UD.4 Realización de consultas

1.- INTRODUCCIÓN.

En unidades anteriores has aprendido que SQL es un conjunto de sentencias u órdenes que se necesitan para acceder a los datos. Este lenguaje es utilizado por la mayoría de las aplicaciones donde se trabaja con datos para acceder a ellos, crearlos, y actualizarlos. 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 Corporation.

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, eso sí, cada fabricante añade sus mejoras al lenguaje SQL.

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

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

Las sentencias SQL que se verán a continuación pueden ser escritas y ejecutadas de dos formas:

- **Desde el entorno gráfico con SQLDeveloper** que ya instalaste en unidades anteriores. Accede a SQLDeveloper ejecutando el archivo sqldeveloper.exe. Si no tienes un acceso directo en el escritorio, y no sabes en qué carpeta está, escribe desde el menú de inicio de windows sqldeveloper.exe. Ejecútalo con permiso de Administrador pulsando el botón derecho del ratón y

eligiendo esa opción. Abre la conexión para el usuario que creaste y tendrás acceso al editor de SQL para escribir y ejecutar las sentencias pulsando el botón (flecha verde) que aparece en la imagen o con la combinación de teclas.

- **Desde el entorno de SQL*Plus** con el intérprete de comandos de SQL que ofrece Oracle y que puedes encontrar desde el menu de windows: Inicio > Oracle-OraDB18Home1-> SQL Plus.

Para ejecutar cualquiera de las sentencias SQL que aprenderás en los siguientes puntos, simplemente debes escribirla completa, con la misma sintaxis, finalizando con el caracter punto y coma. (;) y pulsar Intro para que se inicie su ejecución.Si optas por trabajar con SQLPlus , 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. Como muestra la imagen te pide el nombre usuario y la contraseña. Escribe el usuario que creaste en las unidades anteriores. No olvides que en esta versión el nombre de usuario va precedido de c##.

La descripción y contenido de las tablas a las que se hace referencia en los ejemplos de las sentencias a lo largo de la unidad, así como la descripción del Sistema de Información y el DER, las puedes encontrar, al final del Contenido, en el Anexo I.- Base de datos de ejemplo (Juegos online). También encontrarás las sentencias de creación de las tablas y.algunos registros de ejemplo con las sentencias de inserción correspondientes. Todo esto te ayudará a entender y comprobar algunos de los ejemplos que van a aparecer a partir de ahora. Puedes seguir los pasos explicados en el Anexo II para crear el usuario y las tablas en tu ordenador

Ten en cuenta que los campos que se definen pueden ir variando a lo largo de esta y las siguientes unidades ya que se van adaptando a las necesidades de la teoría en la que se presentan. Es por esto que por ejemplo, el tipo de datos que se incluye y su tamaño pueden variar del que aparece en el documento que se anexa.

Conviene que las tengas a mano. Observa el resultado de pasar del DER al modelo relacional e identifica las claves primarias y ajenas en cada una de las tablas.

Además, durante la unidad se irán presentando ejercicios en el apartado Ejercicios Resueltos para que practiques lo que se expone en cada uno de los apartados de la teoría. Esa es la forma de aprender. Para poder realizarlos vamos a crear las tablas e insertar algunos datos para probar las distintas consultas que crees. A partir de ahora nos referiremos a estos datos como tablas de la aplicación empresa.

En el Anexo II.- Creación y carga de tablas de la aplicación empresa en Oracle tienes el script que lo realiza y los pasos a seguir para ejecutarlo.

2.- LA SENTENCIA SELECT.

¿Cómo podemos seleccionar los datos que nos interesen dentro de una base de datos? Para recuperar o seleccionar los datos, de una o varias tablas puedes valerte del lenguaje SQL, para ello utilizarás la sentencia SELECT, que 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 separadas por comas simples (" , "). Esta parte es obligatoria.
- **Cláusula FROM** seguida del nombre de la tabla o tablas de las que proceden las columnas de arriba, es decir, de donde vas a extraer los datos. Esta parte también es obligatoria.
- **Cláusula WHERE** seguida de un criterio de selección o condición. Esta parte es opcional.
- **Cláusula ORDER BY** seguida por un criterio de ordenación. Esta parte también es opcional.

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

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

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 que otras.

En la especificación de los formatos o sintaxis de las sentencias de cualquier lenguaje de programación hay caracteres que tienen un significado especial y no se escriben en el uso de la sentencia:

- Los corchetes [ ] especifican opcionalidad.
- La barra vertical | indica que has de elegir uno de los elementos que relaciona.
- Los puntos .... indican que puede haber más elementos de ese tipo, es decir que puedes incluir más columnas, más tablas y más condiciones.

Así en el formato básico de la sentencia SELECT

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

indica que de forma opcional puedes escribir la cláusula ALL o bien DISTINCT (nunca las dos a la vez) y que se pueden incluir más columnas, tablas y condiciones.

2.1.- Claúsula SELECT.

Ya has visto que a continuación de la sentencia SELECT debemos especificar cada una de las columnas que queremos seleccionar. Además, debemos tener en cuenta lo siguiente:

- **Se pueden nombrar** (o calificar) a las columnas anteponiendo el nombre de la tabla de la que proceden, pero esto es opcional solo si no existe el mismo nombre de columna en dos tablas Se realiza anteponiendo el nombre de la tabla a la que pertenece la columna seguida por un punto, es decir, NombreTabla.NombreColumna.
- Si queremos **incluir todas las columnas** de una tabla podemos utilizar el **comodín asterisco** ("*"). Quedaría así: SELECT * FROM NombreTabla;
- 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:

```
SELECT F_Nacimiento "Fecha de Nacimiento" FROM USUARIOS;
```

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

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

2.2.- Claúsula FROM.

Al realizar la consulta o selección has visto que puedes elegir las columnas que necesites, pero ¿de dónde extraigo 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 añadir el nombre del usuario que es propietario de esas tablas, indicándolo de la siguiente manera:

USUARIO.TABLA

De este modo podemos distinguir entre las tablas de un usuario y otro (ya que esas tablas pueden tener el mismo nombre).

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

Pongamos varios ejemplos:

```
SELECT * FROM USUARIOS U;

SELECT LOGIN,NOMBRE,APELLIDOS FROM USUARIOS;
```

En los dos ejemplos devuelve todas las filas de la tabla USUARIOS. En el primero muestra todas las columnas y en el segundo solo las columnas LOGIN,NOMBRE y APELLIDOS

2.3.- Claúsula WHERE.

¿Podríamos desear seleccionar los datos de una tabla que cumplan una determinada condición? 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.

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 te mostraremos los operadores con los que podrás crear condiciones de diverso tipo.

2.4.- Ordenación de registros. Cláusula ORDER BY.

En la consulta del ejemplo anterior hemos obtenido una lista de los 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. Tendríamos:

```
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.

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.

En el siguiente ejemplo, ordenamos por apellidos y, en caso de que sean iguales, 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.

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:

1. Relacionales o de comparación.
2. Aritméticos.
3. De concatenación.
4. Lógicos.

¿Cómo se utilizan y para qué sirven? En los siguientes apartados responderemos a estas cuestiones.

3.1.- Operadores de comparación.

Llamados operadores relacionales en informática, 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 y su operación:

Operadores y su significado.	
OPERADOR	**SIGNIFICADO**
=	Igualdad.
!=, < >, ^ =	Desigualdad. Distinto
<	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** significaba 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 validos 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;
```

Ahora queremos aquellos empleados cuyo apellido comienza por R:

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

3.2.- 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.

Mira este ejemplo sobre una tabla TRABAJADORES en el que obtenemos el salario aumentado en un 5 % de aquellos trabajadores que cobran 1000 € o menos.

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

Cuando una expresión aritmética se calcula sobre valores NULL, el resultado es el propio valor NULL.
Para concatenar cadenas de caracteres existe el operador de concatenación (" || "). Oracle puede convertir automáticamente valores numéricos a cadenas para una concatenación.
En la tabla EMPLEADOS 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 EMPLEADOS;
```

Podemos poner el alias Apellidos al resultado de la concatenación, para que se utilice como cabecera en la salida. 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 Apellidos
FROM EMPLEADOS;
```

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.

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 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.

Fíjate en los siguientes ejemplos:

Si 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;
```

3.4.- Precedencia.

Con frecuencia utilizaremos la sentencia SELECT acompañada de expresiones muy extensas y resultará difícil saber qué parte de dicha expresión se evaluará primero, por ello es conveniente conocer el orden de precedencia que tiene Oracle. En casos de igualdad se evalúa de izquierda a derecha.

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.
7. AND.
8. 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:

```
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;
```

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,
- de cadena de caracteres,
- de manejo de fechas,
- de conversión,

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.

5.1.- Funciones numéricas.

¿Cómo obtenemos el cuadrado de un número o su valor absoluto? Nos referimos a valores numéricos y por tanto necesitaremos utilizar funciones numéricas.

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

- **ABS(n):** Calcula el valor absoluto de un número n.
  Ejemplo:
  ```
  SELECT ABS(-17) FROM DUAL; -- Resultado: 17
  ```
- **EXP(n):** Calcula en , es decir, el exponente en base e del número n.
  Ejemplo:
  ```
  SELECT EXP(2) FROM DUAL; -- Resultado: 7,38
  ```
- **FLOOR(n):** Calcula el valor entero inmediatamente inferior o igual al parámetro n.
  Ejemplo:
  ```
  SELECT FLOOR(17.4) FROM DUAL; -- Resultado: 17
  ```
- **MOD(m,n):** Calcula el resto resultante de dividir m entre n.
  Ejemplo:
  ```
  SELECT MOD(15, 2) FROM DUAL; --Resultado: 1
  ```
- **POWER(valor, exponente):** Eleva el valor al exponente indicado.
  Ejemplo:
  ```
  SELECT POWER(4, 5) FROM DUAL; -- Resultado: 1024
  ```
- **ROUND(n, decimales):** Redondea el número n al siguiente número con el número de decimales que se indican.
  Ejemplo:
  ```
  SELECT ROUND(12.5874, 2) FROM DUAL; -- Resultado: 12.59
  ```
- **SQRT(n):** Calcula la raíz cuadrada de n.
  Ejemplo:
  ```
  SELECT SQRT(25) FROM DUAL; --Resultado: 5
  ```
- **TRUNC(m,n):** Trunca un número a la cantidad de decimales especificada por el segundo argumento. Si se omite el segundo argumento, se truncan todos los decimales. Si "n" es negativo, el número es truncado desde la parte entera.
  Ejemplos:
  ```
  SELECT TRUNC(127.4567, 2) FROM DUAL; -- Resultado: 127.45
  SELECT TRUNC(4572.5678, -2) FROM DUAL; -- Resultado: 4500
  SELECT TRUNC(4572.5678, -1) FROM DUAL; -- Resultado: 4570
  ```

```
SELECT TRUNC(4572.5678) FROM DUAL; -- Resultado: 4572
```

- **SIGN(n):** Si el argumento "n"es un valor positivo, retorna 1, si es negativo, devuelve -1 y 0 si es 0.

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

5.2.- Funciones de cadena de caracteres.

Ya verás como es muy común manipular campos de tipo carácter o cadena de caracteres. Como resultado podremos obtener caracteres o números. Estas son las funciones más habituales:

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

  Ejemplo:

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

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

  Ejemplo:

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

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

  Ejemplo:

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

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

  Ejemplo:

```
SELECT LOWER('En MInúsculAS') FROM DUAL; --Resultado: en minúscula.
```

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

  Ejemplo:

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

  Esta función y la siguiente son muy utilizadas, y necesarias, al comparar con cadenas cuando hay dudas sobre si todos los caracteres de la cadena a comparar están en mayúsculas, minúsculas o mezcla. Por ejemplo si queremos conocer los datos de la tabla JUEGOS cuyo nombre es AJEDREZ podemos utilizar cualquiera de las dos sentencias siguientes. Así se seleccionará la fila, tanto si está escrito en la tabla como AJEDREZ, ajedrez o Ajedrez.

```
SELECT * FROM JUEGOS WHERE UPPER(NOMBRE)='AJEDREZ';
SELECT * FROM JUEGOS WHERE LOWER(NOMBRE)='ajedrez';
```

- **INITCAP(cad):** Devuelve la cadena cad con su primer carácter en mayúscula.

  Ejemplo:

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

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

  Ejemplo:

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

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

  Ejemplo:

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

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

  Ejemplo:

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

  Resultado: correo@gmail.com

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

  Ejemplo:

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

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

  Ejemplo:

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

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

  Ejemplo:

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

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

  Ejemplo:

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

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

  Ejemplo:

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

- **INSTR(cad, cadBuscada [, posInicial [, nAparición]]):** Obtiene la posición en la que se encuentra la cadena buscada en la cadena inicial cad. Se puede comenzar a buscar desde una posición inicial

concreta e incluso indicar el número de aparición de la cadena buscada. Si no encuentra nada devuelve cero.

Ejemplo:

```
SELECT INSTR('usuarios', 'u') FROM DUAL; --Resultado: 1
SELECT INSTR('usuarios', 'u', 2) FROM DUAL; --Resultado: 3
SELECT INSTR('usuarios', 'u', 2, 2) FROM DUAL; --Resultado: 0
```

5.3.- Funciones de manejo de fechas.

La fecha de emisión de una factura, de llegada de un avión, de ingreso en una web, podríamos seguir poniendo infinidad de ejemplos, lo que significa que es una información que se requiere en muchas situaciones y es importante guardar.

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. El resultado es una fecha.
- Le podemos restar números y esto se entiende como restarle días, ir atrás en el calendario, si ese número tiene decimales se restan días, horas, minutos y segundos. El resultado es una fecha.
- La diferencia o resta entre dos fechas nos dará el número de días entre esas fechas.

En Oracle tenemos las siguientes funciones más comunes:

- **SYSDATE**: Devuelve la fecha y hora actuales. Ejemplo:

```
SELECT SYSDATE FROM DUAL; --Resultado: 15/08/20
```

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

```
SELECT SYSTIMESTAMP FROM DUAL; --Resultado: 15/08/20 11:40:41,969000 +02:00
```

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

```
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. Ejemplo:

```
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. Ejemplo:

```
SELECT LAST_DAY('27/07/11') FROM DUAL; --Resultado: 31/07/11
```

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

```
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. Ejemplo:

```
SELECT EXTRACT(MONTH FROM SYSDATE) FROM DUAL; --Resultado: 8
```

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

```
SELECT SYSDATE - 5 FROM DUAL; -- Devuelve la fecha correspondiente a 5 días antes de la fecha actual
```

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

5.3.- Funciones de conversión.

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

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

Formatos para números y su significado.

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

- **TO_CHAR (d, formato):** Convierte un número o fecha d a cadena de caracteres, se utiliza normalmente para fechas ya que de número a texto se hace de forma implícita como hemos visto antes.

- **TO_DATE (cad, formato):** Convierte textos a fechas. Podemos indicar el formato con el que queremos que aparezca.

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

Formatos para fechas y su significado.

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

5.3.- Otras funciones: NVL y DECODE.

¿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 NULL, no nos aparecerá ningún error sino que como resultado obtendremos un NULL (no se producirá ningún error tal y como puede suceder si intentáramos dividir por cero).

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

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

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

¿Y no habrá alguna función que nos permita evaluar expresiones? La respuesta es afirmativa y esa función se llama DECODE.

- **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.

Por ejemplo, si en la tabla USUARIOS queremos un listado de sus correos, podemos pedir que cuando un correo esté a nulo, es decir, no tenga valor, aparezca el texto No tiene correo.

```
SELECT NVL(correo, 'No tiene correo') FROM USUARIOS;
```

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 cual 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) y debemos tener en cuenta que:

- La palabra **ALL** indica que se tienen que tomar todos los valores de la columna. Es el valor por defecto.
- La palabra **DISTINCT** indica que se considerarán todas las repeticiones del mismo valor como uno solo (considera valores distintos).

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

Ya estamos preparados para conocer cuáles son estas funciones de agregado (o agrupamiento). Las veremos a continuación.

6.1.- Funciones de agregado: SUM y COUNT.

Sumar y contar filas o datos contenidos en los campos es algo bastante común. Imagina que para nuestra tabla Usuarios necesitamos sumar el número de créditos total que tienen nuestros jugadores. Con una función que sumara los valores de la columna crédito sería suficiente, siempre y cuando lo agrupáramos por cliente, ya que de lo contrario lo que obtendríamos sería el total de todos los clientes jugadores.

- **La función SUM.** `SUM([ALL|DISTINCT] expresión)`

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

  ```
  SELECT SUM( credito) FROM Usuarios;
  ```

- **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, una función o el caracter * en cuyo caso contaría el número de filas que cumplen la condición especificada, si la hay. Puede contar cualquier tipo de datos incluido texto. COUNT simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan. La función COUNT no cuenta los registros que tienen campos NULL a menos que expresión sea el carácter comodín asterisco (*).

  Si utilizamos COUNT(*), calcularemos el total de filas, incluyendo aquellas que contienen valores NULL. Por ejemplo,

  ```
  SELECT COUNT(nombre) FROM Usuarios;
  SELECT COUNT(*) FROM Usuarios;
  ```

6.2.- Funciones de agregado: MIN y MAX.

¿Y si pudiéramos encontrar el valor máximo y mínimo de una lista enormemente grande? Esto es lo que nos permiten hacer las siguientes funciones.

- **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). Por ejemplo para obtener el valor más bajo de la columna credito.

```
SELECT MIN(credito) FROM Usuarios;
```

- **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). Por ejemplo para obtener el valor más alto de la columna credito.

```
SELECT MAX (credito) FROM Usuarios;
```

6.2.- Funciones de agregado: AVG, VAR y 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 o media 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. Ejemplo que obtiene la media de credito de los usuarios

```
SELECT AVG(CREDITO) FROM USUARIOS;
```

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

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

En todos estos casos en lugar de una única fila de resultados necesitaremos una fila por cada actividad, cada cliente, etc. Podemos obtener estos subtotales utilizando la cláusula GROUP BY. También podemos poner condiciones a esos grupos con la cláusula HAVING.

La sintaxis es la siguiente:

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

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

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

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

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

Veamos un par de ejemplos:

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

Obtenemos la suma de créditos de nuestros usuarios agrupados por provincia. Si estuviéramos interesados en la suma de créditos agrupados por provincia pero únicamente de las provincias de Sevilla y Badajoz nos quedaría:

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

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.

Disponemos de una tabla USUARIOS cuya clave principal es Login y esta tabla a su vez está relacionada con la tabla PARTIDAS a través del campo Cod_Creador_partida. Si quisiéramos obtener el nombre de los usuarios y las horas de las partidas de cada jugador necesitaríamos coger datos de ambas tablas pués las horas se guardan en la tabla PARTIDAS. Esto significa que cogeremos filas de una y de otra.

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

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

- La composición interna.
- La composición externa.

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

La sintaxis es la siguiente:

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

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.

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 calificarse o identificarse especificando la tabla de procedencia seguida de un punto o utilizando un alias de tabla.

Las columnas relacionadas que aparecen en la cláusula **WHERE** se denominan **columnas de join o emparejamiento** ya que son las que permiten emparejar las filas de las dos tablas. Éstas no tienen por qué estar incluidas en la lista de selección. Emparejaremos tablas que estén relacionadas entre sí siendo usualmente las columnas de emparejamiento la clave principal y la clave ajena. Cuando emparejamos campos debemos especificarlo de la siguiente forma: NombreTabla1. Camporelacionado1 = NombreTabla2.Camporelacionado2.

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.

Veamos un ejemplo, si queremos obtener el historial laboral de los empleados incluyendo nombres y apellidos de los empleados, la fecha en que entraron a trabajar y la fecha de fin de trabajo si ya no continúan en la empresa, tendremos:

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

Vamos a obtener el historial con los nombres de departamento, nombre y apellidos del empleado de todos los departamentos:

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

8.2.- Composiciones externas.

¿Has pensado que puede que te interese seleccionar algunas filas de una tabla aunque éstas no tengan correspondencia con las filas de la otra tabla? Esto puede ser necesario.

Imagina que tenemos en una base de datos guardadas en dos tablas la información de los empleados de la empresa (Cod_empleado, Nombre, Apellidos, salario y Cod_dpto) por otro lado los departamentos (Codigo_dep, Nombre) de esa empresa. Recientemente se ha remodelado la empresa y se han creado un par de departamentos más pero no se les ha asignado los empleados. Si tuviéramos que obtener un informe con los datos de los empleados por departamento, seguro que deben aparecer esos departamentos aunque no tengan empleados. Para poder hacer esta combinación usaremos las composiciones externas.

¿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.

En nuestro ejemplo, la igualdad que tenemos en la cláusula WHERE es Cod_dpto (+)= Codigo_dep ya que es en la tabla empleados donde aparecerán valores nulos.

8.3.- Composiciones en la versión SQL99.

Como has visto, SQL incluye en esta versión mejoras de la sintaxis a la hora de crear composiciones en consultas. Recuerda que la sintaxis es la siguiente:

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

- **CROSS JOIN:** creará un producto cartesiano de las filas de ambas tablas por lo que podemos olvidarnos de la cláusula WHERE.

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

Podríamos transformar algunas de las consultas con las que hemos estado trabajando:

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

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

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

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

9.- OTRAS CONSULTAS MULTITABLAS: UNIÓN, INTERSECCIÓN Y DIFERENCIA DE CONSULTAS .

Seguro que cuando empieces a trabajar con bases de datos llegará un momento en que dispongas de varias tablas con los mismos datos guardados para distintos registros y quieras unirla en una única tabla. ¿Esto se puede hacer? Es una operación muy común junto a otras. Al fin y al cabo, 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.
- **INTERSECT:** examina las filas de dos SELECT y devolverá aquellas que aparezcan en ambos conjuntos. Las filas duplicadas se eliminarán.
- **MINUS:** devuelve aquellas filas que están en el primer SELECT pero no en el segundo. Las filas duplicadas del primer SELECT se reducirán a una antes de comenzar la comparación.

Para estas tres operaciones es muy importante que utilices en los dos SELECT el mismo número y tipo de columnas y en el mismo orden.

Estas operaciones se pueden combinar anidadas, pero es conveniente utilizar paréntesis para indicar que operación quieres que se haga primero.

Veamos un ejemplo de cada una de ellas.

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

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

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

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

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

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

8.- SUBCONSULTAS.

A veces tendrás que utilizar en una consulta los resultados de otra que llamaremos subconsulta o consulta subordinada. La sintaxis es:

```
SELECT listaExpr
                FROM tabla
                WHERE expresión_o_columna OPERADOR
                ( SELECT expresion_o_columna
                FROM tabla);
```

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

Dependiendo de los operadores utilizados las subconsultas pueden devolver 1 o varias filas:

- El **OPERADOR** puede ser >, <, >=, <=, !=, = o IN. Las subconsultas que se utilizan con estos operadores devuelven un único valor. Si la subconsulta devolviera más de un valor devolvería un error.Como puedes ver en la sintaxis, las subconsultas deben ir entre paréntesis y a la derecha del operador.

  Pongamos un ejemplo:

```
SELECT Nombre, salario
                    FROM EMPLEADOS
                    WHERE salario <
                    (SELECT salario FROM EMPLEADOS
                    WHERE Nombre= 'Ana');
```

Obtendríamos el nombre de los empleados y el sueldo de aquellos que cobran menos que Ana. Si hubiese más de un empleado que se llamase Ana esta consulta daría error ya que la subconsulta devolvería más de una fila.

El tipo de dato que devuelve la subconsulta y la columna con la que se compara ha de ser el mismo.

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

- Cuando el resultado de la subconsulta es **más de una fila**, SQL utiliza palabras reservadas entre el operador y la consulta. Estas 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 la comparación con todas las filas devuelas por 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.

En la siguiente consulta obtenemos el empleado que menos cobra:

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

Esa misma consulta se podría haber realizado utilizando la función de agregado o colectiva MIN de la siguiente forma:

```
SELECT nombre, salario
FROM EMPLEADOS
WHERE salario =  (SELECT MIN(salario) FROM EMPLEADOS);
```

Anexo I.- Base de datos de ejemplo (Juegos online).

Anexo II.- Creación y carga de tablas de la aplicación empresa en Oracle.

Anexo III. Ejercicios SQL propuestos con posible solución