# Crear una tabla (create table - sp\_tables - sp\_columns - drop table)

Una base de datos almacena su información en tablas.

Una tabla es una estructura de datos que organiza los datos en columnas y filas; cada columna es un campo (o atributo) y cada fila, un registro. La intersección de una columna con una fila, contiene un dato específico, un solo valor.

Cada registro contiene un dato por cada columna de la tabla.

Cada campo (columna) debe tener un nombre. El nombre del campo hace referencia a la información que almacenará.

Cada campo (columna) también debe definir el tipo de dato que almacenará.

Las tablas forman parte de una base de datos.

Nosotros trabajaremos con la base de datos llamada wi520641\_sqlserverya (este nombre se debe a que las empresas de hosting es la que lo define), que ya he creado en el servidor sqlserverya.com.ar.

Para ver las tablas existentes creadas por los usuarios en una base de datos usamos el procedimiento almacenado "sp\_tables @table\_owner='dbo';":

sp\_tables @table\_owner='dbo';

El parámetro @table\_owner='dbo' indica que solo muestre las tablas de usuarios y no las que crea el SQL Server para administración interna.

Finalizamos cada comando con un punto y coma.

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

La sintaxis básica y general para crear una tabla es la siguiente:

create table NOMBRETABLA(

NOMBRECAMPO1 TIPODEDATO,

...

NOMBRECAMPON TIPODEDATO

);

La tabla debe ser definida con un nombre que la identifique y con el cual accederemos a ella.

Creamos una tabla llamada "usuarios" y entre paréntesis definimos los campos y sus tipos:

create table usuarios (

nombre varchar(30),

clave varchar(10)

);

Cada campo con su tipo debe separarse con comas de los siguientes, excepto el último.

Cuando se crea una tabla debemos indicar su nombre y definir al menos un campo con su tipo de dato. En esta tabla "usuarios" definimos 2 campos:

* nombre: que contendrá una cadena de caracteres de 30 caracteres de longitud, que almacenará el nombre de usuario y
* clave: otra cadena de caracteres de 10 de longitud, que guardará la clave de cada usuario.

Cada usuario ocupará un registro de esta tabla, con su respectivo nombre y clave.

Para nombres de tablas, se puede utilizar cualquier caracter permitido para nombres de directorios, el primero debe ser un caracter alfabético y no puede contener espacios. La longitud máxima es de 128 caracteres.

Si intentamos crear una tabla con un nombre ya existente (existe otra tabla con ese nombre), mostrará un mensaje indicando que ya hay un objeto llamado 'usuarios' en la base de datos y la sentencia no se ejecutará. Esto es muy importante ya que cuando haga los ejercicios en este sitio puede haber otra persona que haya creado una tabla con el nombre que usted especifique.

Para ver la estructura de una tabla usamos el procedimiento almacenado "sp\_columns" junto al nombre de la tabla:

sp\_columns usuarios;

aparece mucha información que no analizaremos en detalle, como el nombre de la tabla, su propietario, los campos, el tipo de dato de cada campo, su longitud, etc.:

...COLUMN\_NAME TYPE\_NAME LENGHT

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

nombre varchar 30

clave varchar 10

Para eliminar una tabla usamos "drop table" junto al nombre de la tabla a eliminar:

drop table usuarios;

Si intentamos eliminar una tabla que no existe, aparece un mensaje de error indicando tal situación y la sentencia no se ejecuta. Para evitar este mensaje podemos agregar a la instrucción lo siguiente:

if object\_id('usuarios') is not null

drop table usuarios;

En la sentencia precedente especificamos que elimine la tabla "usuarios" si existe.

# Insertar y recuperar registros de una tabla (insert into - select)

Un registro es una fila de la tabla que contiene los datos propiamente dichos. Cada registro tiene un dato por cada columna (campo). Nuestra tabla "usuarios" consta de 2 campos, "nombre" y "clave".

Al ingresar los datos de cada registro debe tenerse en cuenta la cantidad y el orden de los campos.

La sintaxis básica y general es la siguiente:

insert into NOMBRETABLA (NOMBRECAMPO1, ..., NOMBRECAMPOn)

values (VALORCAMPO1, ..., VALORCAMPOn);

Usamos "insert into", luego el nombre de la tabla, detallamos los nombres de los campos entre paréntesis y separados por comas y luego de la cláusula "values" colocamos los valores para cada campo, también entre paréntesis y separados por comas.

Para agregar un registro a la tabla tipeamos:

insert into usuarios (nombre, clave) values ('Mariano','payaso');

Note que los datos ingresados, como corresponden a cadenas de caracteres se colocan entre comillas simples.

Para ver los registros de una tabla usamos "select":

select \* from usuarios;

El comando "select" recupera los registros de una tabla.

Con el asterisco indicamos que muestre todos los campos de la tabla "usuarios".

Es importante ingresar los valores en el mismo orden en que se nombran los campos:

insert into usuarios (clave, nombre) values ('River','Juan');

En el ejemplo anterior se nombra primero el campo "clave" y luego el campo "nombre" por eso, los valores también se colocan en ese orden.

Si ingresamos los datos en un orden distinto al orden en que se nombraron los campos, no aparece un mensaje de error y los datos se guardan de modo incorrecto.

En el siguiente ejemplo se colocan los valores en distinto orden en que se nombran los campos, el valor de la clave (la cadena "Boca") se guardará en el campo "nombre" y el valor del nombre (la cadena "Luis") en el campo "clave":

insert into usuarios (nombre,clave) values ('Boca','Luis');

# 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.

# Recuperar algunos campos (select)

Hemos aprendido cómo ver todos los registros de una tabla, empleando la instrucción "select".

La sintaxis básica y general es la siguiente:

select \* from NOMBRETABLA;

El asterisco (\*) indica que se seleccionan todos los campos de la tabla.

Podemos especificar el nombre de los campos que queremos ver separándolos por comas:

select titulo,autor from libros;

La lista de campos luego del "select" selecciona los datos correspondientes a los campos nombrados. En el ejemplo anterior seleccionamos los campos "titulo" y "autor" de la tabla "libros", mostrando todos los registros. Los datos aparecen ordenados según la lista de selección, en dicha lista los nombres de los campos se separan con comas.

# Recuperar algunos registros (where)

Hemos aprendido a seleccionar algunos campos de una tabla.

También es posible recuperar algunos registros.

Existe una cláusula, "where" con la cual podemos especificar condiciones para una consulta "select". Es decir, podemos recuperar algunos registros, sólo los que cumplan con ciertas condiciones indicadas con la cláusula "where". Por ejemplo, queremos ver el usuario cuyo nombre es "Marcelo", para ello utilizamos "where" y luego de ella, la condición:

select nombre, clave

from usuarios

where nombre='Marcelo';

La sintaxis básica y general es la siguiente:

select NOMBRECAMPO1, ..., NOMBRECAMPOn

from NOMBRETABLA

where CONDICION;

Para las condiciones se utilizan operadores relacionales (tema que trataremos más adelante en detalle). El signo igual(=) es un operador relacional. Para la siguiente selección de registros especificamos una condición que solicita los usuarios cuya clave es igual a "River":

select nombre,clave

from usuarios

where clave='River';

Si ningún registro cumple la condición establecida con el "where", no aparecerá ningún registro.

Entonces, con "where" establecemos condiciones para recuperar algunos registros.

Para recuperar algunos campos de algunos registros combinamos en la consulta la lista de campos y la cláusula "where":

select nombre

from usuarios

where clave='River';

En la consulta anterior solicitamos el nombre de todos los usuarios cuya clave sea igual a "River".

# Operadores relacionales

Los operadores son símbolos que permiten realizar operaciones matemáticas, concatenar cadenas, hacer comparaciones.

SQL Server tiene 4 tipos de operadores:

* relacionales (o de comparación)
* aritméticos
* de concatenación
* lógicos.

Por ahora veremos solamente los primeros.

Los operadores relacionales (o de comparación) nos permiten comparar dos expresiones, que pueden ser variables, valores de campos, etc.

Hemos aprendido a especificar condiciones de igualdad para seleccionar registros de una tabla; por ejemplo:

select \*from libros

where autor='Borges';

Utilizamos el operador relacional de igualdad.

Los operadores relacionales vinculan un campo con un valor para que SQL Server compare cada registro (el campo especificado) con el valor dado.

Los operadores relacionales son los siguientes:

* = igual
* <> distinto
* > mayor
* < menor
* >= mayor o igual
* <= menor o igual

Podemos seleccionar los registros cuyo autor sea diferente de "Borges", para ello usamos la condición:

select \* from libros

where autor<>'Borges';

Podemos comparar valores numéricos. Por ejemplo, queremos mostrar los títulos y precios de los libros cuyo precio sea mayor a 20 pesos:

select titulo, precio

from libros

where precio>20;

Queremos seleccionar los libros cuyo precio sea menor o igual a 30:

select \*from libros

where precio<=30;

Los operadores relacionales comparan valores del mismo tipo. Se emplean para comprobar si un campo cumple con una condición.

# 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 cláusula "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.

# Actualizar registros (update)

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.

# Comentarios

Para aclarar algunas instrucciones, en ocasiones, necesitamos agregar comentarios.

Es posible ingresar comentarios en la línea de comandos, es decir, un texto que no se ejecuta; para ello se emplean dos guiones (--) al comienzo de la línea:

select \* from libros --mostramos los registros de libros;

en la línea anterior, todo lo que está luego de los guiones (hacia la derecha) no se ejecuta.

Para agregar varias líneas de comentarios, se coloca una barra seguida de un asterisco (/\*) al comienzo del bloque de comentario y al finalizarlo, un asterisco seguido de una barra (\*/).

select titulo, autor

/\*mostramos títulos y

nombres de los autores\*/

from libros;

todo lo que está entre los símbolos "/\*" y "\*/" no se ejecuta.

# Valores null (is null)

"null" significa "dato desconocido" o "valor inexistente". No es lo mismo que un valor "0", una cadena vacía o una cadena literal "null".

A veces, puede desconocerse o no existir el dato correspondiente a algún campo de un registro. En estos casos decimos que el campo puede contener valores nulos.

Por ejemplo, en nuestra tabla de libros, podemos tener valores nulos en el campo "precio" porque es posible que para algunos libros no le hayamos establecido el precio para la venta.

En contraposición, tenemos campos que no pueden estar vacíos jamás.

Veamos un ejemplo. Tenemos nuestra tabla "libros". El campo "titulo" no debería estar vacío nunca, igualmente el campo "autor". Para ello, al crear la tabla, debemos especificar que dichos campos no admitan valores nulos:

create table libros(

titulo varchar(30) not null,

autor varchar(20) not null,

editorial varchar(15) null,

precio float

);

Para especificar que un campo no admita valores nulos, debemos colocar "not null" luego de la definición del campo. En el ejemplo anterior, los campos "editorial" y "precio" si admiten valores nulos. Cuando colocamos "null" estamos diciendo que admite valores nulos (caso del campo "editorial"); por defecto, es decir, si no lo aclaramos, los campos permiten valores nulos (caso del campo "precio").

Si ingresamos los datos de un libro, para el cual aún no hemos definido el precio podemos colocar "null" para mostrar que no tiene precio:

insert into libros (titulo,autor,editorial,precio)

values('El aleph','Borges','Emece',null);

Note que el valor "null" no es una cadena de caracteres, no se coloca entre comillas. Entonces, si un campo acepta valores nulos, podemos ingresar "null" cuando no conocemos el valor.

También podemos colocar "null" en el campo "editorial" si desconocemos el nombre de la editorial a la cual pertenece el libro que vamos a ingresar:

insert into libros (titulo,autor,editorial,precio)

values('Alicia en el pais','Lewis Carroll',null,25);

Si intentamos ingresar el valor "null" en campos que no admiten valores nulos (como "titulo" o "autor"), SQL Server no lo permite, muestra un mensaje y la inserción no se realiza; por ejemplo:

insert into libros (titulo,autor,editorial,precio)

values(null,'Borges','Siglo XXI',25);

Para ver cuáles campos admiten valores nulos y cuáles no, podemos emplear el procedimiento almacenado "sp\_columns" junto al nombre de la tabla. Nos muestra mucha información, en la columna "IS\_NULLABLE" vemos que muestra "NO" en los campos que no permiten valores nulos y "YES" en los campos que si los permiten.

Para recuperar los registros que contengan el valor "null" en algún campo, no podemos utilizar los operadores relacionales vistos anteriormente: = (igual) y <> (distinto); debemos utilizar los operadores "is null" (es igual a null) y "is not null" (no es null):

select \* from libros

where precio is null;

La sentencia anterior tendrá una salida diferente a la siguiente:

select \* from libros

where precio=0;

Con la primera sentencia veremos los libros cuyo precio es igual a "null" (desconocido); con la segunda, los libros cuyo precio es 0.

Igualmente para campos de tipo cadena, las siguientes sentencias "select" no retornan los mismos registros:

select \* from libros where editorial is null;

select \* from libros where editorial='';

Con la primera sentencia veremos los libros cuya editorial es igual a "null", con la segunda, los libros cuya editorial guarda una cadena vacía.

Entonces, para que un campo no permita valores nulos debemos especificarlo luego de definir el campo, agregando "not null". Por defecto, los campos permiten valores nulos, pero podemos especificarlo igualmente agregando "null".

# Clave primaria

Una clave primaria es un campo (o varios) que identifica un solo registro (fila) en una tabla. Para un valor del campo clave existe solamente un registro.

Veamos un ejemplo, si tenemos una tabla con datos de personas, el número de documento puede establecerse como clave primaria, es un valor que no se repite; puede haber personas con igual apellido y nombre, incluso el mismo domicilio (padre e hijo por ejemplo), pero su documento será siempre distinto.

Si tenemos la tabla "usuarios", el nombre de cada usuario puede establecerse como clave primaria, es un valor que no se repite; puede haber usuarios con igual clave, pero su nombre de usuario será siempre diferente.

Podemos establecer que un campo sea clave primaria al momento de crear la tabla o luego que ha sido creada. Vamos a aprender a establecerla al crear la tabla. Hay 2 maneras de hacerlo, por ahora veremos la sintaxis más sencilla.

Tenemos nuestra tabla "usuarios" definida con 2 campos ("nombre" y "clave"). La sintaxis básica y general es la siguiente:

create table NOMBRETABLA(

CAMPO TIPO,

...

primary key (NOMBRECAMPO)

);

En el siguiente ejemplo definimos una clave primaria, para nuestra tabla "usuarios" para asegurarnos que cada usuario tendrá un nombre diferente y único:

create table usuarios(

nombre varchar(20),

clave varchar(10),

primary key(nombre)

);

Lo que hacemos agregar luego de la definición de cada campo, "primary key" y entre paréntesis, el nombre del campo que será clave primaria.

Una tabla sólo puede tener una clave primaria. Cualquier campo (de cualquier tipo) puede ser clave primaria, debe cumplir como requisito, que sus valores no se repitan ni sean nulos. Por ello, al definir un campo como clave primaria, automáticamente SQL Server lo convierte a "not null".

Luego de haber establecido un campo como clave primaria, al ingresar los registros, SQL Server controla que los valores para el campo establecido como clave primaria no estén repetidos en la tabla; si estuviesen repetidos, muestra un mensaje y la inserción no se realiza. Es decir, si en nuestra tabla "usuarios" ya existe un usuario con nombre "juanperez" e intentamos ingresar un nuevo usuario con nombre "juanperez", aparece un mensaje y la instrucción "insert" no se ejecuta.

Igualmente, si realizamos una actualización, SQL Server controla que los valores para el campo establecido como clave primaria no estén repetidos en la tabla, si lo estuviese, aparece un mensaje indicando que se viola la clave primaria y la actualización no se realiza.

# Campo con atributo Identity

Un campo numérico puede tener un atributo extra "identity". Los valores de un campo con este atributo genera valores secuenciales que se inician en 1 y se incrementan en 1 automáticamente.

Se utiliza generalmente en campos correspondientes a códigos de identificación para generar valores únicos para cada nuevo registro que se inserta.

Sólo puede haber un campo "identity" por tabla.

Para que un campo pueda establecerse como "identity", éste debe ser entero (también puede ser de un subtipo de entero o decimal con escala 0, tipos que estudiaremos posteriormente).

Para que un campo genere sus valores automáticamente, debemos agregar el atributo "identity" luego de su definición al crear la tabla:

create table libros(

codigo int identity,

titulo varchar(40) not null,

autor varchar(30),

editorial varchar(15),

precio float

);

Cuando un campo tiene el atributo "identity" no se puede ingresar valor para él, porque se inserta automáticamente tomando el último valor como referencia, o 1 si es el primero.

Para ingresar registros omitimos el campo definido como "identity", por ejemplo:

insert into libros (titulo,autor,editorial,precio)

values('El aleph','Borges','Emece',23);

Este primer registro ingresado guardará el valor 1 en el campo correspondiente al código.

Si continuamos ingresando registros, el código (dato que no ingresamos) se cargará automáticamente siguiendo la secuencia de autoincremento.

No está permitido ingresar el valor correspondiente al campo "identity", por ejemplo:

insert into libros (codigo,titulo,autor,editorial,precio)

values(5,'Martin Fierro','Jose Hernandez','Paidos',25);

generará un mensaje de error.

"identity" permite indicar el valor de inicio de la secuencia y el incremento, pero lo veremos posteriormente.

Un campo definido como "identity" generalmente se establece como clave primaria.

Un campo "identity" no es editable, es decir, no se puede ingresar un valor ni actualizarlo.

Un campo de identidad no permite valores nulos, aunque no se indique especificamente. Si ejecutamos el procedimiento "sp\_columns()" veremos que en el campo "codigo" en la columna "TYPE\_NAME" aparece "int identity" y en la columna "IS\_NULLABLE" aparece "NO".

Los valores secuenciales de un campo "identity" se generan tomando como referencia el último valor ingresado; si se elimina el último registro ingresado (por ejemplo 3) y luego se inserta otro registro, SQL Server seguirá la secuencia, es decir, colocará el valor "4".

# Otras características del atributo Identity

El atributo "identity" permite indicar el valor de inicio de la secuencia y el incremento, para ello usamos la siguiente sintaxis:

create table libros(

codigo int identity(100,2),

titulo varchar(20),

autor varchar(30),

precio float

);

Los valores comenzarán en "100" y se incrementarán de 2 en 2; es decir, el primer registro ingresado tendrá el valor "100", los siguientes "102", "104", "106", etc.

La función "ident\_seed()" retorna el valor de inicio del campo "identity" de la tabla que nombramos:

select ident\_seed('libros');

La función "ident\_incr()" retorna el valor de incremento del campo "identity" de la tabla nombrada:

select ident\_incr('libros');

Hemos visto que en un campo declarado "identity" no puede ingresarse explícitamente un valor. Para permitir ingresar un valor en un campo de identidad se debe activar la opción "identity\_insert":

set identity\_insert libros on;

Es decir, podemos ingresar valor en un campo "identity" seteando la opción "identity\_insert" en "on".

Cuando "identity\_insert" está en ON, las instrucciones "insert" deben explicitar un valor:

insert into libros (codigo,titulo)

values (5,'Alicia en el pais de las maravillas');

Si no se coloca un valor para el campo de identidad, la sentencia no se ejecuta y aparece un mensaje de error:

insert into libros (titulo,autor, editorial)

values ('Matematica estas ahi','Paenza','Paidos');

El atributo "identity" no implica unicidad, es decir, permite repetición de valores; por ello hay que tener cuidado al explicitar un valor porque se puede ingresar un valor repetido.

Para desactivar la opción "identity\_insert" tipeamos:

set identity\_insert libros off;

# Truncate table

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.

# 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:

* 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.
* 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.
* 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".

A continuación analizaremos en detalle cada tipo de dato básico.

# Tipo de dato (texto)

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:

* 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.
* 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.
* text: guarda datos binarios de longitud variable, puede contener hasta 2000000000 caracteres. No admite argumento para especificar su longitud.
* 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.
* 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.
* 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

# Tipo de dato (numérico)

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

# Tipo de dato (fecha y hora)

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

# Ingresar algunos campos (insert into)

Hemos aprendido a ingresar registros listando todos los campos y colocando valores para todos y cada uno de ellos luego de "values".

Si ingresamos valores para todos los campos, podemos omitir la lista de nombres de los campos. Por ejemplo, si tenemos creada la tabla "libros" con los campos "titulo", "autor" y "editorial", podemos ingresar un registro de la siguiente manera:

insert into libros

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

También es posible ingresar valores para algunos campos. Ingresamos valores solamente para los campos "titulo" y "autor":

insert into libros (titulo, autor)

values ('El aleph','Borges');

SQL Server almacenará el valor "null" en el campo "editorial", para el cual no hemos explicitado un valor.

Al ingresar registros debemos tener en cuenta:

- la lista de campos debe coincidir en cantidad y tipo de valores con la lista de valores luego de "values". Si se listan más (o menos) campos que los valores ingresados, aparece un mensaje de error y la sentencia no se ejecuta.

- si ingresamos valores para todos los campos podemos obviar la lista de campos.

- podemos omitir valores para los campos que NO hayan sido declarados "not null", es decir, que permitan valores nulos (se guardará "null"); si omitimos el valor para un campo "not null", la sentencia no se ejecuta.

- se DEBE omitir el valor para el campo"identity". Salvo que identity\_insert este en on.

- se pueden omitir valores para campos declarados "not null" siempre que tengan definido un valor por defecto con la cláusula "default" (tema que veremos a continuación).

# Valores por defecto (default)

Hemos visto que si al insertar registros no se especifica un valor para un campo que admite valores nulos, se ingresa automaticamente "null" y si el campo está declarado "identity", se inserta el siguiente de la secuencia. A estos valores se les denomina valores por defecto o predeterminados.

Un valor por defecto se inserta cuando no está presente al ingresar un registro y en algunos casos en que el dato ingresado es inválido.

Para campos de cualquier tipo no declarados "not null", es decir, que admiten valores nulos, el valor por defecto es "null". Para campos declarados "not null", no existe valor por defecto, a menos que se declare explícitamente con la cláusula "default".

Para todos los tipos, excepto los declarados "identity", se pueden explicitar valores por defecto con la cláusula "default".

Podemos establecer valores por defecto para los campos cuando creamos la tabla. Para ello utilizamos "default" al definir el campo. Por ejemplo, queremos que el valor por defecto del campo "autor" de la tabla "libros" sea "Desconocido" y el valor por defecto del campo "cantidad" sea "0":

create table libros(

codigo int identity,

titulo varchar(40),

autor varchar(30) not null default 'Desconocido',

editorial varchar(20),

precio decimal(5,2),

cantidad tinyint default 0

);

Si al ingresar un nuevo registro omitimos los valores para el campo "autor" y "cantidad", Sql Server insertará los valores por defecto; el siguiente valor de la secuencia en "codigo", en "autor" colocará "Desconocido" y en cantidad "0".

Entonces, si al definir el campo explicitamos un valor mediante la cláusula "default", ése será el valor por defecto.

Ahora, al visualizar la estructura de la tabla con "sp\_columns" podemos entender lo que informa la columna "COLUMN\_DEF", muestra el valor por defecto del campo.

También se puede utilizar "default" para dar el valor por defecto a los campos en sentencias "insert", por ejemplo:

insert into libros (titulo,autor,precio,cantidad)

values ('El gato con botas',default,default,100);

Si todos los campos de una tabla tienen valores predeterminados (ya sea por ser "identity", permitir valores nulos o tener un valor por defecto), se puede ingresar un registro de la siguiente manera:

insert into libros default values;

La sentencia anterior almacenará un registro con los valores predetermiandos para cada uno de sus campos.

Entonces, la cláusula "default" permite especificar el valor por defecto de un campo. Si no se explicita, el valor por defecto es "null", siempre que el campo no haya sido declarado "not null".

Los campos para los cuales no se ingresan valores en un "insert" tomarán los valores por defecto:

- si tiene el atributo "identity": el valor de inicio de la secuencia si es el primero o el siguiente valor de la secuencia, no admite cláusula "default";

- si permite valores nulos y no tiene cláusula "default", almacenará "null";

- si está declarado explícitamente "not null", no tiene valor "default" y no tiene el atributo "identity", no hay valor por defecto, así que causará un error y el "insert" no se ejecutará.

- si tiene cláusula "default" (admita o no valores nulos), el valor definido como predeterminado;

- para campos de tipo fecha y hora, si omitimos la parte de la fecha, el valor predeterminado para la fecha es "1900-01-01" y si omitimos la parte de la hora, "00:00:00".

Un campo sólo puede tener un valor por defecto. Una tabla puede tener todos sus campos con valores por defecto. Que un campo tenga valor por defecto no significa que no admita valores nulos, puede o no admitirlos.