

Sitio:	<a href="#">ALONSO DE AVELLANEDA</a>
Curso:	Bases de datos DAM Vesp
Libro:	3.B. SQL

Imprimido por:	Oliver Bitica
Día:	lunes, 12 de enero de 2026, 10:35

## Tabla de contenidos

### 1. Introducción

### 2. Elementos del lenguaje. Normas de escritura.

### 3. Lenguaje de descripción de datos (DDL)

3.1. Creación de bases de datos. Objetos de la base de datos

3.2. Creación de tablas

3.3. Restricciones

3.4. Restricción NOT NULL

3.5. Restricción UNIQUE

3.6. Restricción PRIMARY KEY

3.7. Restricción Clave ajena. REFERENCES. FOREIGN KEY.

3.8. Restricciones DEFAULT y CHECK

3.9. Eliminación de tablas - DROP TABLE

3.10. Modificación de tablas - ALTER TABLE

3.11. Modificación de tablas - ALTER TABLE sobre restricciones

3.12. Creación y eliminación de Indices - CREATE INDEX y DROP INDEX

# 1. Introducción

**SQL** (Structured Query Language) es el lenguaje fundamental de los SGBD relacionales. Es uno de los lenguajes más utilizados en informática en todos los tiempos. Es un lenguaje declarativo y por tanto, lo más importante es definir **qué** se desea hacer, y **no cómo** hacerlo. De esto último ya se encarga el SGBD.

Hablamos por tanto de un lenguaje normalizado que se puede utilizar con cualquier tipo de lenguaje (Java, C#, PHP) en combinación con cualquier tipo de base de datos (Access, SQL Server, MySQL, Oracle, etc.).

El hecho de que sea estándar no quiere decir que sea idéntico para cada base de datos. Así es, determinadas bases de datos implementan funciones específicas que no tienen necesariamente que funcionar en otras.

Aunque SQL está estandarizado, siempre es recomendable revisar la documentación del SGBD con el que estemos trabajando para conocer su sintaxis concreta, ya que algún comando, tipo de dato, etc., puede no seguir el estándar.

SQL posee dos características muy apreciadas, **potencia y versatilidad**, que contrastan con su facilidad para el aprendizaje, ya que utiliza un lenguaje bastante natural. Es por esto que las instrucciones son muy parecidas a órdenes humanas. Por esta característica se le considera un Lenguaje de Cuarta Generación.

Aunque frecuentemente oigas que SQL es un "lenguaje de consulta", ten en cuenta que no es exactamente cierto ya que contiene muchas otras capacidades además de la de consultar la base de datos:

- La definición de la propia estructura de los datos,
- Su manipulación,
- La especificación de conexiones seguras.

Por tanto, el lenguaje estructurado de consultas SQL es un lenguaje que permite operar con los datos almacenados en las bases de datos relacionales.

## Para saber más

Ya hemos llegado a los lenguajes de quinta generación, en el siguiente enlace puedes ver sus características más generales - [Generaciones de lenguajes de programación](#)

En este enlace encontrarás de una manera breve, pero interesante, la historia del SQL. - [Historia de SQL](#).

## 2. Elementos del lenguaje. Normas de escritura.

Imagínate que cada programador utilizará sus propias reglas para escribir. Esto sería un caos. Es muy importante establecer los elementos con los que vamos a trabajar y unas normas que seguir.

El lenguaje SQL está compuesto 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. Estos conceptos son bastante amplios por eso será mejor que vayamos por partes.

- **COMANDOS:** Van a ser las instrucciones que se pueden crear en SQL. Se pueden distinguir en tres grupos que veremos con más detenimiento a lo largo de las siguientes unidades:
  - 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 y comprensión.
- Los comentarios comienzan por /\* y terminan con \*/ (excepto en algunos SGBD).

### Para saber más

Otro Sistema Gestor de Base de Datos muy utilizado en algunos entornos como el de desarrollo web es MySQL. - [Documentación MySQL](#).

Otra página recomendable donde puedes aprender MySQL desde cero es la siguiente - [MySQL con Clase](#).

### 3. Lenguaje de descripción de datos (DDL)

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.

Conocer el **Lenguaje de Definición de Datos (DDL)** es imprescindible para crear, modificar y eliminar objetos de la base de datos (es decir, los metadatos). En el mercado hay suficientes aplicaciones y asistentes que nos facilitan esta labor, a través de una interfaz visual que nos oculta el lenguaje **SQL** y en los cuales nos limitamos a poner nombres a los campos, elegir el tipo de datos y activar una serie de propiedades.

Es cierto que estas herramientas nos facilitan el trabajo, pero resulta imprescindible comprender y conocer en profundidad el lenguaje, ya que nos veremos en muchas situaciones donde necesitaremos crear un objeto, modificarlo o eliminarlo sin depender de esas herramientas visuales.

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 almacenar los objetos que posea ese usuario.

¿De qué objetos estamos hablando? Éstos podrán ser tablas, vistas, índices u otros objetos relacionados con la definición de la base de datos. ¿Y quién puede crear y manipularlos? En principio el usuario propietario (el que los creó) y los administradores de la base de datos. Más adelante veremos que podemos modificar los privilegios de los objetos para permitir el acceso a otros usuarios.

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.

#### Para saber más

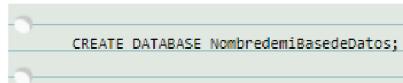
Si quieras saber un poco más sobre el Lenguaje de Definición de Datos, puedes visitar la Wikipedia, aquí tienes el enlace - [Lenguaje de Definición de Datos](#)

## 3.1. Creación de bases de datos. Objetos de la base de datos

Básicamente, la creación de la base de datos 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**.

Crear una base de datos implica indicar los archivos y ubicaciones que se van a utilizar además de otras indicaciones técnicas y administrativas. Es obvio que todo esto sólo lo puede realizar si se tiene privilegio de Administrador.

Con el estándar de SQL la instrucción a usar sería **Create Database**, pero cada SGBD tiene un procedimiento para crear las bases de datos. Crearíamos una base de datos con el nombre que se indique a continuación.



Hemos estado hablando de objetos de la base de datos, ahora veremos a qué nos referimos.

Según los estándares, **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.

### Para saber más

[Sentencia CREATE DATABASE - Manual oficial de Oracle](#)

## 3.2. Creación de tablas

Qué necesitamos para poder guardar los datos? Lo primero será definir los objetos donde vamos a agrupar esos datos. 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 algunos detalles:

- **Qué nombre** le vamos a dar a la tabla.
- **Qué nombre** le vamos a dar a cada una de las **columnas**.
- **Qué tipo y tamaño de datos** vamos a almacenar en cada columna.
- **Qué restricciones** tenemos sobre los datos.
- 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 (por ejemplo, no podemos llamar a una tabla WHERE).
- 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 entremillar 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").

La sintaxis básica del comando que permite crear una tabla es la siguiente:



donde:

- columna1, columna2, ..., columnaN son los nombres de las columnas que contendrá la tabla.
- Tipo\_Dato indica el tipo de dato de cada columna.
- [esquema] indica que esquema es opcional. (Oracle define esquema como la **colección de objetos o estructuras lógicas que corresponden directamente a los datos almacenados**, y crea un nuevo esquema por cada usuario que crea objetos en la base de datos.)

Veamos un ejemplo en el que crearemos una tabla llamada USUARIOS con un solo campo denominado NOMBRE de tipo VARCHAR(25).

A screenshot of a database interface showing the execution of a SQL command. The command is 'CREATE TABLE USUARIOS (Nombre VARCHAR(25));' and it has been successfully executed.

**Recuerda** que solo podrás crear tablas si posees los permisos necesarios para ello.

Durante nuestro aprendizaje vamos a tener que crear muchas tablas, para ello necesitaremos manejar los tipos de datos que utiliza Oracle. En el siguiente enlace tienes una relación de estos tipos y su descripción. [Tipos de datos Oracle](#).

### Para saber más

MySQL trabaja con otros tipos de datos. Si quieres conocerlos puedes entrar en este enlace - [Tipos de datos en MySQL](#).

### 3.3. Restricciones

Hay veces que necesitamos que un dato se incluya en una tabla de manera obligatoria, otras veces necesitaremos definir uno de los campos como llave primaria o ajena. Todo esto podremos hacerlo cuando definamos la tabla, además de otras opciones.

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

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. La sintaxis en SQL estándar es la siguiente:

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

Por ejemplo:

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

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.

En los siguientes apartados veremos cada una de las restricciones, su significado y su uso.

#### Recomendación

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.
- Dos letras con la abreviatura del tipo de restricción. La abreviatura puede ser:
  - PK = Primary Key.
  - FK = Foreign Key.
  - NN = Not Null.
  - UK = Unique.
  - CK = Check (validación).

## 3.4. 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.

Podremos ponerlo cuando creamos o modificamos el campo añadiendo la palabra **NOT NULL** después de poner el tipo de dato.

Si en la tabla USUARIOS queremos que el campo "F\_Nacimiento" sea obligatorio ponerlo, nos quedaría así:

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

o bien, de esta otra forma:

```
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.

### 3.5. 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.

También para esta restricción tenemos dos posibles formas de ponerla, veámoslo con un ejemplo. Supongamos que el campo Login de nuestra tabla va a ser único. Lo incluiremos en la tabla que estamos creando. Nos quedaría así:

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

Veamos otra forma:

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

También podemos poner esta restricción a varios campos a la vez, por ejemplo, si queremos que Login y correo electrónico sean únicos podemos ponerlo así:

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

Si te fijas, 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** hemos puesto entre paréntesis los nombres de los campos a los que afecta la restricción.

## 3.6. Restricción PRIMARY KEY

En el modelo relacional las tablas deben tener una clave primaria. Es evidente que cuando creamos la tabla tendremos que indicar a quién corresponde.

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**.

Veamos como quedaría la si la clave fuese el campo Login:

- Si la clave la forma un único campo:

```
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 la clave 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));
```

### 3.7. Restricción Clave ajena. 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 creamos 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 nombre del 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);
```

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.

**Integridad referencial:** La integridad referencial es propiedad de la base de datos. La misma significa que la clave externa de una tabla de referencia siempre debe aludir a una fila válida de la tabla a la que se haga referencia.

**On delete cascade:** Hace que al borrar registros en la tabla principal se borren registros en la tabla relacionada.

**On delete set null:** Hace que si se borra un registro en la tabla principal, los valores de los campos relacionados se coloquen a null.

## 3.8. Restricciones DEFAULT y CHECK

A veces es muy tedioso insertar siempre lo mismo en un campo. Imagínate que casi todos los jugadores fuesen de España y tenemos un campo País. ¿No sería cómodo asignarle un valor por defecto? Eso es lo que hace la restricción **DEFAULT**.

En nuestro ejemplo vamos a añadir a la tabla USUARIOS el campo País y le daremos por defecto el valor "España".

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

En las especificaciones de **DEFAULT** vamos a poder añadir distintas expresiones: constantes, funciones SQL y variables.

Si queremos incluir en un campo la fecha actual, independientemente del día en el que estemos, podemos utilizar la función **SYSDATE** como valor por defecto:

```
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 llenar 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 (
    Crédito NUMBER(4) CHECK (Crédito BETWEEN 0 AND 2000));
```

### Para saber más

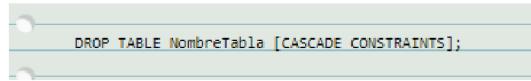
Si queremos obtener una descripción de una tabla, sinónimo, paquete o función, podemos utilizar el comando **DESCRIBE**

Oracle Documentación Oficial - [Comando DESCRIBE](#)

## 3.9. Eliminación de tablas - DROP TABLE

Cuando una tabla ya no es útil y no la necesitamos es mejor borrarla, de este modo no ocupará espacio y podremos utilizar su nombre en otra ocasión.

Para eliminar una tabla utilizaremos el comando `DROP TABLE`.



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.

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 colocar esta opción las restricciones donde es clave ajena se borrarán antes y a continuación se eliminará la tabla en cuestión.

Vamos a eliminar la tabla con la que hemos estado trabajando:



Ten cuidado al utilizar este comando, el borrado de una tabla es irreversible y no hay una petición de confirmación antes de ejecutarse.

Al borrar una tabla:

- Desaparecen todos sus datos
- Cualquier vista asociada a esa tabla seguirá existiendo pero ya no funcionará.

Oracle dispone de la orden `TRUNCATE TABLE` que te permitirá eliminar los datos (filas) de una tabla sin eliminar su estructura.

Y recuerda que solo podrás borrar aquellas tablas sobre las que tengas permiso de borrado.

## 3.10. Modificación de tablas - ALTER TABLE

Es posible que después de crear una tabla nos demos cuenta que se nos ha olvidado añadir algún campo o restricción, quizás alguna de las restricciones que añadimos ya no es necesaria o tal vez queramos cambiar el nombre de alguno de los campos. ¿Es posible esto? Ahora veremos que sí y en casi todos los casos utilizaremos el comando ALTER TABLE.

- Si queremos cambiar el nombre de una tabla:

```
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;
```

### Ejemplo

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;
```

Ejercicio resuelto

Tenemos creada la siguiente tabla:

```
CREATE TABLE EMPLEADOS (
    Cod_Cliente VARCHAR(5) PRIMARY KEY,
    Nombre VARCHAR(10),
    Apellidos VARCHAR(25),
    Sueldo NUMBER(2));
```

Ahora queremos poner una restricción a sueldo para que tome valores entre 1000 y 1200, ¿cómo lo harías?

```
ALTER TABLE EMPLEADOS MODIFY (Sueldo NUMBER(2) CHECK (Sueldo BETWEEN 1000 AND 1200));
```

## 3.11. Modificación de tablas - ALTER TABLE sobre restricciones

Utilizando el comando **ALTER TABLE**, podemos modificar las restricciones o bien eliminarlas:

- Si queremos borrar restricciones:

```
ALTER TABLE NombreTabla DROP CONSTRAINT NombreRestriccion;
```

- Si queremos modificar el nombre de las restricciones:

```
ALTER TABLE NombreTabla RENAME CONSTRAINT NombreViejo TO NombreNuevo;
```

- 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 NombreTabla DISABLE CONSTRAINT NombreRestriccion [CASCADE];
```

La opción **CASCADE** desactiva las restricciones que dependan de ésta.

Para activar de nuevo la restricción:

```
ALTER TABLE NombreTabla ENABLE CONSTRAINT NombreRestriccion [CASCADE];
```

**Si queremos añadir restricciones:**

La sintaxis básica de un comando **ALTER TABLE** para **AÑADIR CLAVE PRINCIPAL** a una tabla es la siguiente.

```
ALTER TABLE table_name ADD CONSTRAINT FK_MyPrimaryKey PRIMARY KEY (column1, column2...);
```

La sintaxis básica de un comando **ALTER TABLE** para **AÑADIR CLAVE FORANEA** a una tabla es la siguiente.

```
ALTER TABLE table_hijo ADD CONSTRAINT FK_MyForeignKey FOREIGN KEY (c1) REFERENCES tabla_padre (p1);
```

Tabla hijo es la tabla que contendrá la clave externa.

c1 es la columna de la tabla hija

p1 es la columna de la tabla padre .

La sintaxis básica de un comando **ALTER TABLE** para **AGREGAR RESTRICCIÓN CHECK (DE VALIDACIÓN)** a una tabla es la siguiente.

```
ALTER TABLE table_name ADD CONSTRAINT Constraint_name CHECK (CONDITION);
```

La sintaxis básica de un comando **ALTER TABLE** para agregar restricciones Unicas :

```
ALTER TABLE TABLE_NAME ADD CONSTRAINT constraint_name UNIQUE (column1, column2, ... column_n);
```

[Aregar restricciones](#)

[Comando ALTER TABLE](#)

Puede ocurrir que no hayamos puesto nombre a las restricciones o bien que lo hiciéramos pero no lo recordemos. Sería interesante que se pudiera consultar en algún lado.

[Nombre de todas las restricciones.](#)

## 3.12. Creación y eliminación de Indices - CREATE INDEX y DROP INDEX

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 que utilices 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.

El diseño de indices es un tema bastante complejo para los Administradores de Bases de Datos, ya que 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.

### Ejercicio resuelto

Tenemos creada la siguiente tabla:

```
CREATE TABLE EMPLEADOS (
    Cod_Cliente VARCHAR(5) PRIMARY KEY,
    Nombre VARCHAR(10),
    Apellidos VARCHAR(25),
    sueldo NUMBER(2));
```

Crea un índice con el campo Apellidos, luego elimínalo.

La solución sería:

```
CREATE INDEX miIndice ON EMPLEADOS (Apellidos);
```

```
DROP INDEX miIndice;
```