

Lenguaje de Definición de Datos DDL

Javier Ivorra – javierivorra@iessanvicente.com

IES San Vicente

1. Introducción

En este tema estudiaremos el lenguaje de definición de datos (DDL) empleado en las bases de datos Oracle, en concreto el utilizado en Oracle Database 18c. Este lenguaje nos permite crear y definir la estructura de una base de datos.

Empezaremos tratando algunos conceptos básicos en las bases de datos Oracle, así como los primeros pasos que deberemos seguir para trabajar con un servidor Oracle Database.

2. Oracle

Oracle corporation es una multinacional americana con sede en Redwood Shores, California. Está especializada principalmente en el desarrollo y comercialización de aplicaciones de bases de datos, sistemas en la nube y software para grandes empresas. En el año 2018 fué la tercera mayor compañía de software por ingresos.

2.1. Bases de datos de Oracle

Las últimas versiones que han ido apareciendo en el mercado y que se suelen encontrar fácilmente en las empresas serían:

- Release 10g (2003):
- Release 11g (2007):
- Release 12c (2013):
- Release 18c (2018):

Oracle corporation ha adquirido y desarrollado las siguientes tecnologías de bases de datos:

- Berkley DB, que ofrece manejo de bases de datos embebidas
- Oracle Rdb, un sistema de bases de datos relacionales que se puede ejecutar en plataformas OpenVMS.
- TimesTen, que optimiza las operaciones de las bases de datos en memoria
- Oracle Essbase, que continúa con la tradición de Hyperion Essbase en el manejo de bases de datos multidimensionales
- MySQL, un sistema de bases de datos relacionales, bajo licencia GNU
- Oracle NoSQL Database

1.1. Instalación

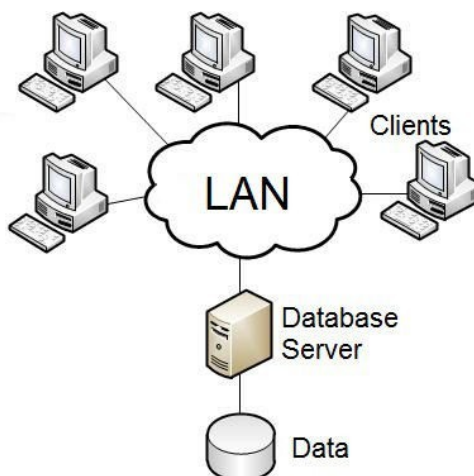
Utilizaremos Oracle Database 18c Express Edition (XE), junto con la herramienta SQL Developer. La instalación y puesta en marcha de ambas aplicaciones se ha tratado en un documento aparte.

1.2. Arquitectura Cliente/Servidor

En la arquitectura cliente/servidor de Oracle nos encontramos con dos partes claramente diferenciadas, lo que llamamos el front-end o parte cliente y el back-end o parte servidor. El cliente ejecuta una aplicación que accede a la información de la base de datos, a través de peticiones que son procesadas en el servidor, quien devolverá una respuesta al cliente con los resultados de su petición.

Aunque la aplicación cliente y servidor se pueden ejecutar en la misma máquina, siempre es más eficiente si

trabajamos con diferentes máquinas conectadas en red



En nuestro caso concreto para este curso, lo que llamamos parte servidor estaría formado por Oracle Database 18c, mientras que el cliente sería la aplicación SQL Developer.

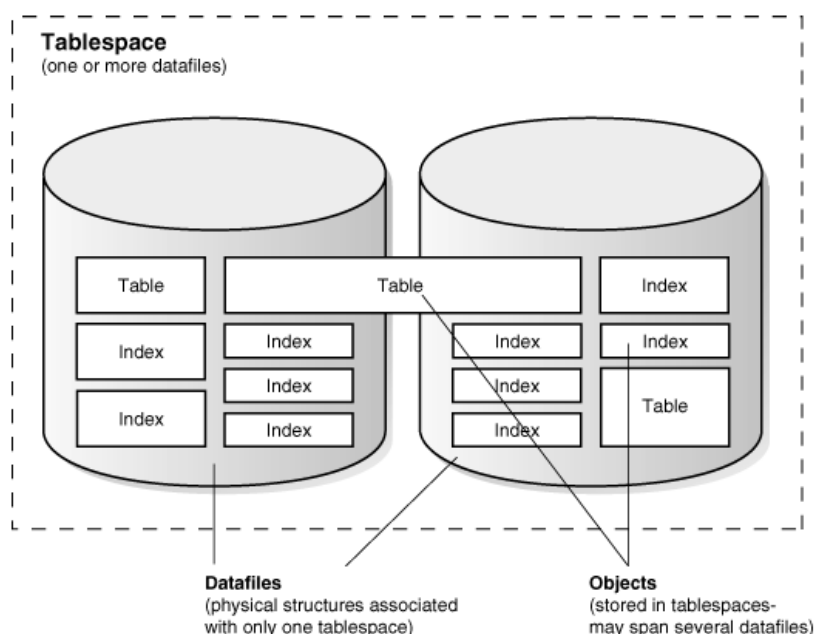
1.3. Instancia de base de datos

Una instancia de Oracle está formada por el conjunto de procesos y estructuras de datos en memoria que el servidor de base de datos necesita cuando está en funcionamiento. Estos procesos proporcionan el mecanismo necesario para acceder a un conjunto de archivos físicos donde se almacena la información de la base de datos.

Es posible tener múltiples instancias de Oracle 18c ejecutándose en un mismo servidor, con espacio de memoria diferente para dar soporte a cada una de ellas. Una instancia de Oracle se asigna siempre a una única base de datos.

1.4. Tablespaces

Oracle almacena los datos de manera lógica en tablespaces y físicamente en ficheros asociados con su correspondiente tablespace. La siguiente imagen muestra esta relación



Bases de datos, tablespaces y ficheros de datos están estrechamente relacionados, pero presentan importantes

diferencias:

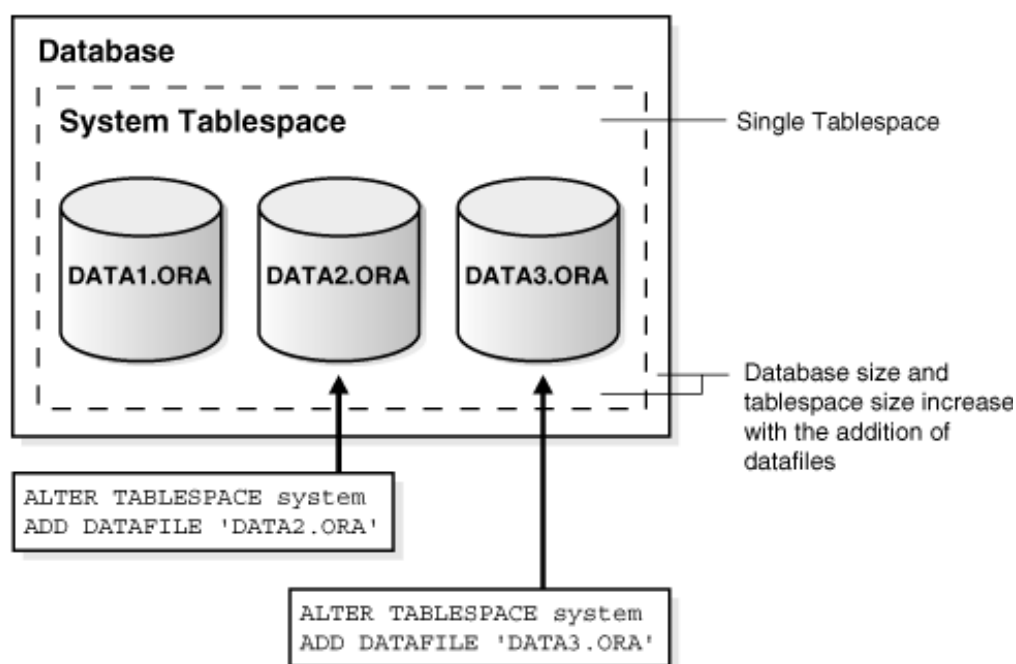
- Una base de datos de Oracle está formada de una o más unidades lógicas de almacenamiento, llamadas tablespaces que en su conjunto almacenan toda la información de la base de datos
- Cada tablespace de una base de datos contiene uno o más archivos de datos, con una estructura física conforme al sistema operativo donde Oracle se está ejecutando
- La información de la base de datos se almacena en los archivos de datos que forman cada tablespace. Por ejemplo, la base de datos más simple de Oracle, tendría un único tablespace con un único archivo de datos. Otra base de datos podría tener cuatro tablespaces con dos archivos de datos cada uno, con un total de ocho archivos

Tablas, índices y otros objetos se colocan dentro de tablespaces, por lo que el primer paso para introducir información en una base de datos nueva e independiente del resto, debería ser crear un primer tablespace y asociarlo a uno o varios archivos físicos.

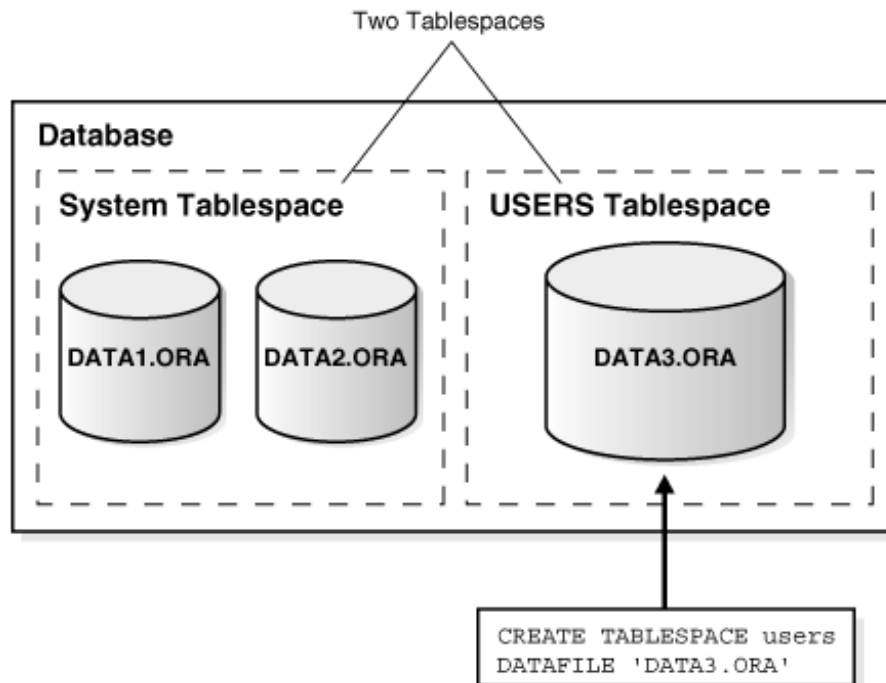
El tamaño de un tablespace es el tamaño de los archivos de datos que lo forman. El tamaño de una base de datos es la suma de los tamaños de todos los tablespaces que la forman.

Podemos ampliar el tamaño reservado para una base de datos de Oracle de tres modos:

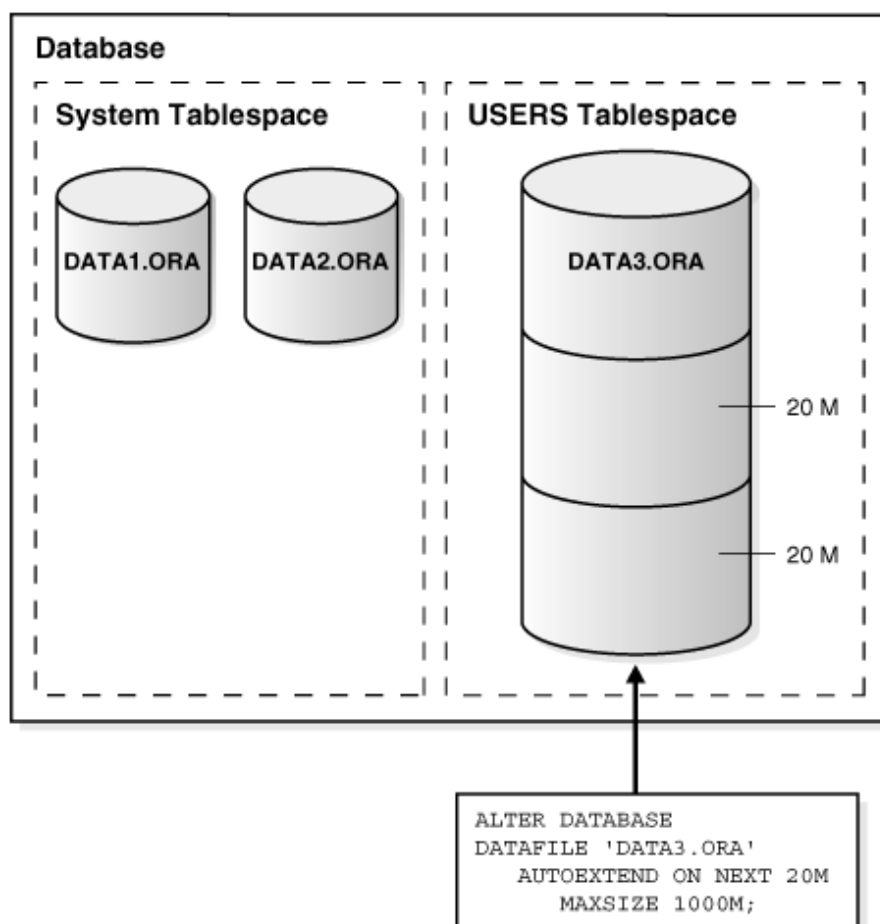
- Añadiendo un archivo de datos a un tablespace



- Añadiendo un nuevo tablespace



- Aumentando el tamaño de un archivo de datos



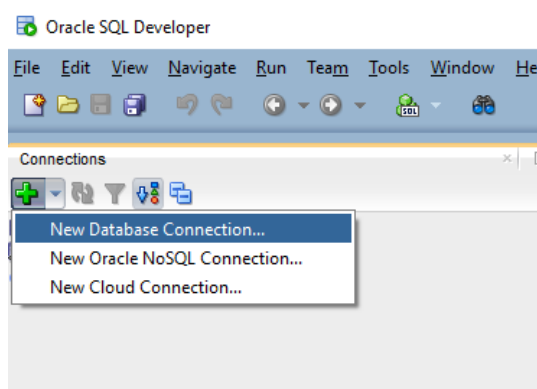
Por defecto, Oracle proporciona los siguientes tablespaces:

- SYSTEM: almacén de objetos del sistema como el diccionario de datos
- SYSAUX: dedicado a componentes adicionales de la base de datos, como por ejemplo el repositorio del Enterprise Manager
- USERS: almacén por defecto para los diferentes usuarios de la base de datos almacenen sus objetos
- TEMP: utilizado para almacenar resultados parciales en ordenaciones, operaciones complejas, etc...

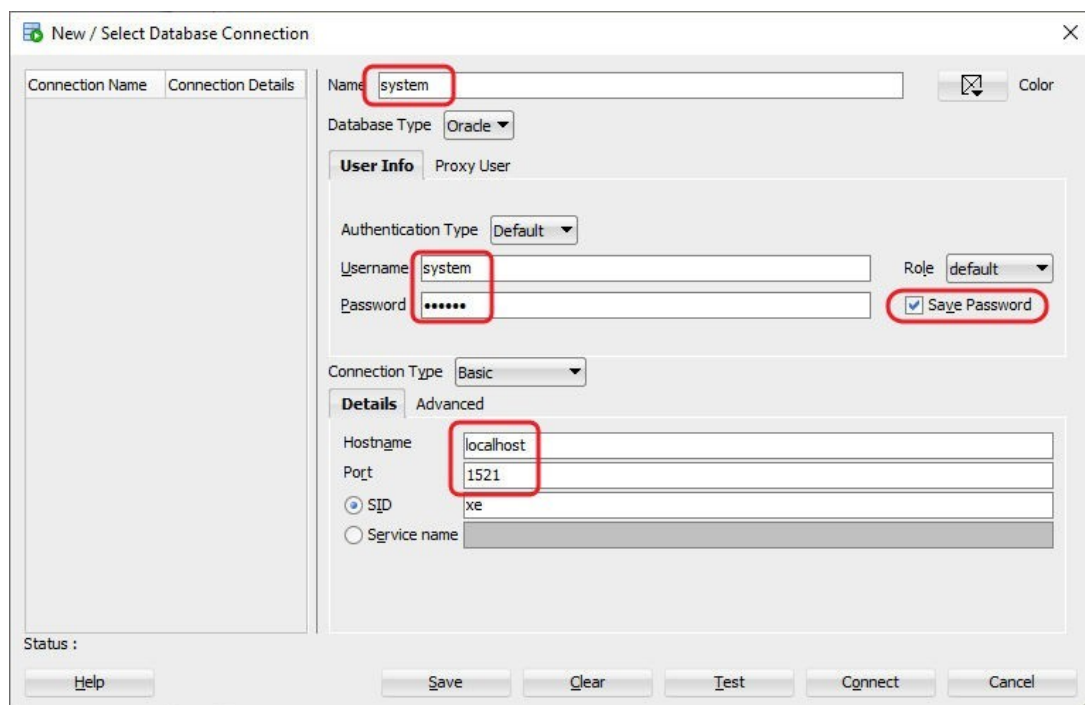
1.1. Conexión a la base de datos como DBA

En un documento aparte se ha tratado la instalación y puesta en marcha del servidor de base de datos Oracle Database 18c Express Edition, así como la herramienta gráfica SQL Developer, que nos permite interactuar de manera cómoda contra la base de datos.

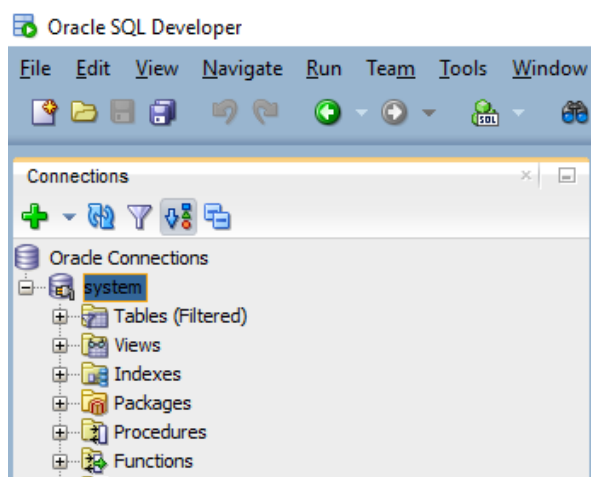
Recordemos que para crear una conexión a nuestro servidor Oracle Database en SQL Developer, lo haremos desde el menú "New Database Connection" o directamente haciendo clic sobre el botón "+"



A continuación, completaremos toda la información necesaria para conectarnos al servidor de base de datos, en este caso, nuestra primera conexión, utilizaremos el usuario `system` (lo que generalmente llamaremos usuario DBA)



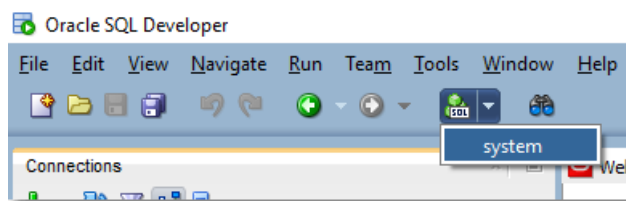
Después de guardar esta nueva conexión, podremos trabajar con ella a través del árbol de objetos



Una vez que tengamos esta conexión con privilegios de DBA (usuario `system`), nos vamos a centrar en crear aquellos elementos que nos permitan trabajar contra esa instancia de base de datos que ya está funcionando y podamos crear tablas, insertar datos en ellas, etc...

1.2. Creación de nuestro tablespace

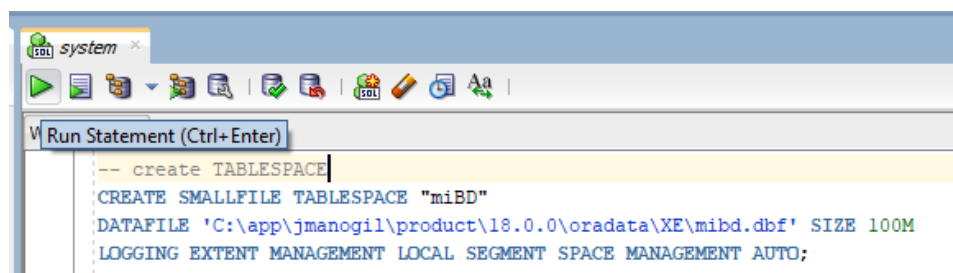
En primer lugar vamos a crear un tablespace que será nuestro espacio de trabajo. Para ello abriremos lo que SQL Developer llama "SQL Worksheet" (una hoja de trabajo SQL)



Copiaremos en la nueva pestaña el siguiente comando, modificando la ubicación del archivo para que coincida con los directorios de instalación de Oracle 18c en nuestra máquina

```
CREATE SMALLFILE TABLESPACE "miBD"  
DATAFILE 'C:\app\jmanogil\product\18.0.0\oradata\XE\mibd.dbf' SIZE 100M  
LOGGING EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO;
```

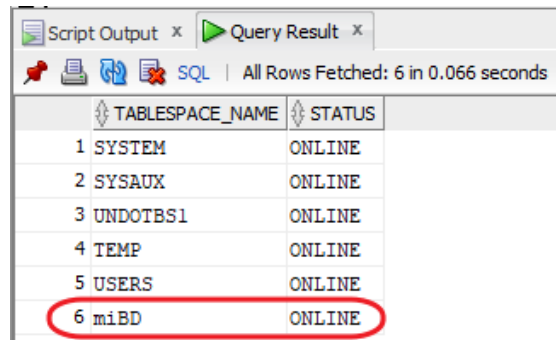
Lo ejecutamos



Con este comando hemos creado un tablespace, compuesto por un único archivo con un tamaño de 100M. Podemos comprobar que se ha creado correctamente ejecutando el comando siguiente en la "SQL Worksheet"

```
SELECT tablespace_name, status FROM dba_tablespaces;
```


Obteniendo en la pestaña "Query result"



The screenshot shows a window titled 'Query Result' with a table of database tablespaces. The table has two columns: 'TABLESPACE_NAME' and 'STATUS'. There are six rows of data. The last row, 'miBD', is circled in red.

TABLESPACE_NAME	STATUS
1 SYSTEM	ONLINE
2 SYSAUX	ONLINE
3 UNDOTBS1	ONLINE
4 TEMP	ONLINE
5 USERS	ONLINE
6 miBD	ONLINE

Si utilizamos el explorador de archivos, veremos que un nuevo archivo se ha creado con el nombre mibd.dbf



The screenshot shows a file explorer window with a list of files. The file 'MIBD.DBF' is circled in red.

File Name	Size
XEEDB1	
CONTROL01.CTL	18,288 KB
CONTROL02.CTL	18,288 KB
MIBD.DBF	102,408 KB
REDO01.LOG	204,801 KB

Podemos aumentar el tamaño del tablespace con el comando

```
ALTER DATABASE DATAFILE 'C:\app\jmanogil\product\18.0.0\oradata\XE\mibd.dbf'  
RESIZE 200M;
```

También podríamos borrarlo

```
DROP TABLESPACE "miBD";
```

El comando anterior no eliminará el archivo físico del disco, para ello deberemos hacer un borrado completo con el comando

```
DROP TABLESPACE "miBD" INCLUDING CONTENTS AND DATAFILES;
```

Estos últimos comandos son irreversibles, por lo que hay que tener muy claro lo que se está haciendo

1.3. Creación de nuestro usuario

Vamos ahora a crear un usuario de base de datos cuyo espacio de trabajo por defecto sea el tablespace que hemos creado. Para ello, primero ejecutaremos el siguiente comando SQL (necesario desde la versión 12c para crear este tipo de usuarios)

```
ALTER SESSION SET "_ORACLE_SCRIPT"=true;
```

con lo que ya podremos crear el usuario

```
CREATE USER usuario IDENTIFIED BY contraseña  
DEFAULT TABLESPACE "miBD" QUOTA UNLIMITED ON "miBD"  
TEMPORARY TABLESPACE "TEMP";
```

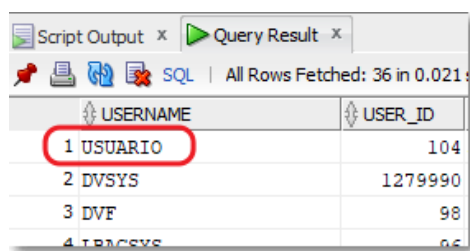
Podemos usar el nombre y contraseña que queramos. La modificación de la contraseña se puede hacer también por comandos

```
ALTER USER usuario IDENTIFIED BY usuario_nueva;
```

Si queremos ver la lista de usuarios, incluyendo el que acabamos de crear, lo podemos hacer con el comando

```
SELECT * FROM all_users ORDER BY created DESC;
```

Que mostraría una pestaña "Query result"



USERNAME	USER_ID
1 USUARIO	104
2 DVSYS	1279990
3 DVF	98
4 TRACESYS	96

El borrado de un usuario también es muy sencillo

```
DROP USER usuario;
```

1.4. Roles de usuario

Una vez creado nuestro primer usuario, deberemos asignarle una serie de permisos de uso y creación sobre ciertos objetos de la base de datos, es lo que llamaremos privilegios. Estos privilegios se suelen agrupar en lo que se conoce como roles, de tal modo que podremos organizar mejor esa asignación de permisos a los usuarios.

Cuando un usuario se conecta a la base de datos, en primer lugar se comprobará que el usuario existe y si su contraseña es correcta. Después, se recogerán aquellos privilegios que se le han concedido y en base a ellos se le permitirán o no hacer determinadas acciones sobre la base de datos.

Oracle nos ofrece unos roles predefinidos

Rol	Privilegios
CONNECT	Todos los permisos necesarios para iniciar una sesión en Oracle
RESOURCE	Todos los permisos necesarios para la creación de objetos (menos vistas y sinónimos)
DBA	Todos los permisos para un administrador de base de datos (DBA)

Asignaremos a nuestro usuario los roles CONNECT y RESOURCE utilizando el comando GRANT. Además, añadiremos los privilegios para crear vistas (CREATE VIEW) y sinónimos (CREATE SYNONYM), ya que curiosamente, no están incluidos en RESOURCE.

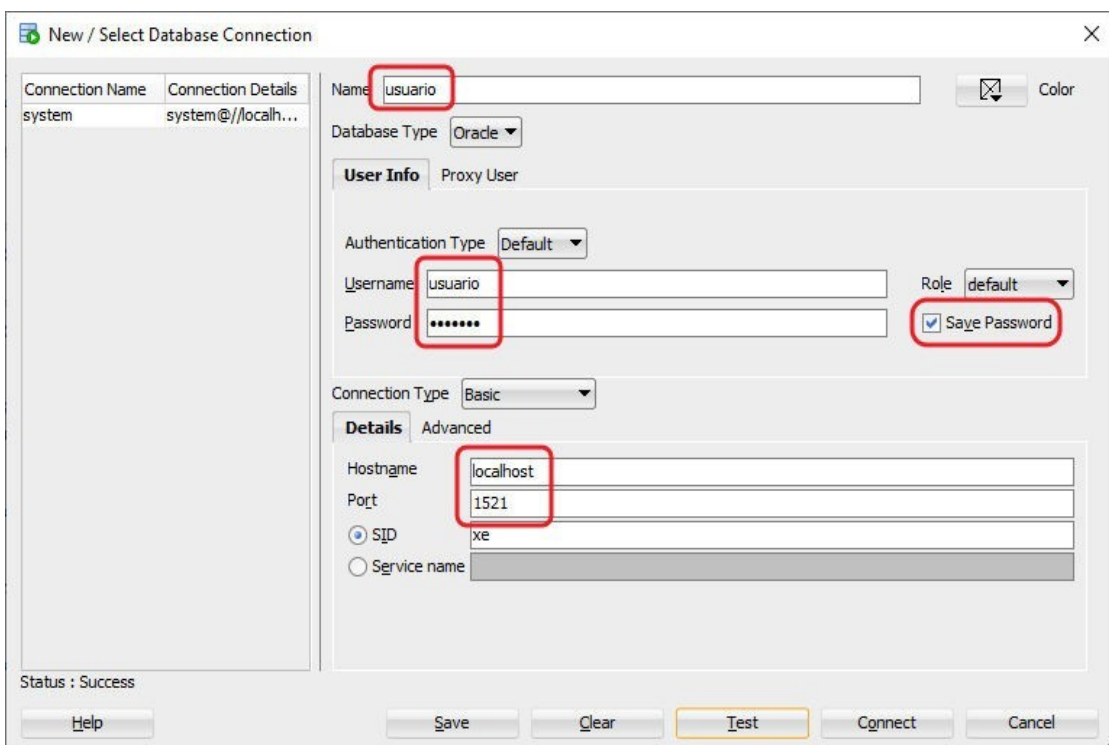
```
GRANT CONNECT, RESOURCE, CREATE VIEW, CREATE SYNONYM TO usuario;
```

Para eliminar el rol asignado podemos utilizar el comando REVOKE

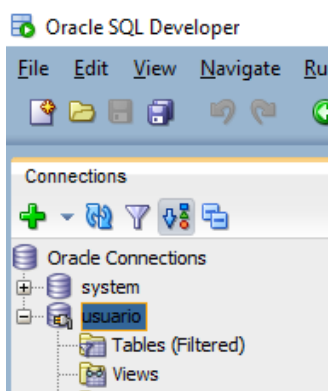
```
REVOKE RESOURCE, CONNECT, CREATE VIEW, CREATE SYNONYM FROM usuario;
```

1.5. Primeros pasos con nuestro usuario

En este punto ya tenemos todo lo necesario para poder trabajar contra Oracle Database utilizando nuestro propio usuario y su espacio de trabajo propio. Nos toca ahora comprobar que podemos conectarnos y crear objetos en la base de datos. Primero crearemos una nueva conexión

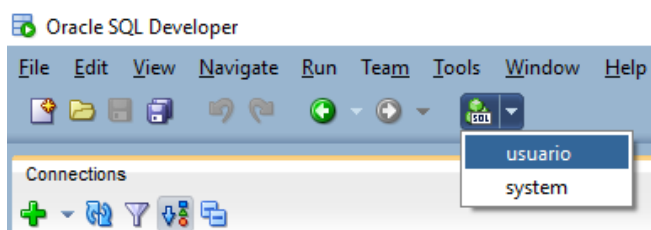


Haciendo clic sobre el botón "Test" podremos comprobar que ese usuario puede conectarse a la base de datos. Guardamos la nueva conexión y veremos que ahora tenemos dos conexiones disponibles en nuestro SQL Developer



Aunque estamos usando el mismo nombre para las conexiones que el usuario utilizado en ellas, no es necesario que se haga así, pero si que es cierto que puede ser muy cómodo hacerlo de este modo. Hay que tener en cuenta que las operaciones que podamos realizar con la conexión, van a estar supeditadas a los permisos (privilegios) que el usuario utilizado tenga.

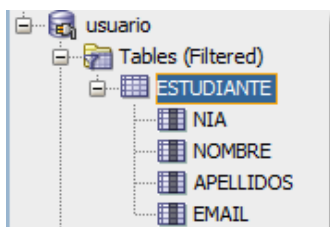
Abrimos una nueva "SQL Worksheet", pero que utilice la nueva conexión



Vamos a crear nuestra primera tabla, para ello copiaremos el siguiente comando SQL y lo ejecutaremos

```
CREATE TABLE estudiante (  
    NIA NUMBER NOT NULL,  
    nombre VARCHAR2(25),  
    apellidos VARCHAR2(50),  
    email VARCHAR2(100),  
    CONSTRAINT pk_estudiante PRIMARY KEY (NIA)  
);
```

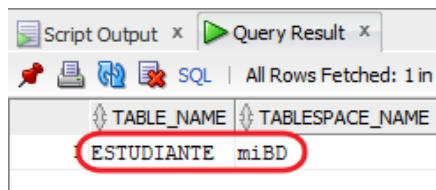
En el árbol de objetos podremos ver nuestra primera tabla creada



También podemos comprobar que se ha creado con el comando

```
SELECT * FROM USER_ALL_TABLES;
```

Donde veremos que la nueva tabla se ha creado en el tablespace que tiene por defecto nuestro usuario



TABLE_NAME	TABLESPACE_NAME
ESTUDIANTE	miBD

Para borrar una tabla usamos, como en el caso de los usuarios, el comando DROP

```
DROP TABLE estudiante;
```

2. SQL y su lenguaje de Definición de Datos DDL

SQL es un lenguaje para organizar, gestionar y recuperar datos almacenados en una base de datos informática. El nombre "SQL" es una abreviatura de Structured Query Language (Lenguaje Consultas Estructuradas). Es un lenguaje informático que se puede utilizar para interacciones con una base de datos de tipo relacional.

SQL se utiliza para controlar todas las funciones que un sistema de gestión de base de datos (SGBS, DBMS en inglés) proporciona a sus usuarios, incluyendo:

- Definición de datos: permite a un usuario definir la estructura y organización de los datos almacenados y las relaciones existentes entre ellos.
- Recuperación de datos: permite a un usuario o aplicación recuperar los datos almacenados de la base de datos y utilizarlos.
- Manipulación de datos: permite a un usuario o aplicación actualizar la base de datos añadiendo nuevos datos, suprimiendo datos antiguos y modificando datos previamente almacenados.
- Control de acceso: puede ser utilizado para restringir la capacidad de un usuario para recuperar, añadir y modificar datos, protegiendo así los datos almacenados frente a accesos no autorizados.
- Compartición de datos: se utiliza para coordinar la compartición de datos por parte de usuarios concurrentes, asegurando que no interfieren unos con otros.
- Integridad de datos: define restricciones de integridad en la base de datos, protegiéndola contra corrupciones debidas a actualizaciones inconsistentes o a fallos del sistema.

1.1. Lenguajes SQL

Tradicionalmente los comandos SQL se han agrupado en cuatro grupos principales:

- DDL (lenguaje de definición de datos): posibilita el crear, modificar y eliminar objetos. Los más importantes son CREATE, ALTER y DROP
- DML (lenguaje de manipulación de datos): permiten recuperar, insertar, actualizar y eliminar filas de una tabla. Los cuatro comandos DML básicos serían SELECT, INSERT, UPDATE y DELETE
- TCL (lenguaje de control de transacciones): permiten controlar el resultado de una transacción en una base de datos. Tendríamos los comandos COMMIT y ROLLBACK
- DCL (lenguaje de control de datos): utilizado para controlar el acceso de los usuarios a los objetos de la base de datos. Los más comunes son GRANT y REVOKE.

En este documento nos vamos a centrar en el primero de ellos, el DDL. Para poder comprobar que los comandos DDL se han ejecutado correctamente, en ocasiones utilizaremos algunos comandos sencillos del lenguaje DML, sin embargo, serán en el tema siguiente donde los trataremos en profundidad.

1.1. Tipos de datos

A la hora de crear las tablas, tendremos que indicar el tipo de dato de cada una de sus columnas, por lo que necesitaremos conocer los diferentes tipos de datos que SQL nos ofrece. Vamos a ver los que se consideran más importantes y que prácticamente todos los sistemas de gestión de bases de datos utilizan, con muy pequeñas variaciones.

1.1.1. Textos

Sintaxis	Descripción	Rango
CHAR(n)	Almacena textos de longitud fija indicado por el valor de n	Hasta 2000 bytes
NCHAR(n)	Almacena textos de longitud fija indicado por el valor de n. Puede utilizar diferentes juegos de caracteres, como UTF8	Hasta 2000 bytes
NVARCHAR2(n)	Almacena textos de longitud variable, hasta el número indicado de caracteres. Puede utilizar diferentes juegos de caracteres, como UTF8	Hasta 4000 bytes
VARCHAR2(n)	Almacena textos de longitud variable, hasta el número indicado de caracteres	Hasta 4000 bytes y 32KB en PL/SQL
LONG	Almacena textos de longitud variable. Se mantiene por compatibilidad con versiones anteriores (preferible usar tipo LOB)	Hasta 2GB

Hay que tener en cuenta que si definimos una columna de tipo VARCHAR2(1000), pero introducimos un texto que contenga solamente 10 caracteres, la columna ocupará 10 bytes, esto es, lo que ocupen los caracteres introducidos. No ocurre lo mismo con el tipo CHAR, ya que en ese caso, el resto de caracteres no utilizados se rellenarían con espacios.

Se recomienda el uso de NCHAR y NVARCHAR, cuando se quieran almacenar textos en diferentes lenguajes, en la misma columna o cuando se quiera permitir que el usuario introduzca el texto que desee sin ninguna restricción.

1.1.2. Números

Sintaxis	Descripción	Rango
NUMBER(p, e)	Donde p es la precisión y e la escala	Precisión de 1 a 38 y escala de -84 a 127
INTEGER	Equivale a NUMBER(38, 0)	
DECIMAL(p, e)	Equivale a NUMBER(p, e)	

El tipo de dato NUMBER es el que normalmente utilizaremos para representar valores numéricos que pueden ser positivos y negativos. Es cierto que Oracle ofrece más variedad de tipos de datos numéricos, pero casi todos acaban representándose internamente como del tipo NUMBER.

Para este tipo de dato podemos especificar precisión (número de dígitos de un número) y escala (número de dígitos a la derecha del punto decimal). Por ejemplo, el número 123.45 tiene una precisión de 5 y una escala de 2, por lo que para almacenar este número necesitamos un NUMBER(5, 2). Si no indicamos precisión y/o escala, tomarían los valores máximos posibles.

En el caso de insertar un número que excede la precisión definida, Oracle lanzará un error. Sin embargo, si el número excede la escala, se redondea. Por ejemplo, si la columna es del tipo NUMBER(5,1) e insertamos el valor 123.45, el valor finalmente insertado es 123.5.

También podemos utilizar escalas con valor negativo, con el efecto de un redondeo hacia la parte entera. Por ejemplo, con una columna del tipo NUMBER(4, -1) al insertar el valor 123.5, estaremos insertando 120.

El tipo de dato BOOLEAN no existe en Oracle, en su lugar usaremos NUMBER(1).

1.1.3. Fechas y horas

Sintaxis	Descripción	Rango
DATE	Almacena fecha y hora: siglo, año, mes, día, hora, minutos y segundos	Desde 1 de Enero -4712 hasta 31 de Diciembre 9999
TIMESTAMP(p)	Ampliación del tipo DATE, pero almacena también fracciones de segundos. El valor p nos indica el número de dígitos en la parte fraccional de los segundos, puede tomar el valor entre 0 y 9, por defecto 6.	
INTERVAL YEAR (p) TO MONTH	Periodo de tiempo en términos de años y meses, donde p indica el número de dígitos del año	
INTERVAL DAY (pd) TO SECOND (ps)	Periodo de tiempo en términos de días, horas, minutos y segundos, donde pd indica el número de dígitos del día y ps el de los segundos	

El formato por defecto para las fechas en Oracle SQL es 'DD-MES-YYYY', por ejemplo: '01-DEC-2015'. Sin embargo recomendamos utilizar el formato de fechas ISO: 'YYYY-MM-DD'

Un valor de tipo DATE sin el componente de hora, se le asignará la hora por defecto 12:00:00 AM. Si por contra no se incluye el componente fecha, esto es, solamente la hora, se le asigna el primer día del mes actual.

El tipo INTERVAL resulta muy útil cuando se quiere almacenar un periodo concreto de tiempo entre dos fechas y horas. Podemos trabajar con intervalos año-mes, expresando periodos de tiempo en términos de años y un número de meses, y con intervalos día-hora, que nos permiten una mayor precisión almacenando periodos en términos de días, horas, minutos y segundos. Hay que tener en cuenta que los intervalos pueden ser positivos o negativos.

1.1.4. Objetos de gran tamaño

Oracle ofrece tres tipos de datos para almacenar objetos de gran tamaño, son los tipos de datos LOB (Large Object)

Sintaxis	Descripción	Rango
BLOB	Almacena datos binarios sin estructura	Hasta 4GB
CLOB	Almacena texto	Hasta 4GB
NBLOB	Almacena texto con diferentes juegos de caracteres	Hasta 4GB
BFILE	Almacena un localizador que apunta a un archivo binario fuera de la base de datos	Hasta 4GB

Estos tipos de datos nos permiten almacenar grandes estructuras de información como texto, gráficos, audio y video, de este modo, archivos multimedia pueden residir en la misma base de datos.

1.2. Creación de Tablas

Después de conocer los principales tipos de datos que Oracle SQL nos ofrece, es hora de centrarnos en la creación de tablas, formadas por columnas que definiremos en base a esos tipos de datos.

Para crear una tabla utilizaremos el comando CREATE TABLE, con la siguiente sintaxis

```
CREATE TABLE nombre_tabla(  
    nombre_columna_1 tipo_dato  
        [NOT NULL]  
        [UNIQUE]  
        [PRIMARY KEY]  
        [DEFAULT valor]  
        [REFERENCES nombre_tabla] [columna[,columna]] [ON DELETE CASCADE]]  
    [CHECK condicion],  
  
    .....  
    {CONSTRAINT ck_nombre CHECK (condicion)}  
    {CONSTRAINT uk_nombre UNIQUE(columnas)}  
    {CONSTRAINT pk_nombre PRIMARY KEY(columnas_clave)}  
    {CONSTRAINT fk_nombre FOREIGN KEY(columnas) REFERENCES tabla(columnas_clave)  
    {ON DELETE CASCADE}}  
)
```

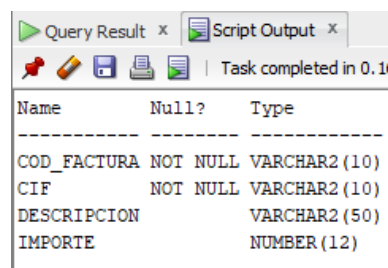
Veamos un ejemplo

```
CREATE TABLE cliente(  
    CIF VARCHAR2(10) NOT NULL,  
    nombre VARCHAR2(20),  
    direccion VARCHAR2(50),  
    ciudad VARCHAR2(20),  
    CP VARCHAR2(5) DEFAULT '00000',  
    detalles VARCHAR2(1000),  
    CONSTRAINT pk_cliente PRIMARY KEY(CIF)  
);  
  
CREATE TABLE factura(  
    cod_factura VARCHAR2(10) NOT NULL,  
    CIF VARCHAR2(10) NOT NULL,  
    descripcion VARCHAR2(50),  
    importe NUMBER(12),  
    CONSTRAINT pk_factura PRIMARY KEY(cod_factura),  
    CONSTRAINT fk_factura_cliente  
    FOREIGN KEY(CIF) REFERENCES cliente(CIF) ON DELETE CASCADE  
);
```

Con el comando DESCRIBE obtenemos la estructura de una tabla de manera rápida, por ejemplo

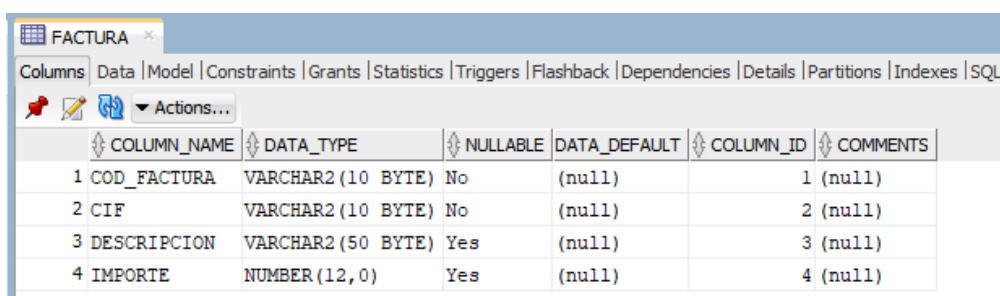
```
DESCRIBE factura;
```

Nos muestra



Name	Null?	Type
COD_FACTURA	NOT NULL	VARCHAR2(10)
CIF	NOT NULL	VARCHAR2(10)
DESCRIPCION		VARCHAR2(50)
IMPORTE		NUMBER(12)

También podemos en SQL Developer ir al árbol de objetos, seleccionar la tabla y nos aparecerá una nueva pestaña con multitud de detalles



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	COD_FACTURA	VARCHAR2(10 BYTE)	No	(null)	1 (null)	
2	CIF	VARCHAR2(10 BYTE)	No	(null)	2 (null)	
3	DESCRIPCION	VARCHAR2(50 BYTE)	Yes	(null)	3 (null)	
4	IMPORTE	NUMBER(12,0)	Yes	(null)	4 (null)	

Las restricciones o CONSTRAINT se pueden definir tanto a nivel de columna o tabla, con la excepción de NOT NULL que solamente se acepta en la definición de la columna.

Los tipos de CONSTRAINT de columna incluyen

- NOT NULL: la columna no acepta valores nulos (NULL)
- UNIQUE: el contenido de esa columna o columnas no puede repetir valores
- PRIMARY KEY: los valores en esa columna o columnas identifican de manera única una fila en la tabla, haciendo que los campos sean NOT NULL y UNIQUE
- FOREIGN KEY: los valores de la columna o columnas deben corresponderse con valores en las claves primarias en la tabla referenciada
- CHECK: establece restricciones de validación que deben cumplirse en

cada fila Por ejemplo

```
CREATE TABLE proveedor(  
    codigo NUMERIC(4) PRIMARY KEY,  
    nombre VARCHAR2(50) NOT NULL,  
    CONSTRAINT ck_codigo CHECK (codigo BETWEEN 5000 AND 9999),  
    CONSTRAINT ck_nombre CHECK (nombre = UPPER(nombre)),  
    CONSTRAINT uk_nombre UNIQUE(nombre)  
);
```

En la tabla anterior las restricciones nos impiden insertar códigos de proveedor fuera del rango entre 5000 y 9999. Además el nombre no puede ser NULL, no se puede repetir y tiene que estar en mayúsculas. Se producirá un error en todas y cada una de las siguientes inserciones

```
INSERT INTO proveedor VALUES (1, 'ACME');  
INSERT INTO proveedor VALUES (5500, NULL);  
INSERT INTO proveedor VALUES (5500, 'acme');
```

La siguiente inserción se hará correctamente

```
INSERT INTO proveedor VALUES (5500, 'ACME');
```

Podemos comprobarlo con el comando

```
SELECT * FROM proveedor;
```

Pero las siguientes también producen un error

```
INSERT INTO proveedor VALUES (5500, 'TIA');  
INSERT INTO proveedor VALUES (5501, 'ACME');
```

La restricción UNIQUE, a diferencia de la PRIMARY KEY, permite que las columnas que lo forman puedan contener valores NULL, siempre y cuando la combinación de valores sea única. Oracle no permite crear una PRIMARY KEY y una restricción UNIQUE con las mismas columnas.

Las FOREIGN KEYS o claves ajenas obligan al cumplimiento de lo de que en bases de datos relacionales llamamos integridad referencial. Esta integridad, por ejemplo en el caso de las tablas `cliente` y `factura`, obliga a que toda factura que se cree, el CIF que almacene tenga un valor que exista en la tabla referenciada cliente, en caso contrario se produciría un error. Además, en el ejemplo añadimos la sentencia ON DELETE CASCADE, consiguiendo que cuando un cliente se elimine de su tabla, todas las facturas que tuviera ese cliente también se eliminarán automáticamente.

Si no indicamos ese ON DELETE CASCADE en la FOREIGN KEY `fk_factura_cliente` e intentamos por ejemplo eliminar un cliente con facturas asociadas, se producirá un error. Del mismo modo que también se produciría un error cuando intentemos insertar una factura con un CIF que no pertenezca a ningún cliente, en ambos casos se está intentando violar la integridad referencial establecida en la base de datos.

Vamos a insertar un registro en la tabla `cliente` y otro en `factura` con los comandos

```
INSERT INTO cliente (CIF, nombre) VALUES ('12345678A', 'Luis Rodriguez');  
INSERT INTO factura (cod_factura, CIF, importe) VALUES ('1A', '12345678A', 1000);
```

Debido a la FOREIGN KEY que asocia `factura` y `cliente`, el siguiente comando nos produciría un error

```
INSERT INTO factura (cod_factura, CIF, importe) VALUES ('2A', '87654321A', 1000);
```

Es muy útil prestar atención al mensaje de error que Oracle nos devuelve, os dará muchas pistas.

Tampoco podríamos modificar el CIF del cliente asociado con la factura. El siguiente comando también produciría un error

```
UPDATE cliente SET CIF = '87654321B' WHERE nombre = 'Luis Rodriguez';
```

¿ Qué sucede si elimino el cliente con CIF '12345678A' ? Comprobémoslo con el siguiente comando

```
DELETE cliente WHERE CIF = '12345678A';
```

La factura asociada a ese cliente se ha eliminado automáticamente.

Hay que tener en cuenta que Oracle Database nos ofrece solamente dos posibilidades que podemos usar con las claves ajenas:

- **ON DELETE SET NULL:** actualiza a NULL las claves relacionadas con la borrada
- **ON DELETE CASCADE:** borra todas las filas cuando se eliminan las filas de la tabla referenciada

En ocasiones puede interesar crear una tabla partiendo de otra tabla ya existente, copiando sus columnas y todas o parte de sus filas, para ello utilizaremos el comando **CREATE TABLE AS**. Veamos un ejemplo utilizando la anterior tabla `proveedor` y creando una nueva a partir de ella

```
CREATE TABLE proveedor_5000 AS
(SELECT codigo, nombre FROM proveedor WHERE CODIGO >= 5000);
```

En posteriores documentos profundizaremos más en el comando **SELECT** que, como hemos visto, nos permite filtrar las filas y columnas de una tabla.

Para acabar con la creación de tablas, comentaremos el caso de las columnas del tipo **IDENTITY**. Este tipo de columnas se introdujeron a partir de la versión 12c de Oracle y nos permiten crear fácilmente columnas que almacenen números que automáticamente se incrementen con cada inserción. Esto resulta muy útil cuando necesitemos crear un campo clave y no necesitamos darle un valor en concreto, es suficiente con que el SGBD se encargue de ello.

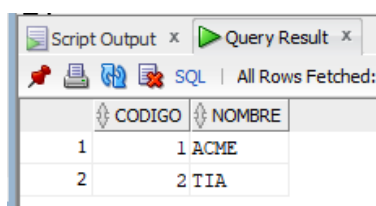
Hasta la versión 12c, podíamos conseguir el mismo resultado empleando lo que Oracle denomina **SEQUENCES**, sin embargo es mucho más sencillo trabajar con este tipo de columna.

Veamos como utilizar esta columna **IDENTITY** creando una variante de la tabla `proveedor` e insertando algunas filas

```
CREATE TABLE proveedor_identity(
    codigo NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    nombre VARCHAR2(50) NOT NULL
);
INSERT INTO proveedor_identity (nombre) VALUES ('ACME');
INSERT INTO proveedor_identity (nombre) VALUES ('TIA');
```

Si consultamos el contenido de la tabla, veremos que el campo `codigo` tiene un valor que nosotros no hemos asignado, se ha hecho automáticamente

```
SELECT * FROM proveedor_identity;
```



CODIGO	NOMBRE
1	ACME
2	TIA

1.3. Modificación de Tablas

En ocasiones puede ser necesario modificar la estructura de una tabla ya creada. Si la tabla ya se ha cargado con datos, la opción de borrarla y volver a crearla no es conveniente. Para ello tenemos el comando ALTER TABLE, con la sintaxis

```
ALTER TABLE nombre_tabla {  
    ADD [ COLUMN ] definicion_columna |  
    MODIFY [ COLUMN ] definicion_columna  
    ADD CONSTRAINT restriccion |  
    DROP [ COLUMN ] nombre_columna [ CASCADE | RESTRICT ]  
    DROP { PRIMARY KEY | FOREIGN KEY nombre | UNIQUE nombre  
        | CHECK nombre | CONSTRAINT nombre }  
}
```

Veamos ejemplos de los cambios más habituales

1.3.1. Renombrar la tabla

```
ALTER TABLE cliente RENAME TO cliente_empresa;
```

1.3.2. Añadir columnas

```
ALTER TABLE cliente_empresa ADD (fecha_alta DATE);
```

1.3.3. Modificar columnas

```
ALTER TABLE cliente_empresa MODIFY (  
    fecha_alta TIMESTAMP,  
    nombre VARCHAR2(100) NULL,  
    direccion VARCHAR2(100) NOT NULL);
```

Lógicamente, los cambios que se pueden hacer en las columnas serán posibles si los datos que ya están en la tabla lo permiten. Por ejemplo, si intentamos modificar una columna tipo VARCHAR(50) a VARCHAR(10) y hay valores con más de 10 caracteres, nos dará error.

1.3.4. Borrar columnas

```
ALTER TABLE cliente_empresa DROP (fecha_alta);
```

1.3.5. Renombrar columnas

```
ALTER TABLE cliente_empresa RENAME COLUMN nombre TO nombre_completo;
```

1.3.6. Borrar una CONSTRAINT (también FOREIGN KEY)

```
ALTER TABLE factura DROP CONSTRAINT fk_factura_cliente;
```

1.3.7. Añadir una FOREIGN KEY

```
ALTER TABLE factura ADD CONSTRAINT fk_factura_cliente  
FOREIGN KEY (CIF) REFERENCES cliente_empresa (CIF) ON DELETE CASCADE;
```

1.3.8. Añadir una CHECK CONSTRAINT

```
ALTER TABLE factura ADD CONSTRAINT ck_importe CHECK(importe > 1000);
```

1.4. Vaciar Tablas

Disponemos de un comando para vaciar el contenido de una tabla, esto es, todos sus registros sin afectar a la estructura de la misma. Su uso es muy sencillo

```
TRUNCATE TABLE factura;
```

1.5. Índices

Los índices son objetos que aceleran las operaciones de consulta y ordenación sobre determinadas columnas en unas determinadas tablas. Son estructuras de datos que se almacenan aparte de la tabla a la que referencian, por lo que se pueden crear y borrar en cualquier momento.

Se basan en crear y mantener una lista ordenada, que es la que Oracle utilizará para hacer las búsquedas y ordenaciones. Una búsqueda en una lista ordenada siempre es más rápida que si los datos no siguen ningún orden.

Cada vez que se añade un nuevo registro, los índices involucrados se actualizan, de ahí que cuantos más índices tengamos, más le cuesta a Oracle (o el SGBD con el que trabajemos) añadir registros, pero más rápidas se realizan las consultas.

Las restricciones PRIMARY KEY, UNIQUE y FOREIGN KEY, crean automáticamente índices asociados. Se trata de índices obligatorios que el SGBD necesita, pero podemos crear nuevos índices con la sintaxis

```
CREATE [UNIQUE] INDEX nombre_indice  
ON nombre_tabla (columna_1, columna_2, ... columna_n)
```

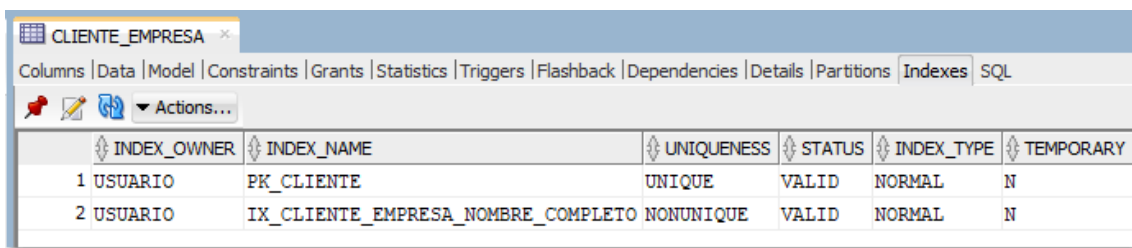
Un índice puede hacer referencia a valores que se puedan repetir, no hay que confundirlo con una clave primaria. Además puede estar formado por más de una columna.

Por ejemplo

```
CREATE INDEX ix_cliente_empresa_nombre_completo  
ON cliente_empresa (nombre_completo);
```

Con este índice conseguiremos que las búsquedas en la tabla, a través del campo nombre_completo, sean más rápidas.

En la ficha de la tabla en SQL Developer, pestaña "Indexes", podemos comprobar que el índice se ha creado.



	INDEX_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	TEMPORARY
1	USUARIO	PK_CLIENTE	UNIQUE	VALID	NORMAL	N
2	USUARIO	IX_CLIENTE_EMPRESA_NOMBRE_COMPLETO	NONUNIQUE	VALID	NORMAL	N

Una vez creado un índice, Oracle no permite modificarlo añadiéndole más columnas, deberemos eliminarlo y volverlo a crear de nuevo. El borrado se realiza con el comando DROP, por ejemplo

```
DROP INDEX ix_cliente_empresa_nombre_completo;
```

1.6. Vistas

Una vista o VIEW, no es más que una consulta SELECT almacenada en la base de datos con un nombre, de tal manera que podremos obtener el resultado de esa consulta, tantas veces como se desee, haciendo referencia a ese nuevo objeto VIEW creado.

Por ejemplo, en el apartado dedicado a la creación de tablas, hemos creado una tabla a partir de una consulta

```
CREATE TABLE proveedor_5000 AS  
(SELECT codigo, nombre FROM proveedor WHERE CODIGO >= 5000);
```

Crear una vista a partir de esa SELECT hubiera sido una opción también

```
CREATE VIEW v_proveedor_5000 AS  
(SELECT codigo, nombre FROM proveedor WHERE CODIGO >= 5000);
```

Y la consulta de esa vista sería muy sencilla

```
SELECT * FROM v_proveedor_5000;
```

Mientras que la tabla `proveedor_5000` contiene los datos que hubieran en la tabla `proveedor` justo cuando se creó, la vista `v_proveedor_5000` nos devuelve los datos que hubieran en `proveedor` en el momento justo de hacer la consulta, contra la vista.

Una vista no contiene datos, sino la instrucción `SELECT` necesaria para generar los resultados, por lo que las vistas ocupan muy poco espacio en disco. Se suelen emplear para

- Realizar consultas complejas más fácilmente, ya que permiten dividir la consulta en varias partes
- Proporcionar resultados con datos complejos
- Utilizar visiones especiales de los datos
- Servir como tablas que resumen de todos los datos

Hay que tener en cuenta que una vista puede ser tan compleja como la `SELECT` que la forme, por lo que podría estar formada por datos de diferentes tablas, agrupaciones, etc...

La sintaxis de una `VIEW` es

```
CREATE OR REPLACE VIEW nombre_vista AS consulta;
```

El borrado es como en el resto de objetos, con la sentencia `DROP`

```
DROP VIEW v_proveedor_5000;
```

Profundizaremos más el tema de las vistas cuando estudiemos las consultas avanzadas en el lenguaje `DML`.

1.7. Sinónimos

Un sinónimo o `SYNONYM` es un nombre alternativo para un objeto de la base de datos. Se suele utilizar para tablas o vistas, el que se pueda acceder a ellas a través de otro nombre puede ser útil en ocasiones.

La sintaxis sería

```
CREATE SYNONYM nombre_sinónimo FOR objeto;
```

Por ejemplo, hay gente que está acostumbrado a utilizar plurales para nombrar las tablas, podríamos crear un

sinónimo para la tabla proveedor

```
CREATE SYNONYM proveedores FOR proveedor;
```

Con ello la consulta siguiente, en realidad está buscando los datos en proveedor

```
SELECT * FROM proveedores;
```

1.8. Comentarios

Para marcar un texto como comentario en SQL usaremos el doble guión "--", para un comentario de una única línea y "/* */" para una o varias líneas.

```
-- Esto es un comentario
/* Esto también */
/*
    Esto también
*/
```

1.9. Comprobar objetos creados

Oracle nos ofrece unas vistas predefinidas que nos permiten consultar los objetos que vamos creando en la base de datos. De hecho, ya usamos en un apartado anterior la siguiente consulta que nos muestra las tablas que hemos creado:

```
SELECT * FROM USER_ALL_TABLES;
```

Restricciones

```
SELECT * FROM USER_CONSTRAINTS;
```

Índices

```
SELECT * FROM USER_INDEXES;
```

Todos los objetos


```
SELECT * FROM USER_OBJECTS;
```

Sinónimos

```
SELECT * FROM USER_SYNONYMS;
```

Columnas de tablas y vistas

```
SELECT * FROM USER_TAB_COLUMNS;
```

Vistas

```
SELECT * FROM USER_VIEWS;
```