CREACIÓN

SQL (Structured Query Language)

Es un lenguaje declarativo de cuarta generación: define qué se desea hacer, y no cómo hacerlo en una base de datos relacional. Además de consultas permite la definición de la propia estructura de los datos, su manipulación, y la especificación de conexiones seguras.

Elementos y normas del lenguaje

- **COMANDOS**: Instrucciones que se pueden crear en SQL. Hay tres grupos:
  - De definición de datos (DDL, Data Definition Language), que permiten crear y definir nuevas bases de datos, tablas, campos, etc.
  - De manipulación de datos (DML, Data Manipulation Language), que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.
  - De control y seguridad de datos (DCL, Data Control Language), que administran los derechos y restricciones de los usuarios.
- **CLÁUSULAS:** Llamadas también condiciones o criterios, son palabras especiales que permiten modificar el funcionamiento de un comando.
- **OPERADORES**: Permiten crear expresiones complejas. Pueden ser aritméticos (+, -, *, /, ...) o lógicos (< , >, , < >, And, Or, etc.).
- **FUNCIONES:** Para conseguir valores complejos. Por ejemplo, la función promedio para obtener la media de un salario...
- **LITERALES**: Les podemos llamar también constantes y serán valores concretos, como por ejemplo un número, una fecha, un conjunto de caracteres, etc.

Y tendremos que seguir unas normas sencillas pero primordiales:

- Todas las instrucciones **terminan con un signo de punto y coma**.
- No se distingue entre mayúsculas y minúsculas.
- Cualquier comando puede ser partido con saltos de línea o espacios para facilitar su lectura.

Comandos

Comandos DDL. Lenguaje de Definición de Datos:

`CREATE`	Se utiliza para crear nuevas tablas, campos e índices.
`DROP`	Se utiliza para eliminar tablas e índices.
`ALTER`	Se utiliza para modificar tablas.

Comandos DML. Languaje de Manipulación de Datos:

`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.

Comandos DCL. Lenguaje de Control de Datos:

`GRANT`	Da permisos a uno o varios usuarios o roles para realizar tareas determinadas.
`REVOKE`	Permite eliminar permisos que previamente se han concedido con GRANT.

Cláusulas

`FROM`	Se utiliza para especificar la tabla de la que se van a seleccionar las filas.
`WHERE`	Especifica 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`	Ordena las filas seleccionadas de acuerdo a un orden específico.

Operadores

Operadores lógicos

AND	Evalúa dos condiciones y devuelve 'verdad' (TRUE) sólo si ambas son ciertas.
OR	Evalúa dos condiciones y devuelve 'verdad' si alguna de las dos es cierta.
NOT	Devuelve el valor contrario de la expresión.

Operadores de comparación

<	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

Existen muchas funciones distintas, aquí hay un ejemplo, funciones de agregado:

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

23/03/97	Literal fecha.
María	Literal caracteres.
5	Literal número.

Lenguaje de descripción de datos (DDL)

Una base de datos es un conjunto de objetos que nos servirán para gestionar los datos. Estos objetos están contenidos en esquemas y éstos a su vez suelen estar asociados a un usuario. De ahí que antes dijéramos que cada base de datos tiene un esquema que está asociado a un usuario.

La primera fase del trabajo con cualquier base de datos comienza con sentencias DDL, puesto que antes de poder almacenar y recuperar información debemos definir las estructuras donde almacenar la información. Las estructuras básicas con las que trabaja SQL son las tablas.
En Oracle, cada usuario de una base de datos tiene un esquema, que tendrá el mismo nombre que el usuario con el que se ha accedido, y sirve para crear, manipular y almacenar los objetos que posea ese usuario (tablas, vistas, índices u otros objetos relacionados).

Las instrucciones DDL generan acciones que no se pueden deshacer, por eso es conveniente usarlas con precaución y tener copias de seguridad cuando manipulamos la base de datos.

Creación de BBDD

Consiste en crear las tablas que la componen. Aunque antes de ésto tendríamos que definir un espacio de nombres separado para cada conjunto de tablas. Es lo que antes hemos llamado **esquemas** o **usuarios**. `CREATE DATABASE NombredemiBasedeDatos;`

Creación de tablas

Los objetos básicos con los que trabaja SQL son las tablas, que como ya sabemos es un conjunto de filas y columnas cuya intersección se llama celda. Es ahí donde se almacenarán los elementos de información, los datos que queremos recoger.

Antes de crear la tabla es conveniente planificar detalles: Qué nombre daremos a la tabla, qué nombre daremos a cada columna, qué tipo y tamaño de datos vamos a almacenar en cada columna, qué restricciones tenemos sobre los datos y alguna otra información adicional que necesitemos.
Y debemos tener en cuenta otras reglas que se deben cumplir para los nombres de las tablas:

- No podemos tener nombres de tablas duplicados en un mismo esquema (usuario).
- Deben comenzar por un carácter alfabético.
- Su longitud máxima es de 30 caracteres.
- Solo se permiten letras del alfabeto inglés, dígitos o el signo de guión bajo.
- No puede coincidir con las palabras reservadas de SQL.
- No se distingue entre mayúsculas y minúsculas.
- En el caso de que el nombre tenga espacios en blanco o caracteres nacionales (permitido sólo en algunas bases de datos), entonces se suele entrecomillar con comillas dobles. En el estándar SQL99 (respetado por Oracle) se pueden utilizar comillas dobles al poner el nombre de la tabla a fin de hacerla sensible a las mayúsculas (se diferenciará entre "USUARIOS"y "Usuarios").

```
CREATE TABLE [esquema.] nombredeTabla (
    columna1 Tipo_Dato,
    columna2 Tipo_Dato,
    ...
    columnaN Tipo_Dato );
```

Columna1, columna2, ..., columnaN son los nombres de las columna que contendrá la tabla.
Tipo_Dato indica el tipo de dato de cada columna.

```
CREATE TABLE USUARIOS (Nombre VARCHAR(25));
```

Tipos de datos en Oracle

CHAR	Cadena de caracteres (alfanuméricos) de longitud fija. Entre 1 y 2000 bytes como máximo. Aunque se introduzca un valor más corto que el indicado en el tamaño, se rellenará al tamaño indicado. Es de longitud fija, siempre ocupará lo mismo, independientemente del valor que contenga .
VARCHAR2	Cadena de caracteres de longitud variable. Entre 1 y 4000 bytes máximo. El tamaño del campo dependerá del valor que contenga, es de longitud variable.
VARCHAR	Cadena de caracteres de longitud variable. En desuso, se utiliza VARCHAR2.
NCHAR	Cadena de caracteres de longitud fija que sólo almacena caracteres Unicode. Entre 1 y 2000 bytes como máximo. El juego de caracteres del tipo de datos (datatype) NCHAR sólo puede ser AL16UTF16 ó UTF8. El juego de caracteres se especifica cuando se crea la base de datos Oracle
NVARCHAR2	Cadena de caracteres de longitud variable que sólo almacena caracteres Unicode Entre 1 y 4000 bytes como máximo. El juego de caracteres del tipo de datos (datatype) NCHAR sólo puede ser AL16UTF16 ó UTF8. El juego de caracteres se especifica cuando se crea la base de datos Oracle
LONG	Cadena de caracteres de longitud variable Como máximo admite hasta 2 GB (2000 MB). Los datos LONG deberán ser convertidos apropiadamente al moverse entre diversos sistemas. Este tipo de datos está obsoleto (en desuso), en su lugar se utilizan los datos de tipo LOB (CLOB, NCLOB). Oracle recomienda que se convierta el tipo de datos LONG a alguno LOB si aún se está utilizando. No se puede utilizar en claúsulas WHERE, GROUP BY, ORDER BY, CONNECT BY ni DISTINCT . Una tabla sólo puede contener una columna de tipo LONG. Sólo soporta acceso secuencial.
LONG RAW	Almacenan cadenas binarias de ancho variable. Hasta 2 GB. En desuso, se sustituye por los tipos LOB.
RAW	Almacenan cadenas binarias de ancho variable. . Hasta 32767 bytes. En desuso, se sustituye por los tipos LOB.
LOB (BLOG, CLOB, NCLOB, BFILE)	Permiten almacenar y manipular bloques grandes de datos no estructurados (tales como texto, imágenes, vídeos, sonidos, etc) en formato binario o del carácter

	Admiten hasta 8 terabytes (8000 GB). Soportan acceso aleatorio. Una tabla puede contener varias columnas de tipo LOB. Las tablas con columnas de tipo LOB no pueden ser replicadas.
BLOB	Permite almacenar datos binarios no estructurados Admiten hasta 8 terabytes.
CLOB	Almacena datos de tipo carácter. Admiten hasta 8 terabytes .
NCLOB	Almacena datos de tipo carácter. Admiten hasta 8 terabytes. Guarda los datos según el juego de caracteres Unicode nacional.
BFILE	Almacena datos binarios no estructurados en archivos del sistema operativo, fuera de la base de datos. Una columna BFILE almacena un localizador del archivo a uno externo que contiene los datos . Admiten hasta 8 terabytes. El administrador de la base de datos debe asegurarse de que exista el archivo en disco y de que los procesos de Oracle tengan permisos de lectura para el archivo.
ROWID	Almacenar la dirección única de cada fila de la tabla de la base de datos ROWID físico almacena la dirección de fila en las tablas, las tablas en clúster, los índices, excepto en las índices-organizados (IOT). ROWID lógico almacena la dirección de fila en tablas de índice-organizado (IOT). Un ejemplo del valor de un campo ROWID podría ser: "AAAIugAAJAAC4AhAAI". El formato es el siguiente: Para "OOOOOOFFFBBBBBBRRR", donde: OOOOOO: segmento de la base de datos (AAAIug en el ejemplo). Todos los objetos que estén en el mismo esquema y en el mismo segmento tendrán el mismo valor. FFF: el número de fichero del tablespace relativo que contiene la fila (fichero AAJ en el ejemplo). BBBBBB: el bloque de datos que contiene a la fila (bloque AAC4Ah en el ejemplo). El número de bloque es relativo a su fichero de datos, no al tablespace. Por lo tanto, dos filas con números de bloque iguales podrían residir en diferentes datafiles del mismo tablespace. RRR: el número de fila en el bloque (fila AAI). Este tipo de campo no aparece en los SELECT ni se puede modificar en los UPDATE, ni en los INSERT. Tampoco se puede utilizar en los CREATE. Es un tipo de datos utilizado exclusivamente por Oracle. Sólo se puede ver su valor utilizando la palabra reservada ROWID, por ejemplo: SELECT `rowid, nombre, apellidos FROM clientes ...` Ejemplo 2: `SELECT ROWID, SUBSTR(ROWID,15,4) "Fichero", SUBSTR(ROWID,1,8) "Bloque", SUBSTR(ROWID,10,4) "Fila" FROM proveedores ...` Ejemplo 3: una forma de saber en cuántos ficheros de datos está alojada una tabla: `SELECT COUNT(DISTINCT(SUBSTR(ROWID,7,3))) "Numero ficheros" FROM facturacion ...`
UROWID	ROWID universal Admite ROWID a tablas que no sean de Oracle, tablas externas. Admite tanto ROWID lógicos como físicos.
NUMBER	Almacena números fijos y en punto flotante Se admiten hasta 38 dígitos de precisión y son portables a cualquier entre los diversos sistemas en que funcione Oracle. Para declarar un tipo de datos NUMBER en un CREATE o UPDATE es suficiente con: nombre_columna NUMBER opcionalmente se le puede indicar la precisión (número total de dígitos) y la escala (número de dígitos a la derecha de la coma, decimales, los cogerá de la precisión indicada): nombre_columna NUMBER (precision, escala) . Si no se indica la precisión se tomará en función del número a guardar, si no se indica la escala se tomará escala cero. Para no indicar la precisión y sí la escala podemos utilizar: nombre_columna NUMBER (*, escala) . Para introducir números que no estén el el formato estándar de Oracle se puede utilizar la función TO_NUMBER.
FLOAT	Almacena tipos de datos numéricos en punto flotante. Es un tipo NUMBER que sólo almacena números en punto flotante
DATE	Almacena un punto en el tiempo (fecha y hora)

El tipo de datos DATE almacena el año (incluyendo el siglo), el mes, el día, las horas, los minutos y los segundos (después de medianoche).
Oracle utiliza su propio formato interno para almacenar fechas.
Los tipos de datos DATE se almacenan en campos de longitud fija de siete octetos cada uno, correspondiendo al siglo, año, mes, día, hora, minuto, y al segundo.
Para entrada/salida de fechas, Oracle utiliza por defecto el formato DD-MMM-AA.
Para cambiar este formato de fecha por defecto se utiliza el parámetro NLS_DATE_FORMAT.
Para insertar fechas que no estén en el mismo formato de fecha estándar de Oracle, se puede utilizar la función TO_DATE con una máscara del formato: TO_DATE (el "13 de noviembre de 1992", "DD del MES, YYYY")

`TIMESTAMP` Almacena datos de tipo hora, fraccionando los segundos

`TIMESTAMP WITH TIME ZONE` Almacena datos de tipo hora incluyendo la zona horaria (explícita), fraccionando los segundos

`TIMESTAMP WITH LOCAL TIME ZONE` Almacena datos de tipo hora incluyendo la zona horaria local (relativa), fraccionando los segundos. Cuando se usa un SELECT para mostrar los datos de este tipo, el valor de la hora será ajustado a la zona horaria de la sesión actual

`XMLType` Tipo de datos abstracto. En realidad se trata de un CLOB.
Se asocia a un esquema XML para la definición de su estructura.

Restricciones

Una **restricción** es una condición que una o varias columnas deben cumplir obligatoriamente.

```
CREATE TABLE NOMBRETABLA (
     Columna1 Tipo_Dato
          [CONSTRAINT nombredelarestricción]
          [NOT NULL]
          [UNIQUE]
          [PRIMARY KEY]
          [FOREIGN KEY]
          [DEFAULT valor]
          [REFERENCES nombreTabla [(columna [, columna ])]
          [ON DELETE CASCADE]]
          [CHECK condición],
     Columna2 Tipo_Dato
          ...);
```

Ejemplos :

```
     CREATE TABLE USUARIOS (
         Login VARCHAR(15) CONSTRAINT usu_log_PK PRIMARY KEY,
         Password VARCHAR (8) NOT NULL,
         Fecha_Ingreso DATE DEFAULT SYSDATE);

     CREATE TABLE USUARIOS (
         Login VARCHAR2 (25),
         Correo VARCHAR2 (25),
         CONSTRAINT Usuario_UK UNIQUE (Login, Correo));
```

Otra opción es definir las columnas de la tabla y después especificar las restricciones, de este modo podrás referir varias columnas en una única restricción.

Cada restricción que creemos llevará un nombre, si no se lo ponemos nosotros lo hará Oracle o el SGBD que estemos utilizando.
Es conveniente que le pongamos un nombre que nos ayude a identificarla y que sea único para cada

esquema (usuario). Es buena idea incluir de algún modo el nombre de la tabla, los campos involucrados y el tipo de restricción en el nombre de la misma. Oracle nos aconseja la siguiente regla a la hora de poner nombre a las restricciones: Tres letras para el nombre de la tabla, carácter de subrayado, tres letras con la columna afectada por la restricción, carácter de subrayado y dos letras con la abreviatura del tipo de restricción:

```
PK = Primary Key          FK = Foreign Key          NN = Not Null
UK = Unique               CK = Check (validación)
```

Restricción NOT NULL

Con esta restricción obligaremos a que esa columna tenga un valor o lo que es lo mismo, prohíbe los valores nulos para una columna en una determinada tabla.

```
CREATE TABLE USUARIOS (
    F_Nacimiento DATE
    CONSTRAINT Usu_Fnac_NN NOT NULL);
 CREATE TABLE USUARIOS (
    F_Nacimiento DATE NOT NULL);
```

Debemos tener cuidado con los valores nulos en las operaciones, ya que 1*NULL es igual a NULL.

Restricción UNIQUE

Habrá ocasiones en la que nos interese que no se puedan repetir valores en la columna, en estos casos utilizaremos la restricción `UNIQUE`. Oracle crea un índice automáticamente cuando se habilita esta restricción y lo borra al deshabilitarla.

```
CREATE TABLE USUARIOS (
    Login VARCHAR2 (25)
    CONSTRAINT Usu_Log_UK UNIQUE);

CREATE TABLE USUARIOS (
    Login VARCHAR2 (25) UNIQUE);
```

También podemos poner esta restricción a varios campos a la vez:

```
CREATE TABLE USUARIOS (
    Login VARCHAR2 (25),
    Correo VARCHAR2 (25),
    CONSTRAINT Usuario_UK UNIQUE (Login, Correo));
```

Detrás del tipo de datos de Correo hay una coma, eso es así porque la restricción es independiente de ese campo y común a varios. Por eso después de `UNIQUE` ponemos entre paréntesis los nombres de los campos a los que afecta la restricción.

Restricción PRIMARY KEY

En el mod. relacional las tablas deben tener una clave primaria. Al crear las tablas hay que indicarla. Sólo puede haber una clave primaria por tabla pero ésta puede estar formada por varios campos. Dicha clave podrá ser referenciada como clave ajena en otras tablas.
La clave primaria hace que los campos que forman sean `NOT NULL` y que los valores de los campos sean de tipo `UNIQUE`.

```
CREATE TABLE USUARIOS (
     Login VARCHAR2 (25) PRIMARY KEY);
```

O bien poniendo un nombre a la restricción:

```
CREATE TABLE USUARIOS (
     Login VARCHAR2 (25)
     CONSTRAINT Usu_log_PK PRIMARY KEY);
```

Si está formada por más de un campo, por ejemplo Nombre, Apellidos y Fecha de Nacimiento:

```
CREATE TABLE USUARIOS (
```

```
Nombre VARCHAR2 (25),
Apellidos VARCHAR2 (30),
F_Nacimiento DATE,
CONSTRAINT Usu_PK PRIMARY KEY(Nombre, Apellidos,
F_Nacimiento));
```

Restricciones REFERENCES. FOREIGN KEY

Ya vimos que las claves ajenas, secundarias o foráneas eran campos de una tabla que se relacionaban con la clave primaria (o incluso con la clave candidata) de otra tabla.
Cuando creemos la tabla tendremos que indicar de alguna forma quién es clave ajena. Lo haremos "haciendo referencia" a la tabla y los campos de donde procede.

En nuestra tabla vamos a tener una clave ajena procedente de la tabla PARTIDAS que será su Cod_Partida, por tanto tendremos que hacer referencia a éste:

```
CREATE TABLE USUARIOS (
    Cod_Partida NUMBER(8)
    CONSTRAINT Cod_Part_FK
    REFERENCES PARTIDAS(Cod_Partida));
```

Si el campo al que hace referencia es clave principal en su tabla, no es necesario indicar el campo:

```
CREATE TABLE USUARIOS (
    Cod_Partida NUMBER(8)
    CONSTRAINT Cod_Part_FK
    REFERENCES PARTIDAS);
```

Si la definición de la clave ajena se pone al final, tendremos que colocar el texto `FOREIGN KEY` para especificar a qué campo se está refiriendo. Vamos a verlo en el caso en que la clave ajena estuviera formada por Cod_Partida y Fecha de la partida de la tabla PARTIDAS:

```
CREATE TABLE USUARIOS (
    Cod_Partida NUMBER(8),
    F_Partida DATE,
CONSTRAINT Partida_Cod_F_FK FOREIGN KEY (Cod_Partida, F_Partida)
    REFERENCES PARTIDAS);
```

Integridad referencial

Al relacionar campos necesitamos que el dato del campo que es clave ajena en una tabla (que llamaremos secundaria) previamente haya sido incluido en su tabla de procedencia donde es clave primaria o candidata. En nuestro ejemplo, cualquier código de partida que incluyamos en la tabla USUARIO, debería estar previamente en la tabla de la que procede, es decir, en la tabla PARTIDAS. **A esto se le llama Integridad Referencial.**

Esto puede crear algunos errores, pues puede ocurrir lo siguiente:

- Si hacemos referencia a una tabla que no está creada: Oracle buscará la tabla referenciada y al no encontrarla dará fallo. Esto se soluciona creando en primer lugar las tablas que no tengan claves ajenas.
- Si queremos borrar las tablas tendremos que proceder al contrario, borraremos las tablas que tengan claves ajenas antes.

Tenemos otras soluciones y es añadir tras la cláusula `REFERENCE`:

- `ON DELETE CASCADE`: te permitirá 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.

Restricciones DEFAULT y CHECK

Si queremos asignar un valor por defecto a un campo:

```
CREATE TABLE USUARIOS (
    Pais VARCHAR2(20) DEFAULT 'España');
```

Podemos añadir distintas expresiones: constantes, funciones SQL y variables:

```
CREATE TABLE USUARIOS (
    Fecha_ingreso DATE DEFAULT SYSDATE);
```

También vamos a necesitar que se compruebe que los valores que se introducen son adecuados para ese campo. Para ello utilizaremos `CHECK`.
Esta restricción comprueba que se cumpla una condición determinada al rellenar una columna. Dicha condición se puede construir con columnas de esa misma tabla.

Si en la tabla USUARIOS tenemos el campo Crédito y éste sólo puede estar entre 0 y 2000, lo especificaríamos así:

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

Una misma columna puede tener varios `CHECK` asociados a ella, para ello ponemos varios `CONSTRAINT` seguidos y separados por comas.

DESC

Si queremos obtener una descripción de una tabla, sinonimo, paquete o función, podemos utilizar el comando DESCRIBE o DESC. Debido a que es un comando de sqlplus no necesita terminar con ;
DESC table DESC view DESC synonym DESC function DESC package

Eliminación de tablas

Esta instrucción borrará la tabla de la base de datos incluido sus datos (filas). También se borrará toda la información que existiera de esa tabla en el Diccionario de Datos.

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

La opción `CASCADE CONSTRAINTS` se puede incluir para los casos en que alguna de las columnas sea clave ajena en otra tabla secundaria, lo que impediría su borrado. Al usarla, las restricciones donde es clave ajena se borrarán antes y a luego se eliminará la tabla en cuestión.

```
DROP TABLE USUARIOS;
```

El borrado de una tabla es irreversible y no hay una petición de confirmación antes de ejecutarse.

Modificación de tablas

Tras crear una tabla puede que necesitemos añadir o cambiar el nombre de un campo, añadir o quitar una restricción...: `RENAME NombreViejo TO NombreNuevo;`
Si queremos añadir columnas a una tabla: las columnas se añadirán al final de la tabla.

```
ALTER TABLE NombreTabla ADD
( ColumnaNueva1 Tipo_Datos [Propiedades]
[, ColumnaNueva2 Tipo_Datos [Propiedades]
... );
```

Si queremos eliminar columnas de una tabla: se eliminará la columna indicada sin poder deshacer esta acción. Además de la definición de la columna, se eliminarán todos los datos que contuviera. No se puede eliminar una columna si es la única que forma la tabla, para ello tendremos que borrar la tabla directamente.

```
ALTER TABLE NombreTabla DROP COLUMN (Columna1 [, Columna2, ...]);
```

Si queremos modificar columnas de una tabla: podemos modificar el tipo de datos y las propiedades de una columna. Todos los cambios son posibles si la tabla no contiene datos. En general, si la tabla no está vacía podremos aumentar la longitud de una columna, aumentar o disminuir en número de posiciones decimales en un tipo NUMBER, reducir la anchura siempre que los datos no ocupen todo el espacio reservado para ellos.

```
ALTER TABLE NombreTabla MODIFY
(Columna1 TipoDatos [propiedades] [, columna2 TipoDatos
[propiedades] ...] );
```
Si queremos renombrar columnas de una tabla:
```
ALTER TABLE NombreTabla RENAME COLUMN NombreAntiguo TO
NombreNuevo;
```
Tenemos la siguiente tabla creada:
```
CREATE TABLE USUARIOS (
   Credito NUMBER(4) CHECK (Crédito BETWEEN 0 AND 2000));
```
Nos gustaría incluir una nueva columna llamada User que será tipo texto y clave primaria:
```
ALTER TABLE USUARIO ADD
   (User VARCHAR(10) PRIMARY KEY);
```
Nos damos cuenta que ese campo se llamaba Login y no User, vamos a cambiarlo:
```
ALTER TABLE USUARIO RENAME COLUMN User TO Login;
```
Ejemplo. Añadiendo una restricción:
```
ALTER TABLE EMPLEADOS MODIFY (Sueldo NUMBER(4)
CHECK (Sueldo BETWEEN 1000 AND 1200));
```
Utilizando el comando `ALTER TABLE`, podemos modificar las restricciones o bien eliminarlas:
Si queremos borrar restricciones:
```
ALTER TABLA NombreTabla DROP CONSTRAINT NombreRestriccion;
```
Si queremos modificar el nombre de las restricciones:
```
ALTER TABLE NombreTabla RENAME CONSTRAINT NombViejo TO NombNuevo;
```
Si queremos activar o desactivar restricciones:
A veces es conveniente desactivar temporalmente una restricción para hacer pruebas o porque necesitemos saltarnos esa regla. Para ello usaremos esta sintaxis:
```
ALTER TABLE NomTabla DISABLE CONSTRAINT NombRestriccion [CASCADE];
```
`CASCADE` desactiva las restricciones que dependan de ésta. Para activar de nuevo la restricción:
```
ALTER TABLE NombTabla ENABLE CONSTRAINT NombRestriccion [CASCADE];
```

Ver todas las restricciones
```
SELECT * FROM all_constraints
```

Creación y eliminación de índices

Sabemos que crear índices ayuda a la localización más rápida de la información contenida en las tablas. Ahora aprenderemos a crearlos y eliminarlos: `CREATE INDEX NombreIndice ON NombreTabla (Columna1 [, Columna2 ...]);`
No es aconsejable utilizar campos de tablas pequeñas o que se actualicen con mucha frecuencia. Tampoco es conveniente si esos campos no se usan en consultas de manera frecuente o en expresiones. Una mala elección ocasiona ineficiencia y tiempos de espera elevados. Un uso excesivo de ellos puede dejar a la Base de Datos colgada simplemente con insertar alguna fila.
Para eliminar un índice es suficiente con poner la instrucción: `DROP INDEX NombreIndice;`
La mayoría de los índices se crean de manera implícita cuando ponemos las restricciones PRIMARY KEY, FOREIGN KEY o UNIQUE.
ejemplo: Crea un índice con el campo Apellidos, luego elimínalo.
```
CREATE INDEX miIndice ON EMPLEADOS (Apellidos);
DROP INDEX miIndice;
```

Lenguaje de control de datos DCL

Necesitamos una cuenta de usuario para acceder a los datos de una base de datos. Las claves de acceso se establecen cuando se crea el usuario y pueden ser modificados por el Administrador o por el propietario de dicha clave. La Base de Datos almacena encriptadas las claves en una tabla del diccionario llamada DBA_USERS. Sintáxis:

```
CREATE USER NombreUsuario
IDENTIFIED BY ClaveAcceso
[DEFAULT TABLESPACE tablespace ]
[TEMPORARY TABLESPACE tablespace]
[QUOTA int {K | M} ON tablespace]
[QUOTA UNLIMITED ON tablespace]
[PROFILE perfil];
```

`CREATE USER`: crea un nombre de usuario que será identificado por el sistema.
`IDENTIFIED BY`: permite dar una clave de acceso al usuario creado.
`DEFAULT TABLESPACE`: asigna a un usuario el Tablespace por defecto para almacena los objetos que cree. Si no se asigna ninguna, será `SYSTEM`.
`TEMPORARY TABLESPACE`: nombre del Tablespace para trabajos temporales. Por def. `SYSTEM`.
`QUOTA`: asigna un espacio en Megabytes o Kilobytes en el Tablespace asignado.
Si no se especifica 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.

Recuerda que para crear usuarios debes tener una cuenta con privilegios de Administrador.
Para ver todos los usuarios creados utilizamos las vistas ALL_USERS y DBA_USERS. Y para ver en mi sesión los usuarios que existen pondría: `DESC SYS.ALL_USERS;`

Ejemplo. Creemos una cuenta de usuario limitado, que no tenga derecho ni a guardar datos ni a crear objetos, más tarde le daremos permisos:

```
CREATE USER UsuarioLimitado IDENTIFIED BY password;
```

Podemos modificar usuarios mediante el comando ALTER USER, cuya sintaxis es la siguiente:

```
ALTER USER NombreUsuario
IDENTIFIED BY clave_acceso
[DEFAULT TABLESPACE tablespace ]
[TEMPORARY TABLESPACE tablespace]
[QUOTA int {K | M} ON tablespace]
[QUOTA UNLIMITED ON tablespace]
[PROFILE perfil];
```

Un usuario sin privilegios de Administrador únicamente podrá cambiar su clave de acceso.

Para eliminar o borrar un usuario utilizamos el comando `DROP USER` con la siguiente sintaxis:

```
DROP USER NombreUsuario [CASCADE];
```

La opción `CASCADE` borra todos los objetos del usuario antes de borrarlo. Sin esta opción no nos dejaría eliminar al usuario si éste tuviera tablas creadas.

Ningún usuario puede llevar a cabo una operación si antes no se le ha concedido su permiso.
Para poder acceder a los objetos de una base de datos necesitas tener privilegios (permisos). Éstos se pueden agrupar formando **roles**, lo que simplificará la administración. Los roles pueden activarse, desactivarse o protegerse con una clave. Mediante los roles podemos gestionar los comandos que pueden utilizar los usuarios. Un permiso se puede asignar a un usuario o a un rol.

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

ON especifica el objeto sobre el que se conceden los privilegios.
TO señala a los usuarios o roles a los que se conceden privilegios.
ALL concede todos los privilegios sobre el objeto especificado.
[WITH GRANT OPTION] permite que el receptor del privilegio se lo asigne a otros.
PUBLIC hace que un privilegio esté disponible para todos los usuarios.

En el siguiente ejemplo Juan ha accedido a la base de datos y ejecuta los siguientes comandos:
GRANT INSERT TO Usuarios TO Ana; (permitirá a Ana insertar datos en Usuarios)
GRANT ALL ON Partidas TO Ana; (Juan da todos los privilegios sobre Partidas a Ana)

Los privilegios **de sistema** son los que dan derecho a ejecutar comandos SQL o acciones sobre objetos de un tipo especificado. Existen gran cantidad de privilegios distintos. Sintáxis:

```
GRANT {Privilegio1 | rol1 } [, privilegio2 | rol2}, ...]
TO {usuario1 | rol1| PUBLIC} [, usuario2 | rol2 | PUBLIC} ... ]
[WITH ADMIN OPTION];
```

TO señala a los usuarios o roles a los que se conceden privilegios.
WITH ADMIN OPTION es una opción que permite al receptor de esos privilegios que pueda conceder esos mismos privilegios a otros usuarios o roles.
PUBLIC hace que un privilegio esté disponible para todos los usuarios.
Ejemplo: Concede a Ana el rol de CONNECT con todos los privilegios que éste tiene asociados.

```
GRANT CONNECT TO Ana;
```

Concede a Ana borrar usuarios y que ésta puede conceder el privilegio de borrar usuarios a otros.

```
GRANT DROP USER TO Ana WITH ADMIN OPTION;
```

Para retirar privilegios

Sobre objetos:

```
REVOKE {privilegio_objeto [, privilegio_objeto]...|ALL|
[PRIVILEGES]}
ON [usuario.]objeto
FROM {usuario|rol|PUBLIC} [,{usuario|rol|PUBLIC] ...;
```

Del sistema o roles a usuarios:

```
REVOKE {privilegio_stma | rol} [, {privilegio_stma | rol}]...|ALL|
[PRIVILEGES]}
ON [usuario.]objeto
FROM {usuario|rol|PUBLIC} [,{usuario|rol|PUBLIC] ...;
```

Ejemplo: Juan va a quitar el permiso de seleccionar y de actualizar sobre la tabla Usuarios a Ana:

```
REVOKE SELECT, UPDATE ON Usuarios FROM Ana;
```

y va a quitarle el permiso de eliminar usuarios:

```
REVOKE DROP USER FROM Ana;
```