2 - Crear una tabla (create table - sp_tables - sp_columns - drop table)

Problema:

Vamos a crear una tabla llamada "usuarios". En primer lugar vamos a eliminar la tabla "usuarios" averiguando si existe (a esto vamos a repetirlo siempre porque puede haber otro usuario que haya creado una tabla con el mismo nombre):

```
 if object_id('usuarios') is not null
  drop table usuarios;
```

Recordar que debemos finalizar cada comando con un punto y coma.

La tabla "usuarios" contendrá los siguientes campos:

```
- nombre: varchar de 30 caracteres de longitud,
- clave: varchar de 10 caracteres de longitud.
```

Ahora si creamos la tabla:

```
 create table usuarios (
  nombre varchar(30),
  clave varchar(10)
 );
```

aparece un mensaje indicando que el comando se completó exitosamente.

Veamos las tablas existentes:

```
 exec sp_tables @table_owner='dbo';
```

Veamos la estructura de la tabla "usuarios":

```
 exec sp_columns usuarios;
```

aparece mucha información que no analizaremos en detalle, como el nombre de la tabla, su propietario, los campos y sus tipos de datos, su longitud, etc.:

```
...COLUMN_NAME  TYPE_NAME      LENGHT...
_____________________________________
   nombre       varchar        30
   clave        varchar        10
```

Intentemos crear una tabla con el mismo nombre, mostrará un mensaje indicando que ya hay un objeto llamado 'usuarios' en la base de datos y la sentencia no se ejecutará:

```
 create table usuarios (
```

```
  nombre varchar(30),
  clave varchar(10)
 );
```

Eliminemos la tabla:

```
 drop table usuarios;
```

Verifiquemos si se ha eliminado:

```
 exec sp_tables @table_owner='dbo';
```

no debe aparecer la tabla "usuarios".

SQL ALTER TABLE

Para agregar una columna de una tabla, utilice la siguiente sintaxis:

```
ALTER TABLE table_name
ADD column_name datatype
```

Para eliminar una columna de una tabla, utilice la siguiente sintaxis (nótese que algunos sistemas de bases de datos no permiten la eliminación de una columna):

```
ALTER TABLE table_name
DROP COLUMN column_name
```

Para cambiar el tipo de datos de una columna de una tabla, utilice la siguiente sintaxis:

SQL Server / MS Access:

```
ALTER TABLE table_name
ALTER COLUMN column_name datatyp
```

4 - Tipos de datos básicos

Ya explicamos que al crear una tabla debemos resolver qué campos (columnas) tendrá y que tipo de datos almacenará cada uno de ellos, es decir, su estructura.

El tipo de dato especifica el tipo de información que puede guardar un campo: caracteres, números, etc.

Estos son algunos tipos de datos básicos de SQL Server (posteriormente veremos otros):

- **varchar**: se usa para almacenar cadenas de caracteres. Una cadena es una secuencia de caracteres. Se coloca entre comillas (simples); ejemplo: 'Hola', 'Juan Perez'. El tipo "varchar" define una cadena de longitud variable en la cual determinamos el máximo de caracteres entre paréntesis. Puede guardar hasta 8000 caracteres. Por ejemplo, para almacenar cadenas de hasta 30 caracteres, definimos un campo de tipo varchar(30), es decir, entre paréntesis, junto al nombre del campo colocamos la longitud.
  Si asignamos una cadena de caracteres de mayor longitud que la definida, la cadena no se carga, aparece un mensaje indicando tal situación y la sentencia no se ejecuta.
  Por ejemplo, si definimos un campo de tipo varchar(10) e intentamos asignarle la cadena 'Buenas tardes', aparece un mensaje de error y la sentencia no se ejecuta.
- **integer**: se usa para guardar valores numéricos enteros, de -2000000000 a 2000000000 aprox. Definimos campos de este tipo cuando queremos representar, por ejemplo, cantidades.
- **float**: se usa para almacenar valores numéricos con decimales. Se utiliza como separador el punto (.). Definimos campos de este tipo para precios, por ejemplo.

Antes de crear una tabla debemos pensar en sus campos y optar por el tipo de dato adecuado para cada uno de ellos.
Por ejemplo, si en un campo almacenaremos números enteros, el tipo "float" sería una mala elección; si vamos a guardar precios, el tipo "float" es más adecuado, no así "integer" que no tiene decimales. Otro ejemplo, si en un campo vamos a guardar un número telefónico o un número de documento, usamos "varchar", no "integer" porque si bien son dígitos, con ellos no realizamos operaciones matemáticas.

Servidor de SQL Server instalado en forma local.

Ingresemos el siguiente lote de comandos en el SQL Server Management Studio:

```
if object_id('libros') is not null
  drop table libros;

 create table libros(
  titulo varchar(80),
  autor varchar(40),
  editorial varchar(30),
  precio float,
```

```
  cantidad integer
 );

 exec sp_columns libros;

 insert into libros (titulo,autor,editorial,precio,
cantidad)
  values ('El aleph','Borges','Emece',25.50, 100);
 insert into libros
(titulo,autor,editorial,precio,cantidad)
  values ('Matematica estas ahi','Paenza','Siglo
XXI',18.8,200);

 select * from libros;

 insert into libros
(titulo,autor,editorial,precio,cantidad)
  values ('Alicia en el pais de las maravillas','Lewis
Carroll','Atlantida',10,200);

 insert into libros
(titulo,autor,editorial,precio,cantidad)
  values ('Alicia en el pais','Lewis
Carroll','Atlantida',10,200);

 select * from libros;
```

Es importante el valor que definimos entre paréntesis luego de la palabra clave varchar. Si definimos el campo titulo de tamaño 80, luego significa que no podemos almacenar un nombre de libro de más de 80 caracteres.

8 - Borrar registros (delete)

Para eliminar los registros de una tabla usamos el comando "delete":

```
 delete from usuarios;
```

Muestra un mensaje indicando la cantidad de registros que ha eliminado.

Si no queremos eliminar todos los registros, sino solamente algunos, debemos indicar cuál o cuáles, para ello utilizamos el comando "delete"

junto con la clausula "where" con la cual establecemos la condición que deben cumplir los registros a borrar.

Por ejemplo, queremos eliminar aquel registro cuyo nombre de usuario es "Marcelo":

```
 delete from usuarios
 where nombre='Marcelo';
```

Si solicitamos el borrado de un registro que no existe, es decir, ningún registro cumple con la condición especificada, ningún registro será eliminado.

Tenga en cuenta que si no colocamos una condición, se eliminan todos los registros de la tabla nombrada.

Servidor de SQL Server instalado en forma local.

Ingresemos el siguiente lote de comandos en el SQL Server Management Studio:

```
if object_id('usuarios') is not null
  drop table usuarios;

create table usuarios(
  nombre varchar(30),
  clave varchar(10)
);

go

insert into usuarios (nombre,clave)
  values ('Marcelo','River');
insert into usuarios (nombre,clave)
  values ('Susana','chapita');
insert into usuarios (nombre,clave)
  values ('CarlosFuentes','Boca');
insert into usuarios (nombre,clave)
  values ('FedericoLopez','Boca');

select * from usuarios;

-- Eliminamos el registro cuyo nombre de usuario es "Marcelo"
delete from usuarios
  where nombre='Marcelo';

select * from usuarios;

-- Intentamos eliminarlo nuevamente (no se borra registro)
delete from usuarios
 where nombre='Marcelo';
```

```
select * from usuarios;

-- Eliminamos todos los registros cuya clave es 'Boca'
delete from usuarios
  where clave='Boca';

select * from usuarios;

-- Eliminemos todos los registros
delete from usuarios;

select * from usuarios;
```

9 - Actualizar registros (update)

Ver video

Decimos que actualizamos un registro cuando modificamos alguno de sus valores.

Para modificar uno o varios datos de uno o varios registros utilizamos "update" (actualizar).

Por ejemplo, en nuestra tabla "usuarios", queremos cambiar los valores de todas las claves, por "RealMadrid":

```
 update usuarios set clave='RealMadrid';
```

Utilizamos "update" junto al nombre de la tabla y "set" junto con el campo a modificar y su nuevo valor.

El cambio afectará a todos los registros.

Podemos modificar algunos registros, para ello debemos establecer condiciones de selección con "where".
Por ejemplo, queremos cambiar el valor correspondiente a la clave de nuestro usuario llamado "Federicolopez", queremos como nueva clave "Boca", necesitamos una condición "where" que afecte solamente a este registro:

```
 update usuarios set clave='Boca'
  where nombre='Federicolopez';
```

Si Microsoft SQL Server no encuentra registros que cumplan con la condición del "where", no se modifica ninguno.

Las condiciones no son obligatorias, pero si omitimos la cláusula "where", la actualización afectará a todos los registros.

También podemos actualizar varios campos en una sola instrucción:

```
 update usuarios set nombre='Marceloduarte', clave='Marce'
  where nombre='Marcelo';
```

Para ello colocamos "update", el nombre de la tabla, "set" junto al nombre del campo y el nuevo valor y separado por coma, el otro nombre del campo con su nuevo valor.

Servidor de SQL Server instalado en forma local.

Ingresemos el siguiente lote de comandos en el SQL Server Management Studio:

```
if object_id('usuarios') is not null
  drop table usuarios;

create table usuarios(
  nombre varchar(20),
  clave varchar(10)
);

go

insert into usuarios (nombre,clave)
  values ('Marcelo','River');
insert into usuarios (nombre,clave)
  values ('Susana','chapita');
insert into usuarios (nombre,clave)
  values ('Carlosfuentes','Boca');
insert into usuarios (nombre,clave)
  values ('Federicolopez','Boca');

update usuarios set clave='RealMadrid';

select * from usuarios;

update usuarios set clave='Boca'
  where nombre='Federicolopez';

select * from usuarios;

update usuarios set clave='payaso'
  where nombre='JuanaJuarez';

select * from usuarios;

update usuarios set nombre='Marceloduarte', clave='Marce'
  where nombre='Marcelo';
```

```
select * from usuarios;
```

Listado completo de tutoriales

15 - Truncate table

Ver video

Aprendimos que para borrar todos los registro de una tabla se usa "delete" sin condición "where".
También podemos eliminar todos los registros de una tabla con "truncate table".
Por ejemplo, queremos vaciar la tabla "libros", usamos:

```
truncate table libros;
```

La sentencia "truncate table" vacía la tabla (elimina todos los registros) y conserva la estructura de la tabla.

La diferencia con "drop table" es que esta sentencia borra la tabla, "truncate table" la vacía.

La diferencia con "delete" es la velocidad, es más rápido "truncate table" que "delete" (se nota cuando la cantidad de registros es muy grande) ya que éste borra los registros uno a uno.

Otra diferencia es la siguiente: cuando la tabla tiene un campo "identity", si borramos todos los registros con "delete" y luego ingresamos un registro, al cargarse el valor en el campo de identidad, continúa con la secuencia teniendo en cuenta el valor mayor que se había guardado; si usamos "truncate table" para borrar todos los registros, al ingresar otra vez un registro, la secuencia del campo de identidad vuelve a iniciarse en 1.

Por ejemplo, tenemos la tabla "libros" con el campo "codigo" definido "identity", y el valor más alto de ese campo es "2", si borramos todos los registros con "delete" y luego ingresamos un registro, éste guardará el valor de código "3"; si en cambio, vaciamos la tabla con "truncate table", al ingresar un nuevo registro el valor del código se iniciará en 1 nuevamente.

Servidor de SQL Server instalado en forma local.

Ingresemos el siguiente lote de comandos en el SQL Server Management Studio:

```
if object_id('libros') is not null
  drop table libros;

create table libros(
  codigo int identity,
  titulo varchar(30),
  autor varchar(20),
  editorial varchar(15),
  precio float
);

go

insert into libros (titulo,autor,editorial,precio)
  values ('El aleph','Borges','Emece',25.60);
insert into libros (titulo,autor,editorial,precio)
  values ('Uno','Richard Bach','Planeta',18);

select * from libros;

-- Truncamos la tabla:
truncate table libros;

-- Ingresamos nuevamente algunos registros:
insert into libros (titulo,autor,editorial,precio)
  values ('El aleph','Borges','Emece',25.60);
insert into libros (titulo,autor,editorial,precio)
  values ('Uno','Richard Bach','Planeta',18);

-- Si seleccionamos todos los registros vemos que la secuencia se
reinició en 1:
select * from libros;

-- Eliminemos todos los registros con "delete":
delete from libros;

-- Ingresamos nuevamente algunos registros:
insert into libros (titulo,autor,editorial,precio)
  values ('El aleph','Borges','Emece',25.60);
insert into libros (titulo,autor,editorial,precio)
  values ('Uno','Richard Bach','Planeta',18);

-- Seleccionamos todos los registros y vemos que la secuencia continuó:
select * from libros;
```

6 - Otros tipos de datos en SQL Server

Ya explicamos que al crear una tabla debemos elegir la estructura adecuada, esto es, definir los campos y sus tipos más precisos, según el caso.

El tipo de dato especificado en la definición de cada campo indica los valores permitidos para cada uno de ellos.

Hasta ahora hemos visto 3 tipos de datos: varchar, integer y float. Hay más tipos, incluso, subtipos.

Los valores que podemos guardar son:

1. **TEXTO:** Para almacenar texto usamos cadenas de caracteres.
   Las cadenas se colocan entre comillas simples. Podemos almacenar letras, símbolos y dígitos con los que no se realizan operaciones matemáticas, por ejemplo, códigos de identificación, números de documentos, números telefónicos.
   SQL Server ofrece los siguientes tipos: char, nchar, varchar, nvarchar, text y ntext.
2. **NUMEROS:** Existe variedad de tipos numéricos para representar enteros, decimales, monedas.
   Para almacenar valores enteros, por ejemplo, en campos que hacen referencia a cantidades, precios, etc., usamos el tipo integer (y sus subtipos: tinyint, smallint y bigint).
   Para almacenar valores con decimales exactos, utilizamos: numeric o decimal (son equivalentes).
   Para guardar valores decimales aproximados: float y real. Para almacenar valores monetarios: money y smallmoney.
3. **FECHAS y HORAS:** para guardar fechas y horas SQL Server dispone de 2 tipos: datetime y smalldatetime.

Existen otros tipos de datos que analizaremos en secciones próximas.

Entonces, cuando creamos una tabla y definir sus campos debemos elegir el tipo de dato más preciso. Por ejemplo, si necesitamos almacenar nombres usamos texto; si un campo numérico almacenará solamente valores enteros el tipo "integer" es más adecuado que, por ejemplo un "float"; si necesitamos almacenar precios, lo más lógico es utilizar el tipo "money".

Listado completo de tutoriales

17 - Tipo de dato (texto)

Ver video

Ya explicamos que al crear una tabla debemos elegir la estructura adecuada, esto es, definir los campos y sus tipos más precisos, según el caso.

Para almacenar TEXTO usamos cadenas de caracteres.
Las cadenas se colocan entre comillas simples.
Podemos almacenar letras, símbolos y dígitos con los que no se realizan operaciones matemáticas, por ejemplo, códigos de identificación, números de documentos, números telefónicos.

Tenemos los siguientes tipos:

1. **varchar(x):** define una cadena de caracteres de longitud variable en la cual determinamos el máximo de caracteres con el argumento "x" que va entre paréntesis.
   Si se omite el argumento coloca 1 por defecto. Su rango va de 1 a 8000 caracteres.
2. **char(x):** define una cadena de longitud fija determinada por el argumento "x". Si se omite el argumento coloca 1 por defecto. Su rango es de 1 a 8000 caracteres.
   Si la longitud es invariable, es conveniente utilizar el tipo char; caso contrario, el tipo varchar.
   Ocupa tantos bytes como se definen con el argumento "x".
   "char" viene de character, que significa caracter en inglés.
3. **text:** guarda datos binarios de longitud variable, puede contener hasta 2000000000 caracteres. No admite argumento para especificar su longitud.
4. **nvarchar(x):** es similar a "varchar", excepto que permite almacenar caracteres Unicode, su rango va de 0 a 4000 caracteres porque se emplean 2 bytes por cada caracter.
5. **nchar(x):** es similar a "char" excpeto que acepta caracteres Unicode, su rango va de 0 a 4000 caracteres porque se emplean 2 bytes por cada caracter.
6. **ntext:** es similar a "text" excepto que permite almacenar caracteres Unicode, puede contener hasta 1000000000 caracteres. No admite argumento para especificar su longitud.

En general se usarán los 3 primeros.

Si intentamos almacenar en un campo una cadena de caracteres de mayor longitud que la definida, aparece un mensaje indicando tal situación y la sentencia no se ejecuta.

Por ejemplo, si definimos un campo de tipo varchar(10) y le asignamos la cadena 'Aprenda PHP' (11 caracteres), aparece un mensaje y la sentencia no se ejecuta.

Si ingresamos un valor numérico (omitiendo las comillas), lo convierte a cadena y lo ingresa como tal.

Por ejemplo, si en un campo definido como varchar(5) ingresamos el valor 12345, lo toma como si hubiésemos tipeado '12345', igualmente, si ingresamos el valor 23.56, lo convierte a '23.56'. Si el valor numérico, al ser convertido a cadena supera la longitud definida, aparece un mensaje de error y la sentencia no se ejecuta.

Es importante elegir el tipo de dato adecuado según el caso, el más preciso.

Para almacenar cadenas que varían en su longitud, es decir, no todos los registros tendrán la misma longitud en un campo determinado, se emplea "varchar" en lugar de "char".

Por ejemplo, en campos que guardamos nombres y apellidos, no todos los nombres y apellidos tienen la misma longitud.

Para almacenar cadenas que no varían en su longitud, es decir, todos los registros tendrán la misma longitud en un campo determinado, se emplea "char".

Por ejemplo, definimos un campo "codigo" que constará de 5 caracteres, todos los registros tendrán un código de 5 caracteres, ni más ni menos.

Para almacenar valores superiores a 8000 caracteres se debe emplear "text".

```
Tipo            Bytes de almacenamiento
_____________________________________
varchar(x)      0 a 8K
char(x)         0 a 8K
text            0 a 2GB

nvarchar(x)     0 a 8K
nchar(x)        0 a 8K
ntext           0 a 2GB
```

Servidor de SQL Server instalado en forma local.

Ingresemos el siguiente lote de comandos en el SQL Server Management Studio:

```
if object_id('visitantes') is not null
  drop table visitantes;

/* Un comercio que tiene un stand en una feria registra en una tabla
llamada "visitantes"
   algunos datos de las personas que visitan o compran en su stand para
luego enviarle
   publicidad de sus productos. */
create table visitantes(
  nombre varchar(30),
  edad integer,
  sexo char(1),
  domicilio varchar(30),
  ciudad varchar(20),
  telefono varchar(11)
);

go

-- Intentamos ingresar una cadena de mayor longitud que la definida
-- en el campo sexo (se genera un error):
insert into visitantes (nombre,edad,sexo,domicilio,ciudad,telefono)
  values ('Juan Juarez',32,'masc','Avellaneda 789','Cordoba','4234567');

-- Ingresamos un número telefónico olvidando las comillas, es decir,
-- como un valor numérico (lo transforma a cadena):
insert into visitantes (nombre,edad,sexo,domicilio,ciudad,telefono)
  values ('Marcela Morales',43,'f','Colon 456','Cordoba',4567890);

select * from visitantes;
```

Listado completo de tutoriales

18 - Tipo de dato (numérico)

Ver video

Ya explicamos que al crear una tabla debemos elegir la estructura adecuada, esto es, definir los campos y sus tipos más precisos, según el caso.

Para almacenar valores NUMERICOS SQL Server dispone de varios tipos.

Para almacenar valores ENTEROS, por ejemplo, en campos que hacen referencia a cantidades, usamos:

1) integer o int: su rango es de -2000000000 a 2000000000 aprox. El tipo "integer" tiene subtipos:
- smallint: Puede contener hasta 5 digitos. Su rango va desde –32000 hasta

32000 aprox.
- tinyint: Puede almacenar valores entre 0 y 255.
- bigint: De –9000000000000000000 hasta 9000000000000000000 aprox.

Para almacenar valores numéricos EXACTOS con decimales, especificando la cantidad de cifras a la izquierda y derecha del separador decimal, utilizamos:

2) decimal o numeric (t,d): Pueden tener hasta 38 digitos, guarda un valor exacto. El primer argumento indica el total de dígitos y el segundo, la cantidad de decimales.
Por ejemplo, si queremos almacenar valores entre -99.99 y 99.99 debemos definir el campo como tipo "decimal(4,2)". Si no se indica el valor del segundo argumento, por defecto es "0". Por ejemplo, si definimos "decimal(4)" se pueden guardar valores entre -9999 y 9999.

El rango depende de los argumentos, también los bytes que ocupa.
Se utiliza el punto como separador de decimales.

Si ingresamos un valor con más decimales que los permitidos, redondea al más cercano; por ejemplo, si definimos "decimal(4,2)" e ingresamos el valor "12.686", guardará "12.69", redondeando hacia arriba; si ingresamos el valor "12.682", guardará "12.67", redondeando hacia abajo.

Para almacenar valores numéricos APROXIMADOS con decimales utilizamos:

3) float y real: De 1.79E+308 hasta 1.79E+38. Guarda valores aproximados.
4) real: Desde 3.40E+308 hasta 3.40E+38. Guarda valores aproximados.

Para almacenar valores MONETARIOS empleamos:

5) money: Puede tener hasta 19 digitos y sólo 4 de ellos puede ir luego del separador decimal; entre –900000000000000.5808 aprox y 900000000000000.5807.

6) smallmoney: Entre –200000.3648 y 200000.3647 aprox.

Para todos los tipos numéricos:
- si intentamos ingresar un valor fuera de rango, no lo permite.
- si ingresamos una cadena, SQL Server intenta convertirla a valor numérico, si dicha cadena consta solamente de dígitos, la conversión se realiza, luego verifica si está dentro del rango, si es así, la ingresa, sino, muestra un mensaje de error y no ejecuta la sentencia. Si la cadena

contiene caracteres que SQL Server no puede convertir a valor numérico, muestra un mensaje de error y la sentencia no se ejecuta.
Por ejemplo, definimos un campo de tipo decimal(5,2), si ingresamos la cadena '12.22', la convierte al valor numérico 12.22 y la ingresa; si intentamos ingresar la cadena '1234.56', la convierte al valor numérico 1234.56, pero como el máximo valor permitido es 999.99, muestra un mensaje indicando que está fuera de rango. Si intentamos ingresar el valor '12y.25', SQL Server no puede realizar la conversión y muestra un mensaje de error.

Es importante elegir el tipo de dato adecuado según el caso, el más preciso. Por ejemplo, si un campo numérico almacenará valores positivos menores a 255, el tipo "int" no es el más adecuado, conviene el tipo "tinyint", de esta manera usamos el menor espacio de almacenamiento posible.
Si vamos a guardar valores monetarios menores a 200000 conviene emplear "smallmoney" en lugar de "money".

```
Tipo            Bytes de almacenamiento
_____________________________________
int             4
smallint        2
tinyint         1
bigint          8

decimal         2 a 17

float           4 u 8
real            4 u 8

money           8
smallmoney      4
```

Servidor de SQL Server instalado en forma local.

Ingresemos el siguiente lote de comandos en el SQL Server Management Studio:

```
if object_id('libros') is not null
  drop table libros;

create table libros(
  codigo smallint identity,
  titulo varchar(40) not null,
  autor varchar(30),
  editorial varchar(15),
  precio smallmoney,
  cantidad tinyint
);

go
```

```
-- Intentemos ingresar un valor fuera del rango definido, una cantidad
-- que supera el rango del tipo "tinyint", el valor 260 (genera error):
insert into libros (titulo,autor,editorial,precio,cantidad)
  values('El aleph','Borges','Emece',25.60,260);

-- Intentamos ingresar un precio que supera el rango del tipo
"smallmoney",
-- el valor 250000 (genera error):
insert into libros (titulo,autor,editorial,precio,cantidad)
  values('El aleph','Borges','Emece',250000,100);

-- Intentamos ingresar una cadena que SQL Server no pueda convertir a
valor
-- numérico en el campo "precio" (genera error):
insert into libros (titulo,autor,editorial,precio,cantidad)
  values('Uno','Richard Bach','Planeta','a50.30',100);

-- Ingresamos una cadena en el campo "cantidad" (lo convierte a valor
numérico):
insert into libros (titulo,autor,editorial,precio,cantidad)
  values('Uno','Richard Bach','Planeta',50.30,'100');

select * from libros;
```

Listado completo de tutoriales

19 - Tipo de dato (fecha y hora)

Ver video

Ya explicamos que al crear una tabla debemos elegir la estructura adecuada, esto es, definir los campos y sus tipos más precisos, según el caso.

Para almacenar valores de tipo FECHA Y HORA SQL Server dispone de dos tipos:

1) datetime: puede almacenar valores desde 01 de enero de 1753 hasta 31 de diciembre de 9999.

2) smalldatetime: el rango va de 01 de enero de 1900 hasta 06 de junio de 2079.

Las fechas se ingresan entre comillas simples.

Para almacenar valores de tipo fecha se permiten como separadores "/", "-" y ".".

SQL Server reconoce varios formatos de entrada de datos de tipo fecha. Para establecer el orden de las partes de una fecha (dia, mes y año) empleamos "set dateformat". Estos son los formatos:

```
-mdy: 4/15/96 (mes y día con 1 ó 2 dígitos y año con 2 ó 4 dígitos),
-myd: 4/96/15,
-dmy: 15/4/1996
-dym: 15/96/4,
-ydm: 96/15/4,
-ydm: 1996/15/4,
```

Para ingresar una fecha con formato "día-mes-año", tipeamos:

```
 set dateformat dmy;
```

El formato por defecto es "mdy".

Todos los valores de tipo "datetime" se muestran en formato "año-mes-día hora:minuto:segundo .milisegundos", independientemente del formato de ingreso que hayamos seteado.

Podemos ingresar una fecha, sin hora, en tal caso la hora se guarda como "00:00:00". Por ejemplo, si ingresamos '25-12-01' (año de 2 dígitos), lo mostrará así: '2001-12-25 00:00:00.000'.

Podemos ingresar una hora sin fecha, en tal caso, coloca la fecha "1900-01-01". Por ejemplo, si ingresamos '10:15', mostrará '1900-01-01 10:15.000'.

Podemos emplear los operadores relacionales vistos para comparar fechas.

```
Tipo           Bytes de almacenamiento
_____________________________________
datetime       8
smalldatetime  4
```

Servidor de SQL Server instalado en forma local.

Ingresemos el siguiente lote de comandos en el SQL Server Management Studio:

```
if object_id('empleados') is not null
  drop table empleados;

create table empleados(
  nombre varchar(20),
  documento char(8),
  fechaingreso datetime
);

go
```

```
-- Seteamos el formato de la fecha para que guarde día, mes y año:
set dateformat dmy;

insert into empleados values('Ana Gomez','22222222','12-01-1980');
insert into empleados values('Bernardo Huerta','23333333','15-03-81');
insert into empleados values('Carla Juarez','24444444','20/05/1983');
insert into empleados values('Daniel Lopez','25555555','2.5.1990');

-- Note que el formato de visualización es "y-m-d".
select * from empleados;

-- Mostramos los datos de los empleados cuya fecha de ingreso es anterior
a '01-01-1985':
select * from empleados where fechaingreso<'01-01-1985';

-- Actualizamos el nombre a "Maria Carla Juarez'
-- del empleado cuya fecha de ingreso es igual a '20/05/1983':
update empleados set nombre='Maria Carla Juarez' where
fechaingreso='20.5.83';

select * from empleados;

-- Borramos los empleados cuya fecha de ingreso es distinta a '20.5.83':
delete from empleados where fechaingreso<>'20/05/1983';

select * from empleados;
```

20 - Ingresar algunos campos (insert into)

Problema:

Trabajamos con la tabla "libros" que almacena los datos de los libros de una librería.
Eliminamos la tabla, si existe:

```
 if object_id('libros') is not null
  drop table libros;
```

Creamos la tabla:

```
 create table libros(
  codigo int identity,
  titulo varchar(40) not null,
  autor varchar(30),
  editorial varchar(15)
 );
```

Si ingresamos valores para todos los campos, podemos omitir la lista de campos:

```
 insert into libros
```

```
 values ('Uno','Richard Bach','Planeta');
```

Podemos ingresar valores para algunos de los campos:

```
insert into libros (titulo, autor)
 values ('El aleph','Borges');
```

No podemos omitir el valor para un campo declarado "not null", como el campo "titulo":

```
insert into libros (autor,editorial)
 values ('Lewis Carroll','Planeta');
```

aparece un mensaje y la inserción no se realiza.

Veamos cómo SQL Server almacenó los registros:

```
select * from libros;
```

Listado completo de tutoriales

25 - Funciones para el manejo de cadenas

Ver video

Microsoft SQL Server tiene algunas funciones para trabajar con cadenas de caracteres. Estas son algunas:

- **substring**(cadena,inicio,longitud): devuelve una parte de la cadena especificada como primer argumento, empezando desde la posición especificada por el segundo argumento y de tantos caracteres de longitud como indica el tercer argumento. Ejemplo:

```
select substring('Buenas tardes',8,6);
```

retorna "tardes".

- **str**(numero,longitud,cantidaddecimales): convierte números a caracteres; el primer parámetro indica el valor numérico a convertir, el segundo la longitud del resultado (debe ser mayor o igual a la parte entera del número más el signo si lo tuviese) y el tercero, la cantidad de decimales. El segundo y tercer argumento son opcionales y deben ser positivos. String significa cadena en inglés.

Ejemplo: se convierte el valor numérico "123.456" a cadena, especificando 7 de longitud y 3 decimales:

```
select str(123.456,7,3);

select str(-123.456,7,3);
```

retorna '-123.46';

Si no se colocan el segundo y tercer argumeno, la longitud predeterminada es 10 y la cantidad de decimales 0 y se redondea a entero. Ejemplo: se convierte el valor numérico "123.456" a cadena:

```
select str(123.456);
```

retorna '123';

```
select str(123.456,3);
```

retorna '123';

Si el segundo parámetro es menor a la parte entera del número, devuelve asteriscos (*). Ejemplo: select str(123.456,2,3);

retorna "**".

- **stuff**(cadena1,inicio,cantidad,cadena2): inserta la cadena enviada como cuarto argumento, en la posición indicada en el segundo argumento, reemplazando la cantidad de caracteres indicada por el tercer argumento en la cadena que es primer parámetro. Stuff significa rellenar en inglés. Ejemplo:

```
select stuff('abcde',3,2,'opqrs');
```

retorna "abopqrse". Es decir, coloca en la posición 2 la cadena "opqrs" y reemplaza 2 caracteres de la primer cadena.

Los argumentos numéricos deben ser positivos y menor o igual a la longitud de la primera cadena, caso contrario, retorna "null".

Si el tercer argumento es mayor que la primera cadena, se elimina hasta el primer carácter.

- **len**(cadena): retorna la longitud de la cadena enviada como argumento. "len" viene de length, que significa longitud en inglés. Ejemplo:

```
select len('Hola');
```

devuelve 4.

- **char**(x): retorna un caracter en código ASCII del entero enviado como argumento. Ejemplo:

```
select char(65);
```

retorna "A".

- **left**(cadena,longitud): retorna la cantidad (longitud) de caracteres de la cadena comenzando desde la izquierda, primer caracter. Ejemplo:

```
select left('buenos dias',8);
```

retorna "buenos d".

- **right**(cadena,longitud): retorna la cantidad (longitud) de caracteres de la cadena comenzando desde la derecha, último caracter. Ejemplo:

```
select right('buenos dias',8);
```

retorna "nos dias".

-**lower**(cadena): retornan la cadena con todos los caracteres en minúsculas. lower significa reducir en inglés. Ejemplo:

```
select lower('HOLA ESTUDIAnte');
```

retorna "hola estudiante".

-**upper**(cadena): retornan la cadena con todos los caracteres en mayúsculas. Ejemplo:

```
select upper('HOLA ESTUDIAnte');
```

-**ltrim**(cadena): retorna la cadena con los espacios de la izquierda eliminados. Trim significa recortar. Ejemplo:

```
select ltrim('     Hola     ');
```

retorna "Hola ".

- **rtrim**(cadena): retorna la cadena con los espacios de la derecha eliminados. Ejemplo:

```
select rtrim('   Hola   ');
```

retorna " Hola".

- **replace**(cadena,cadenareemplazo,cadenareemplazar): retorna la cadena con todas las ocurrencias de la subcadena reemplazo por la subcadena a reemplazar. Ejemplo:

```
select replace('xxx.sqlserverya.com','x','w');
```

retorna "www.sqlserverya.com'.

- **reverse**(cadena): devuelve la cadena invirtiendo el order de los caracteres. Ejemplo:

```
select reverse('Hola');
```

retorna "aloH".

- **patindex**(patron,cadena): devuelve la posición de comienzo (de la primera ocurrencia) del patrón especificado en la cadena enviada como segundo argumento. Si no la encuentra retorna 0. Ejemplos:

```
select patindex('%Luis%', 'Jorge Luis Borges');
```

retorna 7.

```
select patindex('%or%', 'Jorge Luis Borges');
```

retorna 2.

```
select patindex('%ar%', 'Jorge Luis Borges');
```

retorna 0.

- **charindex**(subcadena,cadena,inicio): devuelve la posición donde comienza la subcadena en la cadena, comenzando la búsqueda desde la posición indicada por "inicio". Si el tercer argumento no se coloca, la búsqueda se inicia desde 0. Si no la encuentra, retorna 0. Ejemplos:

```
select charindex('or','Jorge Luis Borges',5);
```

retorna 13.

```
 select charindex('or','Jorge Luis Borges');
```

retorna 2.

```
 select charindex('or','Jorge Luis Borges',14);
```

retorna 0.

```
 select charindex('or', 'Jorge Luis Borges');
```

retorna 0.

- **replicate**(cadena,cantidad): repite una cadena la cantidad de veces especificada. Ejemplo:

```
 select replicate ('Hola',3);
```

retorna "HolaHolaHola";

- **space**(cantidad): retorna una cadena de espacios de longitud indicada por "cantidad", que debe ser un valor positivo. Ejemplo:

```
 select 'Hola'+space(1)+'que tal';
```

retorna "Hola que tal".

Se pueden emplear estas funciones enviando como argumento el nombre de un campo de tipo caracter.

Servidor de SQL Server instalado en forma local.

Ingresemos el siguiente lote de comandos en el SQL Server Management Studio:

```
if object_id ('libros') is not null
  drop table libros;

create table libros(
  codigo int identity,
  titulo varchar(40) not null,
  autor varchar(20) default 'Desconocido',
  editorial varchar(20),
  precio decimal(6,2),
  cantidad tinyint default 0,
  primary key (codigo)
);

go
```

```
insert into libros (titulo,autor,editorial,precio)
  values('El aleph','Borges','Emece',25);
insert into libros
  values('Java en 10 minutos','Mario Molina','Siglo XXI',50.40,100);
insert into libros (titulo,autor,editorial,precio,cantidad)
  values('Alicia en el pais de las maravillas','Lewis
Carroll','Emece',15,50);

-- Mostramos sólo los 12 primeros caracteres de los títulos de los libros
y
-- sus autores, empleando la función "substring()":
select substring(titulo,1,12) as titulo
  from libros;

-- Mostramos sólo los 12 primeros caracteres de los títulos de los libros
y
-- sus autores, ahora empleando la función "left()":
select left(titulo,12) as titulo
  from libros;

-- Mostramos los títulos de los libros y sus precios convirtiendo este
último a cadena
-- de caracteres con un solo decimal, empleando la función "str":
select titulo,
  str(precio,6,1)
  from libros;

-- Mostramos los títulos de los libros y sus precios convirtiendo este
último a cadena
-- de caracteres especificando un solo argumento:
select titulo,
  str(precio)
  from libros;

-- Mostramos los títulos, autores y editoriales de todos libros, al
último
-- campo lo queremos en mayúsculas:
select titulo, autor, upper(editorial)
   from libros;
```