UT_4 SQL

4.1 Elementos del lenguaje

SQL (Structured Query Language) es el lenguaje fundamental de los SGBD relacionales. Es un lenguaje declarativo, es decir, lo más importante es definir **qué** se desea hacer y **no cómo** hacerlo.

Es un lenguaje normalizado que nos permite trabajar con lenguajes de alto nivel como ASP, PHP o Java en combinación con cualquier tipo de base de datos (Access, SQL Server, MySQL, Oracle, etc.). No es idéntico para cada base de datos, puede presentar funciones específicas en cada una de ellas o cambiar la sintaxis, o los tipos de datos. Se recomienda revisar la documentación del SGBD.

Es un lenguaje muy potente y a la vez fácil de aprender, ya que utiliza un lenguaje bastante natural. Las instrucciones son muy parecidas a las órdenes humanas.

SQL aunque sea un lenguaje fundamentalmente de consulta también nos permite definir la base de datos, manipularla y especificar conexiones seguras.

Normas de escritura

Seguiremos las siguientes normas de escritura:

- Cada **instrucción termina** en (**;**).
- **No** se **distingue** entre **mayúsculas** y **minúsculas**.
- Cualquier comando se puede dividir en dos líneas para facilitar su lectura.
- Los comentarios se indican con : /* para el principio y */ para el fin. En una linea comentario --Comentario

Elementos del lenguaje

El lenguaje está formado por comandos, cláusulas, operadores, funciones y literales. Todos estos elementos se combinan en las instrucciones y se utilizan para crear, actualizar y manipular bases de datos. La definición de estos elementos es la siguiente:

- **Comandos**, son las instrucciones que se pueden crear en SQL y se pueden distinguir de tres tipos:

  - **De definición de datos , DDL,** que permiten crear bases de datos, tablas, campos y otros objetos de la BD.

Comandos DDL. Lenguaje de Definición de Datos.	
Comando:	**Descripción:**
CREATE	Se utiliza para crear nuevas tablas, campos e índices.
DROP	Se utiliza para eliminar tablas e índices.
ALTER	Se utiliza para modificar tablas.

  - **De manipulación de datos, DML,** permiten crear consultas para ordenar, filtrar y extraer datos. También permiten actualizar la información, insertar, borrar o modificar datos.

Comandos DML. Lenguaje de Manipulación de Datos.	
Comando:	**Descripción:**
SELECT	Se utiliza para consultar filas que satisfagan un criterio determinado.
INSERT	Se utiliza para cargar datos en una única operación.
UPDATE	Se utiliza para modificar valores de campos y filas específicos.
DELETE	Se utiliza para eliminar filas de una tabla.

  - **De control y seguridad, DCL,** que permiten administrar los privilegios y restricciones de los usuarios.

Comandos DCL. Lenguaje de Control de Datos.	
Comando:	**Descripción:**
GRANT	Permite dar permisos a uno o varios usuarios o roles para realizar tareas determinadas.
REVOKE	Permite eliminar permisos que previamente se han concedido con GRANT.

- **Cláusulas,** son condiciones o criterios, palabras especiales que permiten modificar el funcionamiento de un comando.

Cláusulas	
Cláusulas:	**Descripción:**
FROM	Se utiliza para especificar la tabla de la que se van a seleccionar las filas.
WHERE	Se utiliza para especificar las condiciones que deben reunir las filas que se van a seleccionar.
GROUP BY	Se utiliza para separar las filas seleccionadas en grupos específicos.
HAVING	Se utiliza para expresar la condición que debe satisfacer cada grupo.
ORDER BY	Se utiliza para ordenar las filas seleccionadas de acuerdo a un orden específico.

- **Operadores,** permiten crear expresiones complejas. Pueden ser aritméticos o lógicos.

Operadores lógicos.	
Operadores:	**Descripción:**
AND	Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.
NOT	Devuelve el valor contrario de la expresión.

Operadores de comparación.	
Operadores:	**Descripción:**
<	Menor que.
>	Mayor que.
< >	Distinto de.
< =	Menor o igual.
> =	Mayor o igual.
=	Igual.
BETWEEN	Se utiliza para especificar un intervalo de valores.
LIKE	Se utiliza para comparar.
IN	Se utiliza para especificar filas de una base de datos.

- **Funciones,** permiten conseguir valores complejos, por ejemplo, media, valor absoluto, etc.

Funciones de agregado.	
Función:	**Descripción:**
AVG	Calcula el promedio de los valores de un campo determinado.
COUNT	Devuelve el número de filas de la selección.
SUM	Devuelve la suma de todos los valores de un campo determinado.
MAX	Devuelve el valor más alto de un campo determinado.
MIN	Devuelve el valor mínimo de un campo determinado.

- **Literales,** o constantes y son valores concretos, una fecha (entre comillas simples), un número, etc.

Literales	
Literales:	**Descripción:**
23/03/97	Literal fecha.
María	Literal caracteres.
5	Literal número.

4.2 Lenguaje de definición de datos

Antes de poder guardar información y recuperarla debemos definir las estructuras donde almacenar la información. Las estructuras básicas con las que trabaja SQL son las tablas.

En el mercado existen herramientas que facilitan las tareas de crear, modificar y eliminar objetos de la base de datos. Proporcionan una interfaz visual que nos oculta el lenguaje SQL, simplemente pondremos nombres a los campos, elegir el tipo de datos y activar una serie de propiedades.

Pero es necesario conocer a fondo el lenguaje SQL, sus características e instrucciones ya que nos veremos en situaciones donde no podamos utilizar estas herramientas visuales.

En Oracle, cada usuario de una base de datos tiene un esquema, que tendrá el mismo nombre que dicho usuario y sirve para almacenar los objetos que sean de su propiedad. Estos objetos podrán ser tablas, vistas, índices u otros relacionados con la definición de la base de datos. Estos objetos podrán ser manipulados en principio por el usuario que los creó y por el administrador.

Todas las instrucciones DDL generan acciones que **no se pueden deshacer**, hay que usarlas con precaución y tener copias de seguridad cuando manipulamos la BD.

4.2.1 Creación de bases de datos

Una base de datos es un conjunto de objetos que nos servirán para gestionar sus datos. Basicamente, crear una base de datos consiste en crear las tablas que la componen. Cada conjunto de tablas de la BD se creará en un esquema y éste suele estar asociado a un usuario. Al crear la bases de datos, si se tiene privilegio de administrador, se podrán indicar los archivos y ubicaciones que se van a utilizar además de otras indicaciones técnicas y administrativas.

La instrucción que permite crear una base de datos es:

```
CREATE DATABASE nombreBD;
```

Si queremos crear una base de datos para una aplicación de una biblioteca podríamos indicar:

```
CREATE DATABASE Biblioteca;
```

En SQL existe el concepto de catálogo que sirve para almacenar esquemas. Así, el nombre completo de un objeto viene dado por:

Catálogo.esquema.objeto

Si no se indica el catálogo se toma el catálogo por defecto, si no se indica el esquema se entiende que el objeto está en el esquema actual.

4.2.2 Creación de tablas

La tabla será el objeto básico para SQL, gráficamente es un conjunto de filas columnas cuya intersección se conoce como celda. Antes de crear la tabla necesitaremos conocer los siguientes detalles:

- Nombre de la tabla
- Nombre de cada una de las columnas
- Tipo y tamaño de datos que se almacenará en cada columna.
- Restricciones sobre los datos.

Para dar nombre a la tabla debemos cumplir las siguientes reglas:

- En el mismo esquema no puede haber nombres de tablas duplicados.
- El nombre debe empezar por un carácter alfabético.

- Su longitud máxima es de 30 caracteres.
- Sólo se permiten en el nombre letras del alfabeto inglés, dígitos y el (_).
- No puede ser una palabra reservada de SQL.
- No se distingue entre mayúsculas y minúsculas.

La sintaxis básica del comando es:

```
CREATE TABLE[esquema.]nombreTabla (
      Columna1 tipoDato [DEFAULT valor][restricciones],
      …
      columnaN tipoDato [DEFAULT valor][restricciones]);
```

Donde:

- Columna1, .. columnaN, son los nombres de las columnas que contendrá cada fila.
- tipoDato indica el tipo de dato de cada columna, se define el dominio de valores para cada columna.
- Default valor, será un valor especificado para la columna que se utilizará si no se indica otro.

Ejemplo:

```
CREATE TABLE proveedores (nombre varchar2(50));
```

Sólo se podrá crear la tabla si el usuario tiene permisos para ello. Si la tabla pertenece a otro esquema (suponiendo que el usuario tenga permiso para crear tablas en ese otro esquema), se antepone al nombre de la tabla, el nombre del esquema.

```
CREATE TABLE otroUsuario.proveedores (nombre varchar2(50));
```

Se puede indicar un valor por defecto para un campo:

```
CREATE TABLE Proveedores (nombre varchar2(50),
                          localidad varchar2(50)DEFAULT 'Palencia');
```

De esta forma, si añadimos un proveedor y no indicamos su localidad, se tomará por defecto el valor 'Palencia'.

Tipos de datos

A la hora de crear tablas hay que especificar el tipo de datos de cada campo. En Oracle los tipos de datos son:

- **Textos**, para los textos tenemos los siguientes tipos:
  - **VARCHAR2**.Para textos de longitud variable, máximo 4000.
  - **CHAR**. Para textos de longitud fija, hasta 2000.
  - **NCHAR**. Para el almacenamiento de caracteres nacionales de longitud fija.

- **NVARCHAR2**. Para el almacenamiento de caracteres nacionales de longitud variable.

En los tipos anteriores se indican los tamaños entre paréntesis tras el nombre del tipo. Conviene reservar suficiente espacio para reservar los valores, en caso del tipo Varchar2, no se pierde espacio por reservar más del deseado, ya que si el texto es más pequeño que el tamaño indicado, el espacio sobrante se libera.

- **Números**, en Oracle el tipo NUMBER permite representar todo tipo de números. Su rango está entre 10^{-130} y $9{,}99999999999*10^{128}$. Fuera de estos límites Oracle devuelve error.

Los números decimales se indican con NUMBER(p,s), donde p es la precisión máxima y s es la escala, es decir, el número de decimales a la derecha de la coma. Por ejemplo, Number (8,3), permite representar hasta números de 8 cifras de las cuales 3 serían decimales (el decimal se representa con (.) y no con (,)).

La precisión debe incluir todos los dígitos del número, hasta 38, y la escala sólo los decimales, pero si es negativa indica ceros a la izquierda del decimal.

Para números enteros se indica NUMBER(p), siendo p el número de dígitos. Es equivalente a NUMBER(p,0).

Para números en coma flotante, (float o double en otros lenguajes), se indica NUMBER sin precisión ni escala.

Ejemplos:

Formato	Número escrito por el usuario	Se almacena como...
NUMBER	345255.345	345255.345
NUMBER(9)	345255.345	345255
NUMBER(9,2)	345255.345	345255.35
NUMBER(7)	345255.345	Da error de precisión
NUMBER(9,-2)	345255.345	345300
NUMBER(7,2)	345255.345	Da error de precisión

El cuarto ejemplo no da error de precisión

- **Fechas y horas**

- **DATE**. Este tipo permite almacenar fechas que se pueden escribir en formato día, mes y año entre comillas. El separador normalmente es una '/' o un '-'. Para almacenar la fecha actual, el sistema proporciona funciones que devuelven ese valor (SYSDATE).Las fechas no se pueden manejar directamente, normalmente se usan funciones de conversión como por ejemplo TO_DATE ('3/7/2015')

- **TIMESTAMP**, almacena valores de día, mes y año, junto con hora, minuto y segundos. Representa un instante concreto en el tiempo. Ejemplo: '2/2/2015 18:34:23,18'.

➢ **Datos de gran tamaño,** estos tipos de datos no pueden ser asociados a índices ni ser parte de claves. Pueden almacenar hasta datos de 4 GB. Son los siguientes:

- **BLOB,** permite almacenar datos binarios no estructurados.
- **CLOB,**almacena datos de tipo carácter.
- **NLOB,** almacena caracteres Unicode
- **BFILE,**almacena datos binarios en archivos de sistema operativo, fuera de la base de datos. La columna definida como BFILE guarda un localizador del archivo externo que contiene los datos.

➢ **Rowid,** representa una dirección de la base de datos ocupada por una única fila. Es el identificador de la fila dentro de la base de datos, realmente se guarda un puntero a una fila concreta. Se puede consultar con la sentencia Select, pero no modificar ni borrar.

➢ **Long, long raw y raw,**puede almacenar una cadena de caracteres de longitud variable de hasta 2GB. Estos tipos de datos están obsoletos y en su lugar se utilizarán los LOB.

4.2.3 Restricciones

Al crear una tabla se pueden establecer las restricciones para ella, es decir, las condiciones que una o varias columnas deben cumplir obligatoriamente.

Cada restricción creada llevará un nombre, si no lo indicamos, Oracle dará uno por defecto y será único por esquema. Si el nombre se lo damos nosotros **se recomienda que se indique nombre de la tabla**, campos involucrados y el tipo de restricción (**PK, FK, CK, UK, NN**). Por ejemplo: EMPLE_DNI_PK podría ser el nombre de la restricción que indica la clave primaria de la tabla EMPLE sobre el campo DNI. La sintaxis de la creación de una tabla especificando las restricciones es:

```
CREATE TABLE[esquema.]nombreTabla (
      Columna1 tipoDato [CONSTRAINT nombreRestricción]
      [NOT NULL][UNIQUE][PRIMARY KEY][FOREIGN KEY][DEFAULT valor]
      [REFERENCES nombreTabla [columna1 [, columna2]]
      [ON DELETE CASCADE][CHECK condicion]
      …
      columnaN tipoDato [CONSTRAINT nombreRestriccion][restricciones]);
```

Por ejemplo:

```
CREATE TABLE USUARIOS(
      Login varchar2(15) CONSTRAINT usu_log_pk PRIMARY KEY,
      password varchar2(8) NOT NULL,
      fechaIngreso DATE DEFAULT SYSDATE));
```

Se crea la tabla USUARIOS con tres campos y cada uno con una restricción, el primero es clave primaria, el segundo no puede ser nulo y el tercero toma un valor por defecto, la fecha del sistema.

- **NOT NULL**, se asocia a una columna y se obliga a que ésta tenga un valor, es decir, prohibe los valores nulos en esa columna.

```
password varchar2(8) NOT NULL, ó
password varchar2(8) CONSTRAINT usu_pas_nn NOT NULL
```

- **UNIQUE**, se asocia a una columna en la que interesa que **no se puedan repetir valores** y Oracle asocia automáticamente a ella un índice. Esta restricción se puede indicar a nivel de campo o a nivel de tabla, es decir, si se incluye en la línea de definición de un campo o si se indica después de definir todos los campos.

Ejemplos:

```
CREATE TABLE USUARIOS (
      login varchar2(15) CONSTRAINT usu_log_uk UNIQUE);
ó
CREATE TABLE USUARIOS (
      login varchar2(15) UNIQUE);
```

Si queremos poner la restricción a varios campos a la vez, por ejemplo que login y correo sean únicos se especificaría:

```
CREATE TABLE USUARIOS (
      login varchar2(25),
      correo varchar2(25),
CONSTRAINT usu_uk UNIQUE(login, correo));
```

- **Restricción PRIMARY KEY,** en el modelo relacional **todas las tablas deben tener una clave primaria** y **sólo puede existir una PK** por tabla aunque **puede estar compuesta** por **varios atributos**. Esta clave primaria podrá ser referenciada como clave ajena en otras tablas.

La clave primaria por defecto ya es NOT NULL y UNIQUE.

```
CREATE TABLE USUARIOS (
      login varchar2(15) CONSTRAINT usu_log_pk PRIMARY KEY);
ó
CREATE TABLE USUARIOS (
      login varchar2(15) PRIMARY KEY);
```

Si la clave primaria está formada por más de un campo:

```
CREATE TABLE USUARIOS (
      Nombre varchar2(25),
      apellidos varchar2(25),
      fecha_nacimiento DATE,
      CONSTRAINT usu_uk PRIMARY KEY(nombre, apellidos, fecha_nacimiento));
```

➢ Restricción REFERENCES. FOREIGN KEY

Una **clave ajena**, **secundaria o foránea** es un **campo** de una tabla que se **relaciona** con la **clave primaria** (o con la candidata de otra tabla). Al crear la tabla se indicará haciendo referencia a la tabla con la que se relaciona y los campos de donde procede.

Ejemplo: creamos una tabla que contiene información de alquileres de películas, el **cliente** que la alquila y la **película** alquilada, estos dos **atributos son FK** de otras dos tablas:

```
CREATE TABLE alquiler (
      dni varchar2(9) CONSTRAINT alq_dni_fk REFERENCES clientes(dni),
      cod_pelicula NUMBER(5) CONSTRAINT alq_pel_fk REFERENCES peliculas(cod),
      CONSTRAINT alq_pel_pk PRIMARY KEY (dni, cod_pelicula);
```

Si el campo al que se hace referencia es la clave primaria, se puede omitir el nombre de este campo:

```
CREATE TABLE alquiler (
      dni varchar2(9) CONSTRAINT alq_dni_fk REFERENCES clientes,
      cod_pelicula NUMBER(5) CONSTRAINT alq_pel_fk REFERENCES peliculas,
      CONSTRAINT alq_pel_pk PRIMARY KEY (dni, cod_pelicula);
```

Se establece una relación entre las tablas que obliga al cumplimiento de la **integridad referencial**. Esta norma obliga a que cualquier dni incluido en la tabla alquiler tenga que estar obligatoriamente en la tabla de clientes, si no es así, se producirá un error.

Cuando las claves ajenas están formadas por más de un campo se pueden definir a nivel de tabla y en este caso será necesario utilizar la expresión FOREIGN KEY:

```
CREATE TABLE existencias (
      Tipo char(9),
      Modelo number(3),
      N_almacen number(1),
      Cantidad number(7),
      CONSTRAINT exi_tm_fk FOREIGN KEY (tipo, modelo) REFERENCES piezas,
      CONSTRAINT exi_nal_fk FOREIGN KEY (n_almacen) REFERENCES almacenes,
      CONSTRAINT exi_pk PRIMARY KEY (tipo,modelo, n_almacen)
      );
```

Al establecer la relación entre dos tablas, la tabla que es referºenciada debe estar creada, primero se crearán las tablas que no tengan claves ajenas.

Si queremos borrar las tablas tendremos que proceder al contrario, borraremos primero las tablas que tengan las claves ajenas.

A la claúsula REFERENCES se pueden añadir varias opciones:

- **ON DELETE CASCADE**, permite borrar todos los registros cuya clave ajena sea igual a la clave del registro borrado.

- **ON DELETE SET NULL**, colocará el valor NULL en todas las claves ajenas relacionadas con la borrada.

➢ Restricción de validación

La restricción **CHECK** permite comprobar que los valores que se introducen en un campo son adecuados evaluando una condición asociada a dicha columna.

Supongamos la tabla USUARIOS, en su campo crédito sólo puede tomar valores entre 0 y 2000, se especificaría:

```
CREATE TABLE USUARIOS (
      credito NUMBER(4) CHECK (credito BETWEEN 0 AND 2000));
```

Una misma columna puede tener varios CHECK asociados, para ello especificamos un CONSTRAINT para cada una de la siguiente forma:

```
CREATE TABLE ingresos (
      cod NUMBER(5) PRIMARY KEY,
      concepto VARCHAR2(50) NOT NULL,
      importe NUMBER (11,2) CONSTRAINT importe_min CHECK (importe > 0)
                            CONSTRAINT importe_max CHECK (importe <8000));
```

En este ejemplo, se prohibe añadir datos cuyo importe no esté entre 0 y 8000.

Si queremos hacer referencia en una validación a otras columnas, hay que incluir la restricción a nivel de tabla, independiente de la columna:

```
CREATE TABLE ingresos (
      cod NUMBER(5) PRIMARY KEY,
      concepto VARCHAR2(50) NOT NULL,
      importe_max NUMBER (11,2),
      importe NUMBER (11,2),
       CONSTRAINT importe_max CHECK (importe < importe_max)
);
```

4.2.4 Eliminación de tablas

Para eliminar una tabla utilizaremos el comando DROP TABLE con la sintaxis:

```
DROP TABLE nombreTabla [CASCADE CONSTRAINTS];
```

Con esta instrucción, se borra la tabla con sus datos y cualquier información que existiera de ella en el diccionario de datos. Si especificamos la opción CASCADE en una tabla referenciada por otra, las restricciones definidas donde es clave ajena se borrarán antes y a continuación se eliminará la tabla, esta operación no será permitida por defecto y nos dará un error del tipo:

```
ORA-02449: unique/primary keys in table referenced by foreign keys
```

Oracle dispone de la orden TRUNCATE TABLE que permite borrar las filas de una tabla sin eliminar su estructura. La sintaxis es la siguiente:

```
TRUNCATE TABLE nombreTabla;
```

4.2.5 Modificación de tablas

Una vez creada una tabla es posible modificar su definición, añadir algún campo o restricción, cambiar el nombre de un campo, desactivar alguna restricción, etc. Estas operaciones se pueden realizar usando el comando **ALTER TABLE.**

- Cambiar el nombre de la tabla:

```
RENAME nombreViejo TO nombreNuevo;
```

- Añadir columnas a la tabla:

```
ALTER TABLE nombreTabla ADD
      (columnaNueva1 tipoDatos [propiedades],
      [, columnaNueva2 tipoDatos [propiedades]] … );
```

- Eliminar columnas de una tabla. Se eliminará la columna indicada sin poder deshacer esta acción. Se eliminarán todos los datos que contuviera. No se puede eliminar una columna si es la única de una tabla, será necesario borrar la tabla.

```
ALTER TABLE nombreTabla DROP COLUMN (columna1 [, columna2, …]);
```

- Modificar columnas de una tabla, su tipo de datos o las propiedades de la columna. Todos estos cambios son posibles si la tabla no contiene datos o si los datos guardados no ocupan todo el espacio reservado para ellos.

```
ALTER TABLE nombreTabla MODIFY (columna1 tipoDato [propiedades] [, columna2 …]);
```

- Si queremos renombrar columnas de una tabla:

```
ALTER TABLE nombreTabla RENAME COLUMN nombreAntiguo TO nombreNuevo;
```

Ejemplos: supongamos la tabla creada USUARIOS

```
CREATE TABLE usuarios (
      crédito NUMBER(4) CHECK (credito BETWEEN 0  AND 2000));
```

Si queremos incluir una nueva columna llamada USER que será de tipo texto y clave primaria:

```
ALTER TABLE usuarios ADD (user varchar2(20) PRIMARY KEY);
```

Cambiamos el nombre del campo:

```
ALTER TABLE usuarios RENAME COLUMN user TO login;
```

Con el comando ALTER también podemos modificar las restricciones asociadas a las tablas o eliminarlas.

- Para borrar restricciones:

```
ALTER TABLE nombreTabla DROP CONSTRAINT nombreRestricción;
```

- Para modificar el nombre de una restricción:

```
ALTER TABLE nombreTabla RENAME CONSTRAINT nombreViejo TO nombreNuevo;
```

- Para activar o desactivar restricciones:

```
ALTER TABLE nombreTabla DISABLE CONSTRAINT nombreRestricción [CASCADE];
```

La opción CASCADE desactiva las restricciones que dependan de ésta.

Para activar de nuevo la restricción:

```
ALTER TABLE nombreTabla ENABLE CONSTRAINT nombreRestricción [CASCADE];
```

4.2.6 Creación y eliminación de índices

Crear índices ayuda a la localización más rápida de la información contenida en las tablas. Para crearlos se utilizará la siguiente sintaxis:

```
CREATE INDEX  nombreIndice ON nombreTable (columna1, [, columna2 …])
```

No es aconsejable asociar índices a campos en tablas pequeñas o que se actualicen con mucha frecuencia. Se recomienda para campos que se usen en consultas de manera frecuente.

Para borrar índices creados:

```
DROP INDEX nombreIndice;
```

Se crean índices de manera implícita al especificar restricciones: PRIMARY KEY, FOREIGN KEY y UNIQUE.

4.3 Lenguaje de control de datos

Para acceder a la base de datos es necesario tener una cuenta de usuario. Al crear un usuario, se establece una clave que puede ser modificada por el administrador o por el propio usario. Estas claves se almacenan en una tabla del diccionario de datos llamada DBA_USERS.

Para crear usuarios utilizaremos la siguiente sentencia:

```
CREATE USER nombreUsuario
IDENTIFIED BY contraseña
[DEFAULT TABLESPACE tablespace]
[TEMPORARY TABLESPACE tablespace]
[QUOTA int {k | M} ON tablespace]
[QUOTA UNLIMITED ON tablespace]
[PROFILE perfil]
```

Donde:

- CREATE USER, crea un nombre de usuario que será identificado por el sistema.

- IDENTIFIED BY, permite dar una clave de acceso para el usuario.

- DEFAULT TABLESPACE, asigna al usuario el Tablespace por defecto para almacenar sus objetos. Si no se asigna otro, por defecto será SYSTEM.

- TEMPORARY TABLESPACE, especifica tablespace para trabajos temporales, por defecto será SYSTEM.

- QUOTA, asigna un espacio en Megabytes o Kilobytes en el Tablespace indicado. Si no se especifica espacio, el usuario no tendrá espacio y no podrá crear objetos.

- PROFILE, asigna un perfil al usuario. Si no se especifica, se asigna el perfil por defecto.

Para consultar todos los usuarios del sistema, consultaremos las tablas ALL_USERS y DBA_USERS.

Ejemplo: creamos un usuario con permisos limitados, no puede ni crear objetos ni guardar datos.

```
CREATE USER usuarioLimitado IDENTIFIED BY password;
```

Podemos modificar usuarios mediante la orden ALTER USER con la siguiente sintaxis:

```
ALTER USER nombreUsuario
IDENTIFIED BY contraseña
[DEFAULT TABLESPACE tablespace]
[TEMPORARY TABLESPACE tablespace]
[QUOTA int {k | M} ON tablespace]
[QUOTA UNLIMITED ON tablespace]
[PROFILE perfil]
```

Un usuario sin permisos de administrador podrá cambiar su clave.

Para eliminar un usuario usamos comando DROP USER:

```
DROP USER nombreUsuario [CASCADE];
```

La opción CASCADE borra todos los objetos del usuario antes de borrarlo. Nunca nos dejará borrar un usuario que tenga objetos creados.

Permisos

Ningún usuario puede realizar una operación si no se le ha asignado permiso para ello. Para acceder a los objetos de una BD necesitamos tener permisos, éstos pueden agruparse en **roles**. Los roles pueden activarse, desactivarse o protegerse con una clave. Mediante los roles podemos gestionar los comandos que pueden utilizar los usuarios. Un permiso puede asignarse a un usuario o a un rol.

Los privilegios **sobre objetos** se asignan con la orden GRANT con la siguiente sintaxis:

```
GRANT {privilegioObjeto [,privilegioObjeto]… |ALL [PRIVILEGES]}
ON [usuario.]objeto
TO {usuario1 | rol1 | PUBLIC} [,usuario2 | rol2 | PUBLIC] …
[WITH GRANT OPTION];
```

Donde:

- ON , especifica el objeto sobre el que se conceden privilegios

- TO, señala a los usuarios o roles a los que se conceden

- ALL, concede todos los privilegios sobre el objeto especificado
- [WITH GRANT OPTION], permite que el receptor del privilegio se lo asigne a otro
- PUBLIC, hace que el privilegio esté disponible para todos los usuarios

Ejemplos:

```
GRANT INSERT ON Empleados TO Ana; //Permite a Ana insertar en la tabla Empleados
GRANT ALL ON Empleados TO Ana; // Se conceden todos los privilegios sobre la tabla Empleados a Ana
```

Los privilegios **de sistema** son los que dan derecho a ejecutar comandos SQL y se asignan con la orden GRANT con la siguiente sintaxis:

```
GRANT {privilegio1 | rol1 [,privilegio2 | rol2]…}
TO {usuario1 | rol1 | PUBLIC} [,usuario2 | rol2 | PUBLIC] …
[WITH ADMIN OPTION];
```

Donde:

- TO, señala a los usuarios o roles a los que se conceden privilegios.
- WITH ADMIN OPTION, permite que el receptor de los privilegios pueda concederlos a otros usuarios o roles.
- PUBLIC, permite que un usuario esté disponible para todos los usuarios.

Ejemplos:

```
GRANT CONNECT TO Ana; //Concede a Ana el rol CONNECt y todos los privilegios que éste tiene asociados.
GRANT DROP USER TO Ana WITH ADMIN OPTION; // Concede a Ana el privilegio de borrar usuarios y que ellab pueda conceder el mismo privilegio a otros.
```

Para comprobar los distintos privilegios de sistema y de objeto que se pueden conceder, consultar las páginas:

http://ora.u440.com/usuarios/grant.html

http://www.redcientifica.com/oracle/c0004p0004.html

En Oracle la orden que permite retirar los permisos es **REVOKE** y su sintaxis es la siguiente:

Si los permisos son sobre objetos:

```
REVOKE {privilegioObjeto [,privilegioObjeto]… |ALL [PRIVILEGES]}
ON [usuario.]objeto
FROM {usuario1 | rol1 | PUBLIC} [,usuario2 | rol2 | PUBLIC] …
```

Si los permisos son de sistema o roles a usuarios:

```
REVOKE {privilegioSistema | rol1 [,privilegio2 | rol2]… | ALL [PRIVILEGES]}
FROM {usuario1 | rol1 | PUBLIC} [,usuario2 | rol2 | PUBLIC] …
```

Ejemplos:

```
REVOKE SELECT, UPDATE ON Empleados FROM Ana; //Se retira los permisos de seleccionar y actualizar la tabla Empleados a Ana
REVOKE DROP USER FROM Ana; // Se retira a Ana el permiso de borrar usuarios.
```