C.F.G.S. DESARROLLO DE APLICACIONES WEB

DISTANCIA

MÓDULO:

BASES DE DATOS

Unidad 4:

Realización de Consultas.

ÍNDICE DE CONTENIDOS

OBJETIVOS

SQL es un lenguaje estándar en la mayoría de las bases de datos relacionales con el que normalmente se crean accesos a la información que contienen, mediante la realización de diferentes tipos de operaciones, lanzando consultas y obteniendo datos de interés para el usuario, de una forma sencilla.
En esta unidad conocerás los distintos tipos de consultas que puedes realizar tanto a una tabla como a varias, aprovechando las relaciones del modelo creado.

Por tanto los objetivos son:

- Conocer el estándar SQL como lenguaje para manipular las tablas y sus datos.
- Utilizar las sentencias, operadores y funciones que constituyen la base de SQL.
- Trabajar con múltiples tablas y vistas.

1.- Introducción.

Ya hemos visto 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. La compañía 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.

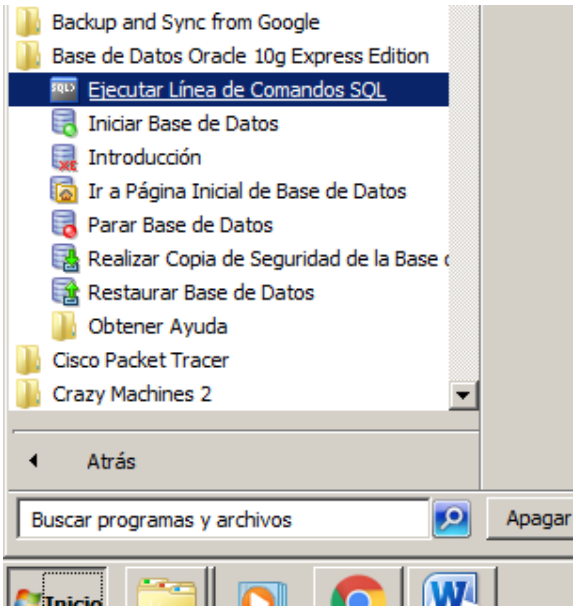
En 1992, ANSI (Instituto Nacional de Normalización) e ISO (Organización Internacional para la estandarización) completaron la estandarización de SQL y se definieron las sentencias básicas que debía contemplar SQL para que fuera estándar. A este SQL se le denominó ANSI-SQL o SQL92.
Hoy en día todas las bases de datos comerciales cumplen con este estándar, eso sí, cada fabricante añade sus mejoras al lenguaje SQL.

La primera fase del trabajo con cualquier base de datos comienza con sentencias DDL (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 (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.
De momento nos centraremos en una de ellas, que es la sentencia para consultas: SELECT

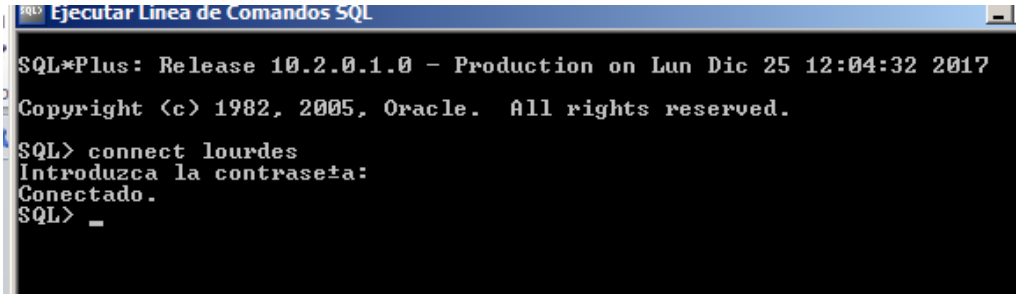
Las sentencias SQL que se verán a continuación pueden ser ejecutadas desde el entorno web Application Express de Oracle utilizando el botón SQL en la página de inicio, y desplegando su lista desplegable elegir **Comandos SQL** → **Introducir Comando**.

También se pueden indicar las sentencias SQL desde el entorno de **SQL*Plus** que ofrece Oracle y que puedes encontrar en:

Inicio → Todos los programas → Base de Datos Oracle Express Edition → Ejecutar Línea de Comandos SQL.



Si optas por abrir esa aplicación (**Ejecutar Línea de Comandos SQL**), el primer paso que debe realizarse para manipular los datos de una determinada tabla, es conectarse utilizando un nombre de usuario con los permisos necesarios para hacer ese tipo de operaciones a la tabla deseada. Utiliza para ello la orden **CONNECT** seguida del nombre de usuario. Posteriormente, solicitará la contraseña correspondiente a dicho usuario.



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.

2.- La sentencia SELECT.

Para recuperar o seleccionar los datos, de una o varias tablas tenemos que utilizar la sentencia SELECT.

Según el nivel de complejidad indicado, se puede obtener:

- ✓ Una unidad de datos
- ✓ Todos los datos
- ✓ Un subconjunto de datos
- ✓ Cualquier conjunto de subconjunto de datos

El formato es el siguiente:

```
SELECT [ALL/DISTINCT]
[expre_colum1, expre_colum2,...., expre_columN || * ]
FROM [nombre_tabla1,...., nombre_tablaN ]
[WHERE condición ]
[ORDER BY expre_colum [DESC || ASC ] [, expre_colum [DESC || ASC ] ..];
```

Este formato consta de varias partes:

- Cláusula **SELECT.** → Obligatoria.
- Cláusula **FROM**. → Obligatoria.
- Cláusula **WHERE**. → Opcional.
- Cláusula **ORDER BY**. → Opcional.

2.1.- Cláusula SELECT.

La 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 separadas por comas simples. Si en lugar de solicitar una columnas queremos ver todas pondremos asterisco *****. Esta parte es obligatoria.

Para referenciar una columna se puede indicar de las siguientes formas:

- ✓ **nombreColumna**
- ✓ **nombreTabla.nombreColumna**
- ✓ **esquemaUsuario.nombreTabla.nombreColumna**

Siempre que la sentencia afecte a más de una tabla es aconsejable especificar la tabla a la que pertenece cada columna.

Las cláusulas **ALL** y **DISTINCT** son opcionales.

- ✓ Si incluyes la cláusula **ALL** después de **SELECT**, indicarás que quieres seleccionar todas las filas estén o no repetidas. Es el valor por defecto y no se suele especificar.
- ✓ Si incluyes la cláusula **DISTINCT** después de **SELECT**, se suprimirán aquellas filas del resultado que tengan igual valor.

2.2.- Cláusula FROM.

La cláusula **FROM** seguida del nombre de las tablas de las que proceden las columnas especificadas anteriormente, es decir, de donde vas a extraer los datos. Esta parte también es obligatoria.

Si el usuario que hace la consulta no es el propietario de la tabla es necesario especificar el nombre de usuario delante de ella

Si el usuario que hace consulta no es el propietario de la tabla es necesario especificar el nombre de usuario delante de ella separadas pos un punto, es decir hay que poner **esquemaUsuario.NombreTabla.NonmbreColumna**

Ejemplo:
Desde el usuario propietario: (En este caso profesor)

```
SELECT * FROM alumnos;
```

Desde otro usuario:

```
SELECT * FROM profesor.alumnos;
```

También podemos ponerle 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. Veamos un ejemplo:

Ejemplo:

```
SELECT FechaNac "Fecha de Nacimiento"  FROM usuarios;
```

También podemos sustituir el nombre de las columnas por **constantes**, **expresiones** o **funciones** SQL. Por ejemplo:

Ejemplo:

```
SELECT 4*3/100 "MiExpresión", password FROM usuarios;
```

- ✓ **A los nombres de las tablas:** Es posible utilizar ALIAS como nombre de una tabla. Por ejemplo:

Ejemplo:

```
SELECT  Alu.nombre, Alu.nota   FROM   alumnos   Alu;
```

2.3.- Cláusula WHERE.

La cláusula **WHERE** seguida de un criterio de selección o condición. Esta parte es opcional.
Hasta ahora hemos podido ver la sentencia **SELECT** para obtener 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 que han de cumplir las filas para salir como resultado de dicha consulta.
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.

Ejemplo:
Si en nuestra tabla **USUARIOS**, necesitáramos un listado de los usuarios que son mujeres, bastaría con crear la siguiente consulta:

```
SELECT nombre, apellidos  FROM usuarios  WHERE sexo = 'M';
```

Más adelante veremos los operadores con los que podrás crear condiciones de diverso tipo.

2.4.- Cláusula ORDER BY.

Cláusula **ORDER BY** seguida por un criterio de ordenación. Esta parte también es opcional.

En la consulta del ejemplo anterior hemos obtenido una lista de nombres y apellidos de las usuarias de nuestro juego. Sería conveniente que aparecieran ordenadas por apellidos, ya que siempre quedará más profesional además de más práctico. De este modo, si necesitáramos localizar un registro concreto la búsqueda sería más rápida. ¿Cómo lo haremos? Para ello usaremos la cláusula ORDER BY.

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

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.

La sintaxis sería:

```
ORDER BY columna1 [ASC | DESC], columna2 [ASC | DESC], …,  columnaN [ASC | DESC];
```

Ejemplo:
Queremos ordenar 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.

> **Ejemplo:**
> Si queremos el resultado del ejemplo anterior ordenado por localidad:
>
> SELECT nombre, apellidos, localidad FROM usurious 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.

¿Se puede utilizar cualquier tipo de datos para ordenar? No todos los tipos de campos te servirán para ordenar, únicamente aquellos de tipo **carácter**, **número** o **fecha.**

3.- Operadores.

Veíamos que en la cláusula WHERE podíamos incluir expresiones para filtrar el conjunto de datos que queríamos obtener. Para crear esas expresiones necesitas utilizar distintos operadores de modo que puedas comparar, utilizar la lógica o elegir en función de una suma, resta, etc.

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

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

3.1.- Operadores de comparación.

Se les conoce con otros nombres como **relacionales**, nos permitirán comparar expresiones, que pueden ser valores concretos de campos, variables, etc.

Los operadores de comparación son símbolos que se usan como su nombre indica para comparar dos valores. Si el resultado de la comparación es correcto la expresión considerada es verdadera, en caso contrario es falsa.

Tenemos los siguientes operadores de comparación:

OPERADOR	SIGNIFICADO
=	Igual
!=, <>	Distinto
<	Menor
<=	Menor o igual
>	Mayor
>=	Mayor o igual
IN	Igual que cualquiera de un conjunto de valores Entre paréntesis.
NOT IN	Distinto que cualquiera de un conjunto de valores Entre paréntesis.
BETWEEN	Entre. Contenido dentro del rango.
NOT BETWEEN	Fuera de rango

Ejemplo:

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

```
SELECT nombre FROM empleados WHERE salario > 1000;
```

3.2.- Operadores aritméticos y de concatenación.

Los operadores aritméticos permiten realizar cálculos con valores numéricos.

Permiten formar expresiones con constantes, valores de columnas y funciones de valores de columnas.

Son los siguientes:

OPERADOR	OPERACIÓN
+	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.

Ejemplo:

Sobre una tabla **trabajadores** en el que obtenemos el salario aumentado en un 5 % de aquellos trabajadores que cobran menos de 1000 €:

```
SELECT salario*1,05 FROM trabajadores WHERE salario <= 1000;
```

Para concatenar cadenas de caracteres existe el operador de **concatenación (" | | ").** Oracle puede convertir automáticamente valores numéricos a cadenas para una concatenación.

Ejemplo:

En la tabla **trabajadores** tenemos separados en dos campos el primer y segundo apellido de los empleados, si necesitáramos mostrarlos juntos podríamos crear la siguiente consulta:

```
SELECT nombre, apellido1 || apellido2  FROM trabajadores;
```

Si queremos dejar un espacio entre un apellido y otro, debemos concatenar también el espacio en blanco de la siguiente manera:

```
SELECT nombre, apellido1 || ' ' ||apellido2 FROM trabajadores;
```

3.3.- Operadores lógicos.

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

Tenemos los siguientes:

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

Ejemplo:

Si queremos obtener aquellos empleados en cuyo salario sea menor o igual a 800€ o superior a 2000€:

```
SELECT dniEmpleado FROM  trabajadores WHERE salario <= 800 OR salario > 2000;
```

3.4.- Operador LIKE y patrones de búsqueda.

El operador LIKE se utiliza sobre todo con textos y permite realizar consulta de datos que se ajusten al formato especificado por los caracteres comodines.

El carácter comodín puede aparecer en cualquier parte del formato.

COMODIN	SIGNIFICADO
'%'	Cualquier cadena de 0 o más caracteres
'_'	Marcador de posición. Representa un carácter cualquiera.

Ejemplos:

like 'P%' → busca cadenas que comiencen por P;

not like 'P%' → busca cadenas que no comiencen P

like '_A%' : → busca cadenas que comiencen con una A en 2º posición

like '%Z' → busca cadenas que acaben en Z

like '___' → busca cadenas que tengan 3 caracteres

Para buscar cadenas que contengan los caracteres usados como comodines en los patrones, es decir como literales, se debe utilizar la opción ESCAPE indicando el carácter de escape

```
.....LIKE 'patron \_ patron'  ESCAPE '\';
.....LIKE 'patron \% patron'  ESCAPE '\';
```

Ejemplos:

... like 'A\%%' ESCAPE '\' : busca cadenas comiencen con A% ....

... like '%_Z' ESCAPE '\' : busca cadenas que acaben en _Z

3.5.- Operadores NULL y NOT NULL.

El valor **NULL** significa valor inexistente y por tanto es tratado de forma distinta a otros valores. Si queremos verificar que un valor es **NULL** no serán válidos los operadores de comparación. 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.

Ejemplo:

```
SELECT * FROM alumnos WHERE nota IS NULL  // alumnos que no tengan nota

SELECT * FROM alumnos WHERE nota IS NOT NULL  //alumnos que tengan un valor numérico en nota
```

3.5.- Precedencia de los operadores.

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, IS NOT NULL, LIKE, BETWEEN.
6.- NOT, AND, OR.

Si quisiéramos variar este orden necesitaríamos utilizar paréntesis.

4.- Consultas Calculadas.

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.

Los operadores aritméticos se pueden utilizar para hacer cálculos en las consultas.

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

Ejemplo:

```
SELECT nombre, credito, credito + 25 FROM trabajadores;
```

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:

Ejemplo:

```
SELECT nombre, credito, credito + 25 AS creditoNuevo FROM trabajadores;
```

5.- Funciones.

¿Has pensado en todas las operaciones que puedes realizar con los datos que guardas en una base de datos? Seguro que son muchísimas. Pues bien, 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 incluir en las cláusulas SELECT, WHERE y ORDER BY.

Las funciones se especifican de la siguiente manera:

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

Puedes anidar funciones dentro de funciones.

Existe una gran variedad para cada tipo de datos:

- ✓ Numéricas o aritméticas.
- ✓ De cadena de caracteres,
- ✓ De manejo de fechas,
- ✓ De 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.

Si queremos ver su descripción utilizamos la sentencia DESCRIBE (abreviado DESC) es una sentencia de SQL que nos permite ver la estructura de una tabla

```
DESC DUAL;
```

Tipo de Objeto **TABLE** Objeto **DUAL**

Table	Column	Tipo De Dato	Longitud	Precisión	Escala	Clave Primaria	Nulo	Valor Por Defecto	Comentario
DUAL	DUMMY	Varchar2	1	-	-	-	✓	-	-
									1 - 1

Vamos a utilizar la tabla Dual para probar algunas de las funciones que dispone Oracle y que veremos a continuación.

5.1.- Funciones Aritméticas o numéricas.

Estas funciones trabajan con datos de tipo NUMBER.

En función al grupo de valores numéricos con los que trabajan tenemos 3 tipos de funciones:

- ✓ Funciones de valores simples.
- ✓ Funciones de grupos de valores.
- ✓ Funciones de lista de valores.

■ Funciones de valores simples:

Son funciones sencillas que trabajan con valores simples: un número, una variable, una columna de una tabla.

- **ABS (n):** Valor Absoluto de N

```
SELECT ABS (-20) FROM DUAL;
Resultado: 20
```

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

```
SELECT EXP (2) FROM DUAL;
Resultado: 7,338
```

- **CEIL(n):** Valor entero inmediatamente superior o igual a “n”

```
SELECT CEIL (30.7) FROM DUAL;
Resultado: 31
```

- **FLOOR (n):** Valor entero inmediatamente inferior o igual a “n”

```
SELECT FLOOR (30.7) FROM DUAL;
Resultado: 30
```

- **MOD (m,n):** Resto de dividir “m” entre “n”.

```
SELECT MOD (20,3) FROM DUAL;
Resultado: 2
```

- **POWER (m, exponente):** Calcula la potencia de un número.

```
SELECT POWER (2,3) FROM DUAL;
Resultado: 8
```

- **ROUND (n [,m]):** Redondea "n" con el número de dígitos de precisión "m". Si m < 0 redondea los "m" dígitos a la izquierda del punto decimal.

```
SELECT ROUND (12.5874,3) FROM DUAL;
Resultado: 12.587
```

- **SIGN (n):** Signo del valor "n". (Si N < 0 → -1, Si N > 0 → 1, Si N = 0 → 0).

```
SELECT SIGN (-10) FROM DUAL;
Resultado: -1
```

- **TRUNC (n [,m]):** Trunca "n" para que tenga "m" dígitos decimales de precisión. Si m < 0 trunca los "m" dígitos a la izquierda del punto decimal.

```
SELECT TRUNC (1.546,2) FROM DUAL;
Resultado: 1.54
```

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

```
SELECT SQRT (36) FROM DUAL;
Resultado: 6
```

- **Funciones de grupos de valores.**

Son funciones que actúan sobre grupos de valores o columnas de una tabla, por ejemplo funciones estadísticas.

- **AVG (n):** Calcula el valor medio de "n" ignorando los valores nulos.

```
SELECT AVG (edad) FROM alumnos;
```

- **COUNT (*| expr):** Cuenta el número de veces que la expresión evalúa con datos no nulos. El "*" cuenta todas las filas. La función COUNT admite la cláusula DISTINCT.
  COUNT (* | [DISTINCT |ALL] EXPRESION) → Contaría filas no duplicadas.

```
SELECT COUNT (numeMatri) FROM alumnos;
```

- **MAX (expr):** Calcula el máximo valor de la expresión.

```
SELECT MAX (edad) FROM alumnos;
```

- **MIN (expr):** Calcula el mínimo valor de la expresión.

```
SELECT MIN (edad) FROM alumnos;
```

- **SUM (expr):** Obtiene la suma de los valores de la expresión.

```
SELECT SUM (unidades) FROM ventas;
```

- **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 menor valor de la lista.

- **GREATEST (valor1, valor2, …):** Obtiene el mayor valor de la lista.

```
SELECT nombreAl, GREASTEST (nota1, nota2, nota3) FROM alumnos;
```

- **LEAST (valor1, valor2,…..):** Obtiene el menor valor de la lista.

```
SELECT nombreAl, LEAST (nota1, nota2, nota3) FROM alumnos;
```

5.2.- Funciones de Cadena de Caracteres.

Trabajan con datos de tipo CHAR o VARCHAR2. Permiten manipular cadenas de letras u otros caracteres.

Se pueden dividir en dos tipos de funciones:

- ✓ Funciones que devuelven valores carácter.
- ✓ Funciones que devuelven valores numéricos.

- **Funciones que devuelven caracteres:**

- **CHR (n):** Devuelve el carácter cuyo valor binario es el equivalente a “n”.

```
SELECT CHR (65) FROM DUAL;
Resultado: A
```

- **CONCAT (cadena1, cadena2):** devuelve cadena1 concatenada con cadena2.

```
SELECT CONCAT ('Hola','mundo') FROM DUAL;
Resultado: Holamundo
```

- **LOWER (cadena):** Convierte la cadena a minúsculas.

```
SELECT LOWER ('HOLA') FROM DUAL;
Resultado: hola
```

- **UPPER (cadena):** Convierte la cadena a mayúsculas.

```
SELECT UPPER ('hola') FROM DUAL;
Resultado: HOLA
```

- **INITCAP (cadena):** Convierte la cadena a tipo título.

```
SELECT INITCAP ('hola mundo') FROM DUAL;
Resultado: Hola Mundo
```

- **LPAD (cadena1, n [, cadena2]):** Añade a la izquierda, los caracteres indicados en cadena2, hasta que cadena1 tenga "n" caracteres de longitud.

```
SELECT LPAD ('X',5,'*') FROM DUAL;
Resultado: ****X

SELECT LPAD ('X',10,'>+') FROM DUAL;
Resultado: >+>+>+>+>X
```

- **RPAD (cadena1, n [,cadena2]):** Añade a la derecha, los caracteres indicados en cadena2, hasta que cadena1 tenga "n" caracteres de longitud.

```
SELECT RPAD ('X',5,'*') FROM DUAL;
Resultado: X****

SELECT RPAD ('X',10,'>+') FROM DUAL;
Resultado: X>+>+>+>+>
```

- **LTRIM (cadena [, set]):** Suprime un conjunto de caracteres (set) a la izquierda de la cadena.

```
SELECT LTRIM ('  hola') FROM DUAL;
Resultado: hola

SELECT LTRIM ('hola', 'h') FROM DUAL;
Resultado: ola
```

- **RTRIM (cadena [, set]):** Elimina caracteres indicados por set a la derecha de la cadena.

```
SELECT RTRIM ('hola     ') FROM DUAL;
Resultado: hola

SELECT RTRIM ('hola', 'la')FROM DUAL;
Resultado: ho
```

- **REPLACE (cadena, cadBusqueda [cadSustitucion]):** Sustituye un carácter o caracteres de una cadena con 0 o más caracteres. Si no se pone cadSustitucuion se supone NULL.

```
SELECT REPLACE ('blanco negro','o','a') FROM DUAL;
Resultado: blanca negra

SELECT REPLACE ('blanco negro','o') FROM DUAL;
Resultado: blanc negr
```

- **SUBSTR (cad, inicio [, n]): Obtiene una subcadena de "n" caracteres desde la posición "inicio"** de la cadena. Si "inicio" es negativo la posición de inicio se calcula empezando desde la derecha hacia atrás.

```
SELECT SUBSTR ('ABCDEFG',3,2) FROM DUAL;
Resultado:  CD

SELECT SUBSTR ('ABCDEFG',-3,2) FROM DUAL;
Resultado: EF
```

- **TRANSLATE (cadena1, cadena2, cadena3):** Convierte los caracteres que se indican en cadena2 por el correspondiente carácter que aparece en la misma posición en cadena3 dentro de cadena1. Si hay menos caracteres en cadena3 que en cadena2 se completa con NULL.

```
SELECT TRANSLATE ('SQLPLUS','SQLU',123) FROM DUAL;
Resultado: 123P31
```

- **Funciones que devuelven valores numéricos:**

  - **ASCII(cadena):** Devuelve el valor ASCII de la primera letra de la cadena

    ```
    SELECT ASCII ('DAW') FROM DUAL;
    Resultado: 68
    ```

  - **INSTR (cadena1, cadena2 [, comienzo [, m]]):** Busca un conjunto de caracteres (cadena2) en la cadena (cadena1) y devuelve la posición de la ocurrencia "m". La búsqueda se realiza desde 'comienzo'. Si comienzo < 0 la posición de comienzo se calcula a partir del final de cadena1.

    ```
    SELECT INSTR ('VUELTA CICLISTA A ESPAÑA','TA',1,2) FROM DUAL;
    Resultado: 14
    ```

  - **LENGTH(cad):** Devuelve el número de caracteres de la cadena

    ```
    SELECT LENGTH ('HOLA MUNDO') FROM DUAL;
    Resultado: 10
    ```

5.3.- Funciones de manejo de fechas.

En los SGBD se utilizan mucho las fechas. Oracle tiene dos tipos de datos para manejar fechas, son **DATE** y **TIMESTAMP.**

- **DATE**: Almacena fechas concretas incluyendo a veces la hora.
- **TIMESTAMP**: Almacena un instante de tiempo más concreto que puede incluir hasta fracciones de segundo.

Podemos realizar operaciones numéricas con las fechas:

- Le podemos **sumar números** y esto se entiende como sumarles días, si ese número tiene decimales se suman días, horas, minutos y segundos.
- La **diferencia** entre dos fechas también nos dará un número de días.

En Oracle tenemos las siguientes funciones más comunes:

- **SYSDATE**: Devuelve la fecha y hora actuales. Fecha del sistema.

  ```
  SELECT SYSDATE FROM DUAL;
  ```

- **SYSTIMESTAM:** Devuelve la fecha y hora actuales en formato TIMESTAMP.

  ```
  SELECT SYSTIMESTAMP FROM DUAL;
  ```

- **ADD_MONTHS (fecha, n)**: Añade a la fecha el número de meses indicado con n.

```
SELECT ADD_MONTHS ('27/07/17', 5) FROM DUAL;
Resultado:  27/12/17
```

- **MONTHS_BETWEEN (fecha1, fecha2)**: Devuelve el número de meses que hay entre fecha1 y fecha2.

```
SELECT MONTHS_BETWEEN ('12/07/17','12/03/17') 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 ('27/07/17') FROM DUAL;
Resultado: 31/07/17
```

- **NEXT_DAY (fecha, d)**: Indica el día que corresponde si añadimos a la fecha el día d. El día devuelto puede ser texto ('Lunes', Martes', ..) o el número del día de la semana (1=lunes, 2=martes, ..) dependiendo de la configuración.

```
SELECT NEXT_DAY ('15/11/17','LUNES') FROM DUAL;
Resultado: 20/11/17
```

- **EXTRACT (valor FROM fecha)**: Extrae un valor de una fecha concreta. El valor puede ser Day, month, year, hours, etc.

```
SELECT EXTRACT (MONTH FROM SYSDATE) FROM DUAL;
Resultado: 12
```

En Oracle: Los operadores aritméticos "+" (más) y "-" (menos) pueden emplearse para las fechas.

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

5.4.- 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.

- **TO_NUMBER (cadena, ['formato']):** Convierte una cadena a tipo NUMBER según el formato especificado.

  La cadena debe de estar formada por números, el carácter decimal o el signo menos a la izquierda. No pueden existir espacios ni caracteres.

  Se suele utilizar para dar un formato concreto a los números. Los formatos que podemos utilizar son los siguientes:

SIMBOLO	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 (numero, formato):** Convierte un número a cadena de caracteres.

SIMBOLO	SIGNIFICADO
9	Devuelve el valor con el número especificado de dígitos
0	Muestra un 0 si el valor contiene 0 en dicha posición
$	Devuelve el valor con el signo $ a la izquierda
B	Muestra un espacio en blanco si el valor es 0
Mi	Si el número es negativo muestra el signo menos después del número
S	Visualiza el signo en la correspondiente posición
PR	Los número negativos se muestran entre <>
L	Visualiza el símbolo de moneda en la posición indicada
,	Devuelve la coma en la posición especificada
.	Devuelve el punto decimal en la posición especificada
V	Devuelve el valor multiplicado por 10^n , donde n es el número de 9 después la v
RN	Devuelve el valor en número romanos

- **TO_CHAR (fecha, formato):** Convierte una fecha a una cadena de caracteres

- **TO_DATE (cadena, 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:

SIMBOLO	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	Semestre
WW	Semana del año
AM	Indicador a.m.
PM	Indicador p.m.
HH12	Hora de 1 a 12
HH24	Hora de 0 a 23
MI	Minutos de 0 a 59
SS	Segundos dentro del minuto
SSSS	Segundos dentro desde las 0 horas

5.5.- Otras funciones: NVL y DECODE.

Función NVL.

¿Recuerdas que era el valor **NULL**? Cualquier columna de una tabla podía contener un valor nulo independientemente al tipo de datos que tuviera definido. Eso sí, esto no era así en los casos en que definíamos esa columna como no nula (**NOT NULL**), o que fuera clave primaria (**PRIMARY KEY**).

Cualquier operación que se haga con un valor **NULL** devuelve un **NULL**. Por ejemplo, si se intenta dividir por **NUL**L, 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.
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.

- **Función DECODE**

  La función DECODE nos permite evaluar expresiones

  - **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**.

6.- Consultas de resumen.

Seguro que alguna vez has necesitado realizar cálculos sobre un campo para obtener algún resultado global, por ejemplo, si tenemos una columna donde estamos guardando las notas que obtienen unos alumnos o alumnas en Matemáticas, podríamos estar interesados en saber cuál es la nota máxima que han obtenido o la nota media.

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)
```

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.

Las principales funciones de **agregado (o agrupamiento)** son las siguientes:

6.1.- Funciones de agregado: SUM y COUNT.

Sumar y contar filas o datos contenidos en los campos es algo bastante común.

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.

- **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**.

6.2.- Funciones de agregado: MIN y MAX.

- **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).
- **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).

6.3.- Funciones de agregado: AVG, VAR, STDEV.

Quizás queramos obtener datos estadísticos de los datos guardados en nuestra base de datos. Para ello podemos hacer uso de las 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.

7.- 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 cómo 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.
En todos estos casos en lugar de una única fila de resultados necesitaremos una fila por cada actividad, cada empleado, 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:

- ✓ **WHERE** que selecciona las filas según las condiciones que pongamos.
- ✓ **GROUP BY** que agrupa estas filas.
- ✓ **HAVING** filtra los grupos. Selecciona y elimina
- ✓ **ORDER BY** clasifica la salida. que ordena los grupos.

Las columnas que aparecen en la **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**.

8.- 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. Así evitábamos repetir datos. Por tanto, también será frecuente que tengamos que consultar datos que se encuentren distribuidos por distintas tablas.

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. La razón de esta nueva sintaxis era separar las condiciones de asociación respecto a las condiciones de selección de registros.

8.1.- Composiciones internas.

¿Qué ocurre si combinamos dos o más tablas sin ninguna restricción? El resultado será un producto cartesiano.
El producto cartesiano entre dos tablas da como resultado todas las combinaciones de todas las filas de esas dos tablas.
Se indica poniendo en la cláusula FROM las tablas que queremos componer separadas por comas. Y puedes obtener el producto cartesiano de las tablas que quieras.

```
SELECT * FROM tabla1, tabla2;
```

Como lo que se obtiene son todas las posibles combinaciones de filas, debes tener especial cuidado con las tablas que combinas. Si tienes dos tablas de 10 filas cada una, el resultado tendrá 10x10 filas, a medida que aumentemos el número de filas que contienen las tablas, mayor será el resultado final, con lo cual se puede considerar que nos encontraremos con una operación costosa.

Esta operación no es de las más utilizadas ya que coge una fila de una tabla y la asocia con todos y cada uno de las filas de la otra tabla, independientemente de que tengan relación o no. Lo más normal es que queramos seleccionar los registros según algún criterio.

Necesitaremos discriminar de alguna forma para que únicamente aparezcan filas de una tabla que estén relacionadas con la otra tabla. A esto se le llama asociar tablas (**JOIN)**.

Para hacer una composición interna se parte de un producto cartesiano y se eliminan aquellas filas que no cumplen la condición de composición.
Lo importante 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.

Las columnas que aparecen en la cláusula **WHERE** se denominan **columnas de emparejamiento** ya que son las que permiten emparejar las filas de las dos tablas. Éstas no tienen por qué estar incluidas en la lista de selección.

Podemos encontrar los siguientes casos:

- Si en la relación de tablas **una de las columnas es clave principal y esa columna aparece en la otra tabla**, es más eficiente utilizar otro tipo de composición, el **INNER JOIN** y su resultado sería el mismo que el producto cartesiano sobre dicha columna

```
SELECT    FROM tabla1  NATURAL [INNER] JOIN tabla2;
```

Es similar a:

```
SELECT  FROM  tabla1, tabla2
WHERE tabal1.comluna = tabla2.columna;
```

Utilizando esta notación, las columnas comunes solo aparecen una vez.

- Si en la relación de tablas **una de las columnas es clave principal y las columnas relacionadas tienen el mismo nombre**, otra formar sería utilizar el **USING.**

```
SELECT …. FROM tabla1 [INNER]
        JOIN tabla2 USING (campo1)
                JOIN tabla 3 USING (campo 2) …
                        WHERE…
```

Las columnas relacionadas tienen el mismo nombre.

- Si en la relación de tablas **una de las columnas es clave principal, pero las columnas relacionadas no tienen el mismo nombre.**

```
SELECT …. FROM tabla1
        [INNER] JOIN tabla2  ON (tabla1.columna = tabla2.columna)
                [INNER] JOIN tabla3  ON (tabla2.columna = tabla3.columna)…
                        WHERE…..
```

- 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.

8. 2.- Composiciones externas.

A veces es necesario seleccionar algunas filas de una tabla aunque éstas no tengan correspondencia con las filas de la otra tabla.

A esta composición se le llama **OUTER JOINS (+). Left Joins / Rigth Joins.** Permite seleccionar filas de una tabla resultante de una combinación aunque éstas no tengan correspondencia. Dependiendo donde se encuentre el **Outer Join** se denominará **Left Join** o **Rigth join**

¿Cómo es el formato? Muy sencillo, añadiremos un signo más entre paréntesis **(+)** en la igualdad entre campos que ponemos en la cláusula **WHERE**. El **carácter (+)** irá detrás del nombre de la tabla en la que deseamos aceptar valores nulos.

```
SELECT tabla1.columna1, tabla2.columna2, tabla2.columna1, tabla2.columna2
FROM tabla1, tabla2
WHERE tabla1.columna1= tabla2.columna1 (+);
```

Seleccionará todas las filas de la tabla1, aunque no tengan correspondencia con las filas de la tabla 2. El resto de las columnas de la tabla2 se rellena con NULL.

El equivalente ANSI a la notación + y pueden ser de tres tipos

- **LEFT OUTER JOIN:** es una composición externa izquierda, todas las filas de la tabla de la izquierda se devuelven, aunque no haya ninguna columna correspondiente en las tablas combinadas.

```
SELECT … FROM tabla1
    LEFT OUTER JOIN tabla2 ON tabla1.columna = tabla2.columna;

Equivale a:

SELECT … FROM tabla1, tabla2
    WHERE  tabla1.columna = tabla2.columna (+);
```

- **RIGTH OUTER JOIN:** es una composición externa derecha, todas las filas de la tabla de la derecha se devuelven, aunque no haya ninguna columna correspondiente en las tablas combinadas.

```
SELECT … FROM tabla1
    RIGHT OUTER JOIN tabla2 ON  tabla1.columna = tabla2.columna;

Equivale a:

SELECT … FROM tabla1, tabla2
WHERE  tabla1.columna (+) =  tabla2.columna
```

- **FULL OUTER JOIN:** es una composición externa en la que se devolverán todas las filas de los campos no relacionados de ambas tablas.

```
SELECT … FROM tabla1
    FULL OUTER JOIN tabla2 ON tabla1.columna = tabla2.columna

Equivale a:

SELECT … FROM tabla1, tabla2
    WHERE tabla1.columna  =  tabla2.columna (+);
```

9.- Otras consultas multitablas: unión, Intersección y diferencia de consultas.

Una consulta da como resultado un conjunto de filas y con conjuntos podemos hacer entre otras, tres tipos de operaciones comunes como son: **unión, intersección y diferencia**.

- **UNION:** combina las filas de un primer SELECT con las filas de otro SELECT, desapareciendo las filas duplicadas.

```
SELECT col1, col2,.... FROM tabla1  WHERE condicion
UNION  [ALL]
SELECT col1, col2,.... FROM tabla2  WHERE condicion;

Si se utiliza la opción ALL, aparecen las filas duplicadas.
```

- **INTERSECT:** examina las filas de dos SELECT y devolverá aquellas que aparezcan en ambos conjuntos. Las filas duplicadas se eliminarán.

```
SELECT col1, col2,.... FROM tabla1  WHERE condicion
INTERSECT
SELECT col1, col2,.... FROM tabla2  WHERE condicion;
```

- **MINUS:** devuelve aquellas filas que están en el primer SELECT pero no en el segundo. Las filas duplicadas del primer SELECT se reducirán a una fila única antes de comenzar la comparación.

```
SELECT col1, col2,.... FROM tabla1  WHERE condicion
MINUS
SELECT col1, col2,.... FROM tabla2  WHERE condicion;
```

Para estas tres operaciones es muy importante que sigas las siguientes reglas:

- Las columnas de las dos consultas se relacionan en orden, de izquierda a derecha.
- Los nombres de las columnas de la primera sentencia SELECT no tienen por qué ser los mismos que los nombres de la segunda.
- Las SELECT necesitan tener el mismo número de columnas.
- Los tipos de datos deben de coincidir, aunque la longitud no tiene que ser la misma.
- Estas operaciones se pueden combinar anidadas, pero es conveniente utilizar paréntesis para indicar que operación quieres que se haga primero. Si no hay paréntesis se evalúan de izquierda a derecha.

10.- Subconsultas.

Una subconsulta en SQL consiste en utilizar los resultados de una consulta dentro de otra, que se considera la principal. Esta posibilidad fue la razón original para la palabra "estructurada" que da el nombre al SQL de Lenguaje de Consultas Estructuradas (Structured Query Language).

Hemos utilizado la cláusula WHERE para seleccionar los datos que deseábamos comparando un valor de una columna con una constante, o un grupo de ellas. Si los valores de dichas constantes son desconocidos, normalmente por proceder de la aplicación de funciones a determinadas columnas de la tabla, tendremos que utilizar subconsultas.

Su Sintaxis o formato es:

```
SELECT …. FROM …WHERE columna operador
    (SELECT  [ALL/DISTINCT] ExpresionColumna [,ExpresionColumna …..]
    FROM NombreTabla [, NombreTabla ]
    [WHERE CondicionSeleccion]
    [GROUP BY ExpresionColumnaAgrupacion  [, ExpresionColumnaAgrupacion … ]
    [HAVING CondicionSeleciónGrupos  ] ] )
```

La subconsulta puede ir dentro de las cláusulas WHERE, HAVING o FROM.

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

- **Subconsultas que generan valores simples.**
  Las subconsultas que se utilizan los operadores **>, <, >=, <=, !=, = o IN.** Devuelven un único valor, si la subconsulta devolviera más de un valor devolvería un error.

  Los **tipos de datos** que devuelve la subconsulta y la columna con la que se compara ha de ser el mismo.

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

  - **ANY:** Compara con cualquier fila de la consulta. La instrucción es válida si hay un registro en la subconsulta que permite que la comparación sea cierta.
  - **ALL:** Compara con todas las filas de la consulta. La instrucción resultará cierta si es cierta toda la comparación con los registros de la subconsulta.

- **IN:** No utiliza comparador, lo que hace es comprobar si el valor se encuentra en el resultado de la subconsulta.
- **NOT IN:** Comprueba si un valor no se encuentra en una subconsulta.

- **Subconsultas en la selección de grupos.**

  Aunque las subconsultas suelen encontrarse sobre todo en la cláusula **WHERE**, también pueden usarse en la **HAVING** formando parte de la selección del grupo de filas efectuada por dicha cláusula.

  Todos los formatos son exactamente iguales a cuando son utilizados en la cláusula **WHERE**.

- **Subconsultas anidadas.**

  Cuando una subconsulta forma parte de una condición de selección en una cláusula **WHERE** o **HAVING** de otra subconsulta se dice que es una subconsulta anidada.

  Las subconsultas se pueden anidar en varias cláusulas a la vez y en varios niveles. Esta posibilidad de anidamiento es lo que le da potencia a la instrucción select.

- **Subconsultas correlacionadas**

  Las subconsultas correlacionadas hacen un proceso fila a fila, de modo que la subconsulta se ejecuta una vez por cada fila de la consulta principal. Esto es absolutamente diferente respecto a la ejecución normal de una subconsulta, ya que normalmente la subconsulta clásica se ejecuta primero, y con sus resultados se ejecuta la consulta principal,

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