Realización de consultas.

Caso práctico

Ministerio de Educación (Uso educativo nc)

Una de las cosas más importantes que ofrece una base de datos es la opción de poder consultar, de múltiples formas, los datos que guarda, por eso Ana y Juan van a intentar sacar el máximo partido a las tablas que han guardado y sobre ellas van a obtener toda aquella información que su cliente les ha solicitado. Sabemos que dependiendo de quién consulte la base de datos, se debe ofrecer un tipo de información u otra. Es por esto que deben crear distintas consultas y vistas.

Ana sabe que existen muchos tipos de operadores con los que puede "jugar" para crear consultas y también tiene la posibilidad de crear campos nuevos donde podrán hacer cálculos e incluso trabajar con varias tablas relacionadas a la vez.

Actualmente están con una base de datos en la que se ha almacenado información sobre los **empleados** de la empresa que tiene la página de juegos online, los **departamentos** en los que trabajan y los **estudios** de sus empleados. Se está guardando el **historial laboral y salarial** de todos los empleados. Ya que tienen una base de datos para sus clientes, han visto que también sería conveniente tener registrada esta otra información interna de la empresa.

De este modo pueden llevar un control más exhaustivo de sus empleados, salario y especialización. Podrán conocer cuánto pagan en sueldos, que departamento es el que posee mayor número de empleados, el salario medio, etc. Para obtener esta información necesitarán consultar la base utilizando principalmente el comando `SELECT`.

1.- Introducción.

Caso práctico

Juan quiere comenzar con consultas básicas a los datos, cosas bastante concretas y sencillas de manera que se obtenga información relevante de cada una de las tablas. También quieren realizar algunos cálculos como conocer el salario medio de cada empleado, o el mayor salario de cada departamento, o saber cuánto tiempo lleva cada empleado en la empresa.

IdITE=111532_im_1.jpg (Uso educativo nc)

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.

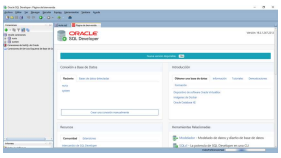
En 1992, ANSI e ISO completaron la estandarización de SQL y se definieron las sentencias básicas que debía contemplar SQL para que fuera estándar. A este SQL se le denominó ANSI-SQL o SQL92.

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.

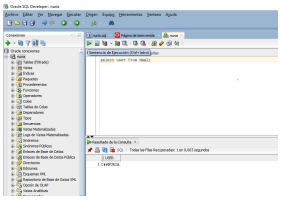
La siguiente fase será manipular los datos, es decir, trabajar con sentencias DML (en español Lenguaje de Manipulación de Datos). Este conjunto de sentencias está orientado a consultas y manejo de datos de los objetos creados. Básicamente consta de cuatro sentencias: **`SELECT`**, **`INSERT`**, **`DELETE`** 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:



Oracle Corporation (Todos los derechos reservados)

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.



Oracle Corporation (Todos los derechos reservados)

Oracle Corporation (Todos los derechos reservados)

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##.

Debes conocer

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.

Autoevaluación

¿Con qué sentencias se definen las estructuras donde agrupar la información, es decir, las tablas?

- ○ DML.
- ○ DDL.
- ○ DCL.

No, este es el lenguaje que incluye manipulación de datos.

Así es, dentro del lenguaje de definición de datos está la creación de tablas.

No, deberías haber leído mejor.

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto

2.- La sentencia SELECT.

Caso práctico

Ana está trabajando con la tabla Partidas, de ahí quiere ver qué información es la más importante, para así crear las consultas más sencillas pero a la vez más frecuentes. Sabe que con SQL y utilizando el comando **SELECT** puede sacar provecho a los datos contenidos en una tabla.

Ministerio de Educación (Uso educativo nc)

¿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.

Recuerda

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.

Autoevaluación

¿Qué se debe indicar a continuación de la cláusula **FROM**?

- ○ Las columnas que queremos seleccionar.
- ○ Los criterios con los que filtro la selección.
- ○ Las tablas de donde se van a extraer los datos.
- ○ La ordenación ascendente.

No, éstas irían junto a la cláusula `SELECT`.

No, éstos irían con la cláusula `WHERE`.

Así es, Aparecerán todos los nombres de las tablas cuyas columnas estemos seleccionando, separadas por coma.

No, el tipo de ordenación aparece junto a la cláusula `ORDER BY`.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta
4. Incorrecto

2.1.- Cláusula 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;
```

Para saber más

Si quieres conocer algo más sobre esta sentencia y ver algunos ejemplos del uso de `SELECT` aquí tienes el siguiente enlace:

La cláusula `SELECT`.

2.2.- Cláusula 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 **l**as 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.- Cláusula 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.

Para saber más

Aquí te adelantamos los operadores para que vayas conociéndolos. Con ellos trabajarás cuando hayas adquirido algunos conocimientos más:

Operadores SQL

2.4.- Ordenación de registros. Cláusula ORDER BY.

Stockbyte. (Uso educativo nc)

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.

¿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**.

Autoevaluación

Relaciona cada cláusula de la sentencia **SELECT** con la información que debe seguirle:

Ejercicio de relacionar

Cláusula	Relación	Información que le sigue.
WHERE	☐	1. Ordenación.
ORDER BY	☐	2. Columnas.
FROM	☐	3. Tablas.
SELECT	☐	4. Condiciones.

Enviar

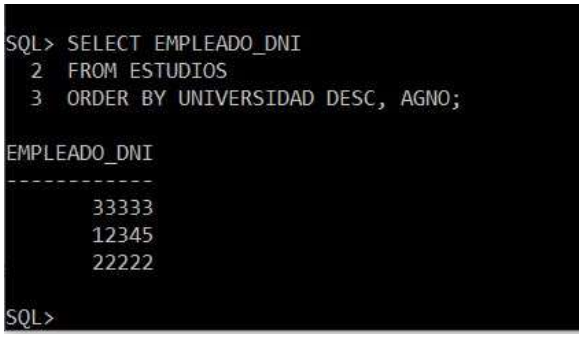
La sintaxis de esta sentencia es muy sencilla y fácil de comprender, ¿no crees?

Ejercicio resuelto

Utilizando las tablas y datos de la aplicación EMPRESA descargados anteriormente, vamos a realizar una consulta donde obtengamos de la tabla ESTUDIOS, el DNI de los empleados ordenados por Universidad descendente y año de manera ascendente. La columna año la hemos llamado agno para no utilizar el caracter ñ que no es admitido en la mayor parte de los SGBD.

Mostrar retroalimentación

```
SELECT EMPLEADO_DNI
FROM ESTUDIOS
ORDER BY UNIVERSIDAD DESC, AGNO;
```



3.- Operadores.

Caso práctico

En el proyecto en el que actualmente trabajan **Ana** y **Juan**, tendrán que realizar consultas que cumplan unos criterios concretos, por ejemplo, obtener el número de jugadores que tienen cierto número de créditos o aquellos que son mujeres e incluso conocer el número de usuarios que son de una provincia y además sean hombres.

Para poder realizar este tipo de consultas necesitaremos utilizar operadores que sirvan para crear las expresiones necesarias. **Ana** y **Juan** conocen los 4 tipos de operadores con los que se puede trabajar: relacionales, aritméticos, de concatenación y lógicos.

Stockbyte. (Uso educativo nc)

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.

Para saber más

Si quieres conocer un poco más sobre los operadores visita este enlace:

Operadores.

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:

Stockbyte. (Uso educativo nc)

Operadores y su significado.

OPERADOR	SIGNIFICADO
=	Igualdad.
!=, < >, ^=	Desigualdad. Distinto
<	<
>	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 %';
```

Ejercicio resuelto

Utilizando las tablas y datos de la aplicación EMPRESA descargados anteriormente, vamos a realizar una consulta donde obtengamos las universidades de Sevilla o Madrid.

Mostrar retroalimentación

```
SQL> SELECT UNIV_COD, NOMBRE_UNIV FROM UNIVERSIDADES WHERE CIUDAD IN ('SEVILLA', 'MADRID');

  UNIV_COD NOMBRE_UNIV
---------- ------------------------
         1 UNED
         2 SEVILLA

SQL>
```

Para saber más

En el siguiente enlace tienes un completo manual de SQL de Oracle explicado de forma sencilla y preparado para hacer ejercicios online. Te será útil para practicar.

Tutorial SQL Oracle

Las **expresiones regulares** permiten forman un patrón, normalmente representativo de otro grupo de carácteres mayor, para comparar el patrón con otro conjunto de carácteres para ver las coincidencias.Es una herramienta potente para comparar cadenas que cumplan un determinado patrón, por ejemplo para determinar si un email está bien compuesto. En el siguiente enlace tienes toda la información con ejemplos.

Uso de expresiones regulares

Los operadores que se utilizan en MySQL puedes verlos en el siguiente enlace:

Operadores de comparación en MySQL.

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;
```

Para saber más

Los operadores que se utilizan en MySQL puedes verlos en el siguiente enlace:

Operadores aritméticos en MySQL.

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:

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€:

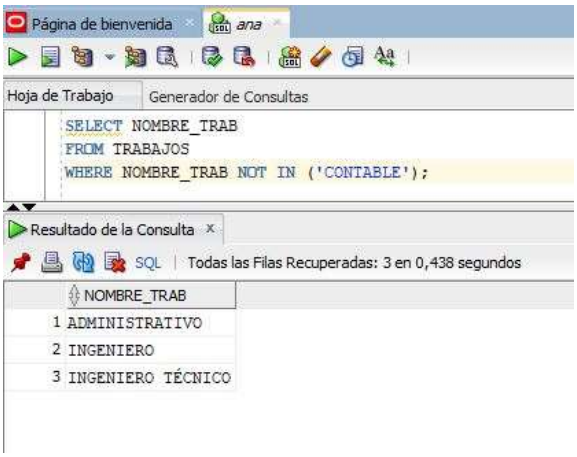
```
SELECT empleado_dni
FROM HISTORIAL_SALARIAL
WHERE salario <=800 OR salario>2000;
```

Ejercicio resuelto

Utilizando las tablas y datos de la aplicación EMPRESA descargados anteriormente, vamos a realizar una consulta donde obtengamos todos nombres de trabajos menos el de contable.

Mostrar retroalimentación

Escribimos la sentencia en SQLDeveloper



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.


Stockbyte. (Uso educativo nc)

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.

Autoevaluación

En la siguiente consulta:

```
SELECT APELLIDOS
FROM JUGADORES
WHERE APELLIDOS LIKE 'A%S%';
```

¿Qué estaríamos seleccionando?

- ○ Aquellos jugadores cuyos apellidos contienen la letra A y la S.
- ○ Aquellos jugadores cuyos apellidos comienzan por la letra A y contienen la letra S.
- ○ Aquellos jugadores cuyos apellidos no contienen ni la letra A ni la S.
- ○ Todos los apellidos de todos los jugadores menos los que su apellido comienza por S.

No, pues queremos que la A tenga una posición concreta.

Efectivamente. El operador **LIKE** es unos de los operadores de comparación más utilizados.

Todo lo contrario, esas letras son las que estamos buscando.

No, creo que no has pensado bien tu respuesta.

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto
4. Incorrecto

4.- Consultas calculadas.

Caso práctico

A la empresa ha llegado **Carlos** que está en fase de prácticas y como anda un poco desubicado ha comenzado su trabajo revisando la teoría y práctica que han dado en clase. No recuerda bien como se creaban campos nuevos a partir de otros ya existentes en la base de datos. Sabe que es algo sencillo pero no quiere meter la pata ya que está ayudando a **Juan** en un proyecto que acaba de entrar.

Ministerio de Educación (Uso educativo nc)

Lo que hará será practicar a partir de una tabla que tenga bastantes campos numéricos de manera que pueda manipular la información sin modificar nada.

En clase trabajaban con la tabla ARTICULOS que tenía, entre otros, los campos Precio y Cantidad. A partir de ellos podría realizar consultas calculadas para obtener el precio con IVA incluido, un descuento sobre el precio e incluso aumentar ese precio en un porcentaje concreto. Seguro que se pone al día rápidamente.

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;
```

Autoevaluación

Los campos calculados pueden ir en:

- [ ] La cláusula `SELECT`.
- [ ] La cláusula `WHERE`.
- [ ] La cláusula `FROM`.

Mostrar retroalimentación

Solución

1. Correcto
2. Correcto
3. Incorrecto

5.- Funciones.

Caso práctico

Juan le ha pedido a **Ana** que calcule la edad actual de los usuarios que tienen registrados en la base de datos pues sería interesante realizar estadísticas mensuales sobre los grupos de edad que acceden al sistema y en función de ello obtener algunos resultados interesantes para la empresa. Para realizar el cálculo de la edad tendríamos que echar mano a funciones que nos ayuden con los cálculos. Existen funciones que nos facilitarán la tarea y nos ayudarán a obtener información que de otro modo resultaría complicado.

Stockbyte (Uso educativo nc)

¿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,
- otras

Oracle proporciona una tabla con la que podemos hacer pruebas, esta tabla se llama Dual y contiene un único campo llamado **`DUMMY`** y una sola fila.

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

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 e^n , es decir, el exponente en base e del número n.

  Ejemplo:

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

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

  Ejemplo:

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

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

  Ejemplo:

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

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

  Ejemplo:

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

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

  Ejemplo:

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

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

  Ejemplo:

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

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

  Ejemplo:

```
SELECT SQRT(25) FROM DUAL; --Resultado: 5
```

- **TRUNC(m,n)**
  Trunca un número a la cantidad de decimales especificada por el segundo argumento. Si se omite el segundo argumento, se truncan todos los decimales. Si "n" es negativo, el número es truncado desde la parte entera.

  Ejemplos:

```
SELECT TRUNC(127.4567, 2) FROM DUAL; -- Resultado: 127.45
SELECT TRUNC(4572.5678, -2) FROM DUAL; -- Resultado: 4500
SELECT TRUNC(4572.5678, -1) FROM DUAL; -- Resultado: 4570
SELECT TRUNC(4572.5678) FROM DUAL; -- Resultado: 4572
```

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

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

Para saber más

Aquí encontrarás las funciones que has visto y algunas más.

Más funciones numéricas.

5.2.- Funciones de cadena de caracteres.

Stockbyte. (Uso educativo nc)

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 minúsculas
```

- **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
```

Autoevaluación

En la siguiente consulta: `SELECT LENGTH("Adiós") FROM DUAL;` ¿qué obtendríamos?

- ○ 5
- ○ 4
- ○ 6
- ○ Nos devolvería un error.

Deberías fijarte bien en el contenido de la función.

No, fíjate bien en el formato de la función.

No, inténtalo de nuevo.

Cierto, las cadenas van con comillas simples.

Solución

1. Incorrecto
2. Incorrecto
3. Incorrecto
4. Opción correcta

5.3.- Funciones de manejo de fechas.

Stockbyte. (Uso educativo nc)

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
```

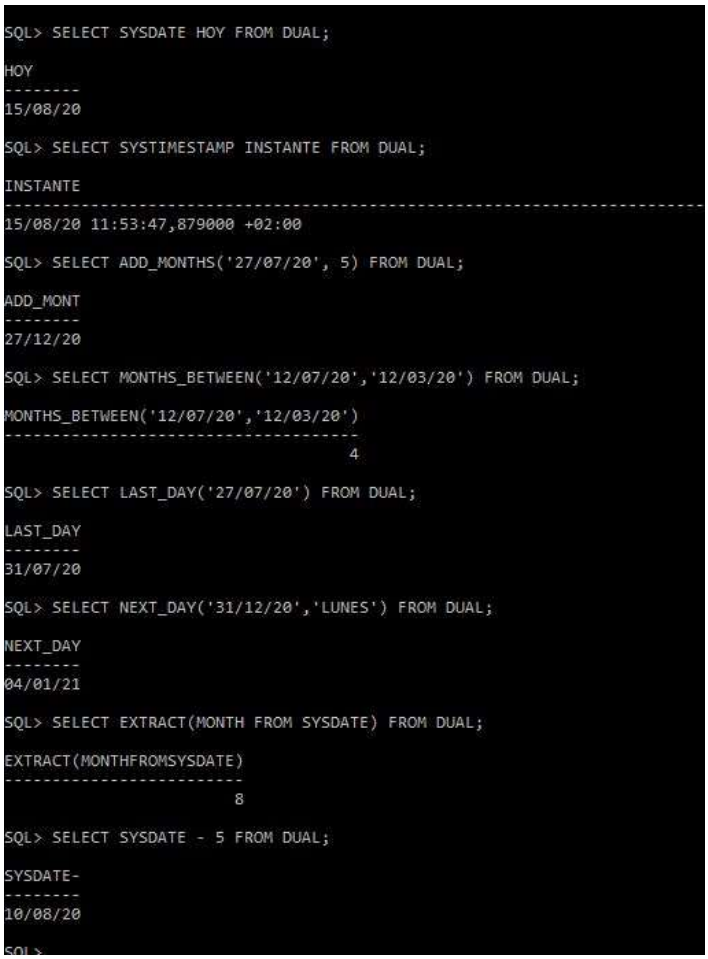
- **`EXTRACT(valor FROM fecha)`** Extrae un valor de una fecha concreta. El valor puede ser day, month, year, hours, etc. Ejemplo:

```
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.



```
SQL> SELECT SYSDATE HOY FROM DUAL;

HOY
--------
15/08/20

SQL> SELECT SYSTIMESTAMP INSTANTE FROM DUAL;

INSTANTE
---------------------------------------------------------------------------
15/08/20 11:53:47,879000 +02:00

SQL> SELECT ADD_MONTHS('27/07/20', 5) FROM DUAL;

ADD_MONT
--------
27/12/20

SQL> SELECT MONTHS_BETWEEN('12/07/20','12/03/20') FROM DUAL;

MONTHS_BETWEEN('12/07/20','12/03/20')
-------------------------------------
                                    4

SQL> SELECT LAST_DAY('27/07/20') FROM DUAL;

LAST_DAY
--------
31/07/20

SQL> SELECT NEXT_DAY('31/12/20','LUNES') FROM DUAL;

NEXT_DAY
--------
04/01/21

SQL> SELECT EXTRACT(MONTH FROM SYSDATE) FROM DUAL;

EXTRACT(MONTHFROMSYSDATE)
-------------------------
                        8

SQL> SELECT SYSDATE - 5 FROM DUAL;

SYSDATE-
--------
10/08/20

SQL>
```


Autoevaluación

¿Cuáles de estas afirmaciones sobre funciones de manejo de fechas son ciertas?

- [ ] Existen dos tipos de fechas de datos con las que podemos trabajar, **`DATE`** y **`TIMESTAMP`**.
- [ ] Se puede poner como argumento el nombre de un campo de cualquier tipo.
- [ ] Le podemos sumar o restar números, lo cual se entiende como sumarle o restarle días.
- [ ] La diferencia entre dos fechas nos dará un número de días.

Mostrar retroalimentación

Solución

1. Correcto
2. Incorrecto
3. Correcto
4. Correcto

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 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

Para saber más

En el siguiente enlace tienes las funciones de conversión vistas y algunas más con ejemplos

Funciones de conversión en Oracle

5.5.- Otras funciones: NVL y DECODE.

Stockbyte. (Uso educativo nc)

¿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;
```

Autoevaluación

¿Qué función convierte un número o fecha a cadena de caracteres?

- ○ **TO_DATE.**
- ○ **TO_CHAR.**
- ○ **DECODE.**
- ○ **TO_NUMBER.**

No es correcto. Esta función convierte texto a fecha.

Así es, se utiliza normalmente para fechas ya que de número a texto se hace de forma implícita

No es correcto. Esta función evalúa expresiones y devuelve un valor según el resultado.

No es correcto. Convierte textos en números, y suele utilizarse para dar un formato concreto a los números.

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto
4. Incorrecto

6.- Consultas de resumen.

Caso práctico

Ada le ha pedido a **Juan** que le eche una mano en otro de los proyectos en los que está inmersa la empresa. Necesita que cree varias consultas de resumen sobre unas tablas de empleados de banca. Está interesada en obtener el salario medio de los empleados clasificado por tipo de empleado, y quiere también que obtenga el total de empleados por sucursal.

Realmente no es un trabajo difícil ya que las consultas de resumen son muy fáciles de crear, pero **Ada** está tan ocupada que no tiene tiempo para esos detalles.

Stockbyte (Uso educativo nc)

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.

Para saber más

Puedes acceder a este enlace si quieres conocer más sobre este tipo de consultas.

Consultas de resumen.

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,

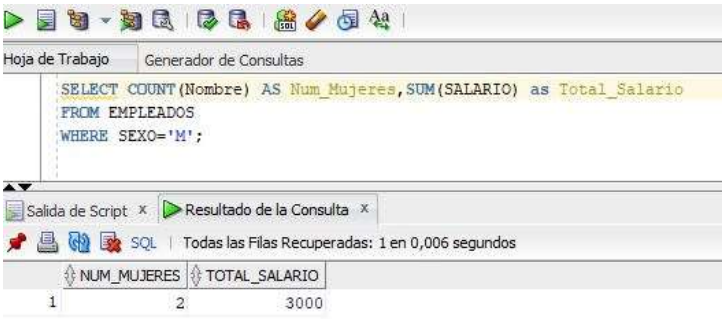
```
SELECT COUNT(nombre) FROM Usuarios;
SELECT COUNT(*) FROM Usuarios;
```

Ejercicio resuelto

Utilizando las tablas y datos de la aplicación EMPRESA descargados anteriormente, vamos a realizar una consulta donde contemos el número de empleados que son mujeres y la suma total de sus salarios.

Mostrar retroalimentación

```
SELECT COUNT(Nombre),SUM(SALARIO) FROM EMPLEADOS WHERE SEXO='M';
```



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.

Stockbyte (Uso educativo nc)

- **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;
```

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

Stockbyte (Uso educativo nc)

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)`

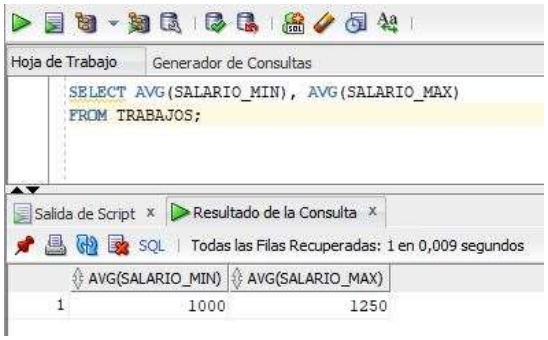
  Devuelve la desviación típica estadística de todos los valores de la expresión.Como tipo de dato admite únicamente columnas numéricas. Los valores nulos (**`NULL`**) se omiten.

Ejercicio resuelto

Utilizando las tablas y datos de la aplicación empresa descargados anteriormente, vamos a realizar una consulta donde obtengamos la media de la columna salario mínimo y de la columna salario máximo de la tabla TRABAJOS.

Mostrar retroalimentación

```
SELECT AVG(SALARIO_MIN), AVG(SALARIO_MAX) FROM TRABAJOS;
```



Elaboración propia (Uso educativo nc)

Autoevaluación

¿Cuáles de las siguientes afirmaciones sobre las consultas de resumen son ciertas?

- [ ] Toman un grupo de datos de una columna.
- [ ] Producen un único dato que resume el grupo.
- [ ] Utilizar una función de agregado en una consulta la convierte en consulta de resumen.
- [ ] Dan como resultado un subconjunto de filas de la tabla.

Mostrar retroalimentación

Solución

1. Correcto
2. Correcto
3. Correcto
4. Incorrecto

7.- Agrupamiento de registros.

Caso práctico

Juan ha estado realizando algunas consultas de resumen y ahora quiere continuar sacando todo el jugo posible a las tablas realizando operaciones como las anteriores pero agrupándolas por algún campo. Hay veces que se obtiene mucha información si estudiamos los datos por grupos, como puede ser el número de jugadores por provincia, o el saldo medio según el sexo del jugador para así poder obtener conclusiones sobre la información guardada.

Stockbyte. (Uso educativo nc)

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.

Oracle/Elaboración propia (Uso educativo nc)

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';
```

Autoevaluación

Relaciona cada cláusula con su orden de ejecución:

Ejercicio de relacionar

Cláusula.	Relación.	Función.
WHERE	☐	1. PRIMERO
ORDER BY	☐	2. SEGUNDO
HAVING	☐	3. TERCERO
GROUP BY	☐	4. CUARTO

Enviar

Efectivamente, es muy importante saber este orden para saber filtrar efectivamente los datos.

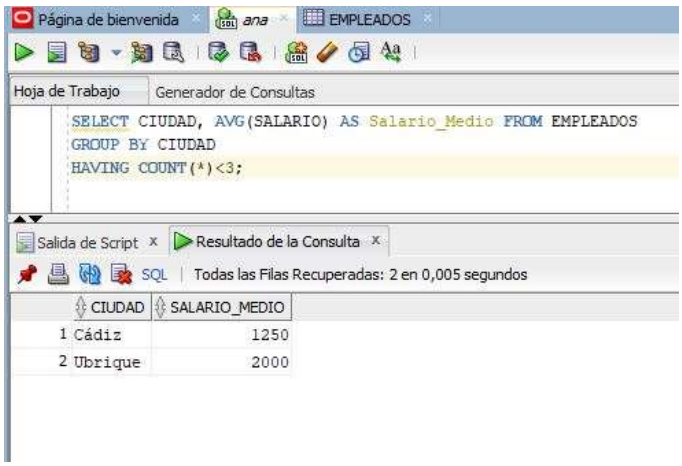
Ejercicio Resuelto

Utilizando las tablas y datos de la aplicación EMPRESA descargados

1. Obtener, agrupando, por ciudad el salario medio de los empleados, siempre y cuando en esa ciudad (grupo) haya menos de 3 empleados.
2. Obtener de la tabla HISTORIAL_LABORAL el número de empleados que ha trabajado en cada Departamento

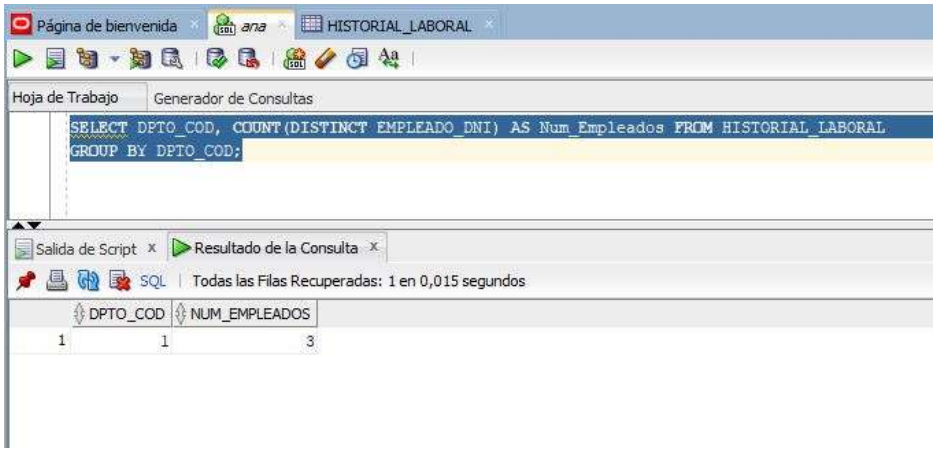
Mostrar retroalimentación

- Obtener, agrupando, por ciudad el salario medio de los empleados, siempre y cuando en esa ciudad (grupo) haya menos de 3 empleados.



Oracle/Elaboración propia (Uso educativo)

Obtener de la tabla HISTORIAL_LABORAL el número de empleados que ha trabajado en cada Departamento. Si un empleado ha trabajado en distintos períodos en el mismo Departamento sólo se contabilizará una vez. Para eso, vamos a utilizar la cláusula DISTINCT para que solo cuente valores distintos.



Oracle / Elaboración propia (Uso educativo)

8.- Consultas multitablas.

Caso práctico

Hasta ahora **Juan** ha estado haciendo uso de consultas a una única tabla, pero eso limita la obtención de resultados. Si esas tablas están relacionadas **Juan** podrá coger información de cada una de ellas según lo que le interese. En las tablas de la aplicación **`EMPRESA`**, tiene por un lado la tabla que recoge los datos del empleado y por otra su historial laboral. En esta última tabla, lo único que recogemos de los empleados es su código. Como a **Juan** le interesa obtener el historial laboral incluyendo nombres y apellidos de sus empleados, debe utilizar la información que viene en ambas tablas.

Ministerio de Educación (Uso educativo nc)

Stockbyte (Uso educativo nc)

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;
```

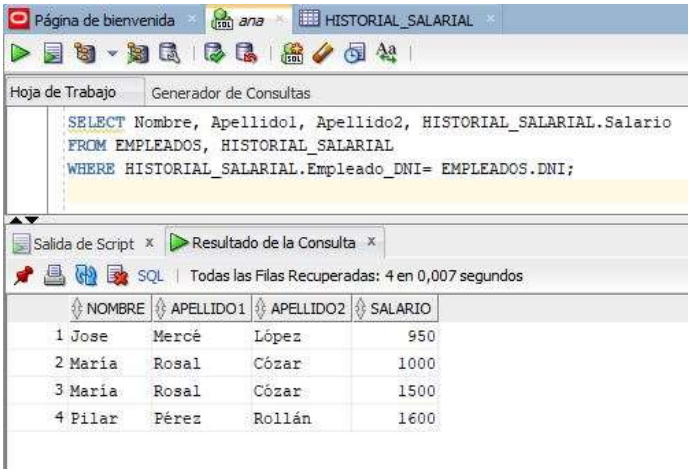
Vamos a obtener el historial con los nombres de departamento, nombre y apellidos del empleado de todos los departamentos:

```
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;
```

Ejercicio resuelto

Utilizando las tablas y datos de la aplicación empresa descargados anteriormente, vamos a realizar una consulta donde obtengamos el nombre de los empleados junto a los salarios que ha tenido consultando para ello la tabla HISTORIAL_SALARIAL

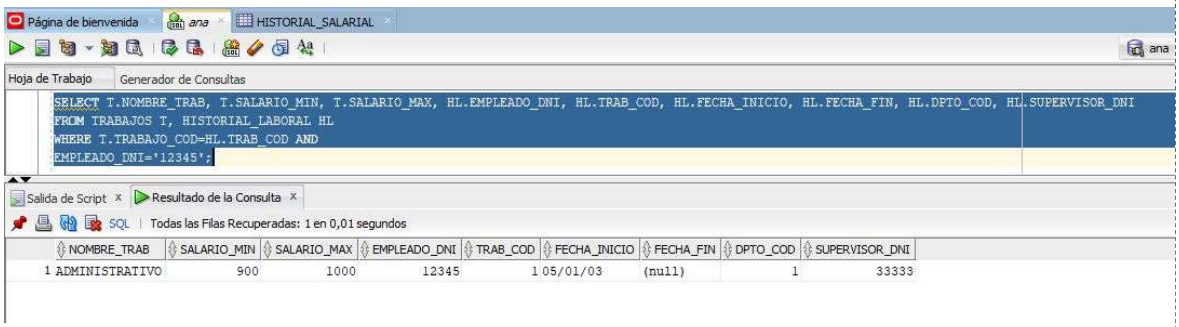
Mostrar retroalimentación



Oracle /Elaboración propia (Uso educativo nc)

Obtener un listado con el histórico laboral de un empleador cuyo DNI sea '12345'. En dicho listado interesa conocer el nombre del puesto, así como el rango salarial.

Mostrar retroalimentación



Oracle / Elaboración propia (Uso educativo nc)

8.2.- Composiciones externas.

Stockbyte (Uso educativo nc)

¿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.

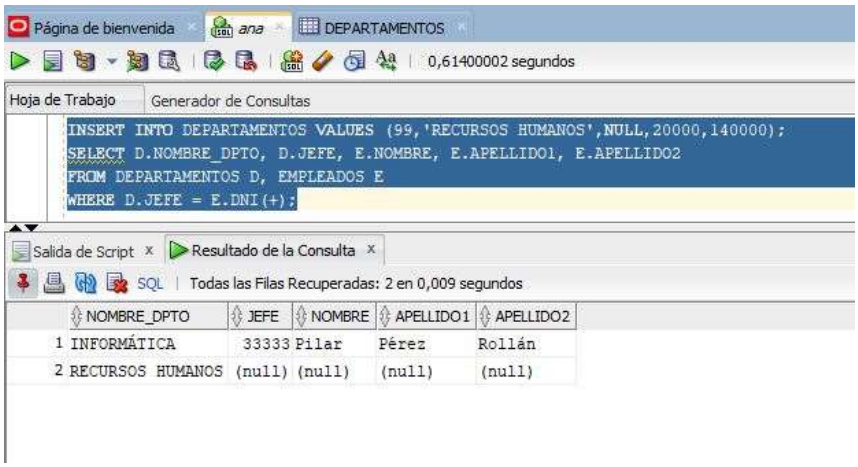
Ejercicio resuelto

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.

Mostrar retroalimentación

```
SELECT D.NOMBRE_DPTO, D.JEFE, E.NOMBRE, E.APELLIDO1, E.APELLIDO2
FROM DEPARTAMENTOS D, EMPLEADOS E
WHERE D.JEFE = E.DNI(+);
```

Para tener algún dato que cumpla la condición, es decir, algún departamento sin jefe insertamos previamente una fila. Si hubieramos realizado un join normal este último departamenteo no habría salido. Pruébalo.



Oracle / Elaboración propia (Uso educativo nc)

Autoevaluación

Si queremos incluir aquellas filas que no tienen aún correspondencia con la tabla relacionada, tendremos que poner un signo más entre paréntesis:

- ○ Delante del nombre de la tabla en la cláusula FROM.
- ○ Delante del nombre del campo que relaciona donde sabemos que hay valores nulos.

- ○ Detrás del nombre del campo que relaciona donde sabemos que hay valores nulos.
- ○ Delante del nombre del campo que relaciona donde sabemos que no hay valores nulos.

Incorrecto, inténtalo de nuevo, en la cláusula FROM se listan las tablas y sus alias.

No es correcto, piénsalo bien, delante del nombre del campo indicamos la tabla acompañada de un punto y seguida del nombre del campo.

Estupendo, ya has aprendido a realizar composiciones externas.

No es cierto, lo siento, debes intentarlo de nuevo.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta
4. Incorrecto

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:

Stockbyte (Uso educativo nc)

```
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);
```

Para saber más

En MySQL también se utilizan las composiciones, aquí puedes verlo:

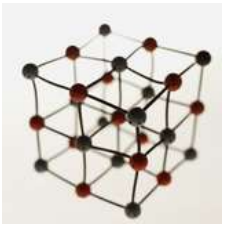
Composiciones.

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

Caso práctico

Ana le cuenta a Carlos que ya tienen terminado casi todo el trabajo, pero que no le importa enseñarle otros tipos de consultas que no han necesitado utilizar en esta ocasión pero que es conveniente conocer, se refiere al uso de uniones, intersecciones y diferencia de consultas. Le explicará que es muy parecido a la teoría de conjuntos que recordará de haber terminado hace poco sus estudios.

Ministerio de Educación (Uso educativo nc)



Stockbyte (Uso educativo nc)

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;
```

Autoevaluación

¿Cuáles de las siguientes afirmaciones son correctas?

- [ ] La unión combina las filas de un primer `SELECT` con las filas de otro `SELECT`, desapareciendo las filas duplicadas.
- [ ] La diferencia devuelve aquellas filas que están en el primer `SELECT` pero no en el segundo. Correcta.
- [ ] La intersección examina las filas de un `SELECT` y de otro y devolverá aquellas que aparezcan en ambos conjuntos.
- [ ] En uniones, intersecciones y diferencias, los dos `SELECT` deben tener el mismo número pero no tienen por qué tener el mismo tipo de columnas.

Mostrar retroalimentación

Solución

1. Correcto
2. Correcto
3. Correcto
4. Correcto

10.- Subconsultas.

Caso práctico

— ¿Es posible consultar dentro de otra consulta? — pregunta **Carlos**.

Ha estado pensando que a veces va a necesitar filtrar los datos en función de un resultado que a priori desconoce. **Ana** se pone manos a la obra porque ve que ha llegado el momento de explicarle a **Carlos** las subconsultas.

Stockbyte (Uso educativo nc)

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);
```

Autoevaluación

Relaciona cada instrucción con su función:

Ejercicio de relacionar

Instrucción.	Relación.	Función.
`ANY`	☐	1. Compara con cualquier fila de la consulta.
`ALL`	☐	2. Comprueba si el valor se encuentra en el resultado de la subconsulta.
`IN`	☐	3. Compara con todas las filas de la consulta.
`NOT IN`	☐	4. Comprueba si un valor no se encuentra en una subconsulta.

Enviar

Recuerda que estas instrucciones se utilizan en subconsultas que devuelven más de una fila.

Para saber más

¿Quieres más ejemplos con los que practicar?

Ejercicios SQL.

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

Enunciado.

Un sitio de juegos online por Internet desea contar con una base de datos para gestionar los usuarios, juegos y partidas que se desarrollan en el mismo. El funcionamiento del sitio es el siguiente:

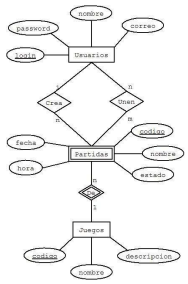
Cuando un usuario intenta entrar en este sitio, se le pedirá un login y un password. El sistema comprobará si el usuario tiene cuenta y en caso negativo se le pedirá el nombre, correo, login y password.

De los juegos se quiere almacenar un código identificador, nombre y descripción.

Los usuarios que tengan en casa el juego apropiado, podrán crear partidas de ese juego para que otros usuarios se unan a la partida o unirse a partidas existentes.

De las partidas se almacenará un código de partida, la fecha y hora de creación, el nombre de la partida y el estado (en curso o finalizada). Además hay que tener en cuenta que una partida sólo puede ser de un juego y un juego tener varias partidas.

Diagrama E-R.



Ministerio de Educación. (Uso Educativo nc)

Estructura de Tablas.

Estructura de tablas que forman la base de datos de juegos online.

USUARIOS	PARTIDAS	JUEGOS	UNEN
login VARCHAR2(15) password VARCHAR2(9) nombre VARCHAR2(25) apellidos VARCHAR2(30) direccion VARCHAR2(30) cp VARCHAR2(5) localidad VARCHAR2(25) provincia VARCHAR2(25) pais VARCHAR2(15) f_nacimiento DATE f_ingreso DATE correo VARCHAR2(25) credito NUMBER sexo VARCHAR2(1)	codigo VARCHAR2(15) nombre VARCHAR2(25) estado VARCHAR2(1) cod_juego VARCHAR2(15) fecha_inicio_partida DATE hora_inicio_partida TIMESTAMP cod_creador_partida VARCHAR2(15)	codigo VARCHAR2(15) nombre VARCHAR2(25) descripcion VARCHAR2(200)	codigo_partida VARCHAR2(15) codigo_usuario VARCHAR2(15)

Sentencias de creación de tablas

A continuación se presentan unas posibles sentencias SQL de creación de las tablas del ejercicio.

TABLA USUARIOS

```
CREATE TABLE USUARIOS (
        login         VARCHAR2(15) PRIMARY KEY NOT NULL,
        password      VARCHAR2(9) NOT NULL,
        nombre        VARCHAR2(25) NOT NULL,
        apellidos     VARCHAR2(30) NOT NULL,
        direccion     VARCHAR2(30) NOT NULL,
        cp            VARCHAR2(5) NOT NULL,
        localidad     VARCHAR2(25) NOT NULL,
```

```
        provincia     VARCHAR2(25) NOT NULL,
        pais          VARCHAR2(15) NOT NULL,
        f_nac      DATE,
        f_ing      DATE DEFAULT (sysdate),
        correo     VARCHAR2(25) NOT NULL,
        credito    NUMBER,
        sexo       VARCHAR2(1));
```

TABLA JUEGOS

```
CREATE TABLE JUEGOS(
        codigo        VARCHAR2(15) PRIMARY KEY NOT NULL,
        nombre        VARCHAR2(15) NOT NULL,
        descripcion   VARCHAR2(200) NOT NULL);
```

TABLA PARTIDAS

```
CREATE TABLE PARTIDAS(
        codigo        VARCHAR2(15) PRIMARY KEY NOT NULL,
        nombre        VARCHAR2(25) NOT NULL,
        estado        VARCHAR2(1) NOT NULL,
        cod_juego     VARCHAR2(15) NOT NULL
                      CONSTRAINT CA_cod_juego REFERENCES JUEGOS(codigo),
        fecha_inicio  DATE,
        hora_inicio   TIMESTAMP,
        cod_creador   VARCHAR2(15)
                      CONSTRAINT CA_cod_creador REFERENCES USUARIOS(login));
```

TABLA UNEN

```
CREATE TABLE UNEN(
       codigo_partida VARCHAR2(15) NOT NULL
                      CONSTRAINT CA_codigo_partida REFERENCES PARTIDAS(codigo),
       codigo_usuario VARCHAR2(15) NOT NULL
                      CONSTRAINT CA_codigo_usuario REFERENCES USUARIOS(login),
       CONSTRAINT PK_UNEN primary key (codigo_partida, codigo_usuario));
```

Ejemplos de datos.

TABLA USUARIOS(Campos desde Login a Código Postal):

Datos de la tabla USUARIOS. Primera parte.

LOGIN	PASSWORD	NOMBRE	APELLIDOS	DIRECCION	CP
anamat56	JD9U6?	ANA M.	MATA VARGAS	GARCILASO DE LA VEGA	08924
alecam89	5;5@PK	ALEJANDRO EMILIO	CAMINO LAZARO	PEDRO AGUADO BLEYE	34004
verbad64	MP49HF	VERONICA	BADIOLA PICAZO	BARRANCO GUINIGUADA	35015
conmar76	O1<N9U	CONSUELO	MARTINEZ RODRIGUEZ	ROSA	04002
encpay57	FYC3L5	ENCARNACIÓN	PAYO MORALES	MULLER,AVINGUDA	43007
mandia79	00JRIH	MANUELA	DIAZ COLAS	214 (GENOVA)	07015
alibar52	IER8S	ALICIA MARIA	BARRANCO CALLIZO	HECTOR VILLALOBOS	29014
adofid63	;82=MH	ADOLFO	FIDALGO DIEZ	FORCALL	12006
jesdie98	X565ZS	JESUS	DIEZ GIL	TABAIBAL	35213
pedsan70	T?5=J@	PEDRO	SANCHEZ GUIL	PINTOR ZULOAGA	03013
diahue96	LSQZMC	DIANA	HUERTA VALIOS	JOAQUIN SALAS	39011
robrod74	<LQMLP	ROBERTO	RODRIGUEZ PARMO	CASTILLO HIDALGO	51002
milgar78	SF=UZ8	MILAGROSA	GARCIA ELVIRA	PEDRALBA	28037
frabar93	19JZ7@	FRANCISCA	BARRANCO RODRIGUEZ	BALSAS, LAS	26006
migarc93	AAFLTW	MIGUEL ANGEL	ARCOS ALONSO	ISAAC ALBENIZ	04008

TABLA USUARIOS (Campos desde Localidad a Sexo):

Datos de la tabla USUARIOS. Segunda parte.

LOCALIDAD	PROVINCIA	PAIS	F_NACIMIENTO	F_INGRESO	CORREO	CREDITO	SEXO
SANTA COLOMA DE GRAMANET	BARCELONA	ESPAÑA	08/25/1974	10/10/2007	anamat56@hotmail.com	213	M
PALENCIA	PALENCIA	ESPAÑA	05/03/1976	10/15/2010	alecam89@hotmail.com	169	H
PALMAS DE GRAN CANARIA,LAS	PALMAS (LAS)	ESPAÑA	01/28/1984	10/23/2010	verbad64@hotmail.com	437	M
ALMERÍA	ALMERÍA	ESPAÑA	08/09/1978	03/25/2007	conmar76@yahoo.com	393	M
TARRAGONA	TARRAGONA	ESPAÑA	05/04/1993	01/06/2010	encpay57@yahoo.com	318	M
PALMA DE MALLORCA	BALEARES	ESPAÑA	07/14/1979	07/16/2008	mandia79@hotmail.com	255	M
MÁLAGA	MÁLAGA	ESPAÑA	08/21/1993	09/19/2010	alibar52@hotmail.com	486	M
CASTELLÓN DE LA PLANA	CASTELLÓN	ESPAÑA	08/11/1981	03/02/2008	adofid63@gmail.com	154	H
TELDE	PALMAS (LAS)	ESPAÑA	10/23/1981	09/13/2009	jesdie98@gmail.com	152	H
ALACANT/ALICANTE	ALICANTE	ESPAÑA	12/01/1983	06/15/2008	pedsan70@yahoo.com	21	H
SANTANDER	CANTABRIA	ESPAÑA	04/25/1984	07/31/2009	diahue96@yahoo.com	395	M
CEUTA	CEUTA	ESPAÑA	06/28/1978	03/16/2009	robrod74@gmail.com	486	H
MADRID	MADRID	ESPAÑA	04/12/1983	05/15/2008	milgar78@gmail.com	330	M
LOGROÑO	RIOJA (LA)	ESPAÑA	09/21/1986	02/16/2008	frabar93@gmail.com	75	M
ALMERÍA	ALMERÍA	ESPAÑA	03/01/1991	06/16/2010	migarc93@hotmail.com	23	H

TABLA PARTIDAS:

Datos de la tabla PARTIDAS.

CODIGO	NOMBRE	ESTADO	COD_JUEGO	FECHA	HORA	COD_CREA
1	Billar_migarc93_18/7	1	12	07/18/2011	00:47:40	migarc93
2	Chinchón_mandia79_2/10	1	6	10/02/2011	01:47:00	mandia79
3	Canasta_alibar52_26/2	0	8	02/26/2011	08:57:33	alibar52
4	Damas_verbad64_16/3	1	4	03/16/2011	00:53:00	verbad64
5	Chinchón_alibar52_9/9	1	6	09/09/2011	09:10:22	alibar52
6	Oca_pedsan70_21/12	0	2	12/21/2011	18:53:17	pedsan70
7	Canasta_encpay57_18/2	0	8	02/18/2011	09:41:02	encpay57
8	Pocha_adofid63_26/10	1	10	10/26/2011	02:23:43	adofid63
9	Damas_diahue96_25/6	1	4	06/25/2011	18:11:14	diahue96
10	Parchís_encpay57_31/7	1	1	07/31/2011	21:21:36	encpay57

TABLA JUEGOS:

Datos de la tabla JUEGOS.

CODIGO	NOMBRE	DESCRIPCION
1	Parchís	El parchís es un juego de mesa derivado del pachisi y similar al ludo y al parcheesi
2	Oca	El juego de la oca es un juego de mesa para dos o más jugadores
3	Ajedrez	El ajedrez es un juego entre dos personas, cada una de las cuales dispone de 16 piezas móviles que se colocan sobre un tablero dividido en 64 escaques

CODIGO	NOMBRE	DESCRIPCION
4	Damas	Las damas es un juego de mesa para dos contrincantes
5	Poker	El póquer es un juego de cartas de los llamados de "apuestas"
6	Chinchón	El chinchón es un juego de naipes de 2 a 8 jugadores
7	Mus	El mus es un juego de naipes, originario de Navarra, que en la actualidad se encuentra muy extendido por toda España
8	Canasta	La canasta o rummy-canasta es un juego de naipes, variante del rummy
9	Dominó	El dominó es un juego de mesa en el que se emplean unas fichas rectangulares
10	Pocha	La pocha es un juego de cartas que se juega con la baraja española
11	Backgammon	Cada jugador tiene quince fichas que va moviendo entre veinticuatro triángulos (puntos) según el resultado de sus dos dados
12	Billar	El billar es un deporte de precisión que se practica impulsando con un taco un número variable de bolas

TABLA UNEN:

Datos de la tabla UNEN.

CODIGO_PARTIDA	CODIGO_USUARIO
4	ascbar65
3	norlob93
6	norlob93
2	antcor77
2	anavaz83
2	jesvel57
4	marram77
3	virlue50
5	susizq56
8	virlue50
6	marram77
4	mirtom67
5	oscmar67
4	virlue50
5	susizq56

Inserción de datos

Sentencias de inserción de datos

TABLA USUARIOS

```
ALTER SESSION SET NLS_DATE_FORMAT='MM/DD/YYYY';
INSERT INTO USUARIOS VALUES('anamat56','JD9U6?','ANA M.','MATA VARGAS','GARCILASO DE LA VEGA','8924','SANTA COLOMA DE GRAMANET','BARCELONA','ESPAÑA','08/25/1974','10/10/2007','anamat56@hotmail.com',213,'M');
INSERT INTO USUARIOS VALUES('alecam89','5;5@PK','ALEJANDRO EMILIO','CAMINO LAZARO','PEDRO AGUADO BLEYE','34004','PALENCIA','PALENCIA','ESPAÑA','05/03/1976','10/15/2010','alecam89@hotmail.com',169,'H');
INSERT INTO USUARIOS VALUES('verbad64','MP49HF','VERONICA','BADIOLA PICAZO','BARRANCO GUINIGUADA','35015','PALMAS GRAN CANARIA,LAS','PALMAS (LAS)','ESPAÑA','01/28/1984','10/23/2010','verbad64@hotmail.com',437,'M');
INSERT INTO USUARIOS VALUES('conmar76','O1<N9U','CONSUELO','MARTINEZ RODRIGUEZ','ROSA','4002','ALMERÍA','ALMERÍA','ESPAÑA','08/09/1978','03/25/2007','conmar76@yahoo.com',393,'M');
INSERT INTO USUARIOS VALUES('encpay57','FYC3L5','ENCARNACIÓN','PAYO MORALES','MULLER,AVINGUDA','43007','TARRAGONA','TARRAGONA','ESPAÑA','05/04/1993','01/06/2010','encpay57@yahoo.com',318,'M');
```

INSERT INTO USUARIOS VALUES('mandia79','00JRIH','MANUELA','DIAZ COLAS','214 (GENOVA)','7015','PALMA DE MALLORCA','BALEARES','ESPAÑA','07/14/1979','07/16/2008','mandia79@hotmail.com',255,'M');
INSERT INTO USUARIOS VALUES('alibar52','IER8S','ALICIA MARIA','BARRANCO CALLIZO','HECTOR VILLALOBOS','29014','MÁLAGA','MÁLAGA','ESPAÑA','08/21/1993','09/19/2010','alibar52@hotmail.com',486,'M');
INSERT INTO USUARIOS VALUES('adofid63',';82=MH','ADOLFO','FIDALGO DIEZ','FORCALL','12006','CASTELLÓN DE LA PLANA','CASTELLÓN','ESPAÑA','08/11/1981','03/02/2008','adofid63@gmail.com',154,'H');
INSERT INTO USUARIOS VALUES('jesdie98','X565ZS','JESUS','DIEZ GIL','TABAIBAL','35213','TELDE','PALMAS (LAS)','ESPAÑA','10/23/1981','09/13/2009','jesdie98@gmail.com',152,'H');
INSERT INTO USUARIOS VALUES('pedsan70','T?5=J@','PEDRO','SANCHEZ GUIL','PINTOR ZULOAGA','3013','ALACANT/ALICANTE','ALICANTE','ESPAÑA','12/01/1983','06/15/2008','pedsan70@yahoo.com',21,'H');
INSERT INTO USUARIOS VALUES('diahue96','LSQZMC','DIANA','HUERTA VALIOS','JOAQUIN SALAS','39011','SANTANDER','CANTABRIA','ESPAÑA','04/25/1984','07/31/2009','diahue96@yahoo.com',395,'M');
INSERT INTO USUARIOS VALUES('robrod74','<LQMLP','ROBERTO','RODRIGUEZ PARMO','CASTILLO HIDALGO','51002','CEUTA','CEUTA','ESPAÑA','06/28/1978','03/16/2009','robrod74@gmail.com',486,'H');
INSERT INTO USUARIOS VALUES('milgar78','SF=UZ8','MILAGROSA','GARCIA ELVIRA','PEDRALBA','28037','MADRID','MADRID','ESPAÑA','04/12/1983','05/15/2008','milgar78@gmail.com',330,'M');
INSERT INTO USUARIOS VALUES('frabar93','19JZ7@','FRANCISCA','BARRANCO RODRIGUEZ','BALSAS, LAS','26006','LOGROÑO','RIOJA (LA)','ESPAÑA','09/21/1986','02/16/2008','frabar93@gmail.com',75,'M');
INSERT INTO USUARIOS VALUES('migarc93','AAFLTW','MIGUEL ANGEL','ARCOS ALONSO','ISAAC ALBENIZ','4008','ALMERÍA','ALMERÍA','ESPAÑA','03/01/1991','06/16/2010','migarc93@hotmail.com',23,'H');

TABLA JUEGOS

INSERT INTO JUEGOS VALUES('1','Parchís','El parchís es un juego de mesa derivado del pachisi y similar al ludo y al parcheesi');
INSERT INTO JUEGOS VALUES('2','Oca','El juego de la oca es un juego de mesa para dos o más jugadores');
INSERT INTO JUEGOS VALUES('3','Ajedrez','El ajedrez es un juego entre dos personas, cada una de las cuales dispone de 16 piezas móviles que se colocan sobre un tablero dividido en 64 escaques');
INSERT INTO JUEGOS VALUES('4','Damas','Las damas es un juego de mesa para dos contrincantes');
INSERT INTO JUEGOS VALUES('5','Poker','El póquer es un juego de cartas de los llamados de "apuestas"');
INSERT INTO JUEGOS VALUES('6','Chinchón','El chinchón es un juego de naipes de 2 a 8 jugadores');
INSERT INTO JUEGOS VALUES('7','Mus','El mus es un juego de naipes, originario de Navarra, que en la actualidad se encuentra muy extendido por toda España');
INSERT INTO JUEGOS VALUES('8','Canasta','La canasta o rummy-canasta es un juego de naipes, variante del rummy');
INSERT INTO JUEGOS VALUES('9','Dominó','El dominó es un juego de mesa en el que se emplean unas fichas rectangulares');
INSERT INTO JUEGOS VALUES('10','Pocha','La pocha es un juego de cartas que se juega con la baraja española');
INSERT INTO JUEGOS VALUES('11','Backgammon','Cada jugador tiene quince fichas que va moviendo entre veinticuatro triángulos (puntos) según el resultado de sus dos dados');
INSERT INTO JUEGOS VALUES('12','Billar','El billar es un deporte de precisión que se practica impulsando con un taco un número variable de bolas');

TABLA PARTIDAS

INSERT INTO PARTIDAS VALUES('1','Billar_migarc93_18/7','1','12','07/18/2011',TO_TIMESTAMP ('00:47:40','HH24:MI:SS'),'migarc93');
INSERT INTO PARTIDAS VALUES('2','Chinchón_mandia79_2/10','1','6','10/02/2011',TO_TIMESTAMP ('01:47:40','HH24:MI:SS'),'mandia79');
INSERT INTO PARTIDAS VALUES('3','Canasta_alibar52_26/2','0','8','02/26/2011',TO_TIMESTAMP ('08:57:33','HH24:MI:SS'),'alibar52');
INSERT INTO PARTIDAS VALUES('4','Damas_verbad64_16/3','1','4','03/16/2011',TO_TIMESTAMP ('00:53:00','HH24:MI:SS'),'verbad64');
INSERT INTO PARTIDAS VALUES('5','Chinchón_alibar52_9/9','1','6','09/09/2011',TO_TIMESTAMP ('09:10:22','HH24:MI:SS'),'alibar52');
INSERT INTO PARTIDAS VALUES('6','Oca_pedsan70_21/12','0','2','12/21/2011',TO_TIMESTAMP ('18:53:17','HH24:MI:SS'),'pedsan70');
INSERT INTO PARTIDAS VALUES('7','Canasta_encpay57_18/2','0','8','02/18/2011',TO_TIMESTAMP ('09:41:02','HH24:MI:SS'),'encpay57');
INSERT INTO PARTIDAS VALUES('8','Pocha_adofid63_26/10','1','10','10/26/2011',TO_TIMESTAMP ('02:23:43','HH24:MI:SS'),'adofid63');
INSERT INTO PARTIDAS VALUES('9','Damas_diahue96_25/6','1','4','06/25/2011',TO_TIMESTAMP ('18:11:14','HH24:MI:SS'),'diahue96');
INSERT INTO PARTIDAS VALUES('10','Parchís_encpay57_31/7','1','1','07/31/2011',TO_TIMESTAMP ('21:21:36','HH24:MI:SS'),'encpay57');

TABLA UNEN

INSERT INTO UNEN VALUES('4','anamat56');
INSERT INTO UNEN VALUES('3','alecam89');
INSERT INTO UNEN VALUES('6','alecam89');
INSERT INTO UNEN VALUES('2','conmar76');
INSERT INTO UNEN VALUES('2','encpay57');
INSERT INTO UNEN VALUES('2','mandia79');
INSERT INTO UNEN VALUES('4','alibar52');
INSERT INTO UNEN VALUES('3','adofid63');
INSERT INTO UNEN VALUES('5','jesdie98');
INSERT INTO UNEN VALUES('8','pedsan70');
INSERT INTO UNEN VALUES('6','diahue96');
INSERT INTO UNEN VALUES('4','robrod74');
INSERT INTO UNEN VALUES('5','milgar78');
INSERT INTO UNEN VALUES('4','frabar93');
INSERT INTO UNEN VALUES('5','encpay57');

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

El primero paso será descargarte el script desde el siguiente enlace:

Script SQL: tablas y registros para realizar los ejercicios (zip - 1,39 KB)

Una vez descargado descomprímelo para obtener el script SQL BD04_CONT_R07_02.sql

Vamos a **crear un usuario llamado Ana** para practicar los ejercicios al mismo tiempo que ella.

Recuerda los pasos:

1.- Nos conectamos como Administradores. Escribimos desde la terminal de windows `SQLPLUS sys as sysdba` o abrimos SQLPlus y cuando nos solicite el usuario escribimos `sys as sysdba`.

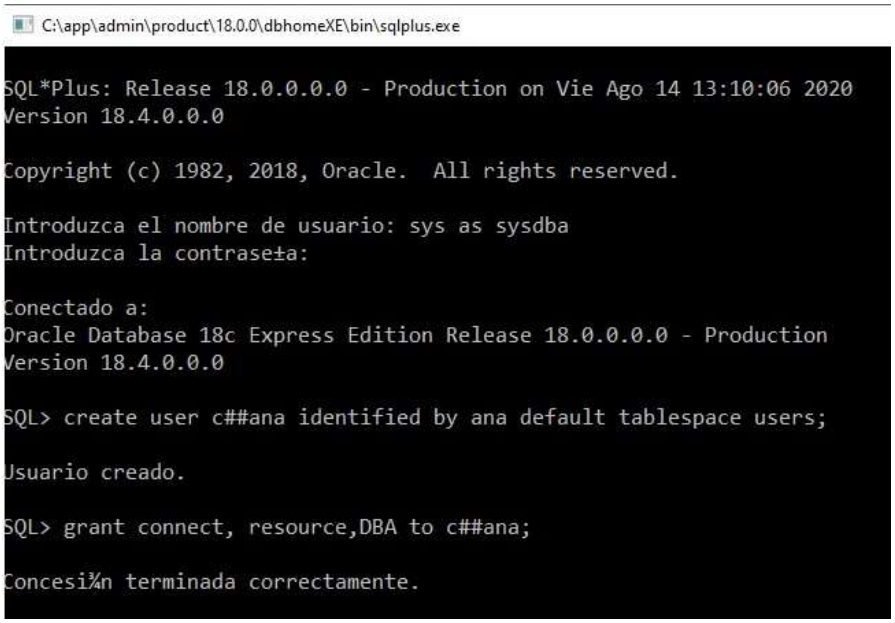
En ambos casos nos pedirá la contraseña que pusimos en la instalación de Oracle.

2.- Creamos el usuario c##ana en el tablespace users. Como clave pondremos ana para recordarlo pero en la realidad es una práctica totalemente desaconsejada por seguridad.

```
create user c##ana identified by ana default tablespace users;
```

3.- Concedemos los roles de sistema de conexión, creación de objetos y DBA al usuario ana.

```
grant connect, resource,DBA to c##ana;
```



Elaboración propia (Uso educativo)

Ejecución del script SQL BD04_CONT_R07_02.sql

Crearemos las tablas en el usuario c##ana que acabamos de crear.

Podemos hacerlo de dos formas:

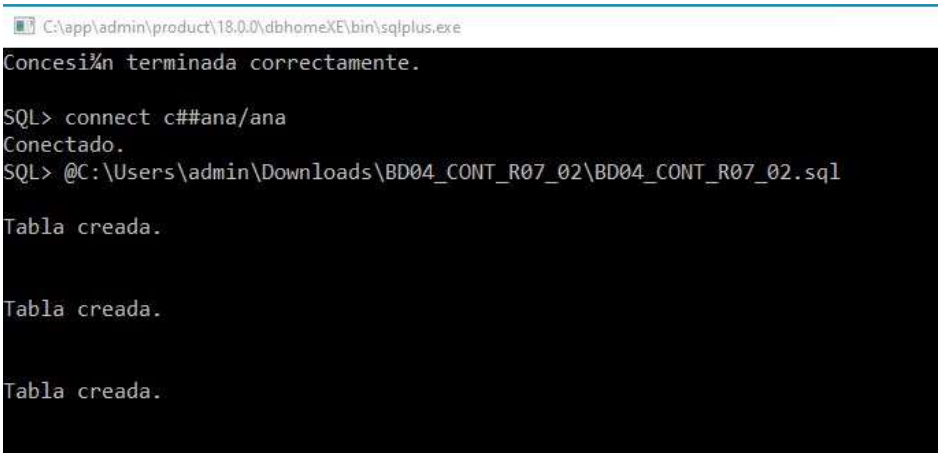
- Con **SQLPlus** escribiendo las sentencias en la **línea de comandos** de SQL
- Utilizando un **entorno gráfico**. En nuestro caso **SQLDeveloper** donde ya has creado una conexión para tu usuario en unidades anteriores.

Con SQLPlus

Desde SQLPlus conectamos con c##ana utilizando la sentencia CONNECT y ejecutamos el script anteponiendo el símbolo @ al script que nos hemos descargado con indicación de la ruta absoluta.

```
CONNECT C##ana/ana

@C:\Users\admin\Downloads\BD04_CONT_R07_02\BD04_CONT_R07_02.sql
```



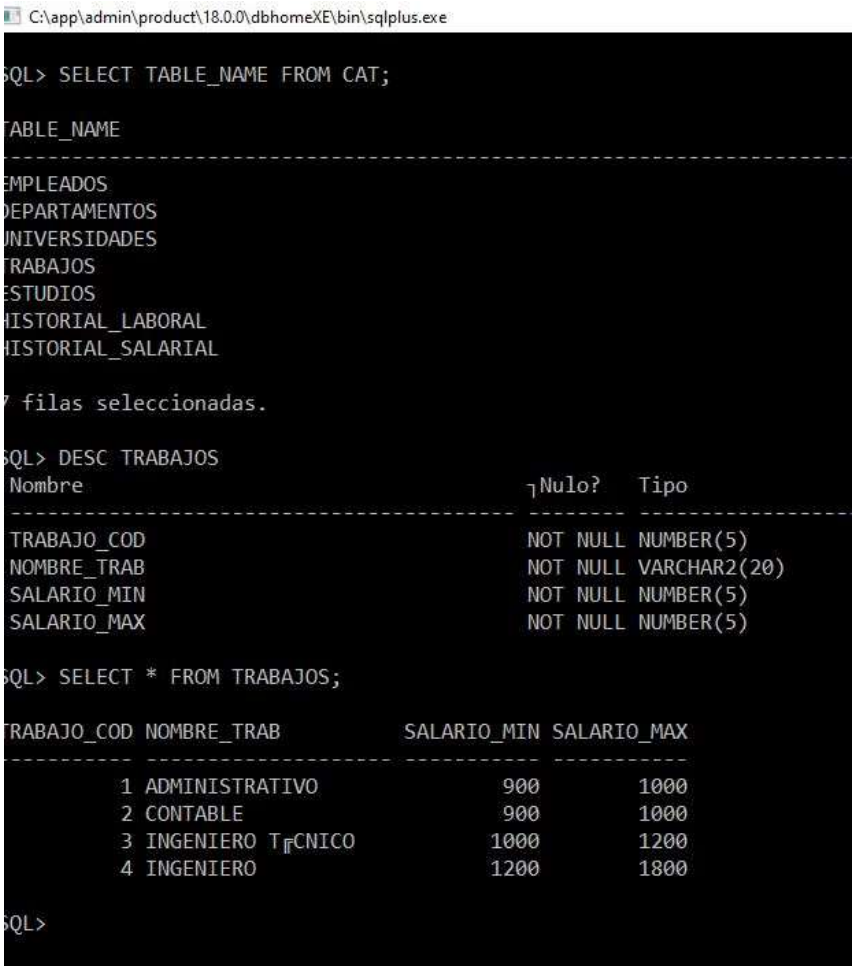
Elaboración propia (Uso educativo)

A partir de aquí ya tienes un usuario con tablas y datos incluidos para poder practicar a la vez que Ana.

Si consultamos las tablas del esquema consultando la vista del DIccionario de Datos, CAT, con la sentencia **`SELECT TABLE_NAME FROM CAT`** podremos ver los nombres de las tablas existentes en el usuario activo.

Para conocer qué columnas tiene cada tabla y su formato podemos ejecutar el comando de SQLPlus **`DESC o DESCRIBE nombre_tabla`**

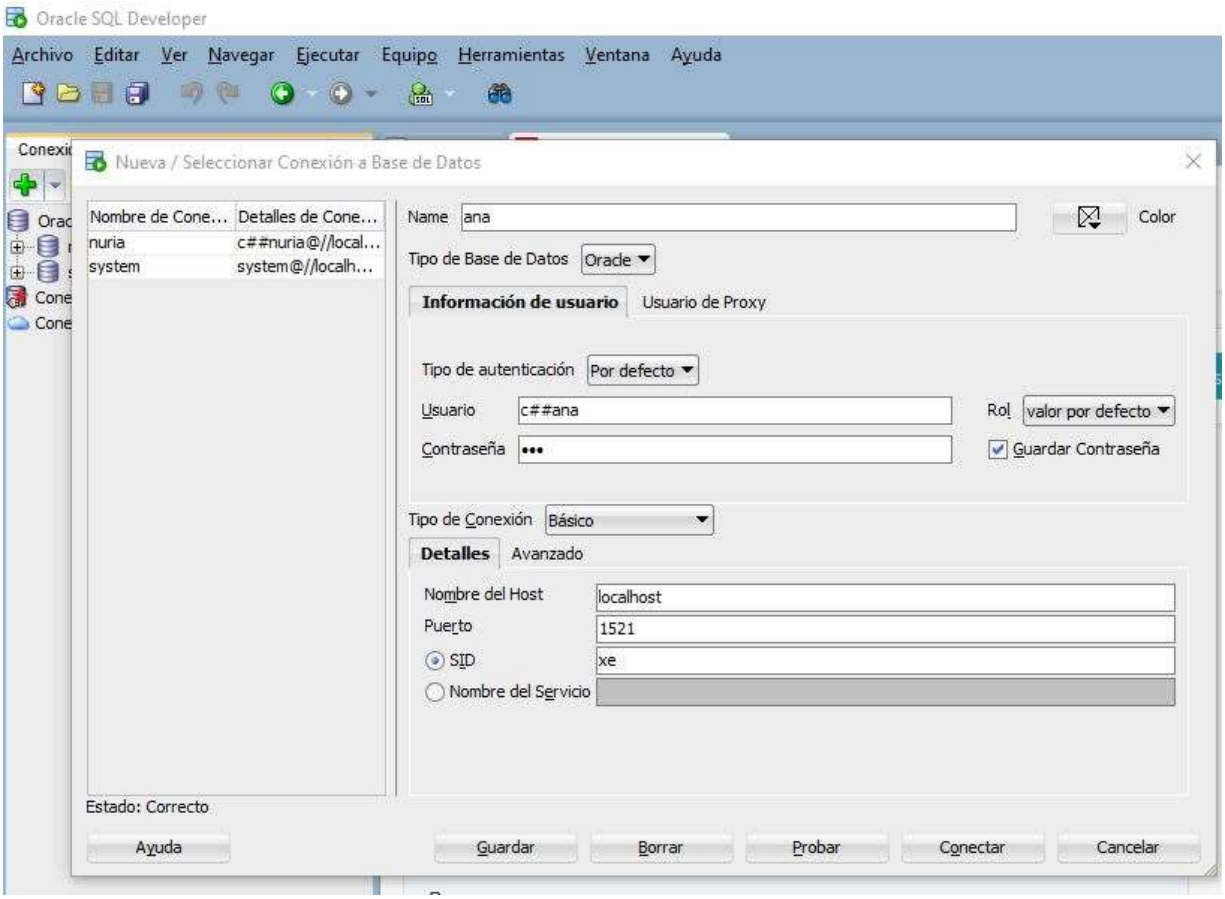
Para visualizar las filas de cualquier tabla, por ejemplo de la tabla TRABAJO, utilizaremos la sentencia **`SELECT * from TRABAJOS;`**



Elaboración propia (Uso educativo)

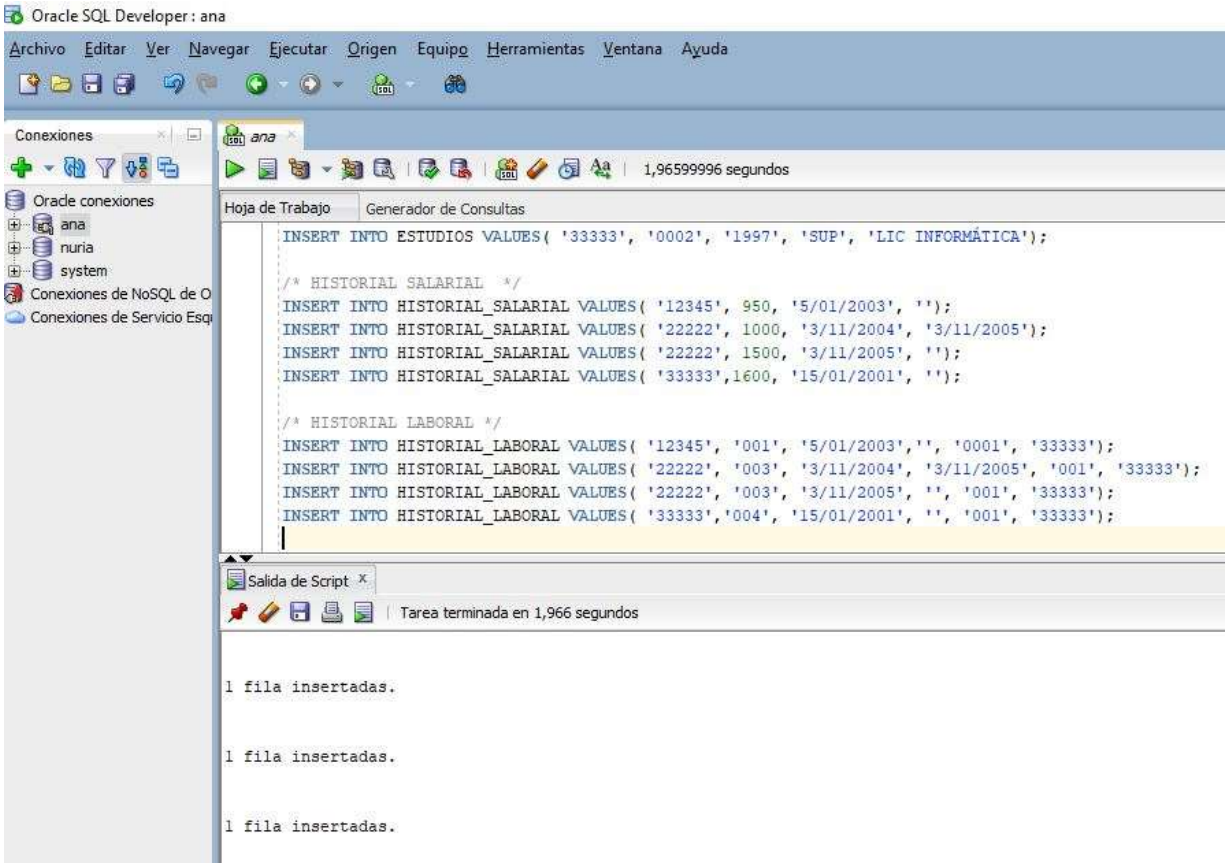
Con SQLDeveloper

Desde el botón  crea una conexión para conectar con c##ana



Elaboración propia

Abre la conexión y ejecuta el script desde el entorno gráfico **SqlDeveloper**, copiando y pegando el contenido del fichero de texto en el editor como si fuera una sola orden y ejecútalo con F5.

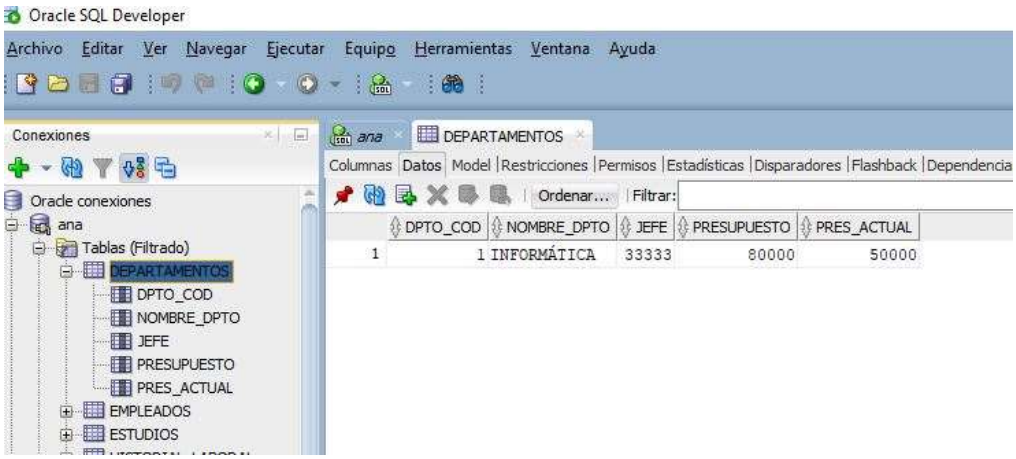


Elaboración propia (Uso educativo)

Para comprobar la correcta ejecución podemos consultar de nuevo la vista CAT con la sentencia `SELECT table_name from CAT` desde la Hoja de trabajo o pulsar en el árbol correspondiente a la conexión de ana. Si pulsamos en una tabla podremos ver las columnas que la forman.

Elaboración propia (Uso educativo)

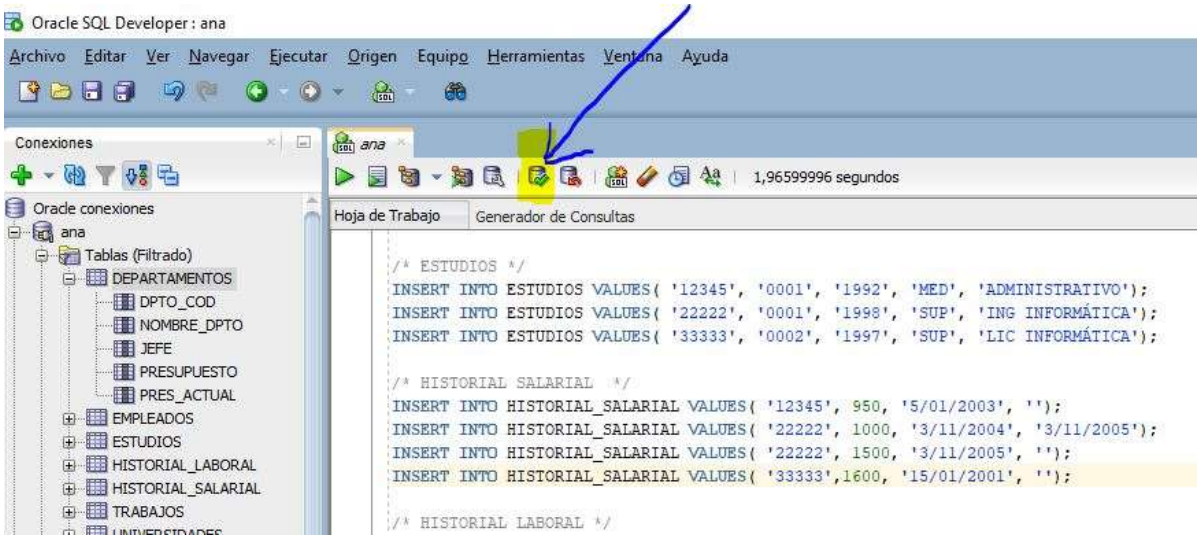
Para visualizar los datos de una tabla podemos escribir la sentencia SELECT en la hoja de trabajo o acceder a la pestaña Datos.



Elaboración propia (Uso educativo)

Para acabar, y antes de salir, debemos confirmar las operaciones para que las filas queden insertadas. Ya veremos más adelante qué significa esto.

Lo haremos pulsando el botón que aparece señalado en la siguiente pantalla.



Elaboración propia (Uso educativo)

Anexo III. Ejercicios SQL propuestos con posible solución

A programar se aprende programando por ello te proponemos 3 bloques de ejercicios para que practiques.

BLOQUE I. EJERCICIOS SQL

A continuación y sobre la base de datos que Juegos Online que has creado resuelve y prueba en tu ordenador las siguientes sentencias SQL.

1 . Nombre y apellidos de los usuarios con crédito entre 200 y 400 ordenado por crédito descendente.

2. ¿Cuántos usuarios son mujeres?

3. Nombre y apellidos de los usuarios que tiene correo en Hotmail ordenado por apellido y nombre.

4. Suma del crédito de los usuarios de la provincia de Barcelona.

5. Nombre, apellidos y fecha de nacimiento del usuario de mas edad.

6. Listado con la suma del crédito de los usuarios de cada una de las provincias ordenado por provincia.

7. Provincias en las que la suma del crédito de los usuarios es menos de 200.

8. ¿Cuál es la provincia en la que la suma de crédito de los usuarios es la mayor de todas?

9. Nombre y apellidos del usuario que ha creado cada una de las partidas indicando de qué juego son.

10. Juegos de los que no hay partida comenzada.

11. Listado de juegos de los que hay partida en funcionamiento.

12. Listado de nombre y apellidos de usuario y número de partidas en las que está jugando ordenador de mayor a menor.

En el siguiente enlace tienes una posible solución

Solucion Ejercicios SQL Anexo III (pdf - 47,85 KB)

BLOQUE II. EJERCICIOS SQL

En el siguiente enlace tienes la descripción de una base de datos llamada empresa para la gestión de los empleados de una empresa, departamentos, centros, etc.. También tienes las sentencias de creación y carga de las tablas. Tendrás que copiar y pegarlo al editor de SQL o en un fichero para ejecutarlo como un script. Tras las sentencias de creación y carga tienes una relación de ejercicios para que los realices en SQL en tu ordenador.

Creación y carga Base de datos empresa gestión empleados (pdf - 194,40 KB)

En este enlace tienes una posible solución a los requerimientos SQL anteriormente planteados.

Posible Solución Consultas (pdf - 75,92 KB)

BLOQUE III. EJERCICIOS SQL

En el siguiente enlace tienes un PDF con ejercicios SQL resueltos. Recuerda, no te limites a leer la solución, pués así solo cogerás destreza en leerlos, no en hacerlos.

Ejercicios SQL Resueltos (pdf - 420,51 KB)