

# **BASES DE DATOS CON ORACLE (+ SQL DEVELOPER)**

Nicolás Trescolí Blasco

IES Henri Matisse

# ÍNDICE

<b>INTRODUCCIÓN</b>	<b>4</b>
SOFTWARE NECESARIO	4
<b>ORACLE (SERVIDOR)</b>	<b>5</b>
INSTALACIÓN	5
<b>SQL DEVELOPER (CLIENTE)</b>	<b>7</b>
USUARIO ADMINISTRADOR Y SU CONEXIÓN	7
CREACIÓN DE USUARIOS	9
CREACIÓN DE CONEXIONES	12
OTRAS CONFIGURACIONES	13
Numeración de líneas	13
<b>PROGRAMACIÓN SQL (LDD y LMD)</b>	<b>14</b>
TIPOS DE DATOS	14
TABLAS	15
Crear una tabla	15
Eliminar una tabla	16
Ver estructura de una tabla:	16
Modificar una tabla	17
RELACIONES	20
REGISTROS	21
Insertar registros en una tabla	21
Eliminar registros de una tabla	22
Modificar registros de una tabla	22
COMENTARIOS	23
<b>CONSULTAS</b>	<b>24</b>
SELECT	24
Mostrar todos los datos	24

Mostrar campos concretos	25
Concatenar Campos (CONCAT)	26
Anidar Concatencaciones	26
Operaciones entre Campos	28
Mostrar valores únicos de la tabla	30
WHERE (Filtros)	31
Comparadores	32
Operadores Lógicos	33
Funciones específicas	33

# **INTRODUCCIÓN**

## **SOFTWARE NECESARIO**

Para crear una base de datos en Oracle, se precisa de un par de aplicaciones: una que actúa como servidor y otra que se conecta a ella como cliente. El objetivo de esta estructura es que dentro de una organización, los trabajadores (clientes) trabajen todos sobre una misma base de datos simultáneamente, accediendo a un servidor.

Para nuestros propósitos, va a ser nuestro equipo de trabajo el que va a funcionar tanto de servidor como de cliente, por ello necesitaremos instalar ambas aplicaciones en el equipo, Oracle como aplicación servidor y SQL developer como aplicación cliente.

# **ORACLE (SERVIDOR)**

## **INSTALACIÓN**

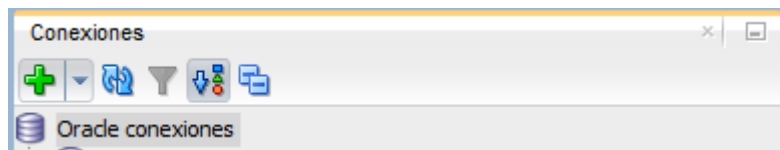
**[Curso de Oracle: Instalación de Oracle Database Express Edition \(XE\) y Oracle SQL Developer - YouTube](#)**

# SQL DEVELOPER (CLIENTE)

## USUARIO ADMINISTRADOR Y SU CONEXIÓN

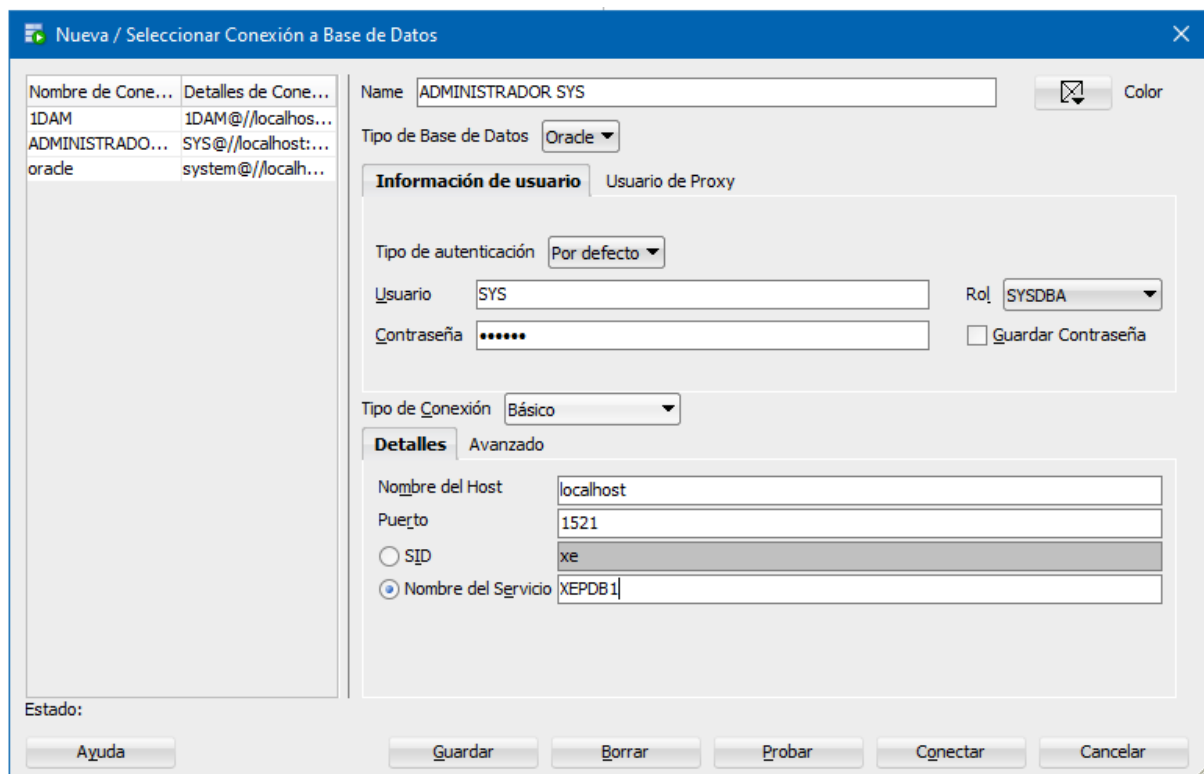
Una vez instalada la aplicación servidor de Oracle necesitamos conectar la aplicación cliente (SQL developer, en nuestro caso) con ella para poder trabajar. Para ello, el primer paso será crear una conexión con el usuario que viene por defecto como administrador en SQL developer.

1. En el panel de conexiones pinchamos en “Nueva Conexión”.



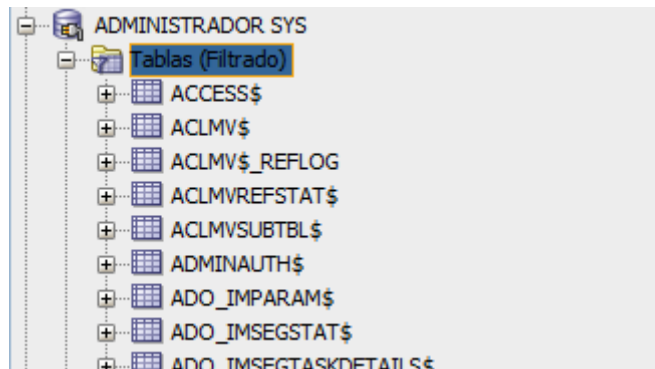
2. Le damos un nombre a la conexión (Ej.: ADMINISTRADOR SYS) y utilizamos el usuario “SYS” con su contraseña (definida cuando se instaló la aplicación servidor).

Debemos asignar el Rol “SYSDBA”. Y como Nombre del Host dejamos “localhost”, puerto “1521” y Nombre de servicio “XEPDB1”.



3. Probamos la conexión y si el Estado es “Correcto”, le damos a Conectar.

La conexión administrador contiene las tablas por defecto. Si quisiéramos trabajar con este usuario, resultaría engorroso manejar nuestras tablas mezcladas con todas las demás. Es por lo que resulta recomendable crear usuarios para trabajar.

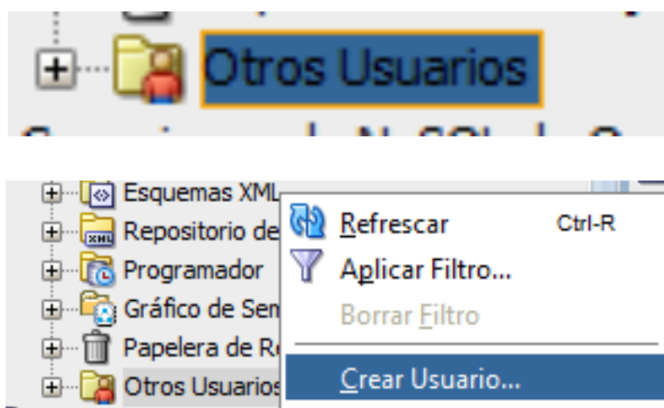


# CREACIÓN DE USUARIOS

El objetivo de crear un usuario es poder realizar proyectos desde cero, sin tener mezcladas las tablas por defecto del usuario SYSTEM con las tablas propias de nuestro proyecto.

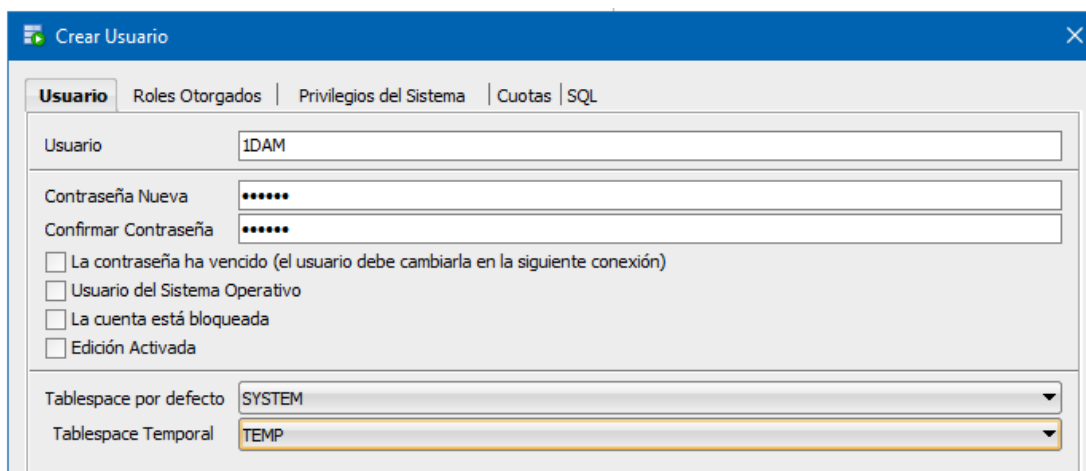
La creación de usuarios en SQL developer está limitada a los usuarios con rol de administrador. Debe configurarse **previamente** el usuario administrador para poder crear un usuario según los siguientes pasos:

1. Dentro de la conexión del usuario administrador, buscamos “Otros usuarios”. Haciendo clic con el botón derecho pinchamos en “Crear Usuario”.



2. Dentro de la ventana “Crear Usuario” rellenamos los campos que aparecen en la imagen.

Introducimos un nombre de usuario (Ej.: 1DAM) y una contraseña. Como tablespace por defecto seleccionamos “SYSTEM” y como tablespace temporal “TEMP”.

The image shows the 'Crear Usuario' dialog box in SQL Developer. The 'Usuario' tab is selected. The 'Usuario' field contains '1DAM'. The 'Contraseña Nueva' and 'Confirmar Contraseña' fields are filled with dots. There are four checkboxes: 'La contraseña ha vencido (el usuario debe cambiarla en la siguiente conexión)', 'Usuario del Sistema Operativo', 'La cuenta está bloqueada', and 'Edición Activada'. The 'Tablespace por defecto' dropdown is set to 'SYSTEM' and the 'Tablespace Temporal' dropdown is set to 'TEMP'.



- Dentro de la pestaña “Roles Otorgados” marcamos las opciones “CONNECT” y “RESOURCE” en las columnas “Otorgado” y “Valor por Defecto”, para que el usuario pueda conectarse a la aplicación servidor y utilizar sus recursos.

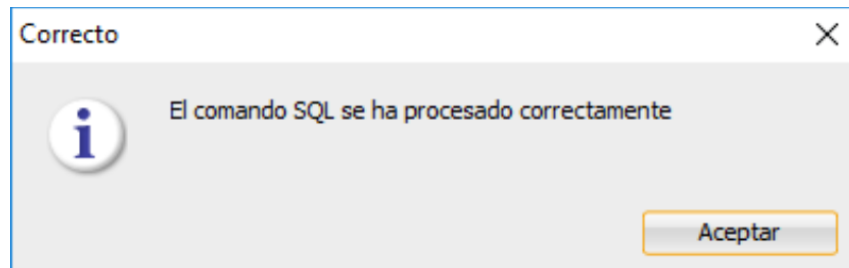
Nombre del Rol	Otorgado	Administrar	Valor por Defecto
AQ_ADMINISTRATOR_ROLE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
AQ_USER_ROLE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
AUDIT_ADMIN	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
AUDIT_VIEWER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
AUTHENTICATEDUSER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CAPTURE_ADMIN	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CDB_DBA	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CONNECT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
CTXAPP	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
DATAPATCH_ROLE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
RESOURCE	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
SCHEDULER_ADMIN	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SELECT_CATALOG_ROLE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SODA_APP	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SYSUMF_ROLE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
WM_ADMIN_ROLE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
XDBADMIN	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
XDB_SET_INVOKER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
XDB_WEBSERVICES	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
XDB_WEBSERVICES_OVER_HTTP	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
XDB_WEBSERVICES_WITH_PUBLIC	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
XS_CACHE_ADMIN	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
XS_CONNECT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
XS_NAMESPACE_ADMIN	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
XS_SESSION_ADMIN	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Nota: Es posible otorgar casi todos los demás roles, pero para nuestra actividad bastará con los anteriormente mencionados.

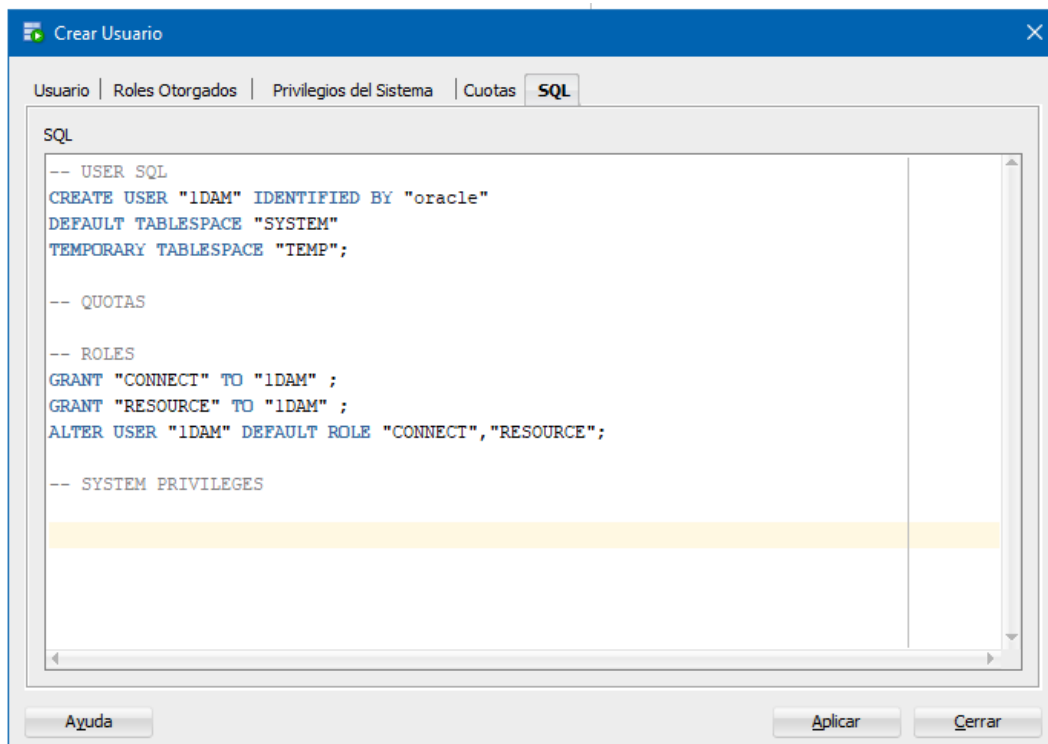
En ocasiones, si tras crear la conexión (siguiente apartado) saltara un error a la hora de introducir registros, probar a marcar también esta casilla en el apartado cuotas:

Tablespace	Ilimitado	Cuota	Unidades
NBA	<input type="checkbox"/>		
SYSAUX	<input type="checkbox"/>		
SYSTEM	<input checked="" type="checkbox"/>		
USERS	<input type="checkbox"/>		

4. Le damos a “Aplicar”.



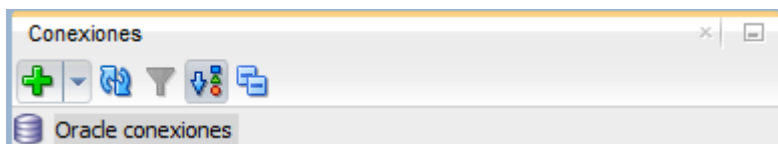
Como curiosidad, en la pestaña “SQL” es posible observar el script de SQL correspondiente a todas las acciones que hemos realizado, es decir, se pueden crear usuarios mediante código.



# CREACIÓN DE CONEXIONES

Una vez creado un usuario, necesitamos conectarlo a la aplicación servidor para poder empezar a trabajar (crear un espacio de trabajo). Para ello:

5. Le damos a “Nueva conexión” en la ventana de conexiones.



6. En la ventana rellenamos con los siguientes parámetros:

Le damos un nombre a la conexión y utilizamos el usuario con el que queramos trabajar con su contraseña.

Dejamos como Nombre del Host “localhost”, puerto “1521” y como Nombre de servicio “XEPDB1”.

A screenshot of a dialog box titled 'Nueva / Seleccionar Conexión a Base de Datos'. The dialog is divided into several sections. On the left, there is a list of existing connections with columns for 'Nombre de Cone...' and 'Detalles de Cone...'. The main area contains fields for 'Name' (set to '1DAM'), 'Tipo de Base de Datos' (set to 'Oracle'), and 'Información de usuario' (set to 'Usuario de Proxy'). Under 'Información de usuario', there are fields for 'Tipo de autenticación' (set to 'Por defecto'), 'Usuario' (set to '1DAM'), 'Contraseña' (masked with dots), and 'Rol' (set to 'valor por defecto'). Below this, there is a 'Tipo de Conexión' dropdown set to 'Básico'. The 'Detalles' section is expanded, showing fields for 'Nombre del Host' (set to 'localhost'), 'Puerto' (set to '1521'), and 'Nombre del Servicio' (set to 'XEPDB1'). At the bottom, there are buttons for 'Ayuda', 'Guardar', 'Borrar', 'Probar', 'Conectar', and 'Cancelar'.

7. Probamos la conexión y si el Estado es “Correcto”, le damos a Conectar.

Se abrirá un espacio de trabajo, lo que significa que estamos listos para empezar a trabajar. Podemos comprobar que este usuario tiene el apartado tablas limpio.

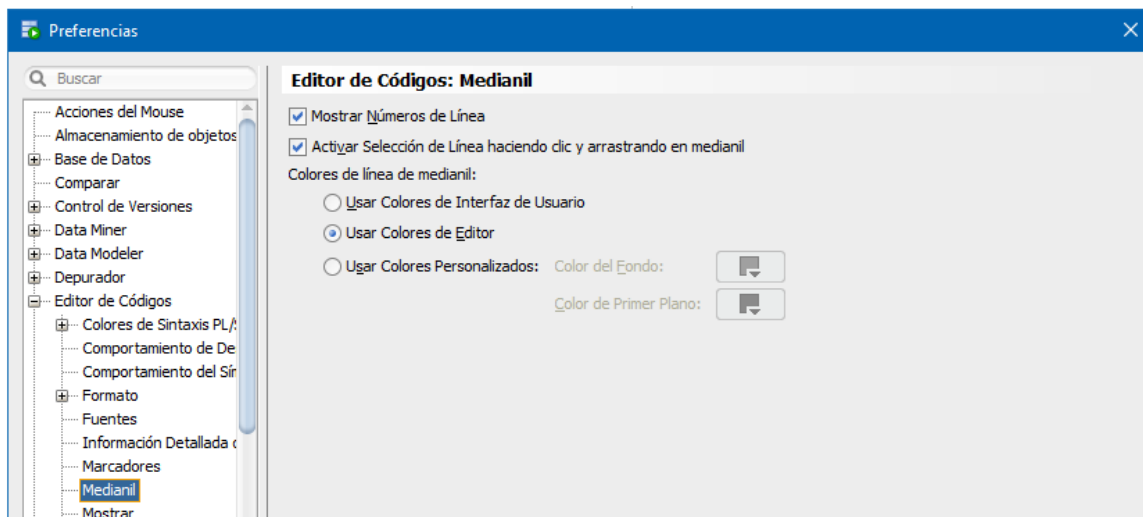
# OTRAS CONFIGURACIONES

## Numeración de líneas

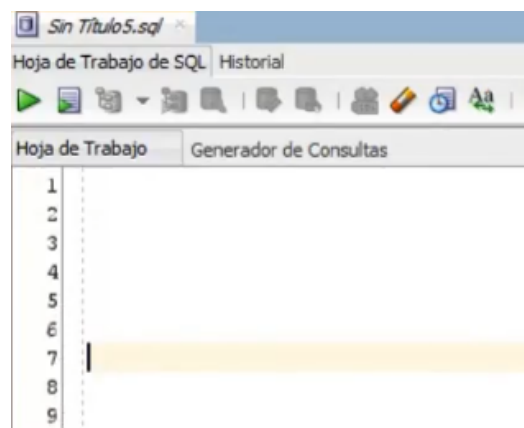
A la hora de programar en SQL, resulta cómodo tener alguna clase de referencia para identificar la línea de código en la que nos encontramos. Es por ello que en esta sección se explica como activar la numeración de filas.

En la ventana de SQL developer pinchamos en: **Herramientas** → **Preferencias**.

Luego buscamos la sección “Editor de Códigos” → “Medianil”, y marcamos la opción “Mostrar Números de Línea”.



Y listo, ya tenemos numeradas las líneas del espacio de trabajo.



-----

# PROGRAMACIÓN SQL (LDD y LMD)

## 1. TIPOS DE DATOS

De entre los muchos tipos de datos que maneja SQL, estos son los más utilizados:

**varchar2(n)** cadena de hasta **n** caracteres (*máximo 240*). Deben especificarse entre comillas simples. *Ej.: 'Mercedes Benz'.*

**number(n,p)** números con signo y punto decimal.

**n** --> número total de dígitos.

**p** --> número de posiciones decimales (*si se omite solo se pueden almacenar hasta 40 dígitos*).

*Ej.: number (5,3): 14.562 será un dato válido, pues en total hay 5 dígitos y el número de decimales es menor o igual a 3.*

(Como alternativa, para los números enteros, se puede escribir simplemente **integer** sin ningún tipo de paréntesis).

**date** formato fecha. Por defecto lleva el formato *dd/mm/aa*. Debe especificarse entre comillas simples. *Ej.: 08/12/21.*

Otros formatos:

**long** cadenas de caracteres de hasta 64 Kb. Solo se admite una por tabla. No es demasiado aconsejable, pues es incompatible con algunos métodos de consulta frecuentes.

**sysdate** fecha actual.

## 2. TABLAS

### Crear una tabla

Para crear una tabla, primero nos aseguramos de que no exista ya una tabla con el nombre que vamos a utilizar. Luego escribimos:

```
CREATE TABLE nombre_tabla (  
  Campo1 tipo_datos posible_restricción,  
  Campo2 tipo_datos posible_restricción,  
  ...  
  Campo n tipo_datos posible_restricción  
);
```

*Ejemplo:*

```
CREATE TABLE alumnos (  
  dni varchar2(10),  
  nombre varchar2(20),  
  fecha_nacimiento date,  
  curso number(1,0),  
  asignaturas_restantes number(1,0)  
);
```

### Eliminar una tabla

Para eliminar una tabla escribimos:

```
DROP TABLE nombre_tabla;
```

*Ejemplo.:*

```
drop table alumnos;
```

# RESTRICCIONES sobre los campos de una tabla

Al crear una tabla, nos podemos encontrar, de forma básica, los siguientes tipos de restricciones sobre los campos:

- **PRIMARY KEY**: Reservada para el o los campos que actúen como clave primaria.

En el caso de una tabla con **PRIMARY KEY COMPUESTA**, la restricción de primary key **NO** se debe indicar al lado del campo, sino al final, como se muestra a continuación:

```
CREATE TABLE nombre_tabla (  
  Campo1 tipo_datos,  
  Campo2 tipo_datos,  
  ...  
  Campo n tipo_datos posible_restricción,  
  PRIMARY KEY(Campo1, Campo2, ...)  
);
```

- **NOT NULL**: Indica que, a la hora de introducir un registro, será obligatorio rellenar ese campo.
- **NULL**: Es lo mismo que no poner nada. Indica que, a la hora de introducir un registro, el campo se puede dejar vacío.

*Ejemplo:*

```
CREATE TABLE alumnos (  
  dni varchar2(10),  
  nombre varchar2(20) ,  
  fecha_nacimiento date not null,  
  curso number(1,0),  
  asignaturas_restantes number(1,0) null,  
  PRIMARY KEY(dni, nombre)  
);
```

## Ver estructura de una tabla:

Para ver las propiedades de los campos de una tabla escribimos el comando:

```
DESCRIBE nombre_tabla;
```

*Ejemplo.:*

```
describe alumnos;
```

Resultado:

<i>Nombre</i>	<i>¿Nulo?</i>	<i>Tipo</i>
-----	-----	-----
<i>DNI</i>	<i>NOT NULL</i>	<i>VARCHAR2(10)</i>
<i>NOMBRE</i>	<i>NOT NULL</i>	<i>VARCHAR2(20)</i>
<i>FECHA_NACIMIENTO</i>		<i>DATE</i>
<i>CURSO</i>		<i>NUMBER(1)</i>
<i>ASIGNATURAS_RESTANTES</i>		<i>NUMBER(1)</i>



## Modificar una tabla

En ocasiones, necesitamos retocar algún aspecto de una tabla. Para no tener que borrar y crear la tabla cada vez, pues esta podría contener registros o referencias de otras tablas, puede utilizarse el lenguaje de modificación de datos: “**ALTER TABLE**”.

Cambiamos el nombre de una tabla así:

```
ALTER TABLE nombre_tabla RENAME TO nuevo_nombre_tabla;
```

Y cambiamos el nombre de un campo de forma similar:

```
ALTER TABLE nombre_tabla RENAME COLUMN nombre_campo TO nuevo_nombre;
```

Añadir un campo a una tabla:

```
ALTER TABLE nombre_tabla ADD campo_nuevo tipo_datos posible_restricción;
```

*Ejemplo:*

```
ALTER TABLE alumnos ADD nº_hermanos integer not null;
```

Eliminar un campo de tabla:

```
ALTER TABLE nombre_tabla DROP COLUMN nombre_campo;
```

*Ejemplo:*

```
ALTER TABLE alumnos DROP COLUMN nº_hermanos;
```

Añadir restricción a un campo:

```
ALTER TABLE nombre_tabla MODIFY nombre_campo posible_restricción;
```

*Ejemplo:*

```
ALTER TABLE alumnos MODIFY dni primary key;
```

Cambiar el tipo de variable de un campo:

```
ALTER TABLE nombre_tabla MODIFY nombre_campo tipo_dato posible_restricción;
```

*Ejemplo:*

```
ALTER TABLE alumnos MODIFY dni varchar2(10) null;
```

Nota: Este es uno de los pocos casos en que la sentencia **null** resulta útil, pues puede utilizarse para sustituir la anterior restricción, eliminándola.

Añadir clave primaria (versión 2):

```
ALTER TABLE nombre_tabla ADD PRIMARY KEY (nombre_campo);
```

*Ejemplo:*

```
ALTER TABLE alumnos ADD PRIMARY KEY (dni);
```

Añadir CLAVE AJENA:

```
ALTER TABLE nombre_tabla ADD FOREIGN KEY (campo_n) REFERENCES tabla_ref  
(campo_ref);
```

*Ejemplo:*

```
ALTER TABLE alumnos ADD FOREIGN KEY (asignatura_pendiente) REFERENCES asignatura  
(cod_asignatura);
```

Tal vez sea este último el recurso de modificación más utilizado, pues la creación de nuevas tablas y el hecho de necesitar establecer relación con las ya creadas, obliga a añadir claves ajenas a posteriori a tablas ya creadas.

Cosas a tener en cuenta:

- Se puede cambiar el tipo de un atributo(columna), utilizando la sentencia ALTER solo si ese atributo tiene valor null en todas los registros de la tabla.
- A un atributo se le puede añadir la restricción NOT NULL, únicamente si todas las tuplas tienen en ese atributo un valor distinto de NULL.

Nota: En el anterior apartado “Modificar una tabla” aparecen comandos aún no explicados, pero desarrollados más adelante en esta guía.

### 3. RELACIONES

Para establecer relaciones entre tablas, se siguen las normas del modelo relacional. De modo que para relacionar dos tablas entre sí, se deberán crear campos que funcionen como CLAVES AJENAS y referenciarlos a la tabla con la que están relacionados.

Especificamos claves ajenas con la siguiente sintaxis:

```
FOREIGN KEY (campo_n) REFERENCES tabla_ref (campo_ref);
```

*Ejemplo:*

```
CREATE TABLE ciclista (  
  dorsal INTEGER PRIMARY KEY,  
  nombre VARCHAR(30) NOT NULL  
);  
  
CREATE TABLE etapa (  
  netapa INTEGER PRIMARY KEY,  
  km INTEGER,  
  dorsal2 INTEGER,  
  FOREIGN KEY (dorsal2) REFERENCES ciclista (dorsal)  
);
```

Aunque evidente, es importante recordar que no se puede hacer referencia a una tabla que no se haya creado previamente.

De la misma forma, si quisiéramos eliminar una tabla que está siendo referenciada por otra, se nos advertirá que no es posible, ya que primero debemos eliminar la relación en la otra tabla.

## 4. REGISTROS

### Insertar registros en una tabla

Cada vez que haya que insertar un registro en una tabla lo haremos de la siguiente forma:

```
INSERT INTO nombre_tabla (campo_1, campo_2..., campo_n)
VALUES ('...', '...', '...');
```

*Ejemplo.:*

```
insert into actores (dni, nombre, telefono)
values ('20969932j', 'Carmen', '767472456');
insert into actores (dni, nombre, telefono)
values ('21349733p', 'Natalia', '555555555');
```

Es importante saber que en el paréntesis de **INSERT INTO** (*campo1, campo2..., campo n*), sólo deben introducirse los campos que van a ser rellenados.

Además, el orden en que se vayan a introducir DEBE coincidir con lo que introduzcamos en values.

Esto es, si el paréntesis superior fuera: (*campo3, campo2, campo1 campo5*), entonces el en el paréntesis de **VALUES** el orden en que se especifica el valor de cada campo debe estar escrito en el mismo orden.

**NOTA:** Al introducir un registro en una tabla, al rellenar un campo que sea FOREIGN KEY, el dato que se introduzca debe ser el valor de clave primaria al que hace referencia (este debe haberse introducido previamente como registro en la otra tabla).

## Eliminar registros de una tabla

Para eliminar registro se utiliza el comando **DELETE FROM**. Sin embargo, cuidado; si solo se escribe esto, se borran TODOS los registros de la tabla.

```
DELETE FROM nombre_tabla;
```

Si lo que queremos es eliminar solo algunos registros, debemos acompañar el comando con la sentencia **WHERE** que veremos más adelante. De todos modos, quedaría así:

```
DELETE FROM nombre_tabla WHERE campo_n='...';
```

*Ejemplo.:*

```
DELETE FROM actores WHERE nombre='Natalia';
```

## Modificar registros de una tabla

Para modificar un dato de un registro se utiliza el comando **UPDATE**. Sin embargo, cuidado; si solo se escribe esto, se modifican TODOS los registros de la tabla.

```
UPDATE nombre_tabla SET campo_n=nuevo_dato;
```

*Ejemplo.:*

```
UPDATE actores SET premios='5';
```

Si lo que queremos es modificar sólo algunos registros, debemos acompañar el comando con la sentencia **WHERE** quedando así:

```
UPDATE nombre_tabla SET campo_n=nuevo_dato WHERE campo_m='...';
```

*Ejemplo.:*

```
UPDATE actores SET premios=premios*2 WHERE dni='239457612P';
```

## 5. COMENTARIOS

Para añadir comentarios dentro del código con el fin de aclarar al resto de programadores las acciones que se van realizando, es posible añadir comentarios solo visibles para estos encerrándolos entre: */\*.....\*/*.

O bien en la misma línea escribiendo tras dos guiones: *--*.

*Ejemplo.:*

```
/*Esto es un comentario dentro del lenguaje SQL*/  
-- Esto es otro comentario dentro del lenguaje SQL
```

-----

# CONSULTAS

Una vez creada una determinada base de datos, es importante conocer cómo se puede acceder a la información y cómo filtrar solo los datos que nos interesan. Para ello utilizaremos los comandos para consultas.

## SELECT

La sentencia SELECT permite seleccionar aquellos datos que nos interesan mediante la siguiente sintaxis:

```
SELECT ..... FROM nombre_tabla ..... ;
```

## Mostrar todos los datos

En lenguaje SQL, el asterisco \*, significa todo. De modo que si quisiéramos mostrar todos los datos de una tabla en concreto, escribiríamos:

```
SELECT * FROM nombre_tabla;
```

*Ejemplo:*

```
SELECT * FROM actores;
```

Resultado:

### ACTORES

dni	nombre	fecha_nac	teléfono	premios	ciudad
20969932j	Carmen	04/06/1994	767472456	4	Madrid
21349733p	Natalia	23/09/1995	555555555	6	Valencia
29879837k	Marcos	15/02/2003	876962983	1	Valencia
98769823y	Pedro	06/06/1998	675456252	8	Barcelona

*(Se muestran todos los campos y todas las tuplas de la tabla actores)*



## Mostrar campos concretos

Para mostrar sólo ciertos campos de las tablas, escribiremos los nombres de dichos campos separados por comas:

```
SELECT campo_1, campo_2, ..., campo_n FROM nombre_tabla;
```

*Ejemplo.:*

```
SELECT dni, nombre FROM actores;
```

Resultado:

**ACTORES**

dni	nombre
20969932j	Carmen
21349733p	Natalia
29879837k	Marcos
98769823y	Pedro

*(Se muestran solo los campos seleccionados y todas las tuplas de la tabla actores)*

## Concatenar Campos (CONCAT)

Para mostrar combinados en un solo campo varios campos de una tabla escribiremos:

```
SELECT CONCAT(campo_1, campo_2) FROM nombre_tabla;
```

Nótese que SÓLO se ha escrito campo\_1 y campo\_2, esto es porque dentro de un CONCAT solamente pueden incluirse 2 campos cualquiera, pero no más de 2.

*Ejemplo.:*

```
SELECT CONCAT(dni, nombre) FROM actores;
```

Resultado:

**ACTORES**

CONCAT(dni,nombre)
20969932jCarmen
21349733pNatalia
29879837kMarcos
98769823yPedro

*(Se muestran los campos seleccionados concatenados sin ninguna separación)*

## Anidar Concatencaciones

Esta forma de mostrar los datos parece engorrosa, es por ello que normalmente se suele añadir algún elemento separados como guiones (-). Además, tal vez queramos concatenar más de 2 campos. Esto se puede conseguir ANIDANDO funciones concat entre sí:

```
SELECT CONCAT(CONCAT(campo_1, campo_2), campo_3) FROM nombre_tabla;
```

*Ejemplo.:*

```
SELECT CONCAT(CONCAT(dni, ' - '), nombre) FROM nombre_tabla;
```

Resultado:

**ACTORES**

<b>CONCAT(CONCAT(dni, ' - '), nombre)</b>
20969932j - Carmen
21349733p - Natalia
29879837k - Marcos
98769823y - Pedro

*(Se muestran los campos seleccionados concatenados con un - como separación)*

*Ejemplo.:*

```
SELECT CONCAT(CONCAT(CONCAT(CONCAT(dni, ' - '), nombre), ' - '), fecha_nac) FROM actores;
```

Resultado:

**ACTORES**

<b>CONCAT(CONCAT(CONCAT(CONCAT(dni, ' - '), nombre), ' - '), fecha_nac)</b>
20969932j - Carmen - 04/06/1994
21349733p - Natalia - 23/09/1995
29879837k - Marcos - 15/02/2003
98769823y - Pedro - 06/06/1998

*(Se muestran los campos seleccionados concatenados con un - como separación)*

## Operaciones entre Campos

Para los campos con formato **number** y **date** es posible realizar operaciones durante la consulta:

```
SELECT campo_1, campo_2 + campo_3, ..., campo_n FROM nombre_tabla;
```

Sin embargo, no todas las operaciones están permitidas.

- Para formato **date** se permite:

→ **SELECT** fecha\_1 - fecha\_2 **FROM** nombre\_tabla;

*Indica el número de días transcurridos entre dos fechas.*

→ **SELECT** fecha\_1 + 5 **FROM** nombre\_tabla;

*Suma (o resta) 5 días a fecha\_1. En este caso, el 5 puede introducirse manualmente o utilizar un campo con formato **number** de la propia tabla.*

- Para formato **number** se permite:

→ **SELECT** number\_1 + number\_2 **FROM** nombre\_tabla;

*Suma (+), resta (-), multiplica (\*), divide (/), resto de la división entera (%) ... number\_1 y number\_2.*

*También puede introducirse manualmente, por ejemplo: number\_1\*60.*

*Ejemplo.: Mostrar edad de los actores*

**SELECT** dni, nombre, **(sysdate - fecha\_nac)/365**, teléfono **FROM** actores;

Resultado:

**ACTORES**

dni	nombre	(sysdate - fecha_nac)/365	teléfono
20969932j	Carmen	<b>27</b> ,34623723476237	767472456
21349733p	Natalia	<b>26</b> ,12341666121345	555555555
29879837k	Marcos	<b>18</b> ,34571347137133	876962983
98769823y	Pedro	<b>23</b> ,472485624578245	675456252

*(Se muestra el campo fecha\_nac con las operaciones indicadas [**SYSDATE** es la fecha actual]. Aún no se ha explicado cómo truncar números, por eso los decimales; lo veremos más adelante).*

Al igual que en las matemáticas, SQL sigue el orden jerárquico de las operaciones. Sin embargo, es posible alterarlo a gusto mediante el uso de **paréntesis ( )**.

## Mostrar valores únicos de la tabla

Para mostrar aquellos valores que son únicos en una tabla, utilizamos la sentencia **DISTINCT**, que permite especificar qué campos deben mostrarse cuyos valores son únicos. Lo escribimos así:

```
SELECT DISTINCT campo_1, campo_2, ..., campo_n FROM nombre_tabla;
```

*Ejemplo.:* Mostrar edad de los actores

```
SELECT DISTINCT Ciudad, FROM actores;
```

Resultado:

**ACTORES**

Ciudad
Madrid
Valencia
Barcelona

*(Se muestra UNA SOLA VEZ, cada uno de los valores de Ciudad almacenadas entre todas las tuplas de la tabla).*

## WHERE (Filtros)

Cuando realizamos consultas en SQL, normalmente no buscamos ver todos los registros que contiene una tabla, es por ello, que podemos mostrar solo las tuplas que cumplan ciertas condiciones mediante la sentencia **WHERE**.

```
SELECT campos FROM nombre_tabla WHERE condiciones;
```

*Ejemplo.:*

```
SELECT * FROM actores WHERE ciudad='Valencia';
```

Resultado:

**ACTORES**

dni	nombre	fecha_nac	teléfono	premios	ciudad
21349733p	Natalia	23/09/1995	555555555	6	Valencia
29879837k	Marcos	15/02/2003	876962983	1	Valencia

*(Se muestran todas las tuplas de la tabla actores cuyo valor para el campo 'Ciudad' es 'Valencia', con todos los campos de la tabla.)*

Las sentencias WHERE suelen ir acompañadas de otras sentencia y/o operadores que permiten acotar mejor la consulta realizada. En los siguientes apartados se muestran los operadores de comparación y los operadores lógicos, además de funciones específicas como IN y BETWEEN propias de oracle.

## Comparadores

Los comparadores son caracteres reservados usados por los lenguajes de programación, en general, para comprobar condiciones. Los resultados de estas comparaciones sólo pueden ser dos: VERDADERO (TRUE) o FALSO (FALSE).

En el caso de SQL, utilizamos los comparadores para las consultas, para mostrar las tuplas en que algunos de los campos pertenecen a un determinado rango de valores.

Los operadores de comparación son los siguientes:

Comparador	Explicación
=	Comprueba si x es igual a y
<	Comprueba si x es menor que y
>	Comprueba si x es mayor que y
<=	Comprueba si x es menor o igual que y
>=	Comprueba si x es mayor o igual que y
<> ó !=	Comprueba si x es distinto de y

*Ejemplo.:* Mostrar actores con 4 o más premios

**SELECT** dni, nombre, premios **FROM** actores **WHERE** premios>='4';

Resultado:

**ACTORES**

dni	nombre	premios
20969932j	Carmen	4
21349733p	Natalia	6



## Operadores Lógicos

Como en la mayoría de lenguajes de programación, SQL incluye los operadores lógicos (**or**, **and...**). Su función básicamente es poder comprobar varias condiciones a la vez a la hora de realizar consultas con WHERE.

Estos son algunos operadores lógicos:

Operador lógico	Explicación	Ejemplo
<b>and</b>	Resulta TRUE solo si se cumplen ambas condiciones.	<i>Muestra los actores franceses con menos de 5 premios.</i>  <b>SELECT</b> nombre <b>FROM</b> actores <b>WHERE</b> país='Francia' <b>and</b> premios<'5';
<b>or</b>	Resulta TRUE si se cumple al menos una de las condiciones.	<i>Muestra los actores franceses e ingleses.</i>  <b>SELECT</b> nombre <b>FROM</b> actores <b>WHERE</b> país='Francia' <b>or</b> país='UK';
<b>not</b>	Resulta TRUE si no se cumple la condición.	<i>Muestra los actores donde se haya rellenado el campo premios.</i>  <b>SELECT</b> nombre <b>FROM</b> actores <b>WHERE</b> premios <b>not</b> null;

## Funciones específicas

Cada SGBD incorpora sus propias funciones para consultas que, en general, no coinciden con las de otros SGBD. Las funciones de Oracle pueden consultarse en su sitio web. A continuación se muestran algunas de las más usuales.

### IS (NULL o NOT NULL)

Utilizamos el operador **IS** para seleccionar registros que contengan algún campo vacío (NULL) o en el que se haya insertado algún valor (NOT NULL)

```
SELECT campos FROM nombre_tabla WHERE nombre_columna IS NULL;  
SELECT campos FROM nombre_tabla WHERE nombre_columna IS NOT NULL;
```

### IN (Conjunto de valores)

El operador de pertenencia a conjuntos **IN** permite comprobar si una columna tiene un valor igual a cualquier de los que están incluidos dentro del paréntesis:

```
SELECT campos FROM nombre_tabla WHERE nombre_columna IN (v1, ..., v2);
```

*Ejemplo.:*

```
SELECT dni, nombre, Ciudad FROM actores WHERE ciudad IN (Barcelona, Madrid);
```

Resultado:

#### ACTORES

dni	nombre	ciudad
20969932j	Carmen	Madrid
98769823y	Pedro	Barcelona

## BETWEEN (Rango de valores)

Utilizamos el operador **BETWEEN** para seleccionar registros que estén incluidos en un rango. La sintaxis es la siguiente:

```
SELECT campos FROM nombre_tabla WHERE nombre_columna BETWEEN v1 AND v2;
```

*Ejemplo.:*

```
SELECT * FROM actores WHERE premios BETWEEN 2 AND 6;
```

Resultado:

### ACTORES

dni	nombre	fecha_nac	teléfono	premios	ciudad
20969932j	Carmen	04/06/1994	767472456	4	Madrid
21349733p	Natalia	23/09/1995	555555555	6	Valencia

*Nótese que los valores 2 y 6 de la sentencia también se incluyen en la búsqueda, es decir, es equivalente a escribir:*

```
SELECT * FROM actores WHERE premios>='2' AND premios<='6';
```

## ROWNUM (Límite de número de registros)

ROWNUM sirve para mostrar solo los primeros *n* registros de una tabla.

```
SELECT campos FROM nombre_tabla WHERE ROWNUM <= número_registros;
```

*Ejemplo.:*

```
SELECT * FROM actores WHERE ROWNUM <= 3;
```

Resultado:

### ACTORES

dni	nombre	fecha_nac	teléfono	premios	ciudad
20969932j	Carmen	04/06/1994	767472456	4	Madrid
21349733p	Natalia	23/09/1995	555555555	6	Valencia
29879837k	Marcos	15/02/2003	876962983	1	Valencia

*(Se muestran solo los 3 primeros registros)*

## Consultas de resumen

En los siguientes apartados se muestra como extraer información sobre estadísticas de los registros de una base de datos. Para ello se utilizan las funciones siguientes:

Función	Explicación	Ejemplo
<b>SUM (columna_n)</b>	Suma los valores de la <b>columna_n</b> de los registros de una tabla.	<i>Muestra el total de premios ganados por los actores.</i>  <b>SELECT</b> sum(premios) <b>FROM</b> actores;
<b>AVG (columna_n)</b>	Calcula la media de los valores de la <b>columna_n</b> de los registros de una tabla.	<i>Muestra la media de premios ganados por los actores.</i>  <b>SELECT</b> avg(premios) <b>FROM</b> actores;
<b>MIN (columna_n)</b>	Obtiene el mínimo de los valores de la <b>columna_n</b> de los registros de una tabla.	<i>¿Cuántos premios tiene el actor que menos premios tiene?</i>  <b>SELECT</b> min(premios) <b>FROM</b> actores;
<b>MAX (columna_n)</b>	Obtiene el máximo de los valores de la <b>columna_n</b> de los registros de una tabla.	<i>¿Cuántos premios tiene el actor que más premios tiene?</i>  <b>SELECT</b> max(premios) <b>FROM</b> actores;
<b>COUNT (columna_n)</b>	Cuenta los valores de la <b>columna_n</b> de los registros de una tabla (excepto los nulos).	<i>¿Cuántos actores han rellenado el campo 'ciudad'?</i>  <b>SELECT</b> count(ciudad) <b>FROM</b> actores;

Función	Explicación	Ejemplo
<b>COUNT (*)</b>	Cuenta el número de registros de una tabla.	<i>¿Cuántos actores hay en la tabla actores?</i>  <b>SELECT</b> count(*) <b>FROM</b> actores;