

UNIDAD 5 DISEÑO FÍSICO DE UNA BASE DE DATOS

1 INTRODUCCIÓN AL LENGUAJE SQL

- 1.1 ¿Cómo se usa SQL?
- 1.2 ¿Qué podemos hacer con SQL?
- 1.3 Componentes del lenguaje SQL
- 1.4 Notación en las instrucciones
- 1.5 Normas de escritura

2 ELEMENTOS DEL LENGUAJE SQL

- 2.1 Tipos de datos
- 2.2 Variables
- 2.3 Identificadores
- 2.4 Constantes
- 2.5 Comentarios
- 2.6 Operadores y expresiones

3 LENGUAJE DDL

- 3.1 Introducción
- 3.2 Creación de bases de datos
 - 3.2.1 Creación de bases de datos en MySQL
 - 3.2.2 Creación de bases de datos en Oracle
- 3.3 Modificación de una base de datos
- 3.4 Borrado de bases de datos
- 3.5 Creación de tablas
 - 3.5.1 Creación de tablas en MySQL
 - 3.5.2 Creación de tablas en Oracle
- 3.6 Implementación de restricciones de tabla y columna
- 3.7 Constraints o restricciones
- 3.8 Consultar las tablas de una base de datos
- 3.9 Modificación de tablas
- 3.10 Borrado de tablas
- 3.11 Renombrado de tablas

4 BIBLIOGRAFÍA

1 INTRODUCCIÓN AL LENGUAJE SQL

En las unidades anteriores hemos visto que mediante el modelo Entidad-Relación se pueden modelar situaciones del mundo real, de manera que disponemos de una herramienta gráfica para trasladar los elementos de un sistema de información y sus relaciones, aun esquema manejable que puede ser fácilmente interpretado por cualquiera que conozca las reglas por las que se rige. Luego hemos comprobado cómo se puede trasladar ese modelo ER a otro esquema de información más orientado a su tratamiento, como es el modelo relacional.

Para almacenar y tratar la información esquematizada de los modelos ER y relacional mediante un Sistema Gestor de Bases de Datos (SGBD) se utiliza SQL.

- SQL son las siglas de Structured Query Language
- SQL es un lenguaje que nos **permite interactuar con los SGBD Relacionales** para especificar las operaciones que deseamos realizar sobre los datos y su estructura.
- Es un **lenguaje declarativo**, lo cual quiere decir que en él se especifica al SGBD qué queremos obtener y no la manera de cómo conseguirlo.
- Es un **lenguaje no procedimental** porque no necesitamos especificar el procedimiento para conseguir el objetivo, sino el objetivo en sí. **No es un lenguaje de programación** como puede ser Java.

Hoy en día **SQL es el lenguaje de consulta y manipulación de datos más extendido** y utilizado por todos los **desarrolladores y fabricantes de SGBD**. A lo largo de los años se han ido ampliando sus características conforme se sucedían avances en la tecnología informática.

SQL no es propiedad de ningún fabricante, sino que es una norma a seguir y los fabricantes de software no suelen implementar SQL puro en sus productos, sino que a menudo incorporan pequeñas variaciones para conseguir funcionalidades concretas en sus desarrollos. Esto hace que lo que debería ser un estándar no lo sea completamente en la realidad.

SQL significa lenguaje estructurado de consulta (Structured Query Language). Es un lenguaje estándar de cuarta generación que se utiliza para definir, gestionar y manipular la información contenida en una Base de Datos Relacional.

Es un lenguaje sencillo y potente que se emplea para la gestión de la base de datos a distintos niveles de utilización: usuarios, programadores y administradores de la base de datos.

1.1 ¿Cómo se usa SQL?

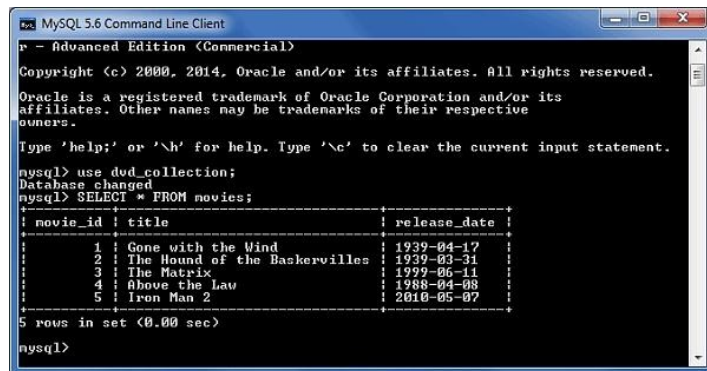
No hay que olvidar que estamos estudiando un lenguaje de interacción con bases de datos y que la herramienta para acceder y manipular esas bases de datos son los sistemas gestores de base de datos.

Según la normativa ANSI/ISO cuando se ejecuta SQL, existen los siguientes elementos a tener en cuenta en todo el entorno involucrado en la ejecución de instrucciones SQL:

- Un **agente SQL**. Entendido como cualquier elemento que cause la ejecución de instrucciones SQL que serán recibidas por un cliente SQL (es un servicio que ejecuta tareas administrativas programadas, denominadas trabajos en SQL, en el lado del servidor)
- Una **implementación SQL**. Se trata de un procesador software capaz de ejecutar las instrucciones pedidas por el agente SQL. Una implementación está compuesta por:
 - Un **cliente SQL**. Software conectado al agente que funciona como interfaz entre el agente SQL y el servidor SQL. Sirve para establecer conexiones entre sí mismo y el servidor SQL.
 - Un **servidor SQL** (puede haber varios). El software encargado de manejar los datos a los que la instrucción SQL lanzada por el agente hace referencia. Es el software que realmente realiza la instrucción, los datos los devuelve al cliente.

Los SGBDR permiten dos modos de acceso a las bases de datos:

- **Modo interactivo**, destinado principalmente a los usuarios finales, avanzados u ocasionales, en el que las diversas sentencias SQL se introducen a través de un cliente que está directamente conectado al servidor SQL; por lo que las instrucciones se traducen sin intermediarios utilizando un intérprete de órdenes y los resultados se muestran en el cliente.



```

MySQL 5.6 Command Line Client
> - Advanced Edition (Commercial)
Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use dvd_collection;
Database changed
mysql> SELECT * FROM movies;
+-----+-----+-----+
| movie_id | title                                | release_date |
+-----+-----+-----+
| 1 | Gone with the Wind                   | 1939-04-17   |
| 2 | The Hound of the Baskervilles        | 1939-03-31   |
| 3 | The Matrix                           | 1999-06-11   |
| 4 | Above the Law                        | 1988-04-08   |
| 5 | Iron Man 2                           | 2010-05-07   |
+-----+-----+-----+
5 rows in set (0.00 sec)

mysql>
  
```

- **Modo embebido**, destinado al uso por parte de los programadores. En este caso las sentencias SQL se introducen en lenguajes de programación, llamados lenguajes anfitrión (por ejemplo Java, lenguajes de la plataforma .NET de Microsoft, PHP, C++, etc.), de manera que el resultado es una mezcla de ambos. En este caso el lenguaje anfitrión aporta lo que le falta a SQL, es decir la programación. Al compilar el código se utiliza un precompilador de la propia base de datos para traducir el SQL y conectar la aplicación resultado con la base de datos a través de un software adaptador (**driver**) como JDBC u ODBC por ejemplo.



```

1 <html>
2 <body>
3 <h3>Conectando e insertando datos de la BD:</h3>
4
5 <?php
6
7 $nombre = $_POST['nombre'];
8 $apellidos = $_POST['apellidos'];
9 $telefono = $_POST['telefono'];
10 $direccion = $_POST['direccion'];
11
12 $conexion = mysql_connect("127.0.0.1", "root", "");
13 mysql_select_db("agenda", $conexion);
14
15 $sql = "INSERT INTO CONTACTOS (nombre, apellidos, telefono, direccion) VALUES ('$nombre', '$apellidos', '$telefono', '$direccion')";
16 mysql_query($sql, $conexion);
17
18 // Comprobamos que todo ha ido correctamente:
19 echo "Los datos han sido introducidos satisfactoriamente";
20
21 mysql_close($conexion);
22
23 </body>
24 </html>
  
```

1.2 ¿Qué podemos hacer con SQL?

Todos los principales SGBDR incorporan un motor SQL en el Servidor de Base Datos, así como herramientas de cliente que permiten enviar comandos SQL para que sean procesadas por el motor del servidor. De esta forma, todas las tareas de gestión de la Base de Datos (BD) pueden realizarse utilizando sentencias SQL.

- Consultar datos de la Base de Datos.
- Insertar, modificar y borrar datos.
- Crear, modificar y borrar objetos de la Base de Datos.
- Controlar el acceso a la información.
- Garantizar la consistencia de los datos.

1.3 Componentes del lenguaje SQL

El lenguaje SQL está compuesto por sentencias. Entre los trabajos que se pueden realizar en una base de datos podemos distinguir tres tipos: definición, manipulación y control de datos. Por ello se distinguen tres tipos de sentencias SQL:

Sentencias de manipulación de datos. (Lenguaje de Manipulación de Datos LMD). Se utilizan para:

- Recuperar información. (SELECT)
- Actualizar la información:
 - Añadir filas (INSERT)
 - Eliminar filas (DELETE)
 - Modificar filas (UPDATE)

Sentencias de definición de datos. (Lenguaje de Definición de Datos LDD). Se utilizan para:

- Eliminar objetos de base de datos (DROP)
- Crear objetos de base de datos (CREATE)
- Modificar objetos de base de datos (ALTER)

Sentencias de control de datos (Lenguaje de Control de Datos LCD). Se utilizan para:

- Determinar quien puede ver o modificar los datos (GRANT, REVOKE, DENY)

Con el tiempo han surgido nuevas necesidades en los SGBDR que han obligado a incorporar nuevas sentencias que no se pueden clasificar en los tres grupos clásicos anteriores, cómo son las sentencias para gestión de transacciones y bloqueos, las sentencias para replicación, etc.

Las sentencias SQL a su vez se construyen a partir de:

- **Cláusulas:** Que modifican el comportamiento de las sentencias. Constan de palabras reservadas y alguno de los siguientes elementos:
 - **Operadores lógicos y de comparación:** Sirven para ligar operandos y producir un resultado booleano (verdadero o falso).
 - **Funciones de agregación:** Para realizar operaciones sobre un grupo de filas de una tabla.
 - **Funciones:** Para realizar cálculos y operaciones de transformación sobre los datos.
 - **Expresiones:** Construidas a partir de la combinación de operadores, funciones, literales y nombres de columna.
- **Metadatos.** Obtenidos de la propia base de datos

La siguiente imagen muestra un resumen de la clasificación de las sentencias estándares del lenguaje SQL:



1.4 Notación en las instrucciones

En estos apuntes vamos a seguir una notación, para indicar la sintaxis en un comando usaremos:

- Las palabras reservadas las pondremos en mayúsculas (aunque da igual si se ponen en mayúsculas o minúsculas) y en negrita.
- Las palabras en cursiva y minúsculas se refieren al tipo de elemento que debe acompañar al comando.
- [], con los corchetes indicaremos que es algo opcional.
- |, con este símbolo indicamos opción, de las palabras separadas por este símbolo tenemos que escoger una.
- { }, indican alternativa obligatoria.
- ... Indica que se puede repetir el texto anterior en el comando continuamente.

1.5 Normas de escritura

En SQL no se distingue entre mayúsculas y minúsculas.

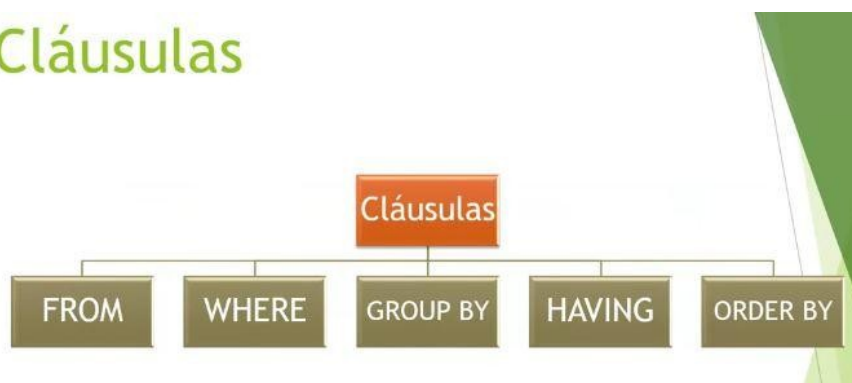
- Las instrucciones finalizan con el signo de punto y coma.
- Cualquier comando SQL (SELECT, INSERT,...) puede ser partido por espacios o saltos de línea antes de finalizar la instrucción
- Se pueden tabular líneas para facilitar la lectura si fuera necesario.
- Los comentarios en el código SQL comienzan por /* y terminan por */ (excepto en algunos SGBD)

Como se ha mostrado en los ejemplos anteriores, una sentencia se compone de algo más que el comando SQL. Las **instrucciones SQL** siguen la estructura mostrada en la siguiente figura:



Las **cláusulas** son condiciones que modifican nuestras consultas, y son utilizadas normalmente para definir los datos que se desean seleccionar o manipular.

Cláusulas



2 ELEMENTOS DEL LENGUAJE SQL

2.1 Tipos de datos

Los tipos de datos limitan los tipos de valores que se pueden almacenar en una base de datos.

Veamos una tabla con los tipos de datos SQL estándar, los tipos de datos en Oracle, y los tipos de datos en MySQL. Podemos encontrar más tipos de datos si consultamos la página del manual oficial de [mysql](https://dev.mysql.com/doc/refman/8.0/en/).

Descripción	Tamaño/ Formato	Tipo estándar SQL	Oracle SQL	MySQL	SQL Server
Texto de anchura fija	Fijo	CHARACTER(n) CHAR(n)	CHAR(n)	CHAR(n)	CHAR(n)
Texto de anchura variable	Variable	CHARACTER VARYING(n) CHAR VARYING(n)	VARCHAR2(n)	VARCHAR(n)	VARCHAR(n)
Texto de anchura fija para caracteres nacionales	Fijo	NATIONAL CHARACTER(n)	NCHAR(n)	NCHAR(n)	NCHAR(n)
Texto de anchura variable para caracteres nacionales	Variable	NATIONAL CHARACTER VARYING(n)	NVARCHAR2(n)	NVARCHAR(n)	NVARCHAR(n)
Enteros	1 byte			TINYINT	
	2 bytes	SMALLINT	SMALLINT	SMALLINT	
	3 bytes			MEDIUMINT	
	4 bytes	INT	INT	INT	INT
	4 bytes	INTEGER	INTEGER	INTEGER	INT
	8 bytes			BIGINT	
Decimal de coma variable (Real aproximado)	4 bytes	FLOAT	FLOAT NUMBER	FLOAT	FLOAT
	8 bytes	DOUBLE		DOUBLE	
		DOUBLE PRECISION			
		REAL	REAL	REAL	
Decimal de coma fija (Real Exacto)	l+2 bytes si d>0, l+1 bytes si d=0	DECIMAL(longitud, decimales)		DECIMAL(l,d)	DECIMAL(l,d)
	l+2 bytes si d>0, l+1 bytes si d=0	NUMERIC(longitud, decimales)	NUMBER(l,d)	NUMERIC(l,d)	NUMERIC(l,d)
Fechas	'aaaa-mm-dd'	DATE	DATE	DATE	
Hora	'hh:mm:ss'	TIME	TIME	TIME	
Fecha y Hora	'aaaa-mm-dd hh:mm:ss'	TIMESTAMP	TIMESTAMP	TIMESTAMP	TIMESTAMP
Fecha y Hora	'aaaa-mm-dd			DATETIME	DATETIME

	hh:mm:ss'				
Texto de gran longitud (campos memo)	Variable	CLOB	CLOB	TEXT, MEDIUM, LONG TEXT, LONG TEXT	TEXT y NTEXT
Binario de gran longitud	Variable	BLOB	BLOB	BLOB	IMAGE
Enumeraciones	Lista de valores			ENUM(valor1, valor2, ...)	
Conjuntos	Conjuntos de valores			SET(valor1, valor2, ...)	

Observamos que no todos los tipos de datos están en todos los gestores. Existe diferencias en cuanto a los tipos de datos que pueden almacenar. Además, estos tipos de datos pueden cambiar dependiendo de la versión del SGBD, por eso es recomendable consultar la documentación oficial.

2.2 Variables

Las variables son elementos del lenguaje con valores asignados. En SQL se pueden utilizar variables locales y variables de usuario.

Una variable local es una variable definida por el usuario en una instrucción DECLARE; se le asigna un valor inicial en una instrucción SET o SELECT y, después se utiliza en una instrucción, programa por lotes o procedimiento en el que se declaró. Las variables de usuario se escriben como @nombre_var. Se les puede asignar valor con SET.

Las variables locales sólo duran el tiempo correspondiente a un proceso por lotes, mientras que las variables de usuario tienen a misma duración que una sesión.

2.3 Identificadores

Son nombres que sirven para identificar objetos de la base de datos: usuarios, tablas, columnas, etc.

Como regla general se intenta poner nombres cortos y utilizar nombres significativos cuando sea posible.

En los identificadores se intenta no utilizar caracteres especiales, por ejemplo “, tildes, ñ, exclamaciones, interrogantes, espacios en blanco.

2.4 Constantes

Podemos utilizar las siguientes constantes:

- **Constantes numéricas**
 - Pueden llevar un punto decimal, y precedidas por + o - 2424.34
 - Con formato en coma flotante 24.356E-8
- **Constantes de cadena**
 - Encerradas en comillas simple ‘Hola Mundo’
- **Constantes de fecha**
- **Constante Binario** 0x05 0xFF 0x5aef1b
- **Constante Nula** NULL

Ejemplo	Fecha	Hora
'801215'	5 Diciembre 1980	00:00:00.000
'15/12/1980'	idem	idem
'15-12-80 8:30'	idem	08:30:00.000
'8:30:2'	1 Enero 1900	08:30:02.000
'15.12.1980 8:30pm'	15 Diciembre 1980	20:30:00.000

2.5 Comentarios

Los comentarios son cadenas que no se ejecutan y que se pueden colocar en las instrucciones para realizar anotaciones o deshabilitar una parte de las mismas durante las pruebas. Existen dos tipos:

- Comentarios de línea - -
- Comentarios de bloque /* */

2.6 Operadores y expresiones

Las sentencias SQL pueden incluir expresiones constituidas por nombres de columnas, constantes, funciones y operadores. Las expresiones dan como único resultado un único valor.

Ejemplo: **SELECT** apellido, salario, salario + 10000 **FROM** Empleados

Los operadores son símbolos para realizar cálculos matemáticos, concatenar cadenas y comparación de columnas, constantes y variables.

Tipos de operadores

Operadores aritméticos

Se emplean para realizar cálculos (+, -, *, /, %). Devuelven un valor numérico como resultado de realizar los cálculos indicados.

Operadores de concatenación de cadenas

El operador de concatenación de cadenas (+) concatena valores de cadena, en Oracle es el símbolo ||. MySQL no soporta este tipo de operador, así que habrá que utilizar la función `CONCAT`

Operadores de comparación

Comparan dos expresiones. Dan como resultado un valor del tipo verdadero/falso (true/false)

3 LENGUAJE DDL

3.1 Introducción

El DDL (Data Definition Language) es la parte del lenguaje que realiza la función de definición de datos. Fundamentalmente se encarga de la creación de esquemas o bases de datos, tablas y vistas.

Cada usuario de la base de datos posee un esquema. El esquema suele tener el mismo nombre que el usuario y sirve para almacenar los objetos de esquema, es decir los objetos que posee el usuario.

Estos objetos pueden ser: tablas, vistas, índices y otros objetos relacionados con la definición de la base de datos. Los objetos son creados y manipulados por los usuarios. En principio sólo los administradores y los usuarios propietarios pueden acceder a cada objeto, salvo que se modifiquen los privilegios del objeto para permitir el acceso a otros usuarios.

3.2 Creación de bases de datos

El comando SQL de creación de una base de datos es **CREATE DATABASE**. Este comando crea una base de datos con el nombre que se indique. Ejemplo.

CREATE DATABASE prueba;

3.2.1 Creación de bases de datos en MySQL

En MySQL, para crear una base de datos se usa la siguiente sintaxis:

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] nombre_bd
    [especificación_create [, especificación_create] ... ]
```

donde especificación_create es:

```
[DEFAULT] CHARACTER SET juego_caracteres
[DEFAULT] COLLATE nombre_colación
```

Vemos que podemos elegir entre `DATABASE` o `SCHEMA`, en cualquier caso, el efecto es el mismo; MySQL usa algunos sinónimos para hacer más compatible su SGBD con otros.

El parámetro opcional (especificación_create) permite elegir el juego de caracteres que va a usar la base de datos (utf8, latin1, mlatin2, etc), y la colación, que especifica cómo va a tratar el alfabeto del juego de caracteres, es decir, cómo se ordena (por ejemplo, la ñ después de la n, y no conforme a la tabla de códigos ascii), y cómo se comparan los caracteres (por ejemplo, si la Á es igual que la á)

Para poder utilizar una base de datos, primero hay que establecer que las operaciones que se realicen a continuación se realicen sobre ella. Para eso se utiliza el comando `USE nombre_db`.

Otro comando muy útil es `SHOW DATABASES`, que obtiene un listado de las bases de datos activas en el servidor.

Si lo ejecutamos observamos que hay dos bases de datos que siempre están presentes. 'information_schema' y 'mysql' que almacenan metadatos (datos sobre los datos) e información de usuarios y permisos.

3.2.2 Creación de bases de datos en Oracle

La creación de bases de datos es una tarea administrativa, la comentaremos más adelante. Crear una base de datos implica indicar los archivos y ubicaciones que se utilizarán por la misma, además de otras indicaciones técnicas y administrativas. Es un proceso más elaborado que en MySQL. En MySQL, una sola instancia controla todas las bases de datos, pero en Oracle cada base de datos está asociada a una instancia. Una instancia es el conjunto de procesos de Oracle que están en ejecución en el sistema operativo y cuya misión es controlar todo lo referente a la gestión de la base de datos.

Sólo es posible crear una base de datos si se tienen privilegios DBA (DataBase Administrator) o SYSDBA.

El comando SQL de creación de una base de datos es `CREATE DATABASE`. Este comando crea una base de datos con el nombre que se indique. Ejemplo.

```
CREATE DATABASE nombre_bd;
```

Pero normalmente se indican más parámetros. Ejemplo:

```
CREATE DATABASE nombre_bd
    MAXLOGFILES 25
    MAXINSTANCES 10
    CHARACTER SET WIN1214
    NATIONAL CHARACTER SET UTF8
    DATAFILE prueba1.dbf AUTOEXTEND ON MAXSIZE 500MB;
```

Veamos algunos de ellos:

- `MAXINSTANCES` indica el número máximo de instancias que pueden estar usando a la vez la base de datos objeto de la creación.
- `MAXLOGFILES` es el número máximo de grupos de ficheros de log que pueden crearse. Un fichero log registra todas las operaciones que realizan los usuarios.
- `DATAFILE` es el fichero para el tablespace SYSTEM.

- `CHARACTER SET` especifica el juego de caracteres de la base de datos

3.3 Modificación de una base de datos

Utilizamos el comando SQL **ALTER DATABASE** *nombre_bd*

En MySQL sólo se permite cambiar el juego de caracteres y su colación.

En Oracle, se puede cambiar cualquiera de sus múltiples parámetros.

3.4 Borrado de bases de datos

El comando SQL para borrar una base de datos es **DROP DATABASE** *nombre_bd*

En MySQL el comando se acompaña por el nombre de la base de datos a eliminar.

En Oracle hay que conectarse primero a la instancia y después usar el comando. No obstante, lo haremos de una manera más sencilla.

3.5 Creación de tablas

Antes de crear una tabla tenemos que tener en cuenta las siguientes consideraciones previas:

- El nombre de la tabla.
- El nombre de cada columna.
- El tipo de dato almacenado en cada columna.
- El tamaño de cada columna.
- Otra información.

La sentencia SQL estándar que permite crear tablas es **CREATE TABLE**.

Formato básico para la creación de tablas

```
CREATE TABLE nombre_de_tabla  
(definición_de_columna);
```

Donde

- ***nombre_de_tabla***: debe cumplir los siguientes requisitos:
 - permite un conjunto de hasta 30 caracteres, comenzando por uno alfabético y con posibilidad de contener alfanuméricos y subrayados, así como mayúsculas y minúsculas indistintamente.
 - No puede haber dos tablas con el mismo nombre para el mismo esquema.
 - No puede coincidir con el nombre de una palabra reservada de SQL.
 - En el caso de que el nombre tenga espacios en blanco o caracteres nacionales, entonces se pueden entrecomillar, en Oracle con comillas dobles, en SQLServer con comillas simples, en SQLServer con comillas simples, en MySQL debe ir entre ``
- ***definición_de_columna*** consiste en indicar para cada columna de la tabla:
 - **Nombre de columna**. Por ejemplo apellido
 - **Tipo de dato** que se va a almacenar en esa columna. En el ejemplo:
apellido VARCHAR2 (en Oracle) apellido VARCHAR (en MYSQL)
 - **Tamaño previsto** para esa columna. En el ejemplo:

apellido VARCHAR2(8) (en Oracle) apellido VARCHAR(8) (en MySQL)

Los tipos de datos serán los de la tabla del principio de la unidad. Además estos tipos de datos dependerán del SGBD utilizado.

Existirán tantas definiciones de columna como datos diferentes se vayan a almacenar en la tabla que estamos creando.

Ejemplo 1. Crear una tabla de artículos con las siguientes descripciones de columna:

- Referencia_artículo -> alfabético de 6 caracteres.
- Descripción_artículo -> alfabético de 30 caracteres.
- Precio_unidad -> real.
- IVA -> numérico de 2 posiciones.
- Existencias_actuales -> entero.
- Stock_mínimo -> entero

```
CREATE TABLE articulos
(referencia_articulo VARCHAR(6),
descripcion_articulo VARCHAR(30),
precio_unidad NUMBER(4,2),
iva NUMBER(2),
existencias_actuales NUMBER(4),
stock_minimo NUMBER(2)
)
```

Sólo se podrá crear la tabla si el usuario posee los permisos necesarios para ello. Si la tabla pertenece a otro bases de datos (suponiendo que el usuario tenga permiso para grabar tablas en ese otra bases de datos), se antepone al nombre de la tabla, el nombre del esquema:

```
CREATE TABLE OtraBasesDeDatos.proveedores (nombre VARCHAR(25));
```

NOTA: OBJETOS DE LA BASE DE DATOS.

Según los estándares actuales, una base de datos es un conjunto de objetos pensados para gestionar datos. Estos objetos están contenidos en esquemas. Los esquemas suelen estar asociados al perfil de un usuario en particular. En el estándar SQL existe el concepto de catálogo que sirve para almacenar esquemas. Así el nombre completo de un objeto vendría 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. En Oracle, cuando se crea un usuario, se crea un esquema cuyo nombre es el del usuario.

3.5.1 Creación de tablas en MySQL

En MySQL se pueden indicar una serie de opciones al crear una tabla, veamos algunas de ellas. La sintaxis sería

```
CREATE TABLE nombre_de_tabla
```

```
(definición_de_columna)
[opciones tabla];
```

donde opciones tabla puede ser:

```
ENGINE = nombre_motor
| AUTO_INCREMENT = valor
| [DEFAULT] CHARACTER SET juego_caracteres [COLLATE colación]
| COMMENT = 'string'
| MAX_ROWS = valor
| MIN_ROWS = valor
```

El almacenamiento físico de una tabla en MySQL está controlado por un software especial llamado Motor de almacenamiento. Mediante la opción `ENGINE = nombre_motor` se indica el motor de almacenamiento. Puede ser, entre otros:

- *innodb*, que son las tablas transaccionales con bloqueo de registro y claves foráneas.
- *myIsam*, es el que usa por defecto MySQL, y que genera tablas operadas a gran velocidad, pero sin control de integridad referencial. Este desaparecerá en la versión 8 de mysql
- *memory*, que genera tablas que están almacenadas en memoria en lugar de un archivo físico.

La opción `AUTO_INCREMENT` permite indicar el valor inicial para campos de tipo `AUTO_INCREMENT`. Un campo definido de esta forma es autoincrementado después de cada inserción. Un campo definido como `auto_increment` debe ser numérico. Es muy útil cuando se emplea en campos que almacenan códigos. En MySQL para definir un campo `AUTO_INCREMENT` se especifica la palabra clave justo a continuación del tipo de dato. En Oracle, para simular este comportamiento se utilizan las secuencias (**SEQUENCE**)

`[DEFAULT] CHARACTER SET` especifica el conjunto de caracteres para la tabla y `COLLATE` define la colación por defecto de la tabla.

`COMMENT` es un comentario para la tabla, hasta 60 caracteres. También es posible crear comentarios para cada uno de los campos de la tabla. Con este comando se puede documentar el diccionario de datos.

`MAX_ROWS` es el máximo número de registros que se quiere almacenar en la tabla. No es un límite, sino sólo un indicador que la tabla debe ser capaz de almacenar al menos estos registros. `MIN_ROWS` es el mínimo número de registros que se planea almacenar en la tabla.

Por último, al igual que en la creación de bases de datos, MySQL dispone de la cláusula `IF NOT EXISTS`, para que solamente se cree la tabla si no está creada previamente.

Veamos un ejemplo:

```
CREATE TABLE IF NOT EXISTS pedidos (
    codigo INT AUTO_INCREMENT PRIMARY KEY,
    fecha DATETIME,
    estado ENUM ('pendiente', 'entregado', 'rechazado')
)
COMMENT = 'tabla de pedidos a proveedores'
AUTO_INCREMENT = 1000
MAX_ROWS = 100000
ENGINE = innodb;
```

3.5.2 Creación de tablas en Oracle

En Oracle, la mayoría de las opciones tiene que ver con su almacenamiento físico. Por ejemplo, las tablas deben ser almacenada en un contenedor llamado *tablespace* (espacio de tablas). Por defecto, si no se indican opciones de almacenamiento, la tabla se ubica en el tablespace del usuario, pero si se quiere ubicar en otro tablespace, se puede incluir la opción *tablespace nombre* para designar otro tablespace. Además, se puede definir cómo se reserva el espacio en disco para controlar el crecimiento desmedido de una tabla mediante la cláusula *storage*.

Ejemplo:

```
CREATE TABLE pedidos (
    codigo INT AUTO_INCREMENT PRIMARY KEY,
    fecha DATETIME,
    CONSTRAINT ck_estado
        CHECK (estado IN ('pendiente', 'entregado', 'rechazado'))
)
TABLESPACE administracion
STORAGE (initial 100k next 100k minextents 1
        maxextents unlimited pctincrease 0);
```

3.6 Implementación de restricciones de tabla y columna

Restricciones son condiciones que imponemos a la hora de crear una tabla para que los datos se ajusten a una serie de características predefinidas que mantengan su integridad.

Se refieren a los siguientes conceptos:

Valor por defecto

DEFAULT. Proporciona un valor por defecto, cuando la columna a la que acompaña no recibe ningún dato cuando se está insertando. Los valores por defecto pueden ser: constantes, funciones SQL o las variables `USER` o `GETDATE`. Se puede usar durante la creación o la modificación de la tabla.

Ejemplo. La fecha de alta de los empleados será la del sistema o fecha del día en que se está realizando la entrada de datos en la base de datos, si no se le indica otra.

```
CREATE TABLE empleados
(.....,
    fecha_alta DATETIME DEFAULT GETDATE(),
    .....
);
```

Las siguientes restricciones son conocidas en SQL por su nombre en inglés, es decir **CONSTRAINT**:

Valor no nulo

NOT NULL. Exige la existencia de dato en la columna que lleva la restricción.

Ejemplo. El número de empleado nunca irá sin información.

```
CREATE TABLE empleados
(emp_no NUMBER(4) NOT NULL
.....
.....)
```

```
);
CREATE TABLE empleados
(emp_no NUMBER(4) CONSTRAINT emp_no_nn NOT NULL,
.....,
,
);
```

Restricciones de validación

CHECK. Comprueba si se cumple una determinada condición. No puede incluir una subconsulta, ni las variables GETDATE o USER. Una misma columna puede tener varios checks en su definición (se pondrán varios CONSTRAINT seguidos sin comas). Esta restricción puede hacer referencia a una o más columnas, pero no a valores de otras filas.

Ejemplos

1. La columna APELLIDO del empleado siempre deberá ir con dato. Podría escribirse una de las restricciones siguientes:

```
CREATE TABLE empleados
(.....,
apellido VARCHAR(8) NOT NULL,
.....,
);

CREATE TABLE empleados
(.....,
apellido VARCHAR2(8) CHECK (apellido IS NOT NULL),
.....,
);
```

2. La columna IMPORTE de la tabla ingresos tiene que tener un valor mínimo y un máximo. Podría escribirse una de las restricciones siguientes:

```
CREATE TABLE ingresos
(.....,
importe NUMBER(11,2),
CONSTRAINT ck_importe_min CHECK (importe>0),
CONSTRAINT ck_importe_max CHECK (importe<8000),
.....,
);
```

Para poder hacer referencia a otro campo en la tabla hay que construir la restricción de forma independiente a la columna, es decir al final de la tabla.

```
CREATE TABLE ingresos
(.....,
importe_max NUMBER(11,2)
importe NUMBER(11,2),
```

```

        CONSTRAINT ck_importe_maximo CHECK (importe<importe_max),
        .....
    );

```

Restricción de valor único

UNIQUE. Evita valores repetidos en la misma columna. Admite valores NULL.

Ejemplos.

```

CREATE TABLE empleados
(emp_no NUMERIC(4) NOT NULL UNIQUE
.....
);

CREATE TABLE empleados
(dni VARCHAR(9) CONSTRAINT un_dni UNIQUE
.....
);

CREATE TABLE alquileres
(dni VARCHAR(9),
cod_pelicula NUMBER(5),
CONSTRAINT uk_alquileres UNIQUE (dni, cod_pelicula)
.....
);

```

La coma tras la definición del campo cod_pelicula hace que la restricción sea independiente de este campo. Esto obliga a que, tras **UNIQUE** se indique la lista de campos. Incluso para un solo campo se puede colocar la restricción al final de la lista en lugar de definirlo a continuación del nombre y tipo de la columna.

Nota: Las claves candidatas deben llevar siempre la restricción de **UNIQUE** y **NOT NULL**.

Claves primarias

La clave primaria de una tabla la forman las columnas que identifican a cada registro de la misma. La clave primaria hace que los campos que la forman sean **NOT NULL** y que los valores de los campos sean de tipo **UNIQUE**.

Cuando se crea una clave primaria en una tabla se crea un índice que facilita el acceso a la tabla.

Un **índice** de una base de datos es una estructura de datos que puede mejorar la velocidad de las operaciones sobre la base de datos, permitiendo un acceso más rápido a los registros de una tabla. Se podría decir que, en una base de datos relacional un índice es una copia de parte de una tabla.

PRIMARY KEY. Indica una o varias columnas como dato o datos que identifican unívocamente cada fila de la tabla. Sólo existe una por tabla y en ninguna fila puede tener valor NULL. En nuestras tablas de *empleados* y *departamentos* serían **emp_no** y **dep_no**, respectivamente.

Ejemplos:

1.- Si la clave está formada por un único campo:

```

CREATE TABLE empleados
(emp_no NUMERIC(4) PRIMARY KEY
.....
);

```

2.- Podemos poner un nombre a la restricción:

```
CREATE TABLE empleados
(emp_no NUMERIC(4) CONSTRAINT pk_empleados PRIMARY KEY
.....,
);
```

3.- Si la clave está formada por varios campos:

```
CREATE TABLE alquileres
(dni VARCHAR(9),
cod_pelicula NUMBER(5),
CONSTRAINT pk_alquileres PRIMARY KEY (dni, cod_pelicula)
.....,
);
```

Claves Ajenas

Una clave ajena, secundaria o foránea, es una o más columnas de una tabla que están relacionadas con la clave principal de otra tabla.

FOREIGN KEY. Indica que una determinada columna de una tabla va a servir para referenciar a otra tabla en la que la misma columna está definida como **PRIMARY KEY** o **UNIQUE**. El valor de la clave foránea o ajena deberá coincidir con uno de los valores de esta clave referenciada o ser NULL. No existe límite en el número de claves foráneas o ajenas que pueda tener una tabla. Como caso particular, una clave foránea o ajena puede referenciar a la misma tabla en la que está. Para poder crear una tabla con clave foránea o ajena deberá estar previamente creada la tabla maestra en la que la misma columna es clave primaria.

Ejemplo: En nuestras tablas de *empleados* y *departamentos*, **dep_no** sería clave foránea o ajena en la tabla de *empleados* porque con ella podemos acceder a la tabla de *departamentos* y ésta tabla deberá crearse antes que la de *empleados*.

```
CREATE TABLE departamentos
(dep_no NUMERIC(4) CONSTRAINT pk_departamentos PRIMARY KEY
.....,
);

CREATE TABLE empleados
(emp_no NUMERIC(4) CONSTRAINT pk_empleados PRIMARY KEY,
dep_no NUMERIC(4) CONSTRAINT fk_departamentos REFERENCES departamentos(dep_no)
.....,
);
```

Significa que el campo **dep_no** se relaciona con la columna **dep_no** de la tabla *departamentos*.

Esto forma una relación entre dichas tablas, que además obliga al cumplimiento de la integridad referencial. Esta integridad obliga a que cualquier **dep_no** incluido en la tabla *empleados* tenga que estar obligatoriamente en la tabla de *departamentos*. De no ser así el registro no será insertado en la tabla (ocurrirá un error)

Otra forma de crear claves ajenas, útil para claves formadas por más de un campo, es:

```
CREATE TABLE existencias
(tipo CHAR(9),
modelo CHAR(3),
n_almacen INTEGER,
cantidad INTEGER,
```



```

CONSTRAINT    fk_existencias_t_m    FOREIGN KEY (tipo, modelo) REFERENCES
piezas (tipo, modelo),

CONSTRAINT    fk_existencias_n_alm    FOREIGN KEY (n_almacen) REFERENCES
almacenes (n_almacen),

CONSTRAINT pk_existencias PRIMARY KEY (tipo, modelo, n_almacen)
.....,
);

```

Si la definición de clave secundaria se pone al final, hace falta colocar el texto **FOREIGN KEY** para indicar en qué campos se coloca la restricción de clave ajena.

La integridad referencial es una herramienta imprescindible de la bases de datos relacionales. Pero puede provocar varios problemas. Por ejemplo, si borramos un registro en la tabla principal que está relacionad con uno o varios de la secundaria ocurrirá un error, ya que de permitirnos borrar el registro ocurrirá fallo de integridad (habrá claves secundarias refiriéndose a una clave principal que ya no existe)

Para mantener la integridad de los datos, al borrar (**DELETE**) o modificar (**UPDATE**) una fila de la tabla referenciada, existen las siguientes opciones:

- **CASCADE**. El borrado o modificación de una fila de la tabla referenciada lleva consigo el borrado o modificación en cascada de las filas de la tabla que contiene la clave ajena. Es la más utilizada.
- **SET NULL**. El borrado o modificación de una fila de la tabla referenciada lleva consigo poner a **NULL** los valores de las claves ajenas en las filas de la tabla que referencia.
- **SET DEFAULT**. El borrado o modificación de una fila de la tabla referenciada lleva consigo poner un valor por defecto en las claves ajenas de la tabla que referencia.
- **NO ACTION**. El borrado o modificación de una fila de la tabla referenciada sólo se lleva a cabo si no existe ninguna fila con el mismo valor en la clave ajena en la tabla que referencia. En algunos gestores se conoce como **RESTRICT**. Es la opción por defecto.

La sintaxis completa para añadir claves ajenas es:

```

CREATE TABLE tabla
(lista_de_campos,
CONSTRAINT nombreRestricción FOREIGN KEY (lista__campos) REFERENCES
tabla (clavePrincipalRelacionada)
[ON DELETE | ON UPDATE]
[SET NULL | CASCADE | DEFAULT | NO ACTION]
);

```

Si la clave ajena es de un solo campo existe esta alternativa

```

CREATE TABLE tabla (lista_de_campos,
CONSTRAINT nombreRestricción REFERENCES tabla (clavePrincipalRelacionada)
[ON DELETE | ON UPDATE]
[SET NULL | CASCADE | DEFAULT | NO ACTION]
);

```

3.7 Constraint o Restricciones

Una restricción o constraint es una condición de obligado cumplimiento para una o más columnas de la tabla.

Las **CONSTRAINTS** se van a almacenar con un nombre. Si no se lo damos nosotros, el sistema las nombrará con un formato del sistema que es poco representativo.

Criterio para dar nombres significativos a las CONSTRAINTS

- **nn_nombre_tabla_nombre_columna** para NOT NULL
- **cki_nombre_tabla_nombre_columna** para posibilitar más de un CHECK a la misma columna. Se puede utilizar nombres más significativos.

Ejemplos:

ck_lugar_nombre_columna para comprobar que el dato de esa columna es una de las localidades permitidas.

- **uq_nombre_tabla_nombre_columna** para UNIQUE
- **pk_nombre_tabla_nombre_columna** para PRIMARY KEY
- **fk_nombre_tabla_nombre_columna** para FOREIGN KEY

Las restricciones que acabamos de relacionar se pueden establecer a nivel de tabla o a nivel de columna, con alguna consideración lógica:

- NOT NULL tiene más sentido a nivel de columna.
- Cuando existan claves compuestas, se definirán a nivel de tabla.

Formato para la creación de tablas con CONSTRAINTS definidas a nivel de tabla

```
CREATE TABLE nombre_de_tabla
(definición_de_columna
CONSTRAINT nombre_constraint
CHECK(condición)
UNIQUE ( columna(s))
PRIMARY KEY ( columna(s))
FOREIGN KEY ( columna(s)) *)
```

(*) Si una columna se define como **FOREIGN KEY**, el formato se amplía con las siguientes indicaciones:

```
FOREIGN KEY[ (columna(s)) ] REFERENCES tabla_referenciada
[ ( columna(s)) ]
ON DELETE CASCADE          ON UPDATE CASCADE
SET NULL                   SET NULL
SET DEFAULT                 SET DEFAULT
NO ACTION                   NO ACTION
```

- La *columna* o *columnas* que siguen a la cláusula **FOREIGN KEY** es aquella o aquellas que están formando la clave ajena o extranjera. Si hay más de una se separan por comas.
- La *tabla referenciada* es el nombre de la tabla a la que se va a acceder con la clave ajena o extranjera y donde la misma es clave primaria.
- Si la columna o columnas que forman la clave primaria en la tabla referenciada no tiene el mismo nombre que en la clave ajena, debe indicarse su nombre detrás del de la tabla referenciada y dentro de paréntesis. Si son más de una columna se separan por comas. Si los nombres de columnas coinciden en la clave ajena y en la primaria, no es necesario realizar esta indicación.
- Para mantener la integridad de los datos, al borrar (DELETE) o modificar (UPDATE) una fila de la tabla referenciada, existen las siguientes opciones:
 - **CASCADE**.

- **SET NULL.**
- **SET DEFAULT.**
- **NO ACTION.**

Formato para la creación de tablas con CONSTRAINTS definidas a nivel de columna

```
CREATE TABLE nombre_de_tabla
(definición_de_columna);
CONSTRAINT nombre_constraint
NOT NULL
CHECK(condición)
UNIQUE
PRIMARY KEY
FOREIGN KEY *
```

(*) Si una columna se define como **FOREIGN KEY**, el formato se amplía con las siguientes indicaciones:

```
FOREIGN KEY[(columna(s))] REFERENCES tabla_referenciada
[( columna(s))]
ON DELETE CASCADE           ON UPDATE CASCADE
SET NULL                    SET NULL
SET DEFAULT                  SET DEFAULT
NO ACTION                    NO ACTION
```

Nota: No se pueden crear tablas con el mismo nombre que otras ya existentes en la misma base de datos.

Ejemplos

1. Crear la tabla de artículos con el stock_mínimo a 0 por defecto.

. Sin nombre de CONSTRAINT:

```
CREATE TABLE articulos
( referencia_articulo VARCHAR(6),
  descripcion_articulo VARCHAR(30),
  precio_unidad NUMBER(4,2),
  iva NUMBER(2),
  existencias_actuales NUMBER(2),
  stock_minimo NUMBER(2) DEFAULT 0
);
```

2. Crear la tabla de compradores con la columna cif_comprador como primary key y utilizando CONSTRAINT a nivel de tabla.

. Sin nombre de CONSTRAINT:

```
CREATE TABLE compradores
( cif_comprador VARCHAR(11) NOT NULL,
  nombre_social VARCHAR(30),
  domicilio_social VARCHAR(30),
  localidad VARCHAR2(30),
```

```

c_postal VARCHAR2(5),
telefono CHAR2(9),
PRIMARY KEY (cif_comprador)
);

```

• Con nombre de CONSTRAINT:

```

CREATE TABLE compradores
(
  CIF_COMPRADOR VARCHAR2(11) NOT NULL,
  NOMBRE_SOCIAL VARCHAR2(30),
  DOMICILIO_SOCIAL VARCHAR2(30),
  LOCALIDAD VARCHAR2(30),
  C_POSTAL VARCHAR2(5),
  TELEFONO CHAR2(9),
  CONSTRAINT PK_COMPRADORES_CIF PRIMARY KEY (CIF_COMPRADOR)
)

```

3. Crear la tabla de líneas de facturas con las columnas factura_no y referencia_articulo como primary key y la columna referencia_articulo como foreign key, utilizando CONSTRAINT a nivel de tabla. La CONSTRAINT para una clave compuesta sólo puede definirse a nivel de tabla. CONSTRAINT para la foreign key a nivel de tabla.

Sin nombre de CONSTRAINT:

```

CREATE TABLE líneas_facturas
(
  factura_no NUMBER(5) NOT NULL,
  referencia_articulo VARCHAR2(6) NOT NULL,
  unidades NUMBER(2),
  PRIMARY KEY(factura_no, referencia_articulo),
  FOREIGN KEY(referencia_articulo)
    REFERENCES articulos2(referencia_articulo)
    ON DELETE CASCADE
);

```

Con nombre de CONSTRAINT:

```

CREATE TABLE líneas_facturas2
(
  factura_no NUMBER(5) NOT NULL,
  referencia_articulo VARCHAR2(6) NOT NULL,
  unidades NUMBER(2),
  CONSTRAINT pk_lineas_factura
    PRIMARY KEY(factura_no, referencia_articulo),
  CONSTRAINT fk_lineas_referencia
    FOREIGN KEY(referencia_articulo)
      REFERENCES articulos2(referencia_articulo)
      ON DELETE CASCADE
);

```

3.8 Consultar las tablas de una base de datos

En MySQL para consultar las tablas disponibles está en comando **SHOW TABLES**.

Todas las bases de datos poseen un diccionario de datos que contiene la lista de tablas. En cada sistema de base de datos se accede a esos datos de una forma. En el caso de Oracle, hay una tabla en el diccionario de datos que se llama **USER_TABLES** y que contiene una lista de las tablas de usuario actual. Así, para sacar una lista de tablas, se usa:

```
SELECT * FROM USER_TABLES;
```

Hay diversas columnas que muestran datos sobre cada tabla, entre ellas la columna **TABLES_NAME** muestra el nombre de cada tabla del usuario; el resto muestra información sobre el almacenamiento de la tabla. La tabla **ALL_TABLES** mostrará una lista de todas las tablas accesibles por el usuario. Finalmente **DBA_TABLES** es una tabla que contiene todas las tablas del sistema; esto es accesible sólo por el usuario administrador DBA.

Orden Describe

El comando describe, permite obtener la estructura de una tabla. No es parte del estándar SQL, pero la consideran la mayoría de SGBD.

```
DESCRIBE nombre_tabla;
```

3.9 Modificación de tablas

Una vez que hemos creado una tabla, a menudo se presenta la necesidad de tener que modificarla. La sentencia SQL que realiza esta función es **ALTER TABLE**.

Formato general para la modificación de tablas

```
ALTER TABLE nombre_de_tabla especificación_de_modificación;
```

Donde:

- **nombre_de_tabla** que se desea modificar.
- **especificación_de_modificación**.

Las modificaciones que pueden realizarse sobre una tabla son las siguientes:

Añadir una nueva columna

La especificación de la modificación es parecida a la de la sentencia **CREATE** pero varía según el producto SQL del que se trate.

Formato para añadir una nueva columna

```
ALTER TABLE tabla ADD definición_columna1 [NOT NULL WITH DEFAULT][,  
definición_columna2 [NOT NULL WITH DEFAULT...]]
```

La adición de una nueva columna con **NOT NULL** está permitida siempre que vaya con un valor por defecto, correspondiente al tipo de dato de la columna. Con este tipo de formato sólo podemos añadir una columna en la misma sentencia **ALTER TABLE**. En MySQL se pueden utilizar las cláusulas **AFTER** (después de una columna) y **FIRST** (la primera columna). Oracle no las admite.

Ejemplo: Añadir el CIF_proveedor a la tabla de artículos.

```
ALTER TABLE articulos  
ADD cif_proveedor VARCHAR(11);
```

Modificar una columna

La especificación de la modificación del campo de una tabla se realiza con `MODIFY`

```
ALTER TABLE tabla MODIFY definición_columnal;
```

Ejemplo: Modificar el campo cif_proveedor de ARTICULOS a NUMBER(5)

```
ALTER TABLE articulos  
MODIFY cif_proveedor NUMBER(5);
```

Renombrar una columna

Para renombrar una columna utilizamos `RENAME` o `CHANGE` (en MySQL). En Oracle solo `RENAME`

```
ALTER TABLE tabla RENAME COLUMN nombre_atiguo TO nombre_nuevo;
```

Ejemplo: Cambiar el nombre de la columna descripcion_articulo por artículo:

```
ALTER TABLE articulos RENAME COLUMN descripcion_articulo TO descripcion;
```

Añadir restricciones de tabla

Las restricciones se utilizan con el mismo formato que vimos en `CREATE TABLE`

Formato para añadir restricciones de tabla

```
ALTER TABLE tabla ADD restricción_de_tabla
```

Ejemplo: Añadir en la tabla artículos la restricción de comprobar que la columna precio_unidad contenga un valor NOT NULL y distinto de 0.

```
ALTER TABLE articulos2  
ADD CONSTRAINT ck_articulos_pu  
CHECK(precio_unidad IS NOT NULL AND precio_unidad<>0);
```

Borrado de restricciones de una tabla

Permite quitar aquella restricción de la tabla que haya dejado de ser necesaria.

Formato para borrar restricciones de una tabla

```
ALTER TABLE tabla DROP UNIQUE nombre_de_columna [CASCADE]  
PRIMARY KEY  
FOREIGN KEY nombre_de_columna  
CONSTRAINT nombre_constraint
```

Este formato para el borrado de alguna restricción es general para casi todos los gestores SQL.

La opción `CASCADE` hace que se eliminen en cascada todas las restricciones de integridad que dependen de la restricción eliminada.

Si en la constraint `FOREIGN KEY` de la `CREATE TABLE` no se ha especificado, y tampoco se ha usado la cláusula `CASCADE`, **para cambiar o borrar una clave primaria es necesario borrar previamente las posibles claves ajenas asociadas a la misma columna, si las hubiera.**

Ejemplos.

1. Borrar de la tabla compradores la restricción ck_compradores_postal aplicada sobre la columna c_postal.

```
ALTER TABLE compradores
DROP CONSTRAINT ck_compradores_postal;
```

2. Borrar de la tabla compradores la restricción pk_compradores_cif para poner como clave primaria la columna cif_comprador.

```
ALTER TABLE compradores
DROP CONSTRAINT pk_compradores_cif;
```

3. Tenemos la siguiente tabla:

```
CREATE TABLE curso (
    cod_curso CHAR(7) PRIMARY KEY,
    fecha_inicio DATE,
    fecha_fin DATE,
    titulo VARCHAR(60),
    cod_siguiente_curso CHAR(7),
    CONSTRAINT ck_fecha CHECK (fecha_fin > fecha_inicio),
    CONSTRAINT fk_curso_cod FOREIGN KEY (cod_siguiente_curso)
    REFERENCES curso ON DELETE SET NULL);
```

Si ahora intentamos borrar la restricción de clave primaria, nos dará un error:

```
ALTER TABLE curso DROP PRIMARY KEY;
```

Por lo que habría que utilizar la siguiente instrucción:

```
ALTER TABLE curso DROP PRIMARY KEY CASCADE;
```

Borrado de columnas de una tabla

Permite quitar aquella columna de la tabla que haya dejado de ser necesaria.

Formato para borrar columnas de una tabla

```
ALTER TABLE tabla DROP COLUMN nombre_de_columna
```

3.10 Borrado de tablas

Para borrar una tabla y su contenido de la base de datos se utiliza la sentencia **DROP TABLE**.

Formato para eliminar una tabla

```
DROP TABLE nombre_de_tabla
```

En Oracle se utiliza la cláusula **CASCADE CONSTRAINT** para borrar las restricciones asociadas a la tabla antes de que sea eliminada.

Ejemplo: Borrar la tabla de artículos.

```
DROP TABLE articulos
```

Si solo se quiere eliminar los datos pero mantener la tabla, existe el comando **TRUNCATE**

3.11 Renombrado de tablas

Para renombrar una tabla en MySQL se usa el comando **RENAME TABLE**:

```
RENAME TABLE nombre_tabla TO nuevo_nombre;
```

En Oracle no hay que indicar el token **TABLE**, basta con poner:

```
RENAME nombre_tabla TO nuevo_nombre;
```

4 BIBLIOGRAFÍA

- Oracle 10 SQL, PL/SQL, SQL*Plus; Jerome Gabillaud; Ediciones ENI
- Gestión de Bases de Datos; Iván López Montalbán; Editorial Garceta.
- Bases de datos relacionales; Celma; Editorial Pearson
- Documentación en Internet