos.
DROP TABLE	Suprime una tabla de la base de datos.
ALTER TABLE	Modifica la estructura de una tabla existente.
CREATE VIEW	Añade una nueva vista a la base de datos.
DROP VIEW	Suprime una vista de la base de datos.
CREATE INDEX	Construye un índice para una columna.
DROP INDEX	Suprime el índice para una columna.
CREATE SYNONYM	Define un alias para un nombre de tabla.

La siguiente fase es la manipulación de los datos DML. Este conjunto de sentencias está orientado a consultas y manejo de datos de los objetos creados:

SELECT	Recupera datos de la base de datos.
INSERT	Añade nuevas filas de datos a la base de datos.
DELETE	Suprime filas de datos de la base de datos.
UPDATE	Modifica datos existentes en la base de datos.

Y finalmente tenemos las sentencias para el control de los datos DCL, que nos permiten controlar el acceso de los usuarios a los datos y el control de transacciones:

Control de acceso

GRANT Concede privilegios de acceso a usuarios.

REVOKE Suprime privilegios de acceso a usuarios.

Control de transacciones

COMMIT Finaliza la transacción actual.

ROLLBACK Aborta la transacción actual.

3. EL LENGUAJE DE MANIPULACIÓN DE DATOS (DML)

3.1.LA SENTENCIA SELECT.

La sentencia SELECT nos va permitir recuperar o seleccionar los datos, de una o varias tablas.

De forma general podemos decir, que una instrucción SELECT consta de cuatro partes básicas:

- ✓ Cláusula **SELECT** seguida de la descripción de lo que se desea ver, es decir, de los nombres de las columnas que quieres que se muestren. Esta parte es obligatoria.
- ✓ Cláusula **FROM** seguida del nombre de las tablas de las que proceden las columnas de arriba, es decir, de donde se van a extraer los datos. Esta parte también es obligatoria.
- ✓ Cláusula **WHERE** seguida de un criterio de selección o condición. Esta parte no es obligatoria, pero aparecerá muchas veces en nuestras sentencias.
- ✓ Cláusula **ORDER BY** seguida por un criterio de ordenación. Esta parte tampoco es obligatoria.

Por tanto, una primera sintaxis quedaría de la siguiente forma:

SELECT [ALL | DISTINCT] columna1, columna2, ...
FROM tabla1, tabla2, ...
WHERE condición1, condición2, ...
ORDER BY ordenación;

¡¡ RECUERDA !! SQL no diferencia entre mayúsculas y minúsculas (no es case sensitive) y todas las instrucciones deben finalizar con punto y coma.

3.1.2. CLÁUSULA SELECT.

Ya has visto que a continuación de la sentencia SELECT debemos especificar cada una de las columnas que queremos seleccionar, es decir, los campos que queremos que aparezcan como resultado de nuestra consulta.

Además, debemos tener en cuenta lo siguiente:

- ✓ **Se pueden nombrar** a las columnas anteponiendo el nombre de la tabla de la que proceden, esto es opcional, pero puede ayudarte cuando haya campos que se llamen igual en diferentes tablas

NombreTabla.NombreColumna

- ✓ Si queremos **incluir todas las columnas** de una tabla podemos utilizar el comodín **asterisco** ("*").

SELECT * FROM NombreTabla;

- ✓ También podemos **ponerles alias a los nombres** de las columnas. Cuando se consulta una base de datos, los nombres de las columnas se usan como cabeceras de presentación. Si éste resulta largo, corto o poco descriptivo, podemos usar un alias. Para ello a continuación del nombre de la columna ponemos entre comillas dobles el alias que demos a esa columna.

SELECT Usuarios.F_Nacimiento "Fecha de Nacimiento" FROM USUARIOS;

- ✓ También podemos **sustituir el nombre** de las columnas por constantes, expresiones o funciones SQL.

SELECT 4*3/100 "MiExpresion", Password FROM USUARIOS;

Las cláusulas ALL y DISTINCT son opcionales.

- ✓ La cláusula **ALL** después de SELECT, indica que quieres seleccionar todas las filas estén o no repetidas. Es el valor por defecto y no se suele especificar.
- ✓ La cláusula **DISTINCT** después de SELECT, suprimirá aquellas filas del resultado que tengan igual valor que otras.

SELECT nombre FROM USUARIOS;

Aparecen todas las filas de la tabla EMPLEADOS donde la columna nombre no sea nula; también aparecen los nombres de los empleados que se llamen igual tantas veces como se repitan, para evitar las repeticiones utilizamos DISTINCT

SELECT DISTINCT nombre FROM USUARIOS;

3.1.1. CLÁUSULA FROM.

Al realizar la consulta o selección has visto que puedes elegir las columnas que necesites, pero no hemos indicado las tablas donde se encuentra la información.

En la sentencia SELECT debemos establecer de dónde se obtienen las columnas que vamos a seleccionar, para ello disponemos en la sintaxis de la cláusula FROM.

Por tanto, en la cláusula FROM se definen los nombres de las tablas de las que proceden las columnas. Si se utiliza más de una, éstas deben aparecer separadas por comas. A este tipo de consulta se denomina consulta combinada o join. Más adelante verás que para que la consulta combinada pueda realizarse, necesitaremos aplicar una condición de combinación a través de una cláusula WHERE.

También puedes asociar un alias a las tablas para abreviar, en este caso no es necesario que lo encierres entre comillas.

SELECT * FROM USUARIOS U;

Realizar ejercicios 1-12 UT5_Partei_basicas

3.1.2. CLÁUSULA WHERE.

Hasta ahora hemos podido ver todas o un subconjunto de columnas de una o varias tablas. Pero esta selección afectaba a todas las filas (registros) de la tabla. Si queremos restringir esta selección a un subconjunto de filas debemos especificar una condición que deben cumplir aquellos registros que queremos seleccionar. Para poder hacer esto vamos a utilizar la cláusula WHERE. A continuación de la palabra WHERE será donde pongamos la condición o condiciones que han de cumplir las filas que queremos como resultado de dicha consulta.

El criterio de búsqueda o condición puede ser más o menos complejo y para crearlo se pueden conjugar operadores de diversos tipos, funciones o expresiones.

Los operadores que podemos utilizar para construir las condiciones, pueden ser aritméticos, lógicos, relacionales…, los veremos en profundidad en el siguiente punto.

A modo de introducción, podemos hacer la siguiente clasificación de los operadores:

TIPOS	OPERADORES
Aritméticos	+ - * /
Lógicos	AND OR NOT
Relacionales	= > < >= <= != <>

3.1.4 CLÁUSULA ORDER BY.

En la consulta del ejemplo anterior hemos obtenido una lista de nombres y apellidos de las usuarias de nuestro juego. Es probable que necesitemos el resultado de la consulta ordenado por apellidos y nombre. Para ello utilizamos la cláusula ORDER BY.

ORDER BY se utiliza para especificar el criterio de ordenación de la respuesta de una consulta.

```
SELECT [ALL | DISTINCT] columna1, columna2, ...
FROM tabla1, tabla2, ...
WHERE condición1, condición2, ...
ORDER BY columna1 [ASC | DESC], columna2 [ASC | DESC], …, columnaN [ASC | DESC];
```

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.

Podemos ordenar la información resultante 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.

En el siguiente ejemplo, ordenamos por apellidos y en caso de empate por nombre:

```
SELECT nombre, apellidos
FROM USUARIOS
ORDER BY apellidos, nombre;
```

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, si queremos el resultado del ejemplo anterior ordenado por localidad:

```
SELECT nombre, apellidos, localidad
FROM usuarios
ORDER BY 3;
```

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.

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

4. OPERADORES.

Los operadores son símbolos que permiten realizar operaciones matemáticas, concatenar cadenas o hacer comparaciones. SQL tiene 4 tipos de operadores:

- ✓ Relacionales o de comparación.
- ✓ Aritméticos.
- ✓ De concatenación.
- ✓ Lógicos.

4.1. OPERADORES RELACIONALES.

Los operadores **relacionales**, nos **permitirán comparar expresiones**, que pueden ser valores concretos de campos, variables, … Si el resultado de la comparación es correcto la expresión considerada es verdadera, en caso contrario es falsa.

OPERADOR	SIGNIFICADO
=	Igualdad.
!=, < >, ^=	Desigualdad.
<	Menor que.
>	Mayor que.
<=	Menor o igual que.
>=	Mayor o igual que.
IN	Igual que cualquiera de los miembros entre paréntesis.
NOT IN	Distinto que cualquiera de los miembros entre paréntesis.
BETWEEN	Entre. Contenido dentro del rango.
NOT BETWEEN	Fuera del rango.
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.
IS NULL	Devuelve verdadero si el valor del campo de la fila que examina es nulo.

El valor NULL significa valor inexistente o desconocido y por tanto es tratado de forma distinta a otros valores. Si queremos verificar que un valor es NULL no serán válidos los operadores que acabamos de ver. Debemos utilizar los valores IS NULL como se indica en la tabla o IS NOT NULL que devolverá verdadero si el valor del campo de la fila no es nulo.

Además, cuando se utiliza un ORDER BY, los valores NULL se presentarán en primer lugar si se emplea el modo ascendente y al final si se usa el descendente.

Si queremos obtener aquellos empleados cuyo salario es superior a 1000€ podemos crear la siguiente consulta:

```
SELECT nombre
FROM EMPLEADOS
WHERE SALARIO > 1000;
```

Si por ejemplo queremos aquellos empleados cuyo apellido comienza por R:

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

Obtener el código y el nombre de las universidades de Sevilla o Cádiz. Resultado:

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

Fíjate que buscará aquellas ciudades que coincidan textualmente con las que ponemos entre comillas.

4.1.1. OPERADORES DE COMPARACIÓN DE CADENAS DE CARACTERES

Para comparar cadenas de caracteres, hasta ahora hemos utilizado el operador de comparación Igual a (=). Así, a partir de la tabla EMPLE, obtenemos el apellido de los ANALISTAS del departamento 10:

```
SELECT  APELLIDO
FROM EMPLE
WHERE  OFICIO = 'ANALISTA';
```

Pero este operador no nos sirve si queremos hacer consultas de este tipo: "obtener los datos de los empleados cuyo apellido empiece por una P" u "obtener los nombres de alumnos que incluyan la palabra Pérez".

Para especificar este tipo de consultas, en SQL usamos el operador LIKE que permite utilizar los siguientes caracteres especiales (comodines) en las cadenas de comparación:

- **'%'** Representa cualquier cadena de 0 o más caracteres.
- **'_'** Representa un carácter cualquiera.

En la cláusula WHERE este operador se utilizará de la siguiente manera:

WHERE columna LIKE 'caracteres_especiales'

Veamos un ejemplo:

```
SELECT  APELLIDO
FROM EMPLE
WHERE  OFICIO LIKE 'ANALIS%';
```

Podemos utilizarlo de diferentes formas. ¿Qué resultado nos darían las siguientes condiciones?

```
WHERE  OFICIO LIKE '%LISTA';
WHERE  OFICIO LIKE '%ALIS%';
WHERE  OFICIO LIKE 'ANA%TA';
WHERE  OFICIO LIKE '%ANALIS%';
WHERE  OFICIO LIKE '_A%TA';
```

4.1.2. BETWEEN Y NOT BETWEEN

El operador BETWEEN comprueba si un valor está comprendido o no (NOT) dentro de un rango de valores, desde un valor inicial a un valor final. Su formato es:

[NOT] BETWEEN valor_inicial AND valor_final

Podemos utilizarlo con valores de tipo numérico:

```
SELECT APELLIDO, SALARIO
FROM EMPLE
WHERE SALARIO BETWEEN 1500 AND 2000;
```

Podemos utilizarlo con valores de tipo fecha:

```
SELECT APELLIDO, SALARIO
FROM EMPLE
WHERE FECHA BETWEEN '01/10/2017' AND '15/07/2018';
```

Y también podemos comprobar si un valor NO se encuentra dentro del rango:

```
SELECT APELLIDO, SALARIO
FROM EMPLE
WHERE SALARIO NOT BETWEEN 1500 AND 2000;
```

4.1.3.IN

El operador IN nos permite comprobar si una expresión pertenece o no (NOT) a un conjunto de valores, haciendo posible la realización de comparaciones múltiples; su formato es:

<expresión> [NOT] IN (lista de valores separados por comas).

La lista de valores está formada por números:

Consultar los apellidos de la tabla EMPLE cuyo número de departamento sea 10 ó 30:

```
SELECT APELLIDO
FROM EMPLE
WHERE DEPT_NO IN(10,30);
```

Consultar los apellidos de la tabla EMPLE cuyo número de departamento no sea ni 10 ni 30:

```
SELECT APELLIDO
FROM EMPLE
WHERE DEPT_NO NOT IN(10,30);
```

La lista de valores está formada por cadenas:

Consultar los apellidos de la tabla EMPLE cuyo oficio sea 'VENDEDOR', 'ANALISTA' o 'EMPLEADO':

```
SELECT APELLIDO FROM EMPLE
WHERE OFICIO IN ('VENDEDOR', 'ANALISTA', 'EMPLEADO');
```

Consultar los apellidos de la tabla EMPLE cuyo oficio no sea ni 'VENDEDOR' ni 'ANALISTA' ni 'EMPLEADO':

```
SELECT APELLIDO FROM EMPLE
WHERE OFICIO NOT IN ('VENDEDOR', 'ANALISTA', 'EMPLEADO');
```

4.1.4. NULL Y NOT NULL

Se dice que una columna de una fila es NULL si está completamente vacía. Para comprobar si el valor de una columna es nulo empleamos la expresión:

columna IS NULL.

Si queremos saber si el valor de una columna **no es nulo**, usamos la expresión:

columna IS NOT NULL.

Cuando comparamos con valores nulos o no nulos no podemos utilizar los operadores de igualdad, mayor o menor.

Por ejemplo: a partir de la tabla EMPLE, consultamos los apellidos de los empleados cuya comisión es nula:

```
SELECT APELLIDO
FROM EMPLE
WHERE COMISION IS NULL;
```

Si queremos consultar los apellidos de los empleados cuya comisión no sea nula teclearemos esto:

```
SELECT APELLIDO
FROM EMPLE
WHERE COMISION IS NOT NULL;
```

4.2. OPERADORES ARITMÉTICOS Y DE CONCATENACIÓN.

Aprendimos que los operadores son símbolos que permiten realizar distintos tipos de operaciones. Los operadores aritméticos permiten realizar cálculos con valores numéricos. Son los siguientes: Operadores aritméticos y su significado.

OPERADOR	SIGNIFICADO
+	Suma
-	Resta
*	Multiplicación
/	División

Utilizando expresiones con operadores es posible obtener salidas en las cuales una columna sea el resultado de un cálculo y no un campo de una tabla.

En este ejemplo obtenemos el salario aumentado en un 5% de aquellos empleados que cobran menos de 1000€:

```
SELECT SALARIO*1,05
FROM EMPLEADOS
WHERE SALARIO<=1000;
```

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

Realizar Ejercicios 13-39 hoja UT5_ParteI_Basicos.

4.3. Operadores lógicos.

Habrá ocasiones en las que tengamos que evaluar más de una expresión y necesites verificar si se cumple una condición u otra o ninguna de ellas. Para poder hacer esto utilizaremos los operadores lógicos.

Tenemos los siguientes: Operadores lógicos y su significado.

OPERADOR	SIGNIFICADO
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 es verdadera.
NOT	Invierte la lógica de la expresión que le precede, si la expresión es verdadera devuelve falsa y si es falsa devuelve verdadera.

Si por ejemplo queremos obtener aquellos empleados en cuyo historial salarial tengan sueldo menor o igual a 800€ o superior a 2000€:

```
SELECT empleado_dni
FROM HISTORIAL_SALARIAL
WHERE salario<=800 OR salario>2000;
```

Vamos a realizar una consulta donde obtengamos todos nombres de trabajos menos el de contable:

```
SELECT NOMBRE_TRAB
FROM TRABAJOS
WHERE NOMBRE_TRAB NOT IN ('CONTABLE', 'ASESOR', 'ADMINISTRATIVO');
```

4.4. Precedencia.

Con frecuencia utilizaremos la sentencia SELECT acompañada de expresiones muy extensas y resultará difícil saber que 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 NOT NULL**, **LIKE**, **BETWEEN**.
6. **NOT**.
7. **AND**.
8. **OR**.

Si quisiéramos variar este orden necesitaríamos utilizar paréntesis.

5. CAMPOS CALCULADOS.

En algunas ocasiones es interesante realizar operaciones con algunos campos para obtener información derivada de éstos. Si tuviéramos un campo Precio, podría interesarnos calcular el precio incluyendo el IVA o si tuviéramos los campos Sueldo y Paga Extra, podríamos necesitar obtener la suma de los dos campos. Estos son dos ejemplos simples, pero podemos construir expresiones mucho más complejas. Para ello haremos uso de la creación de campos calculados.

Estos campos calculados se obtienen a través de la sentencia SELECT poniendo a continuación una expresión. Esta consulta no modificará los valores originales de las columnas ni de la tabla de la que se está obteniendo, únicamente mostrará una columna nueva con los valores calculados.

Por ejemplo:

SELECT Nombre, Credito, Credito + 25 FROM USUARIOS;

Con esta consulta hemos creado un campo que tendrá como nombre la expresión utilizada. Podemos ponerle un alias a la columna creada añadiéndolo detrás de la expresión junto con la palabra AS. En nuestro ejemplo quedaría de la siguiente forma:

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

Realizar ejercicios 03

6. FUNCIONES

Las funciones son operaciones que se realizan sobre un dato o sobre un conjunto de datos, en lugar de hacerlo sobre dos datos, que es lo que hace un operador. Por lo tanto, una función va a tener unos datos de entrada llamados parámetros o argumentos, sobre los que se realizará un cálculo.

La forma de utilizar una función es la siguiente:

NombreFuncion [(parametro1, [parametro2, …)]

Las funciones se pueden incluir en las cláusulas SELECT, WHERE y ORDER BY y se pueden anidar funciones dentro de funciones.

Existe diferentes tipos en función de los datos que tratan:

- ✓ Numéricas,
- ✓ Cadena de caracteres.
- ✓ Grupos de valores.
- ✓ Manejo de fechas.

- ✓ Conversión.
- ✓ Otras…

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.

Podremos utilizar la tabla Dual en algunos de los ejemplos que vamos a ver en los siguientes apartados.

6.1. FUNCIONES NUMÉRICAS.

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

FUNCIONES NUMÉRICAS	
ABS(n)	**Calcula el valor *absoluto* de un número n**
SELECT ABS(-17) FROM DUAL; -- Resultado: 17	
EXP(n)	**Calcula e^n, es decir, el exponente en base e del número n**
SELECT EXP(2) FROM DUAL; -- Resultado: 7,38	
CEIL(n)	**Calcula el valor entero inmediatamente superior o igual al argumento n**
SELECT CEIL(17.4) FROM DUAL; -- Resultado: 18	
FLOOR(n)	**Calcula el valor entero inmediatamente inferior o igual al parámetro n**
SELECT FLOOR(17.4) FROM DUAL; --Resultado: 17	
MOD(m, n)	**Calcula el resto resultante de dividir m entre n**
SELECT MOD(15, 2) FROM DUAL; --Resultado: 1	
POWER(valor, exponente)	**Eleva el valor al exponente indicado**
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.**
SELECT ROUND(12.5874, 2) FROM DUAL; -- Resultado: 12.59	
SQRT(n)	**Calcula la raíz cuadrada de n.**
SELECT SQRT(25) FROM DUAL; --Resultado: 5	

TRUNC(m,n)	**Trunca un número a la cantidad de decimales especificada por el segundo argumento. Si se omite el segundo argumento, se truncan todos los decimales. Si "n" es negativo, el número es truncado desde la parte entera.**
SELECT TRUNC(127.4567, 2) FROM DUAL; -- Resultado: 127.45 **SELECT TRUNC(4572.5678, -2) FROM DUAL; -- Resultado: 4500** **SELECT TRUNC(4572.5678, -1) FROM DUAL; -- Resultado: 4570** **SELECT TRUNC(4572.5678) FROM DUAL; -- Resultado: 4572**	
SIGN(n)	**Si el argumento "n" es un valor positivo, retorna 1, si es negativo, devuelve -1 y 0 si es 0.**
SELECT SIGN(-23) FROM DUAL; – Resultado: -1	

¿Cuál sería la salida de ejecutar estas funciones?:

- ✓ ABS(146)=
- ✓ ABS(-30)=
- ✓ CEIL(2)=
- ✓ CEIL(1.3)=
- ✓ CEIL(-2.3)=
- ✓ CEIL(-2)=
- ✓ FLOOR(-2)=
- ✓ FLOOR(-2.3)=
- ✓ FLOOR(2)=
- ✓ FLOOR(1.3)=
- ✓ MOD(22,23)=
- ✓ MOD(10,3)=
- ✓ POWER(10,0)=
- ✓ POWER(3,2)=
- ✓ POWER(3,-1)=
- ✓ ROUND(33.67)=
- ✓ ROUND(-33.67,2)=
- ✓ ROUND(-33.67,-2)=
- ✓ ROUND(-33.27,1)=
- ✓ ROUND(-33.27,-1)=
- ✓ -TRUNC(67.232)=
- ✓ TRUNC(67.232,3)=
- ✓ TRUNC(67.232,2)=
- ✓ TRUNC(67.58,2)=
- ✓ TRUNC(67.58,1)=

6.2. FUNCIONES DE CADENA DE CARACTERES.

Estas funciones trabajan con datos de tipo CHAR o VARCHAR2. Estos datos incluyen cualquier carácter alfanumérico: letras, números y caracteres especiales. Los literales se deben encerrar entre comillas simples. Ejemplo de una cadena de caracteres: 'El Quijote'.

También es posible anidar funciones de cadenas. Como resultado podremos obtener caracteres o números.

FUNCIONES DE CADENA DE CARACTERES	
CHR(n)	**Devuelve el carácter cuyo valor codificado es n.**

SELECT CHR(81) FROM DUAL; --Resultado: Q	
ASCII(n)	**Devuelve el valor ASCII de n**
SELECT ASCII('O') FROM DUAL; --Resultado: 79	
CONCAT(cad1, cad2)	**Devuelve las dos cadenas unidas. Es equivalente al operador ||**
SELECT CONCAT('Hola', 'Mundo') FROM DUAL; --Resultado: HolaMundo	
LOWER(cad)	**Devuelve la cadena cad con todos sus caracteres en minúsculas.**
SELECT LOWER('En Minúsculas') FROM DUAL; --Resultado: en minúsculas	
UPPER(cad)	**Devuelve la cadena cad con todos sus caracteres en mayúsculas.**
SELECT UPPER('En Mayúsculas') FROM DUAL; --Resultado: EN MAYÚSCULAS	
INITCAP(cad)	**Devuelve la cadena cad con su primer carácter en mayúscula.**
SELECT INITCAP('el patio de mi casa') FROM DUAL; --Resultado: El Patio De Mi Casa	
LPAD(cad1, n, cad2)	**Devuelve cad1 con longitud n, ajustada a la derecha, rellenando por la izquierda con cad2.**
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.**
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.**
SELECT REPLACE('correo@gmail.es', 'es', 'com') FROM DUAL; --Resultado: correo@gmail.com	
SUBSTR(cad, m, n)	**Devuelve una subcadena de la cadena cad compuesta por n caracteres a partir de la posición m.**
SELECT SUBSTR('1234567', 3, 2) FROM DUAL; --Resultado: 34	
LENGTH(cad)	**Devuelve la longitud de cad**
SELECT LENGTH('hola') FROM DUAL; --Resultado: 4	
TRIM(cad)	**Elimina los espacios en blanco a la izquierda y la derecha de cad.**
SELECT TRIM(' Hola de nuevo ') FROM DUAL; --Resultado: Hola de nuevo	
LTRIM(cad)	**Elimina los espacios a la izquierda que posea cad.**

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

6.3. FUNCIONES DE GRUPOS DE VALORES. (*)

Hasta ahora nos hemos ocupado de funciones que operan con valores simples; no obstante, hay otras funciones estadísticas, como SUM, AVG y COUNT, que actúan sobre un grupo de filas para obtener un valor. Los valores nulos son ignorados por las funciones de grupos de valores, y los cálculos se realizan sin contar con ellos.

FUNCIONES DE GRUPOS DE VALORES	
AVG(n)	Calcula el valor medio de "n" ignorando los valores nulos.
1.1.- SELECT AVG(SALARIO) FROM EMPLE WHERE DEPT_NO=10;	
COUNT (* | expresión)	Cuenta el número de veces que la expresión evalúa algún dato con valor no nulo. La opción "*" cuenta todas las filas seleccionadas.
SELECT COUNT(COMISION) FROM EMPLE;	
MAX(expresión)	Calcula el máximo valor de la "expresión".
SELECT MAX(SALARIO) FROM EMPLE;	
MIN(expresión)	Calcula el mínimo valor de la "expresión".
SELECT MIN(SALARIO)FROM EMPLE; *SELECT APELLIDO,SALARIO FROM EMPLE WHERE SALARIO=(SELECT MIN(SALARIO)FROM EMPLE);*	
SUM(expresión)	Obtiene la suma de valores de la "expresión" distintos de nulos.
SELECT SUM(SALARIO) FROM EMPLE;	

VARIANCE (expresión)	Obtiene la varianza de los valores de "expresión" distintos de nulos
SELECT VARIANCE(SALARIO) FROM EMPLE;	

6.3.1. DISTINCT EN FUNCIONES DE GRUPO.

En todas las funciones de grupo, al indicar los argumentos se pueden emplear las cláusulas DISTINCT y ALL, aunque no se suelen utilizar en las funciones AVG, SUM, MAX ni MIN, pero sí es más normal su uso en COUNT.

Recordemos que DISTINCT realiza una selección de filas cuyos valores en la columna especificada no estén duplicados. La cláusula ALL recoge todas las filas, aunque sus valores estén duplicados.

El formato de COUNT incluyendo DISTINCT y ALL es éste:

COUNT (* | [DISTINCT | ALL] expresión)

Si COUNT recibe como argumento una expresión o columna, ésta podrá ir precedida de las cláusulas ALL o DISTINCT.

Calcular el número de oficios que hay en la tabla EMPLE:

SELECT COUNT(OFICIO) "OFICIOS" FROM EMPLE;

Esta consulta cuenta todos los oficios de la tabla EMPLE que no sean nulos, estén repetidos o no. Si queremos contar los distintos oficios que hay en la tabla EMPLE, tendríamos que incluir DISTINCT en la función de grupo:

SELECT COUNT (DISTINCT OFICIO) "OFICIOS" FROM EMPLE;

DISTINCT obliga a COUNT a contar sólo el número de oficios distintos.

6.4. FUNCIONES DE LISTAS.

Las funciones de listas trabajan sobre un grupo de columnas dentro de una misma fila. Comparan los valores de cada una de las columnas en el interior de una fila para obtener el mayor o el menor valor de la lista.

FUNCIONES DE LISTAS	
FUNCIÓN	**PROPÓSITO**
GREATEST (valor1, valor2, ...)	Obtiene el mayor valor de la lista.
LEAST (valor1, valor2, ...)	Obtiene el menor valor de la lista.

Realizar Ejercicio01 Consultas Funciones.

6.5. FUNCIONES DE MANEJO DE FECHAS.

La fecha de emisión de una factura, de llegada de un avión, de ingreso en una web, son informaciones que utilizaremos con frecuencia. Tenemos dos tipos de datos para manejar fechas, que son DATE y TIMESTAMP.

- ✓ SYSDATE almacena una fecha concreta.
- ✓ SYSTIMESTAMP almacena un instante de tiempo concreto proporcionándonos información de la fecha completa, hora, minutos, segundos, fracciones de segundo, hasta el huso horario.

Podemos realizar operaciones numéricas con las fechas:

- ✓ 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.

FUNCIONES DE FECHA	
SYSDATE	Devuelve la **fecha y hora actuales**
SELECT SYSDATE FROM DUAL**; --Resultado: 26/07/11**	
SYSTIMESTAMP	Devuelve la **fecha y hora actuales** en formato TIMESTAMP
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**
SELECT ADD_MONTHS('27/07/11', 5) FROM DUAL; --Resultado: 27/12/11	
MONTHS_BETWEEN (fecha1, fecha2)	Devuelve el **número de meses** que hay **entre fecha1** y **fecha2**
SELECT MONTHS_BETWEEN('12/07/11','12/03/11') FROM DUAL; --Resultado: 4	
LAST_DAY(fecha)	Devuelve el **último día del mes** al que pertenece la fecha. El valor devuelto es tipo DATE
SELECT LAST_DAY('27/07/11') FROM DUAL; --Resultado: 31/07/11	
NEXT_DAY (fecha, d)	Indica el **día** que corresponde **si añadimos** a la fecha el día **d**. El día devuelto puede ser texto ('Lunes', Martes', ...) o el número del día (1=lunes, 2=martes, ...) dependiendo de la configuración.
SELECT NEXT_DAY('31/12/11'**,'LUNES') FROM DUAL; --Resultado: 02/01/12**	
EXTRACT (valor FROM fecha)	Extrae un valor de una fecha concreta. El valor puede ser day, month, year, hours, etc.
SELECT SYSDATE, EXTRACT(MONTH FROM SYSDATE) FROM DUAL; **SELECT SYSDATE, EXTRACT(DAY FROM SYSDATE) FROM DUAL;** **SELECT SYSDATE, EXTRACT(YEAR FROM SYSDATE) FROM DUAL;**	

6.6. 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 queramos realizar esas conversiones de modo explícito, para ello contamos con funciones de conversión.

1.2.- FORMATOS PARA NÚMEROS Y SU SIGNIFICADO.	
1.3.- TO_CHAR (fecha, 'formato')	1.4.- Convierte una “fecha” a tipo “VARCHAR2” en el “formato” especificado. El “formato” es una cadena de caracteres que puede incluir las máscaras de formato definidas en la tabla siguiente.
1.5.- SELECT TO_CHAR (sysdate, 'month DD, YYYY'), TO_CHAR (sysdate, 'mm-dd-YY') yy, **1.6.- TO_CHAR (sysdate, 'mm') mm, TO_CHAR (sysdate, 'YY') aa,** **1.7.- TO_CHAR (sysdate, 'YYYY') aaaa, TO_CHAR (sysdate, 'mon') mes,** **1.8.- TO_CHAR (sysdate, 'MON') mesmayus** **1.9.- FROM DUAL;**	
1.10.- TO_CHAR (numero, 'formato')	1.11.- Esta función convierte un "numero" a tipo VARCHAR2 en el "formato" especificado.
1.12.- SELECT TO_CHAR (2244, '9999L'), TO_CHAR (2345, '09999'), TO_CHAR (1234, '9,999.00L') FROM DUAL;	
1.13.- TO_DATE (cad, 'formato')	1.14.- Convierte "cad", de tipo VARCHAR2 o CHAR, a un valor de tipo DATE según el "formato" especificado.
1.15.- SELECT TO_DATE('01-01-2018', 'dd/mm/yyyy'), TO_DATE ('01012018', 'dd/mm/yyyy'), **1.16.- TO_DATE ('01012018', 'dd/mm/yyyy'), TO_DATE ('01012018', 'ddmmyyyy')** 1.17.- **FROM DUAL;**	
1.18.- TO_NUMBER (cad [,'formato'])	1.19.- Convierte la “cad" a tipo NUMBER según el "formato" especificado. La cadena ha de contener números, el carácter decimal o el signo menos a la izquierda. No puede haber espacios entre los números, ni otros caracteres.
1.20.- **SELECT TO_NUMBER ('1234,23', '9999D99') FROM DUAL;**	

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

MÁSCARAS DE FORMATO FECHA	
CC O SCC	Valor del siglo.
SELECT SYSDATE, TO_CHAR(SYSDATE, 'CC') FROM DUAL; **SELECT SYSDATE, TO_CHAR(SYSDATE, 'SCC') FROM DUAL;**	
YYYY	Año sin signo
SELECT SYSDATE, TO_CHAR(SYSDATE, 'YYYY') FROM DUAL;	
YYY	Últimos tres dígitos del año
SELECT SYSDATE, TO_CHAR(SYSDATE, 'YYY') FROM DUAL;	
YY	Últimos dos dígitos del año
SELECT SYSDATE, TO_CHAR(SYSDATE, 'YY') FROM DUAL;	
Y	Ultimo dígito del año
SELECT SYSDATE, TO_CHAR(SYSDATE, 'Y') FROM DUAL;	
Q	Número del trimestre
SELECT SYSDATE, TO_CHAR(SYSDATE, 'Q') FROM DUAL;	
WW	Número de semana del año
SELECT SYSDATE, TO_CHAR(SYSDATE, 'WW') NUMSEMANAAÑO FROM DUAL;	
W	Número de semana del mes
SELECT SYSDATE, TO_CHAR(SYSDATE, 'W') NUMSEMANAMES, FROM DUAL;	
MM	Número de mes
SELECT SYSDATE, TO_CHAR(SYSDATE, 'MM') MES FROM DUAL;	
DDD	Número de día del año
SELECT SYSDATE, TO_CHAR(SYSDATE, 'DDD') NUMDIAAÑO FROM DUAL;	
DD	Número de día del mes
SELECT SYSDATE, TO_CHAR(SYSDATE, 'DD') DIAMES FROM DUAL;	
D	Número de día de la semana
SELECT SYSDATE, TO_CHAR(SYSDATE, 'D') DIASEMANA FROM DUAL;	
HH o HH12	Muestra la hora en formato (1-12)
SELECT SYSDATE, TO_CHAR(SYSDATE, 'HH12') HORA12 FROM DUAL;	
HH24	Muestra la hora en formato (1-24)
SELECT SYSDATE, TO_CHAR(SYSDATE, 'HH24') HORA24 FROM DUAL;	
MI	Muestra los Minutos
SELECT SYSDATE, TO_CHAR(SYSDATE, 'MI') MINUTOS FROM DUAL;	
SS	Muestra los segundos
SELECT SYSDATE, TO_CHAR(SYSDATE, 'SS') SEG FROM DUAL;	
SSSSS	Muestra los segundos transcurridos desde medianoche
SELECT SYSDATE, TO_CHAR(SYSDATE, 'SSSSS') MINUTOSMEDIANOCHA FROM DUAL;	

J	Juliano
SELECT SYSDATE, TO_CHAR(SYSDATE, 'J') JULIANO FROM DUAL;	

MÁSCARAS DE FORMATO DE FECHA	
YEAR	Año en inglés
SELECT SYSDATE, TO_CHAR(SYSDATE, 'YEAR') AÑO FROM DUAL;	
MONTH - Month - month	Nombre del mes
SELECT SYSDATE, TO_CHAR(SYSDATE, 'MONTH') MESMAYUSCULAS FROM DUAL; **SELECT SYSDATE, TO_CHAR(SYSDATE, 'MONTH') MESPRIMERAMAYUSCULAS FROM DUAL;** **SELECT SYSDATE, TO_CHAR(SYSDATE, 'MONTH') MESMINUSCULAS FROM DUAL;**	
MON - Mon - mon	Abreviatura de tres letras del nombre del mes
SELECT SYSDATE, TO_CHAR(SYSDATE, 'MON') MESCORTOMAYUSCULAS FROM DUAL; **SELECT SYSDATE, TO_CHAR(SYSDATE, 'MON') MESCORTOPRIMERAMAYUSCULAS FROM DUAL;** **SELECT SYSDATE, TO_CHAR(SYSDATE, 'MON') MESCORTOMINUSCULAS FROM DUAL;**	
DAY - Day -day	Nombre del día de la semana
SELECT SYSDATE, TO_CHAR(SYSDATE, 'DAY') DIAMAYUSCULAS FROM DUAL; **SELECT SYSDATE, TO_CHAR(SYSDATE, 'DAY') DIAPRIMERAMAYUSCULAS FROM DUAL;** **SELECT SYSDATE, TO_CHAR(SYSDATE, 'DAY') DIAMINUSCULAS FROM DUAL;**	
DY - Dy - dy	Abreviatura de tres letras del nombre del día (LUN)
SELECT SYSDATE, TO_CHAR(SYSDATE, 'DY') DIACORTOMAYUSCULAS FROM DUAL; **SELECT SYSDATE, TO_CHAR(SYSDATE, 'DY') DIACORTOPRIMERAMAYUSCULAS FROM DUAL;** **SELECT SYSDATE, TO_CHAR(SYSDATE, 'DY') DIACORTOMINUSCULAS FROM DUAL;**	
A.M. o P.M.	Muestra a.m. ó p.m. dependiendo del momento del día
SELECT SYSDATE, TO_CHAR(SYSDATE, 'AM') AÑO FROM DUAL;	
B.C. o A.D.	Indicador para el año (antes de Cristo o después de Cristo)
SELECT SYSDATE, TO_CHAR(SYSDATE, 'A.D') AÑO FROM DUAL;	

Realizar Ejercicio02 Consultas Funciones.

Para las funciones **TO_NUMBER** y **TO_CHAR**, en el caso de valores numéricos, indicamos el formato incluyendo los siguientes símbolos:

MÁSCARAS DE FORMATO NÚMEROS.	
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.

FORMATO	EJEMPLO
9	Devuelve el valor con el número especificado de dígitos. Si el valor tiene ceros a la izquierda, los deja en blanco. Si el valor es 0, la salida es 0. Si el valor es negativo pone el signo delante, a la izquierda.
SELECT TO_CHAR (1,'999'), TO_CHAR (-1,'999'), TO_CHAR (01,'999'), TO_CHAR (0,'999') FROM DUAL;	
0	Muestra un 0 si el valor es 0, o bien devuelve el valor dejando ceros al principio.
SELECT TO_CHAR (10,'0999'), TO_CHAR (10,'9990'), TO_CHAR (10,'990090') FROM DUAL;	
$	Devuelve el valor con el signo dólar a la izquierda.
SELECT TO_CHAR (10,'$9999'), TO_CHAR (10,'$009'), TO_CHAR (10,'99$') FROM DUAL;	
B	Muestra un espacio en blanco si el valor es 0. Es el formato por omisión.
SELECT TO_CHAR(0,'B999'), TO_CHAR(050,'B999') FROM DUAL;	
MI	Si el número es negativo, el signo menos sigue al número. Por omisión, el signo se pone a la izquierda.
SELECT TO_CHAR(-55,'999MI'), TO_CHAR(55,'999MI'), TO_CHAR (-55,'999') FROM DUAL;	
S	'S' representa el signo. Devuelve el valor con el signo '+' si el valor es positivo o con el signo '–' si es negativo.
SELECT TO_CHAR(-55,'999S'), TO_CHAR(-55,'S999'), TO_CHAR(55,'S999'), TO_CHAR(55,'999S') FROM DUAL;	
PR	Los números negativos se muestran entre estos símbolos: < >.
SELECT TO_CHAR(-55.33,'9999PR'), TO_CHAR(55,'9999PR') FROM DUAL;	
D	Devuelve el carácter decimal en la posición especificada.
SELECT TO_CHAR(34.55,'99D99'), TO_CHAR(34.55,'999D999'), TO_CHAR(34.55,'099D990') FROM DUAL;	
G	Devuelve el carácter de grupo (carácter de los miles) en la posición especificada.
SELECT TO_CHAR(1234,'9G999') FROM DUAL; SELECT TO_CHAR(123456.98, '999G999D99') FROM DUAL;	
C	Devuelve el símbolo ISO de la moneda del territorio en la posición especificada.
SELECT TO_CHAR(123,'999C') "ISO" FROM DUAL;	
L	Devuelve el símbolo de la moneda local en la posición indicada.
SELECT TO_CHAR(123,'L999'), TO_CHAR(123,'999L') "MONEDA" FROM DUAL;	

, (coma)	Devuelve la coma en la posición especificada (carácter de los miles).
SELECT TO_CHAR (1234,'9,999'), TO_CHAR (1234,'009,999'), TO_CHAR (1234,'999,999') FROM DUAL;	
. (punto)	Devuelve el punto decimal en la posición especificada. Formato Americano.
SELECT TO_CHAR (12.34,'99.99') FROM DUAL; SELECT TO_CHAR(12345.67,'99,999.99') FROM DUAL;	
V	Devuelve el valor multiplicado por 10^n, donde n es el número de dígitos después V.
SELECT TO_CHAR(123.45,'999V99'), TO_CHAR(123,'999V99') FROM DUAL;	
EEEE	Devuelve el valor usando notación científica.
SELECT TO_CHAR(12345,'9.9EEEE') FROM DUAL;	
RN o rn	Devuelve el valor en números romanos, en mayúsculas (RN), o en minúsculas (rn).
SELECT TO_CHAR(12,'RN'), TO_CHAR(12,'rn') FROM DUAL;	

6.7. VALORES DEVUELTOS POR LOS FORMATOS.

Los caracteres devueltos en algunos de estos formatos se especifican inicializando una serie de parámetros.

PARÁMETRO	VALOR	DESCRIPCIÓN
NLS_NUMERIC_CHARACTERS	**D, G**	Define los caracteres decimales ('D') y separador de los miles ('G'). Formato: NLS_NUMERIC_CHARACTERS="DG" D: carácter decimal. G: separador de miles. Ejemplo: NLS_NUMERIC_CHARACTERS=",." Carácter decimal, la coma, y separador de miles, el punto.
NLS_ISO_CURRENCY	**C**	Especifica el símbolo del territorio. Para España el símbolo es 'ESP'
NLS_CURRENCY	**L**	Especifica el símbolo de la moneda local. Para España el €

Para cambiar el valor de estos parámetros se utiliza la orden ALTER SESSION. Supongamos, por ejemplo, que queremos definir como carácter de los miles el asterisco (*) y como carácter decimal la barra (/). Hemos de usar el parámetro NLS_NUMERIC_CHARACTERS con ALTER SESSION:

ALTER SESSION SET NLS_NUMERIC_CHARACTERS='/*';

Sesión modificada.

SELECT TO_CHAR(12345.67,'999G999D999') FROM DUAL;

Los valores para el carácter decimal y de los miles permanecerán hasta que el usuario finalice la sesión o hasta que el usuario aplique de nuevo la orden ALTER SESSION para cambiar estos caracteres:

ALTER SESSION SET NLS_NUMERIC_CHARACTERS=',.';

TO_NUMBER (cadena [,'formato']). Suponemos que el carácter decimal es la coma y el carácter separador de los miles, el punto.

SELECT TO_NUMBER ('-123456') "NUMERO1", TO_NUMBER('123,99','999D99') "NUMERO2" FROM DUAL;

SELECT TO_NUMBER('123.456','999G999') "NO CONVIERTE" FROM DUAL;

Este ejemplo no convierte porque la cadena '123.456' contiene el carácter que define el separador de los miles (en este ejemplo, el punto), y una cadena válida ha de contener el carácter decimal (en este caso la coma). No da un error de sintaxis, pero no realiza la conversión.

SELECT TO_NUMBER('123,456','999d999') "SI CONVIERTE" FROM DUAL;

TO_DATE (cadena, 'formato'). Cambiar el formato de la fecha para que aparezca el año con cuatro dígitos:

ALTER SESSION SET NLS_DATE_FORMAT='DD/MM/YYYY';

Convertir una cadena a tipo DATE:

SELECT TO_DATE ('01012006') FROM DUAL;

Cuando en la orden **TO_DATE** no se indica el formato, una cadena de caracteres será convertida a fecha sólo si está en el formato que tenga la fecha del sistema. En el siguiente ejemplo no se convierte la cadena a tipo fecha porque no está en el formato 'DDMMYYYY' definido en la sesión para la fecha:

SELECT TO_DATE ('010106') FROM DUAL;

Nos da el siguiente error

ERROR en línea 1:
ORA-01861: el literal no coincide con la cadena de formato

Lo correcto sería:

SELECT TO_DATE ('01012006') FROM DUAL;

Obtener el nombre del mes a partir de la cadena '01012007' (antes hay que convertir la cadena a tipo fecha):

SELECT TO_CHAR (TO_DATE ('01012007', 'ddmmyyyy'), 'Month') "MES" FROM DUAL;

TO_DATE y **TO_CHAR** son similares; la diferencia que los separa estriba en que **TO_DATE** convierte una cadena de caracteres en una fecha y **TO_CHAR** convierte una fecha en una cadena de caracteres. Ambas pueden utilizar las máscaras de formato de fechas.

6.8. OTRAS FUNCIONES.

6.8.1. NVL – NVL2

Los valores de un campo pueden tomar valores nulos (NULL).

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 dé un valor nulo. Por tanto, es habitual encontrarnos con estos valores y es entonces cuando aparece la necesidad de poder hacer algo con ellos.

La función **NVL** nos permitirá evitar la aparición de valores nulos.

NVL (valor, expr1)

Si **valor** es NULL, entonces devuelve **expr1**. Ten en cuenta que **expr1** debe ser del mismo tipo que **valor**.

El valor que evaluamos puede ser una expresión.

```
SELECT NVL (PROVINCIA, 'No tiene provincia')
FROM ALUMNOS;

SELECT NVL (FALAS1, 0)
FROM NOTAS_ALUMNOS;
```

La función **NVL2** comprueba si una expresión es nula, si NO lo es devuelve **expr1**, si lo es devuelve **expr2**.

NVL2 (valor, expr1, expre2)

El valor que evaluamos puede ser una expresión.

```
SELECT NVL2 (PROVINCIA, CONCAT ('La provincia es: ', PROVINCIA), 'No tiene provincia')
FROM ALUMNOS;
```

6.8.2. NULLIF.

Esta función evalúa dos valores, de tal forma que devuelve **null** si los dos valores son iguales y si no lo son devuelve el primer valor.

NULLIF (valor1, valor2).

Si **valor1** y **valor2** son iguales devuelve **null** y no los son devuelve **valor1**

```
SELECT NOMBRE_ALUMNO, NOTA1, NOTA2, NULLIF (NOTA1, NOTA2)
FROM NOTAS_ALUMNOS;
```

6.8.3. COALESCE.

Esta función es similar a **NVL** y **NVL2**. Evalúa un valor y devuelve el primer valor de la lista de valores que no sea **null.**

COALESCE (valor, valor1, valor2, ..., valorn)

El valor que evaluamos puede ser una expresión.

```
SELECT NOMBRE_ALUMNO, NOTA1, NOTA2, NOTA3, COALESCE (NOTA1, NOTA2, NOTA3)
FROM NOTAS_ALUMNOS;
```

Si todos los valores de la lista de parámetros son **null**, devuelve **null**.

6.8.4. DECODE.

Esta función compara el primer parámetro con el resto. Si es igual a cualquier valor de la lista ("valo1", "valor2"...), devuelve el correspondiente código ("codigo1", "codigo2"...). En caso contrario se obtiene el valor por defecto. Funciona como una evaluación múltiple de un valor.

DECODE (valor, valor1, codigo1, valor2, codigo2..., valor_por_defecto).

Veamos un ejemplo:

```
SELECT NOMBRE_ALUMNO, NOTA1, NOTA2, NOTA3,
DECODE (NOTA1, NOTA2,'iguales el primero y el segundo', -- compara NOTA1 con NOTA2
               NOTA3,'iguales el primero y el tercer',  -- compara NOTA1 con NOTA3
```

```
                'distinto')                    -- valor por defecto
    FROM NOTAS_ALUMNOS;
```

Los valores que evaluamos puede ser expresiones:

```
SELECT NOMBRE_ALUMNO, NOTA1, NOTA2, NOTA3,
DECODE (NOTA1, NOTA2, NOTA1 || ' de la nota1 es igual que la nota2 ' || NOTA2,
                NOTA3, NOTA1 || ' de la nota 1 es igual que la nota3 ' || NOTA3,
                'los valores de nota1 ' ||NOTA1 ||', nota2 '||NOTA2 ||' y nota3
'||NOTA3||' son distintos')
FROM NOTAS_ALUMNOS;
```

6.8.5. CASE.

Esta función es muy parecida a un **DECODE,** realiza la evaluación múltiple de un valor.

```
CASE [valor]
        WHEN valor1 THEN valor_retorno1
        [ WHEN valor2 THEN valor_retorno2
        (..)
        WHEN valorN THEN valor_retornoN
        ELSE default]
END
```

Los valores que evaluamos puede ser expresiones.

```
SELECT NOMBRE_ALUMNO,
    round((NVL(NOTA1,0) + NVL(NOTA2,0) + NVL(NOTA3,0))/3) MEDIA,
    CASE round((NVL(NOTA1,0) + NVL(NOTA2,0) + NVL(NOTA3,0)/3))
      WHEN 1 THEN 'MUY DEFICIENTE'
      WHEN 2 THEN 'MUY DEFICIENTE'
      WHEN 3 THEN 'MUY DEFICIENTE'
      WHEN 4 THEN 'INSUFICIENTE'
      WHEN 5 THEN 'SUFICIENTE'
      WHEN 6 THEN 'BIEN'
      WHEN 7 THEN 'NOTABLE'
      WHEN 8 THEN 'NOTABLE'
      WHEN 9 THEN 'SOBRESALIENTE'
      WHEN 10 THEN 'MATRÍCULA'
      ELSE 'INCORRECTA'
      END AS NOTA_LETRA
FROM NOTAS_ALUMNOS;
```

6.8.6. CASE Y DECODE.

CASE y **DECODE** manejan los valores nulos de forma diferente:

- ✓ **CASE** no es capaza de interpretar los valores **NULL** y no satisface la comparación **NULL = NULL** devuelve **false**.
- ✓ **DECODE** sí lo hace y **NULL = NULL** devuelve **true**.

```
SELECT
        CASE NULL
                WHEN NULL THEN 'NULL Es nulo'
                ELSE 'NULL no es nulo'
        END AS "CASE",

        DECODE (NULL, NULL, 'NULL es nulo', 'NULL no es nulo')
                AS "DECODE"
FROM DUAL;
```

Realizar Ejercicio03 Consultas Funciones.

Realizar Ejercicio04 Consultas Funciones.