

- 9.5** ¿Cuál de las siguientes afirmaciones sobre el editor de scripts es falsa?
- Se usa cuando tenemos múltiples sentencias SQL para ejecutar.
 - Se pueden ejecutar varias sentencias a la vez con una sola pulsación del botón *Execute*.
 - Al cerrar una pestaña de script nunca se nos pide nada.
 - Las órdenes escritas desde el Editor de Scripts se pueden almacenar en un archivo.
- 9.6** Descarga el archivo BD_UNIDAD9.SQL de la página web del libro. Carga el archivo en el Editor de Scripts y ejecútalo.

10

El lenguaje SQL. Conceptos básicos

Introducción

El lenguaje SQL (Structured Query Language) es una herramienta para organizar, gestionar y recuperar datos almacenados en una base de datos relacional; por tanto, permite la comunicación con el sistema de gestión de la base de datos. Es tan conocido en bases de datos relacionales que muchos lenguajes de programación incorporan sentencias SQL como parte de su repertorio; tal es el caso de PHP. Entre las principales características de SQL podemos destacar las siguientes:

- *Es un lenguaje para todo tipo de usuarios de la base de datos.*
- *Permite hacer cualquier consulta de datos.*
- *No sólo es un lenguaje de consultas, pues es posible mejorarlo para actualizaciones, definiciones de datos y control de transacciones.*
- *Se puede usar de forma interactiva y de forma embedida.*

Contenido

- 10.1. Un poco de historia y estándares**
 - 10.2. Cómo funciona**
 - 10.3. Tipos de sentencias SQL**
 - 10.4. Elementos de las sentencias SQL**
 - 10.5. Tipos de datos**
- Ejercicios propuestos

Objetivos

- Entender cómo funciona SQL
- Distinguir los tipos de sentencias SQL
- Distinguir los elementos de una sentencia SQL
- Identificar operadores y constantes en una expresión
- Identificar y diferenciar los tipos de datos

10.1 Un poco de historia y estándares

SQL fue desarrollado sobre un prototipo de gestor de bases de datos relacionales denominado SYSTEM R y diseñado por IBM a mediados de los años setenta. Incluía lenguajes de consultas, entre ellos **SEQUEL** (*Structured English Query Language*). Más tarde se renombró como SQL.

En 1979 Oracle Corporation presentó la primera implementación comercial de SQL, que estuvo disponible antes que otros productos de IBM. Por su parte, IBM desarrolló productos herederos del prototipo SYSTEM R, como DB2 y SQL/DS.

El instituto **ANSI** (*American National Standard Institute*) adoptó el lenguaje SQL como estándar para la gestión de bases de datos relacionales en octubre de 1986. En 1987 lo adopta **ISO** (*International Standardization Organization*).

En 1989 el estándar ANSI/ISO revisado y ampliado se llamó **SQL-89** o estándar **SQL1**. En 1992 se aprueba el estándar **ANSI SQL2** o **SQL-92**. En 1999 se aprueba el estándar **SQL:1999** que introduce mejoras con respecto al anterior. **SQL:2003** es el actual estándar que sustituye al anterior. Revisa todos los apartados de SQL:1999 y añade uno nuevo, el apartado 14: SQL/XML. La Figura 10.1 muestra un esquema con la evolución de los estándares y las novedades que va incorporando cada uno con respecto al anterior.



Figura 10.1. Estándares SQL

10.2 Cómo funciona

El gran crecimiento de las redes de ordenadores en los años noventa tuvo importante impacto en la gestión de las bases de datos y dio a SQL una nueva importancia. A medida que las redes se hicieron más comunes, las aplicaciones que se ejecutaban en un miniordenador o en un gran sistema central pasaron a redes de área local de estaciones de trabajo y servidores. En estas redes, SQL desempeña un gran papel como enlace entre la aplicación que se ejecuta en la estación de trabajo con una interfaz gráfica de usuario y el SGBD que gestiona los datos compartidos en un servidor.

La Figura 10.2 muestra cómo funciona SQL en una arquitectura Cliente/Servidor, donde los ordenadores personales se combinan en una red de área local con un servidor

de bases de datos que almacena los datos compartidos. Por un lado se dispone de una máquina servidora con una base de datos que contiene datos importantes para el negocio. Por otro lado tenemos una máquina cliente con un usuario que está ejecutando una aplicación que necesita acceder a los datos allí almacenados. La aplicación realiza una petición al SGBD, éste la procesa y envía los datos a la aplicación que los solicitó.

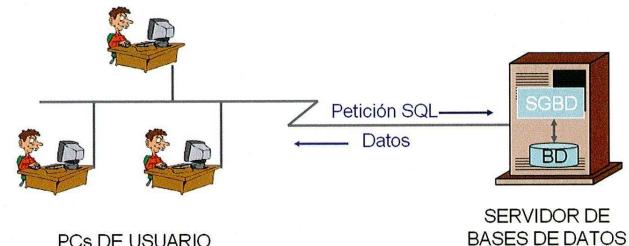


Figura 10.2. Arquitectura Cliente/Servidor

Con el auge de Internet y especialmente del World Wide Web, la arquitectura de las bases de datos de red dio otro paso en su evolución: mediante el uso de un navegador Web se da acceso a los clientes a la información contenida en una base de datos. Esto exige conectar el servidor Web con el sistema de bases de datos que contiene la información. Los métodos utilizados para conectar los servidores Web con los SGBD hicieron que surgiese la arquitectura de tres capas que puede verse en la Figura 10.3.

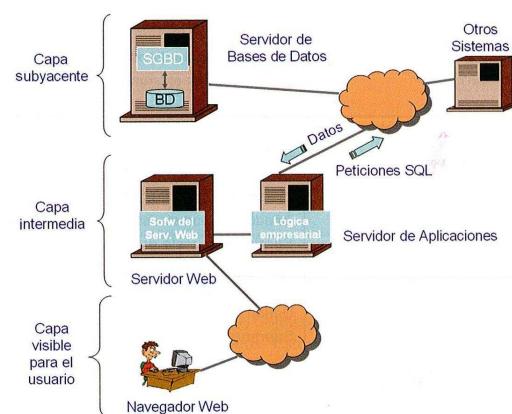


Figura 10.3. Arquitectura de tres capas

Las tres capas son las siguientes:

- **Capa visible para el usuario.** La interfaz de usuario es un navegador Web que se ejecuta en el ordenador personal del usuario o algún otro dispositivo de cliente ligero (móvil, PDA). Desde aquí se realizan las solicitudes.
- **Capa intermedia.** El navegador Web se comunica con un servidor Web en esta capa. Cuando la solicitud del usuario es más compleja que una simple página web,

el servidor Web transfiere la solicitud a un *Servidor de Aplicaciones* cuyo papel es manejar la lógica empresarial necesaria para procesar la solicitud. El servidor Web y el servidor de Aplicaciones pueden residir en la misma o en distinta máquina.

- **Capa subyacente.** A menudo la solicitud implica el acceso a una base de datos empresarial. Este sistema se ejecuta en la capa subyacente.

Aunque hemos visto dos tipos de arquitecturas de red de bases de datos y el papel de SQL en cada una de ellas, también podemos utilizar SQL desde una máquina en la que tengamos instalado todo el software: el SGBD, las aplicaciones para acceder a la base de datos, el servidor Web y el servidor de aplicaciones.

10.3 Tipos de sentencias SQL

El lenguaje SQL proporciona un gran repertorio de sentencias, que se utilizan en variadas tareas, como consultar datos de la base de datos, crear, actualizar y eliminar objetos de la base de datos, crear, actualizar y eliminar datos de los objetos, controlar el acceso a la base de datos y a los objetos. Dependiendo de las tareas, podemos clasificar las sentencias SQL en varios tipos. Véase la Tabla 10.1. Existen más sentencias SQL; se han expuesto las más importantes y usadas con mayor frecuencia.

SENTENCIA		DESCRIPCIÓN
DML (Lenguaje de Manipulación de Datos)	Manipulación de datos SELECT INSERT DELETE UPDATE	Recupera datos de la base de datos. Añade nuevas filas de datos a la base de datos. Suprime filas de datos de la base de datos. Modifica datos existentes en la base de datos.
DDL (Lenguaje de Definición de Datos)	Definición de datos CREATE TABLE DROP TABLE* ALTER TABLE* CREATE VIEW* DROP VIEW* CREATE INDEX* DROP INDEX* CREATE SYNONYM* DROP SYNONYM*	Añade una nueva tabla a la base de datos. Suprime una tabla de la base de datos. Modifica la estructura de una tabla existente. Añade una nueva vista a la base de datos. Suprime una vista de la base de datos. Construye un índice para una columna. Suprime el índice para una columna. Define un alias para un nombre de tabla. Suprime un alias para un nombre de tabla.
DCL (Lenguaje de Control de Datos)	Control de acceso GRANT REVOKE Control de transacciones COMMIT ROLLBACK	Concede privilegios de acceso a usuarios. Suprime privilegios de acceso a usuarios. Finaliza la transacción actual. Aborta la transacción actual.

* No existían en el estándar SQL1.

Tabla 10.1 Tipos de sentencias SQL

Actividad de Aplicación 10.1

Abre MySQL Query Browser y consulta desde el Navegador de Información y desde el grupo *Syntax* las órdenes de manipulación de datos (*Data Manipulation*), definición de datos (*Data Definition*) y de control de transacciones (*Transactional and Locking*) que tiene MySQL.



10.4 Elementos de las sentencias SQL

Casi todas las sentencias SQL tienen una forma básica. Véase la Figura 10.4. Todas comienzan con un **verbo**, que es una palabra clave que describe lo que hace la sentencia (por ejemplo, SELECT, INSERT, UPDATE, CREATE). A continuación le siguen una o más **cláusulas** que especifican los datos con los que opera la sentencia o más detalles acerca de lo que hace la sentencia. Éstas también comienzan con una palabra clave como WHERE o FROM. Algunas son opcionales y otras obligatorias. Muchas contienen nombres de tablas o de columnas, palabras reservadas, operadores, constantes o expresiones adicionales.

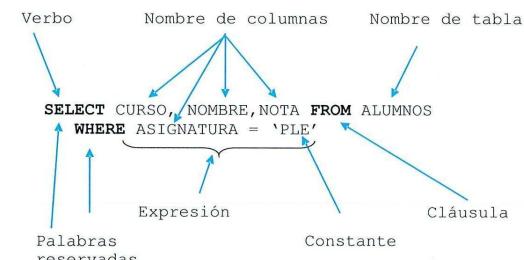


Figura 10.4. Componentes de una sentencia SQL

10.4.1. Nombres de objetos

En una base de datos basada en SQL los objetos, como por ejemplo una tabla o una vista, se identifican asignándoles nombres únicos. Estos nombres se utilizarán en las sentencias SQL para identificar el objeto de base de datos sobre el que actuará la sentencia.

En MySQL las reglas para nombrar los objetos son las siguientes:

- Un nombre consiste en un conjunto de caracteres alfanuméricos (letras y números), incluidos los caracteres guión de subrayado ‘_’ y dólar ‘\$’. Se admiten hasta 64 caracteres.
- Puede empezar por cualquier carácter de los anteriores, pero no puede consistir sólo en números.
- Se aplican las mismas reglas para las columnas de una tabla.

Son nombres válidos:

EMPLEADOS \$SALARIO_ 12SUMA RESTA\$\$

10.4.2. Constantes

Una constante es un valor que se expresa en forma textual. Existen varios tipos de constantes:

- Numéricas.** Pueden ser enteras, decimales y en coma flotante. Las constantes en coma flotante se especifican usando la notación E o e, que significa 10 elevado a. Por ejemplo: 2.4E3 significa 2.4 por 10 elevado a 3. Ejemplos:

Enteras	Decimales	En coma flotante
34 -27 2000	34.76 -27.00 0.06	2.4E3 2.4e3 -2.4E3 2.4E-3

- De cadena.** Una cadena es un conjunto de caracteres encerrados entre comillas simples ('...') o dobles ("..."). Ejemplos:

'Hola' 'Hola y adiós' "1a2sd3ee*" '222aAAA---'

- De fecha y hora.** Se expresan como constantes de caracteres encerradas entre comillas o como constantes enteras. Representan la fecha, la hora o una combinación de ambas.

Fecha	Hora	Combinación de ambas
"20060606"	"17:23:49"	"20060606172349"
'2006-06-06'	'17:23:49'	'2006-06-06 17:23:49'
20060606	172349	20060606172349

- Simbólicas.** Son constantes que devuelven valores de datos mantenidos por el SGBD. Ejemplos: **CURRENT_DATE**, devuelve la fecha actual; **CURRENT_TIME**, devuelve la hora; **CURRENT_USER**, devuelve el usuario conectado; etc.



A las constantes también se las llama *literales*.

Actividad de Aplicación 10.2

Desde el Área de Consulta SQL ejecuta las siguientes órdenes que utilizan constantes:

```
SELECT 34, -27, 34.76, -27.00, 0.06, 2.4E3, 2.4e3, -2.4E3, 2.4E-3;
SELECT "Hola", 'Hola y adiós', "1a2sd3ee*", '222aAAA---';
SELECT "20060606", '2006-06-06', "17:23:49", '2006-06-06 17:23:49';
SELECT CURRENT_DATE, CURRENT_TIME, CURRENT_USER;
```

10.4.3. Operadores

MySQL dispone de todos los operadores habituales en otros lenguajes de programación: aritméticos, de comparación y lógicos.

Los operadores aritméticos sirven para realizar operaciones aritméticas. Forman parte de expresiones con constantes, valores de columnas y funciones de valores de columnas. Son los que podemos observar en la Tabla 10.2.

Operador aritmético	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo. Devuelve el resto de una división. Por ejemplo 9 % 2 es 1, ya que el resto de dividir 9 entre 2 es 1.

Tabla 10.2. Operadores aritméticos

Se pueden combinar entre sí para formar expresiones aritméticas. Se pueden utilizar paréntesis para forzar el orden de evaluación en una expresión. Por ejemplo las siguientes órdenes SQL generan distinto resultado:

```
SELECT 10+20*2;           El valor de la expresión es 50
SELECT (10+20)*2;         El valor de la expresión es 60
```

Los operadores de comparación comparan un valor con otro. El resultado de una operación de comparación es verdadero (TRUE = 1), falso (FALSE = 0) o nulo (NULL). Son los que podemos observar en la Tabla 10.3.

Operador de comparación	Operación
=	Igual a
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
!= <>	Distinto de
<=>	Igual a y además NULL. Es similar al operador =, pero devuelve 1 en vez de NULL si los valores que se comparan son NULOS, y 0 si uno de los valores es NULO. Un valor NULO es un valor que no tiene información.

Tabla 10.3. Operadores de comparación

Ejemplos:

```
SELECT 10 = 10; Devuelve 1 (verdadero) ya que 10 es igual a 10.
SELECT 10 > 20; Devuelve 0 (falso) ya que 10 no es mayor que 20.
SELECT 10 < 20; Devuelve 1 (verdadero) ya que 10 es menor que 20.
SELECT 'pepe' = 'PEPE'; Devuelve 1 (verdadero) ya que, por defecto, las comparaciones de cadenas se hacen ignorando el uso de mayúsculas/minúsculas.
SELECT 'pepe' >= NULL; Devuelve NULL ya que uno de los valores es nulo.
SELECT 'pepe' <= 'pepa'; Devuelve 0 (falso) ya que la cadena 'pepe' no es alfádicamente menor que la cadena 'pepa'.
SELECT 'pepe' <=> NULL; Devuelve 0 ya que uno de los valores es nulo.
```

Actividad de Aplicación 10.3

Desde el Área de Consulta SQL ejecuta las siguientes órdenes y analiza los resultados:

```
SELECT 34<>27, 34 = 76, 10<=9;
SELECT "Hola" = 'Hola', "Hola" <=> 'Hola', "Hola" > "SQL";
```





Actividad de Aplicación 10.4

Sean A = -10, B = 32, C = 9.25 y D = 9. ¿Qué resultados TRUE (1) o FALSE (0) devuelven estas comparaciones?:

A <= B C > A D = C D <> C D > C D = C A > C

Los operadores lógicos se utilizan para combinar y comparar expresiones. Devuelven un valor cierto (TRUE = 1) si la expresión es verdadera y falso (FALSE = 0) si es falsa. Si una de las expresiones es nula devuelve un valor NULL. Se resumen en la Tabla 10.4.

Operador lógico	Operación
NOT	<i>NOT Expresión1</i> Devuelve el valor TRUE (1) si la expresión es falsa.
AND	<i>Expresión1 AND Expresión2</i> Devuelve el valor TRUE (1) cuando las dos expresiones son verdaderas.
OR	<i>Expresión1 OR Expresión2</i> Devuelve el valor TRUE (1) cuando una de las expresiones es verdadera.

Tabla 10.4. Operadores lógicos

Ejemplos:

NOT. Devuelve 1 si la expresión es 0. Devuelve 0 si la expresión es distinta de 0. La expresión NOT NULL devuelve 0:

```
SELECT NOT 20;           Devuelve 0.
SELECT NOT 0;            Devuelve 1.
SELECT NOT NULL;         Devuelve 0.
```

AND. Devuelve 1 si la expresión es cierta; en caso contrario devuelve 0. Devuelve NULL si alguna de las expresiones es nula:

```
SELECT 20 > 30 AND 10 = 10;    Devuelve 0 ya que la primera expresión no es verdadera.
SELECT 40 > 30 AND 10 = 10;    Devuelve 1 ya que las dos expresiones son verdaderas.
SELECT 40 = 30 AND 10 < 10;    Devuelve 0 ya que ninguna expresión es verdadera.
```

OR. Devuelve 1 siempre que una de las expresiones sea verdadera:

```
SELECT 20 > 30 OR 10 = 10;    Devuelve 1 ya que la segunda expresión es verdadera.
SELECT 40 > 30 OR 10 = 10;    Devuelve 1 ya que las dos expresiones son verdaderas.
SELECT 40 = 30 OR 10 < 10;    Devuelve 0 ya que ninguna expresión es verdadera.
```

Actividad de Aplicación 10.5

Desde el Área de Consulta SQL ejecuta las siguientes órdenes y analiza los resultados:

```
SELECT 34<>27 AND 34 = 76;
SELECT 34<>27 OR 34 = 76;
SELECT 10<=9 OR "Hola" = 'Hola';
SELECT 10<=9 AND "Hola" = 'Hola';
```



Actividad de Aplicación 10.6

Sean A = -10, B = 32, C = 9.25 y D = 9. ¿Qué resultados TRUE (1) o FALSE (0) devuelven estas comparaciones?:

```
A <= B AND C > A
D = C OR D <> C
D > C OR D = C
NOT A > C
```



Existen otros operadores de comparación como LIKE, BETWEEN o IN que se verán en el siguiente capítulo.

10.4.4. Expresiones

Las expresiones se utilizan en las órdenes SQL. Pueden contener constantes, operadores, expresiones aritméticas y lógicas, columnas de tablas y funciones. Una expresión se evalúa de izquierda a derecha y se pueden utilizar paréntesis para forzar el orden de evaluación. Ejemplo de expresiones:

```
(7 + 3) < 10
(A + B) < 10 AND C = 3
NOTA1=7 AND (NOTA1+NOTA2+NOTA3)/3 >6;
```

La precedencia de los operadores se muestra en la siguiente lista ordenados de menor a mayor prioridad; los operadores que están en la misma línea tienen la misma prioridad:

OR
AND
NOT
BETWEEN
=, <=>, >=, >, <, <>, !=, IS, LIKE, IN
- , +
* , / , %



Actividad de Aplicación 10.7

Sean A = -10, B = 32, C = 9.25 y D = 9. ¿Qué resultados TRUE (1) o FALSE (0) devuelven estas comparaciones?:

```
(A > B) AND (B - A) > 40
(A + B) < 10 OR (B / 8) < 5
(A + B) < 10 AND C = 3
((D > C) AND (D = C)) OR C > 9
(A <= B + 5) OR NOT A > C
```

10.4.5. Cómo procesa el sistema gestor de base de datos una sentencia

Para procesar una sentencia SQL el sistema gestor de base de datos recorre una serie de pasos:

- Analiza la sentencia. Comprueba que está sintácticamente bien escrita.
- Valida la sentencia. Comprueba la sentencia semánticamente. Es decir, comprueba si las tablas y las columnas referenciadas en la sentencia existen, si el usuario tiene privilegios para realizar esa operación sobre la tabla, etc.
- Optimiza la sentencia. El sistema gestor de base de datos explora la forma de llevar a cabo la ejecución de la sentencia.
- Genera un plan de aplicación para la sentencia. Se genera código ejecutable.
- Ejecución del plan de aplicación.

El análisis de la sentencia no requiere acceso a la base de datos y, por tanto, suele realizarse rápidamente. Sin embargo, la optimización es un proceso muy intensivo de CPU y precisa acceder a la base de datos.

10.5 Tipos de datos

Son varios los tipos de datos que se pueden almacenar en una base de datos basada en MySQL y manipular por el lenguaje SQL. Se pueden clasificar en tres categorías: numéricos, de fecha y hora y de cadenas de caracteres.

10.5.1. Tipos numéricos

MySQL soporta todos los tipos de datos numéricos del estándar ANSI SQL-92. Estos tipos incluyen los tipos de datos exactos (**NUMERIC**, **DECIMAL**, **INTEGER** y **SMALLINT**) y los tipos de datos aproximados (**FLOAT**, **REAL** y **DOUBLE**).

Los números enteros se representan como una secuencia de dígitos. Los números en coma flotante usan el punto como separador decimal. Cualquier tipo de número puede ir precedido por un signo ‘-’ para indicar un valor negativo. Un entero puede ser usado

en un contexto de valores de coma flotante; éste es interpretado como el equivalente en valor de coma flotante.

La Tabla 10.5 muestra los tipos de datos numéricos y la cantidad de almacenamiento requerido.

Se utilizan las siguientes convenciones en las descripciones de los tipos:

N: En tipos enteros se usa a efectos de visualización para ajustar por la izquierda la representación de valores cuyo tamaño es inferior al especificado en la columna. Cuando se utiliza conjuntamente con el atributo **ZEROFILL**, se rellena con ceros a la izquierda hasta tener una longitud N. Si no se especifica, se le asigna como ancho predeterminado la longitud más larga de los valores de cada tipo. Para los tipos de coma flotante, N es el número de cifras enteras más el número de cifras decimales. El valor está comprendido entre 1 y 255.

D: Se aplica a los datos de coma flotante e indica el número de dígitos que siguen al punto decimal. El valor máximo posible es de 30, pero no debe ser mayor de N-2.

[]: Los corchetes indican que el elemento es opcional.

A todos estos tipos se les pueden añadir los atributos **UNSIGNED** y **ZEROFILL**. **ZEROFILL** significa que se rellena toda la anchura del valor con ceros a la izquierda. **UNSIGNED** significa que el tipo almacena datos sin signo; los datos con signo negativo no se almacenarán.

TIPO	DESCRIPCIÓN	ALMACENAMIENTO REQUERIDO
TINYINT [(N)]	Representa un entero muy pequeño. El rango de valores que puede almacenar va de -128 a 127. Para enteros sin signo el rango es de 0 a 255.	1 byte
SMALLINT [(N)]	Representa un entero pequeño. Su rango es de -32.768 a 32.767. Para enteros sin signo el rango es de 0 a 65535.	2 bytes
MEDIUMINT [(N)]	Representa un entero de tamaño medio. Su rango es de -8.388.608 a 8.388.607. Para enteros sin signo el rango es de 0 a 16.777.215.	3 bytes
INT [(N)] INTEGER [(N)]	Representa un entero de tamaño normal. Su rango es de -2.147.483.648 a 2.147.483.647. Para enteros sin signo el rango es de 0 a 4.294.967.295.	4 bytes
BIGINT [(N)]	Es un entero grande. Su rango es de -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807. Para enteros sin signo el rango es de 0 a 18.446.744.073.709.551.615.	8 bytes
FLOAT [(N, D)]	Representa un número pequeño de coma flotante de simple precisión. El rango de valores permitidos es de -3.402823466E+38 a -1.175494351E-38, 0, y de 1.175494351E-38 a 3.402823466E+38. N es el número de dígitos totales del número y D el número de decimales.	4 bytes
DOUBLE [(N, D)] REAL [(N, D)]	Representa un número grande en coma flotante y doble precisión. Los valores permitidos son de: -1.797693138623157E+308 a -2.2250738585072014E-308 y de 2.2250738585072014E-308 a 1.797693138623157E+308.	8 bytes
DECIMAL [(N[, D])] NUMERIC [(N[, D])]	Representa un número de coma flotante almacenado como cadena. El punto decimal, y para números negativos el signo menos, no son contados en N (sin embargo se les reserva espacio). El máximo rango de valores es el mismo que para DOUBLE, pero el rango para una columna DECIMAL puede ser restringido con la elección de valores en N y D.	N+2 bytes si D > 0, N+1 bytes si D = 0, D+2, si N < D
DEC [(N[, D])]	Sinónimo de DECIMAL.	

Tabla 10.5. Tipos de datos numéricos

Para declarar una columna de una tabla de alguno de estos tipos es necesario indicar el nombre de la columna y el tipo. Ejemplos:

salario `FLOAT(6,2)` => define la columna *SALARIO* de tipo FLOAT con 6 dígitos de precisión; de ellos, dos son decimales. Se pueden almacenar cantidades positivas y negativas desde -9999.99 a 9999.99.

edad `INT` => define la columna *EDAD* de tipo entero. Se podrán almacenar valores que estén dentro del rango definido para el tipo INT, es decir, de -2.147.483.648 a 2.147.483.647.

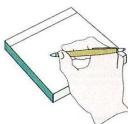
curso `INT(1)` => en este caso se define la columna *CURSO* de tipo entero. Admite cualquier valor dentro del rango definido para el tipo INT. El valor 1 no restringe el rango de valores que se pueden almacenar en la columna.



Actividad de Aplicación 10.8

Abre MySQL Query Browser y crea un nuevo esquema de base de datos de nombre UNIDAD10 para cargar el script que crea las tablas con las que trabajaremos en el siguiente ejercicio.

A continuación carga el script TABLA_TIPOSNUMERICOS.SQL que se encuentra en la página web del libro en el Editor de Scripts y ejecútalo.



Ejercicio Resuelto 10.1

En primer lugar revisamos el contenido del script. Se crean dos tablas:

Tabla TIPOSDECIMALES

```
CREATE TABLE tiposdecimales(
    decimal1 DECIMAL,
    decimal2 DECIMAL(5,2),
    decimal3 FLOAT(4,2),
    decimal4 DOUBLE(7,3) UNSIGNED
);
```

Tabla TIPOSENTEROS

```
CREATE TABLE tiposenteros(
    entero INT UNSIGNED,
    enteropeque TINYINT(4),
    enterograndel BIGINT(2) ZERO-
    FILL,
    enterograndel2 BIGINT
);
```

Ambas tablas definen 4 columnas. Las columnas de la tabla TIPOSDECIMALES son:

- `decimal1 DECIMAL` => Define un número en coma flotante. Admite el máximo rango de valores definidos para este tipo.
- `decimal2 DECIMAL(5,2)` => Define un número en coma flotante con 5 posiciones numéricas; dos de ellas son decimales. Admite los valores de -999.99 a 999.99.
- `decimal3 FLOAT(4,2)` => Define un número en coma flotante con 4 posiciones numéricas; dos de ellas son decimales. Admite los valores de -99.99 a 99.99.
- `decimal4 DOUBLE(7,3) UNSIGNED` => Define un número en coma flotante con 7 posiciones numéricas; tres de ellas son decimales, y sin signo; es decir, no admite números negativos. Admite los valores de 0 a 9999.999.

Escribimos las siguientes órdenes desde el Área de Consulta SQL:

1. Se inserta una fila en la tabla dando valor a la columna decimal4:

```
INSERT INTO tiposdecimales (decimal4) VALUES (-23);
```

La orden no se ejecuta. Aparece el error mostrado en la Figura 10.5.

The query could not be executed.		Edit	Apply Changes	Discard Changes	First	Last	Search	ErrorNr.
!	Description							
!	Out of range value adjusted for column 'decimal4' at row 1							1264

Figura 10.5. Consulta 1. Ejercicio 10.1

El error se debe a que el valor que dimos a la columna es negativo; y dicha columna no admite valores negativos ya que está definida con el atributo UNSIGNED. Modificamos la orden y escribimos lo siguiente:

```
INSERT INTO tiposdecimales (decimal4) VALUES (23);
```

Ahora si se inserta correctamente la fila.

2. Se inserta una fila en la tabla dando valor a la columna decimal2:

```
INSERT INTO tiposdecimales (decimal2) VALUES (12345);
```

La orden no se ejecuta. Aparece el error anterior debido a que esta columna almacena valores comprendidos entre -999.99 y 999.99. Probamos la siguiente orden:

```
INSERT INTO tiposdecimales (decimal2) VALUES (123.45);
```

Ahora si se inserta correctamente la fila.

3. Ésta también se insertará correctamente:

```
INSERT INTO tiposdecimales (decimal2) VALUES (-123.45);
```

Actividad de Aplicación 10.9

A partir de la tabla TIPOSENTEROS vista anteriormente, analiza sus columnas y ejecuta las siguientes órdenes SQL. Analiza los resultados de la ejecución.

```
INSERT INTO tiposenteros (entero) VALUES (-23);
```

```
INSERT INTO tiposenteros (enteropeque) VALUES (1234);
```

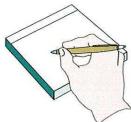


10.5.2. Tipos fecha y hora

MySQL soporta tipos de datos para fecha, hora y combinaciones de ambos. Se pueden utilizar cadenas de caracteres y números para almacenar fechas. Por ejemplo, son fechas válidas: 20060531 y "2006-05-31". La Tabla 10.6 muestra los tipos fecha y hora y la cantidad de almacenamiento requerido.

TIPO	DESCRIPCIÓN	ALMACENAMIENTO REQUERIDO
DATE	Representa una fecha en formato 'AAAA-MM-DD'. El rango válido es de '1000-01-01' a '9999-12-31'.	3 bytes
DATETIME	Representa una combinación de fecha y hora en formato: 'AAAA-MM-DD HH:MM:SS'. El rango de valores válidos es de '1000-01-01 00:00:00' a '9999-12-31 23:59:59'.	8 bytes
TIMESTAMP	Representa una combinación de fecha y hora en formato 'AAAAMMDD HHMMSS'. El rango de valores es de '19700101000000' a algo antes del año 2037.	4 bytes
TIME	Representa la hora en formato 'HH:MM:SS'. Su rango es de '-838:59:59' a '838:59:59'.	3 bytes
YEAR [(2 4)]	Representa un año en formato AAAA. Los valores permitidos son de 1901 a 2155 en el formato de 4 dígitos (almacena los cuatro dígitos del año), y 1970-2069 si se usa el formato de 2 dígitos (70-69, almacena los dos últimos dígitos del año).	1 byte

Tabla 10.6. Tipos fecha y hora



Ejercicio Resuelto 10.2

Carga el script TABLA_TIPOSFECHAHORA.SQL que se encuentra en la página web del libro en el Editor de Scripts y ejecútalo. El contenido del script crea una tabla con varias columnas de tipo fecha y hora. El tipo definido para cada columna determina la información que se puede almacenar en ella:

```
CREATE TABLE tiposfechahora(
    fecha1      DATE,
    fecha2      DATETIME,
    fecha3      TIMESTAMP,
    fecha4      TIME,
    fecha5      YEAR(2)
);
```

Los distintos valores que se pueden asignar para cada columna se muestran en la Tabla 10.7.

COLUMNA	TIPO	VALORES CORRECTOS	FORMATO PERMITIDO
fecha1	DATE	"2006-05-31" "06-05-31" "20060531", 20060531 "060531", 060531	"AAAA-MM-DD" "AA-MM-DD" "AAAAMMDD", AAAAMMDD "AAMMDD", AAMMD
fecha2	DATETIME	"2006-05-31 10:10:56" "06-05-31 10:10:56" "20060531101056" "060531101056" "20060531101056 "060531101056"	"AAAA-MM-DD HH:MM:SS" "AA-MM-DD HH:MM:SS" "AAAAMMDD HHMMSS" "AAMMDD HHMMSS" "AAAAMMDD HHMMSS "AAMMDD HHMMSS"
fecha3	TIMESTAMP	Igual que el tipo DATETIME	
fecha4	TIME	"10:10:56" "101056" 101056	"HH:MM:SS" "HHMMSS" HHMMSS
fecha5	YEAR (2)	06, 2006 "06", "2006"	AA, AAAA "AA", "AAAA"

Tabla 10.7. Valores para los formatos de fecha

Escribimos las siguientes órdenes desde el Área de Consulta SQL:

1. Se inserta una fila en la tabla dando valor a todas las columnas menos a la columna fecha3 definida como TIMESTAMP:

```
INSERT INTO tiposfechahora (fecha1, fecha2, fecha4, fecha5)
VALUES ("06-05-31", 060531101056, "10:10:56", 2006);
```

2. A continuación ejecutamos la siguiente orden para visualizar la fila que hemos insertado:

```
SELECT * FROM tiposfechahora;
```

fecha1	fecha2	fecha3	fecha4	fecha5
2006-05-31	2006-05-31 10:10:56	2006-10-26 01:22:14	10:10:56	06

Figura 10.6. Consulta 2. Ejercicio 10.2.

Se observa que la columna FECHA3 a la que no dimos valor al insertar aparece con un valor. Y es que al ser de tipo TIMESTAMP se le proporciona un valor que es aplicable automáticamente en operaciones como INSERT y UPDATE con la fecha y hora actuales. Si se tienen múltiples columnas TIMESTAMP, sólo la primera es actualizada automáticamente.



Actividad de Aplicación 10.10

Ejecuta las siguientes órdenes SQL y analiza lo que ocurre. Comenta los resultados de la ejecución.

```
INSERT INTO tiposfechahora (fecha1) VALUES ("2006-06-31");
INSERT INTO tiposfechahora (fecha4) VALUES ("10:10:66");
```

10.5.3. Tipos cadena de caracteres

Una cadena es una secuencia de caracteres encerrados entre comillas simples o dobles. Son cadenas válidas las siguientes: 'Unidad 10', "Unidad 10".

La Tabla 10.8 muestra los tipos cadena de caracteres y la cantidad de almacenamiento requerido. Se utilizan las siguientes convenciones en las descripciones de los tipos:

N: representa el número máximo de caracteres de la cadena.

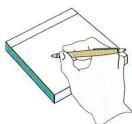
L: representa la longitud de la cadena.

Los tipos de columna BLOB representan objetos binarios largos. Son de longitud variable. BLOB es una familia de tipos (TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB) que se diferencian en la cantidad máxima de información que pueden almacenar.

El tipo TEXT es como un tipo VARCHAR cuya longitud es tan grande como se quiera. De un modo similar a BLOB, TEXT es una familia de tipos (TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT) que se diferencian en la cantidad máxima de información que pueden almacenar.

TIPO	DESCRIPCIÓN	ALMACENAMIENTO REQUERIDO
CHAR (N)	Representa una cadena de longitud fija N. La cadena siempre se rellena con espacios a la derecha para completar la longitud N. El rango de N es de 1 a 255 caracteres.	N bytes
VARCHAR (N)	Representa una cadena de longitud variable. N representa el máximo número de caracteres que puede tener la cadena. El rango de N es de 1 a 255 caracteres.	L+1 byte para almacenar la longitud de la cadena, donde $L \leq N$ y $1 \leq N \leq 255$
TINYBLOB BLOB MEDIUMBLOB LONGBLOB	Pequeño valor BLOB BLOB Normal BLOB Medio BLOB Grande	L+1 byte, donde $L < 2^8$ L+2 bytes, donde $L < 2^{16}$ L+3 bytes, donde $L < 2^{24}$ L+4 bytes, donde $L < 2^{32}$
TINYTEXT TEXT MEDIUMTEXT LONGTEXT	Pequeño valor TEXT TEXT Normal TEXT Medio TEXT Grande	L+1 byte, donde $L < 2^8$ L+2 bytes, donde $L < 2^{16}$ L+3 bytes, donde $L < 2^{24}$ L+4 bytes, donde $L < 2^{32}$
ENUM('valor1','valor2',...)	Una columna de tipo ENUM define un conjunto de cadenas. A la columna ENUM se le asignará un valor del conjunto.	1 o 2 bytes, dependiendo del número de valores (máximo 65.535)
SET('valor1','valor2',...)	Igual que ENUM, pero una columna de tipo SET puede tener 0 o más valores del conjunto.	1, 2, 3, 4, u 8 bytes, dependiendo del número de valores (máximo 64)

Tabla 10.8. Tabla de cadenas de caracteres



Ejercicio Resuelto 10.3

Carga el script TABLA_TIPOS CADENA.SQL que está en la página web del libro en el Editor de Scripts y ejecútalo. El contenido del script crea una tabla con varias columnas de tipo cadena de caracteres. El tipo definido para cada columna determina la información que se puede almacenar en ella:

```
CREATE TABLE tiposcadena(
cad1    CHAR(5),
cad2    VARCHAR(5),
cad3    ENUM("Primera", "Segunda", "Tercera"),
cad4    SET("Uno", "Dos", "Tres"),
cad5    TINYBLOB,
cad6    TINYTEXT
);
```

Las columnas de la tabla TIPOS CADENA son:

- cad1 CHAR (5) => CAD1 es una cadena de longitud fija de 5 caracteres.
- cad2 VARCHAR (5) => CAD2 es una cadena de longitud variable. El máximo número de caracteres que puede almacenar es 5.
- cad3 ENUM ("Primera", "Segunda", "Tercera") => CAD3, al ser de tipo ENUM, los únicos valores válidos que puede almacenar son "Primera", "Segunda" o "Tercera".
- cad4 SET ("Uno", "Dos", "Tres") => CAD4, al ser de tipo SET, los únicos valores válidos que puede almacenar son "Uno", "Dos" o "Tres" o cualquier combinación de ambos separados por comas, por ejemplo: "Dos, Uno". Cuando se combinan varios valores, el orden de aparición será el definido en la sentencia SET, es decir, para el caso anterior se almacenarán como "Uno, Dos".

- cad5 TINYBLOB => CAD5 es de tipo BLOB de pequeño tamaño.
- cad6 TINYTEXT => CAD6 es de tipo TEXT de pequeño tamaño.

Se van a dar los siguientes valores a las columnas:

cad1	"Hola"
cad2	"Adiós"
cad3	"Primera"
cad4	"Tres,Dos"
cad5	"Lenguaje SQL"
cad6	"Aprendiendo SQL"

Para ello se escribe la orden INSERT siguiente desde el Área de Consulta SQL:

1. Se inserta una fila en la tabla dando valor a todas las columnas:

```
INSERT INTO tiposcadena (cad1, cad2, cad3, cad4, cad5, cad6)
VALUES ("Hola", "Adiós", "Primera",
        "Tres,Dos", "Lenguaje SQL", "Aprendiendo SQL");
```

2. Visualizamos la fila insertada ejecutando la orden siguiente. El resultado de la inserción se muestra en la Figura 10.7:

```
SELECT * FROM tiposcadena;
```

cad1	cad2	cad3	cad4	cad5	cad6
Hola	Adiós	Primera	Dos,Tres	BLOB	Aprendiendo SQL

Figura 10.7. Consulta 2. Ejercicio 10.3

Al ejecutar la sentencia SELECT, la columna BLOB no visualiza nada. Al hacer clic en la lupita que se muestra en el campo se visualizará su contenido. Véase la Figura 10.8. El icono de disquete que acompaña al campo nos permitirá almacenar el contenido del mismo en un archivo.

cad1	cad2	cad3	cad4	cad5	cad6
Hola	Adiós	Primera	Dos,Tres	BLOB	Aprendiendo SQL

Field Viewer

Binary

0 4C 65 6E 67 75 61 6A 65 20 53 51 4C	Lenguaje SQL
---	--------------

Size: 12 Byte

OK

Figura 10.8. Visualizando una columna BLOB



Actividad de Aplicación 10.11

Ejecuta las siguientes órdenes SQL y analiza lo que ocurre. Comenta los resultados de la ejecución.

```
INSERT INTO tiposcadena (cad4) VALUES ("Tres,Dos,Uno");
SELECT cad4 FROM tiposcadena;
INSERT INTO tiposcadena (cad2) VALUES ("abcdef");
INSERT INTO tiposcadena (cad2) VALUES (123);
INSERT INTO tiposcadena (cad2) VALUES (123aa);
```



Direcciones Web de interés:

Zona de descarga de documentación sobre MySQL:
<http://dev.mysql.com/doc/>

Manual de referencia de MySQL en castellano:
<http://dev.mysql.com/doc/refman/5.0/es/index.html>

Fuente de consulta y referencia para el aprendizaje de MySQL: artículos, noticias, eventos, etc.
<http://www.mysql-hispano.org/>

Tutorial básico de MySQL:
http://www.programacion.com/bbdd/tutorial/mysql_basico/



10.1 La siguiente tabla muestra diferentes valores que hay que relacionar con su tipo de dato.

Valor	Tipo de dato
"2006-05-31"	
1020	Numérico entero
12.78	Numérico decimal
'valor2'	Numérico en coma flotante
"06-05-31 10:10:56"	Cadena de caracteres
123.56e32	Fecha y hora
-12e-3	Fecha
"2006-05-31 10:10:56"	Hora
2.45440E-3	
"Pepe y Pepa"	
"10:10:56"	

10.2 Rellena la siguiente tabla obteniendo el resultado de las expresiones y sabiendo que los valores de A, B, C y D son éstos: A = 15, B = -12, C = 10.45, D = 21.54.

Expresión	Resultado TRUE (1) o FALSE (0)
A > B	
NOT C <= D	
B<> D OR NOT A = B	
(B > A) AND (B - A) > 30	
(A + C) < 10 AND C = (3 - B)	
NOT (B + C) < 10 OR (A / 8) < 5	
(C + D) < 200 AND C = 3	
NOT (A > B AND D < C)	
NOT A > C AND ((A <= B + 5) OR (A = 15))	
((D > C) AND (D = C)) OR C > 9) AND (D < 20)	

10.3 Describe las características de las siguientes columnas. Para las numéricas, indica el valor máximo y mínimo que pueden almacenar.

cantidad1 FLOAT	cadena1 CHAR (10)
cantidad2 INT	cadena2 VARCHAR (50)
cantidad3 FLOAT(5,3)	cadena3 ENUM("A", "B", "C")
cantidad4 SMALLINT(4) UNSIGNED	cadena4 SET("A", "B", "C")
cantidad5 TINYINT (3) UNSIGNED	

10.4 A partir de las definiciones anteriores indica si los valores que aparecen en la tabla se pueden almacenar en la columna definida.

COLUMNA	VALOR	¿Se puede almacenar? (SÍ/NO)
cantidad1	-3402E+20	
cantidad2	123456789078	
cantidad3	65781.876	
cantidad4	-30000	
cantidad5	250	
cadena1	"LA CASA DE LA PRADERA"	
cadena2	"LA CASA DE LA PRADERA"	
cadena3	""	
cadena4	""	