101 - Crear tabla a partir de otra (select - into)

Podemos crear una tabla e insertar datos en ella en una sola sentencia consultando otra tabla (o varias) con esta sintaxis:

```
select CAMPOSNUEVATABLA
 into NUEVATABLA
 from TABLA
 where CONDICION;
```

Es decir, se crea una nueva tabla y se inserta en ella el resultado de una consulta a otra tabla.

Tenemos la tabla "libros" de una librería y queremos crear una tabla llamada "editoriales" que contenga los nombres de las editoriales.

La tabla "editoriales", que no existe, contendrá solamente un campo llamado "nombre". La tabla libros contiene varios registros.

Podemos crear la tabla "editoriales" con el campo "nombre" consultando la tabla "libros" y en el mismo momento insertar la información:

```
select distinct editorial as nombre
 into editoriales
 from libros;
```

La tabla "editoriales" se ha creado con el campo "nombre" seleccionado del campo "editorial" de "libros".

Los campos de la nueva tabla tienen el mismo nombre, tipo de dato y valores almacenados que los campos listados de la tabla consultada; si se quiere dar otro nombre a los campos de la nueva tabla se deben especificar alias.

Entonces, luego de la lista de selección de campos de la tabla a consultar, se coloca "into" seguido del nombre de la nueva tabla y se sigue con la consulta.

Podemos emplear "group by", funciones de agrupamiento y "order by" en las consultas. También podemos emplear "select... into" con combinaciones, para crear una tabla que contenga datos de 2 o más tablas.

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;
if object_id('editoriales')is not null
  drop table editoriales;

create table libros(
  codigo int identity,
  titulo varchar(40) not null,
  autor varchar(30),
  editorial varchar(20),
  precio decimal(5,2),
  primary key(codigo)
);

go

insert into libros values('Uno','Richard
Bach','Planeta',15);
insert into libros values('El
aleph','Borges','Emece',25);
insert into libros values('Matematica estas
ahi','Paenza','Nuevo siglo',18);
insert into libros values('Aprenda PHP','Mario
Molina','Nuevo siglo',45);
insert into libros values('Ilusiones','Richard
Bach','Planeta',14);
insert into libros values('Java en 10 minutos','Mario
Molina','Nuevo siglo',50);

-- Creamos una tabla llamada "editoriales" que contenga
los nombres de las editoriales:
select distinct editorial as nombre
  into editoriales
  from libros;

-- Veamos la nueva tabla:
select * from editoriales;
```

```
-- Necesitamos una nueva tabla llamada
"librosporeditorial" que contenga la cantidad
-- de libros de cada editorial. Primero eliminamos la
tabla, si existe:
if object_id('cantidadporeditorial') is not null
  drop table cantidadporeditorial;

-- Creamos la nueva tabla:
select editorial as nombre,count(*) as cantidad
  into cantidadporeditorial
  from libros
  group by editorial;

select * from cantidadporeditorial;

-- Queremos una tabla llamada "ofertas4" que contenga
los mismos campos que "libros"
-- y guarde los 4 libros más económicos. Primero
eliminamos, si existe, la tabla "ofertas4":
if object_id('ofertas4') is not null
  drop table ofertas4;

-- Creamos "ofertas4" e insertamos la consulta de
"libros":
select top 4 *
  into ofertas4
  from libros
  order by precio asc;

select * from ofertas4;

-- Agregamos una columna a la tabla "editoriales" que
contiene la ciudad
-- en la cual está la casa central de cada editorial:
alter table editoriales add ciudad varchar(30);

go

-- Actualizamos dicho campo:
update editoriales set ciudad='Cordoba' where
nombre='Planeta';
update editoriales set ciudad='Cordoba' where
nombre='Emece';
```

```
update editoriales set ciudad='Buenos Aires' where
nombre='Nuevo siglo';

-- Queremos una nueva tabla llamada "librosdecordoba"
que contenga los títulos y autores
-- de los libros de editoriales de Cordoba. En primer
lugar, la eliminamos, si existe:
if object_id('librosdecordoba') is not null
  drop table librosdecordoba;

 -- Consultamos las 2 tablas y guardamos el resultado en
la nueva tabla que estamos creando:
select titulo,autor
  into librosdecordoba
  from libros
  join editoriales
  on editorial=nombre
  where ciudad='Cordoba';

select * from librosdecordoba;
```

102 - go

Esto solo se aplica cuando instale el SQL Server en su máquina y utiliza el "SQL Server Management Studio".

"go" es un signo de finalización de un lote de sentencias SQL. No es una sentencia, es un comando.
El lote de sentencias está compuesto por todas las sentencias antes de "go" o todas las sentencias entre dos "go".

Las sentencias no deben ocupar la misma linea en la que está "go".

Habrá notado que no se puede ejecutar un procedimiento almacenado luego de otras sentencias a menos que se incluya "execute" (o "exec").

Por ejemplo, si tipeamos:

```
 select * from empleados;
 sp_helpconstraint empleados;
```

muestra un mensaje de error porque no puede procesar ambas sentencias como un solo lote. Para que no ocurra debemos tipear:

```
select * from empleados;
exec sp_helpconstraint empleados;
```

o separar los lotes con "go":

```
select * from empleados;
go
sp_helpconstraint empleados;
```

Las siguientes sentencias no pueden ejecutarse en el mismo lote: create rule, create default,create view, create procedure, create trigger. Cada una de ellas necesita ejecutarse separándolas con "go". Por ejemplo:

```
create table....
go
create rule...
go
```

Recuerde que si coloca "go" no debe incluir el "punto y coma" (;) al finalizar una instrucción.

No está de más recordar que esto solo se aplica cuando instale el SQL Server en su máquina y ejecute los comandos desde el "SQL Server Management Studio".

103 - Vistas

Una vista es una alternativa para mostrar datos de varias tablas. Una vista es como una tabla virtual que almacena una consulta. Los datos accesibles a través de la vista no están almacenados en la base de datos como un objeto.

Entonces, una vista almacena una consulta como un objeto para utilizarse posteriormente. Las tablas consultadas en una vista se llaman tablas base. En general, se puede dar un nombre a cualquier consulta y almacenarla como una vista.

Una vista suele llamarse también tabla virtual porque los resultados que retorna y la manera de referenciarlas es la misma que para una tabla.

Las vistas permiten:

- ocultar información: permitiendo el acceso a algunos datos y manteniendo oculto el resto de la información que no se incluye en la vista. El usuario

opera con los datos de una vista como si se tratara de una tabla, pudiendo modificar tales datos.

- simplificar la administración de los permisos de usuario: se pueden dar al usuario permisos para que solamente pueda acceder a los datos a través de vistas, en lugar de concederle permisos para acceder a ciertos campos, así se protegen las tablas base de cambios en su estructura.

- mejorar el rendimiento: se puede evitar tipear instrucciones repetidamente almacenando en una vista el resultado de una consulta compleja que incluya información de varias tablas.

Podemos crear vistas con: un subconjunto de registros y campos de una tabla; una unión de varias tablas; una combinación de varias tablas; un resumen estadístico de una tabla; un subconjunto de otra vista, combinación de vistas y tablas.

Una vista se define usando un "select".

La sintaxis básica parcial para crear una vista es la siguiente:

```
create view NOMBREVISTA as
 SENTENCIASSELECT
  from TABLA;
```

El contenido de una vista se muestra con un "select":

```
select *from NOMBREVISTA;
```

En el siguiente ejemplo creamos la vista "vista_empleados", que es resultado de una combinación en la cual se muestran 4 campos:

```
create view vista_empleados as
 select (apellido+' '+e.nombre) as nombre,sexo,
  s.nombre as seccion, cantidadhijos
  from empleados as e
  join secciones as s
  on codigo=seccion
```

Para ver la información contenida en la vista creada anteriormente tipeamos:

```
select *from vista_empleados;
```

Podemos realizar consultas a una vista como si se tratara de una tabla:

```
select seccion,count(*) as cantidad
 from vista_empleados
 group by seccion;
```

Los nombres para vistas deben seguir las mismas reglas que cualquier identificador. Para distinguir una tabla de una vista podemos fijar una convención para darle nombres, por ejemplo, colocar el sufijo “vista” y luego el nombre de las tablas consultadas en ellas.

Los campos y expresiones de la consulta que define una vista DEBEN tener un nombre. Se debe colocar nombre de campo cuando es un campo calculado o si hay 2 campos con el mismo nombre. Note que en el ejemplo, al concatenar los campos "apellido" y "nombre" colocamos un alias; si no lo hubiésemos hecho aparecería un mensaje de error porque dicha expresión DEBE tener un encabezado, SQL Server no lo coloca por defecto.

Los nombres de los campos y expresiones de la consulta que define una vista DEBEN ser únicos (no puede haber dos campos o encabezados con igual nombre). Note que en la vista definida en el ejemplo, al campo "s.nombre" le colocamos un alias porque ya había un encabezado (el alias de la concatenación) llamado "nombre" y no pueden repetirse, si sucediera, aparecería un mensaje de error.

Otra sintaxis es la siguiente:

```
create view NOMBREVISTA (NOMBRESDEENCABEZADOS)
 as
 SENTENCIASSELECT
  from TABLA;
```

Creamos otra vista de "empleados" denominada "vista_empleados_ingreso" que almacena la cantidad de empleados por año:

```
create view vista_empleados_ingreso (fecha,cantidad)
 as
 select datepart(year,fechaingreso),count(*)
  from empleados
  group by datepart(year,fechaingreso)
```

La diferencia es que se colocan entre paréntesis los encabezados de las columnas que aparecerán en la vista. Si no los colocamos y empleamos la sintaxis vista anteriormente, se emplean los nombres de los campos o alias

(que en este caso habría que agregar) colocados en el "select" que define la vista. Los nombres que se colocan entre paréntesis deben ser tantos como los campos o expresiones que se definen en la vista.

Las vistas se crean en la base de datos activa.

Al crear una vista, SQL Server verifica que existan las tablas a las que se hacen referencia en ella.

Se aconseja probar la sentencia "select" con la cual definiremos la vista antes de crearla para asegurarnos que el resultado que retorna es el imaginado.

Existen algunas restricciones para el uso de "create view", a saber:

- no puede incluir las cláusulas "compute" ni "compute by" ni la palabra clave "into";

- no se pueden crear vistas temporales ni crear vistas sobre tablas temporales.

- no se pueden asociar reglas ni valores por defecto a las vistas.

- no puede combinarse con otras instrucciones en un mismo lote.

Se pueden construir vistas sobre otras vistas.

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;
if object_id('secciones') is not null
  drop table secciones;

create table secciones(
  codigo tinyint identity,
  nombre varchar(20),
  sueldo decimal(5,2)
   constraint CK_secciones_sueldo check (sueldo>=0),
  constraint PK_secciones primary key (codigo)
);
```

```
create table empleados(
  legajo int identity,
  documento char(8)
   constraint CK_empleados_documento check (documento
like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
  sexo char(1)
   constraint CK_empleados_sexo check (sexo in
('f','m')),
  apellido varchar(20),
  nombre varchar(20),
  domicilio varchar(30),
  seccion tinyint not null,
  cantidadhijos tinyint
   constraint CK_empleados_hijos check
(cantidadhijos>=0),
  estadocivil char(10)
   constraint CK_empleados_estadocivil check
(estadocivil in
('casado','divorciado','soltero','viudo')),
  fechaingreso datetime,
   constraint PK_empleados primary key (legajo),
  constraint FK_empleados_seccion
   foreign key (seccion)
   references secciones(codigo)
   on update cascade,
  constraint UQ_empleados_documento
   unique(documento)
);

go

insert into secciones values('Administracion',300);
insert into secciones values('Contaduría',400);
insert into secciones values('Sistemas',500);

insert into empleados
values('22222222','f','Lopez','Ana','Colon
123',1,2,'casado','1990-10-10');
insert into empleados
values('23333333','m','Lopez','Luis','Sucre
235',1,0,'soltero','1990-02-10');
```

```
insert into empleados
values('24444444','m','Garcia','Marcos','Sarmiento
1234',2,3,'divorciado','1998-07-12');
insert into empleados
values('25555555','m','Gomez','Pablo','Bulnes
321',3,2,'casado','1998-10-09');
insert into empleados
values('26666666','f','Perez','Laura','Peru
1254',3,3,'casado','2000-05-09');

-- Eliminamos la vista "vista_empleados" si existe:
if object_id('vista_empleados') is not null
  drop view vista_empleados;

go

-- Creamos la vista "vista_empleados", que es resultado
de una combinación
-- en la cual se muestran 5 campos:
create view vista_empleados as
  select (apellido+' '+e.nombre) as nombre,sexo,
   s.nombre as seccion, cantidadhijos
   from empleados as e
   join secciones as s
   on codigo=seccion;

go

-- Vemos la información de la vista:
select * from vista_empleados;

-- Realizamos una consulta a la vista como si se tratara
de una tabla:
select seccion,count(*) as cantidad
  from vista_empleados
  group by seccion;

-- Eliminamos la vista "vista_empleados_ingreso" si
existe:
if object_id('vista_empleados_ingreso') is not null
  drop view vista_empleados_ingreso;

go
```

```
-- Creamos otra vista de "empleados" denominada
"vista_empleados_ingreso"
-- que almacena la cantidad de empleados por año:
create view vista_empleados_ingreso (fecha,cantidad)
  as
  select datepart(year,fechaingreso),count(*)
   from empleados
   group by datepart(year,fechaingreso);

go

-- Vemos la información de la vista creada:
select * from vista_empleados_ingreso;
```

104 - Vistas (información)

Las vistas son objetos, así que para obtener información de ellos pueden usarse los siguientes procedimientos almacenados del sistema:

"sp_help" sin parámetros nos muestra todos los objetos de la base de datos seleccionada, incluidas las vistas. En la columna "Object_type" aparece "view" si es una vista. Si le enviamos como argumento el nombre de una vista, obtenemos la fecha de creación, propietario, los campos y demás información.

"sp_helptext" seguido del nombre de una vista nos muestra el texto que la define, excepto si ha sido encriptado.

Ejecutando "sp_depends" seguido del nombre de un objeto, obtenemos 2 resultados:

- nombre, tipo, campos, etc. de los objetos de los cuales depende el objeto nombrado y

- nombre y tipo de los objetos que dependen del objeto nombrado.

Si ejecutamos el procedimiento "sp_depends" seguido del nombre de una vista:

```
 exec sp_depends vista_empleados;
```

aparecen las tablas (y demás objetos) de las cuales depende la vista, es decir, las tablas referenciadas en la misma.

Si ejecutamos el procedimiento seguido del nombre de una tabla:

```
 exec sp_depends empleados;
```

aparecen los objetos que dependen de la tabla, vistas, restricciones, etc.

También se puede consultar la tabla del sistema "sysobjects":

```
 select *from sysobjects;
```

Nos muestra nombre y varios datos de todos los objetos de la base de datos actual. La columna "xtype" indica el tipo de objeto, si es una vista, aparece 'V'.

Si queremos ver todas las vistas creadas por nosotros, podemos tipear:

```
 select *from sysobjects
  where xtype='V' and-- tipo vista
  name like 'vista%';--búsqueda con comodín
```

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;
if object_id('secciones') is not null
  drop table secciones;

create table secciones(
  codigo tinyint identity,
  nombre varchar(20),
  sueldo decimal(5,2)
   constraint CK_secciones_sueldo check (sueldo>=0),
  constraint PK_secciones primary key (codigo)
);

create table empleados(
  legajo int identity,
  documento char(8)
   constraint CK_empleados_documento check (documento
like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
  sexo char(1)
```

```
   constraint CK_empleados_sexo check (sexo in
('f','m')),
  apellido varchar(20),
  nombre varchar(20),
  domicilio varchar(30),
  seccion tinyint not null,
  cantidadhijos tinyint
   constraint CK_empleados_hijos check
(cantidadhijos>=0),
  estadocivil char(10)
   constraint CK_empleados_estadocivil check
(estadocivil in
('casado','divorciado','soltero','viudo')),
  fechaingreso datetime,
   constraint PK_empleados primary key (legajo),
  constraint FK_empleados_seccion
   foreign key (seccion)
   references secciones(codigo)
   on update cascade,
  constraint UQ_empleados_documento
   unique(documento)
);

go

insert into secciones values('Administracion',300);
insert into secciones values('Contaduría',400);
insert into secciones values('Sistemas',500);

insert into empleados
values('22222222','f','Lopez','Ana','Colon
123',1,2,'casado','1990-10-10');
insert into empleados
values('23333333','m','Lopez','Luis','Sucre
235',1,0,'soltero','1990-02-10');
insert into empleados
values('24444444','m','Garcia','Marcos','Sarmiento
1234',2,3,'divorciado','1998-07-12');
insert into empleados
values('25555555','m','Gomez','Pablo','Bulnes
321',3,2,'casado','1998-10-09');
insert into empleados
values('26666666','f','Perez','Laura','Peru
1254',3,3,'casado','2000-05-09');
```

```
-- Eliminamos la vista "vista_empleados" si existe:
if object_id('vista_empleados') is not null
  drop view vista_empleados;

go

-- Creamos la vista "vista_empleados", que es resultado
de una combinación
-- en la cual se muestran 5 campos:
create view vista_empleados as
  select (apellido+' '+e.nombre) as nombre,sexo,
   s.nombre as seccion, cantidadhijos
   from empleados as e
   join secciones as s
   on codigo=seccion;

go

-- Vemos la información de la vista:
select * from vista_empleados;

-- Ejecutamos "sp_help" enviándole como argumento el
nombre de la vista:
exec sp_help vista_empleados;

-- Vemos el texto que define la vista:
exec sp_helptext vista_empleados;

-- Ejecutamos el procedimiento almacenado del sistema
"sp_depends"
-- seguido del nombre de la vista. Aparecen las tablas y
campos de las
-- cuales depende la vista, es decir, las tablas
referenciadas en la misma:
exec sp_depends vista_empleados;

-- Ejecutamos el procedimiento "sp_depends" seguido del
nombre de la
-- tabla "empleados". Aparece la vista "vista_empleados"
y las
-- restricciones que dependen de ella.
exec sp_depends empleados;
```

```
-- Consultamos la tabla del sistema "sysobjects":
select * from sysobjects;

-- Si queremos ver todas las vistas creadas por
nosotros, podemos tipear:
select * from sysobjects
  where xtype='V' and-- tipo vista
  name like 'vista%'--búsqueda con comodín ;
```

105 - vistas (encriptar)

Podemos ver el texto que define una vista ejecutando el procedimiento almacenado del sistema "sp_helptext" seguido del nombre de la vista:

```
 exec sp_helptext NOMBREVISTA;
```

Podemos ocultar el texto que define una vista empleando la siguiente sintaxis al crearla:

```
 create view NOMBREVISTA
  with encryption
  as
  SENTENCIASSELECT
   from TABLA;
```

"with encryption" indica a SQL Server que codifique las sentencias que definen la vista.

Creamos una vista con su definición oculta:

```
 create view vista_empleados
  with encryption
 as
  select (apellido+' '+e.nombre) as nombre,sexo,
   s.nombre as seccion, cantidadhijos
   from empleados as e
   join secciones as s
   on codigo=seccion
```

Si ejecutamos el procedimiento almacenado del sistema "sp_helptext" seguido del nombre de una vista encriptada, aparece un mensaje indicando tal situación y el texto no se muestra.

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;
if object_id('secciones') is not null
  drop table secciones;

create table secciones(
  codigo tinyint identity,
  nombre varchar(20),
  sueldo decimal(5,2)
   constraint CK_secciones_sueldo check (sueldo>=0),
  constraint PK_secciones primary key (codigo)
);

create table empleados(
  legajo int identity,
  documento char(8)
   constraint CK_empleados_documento check (documento
like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
  sexo char(1)
   constraint CK_empleados_sexo check (sexo in
('f','m')),
  apellido varchar(20),
  nombre varchar(20),
  domicilio varchar(30),
  seccion tinyint not null,
  cantidadhijos tinyint
   constraint CK_empleados_hijos check
(cantidadhijos>=0),
  estadocivil char(10)
   constraint CK_empleados_estadocivil check
(estadocivil in
('casado','divorciado','soltero','viudo')),
  fechaingreso datetime,
   constraint PK_empleados primary key (legajo),
  constraint FK_empleados_seccion
   foreign key (seccion)
   references secciones(codigo)
   on update cascade,
  constraint UQ_empleados_documento
```

```
   unique(documento)
);

go

if object_id('vista_empleados') is not null
  drop view vista_empleados;

go

create view vista_empleados
  with encryption
 as
  select (apellido+' '+e.nombre) as nombre,sexo,
   s.nombre as seccion, cantidadhijos
   from empleados as e
   join secciones as s
   on codigo=seccion;

go

exec sp_helptext vista_empleados;
```

106 - Vistas (eliminar)

Para quitar una vista se emplea "drop view":

```
 drop view NOMBREVISTA;
```

Si se elimina una tabla a la que hace referencia una vista, la vista no se elimina, hay que eliminarla explícitamente.

Solo el propietario puede eliminar una vista.

Antes de eliminar un objeto, se recomienda ejecutar el procedimiento almacenado de sistema "sp_depends" para averiguar si hay objetos que hagan referencia a él.

Eliminamos la vista denominada "vista_empleados":

```
 drop view vista_empleados;
```

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;
if object_id('secciones') is not null
  drop table secciones;

create table secciones(
  codigo tinyint identity,
  nombre varchar(20),
  sueldo decimal(5,2)
   constraint CK_secciones_sueldo check (sueldo>=0),
  constraint PK_secciones primary key (codigo)
);

create table empleados(
  legajo int identity,
  documento char(8)
   constraint CK_empleados_documento check (documento
like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
  sexo char(1)
   constraint CK_empleados_sexo check (sexo in
('f','m')),
  apellido varchar(20),
  nombre varchar(20),
  domicilio varchar(30),
  seccion tinyint not null,
  cantidadhijos tinyint
   constraint CK_empleados_hijos check
(cantidadhijos>=0),
  estadocivil char(10)
   constraint CK_empleados_estadocivil check
(estadocivil in
('casado','divorciado','soltero','viudo')),
  fechaingreso datetime,
   constraint PK_empleados primary key (legajo),
  constraint FK_empleados_seccion
   foreign key (seccion)
   references secciones(codigo)
   on update cascade,
  constraint UQ_empleados_documento
```

```
   unique(documento)
);

go

insert into secciones values('Administracion',300);
insert into secciones values('Contaduría',400);
insert into secciones values('Sistemas',500);

insert into empleados
values('22222222','f','Lopez','Ana','Colon
123',1,2,'casado','1990-10-10');
insert into empleados
values('23333333','m','Lopez','Luis','Sucre
235',1,0,'soltero','1990-02-10');
insert into empleados
values('24444444','m','Garcia','Marcos','Sarmiento
1234',2,3,'divorciado','1998-07-12');
insert into empleados
values('25555555','m','Gomez','Pablo','Bulnes
321',3,2,'casado','1998-10-09');
insert into empleados
values('26666666','f','Perez','Laura','Peru
1254',3,3,'casado','2000-05-09');

-- Eliminamos la vista "vista_empleados" si existe:
if object_id('vista_empleados') is not null
  drop view vista_empleados;

go

-- Creamos la vista "vista_empleados", que es resultado
de una combinación
-- en la cual se muestran 5 campos:
create view vista_empleados as
  select (apellido+' '+e.nombre) as nombre,sexo,
   s.nombre as seccion, cantidadhijos
   from empleados as e
   join secciones as s
   on codigo=seccion;

go

-- Veamos la información de la vista:
```

```
select * from vista_empleados;

-- Eliminamos la tabla "empleados":
drop table empleados;

-- Verificamos que la vista aún existe:
exec sp_help;

-- Eliminamos la vista:
drop view vista_empleados;

-- Verificamos que la vista ya no existe:
exec sp_help vista_empleados;
```

107 - Vistas (with check option)

Es posible obligar a todas las instrucciones de modificación de datos que se ejecutan en una vista a cumplir ciertos criterios.

Por ejemplo, creamos la siguiente vista:

```
 create view vista_empleados
 as
  select apellido, e.nombre, sexo, s.nombre as seccion
  from empleados as e
  join secciones as s
  on seccion=codigo
  where s.nombre='Administracion'
  with check option;
```

La vista definida anteriormente muestra solamente algunos de los datos de los empleados de la sección "Administracion". Además, solamente se permiten modificaciones a los empleados de esa sección.

Podemos actualizar el nombre, apellido y sexo a través de la vista, pero no el campo "seccion" porque está restringuido.

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;
if object_id('secciones') is not null
  drop table secciones;

create table secciones(
  codigo tinyint identity,
  nombre varchar(20),
  sueldo decimal(5,2)
   constraint CK_secciones_sueldo check (sueldo>=0),
  constraint PK_secciones primary key (codigo)
);

create table empleados(
  legajo int identity,
  documento char(8)
   constraint CK_empleados_documento check (documento
like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
  sexo char(1)
   constraint CK_empleados_sexo check (sexo in
('f','m')),
  apellido varchar(20),
  nombre varchar(20),
  domicilio varchar(30),
  seccion tinyint not null,
  cantidadhijos tinyint
   constraint CK_empleados_hijos check
(cantidadhijos>=0),
  estadocivil char(10)
   constraint CK_empleados_estadocivil check
(estadocivil in
('casado','divorciado','soltero','viudo')),
  fechaingreso datetime,
   constraint PK_empleados primary key (legajo),
  constraint FK_empleados_seccion
   foreign key (seccion)
   references secciones(codigo)
   on update cascade,
  constraint UQ_empleados_documento
   unique(documento)
);

go

insert into secciones values('Administracion',300);
```

```
insert into secciones values('Contaduría',400);
insert into secciones values('Sistemas',500);

insert into empleados
values('22222222','f','Lopez','Ana','Colon
123',1,2,'casado','1990-10-10');
insert into empleados
values('23333333','m','Lopez','Luis','Sucre
235',1,0,'soltero','1990-02-10');
insert into empleados
values('24444444','m','Garcia','Marcos','Sarmiento
1234',2,3,'divorciado','1998-07-12');
insert into empleados
values('25555555','m','Gomez','Pablo','Bulnes
321',3,2,'casado','1998-10-09');
insert into empleados
values('26666666','f','Perez','Laura','Peru
1254',3,3,'casado','2000-05-09');

if object_id('vista_empleados') is not null
  drop view vista_empleados;

go

create view vista_empleados
 as
  select apellido, e.nombre, sexo, s.nombre as seccion
  from empleados as e
  join secciones as s
  on seccion=codigo
  where s.nombre='Administracion'
  with check option;

go

select * from vista_empleados;

update vista_empleados set nombre='Beatriz' where
nombre='Ana';

select * from empleados;
```

108 - Vistas (modificar datos de una tabla a través de vistas)

Si se modifican los datos de una vista, se modifica la tabla base.

Se puede insertar, actualizar o eliminar datos de una tabla a través de una vista, teniendo en cuenta lo siguiente, las modificaciones que se realizan a las vistas:

- no pueden afectar a más de una tabla consultada. Pueden modificarse datos de una vista que combina varias tablas pero la modificación solamente debe afectar a una sola tabla.

- no se pueden cambiar los campos resultado de un cálculo.

- pueden generar errores si afectan a campos a las que la vista no hace referencia. Por ejemplo, si se ingresa un registro en una vista que consulta una tabla que tiene campos not null que no están incluidos en la vista.

- la opción "with check option" obliga a todas las instrucciones de modificación que se ejecutan en la vista a cumplir ciertos criterios que se especifican al definir la vista.

- para eliminar datos de una vista solamente UNA tabla puede ser listada en el "from" de la definicion de la misma.

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;
if object_id('secciones') is not null
  drop table secciones;

create table secciones(
  codigo tinyint identity,
  nombre varchar(20),
  sueldo decimal(5,2)
   constraint CK_secciones_sueldo check (sueldo>=0),
  constraint PK_secciones primary key (codigo)
);

create table empleados(
```

```
  legajo int identity,
  documento char(8)
   constraint CK_empleados_documento check (documento
like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
  sexo char(1)
   constraint CK_empleados_sexo check (sexo in
('f','m')),
  apellido varchar(20),
  nombre varchar(20),
  domicilio varchar(30),
  seccion tinyint not null,
  cantidadhijos tinyint
   constraint CK_empleados_hijos check
(cantidadhijos>=0),
  estadocivil char(10)
   constraint CK_empleados_estadocivil check
(estadocivil in
('casado','divorciado','soltero','viudo')),
  fechaingreso datetime,
   constraint PK_empleados primary key (legajo),
  sueldo decimal(6,2),
  constraint FK_empleados_seccion
   foreign key (seccion)
   references secciones(codigo)
   on update cascade,
  constraint UQ_empleados_documento
   unique(documento)
);

go

insert into secciones values('Administracion',300);
insert into secciones values('Contaduría',400);
insert into secciones values('Sistemas',500);

insert into empleados
values('22222222','f','Lopez','Ana','Colon
123',1,2,'casado','1990-10-10',600);
insert into empleados
values('23333333','m','Lopez','Luis','Sucre
235',1,0,'soltero','1990-02-10',650);
insert into empleados values('24444444', 'm', 'Garcia',
'Marcos', 'Sarmiento 1234', 2, 3, 'divorciado', '1998-
07-12',800);
```

```
insert into empleados
values('25555555','m','Gomez','Pablo','Bulnes
321',3,2,'casado','1998-10-09',900);
insert into empleados
values('26666666','f','Perez','Laura','Peru
1254',3,3,'casado','2000-05-09',700);

if object_id('vista_empleados') is not null
  drop view vista_empleados;

go

-- Creamos la vista "vista_empleados", que es resultado
de una combinación en la cual se muestran 5 campos:
create view vista_empleados as
  select (apellido+' '+e.nombre) as nombre,sexo,
   s.nombre as seccion, cantidadhijos
   from empleados as e
   join secciones as s
   on codigo=seccion;

go

select * from vista_empleados;

if object_id('vista_empleados2') is not null
  drop view vista_empleados2;

go

-- Creamos otra vista de "empleados" denominada
"vista_empleados2"
-- que consulta solamente la tabla "empleados" con "with
check option":
create view vista_empleados2
  as
  select nombre,
apellido,fechaingreso,seccion,estadocivil,sueldo
   from empleados
  where sueldo>=600
  with check option;

go
```

```
-- Consultamos la vista:
 select * from vista_empleados2;

-- Ingresamos un registro en la vista
"vista_empleados2":
insert into vista_empleados
values('Pedro','Perez','2000-10-10',1,'casado',800);
-- No es posible insertar un registro en la vista
"vista_empleados2" porque el campo de la vista "nombre"
es un campo calculado.

-- Actualizamos la sección de un registro de la vista
"vista_empleados":
 update vista_empleados set seccion='Sistemas' where
nombre='Lopez Ana';

-- Si intentamos actualizar el nombre de un empleado no
lo permite porque es una columna calculada.
-- Actualizamos el nombre de un registro de la vista
"vista_empleados2":
 update vista_empleados set nombre='Beatriz' where
nombre='Ana';

-- Verifique que se actualizó la tabla:
 select * from empleados;

-- Eliminamos un registro de la vista
"vista_empleados2":
 delete from vista_empleados2 where apellido='Lopez';

-- Si podemos eliminar registros de la vista
"vista_empleados2" dicha vista solamente consulta una
tabla.
-- No podemos eliminar registros de la vista
"vista_empleados" porque hay varias tablas en su
definición.
```

109 - Vistas modificar (alter view)

Para modificar una vista puede eliminarla y volver a crearla o emplear "alter view".

Con "alter view" se modifica la definición de una vista sin afectar los procedimientos almacenados y los permisos. Si elimina una vista y vuelve a crearla, debe reasignar los permisos asociados a ella.

Sintaxis básica para alterar una vista:

```
 alter view NOMBREVISTA
  with encryption--opcional
 as SELECT
```

En el ejemplo siguiente se altera vista_empleados para agregar el campo "domicilio":

```
 alter view vista_empleados
  with encryption
 as
  select (apellido+' '+e.nombre) as nombre,sexo,
   s.nombre as seccion, cantidadhijos,domicilio
   from empleados as e
   join secciones as s
   on codigo=seccion
```

Si creó la vista con "with encryption" y quiere modificarla manteniendo la encriptación, debe colocarla nuevamente, en caso de no hacerlo, desaparece.

Si crea una vista con "select *" y luego agrega campos a la estructura de las tablas involucradas, los nuevos campos no aparecerán en la vista; esto es porque los campos se seleccionan al ejecutar "create view"; debe alterar la vista.

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;
if object_id('secciones') is not null
  drop table secciones;

create table secciones(
  codigo tinyint identity,
  nombre varchar(20),
```

```
  constraint PK_secciones primary key (codigo)
);

create table empleados(
  legajo int identity,
  documento char(8),
  nombre varchar(30),
  domicilio varchar(30),
  seccion tinyint not null,
  constraint FK_empleados_seccion
   foreign key (seccion)
   references secciones(codigo)
   on update cascade,
  constraint PK_empleados
   primary key (documento)
);

go

insert into secciones values('Administracion');
insert into secciones values('Contaduría');
insert into secciones values('Sistemas');

insert into empleados values('22222222','Lopez
Ana','Colon 123',1);
insert into empleados values('23333333','Lopez
Luis','Sucre 235',1);
insert into empleados values('24444444','Garcia
Marcos','Sarmiento 1234',2);
insert into empleados values('25555555','Gomez
Pablo','Bulnes 321',3);
insert into empleados values('26666666','Perez
Laura','Peru 1254',3);

if object_id('vista_empleados') is not null
  drop view vista_empleados;

go
-- Creamos la vista "vista_empleados" encriptada que
muestre algunos campos
-- de los empleados de la sección 1 y colocamos "with
check option":
create view vista_empleados
  with encryption
```

```
 as
  select documento,nombre,seccion
  from empleados
  where seccion=1
  with check option;

go

select * from vista_empleados;

-- Veamos el texto de la vista (No lo permite porque
está encriptada):
exec sp_helptext vista_empleados;

go

-- Modificamos la vista para que muestre el domicilio y
no colocamos
-- la opción de encriptación ni "with check option":
alter view vista_empleados
 as
  select documento,nombre,seccion, domicilio
  from empleados
  where seccion=1;

go

select * from vista_empleados;

-- Veamos el texto de la vista(Lo permite porque ya no
está encriptada):
exec sp_helptext vista_empleados;

-- Actualizamos la sección de un empleado:
update vista_empleados set seccion=2 where
documento='22222222';

select * from vista_empleados;

if object_id('vista_empleados2') is not null
  drop view vista_empleados2;

go
```

```
-- Creamos la vista "vista_empleados2" que muestre todos
los campos
-- de la tabla "empleados":
create view vista_empleados2
 as
  select * from empleados;

go

select * from vista_empleados2;

-- Agregamos un campo a la tabla "empleados":
alter table empleados
 add sueldo decimal(6,2);

-- Consultamos la vista "vista_empleados2" (Note que el
nuevo campo
-- agregado a "empleados" no aparece):
select * from vista_empleados2;

go

-- Alteramos la vista para que se muestre el campo
sueldo:
alter view vista_empleados2
 as
  select * from empleados;

go

select * from vista_empleados2;
```

110 - Lenguaje de control de flujo (case)

La sentencia "case" compara 2 o más valores y devuelve un resultado.
La sintaxis es la siguiente:

```
 case VALORACOMPARAR
  when VALOR1 then RESULTADO1
  when VALOR2 then RESULTADO2
  ...
  else RESULTADO3
 end
```

Por cada valor hay un "when" y un "then"; si encuentra un valor coincidente en algún "when" ejecuta el "then" correspondiente a ese "when", si no encuentra ninguna coincidencia, se ejecuta el "else"; si no hay parte "else" retorna "null". Finalmente se coloca "end" para indicar que el "case" ha finalizado.

Un profesor guarda las notas de sus alumnos de un curso en una tabla llamada "alumnos" que consta de los siguientes campos:

- nombre (30 caracteres),
- nota (valor entero entre 0 y 10, puede ser nulo).

Queremos mostrar los nombres, notas de los alumnos y en una columna extra llamada "resultado" empleamos un case que testee la nota y muestre un mensaje diferente si en dicho campo hay un valor:

- 0, 1, 2 ó 3: 'libre';
- 4, 5 ó 6: 'regular';
- 7, 8, 9 ó 10: 'promocionado';

Esta es la sentencia:

```
 select nombre,nota, resultado=
 case nota
  when 0 then 'libre'
  when 1 then 'libre'
  when 2 then 'libre'
  when 3 then 'libre'
  when 4 then 'regular'
  when 5 then 'regular'
  when 6 then 'regular'
  when 7 then 'promocionado'
  when 8 then 'promocionado'
  when 9 then 'promocionado'
  when 10 then 'promocionado'
 end
from alumnos;
```

Note que cada "when" compara un valor puntual, por ello los valores devueltos son iguales para algunos casos. Note que como omitimos la parte "else", en caso que el valor no encuentre coincidencia con ninguno valor "when", retorna "null".

Podemos realizar comparaciones en cada "when". La sintaxis es la siguiente:

```
 case
  when VALORACOMPARAR OPERADOR VALOR1 then RESULTADO1
  when VALORACOMPARAR OPERADOR VALOR2 then RESULTADO2
  ...
  else RESULTADO3
 end
```

Mostramos los nombres de los alumnos y en una columna extra llamada "resultado" empleamos un case que testee si la nota es menor a 4, está entre 4 y 7 o supera el 7:

```
select nombre, nota, condicion=
  case
   when nota<4 then 'libre'
   when nota >=4 and nota<7 then 'regular'
   when nota>=7 then 'promocionado'
   else 'sin nota'
  end
 from alumnos;
```

Puede utilizar una expresión "case" en cualquier lugar en el que pueda utilizar una expresión.

También se puede emplear con "group by" y funciones de agrupamiento.

Servidor de SQL Server instalado en forma local.

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

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

create table alumnos(
  nombre varchar(40),
  nota tinyint,
    constraint CK_alunos_nota check (nota>=0 and
nota<=10)
);

go
```

```
insert into alumnos values('Ana Acosta',8);
insert into alumnos values('Carlos Caseres',4);
insert into alumnos values('Federico Fuentes',2);
insert into alumnos values('Gaston Guzman',3);
insert into alumnos values('Hector Herrero',5);
insert into alumnos values('Luis Luna',10);
insert into alumnos values('Marcela Morales',7);
insert into alumnos values('Marcela Morales',null);

-- Queremos mostrar el nombre y nota de cada alumno y en
una columna
--  extra llamada "condicion" empleamos un case que
testee la nota
-- y muestre un mensaje diferente según la nota:
select nombre,nota, condicion=
 case nota
  when 0 then 'libre'
  when 1 then 'libre'
  when 2 then 'libre'
  when 3 then 'libre'
  when 4 then 'regular'
  when 5 then 'regular'
  when 6 then 'regular'
  when 7 then 'promocionado'
  when 8 then 'promocionado'
  when 9 then 'promocionado'
  when 10 then 'promocionado'
 end
 from alumnos;

-- Obtenemos la misma salida pero empleando el "case"
con operadores de comparación:
select nombre, nota, condicion=
  case
   when nota<4 then 'libre'
   when nota >=4 and nota<7 then 'regular'
   when nota>=7 then 'promocionado'
   else 'sin nota'
  end
 from alumnos;

-- Vamos a agregar el campo "condicion" a la tabla:
alter table alumnos
```

```
  add condicion varchar(20);

go

select * from alumnos;

-- Recordemos que se puede emplear una expresión "case"
en cualquier lugar en el
-- que pueda utilizar una expresión.
-- Queremos actualizar el campo "condicion" para guardar
la condición de cada
-- alumno según su nota:
update alumnos set condicion=
  case
   when nota<4 then 'libre'
   when nota between 4 and 7 then 'regular'
   when nota >7 then 'promocionado'
  end;

 select * from alumnos;

 -- Mostramos la cantidad de alumnos libres, regulares y
aprobados
 -- y en una columna extra mostramos un mensaje,
ordenamos por cantidad:
 select condicion, count(*) as cantidad,resultado=
  case condicion
    when 'libre' then 'repitentes'
    when 'regular' then 'rinden final'
    when 'promocionado' then 'no rinden final'
    else 'sin datos'
  end
 from alumnos
 group by condicion
 order by cantidad;
```

111 - Lenguaje de control de flujo (if)

Existen palabras especiales que pertenecen al lenguaje de control de flujo que controlan la ejecución de las sentencias, los bloques de sentencias y procedimientos almacenados.

Tales palabras son: begin... end, goto, if... else, return, waitfor, while, break y continue.

- "begin... end" encierran un bloque de sentencias para que sean tratados como unidad.

- "if... else": testean una condición; se emplean cuando un bloque de sentencias debe ser ejecutado si una condición se cumple y si no se cumple, se debe ejecutar otro bloque de sentencias diferente.

- "while": ejecuta repetidamente una instrucción siempre que la condición sea verdadera.

- "break" y "continue": controlan la operación de las instrucciones incluidas en el bucle "while".

Veamos un ejemplo. Tenemos nuestra tabla "libros"; queremos mostrar todos los títulos de los cuales no hay libros disponibles (cantidad=0), si no hay, mostrar un mensaje indicando tal situación:

```
 if exists (select * from libros where cantidad=0)
  (select titulo from libros where cantidad=0)
 else
  select 'No hay libros sin stock';
```

SQL Server ejecuta la sentencia (en este caso, una subconsulta) luego del "if" si la condición es verdadera; si es falsa, ejecuta la sentencia del "else" (si existe).

Podemos emplear "if...else" en actualizaciones. Por ejemplo, queremos hacer un descuento en el precio, del 10% a todos los libros de una determinada editorial; si no hay, mostrar un mensaje:

```
 if exists (select * from libros where
editorial='Emece')
 begin
  update libros set precio=precio-(precio*0.1) where
editorial='Emece'
  select 'libros actualizados'
 end
 else
  select 'no hay registros actualizados';
```

Note que si la condición es verdadera, se deben ejecutar 2 sentencias. Por lo tanto, se deben encerrar en un bloque "begin...end".

En el siguiente ejemplo eliminamos los libros cuya cantidad es cero; si no hay, mostramos un mensaje:

```
 if exists (select * from libros where cantidad=0)
  delete from libros where cantidad=0
 else
  select 'No hay registros eliminados;
```

Dentro de los comandos SQL select, update y delete no podemos hacer uso de la sentencia de control de flujo if, debemos utilizar la sentencia case que vimos en el concepto anterior.

A partir de la versión 2012 de SQL Server disponemos de la función integrada iif:

```
select titulo,costo=iif(precio<38,'barato','caro') from
libros;
```

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(30),
  editorial varchar(20),
  precio decimal(5,2),
  cantidad tinyint,
  primary key (codigo)
);

go

insert into libros values('Uno','Richard
Bach','Planeta',15,100);
insert into libros values('El
aleph','Borges','Emece',20,150);
```

```
insert into libros values('Aprenda PHP','Mario
Molina','Nuevo siglo',50,200);
insert into libros values('Alicia en el pais de las
maravillas','Lewis Carroll','Emece',15,0);
insert into libros values('Java en 10 minutos','Mario
Molina','Emece',40,200);

-- Mostramos los títulos de los cuales no hay libros
disponibles (cantidad=0);
-- en caso que no haya, mostramos un mensaje:
if exists (select * from libros where cantidad=0)
  (select titulo from libros where cantidad=0)
else
  select 'No hay libros sin stock';

-- Hacemos un descuento del 10% a todos los libros de
editorial "Emece";
-- si no hay, mostramos un mensaje:
if exists (select * from libros where editorial='Emece')
begin
  update libros set precio=precio-(precio*0.1) where
editorial='Emece'
  select 'libros actualizados'
end
else
  select 'no hay registros actualizados';

-- Veamos si se actualizaron:
select * from libros where editorial='Emece';

-- Eliminamos los libros de los cuales no hay stock
(cantidad=0);
-- si no hay, mostramos un mensaje:
if exists (select * from libros where cantidad=0)
  delete from libros where cantidad=0
else
  select 'No hay registros eliminados';

-- Ejecutamos nuevamente la sentencia anterior (Ahora se
ejecuta la sentencia
-- del "else" porque no hay registros que cumplieran la
condición.):
if exists (select * from libros where cantidad=0)
  delete from libros where cantidad=0
```

```
 else
  select 'No hay registros eliminados';

select titulo,costo=iif(precio<38,'barato','caro') from
libros;
```

112 - Variables de usuario

Las variables nos permiten almacenar un valor y recuperarlo más adelante para emplearlos en otras sentencias.

Las variables de usuario son específicas de cada conexión y son liberadas automáticamente al abandonar la conexión.

Las variables de usuario comienzan con "@" (arroba) seguido del nombre (sin espacios), dicho nombre puede contener cualquier caracter.

Una variable debe ser declarada antes de usarse. Una variable local se declara así:

```
 declare @NOMBREVARIABLE TIPO
```

colocando "declare" el nombre de la variable que comienza con el símbolo arroba (@) y el tipo de dato. Ejemplo:

```
 declare @nombre varchar(20)
```

Puede declarar varias variables en una misma sentencia:

```
 declare @nombre varchar(20), @edad int
```

No existen variables globales en SQL Server.

Una variable declarada existe dentro del entorno en que se declara; debemos declarar y emplear la variable en el mismo lote de sentencias, porque si declaramos una variable y luego, en otro bloque de sentencias pretendemos emplearla, dicha variable ya no existe.

Una variable a la cual no se le ha asignado un valor contiene "null".

Se le asigna un valor inicial con "set":

```
 set @edad=45
```

Para almacenar un valor en una variable se coloca el signo igual (=) entre la variable y el valor a asignar.

Si le asignamos un valor resultado de una consulta, la sintaxis es:

```
select @nombre = autor from libros where titulo='Uno'
```

Podemos ver el contenido de una variable con:

```
select @nombre;
```

Una variable puede tener comodines:

```
declare @patron varchar(30)
set @patron='B%';
select autor
from libros
 where autor like @patron;
```

La utilidad de las variables consiste en que almacenan valores para utilizarlos en otras consultas.

Por ejemplo, queremos saber todos los datos del libro con mayor precio de la tabla "libros" de una librería. Para ello podemos emplear una variable para almacenar el precio más alto:

```
declare @mayorprecio decimal(5,2)
select @mayorprecio=max(precio) from libros;
```

y luego mostrar todos los datos de dicho libro empleando la variable anterior:

```
select *from libros
 where precio=@mayorprecio;
```

Es decir, declaramos la variable y guardamos en ella el precio más alto y luego, en otra sentencia, mostramos los datos de todos los libros cuyo precio es igual al valor de la variable.

Una variable puede ser definida con cualquier tipo de dato, excepto text, ntext e image; incluso de un tipo de dato definido por el usuario.

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(30),
  autor varchar(25),
  editorial varchar(20),
  precio decimal(5,2)
);

go

insert into libros values('Uno','Bach
Richard','Planeta',15);
insert into libros values('El aleph','Borges J.
L.','Emece',25);
insert into libros values('Matematica estas ahi','Paenza
Adrian','Siglo XXI',15);
insert into libros values('Aprenda PHP','Mario
Molina','Siglo XXI',35);
insert into libros values('Java en 10 minutos','Mario
Molina','Siglo XXI',35);

-- Declare una variable llamada "@valor" de tipo "int" y
vea su contenido:
declare @valor int
select @valor;

-- Declare una variable llamada "@nombre" de tipo
"varchar(20)",
-- asígnele un valor y vea su contenido:
declare @nombre varchar(20)
set @nombre='Juan';
select @nombre;

-- Queremos saber todos los datos de los libros con
mayor precio de la tabla "libros".
-- Declare una variable de tipo decimal, busque el
precio más alto de "libros"
```

```
-- y almacénelo en una variable,
-- luego utilice dicha variable para mostrar todos los
datos del libro:
declare @mayorprecio  decimal(5,2)
select @mayorprecio=max(precio) from libros
select * from libros where precio=@mayorprecio;
```

113 - Tipos de datos text, ntext y image (remplazados por varchar(max), nvarchar(max) y varbinary(max))

Los tipos de datos text, ntext e image se eliminarán en versiones futuras de SQL Server. Evite utilizar estos tipos de datos en nuevos proyectos de desarrollo y planee modificar las aplicaciones que los utilizan actualmente. Se debe utilizar los tipos varchar (max), nvarchar (max) y varbinary (max) en su lugar.

Los tipos de datos "ntext", "text" e "image" representan tipos de datos de longitud fija y variable en los que se pueden guardar gran cantidad de información, caracteres unicode y no unicode y datos binarios.

"ntext" almacena datos unicode de longitud variable y el máximo es de aproximadamente 1000000000 caracteres, en bytes, el tamaño es el doble de los caracteres ingresados (2 GB).

"text" almacena datos binarios no unicode de longitud variable, el máximo es de 2000000000 caracteres aprox. (2 GB). No puede emplearse en parámetros de procedimientos almacenados.

"image" es un tipo de dato de longitud variable que puede contener de 0 a 2000000000 bytes (2 GB) aprox. de datos binarios. Se emplea para almacenar gran cantidad de información o gráficos.

Se emplean estos tipos de datos para almacenar valores superiores a 8000 caracteres.
Ninguno de estos tipos de datos admiten argumento para especificar su longitud, como en el caso de los tipos "char", o "varchar".

Como estos tipos de datos tiene gran tamaño, SQL Server los almacena fuera de los registros, en su lugar guarda un puntero (de 16 bytes) que apunta a otro sitio que contiene los datos.

Para declarar un campo de alguno de estos tipos de datos, colocamos el nombre del campo seguido del tipo de dato:

```
 ...
 NOMBRECAMPO text
 ....
```

Otras consideraciones importantes:

- No pueden definirse variables de estos tipos de datos.

- Los campos de estos tipos de datos no pueden emplearse para índices.

- La única restricción que puede aplicar a estos tipos de datos es "default".

- Se pueden asociar valores predeterminados pero no reglas a campos de estos tipos de datos.

- No pueden alterarse campos de estos tipos con "alter table".

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(40),
  autor varchar(30),
  editorial varchar(20),
  precio decimal(6,2),
  sinopsis text
);

go

insert into libros values
 ('Ilusiones','Richard Bach','Planeta',15,null);
insert into libros values
 ('Aprenda PHP','Mario Molina','Nuevo Siglo',45,'Trata
todos los temas necesarios para el aprendizaje de PHP');
insert into libros (titulo,autor,editorial) values
```

```
 ('Uno','Richard Bach','Planeta');
insert into libros values
 ('El aleph','Borges','Emece',18,'Uno de los más
célebres libros de este autor');

-- La siguiente consulta muestra la cantidad de libros
que tienen sinopsis:
select count(*)
  from libros
  where sinopsis is not null;

-- Agregamos una restricción "default" al campo
"sinopsis" (es la única restricción que
-- puede aplicarse a campos de tipo "text"):
alter table libros
 add constraint DF_libros_sinopsis
 default 'No hay datos'
 for sinopsis;

-- Ingresamos un registro con valores por defecto:
insert into libros default values;

-- Recuperamos los registros y vemos que se almacenó el
valor por defecto:
select * from libros;

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

-- Creamos ahora la tabla con el campo sinopsis con la
nueva sintaxis propuesta
-- para SQL Server
create table libros(
  titulo varchar(40),
  autor varchar(30),
  editorial varchar(20),
  precio decimal(6,2),
  sinopsis varchar(max)
);

go

insert into libros values
 ('Ilusiones','Richard Bach','Planeta',15,null);
```

```
insert into libros values
 ('Aprenda PHP','Mario Molina','Nuevo Siglo',45,'Trata
todos los temas necesarios para el aprendizaje de PHP');
insert into libros (titulo,autor,editorial) values
 ('Uno','Richard Bach','Planeta');
insert into libros values
 ('El aleph','Borges','Emece',18,'Uno de los más
célebres libros de este autor');

-- La siguiente consulta muestra la cantidad de libros
que tienen sinopsis:
select count(*)
  from libros
  where sinopsis is not null;

-- Agregamos una restricción "default" al campo
"sinopsis" (es la única restricción que
-- puede aplicarse a campos de tipo "text"):
alter table libros
 add constraint DF_libros_sinopsis
 default 'No hay datos'
 for sinopsis;

-- Ingresamos un registro con valores por defecto:
insert into libros default values;

-- Recuperamos los registros y vemos que se almacenó el
valor por defecto:
select * from libros;
```

114 - Tipo de dato text - ntext e image (punteros)

Los tipos de datos text, ntext e image se eliminarán en versiones futuras de SQL Server. Evite utilizar estos tipos de datos en nuevos proyectos de desarrollo y planee modificar las aplicaciones que los utilizan actualmente. Se debe utilizar los tipos varchar(max), nvarchar(max) y varbinary(max) en su lugar.

Explicamos anteriormente que como estos tipos de datos tiene gran tamaño, SQL Server almacena los datos fuera de los registros; en el registro

guarda un puntero (de 16 bytes) que apunta a otro sitio, que contiene la dirección en la cual se guardan los datos propiamente dichos.

La función "textptr" devuelve el valor del puntero a texto que corresponde al campo text, ntext o image; tal valor puede emplearse para manipular los datos de este tipo, con las funciones para leer, escribir y actualizar.
Sintaxis:

```
 textptr(CAMPO);
```

El campo debe ser tipo text, ntext o image.

En el campo de tipo "text" no se almacenan los datos sino la dirección en la cual se encuentran los datos. Podemos ver esa dirección tipeando la siguiente sentencia:

```
 select titulo, textptr(sinopsis) from libros;
```

La función "textptr" retorna un puntero a texto (valor binario de 16). Si el campo no tiene texto, retorna un puntero a null; por ello se debe usar la función "textvalid" para confirmar si el puntero a texto existe.

Si la consulta retorna más de un registro, "textptr" retorna un puntero a texto del último registro devuelto.

La funcion "textvalid" controla si un puntero a texto es válido. Sintaxis:

```
 textvalid ('TABLA.CAMPO', PUNTEROATEXTO);
```

Los argumentos son: el nombre de la tabla y campo y el nombre del puntero a texto que se va a controlar. Retorna 1 si el puntero es válido y 0 si no lo es. No se puede emplear "updatetext", "writetext" y "readtext" si el puntero no es válido.

La siguiente consulta muestra si los punteros son válidos en cada registro del campo "sinopsis" de la tabla "libros":

```
 select titulo,
  textvalid('libros.sinopsis', textptr(sinopsis)) as
'Puntero valido'
  from libros;
```

En el siguiente ejemplo, declaramos una variable de tipo "varbinary" a la cual le asignamos el valor del puntero a texto de un registro y luego vemos si dicho puntero es válido, empleando la variable:

```
declare @puntero varbinary(16)
select @puntero = textptr(sinopsis)
 from libros
 where titulo= 'Ilusiones'
select textvalid('libros.sinopsis', @puntero);
```

Solo disponemos punto y coma al final para que SQL Server ejecute todas las instrucciones en un solo lote y exista la variable @puntero.

Si al insertar registros se ingresa un valor "null" en un campo "text", "ntext" o "image" o no se ingresa valor, no se crea un puntero válido. Para crear un puntero a texto válido ejecute un "insert" o "update" con datos que no sean nulos para el campo text, ntext o image.

Problema:

Una librería almacena los datos de sus libros en una tabla llamada "libros". Eliminamos la tabla, si existe:

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

Creamos la tabla con la siguiente estructura:

```
create table libros(
 titulo varchar(40),
 autor varchar(30),
 editorial varchar(20),
 precio decimal(6,2),
 sinopsis text
);
```

Ingresamos algunos registros:

```
insert into libros values
('Ilusiones','Richard Bach','Planeta',15,null);
insert into libros values
('Aprenda PHP','Mario Molina','Nuevo Siglo',45,
   'Trata todos los temas necesarios para el aprendizaje de PHP');
insert into libros (titulo,autor,editorial) values
('Uno','Richard Bach','Planeta');
```

Veamos la dirección que almacena cada registro en el campo de tipo "text" ("sinopsis") de la tabla "libros":

```
select titulo, textptr(sinopsis) as direccion from libros;
```

Note que un sólo libro, el segundo, tiene una dirección, los demás contienen "null".

La siguiente consulta muestra si los punteros son válidos en cada registro del campo "sinopsis" de la tabla "libros":

```
select titulo,
 textvalid('libros.sinopsis', textptr(sinopsis)) as 'Puntero valido'
 from libros;
```

Note que el primer y tercer libro tienen punteros inválidos, en el primero porque se almacenó "null" y en el segundo, porque el campo no fue inicializado.

Declaramos una variable a la cual le asignamos el valor del puntero a texto de un registro y luego vemos si dicho puntero es válido:

```
declare @puntero varbinary(16)
select @puntero = textptr(sinopsis) from libros
 where titulo= 'Ilusiones'
select textvalid('libros.sinopsis', @puntero);
```

Veamos el siguiente ejemplo:

```
declare @puntero varbinary(16)
select @puntero = textptr(sinopsis) from libros
select textvalid('libros.sinopsis', @puntero);
```

Declaramos una variable, almacenamos en ella el puntero a texto de "libros"; esta consulta retorna varios registros, por lo tanto, el puntero devuelve el valor del último registro (cuyo título es 'Uno'). Finalmente chequeamos si el valor del puntero es válido.

115 - Tipo de dato text - ntext e image (leer)

Los tipos de datos text, ntext e image se eliminarán en versiones futuras de SQL Server. Evite utilizar estos tipos de datos en nuevos proyectos de desarrollo y planee modificar las aplicaciones que los utilizan actualmente. Se debe utilizar los tipos varchar(max), nvarchar(max) y varbinary(max) en su lugar.

La función "readtext" lee valores de un campo text, ntext o image, comenzando desde una posición y leyendo un específico número de bytes.
Sintaxis:

```
readtext TABLA.CAMPO PUNTEROATEXTO
 DESPLAZAMIENTO CANTIDAD;
```

Analicemos la sintaxis:

- PUNTEROATEXTO: puntero a texto válido, binary(16).

- DESPLAZAMIENTO: número de bytes (para text o image) o caracteres (ntext) que se mueve el puntero antes de comenzar a leer.

- CANTIDAD: número de bytes o caracteres a leer desde la posición indicada por DESPLAZAMIENTO. Si es 0, se leen 4KB bytes o hasta el final.

Leemos la información almacenada en el campo "sinopsis" de "libros" del registro con código 2, desde la posición 9, 50 caracteres:

```
declare @puntero varbinary(16)
select @puntero=textptr(sinopsis)
 from libros
 where codigo=2
readtext libros.sinopsis @puntero 9 50;
```

Si al insertar registros se ingresa un valor "null" en un campo "text", "ntext" o "image" o no se ingresan datos, no se crea un puntero válido y al intentar leer dicho campo ocurre un error, porque la función "readtext" requiere un puntero válido. Para evitarlo podemos chequear el puntero antes de pasárselo a la función de lectura:

```
declare @puntero varbinary(16)
select @puntero=textptr(sinopsis)
 from libros where codigo=1
if (textvalid('libros.sinopsis', @puntero)=1)
 readtext libros.sinopsis @puntero 9 50
else select 'puntero invalido';
```

Problema:

Una librería almacena los datos de sus libros en una tabla llamada "libros".
Eliminamos la tabla, si existe:

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

Creamos la tabla con la siguiente estructura:

```
 create table libros(
  codigo int identity,
  titulo varchar(40),
  autor varchar(30),
  editorial varchar(20),
  sinopsis text
 );
```

Ingresamos algunos registros:

```
 insert into libros values ('Ilusiones','Richard Bach','Planeta',null);
 insert into libros values ('Aprenda PHP','Mario Molina','Nuevo Siglo',
   'Contiene todos los temas necesarios para el aprendizaje de PHP');
 insert into libros (titulo,autor,editorial) values ('Uno','Richard
Bach','Planeta');
 insert into libros values ('El Aleph','Borges','Emece',
   'Uno de los libros más célebres de este autor.');
```

Leemos la información almacenada en el campo "sinopsis" de "libros" del registro con código 2, desde la posición 9, 50 caracteres:

```
 declare @puntero varbinary(16)
 select @puntero=textptr(sinopsis)
  from libros
  where codigo=2
 readtext libros.sinopsis @puntero 9 50;
```

Leemos la información almacenada en el campo "sinopsis" de "libros" del registro con código 3, tal registro no tiene inicializado el campo "sinopsis", para que no ocurra un error, vamos a controlar el puntero antes de leer el campo:

```
 declare @puntero varbinary(16)
 select @puntero=textptr(sinopsis)
  from libros where codigo=3
 if (textvalid('libros.sinopsis', @puntero)=1)
  readtext libros.sinopsis @puntero 0 0
 else select 'puntero invalido';
```

Recuperamos la sinopsis del libro con código 4 (desde el comienzo al final), controlando que el puntero sea válido:

```
 declare @puntero varbinary(16)
 select @puntero=textptr(sinopsis)
  from libros where codigo=4
 if (textvalid('libros.sinopsis', @puntero)=1)
  readtext libros.sinopsis @puntero 0 0
 else select 'puntero invalido';
```

116 - Tipo de dato text - ntext e image (escribir)

Los tipos de datos text, ntext e image se eliminarán en versiones futuras de SQL Server. Evite utilizar estos tipos de datos en nuevos proyectos de desarrollo y planee modificar las aplicaciones que los utilizan actualmente. Se debe utilizar los tipos varchar(max), nvarchar(max) y varbinary(max) en su lugar.

La función "writetext" sobreescribe (reemplaza) el texto de un campo "text", "ntext" o "image".
No puede emplearse en vistas.

Sintaxis:

```
 writetext TABLA.CAMPO PUNTEROATEXTO
   DATO;
```

Luego de "writetext" se coloca el nombre de la tabla y el campo (text, ntext o image) a actualizar. "PUNTEROATEXTO" es el valor que almacena el puntero a texto del dato de tipo "text", "ntext" o "image", tal puntero debe ser válido. "DATO" es el texto que almacena, puede ser una variable o un literal.

Este ejemplo coloca el puntero a texto en una variable "@puntero" y luego "writetext" almacena el nuevo texto en el registro apuntado por "@puntero":

```
 declare @puntero binary(16)
 select @puntero = textptr (sinopsis)
  from libros
  where codigo=2
  writetext libros.sinopsis @puntero 'Este es un nuevo
libro acerca de PHP escrito por el profesor
Molina que aborda todos los temas necesarios para el
aprendizaje desde cero de este lenguaje.';
```

Recuerde que si al insertar registros se ingresa un valor "null" en un campo "text", "ntext" o "image" o no se ingresan datos, no se crea un puntero válido y al intentar escribir dicho campo ocurre un error, porque la función "writetext" requiere un puntero válido. Para evitarlo podemos chequer el puntero antes de pasárselo a la función de escritura:

```
 declare @puntero varbinary(16)
 select @puntero=textptr(sinopsis)
  from libros where codigo=1
 if (textvalid('libros.sinopsis', @puntero)=1)
  writetext libros.sinopsis @puntero 'Trata de una
gaviota que vuela más alto que las demas.'
 else select 'puntero invalido, no se actualizó el
registro';
```

Problema:

Una librería almacena los datos de sus libros en una tabla llamada "libros".
Eliminamos la tabla, si existe:

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

Creamos la tabla con la siguiente estructura:

```
 create table libros(
  codigo int identity,
  titulo varchar(40),
  autor varchar(30),
  editorial varchar(20),
  sinopsis text
 );
```

Ingresamos algunos registros:

```
 insert into libros values ('Ilusiones','Richard Bach','Planeta',null);
 insert into libros values ('Aprenda PHP','Mario Molina','Nuevo Siglo',
   'Contiene todos los temas necesarios para el aprendizaje de PHP');
 insert into libros (titulo,autor,editorial) values ('Uno','Richard
Bach','Planeta');
 insert into libros values ('El Aleph','Borges','Emece','Uno de los
libros más célebres de este autor.');
```

Colocamos el puntero a texto del campo "sinopsis" del registro con código 2 en una variable "@puntero" y luego "writetext" almacena el nuevo texto en el registro apuntado por "@puntero":

```
 declare @puntero binary(16)
 select @puntero = textptr (sinopsis)
  from libros
  where codigo=2
 writetext libros.sinopsis @puntero 'Este es un nuevo libro acerca de PHP
escrito por el profesor
Molina que aborda todos los temas necesarios para el aprendizaje desde
cero de este lenguaje.';
```

Leemos el campo "sinopsis" del libro cuyo código es 2, para ver si se actualizó:

```
 declare @puntero binary(16)
 select @puntero = textptr (sinopsis)
  from libros
  where codigo=2
 readtext libros.sinopsis @puntero 0 0;
```

Si intentamos actualizar un campo "text", "ntext" o "image" que tiene valor nulo o un puntero inválido, aparece un mensaje de error, por lo tanto, en el siguiente ejemplo, verificamos si el puntero es válido antes de pasárselo a la función "writetext":

```
 declare @puntero varbinary(16)
 select @puntero=textptr(sinopsis)
  from libros where codigo=1
 if (textvalid('libros.sinopsis', @puntero)=1)
  writetext libros.sinopsis @puntero 'Trata de una gaviota que vuela más
alto que las demás.'
 else select 'puntero invalido, no se actualizó el registro';
```

Para crear un puntero válido debemos ingresar un valor con "insert" o "update":

```
 update libros set sinopsis='xx' where codigo=1;
```

Ahora si podemos actualizar el campo "sinopsis" del registro con código 1 con "writetext" porque ya tiene un puntero válido:

```
 declare @puntero varbinary(16)
 select @puntero=textptr(sinopsis)
  from libros where codigo=1
 if (textvalid('libros.sinopsis', @puntero)=1)
  writetext libros.sinopsis @puntero 'Trata de una gaviota que vuela más
alto que las demás.'
 else select 'puntero invalido, no se actualizó el registro';
```

Veamos si se actualizó:

```
 declare @puntero binary(16)
 select @puntero = textptr (sinopsis)
  from libros
  where codigo=1
 readtext libros.sinopsis @puntero 0 0;
```

Creamos un puntero válido al ingresar un nuevo registro:

```
 insert into libros values('Alicia en el pais de las maravillas','Lewis
Carroll','Planeta','');
```

Note que ingresamos una cadena vacía para el campo "sinopsis", con ello creamos un puntero válido.

Actualizamos el campo "sinopsis" del último registro ingresado con "writetext":

```
 declare @puntero varbinary(16)
 select @puntero=textptr(sinopsis)
  from libros where codigo=5
 if (textvalid('libros.sinopsis', @puntero)=1)
  writetext libros.sinopsis @puntero 'Trata de las aventuras de una niña
en un país muy extraño.'
 else select 'puntero invalido, no se actualizó el registro';
```

Veamos si se actualizó:

```
 declare @puntero binary(16)
 select @puntero = textptr (sinopsis)
  from libros
  where codigo=5
 readtext libros.sinopsis @puntero 0 0;
```

117 - Tipo de dato text - ntext e image (actualizar)

Los tipos de datos text, ntext e image se eliminarán en versiones futuras de SQL Server. Evite utilizar estos tipos de datos en nuevos proyectos de desarrollo y planee modificar las aplicaciones que los utilizan actualmente. Se debe utilizar los tipos varchar(max), nvarchar(max) y varbinary(max) en su lugar.

Aprendimos que la función "writetext" sobreescribe, reemplaza el contenido completo de un campo de tipo "text", "ntext" o "image".

Para actualizar campos de estos tipos también empleamos "updatetext", que permite cambiar una porción del campo (o todo el campo). La sintaxis básica es la siguiente:

```
 updatetext TABLA.CAMPO PUNTEROATEXTO
  DESPLAZAMIENTODELPUNTERO
  LONGITUDDEBORRADO
  DATOAINSERTAR;
```

Analizamos la sintaxis:

- TABLA.CAMPO: campo y tabla que se va a actualizar.

- PUNTEROATEXTO: valor del puntero, retornado por la función "textptr", que apunta al dato text, ntext o image que se quiere actualizar.

- DESPLAZAMIENTODELPUNTERO: indica la posición en que inserta el nuevo dato. Especifica la cantidad de bytes (para campos text e image) o caracteres (para campos ntext) que debe moverse el puntero para insertar el dato. Los valores pueden ser: 0 (el nuevo dato se inserta al comienzo), "null" (coloca el puntero al final), un valor mayor a cero y menor o igual a la longitud total del texto (inserta el nuevo dato en la posición indicada) y un valor mayor a la longitud total del campo (genera un mensaje de error).
Es importante recordar que cada caracter ntext ocupa 2 bytes.

- LONGITUDDEBORRADO: indica la cantidad de bytes (para text e image) o caracteres (para ntext) a borrar comenzando de la posición indicada por el parámetro DESPLAZAMIENTODELPUNTERO. Si colocamos el valor 0 (no se borra ningún dato), "null" (borra todos los datos desde la posición indicada por el parámetro DESPLAZAMIENTODELPUNTERO hasta el final), un valor mayor que cero y menor o igual a la longitud del texto (borra tal cantidad) y un valor inválido, es decir, mayor a la longitud del texto (genera un mensaje de error).
Es importante recordar que cada caracter "ntext" ocupa 2 bytes.

- DATOAINSERTAR: el dato que va a ser insertado en el campo. Puede ser char, nchar, varchar, nvarchar, binary, varbinary, text, ntext, image, un literal o una variable. Si el dato es un campo text, ntext o image de otra tabla, se debe indicar el nombre de la tabla junto con el campo y el valor del puntero que apunta al tipo de dato text, ntext o image (retornado por la función "textptr"), de esta forma:

```
TABLA.CAMPO PUNTERO;
```

Tenemos la tabla libros, con un campo de tipo text llamado "sinopsis"; hay un registro cargado con el siguiente texto: "Para aprender PHP a paso." Necesitamos agregar antes de "a paso" el texto "paso " para que el texto completo sea "Para aprender PHP paso a paso", tipeamos:

```
declare @puntero binary(16)
select @puntero = textptr(sinopsis)
 from libros
 where titulo='Aprenda PHP'
updatetext libros.sinopsis @puntero
 18 0 'paso ';
```

Entonces, declaramos una variable llamada "@puntero"; guardamos en la variable el valor del puntero, obtenido con la función "textptr(sinopsis)", tal puntero apunta al campo "sinopsis" del libro "Aprenda PHP". Luego actualizamos el campo, colocando el puntero en la posición 18, no borramos ningún byte y colocamos el texto a agregar; el campo ahora contendrá "Para aprencer PHP paso a paso".

Es posible guardar en un campo "text" de una tabla el contenido del campo "text" de otra tabla; para ello debemos utilizar 2 punteros, uno para obtener la dirección del campo que queremos actualizar y otro para obtener la dirección del campo del cual extraemos la información.

En el siguiente ejemplo guardamos en una variable el valor del puntero a texto al campo "sinopsis" del libro "Aprenda PHP" de la tabla "libros"; en otra variable guardamos el valor del puntero a texto al campo "sinopsis" del libro con código 1 de la tabla "ofertas"; finalmente actualizamos el registro de "ofertas" con el texto de "libros".

```
declare @puntero1 binary(16)
select @puntero1 = textptr(sinopsis)
 from libros
 where titulo='Aprenda PHP'

declare @puntero2 binary(16)
```

```
 select @puntero2 = textptr(sinopsis)
  from ofertas
  where titulo='Aprenda PHP'

  updatetext ofertas.sinopsis @puntero2 0 null
   libros.sinopsis @puntero1;
```

Entonces, se emplea "updatetext" para modificar datos de campos de tipo text, ntext e image, pudiendo cambiar una porción del texto.

Problema:

Una librería almacena los datos de sus libros en una tabla llamada "libros" y en otra tabla denominada "ofertas" almacena el título del libro y la sinopsis.
Eliminamos las tablas, si existen:

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

Creamos las tablas:

```
 create table libros(
  codigo int identity,
  titulo varchar(40),
  autor varchar(30),
  editorial varchar(20),
  sinopsis text
 );

 create table ofertas(
  titulo varchar(40),
  sinopsis text
 );
```

Ingresamos algunos registros:

```
 insert into libros values ('Ilusiones','Richard Bach','Planeta',null);
 insert into libros values ('Aprenda PHP','Mario Molina','Nuevo
Siglo','Para aprender PHP a paso');
 insert into libros values ('Uno','Richard Bach','Planeta','');
 insert into libros values ('El Aleph','Borges','Emece','Uno de los
libros más célebres de este autor.');

 insert into ofertas values ('Aprenda PHP','');
```

Recuperamos todos los registros de ambas tablas:

```
 select *from libros;
```

```
select *from ofertas;
```

Necesitamos que en la sinopsis del libro "Aprenda PHP" se guarde "Para aprender PHP paso a paso"; debemos insertar en la posición 18, el texto "paso ", sin eliminar ningún caracter. Verificamos que el puntero sea válido, en caso de no serlo, mostramos un mensaje de error:

```
declare @puntero binary(16)
select @puntero = textptr(sinopsis)
 from libros
 where titulo='Aprenda PHP'
 if (textvalid('libros.sinopsis',@puntero)=1)
  updatetext libros.sinopsis @puntero 18 0 'paso '
 else
  select 'Puntero inválido';
```

Leemos el campo "sinopsis" actualizado anteriormente para verificar que se actualizó:

```
declare @puntero binary(16)
select @puntero = textptr (sinopsis)
 from libros
 where titulo='Aprenda PHP'

readtext libros.sinopsis @puntero 0 0;
```

Necesitamos actualizar la sinopsis del libro "Aprenda PHP" de la tabla "ofertas" con la sinopsis del mismo libro de la tabla "libros":

```
declare @puntero1 binary(16)
select @puntero1 = textptr(sinopsis)
 from libros
 where titulo='Aprenda PHP'

declare @puntero2 binary(16)
select @puntero2 = textptr(sinopsis)
 from ofertas
 where titulo='Aprenda PHP'

 updatetext ofertas.sinopsis @puntero2 0 null
  libros.sinopsis @puntero1;
```

Leemos el campo "sinopsis" actualizado anteriormente para verificar que se actualizó:

```
declare @puntero binary(16)
select @puntero = textptr (sinopsis)
 from ofertas
 where titulo='Aprenda PHP'

readtext ofertas.sinopsis @puntero 0 0;
```

118 - Tipo de dato text - ntext e image (funciones)

Los tipos de datos text, ntext e image se eliminarán en versiones futuras de SQL Server. Evite utilizar estos tipos de datos en nuevos proyectos de desarrollo y planee modificar las aplicaciones que los utilizan actualmente. Se debe utilizar los tipos varchar(max), nvarchar(max) y varbinary(max) en su lugar.

Las siguientes son otras funciones que pueden emplearse con estos tipos de datos:

- datalenght(CAMPO): devuelve el número de bytes de un determinado campo. Retorna "null" si el campo es nulo. Ejemplo:

```
select titulo, datalength(sinopsis) as longitud
 from libros
 order by titulo;
```

- patindex ('PATRON',CAMPO): retorna el comienzo de la primera ocurrencia de un patrón de la expresión especificada, si el patrón no se encuentra, retorna cero. El patrón es una cadena que puede incluir comodines.
Ejemplo:

```
select patindex('%PHP%', sinopsis)
 from libros;
```

Con este tipo de datos también puede utilizarse "like", pero "like" solamente puede incluirse en la cláusula "where".

- substring (TEXTO,INICIO,LONGITUD): devuelve una parte del texto especificado 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 titulo,substring(sinopsis,1,20)
 from libros;
```

Problema:

Una librería almacena los datos de sus libros en una tabla llamada "libros".
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),
  autor varchar(30),
  editorial varchar(20),
  sinopsis text
 );
```

Ingresamos algunos registros:

```
 insert into libros values ('Ilusiones','Richard Bach','Planeta',null);
 insert into libros values ('Aprenda PHP','Mario Molina','Nuevo Siglo',
   'Para aprender PHP paso a paso');
 insert into libros values ('Programación elemental en PHP','Mario
Molina','Planeta',
   'Contiene conceptos básicos de PHP');
```

Veamos la longitud del campo "sinopsis" de todos los libros:

```
 select titulo, datalength(sinopsis) as longitud
  from libros
  order by titulo;
```

Buscamos todos los libros en los cuales en su sinopsis se encuentre el texto "PHP":

```
 select titulo
  from libros
  where patindex('%PHP%', sinopsis)>0;
```

Empleamos la función "patindex" para mostrar la posición en que se encuentra la cadena "PHP" en la sinopsis de todos los libros:

```
 select titulo, patindex('%PHP%', sinopsis) as posicion
  from libros;
```

Note que en el libro en el cual no se encuentra, retorna "0".

Seleccionamos los títulos y los primeros 10 caracteres de la sinopsis de cada uno de ellos:

```
 select titulo,substring(sinopsis,1,10) as sinopsis
  from libros;
```

119 - Procedimientos almacenados

Vimos que SQL Server ofrece dos alternativas para asegurar la integridad de datos, la integridad:

1) DECLARATIVA, mediante el uso de restricciones (constraints), valores predeterminados (defaults) y reglas (rules) y

2) PROCEDIMENTAL, mediante la implementación de procedimientos almacenados y desencadenadores (triggers).

Nos detendremos ahora en procedimientos almacenados.

Un procedimiento almacenado es un conjunto de instrucciones a las que se les da un nombre, que se almacena en el servidor. Permiten encapsular tareas repetitivas.

SQL Server permite los siguientes tipos de procedimientos almacenados:

1) del sistema: están almacenados en la base de datos "master" y llevan el prefijo "sp_"; permiten recuperar información de las tablas del sistema y pueden ejecutarse en cualquier base de datos.

2) locales: los crea el usuario (próximo tema).

3) temporales: pueden ser locales, cuyos nombres comienzan con un signo numeral (#), o globales, cuyos nombres comienzan con 2 signos numeral (##). Los procedimientos almacenados temporales locales están disponibles en la sesión de un solo usuario y se eliminan automáticamente al finalizar la sesión; los globales están disponibles en las sesiones de todos los usuarios.

4) extendidos: se implementan como bibliotecas de vínculos dinámicos (DLL, Dynamic-Link Libraries), se ejecutan fuera del entorno de SQL Server. Generalmente llevan el prefijo "xp_". No los estudiaremos.

Al crear un procedimiento almacenado, las instrucciones que contiene se analizan para verificar si son correctas sintácticamente. Si no se detectan errores, SQL Server guarda el nombre del procedimiento almacenado en la tabla del sistema "sysobjects" y su contenido en la tabla del sistema "syscomments" en la base de datos activa. Si se encuentra algún error, no se crea.

Un procedimiento almacenados puede hacer referencia a objetos que no existen al momento de crearlo. Los objetos deben existir cuando se ejecute el procedimiento almacenado.

Ventajas:

- comparten la lógica de la aplicación con las otras aplicaciones, con lo cual el acceso y las modificaciones de los datos se hacen en un solo sitio.

- permiten realizar todas las operaciones que los usuarios necesitan evitando que tengan acceso directo a las tablas.

- reducen el tráfico de red; en vez de enviar muchas instrucciones, los usuarios realizan operaciones enviando una única instrucción, lo cual disminuye el número de solicitudes entre el cliente y el servidor.

120 - Procedimientos almacenados (crear - ejecutar)

Los procedimientos almacenados se crean en la base de datos seleccionada, excepto los procedimientos almacenados temporales, que se crean en la base de datos "tempdb".

En primer lugar se deben tipear y probar las instrucciones que se incluyen en el procedimiento almacenado, luego, si se obtiene el resultado esperado, se crea el procedimiento.

Los procedimientos almacenados pueden hacer referencia a tablas, vistas, a funciones definidas por el usuario, a otros procedimientos almacenados y a tablas temporales.

Un procedimiento almacenado pueden incluir cualquier cantidad y tipo de instrucciones, excepto:

create default, create procedure, create rule, create trigger y create view. Se pueden crear otros objetos (por ejemplo índices, tablas), en tal caso deben especificar el nombre del propietario; se pueden realizar inserciones, actualizaciones, eliminaciones, etc.

Si un procedimiento almacenado crea una tabla temporal, dicha tabla sólo existe dentro del procedimiento y desaparece al finalizar el mismo. Lo mismo sucede con las variables.

Hemos empleado varias veces procedimientos almacenados del sistema ("sp_help", "sp_helpconstraint", etc.), ahora aprenderemos a crear nuestros propios procedimientos almacenados.

Para crear un procedimiento almacenado empleamos la instrucción "create procedure".
La sintaxis básica parcial es:

```
create procedure NOMBREPROCEDIMIENTO
 as INSTRUCCIONES;
```

Para diferenciar los procedimientos almacenados del sistema de los procedimientos almacenados locales use un prefijo diferente a "sp_" cuando les de el nombre.

Con las siguientes instrucciones creamos un procedimiento almacenado llamado "pa_libros_limite_stock" que muestra todos los libros de los cuales hay menos de 10 disponibles:

```
create proc pa_libros_limite_stock
 as
  select *from libros
  where cantidad <=10;
```

Entonces, creamos un procedimiento almacenado colocando "create procedure" (o "create proc", que es la forma abreviada), luego el nombre del procedimiento y seguido de "as" las sentencias que definen el procedimiento.

"create procedure" debe ser la primera sentencia de un lote.

Para ejecutar el procedimiento almacenado creado anteriormente tipeamos:

```
exec pa_libros_limite_stock;
```

Entonces, para ejecutar un procedimiento almacenado colocamos "execute" (o "exec") seguido del nombre del procedimiento.

Cuando realizamos un ejercicio nuevo, siempre realizamos las mismas tareas: eliminamos la tabla si existe, la creamos y luego ingresamos algunos registros. Podemos crear un procedimiento almacenado que contenga todas estas instrucciones:

```
create procedure pa_crear_libros
```

```
  as
  if object_id('libros')is not null
   drop table libros;
  create table libros(
   codigo int identity,
   titulo varchar(40),
   autor varchar(30),
   editorial varchar(20),
   precio decimal(5,2),
   primary key(codigo)
  );

  insert into libros values('Uno','Richard
Bach','Planeta',15);
  insert into libros values('Ilusiones','Richard
Bach','Planeta',18);
  insert into libros values('El
aleph','Borges','Emece',25);
  insert into libros values('Aprenda PHP','Mario
Molina','Nuevo siglo',45);
  insert into libros values('Matematica estas
ahi','Paenza','Nuevo siglo',12);
  insert into libros values('Java en 10 minutos','Mario
Molina','Paidos',35);
```

Y luego lo ejecutamos cada vez que comenzamos un nuevo ejercicio y así evitamos tipear tantas sentencias:

```
 exec pa_crear_libros;
```

Servidor de SQL Server instalado en forma local.

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

```
-- En primer lugar, debemos eliminarlo, si existe (no
hemos aprendido aún a eliminar procedimientos
-- almacenados, en próximos capítulos lo veremos):
if object_id('pa_crear_libros') is not null
  drop procedure pa_crear_libros;

go

-- Creamos el procedimiento:
create procedure pa_crear_libros
```

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

  create table libros(
   codigo int identity,
   titulo varchar(40),
   autor varchar(30),
   editorial varchar(20),
   precio decimal(5,2),
   cantidad smallint,
   primary key(codigo)
  )

  insert into libros values('Uno','Richard
Bach','Planeta',15,5)
  insert into libros values('Ilusiones','Richard
Bach','Planeta',18,50)
  insert into libros values('El
aleph','Borges','Emece',25,9)
  insert into libros values('Aprenda PHP','Mario
Molina','Nuevo siglo',45,100)
  insert into libros values('Matematica estas
ahi','Paenza','Nuevo siglo',12,50)
  insert into libros values('Java en 10 minutos','Mario
Molina','Paidos',35,300);

go

-- Ejecutamos el procedimiento:
exec pa_crear_libros;

-- Veamos si ha creado la tabla:
select * from libros;

-- Ejecutamos el procedimiento almacenado del sistema
"sp_help"
-- y el nombre del procedimiento almacenado para
verificar que existe
-- el procedimiento creado recientemente:
exec sp_help pa_crear_libros;

-- Necesitamos un procedimiento almacenado que muestre
los libros de los cuales
```

```
-- hay menos de 10. En primer lugar, lo eliminamos si
existe:
if object_id('pa_libros_limite_stock') is not null
 drop procedure pa_libros_limite_stock;

go

-- Creamos el procedimiento:
create proc pa_libros_limite_stock
  as
   select *from libros
   where cantidad <=10;

go

-- Ejecutamos el procedimiento almacenado del sistema
"sp_help"
-- junto al nombre del procedimiento creado
recientemente para verificar que existe:
exec sp_help pa_libros_limite_stock;

-- Lo ejecutamos:
exec pa_libros_limite_stock;

-- Modificamos algún registro y volvemos a ejecutar el
procedimiento:
update libros set cantidad=2 where codigo=4;
exec pa_libros_limite_stock;
```

121 - Procedimientos almacenados (eliminar)

Los procedimientos almacenados se eliminan con "drop procedure".
Sintaxis:

```
 drop procedure NOMBREPROCEDIMIENTO;
```

Eliminamos el procedimiento almacenado llamado "pa_libros_autor":

```
 drop procedure pa_libros_autor;
```

Si el procedimiento que queremos eliminar no existe, aparece un mensaje de error, para evitarlo, podemos emplear esta sintaxis:

```
 if object_id('NOMBREPROCEDIMIENTO') is not null
  drop procedure NOMBREPROCEDIMIENTO;
```

Eliminamos, si existe, el procedimiento "pa_libros_autor", si no existe, mostramos un mensaje:

```
 if object_id('pa_libros_autor') is not null
  drop procedure pa_libros_autor
 else
  select 'No existe el procedimiento "pa_libros_autor"';
```

"drop procedure" puede abreviarse con "drop proc".

Se recomienda ejecutar el procedimiento almacenado del sistema "sp_depends" para ver si algún objeto depende del procedimiento que deseamos eliminar.

Podemos eliminar una tabla de la cual dependa un procedimiento, SQL Server lo permite, pero luego, al ejecutar el procedimiento, aparecerá un mensaje de error porque la tabla referenciada no existe.

Servidor de SQL Server instalado en forma local.

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

```
-- Eliminamos, si existe, el procedimiento almacenado
"pa_crear_libros":
if object_id('pa_crear_libros') is not null
  drop procedure pa_crear_libros;

-- Verificamos que no existe ejecutando "sp_help":
exec sp_help pa_crear_libros;

go

-- Creamos el procedimiento:
create procedure pa_crear_libros
 as
  if object_id('libros')is not null
   drop table libros
  create table libros(
   codigo int identity,
   titulo varchar(40),
```

```
   autor varchar(30),
   editorial varchar(20),
   precio decimal(5,2),
   cantidad smallint,
   primary key(codigo)
  )
  insert into libros values('Uno','Richard
Bach','Planeta',15,5)
  insert into libros values('Ilusiones','Richard
Bach','Planeta',18,50)
  insert into libros values('El
aleph','Borges','Emece',25,9)
  insert into libros values('Aprenda PHP','Mario
Molina','Nuevo siglo',45,100)
  insert into libros values('Matematica estas
ahi','Paenza','Nuevo siglo',12,50)
  insert into libros values('Java en 10 minutos','Mario
Molina','Paidos',35,300);

go

-- Verificamos que existe:
exec sp_help pa_crear_libros;

-- Lo eliminamos sin corroborar su existencia:
drop proc pa_crear_libros;

-- Vemos si aparece en la lista de objetos que muestra
"sp_help":
exec sp_help pa_crear_libros;

-- Solicitamos su eliminación nuevamente (No existe,
aparece un mensaje de error):
drop proc pa_crear_libros;

-- Solicitamos su eliminación verificando si existe, si
no existe, mostramos un mensaje:
if object_id('pa_crear_libros') is not null
  drop proc pa_crear_libros
else
  select 'No existe el procedimiento "pa_crear_libros"';
```

122 - Procedimientos almacenados (parámetros de entrada)

Los procedimientos almacenados pueden recibir y devolver información; para ello se emplean parámetros, de entrada y salida, respectivamente.

Veamos los primeros. Los parámetros de entrada posibilitan pasar información a un procedimiento.
Para que un procedimiento almacenado admita parámetros de entrada se deben declarar variables como parámetros al crearlo. La sintaxis es:

```
create proc NOMBREPROCEDIMIENTO
 @NOMBREPARAMETRO TIPO =VALORPORDEFECTO
 as SENTENCIAS;
```

Los parámetros se definen luego del nombre del procedimiento, comenzando el nombre con un signo arroba (@). Los parámetros son locales al procedimiento, es decir, existen solamente dentro del mismo. Pueden declararse varios parámetros por procedimiento, se separan por comas.

Cuando el procedimiento es ejecutado, deben explicitarse valores para cada uno de los parámetros (en el orden que fueron definidos), a menos que se haya definido un valor por defecto, en tal caso, pueden omitirse. Pueden ser de cualquier tipo de dato (excepto cursor).

Luego de definir un parámetro y su tipo, opcionalmente, se puede especificar un valor por defecto; tal valor es el que asume el procedimiento al ser ejecutado si no recibe parámetros. Si no se coloca valor por defecto, un procedimiento definido con parámetros no puede ejecutarse sin valores para ellos. El valor por defecto puede ser "null" o una constante, también puede incluir comodines si el procedimiento emplea "like".

Creamos un procedimiento que recibe el nombre de un autor como parámetro para mostrar todos los libros del autor solicitado:

```
create procedure pa_libros_autor
 @autor varchar(30)
as
 select titulo, editorial,precio
  from libros
  where autor= @autor;
```

El procedimiento se ejecuta colocando "execute" (o "exec") seguido del nombre del procedimiento y un valor para el parámetro:

```
 exec pa_libros_autor 'Borges';
```

Creamos un procedimiento que recibe 2 parámetros, el nombre de un autor y el de una editorial:

```
 create procedure pa_libros_autor_editorial
  @autor varchar(30),
  @editorial varchar(20)
 as
  select titulo, precio
   from libros
   where autor= @autor and
   editorial=@editorial;
```

El procedimiento se ejecuta colocando "execute" (o "exec") seguido del nombre del procedimiento y los valores para los parámetros separados por comas:

```
 exec pa_libros_autor_editorial 'Richard
Bach','Planeta';
```

Los valores de un parámetro pueden pasarse al procedimiento mediante el nombre del parámetro o por su posición. La sintaxis anterior ejecuta el procedimiento pasando valores a los parámetros por posición. También podemos emplear la otra sintaxis en la cual pasamos valores a los parámetros por su nombre:

```
 exec pa_libros_autor_editorial @editorial='Planeta',
@autor='Richard Bach';
```

Cuando pasamos valores con el nombre del parámetro, el orden en que se colocan puede alterarse.

No podríamos ejecutar el procedimiento anterior sin valores para los parámetros. Si queremos ejecutar un procedimiento que permita omitir los valores para los parámetros debemos, al crear el procedimiento, definir valores por defecto para cada parámetro:

```
 create procedure pa_libros_autor_editorial2
  @autor varchar(30)='Richard Bach',
  @editorial varchar(20)='Planeta'
 as
  select titulo, autor,editorial,precio
   from libros
```

```
   where autor= @autor and
   editorial=@editorial;
```

Podemos ejecutar el procedimiento anterior sin enviarle valores, usará los predeterminados.

Si enviamos un solo parámetro a un procedimiento que tiene definido más de un parámetro sin especificar a qué parámetro corresponde (valor por posición), asume que es el primero. Es decir, SQL Server asume que los valores se dan en el orden que fueron definidos, no se puede interrumpir la secuencia.

Si queremos especificar solamente el segundo parámetro, debemos emplear la sintaxis de paso de valores a parámetros por nombre:

```
exec pa_libros_autor_editorial2 @editorial='Paidos';
```

Podemos emplear patrones de búsqueda en la consulta que define el procedimiento almacenado y utilizar comodines como valores por defecto:

```
create proc pa_libros_autor_editorial3
 @autor varchar(30) = '%',
 @editorial varchar(30) = '%'
as
 select titulo,autor,editorial,precio
  from libros
  where autor like @autor and
  editorial like @editorial;
```

La sentencia siguiente ejecuta el procedimiento almacenado "pa_libros_autor_editorial3" enviando un valor por posición, se asume que es el primero.

```
exec pa_libros_autor_editorial3 'P%';
```

La sentencia siguiente ejecuta el procedimiento almacenado "pa_libros_autor_editorial3" enviando un valor para el segundo parámetro, para el primer parámetro toma el valor por defecto:

```
exec pa_libros_autor_editorial3 @editorial='P%';
```

También podríamos haber tipeado:

```
exec pa_libros_autor_editorial3 default, 'P%';
```

Servidor de SQL Server instalado en forma local.

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

```
-- Trabajamos con la tabla "libros" de una librería.
-- Eliminamos la tabla si existe y la creamos
nuevamente:
if object_id('libros') is not null
  drop table libros;

create table libros(
  codigo int identity,
  titulo varchar(40),
  autor varchar(30),
  editorial varchar(20),
  precio decimal(5,2),
  primary key(codigo)
);

go

insert into libros values ('Uno','Richard
Bach','Planeta',15);
insert into libros values ('Ilusiones','Richard
Bach','Planeta',12);
insert into libros values ('El
aleph','Borges','Emece',25);
insert into libros values ('Aprenda PHP','Mario
Molina','Nuevo siglo',50);
insert into libros values ('Matematica estas
ahi','Paenza','Nuevo siglo',18);
insert into libros values ('Puente al infinito','Bach
Richard','Sudamericana',14);
insert into libros values ('Antología','J. L.
Borges','Paidos',24);
insert into libros values ('Java en 10 minutos','Mario
Molina','Siglo XXI',45);
insert into libros values ('Cervantes y el
quijote','Borges- Casares','Planeta',34);

-- Eliminamos el procedimiento almacenado
"pa_libros_autor" si existe:
if object_id('pa_libros_autor') is not null
```

```
  drop procedure pa_libros_autor;

go

-- Creamos el procedimiento para que reciba el nombre de
un autor y
-- muestre todos los libros del autor solicitado:
create procedure pa_libros_autor
  @autor varchar(30)
 as
  select titulo, editorial,precio
   from libros
   where autor= @autor;

go

-- Ejecutamos el procedimiento:
exec pa_libros_autor 'Richard Bach';

-- Empleamos la otra sintaxis (por nombre) y pasamos
otro valor:
exec pa_libros_autor @autor='Borges';

-- Eliminamos, si existe, el procedimiento
"pa_libros_autor_editorial":
if object_id('pa_libros_autor_editorial') is not null
  drop procedure pa_libros_autor_editorial;

go

-- Creamos un procedimiento "pa_libros_autor_editorial"
que recibe 2 parámetros,
-- el nombre de un autor y el de una editorial:
create procedure pa_libros_autor_editorial
  @autor varchar(30),
  @editorial varchar(20)
 as
  select titulo, precio
   from libros
   where autor= @autor and
   editorial=@editorial;

go
```

```
-- Ejecutamos el procedimiento enviando los parámetros
por posición:
exec pa_libros_autor_editorial 'Richard Bach','Planeta';

-- Ejecutamos el procedimiento enviando otros valores y
lo hacemos por nombre
--(Si ejecutamos el procedimiento omitiendo los
parámetros, aparecerá un mensaje de error.):
exec pa_libros_autor_editorial
@autor='Borges',@editorial='Emece';

-- Eliminamos, si existe, el procedimiento
"pa_libros_autor_editorial2":
if object_id('pa_libros_autor_editorial2') is not null
  drop procedure pa_libros_autor_editorial2;

go

-- Creamos el procedimiento almacenado
"pa_libros_autor_editorial2" que recibe los mismos
-- parámetros, esta vez definimos valores por defecto
para cada parámetro:
create procedure pa_libros_autor_editorial2
  @autor varchar(30)='Richard Bach',
  @editorial varchar(20)='Planeta'
 as
  select titulo,autor,editorial,precio
   from libros
   where autor= @autor and
   editorial=@editorial;

go

-- Ejecutamos el procedimiento anterior sin enviarle
valores para verificar que usa
-- los valores por defecto (Muestra los libros de
"Richard Bach" y editorial
-- "Planeta" (valores por defecto)):
exec pa_libros_autor_editorial2;

-- Enviamos un solo parámetro al procedimiento (SQL
Server asume que es el primero,
-- y no hay registros cuyo autor sea "Planeta"):
exec pa_libros_autor_editorial2 'Planeta';
```

```
-- Especificamos el segundo parámetro, enviando
parámetros por nombre:
exec pa_libros_autor_editorial2 @editorial='Planeta';

-- Ejecutamos el procedimiento enviando parámetros por
nombre en distinto orden:
exec pa_libros_autor_editorial2 @editorial='Nuevo
siglo',@autor='Paenza';

-- Definimos un procedimiento empleando patrones de
búsqueda
-- (antes verificamos si existe para eliminarlo):
if object_id('pa_libros_autor_editorial3') is not null
  drop procedure pa_libros_autor_editorial3;

go

 create proc pa_libros_autor_editorial3
  @autor varchar(30) = '%',
  @editorial varchar(30) = '%'
 as
  select titulo,autor,editorial,precio
   from libros
   where autor like @autor and
   editorial like @editorial;

go

-- Ejecutamos el procedimiento enviando parámetro por
posición, asume que es el primero:
exec pa_libros_autor_editorial3 'P%';

-- Ejecutamos el procedimiento especificando que el
valor corresponde al segundo parámetro:
exec pa_libros_autor_editorial3 @editorial='P%';

-- La sentencia siguiente muestra lo mismo que la
anterior:
exec pa_libros_autor_editorial3 default, 'P%';
```

123 - Procedimientos almacenados (parámetros de salida)

Dijimos que los procedimientos almacenados pueden devolver información; para ello se emplean parámetros de salida. El valor se retorna a quien realizó la llamada con parámetros de salida. Para que un procedimiento almacenado devuelva un valor se debe declarar una variable con la palabra clave "output" al crear el procedimiento:

```
create procedure NOMBREPROCEDIMIENTO
 @PARAMETROENTRADA TIPO =VALORPORDEFECTO,
 @PARAMETROSALIDA TIPO=VALORPORDEFECTO output
 as
  SENTENCIAS
  select @PARAMETROSALIDA=SENTENCIAS;
```

Los parámetros de salida pueden ser de cualquier tipo de datos, excepto text, ntext e image.

Creamos un procedimiento almacenado al cual le enviamos 2 números y retorna el promedio:

```
create procedure pa_promedio
 @n1 decimal(4,2),
 @n2 decimal(4,2),
 @resultado decimal(4,2) output
 as
  select @resultado=(@n1+@n2)/2;
```

Al ejecutarlo también debe emplearse "output":

```
 declare @variable decimal(4,2)
 execute pa_promedio 5,6, @variable output
 select @variable;
```

Declaramos una variable para guardar el valor devuelto por el procedimiento; ejecutamos el procedimiento enviándole 2 valores y mostramos el resultado.

La instrucción que realiza la llamada al procedimiento debe contener un nombre de variable para almacenar el valor retornado.

Creamos un procedimiento almacenado que muestre los títulos, editorial y precio de los libros de un determinado autor (enviado como parámetro de entrada) y nos retorne la suma y el promedio de los precios de todos los libros del autor enviado:

```
 create procedure pa_autor_sumaypromedio
  @autor varchar(30)='%',
  @suma decimal(6,2) output,
  @promedio decimal(6,2) output
  as
   select titulo,editorial,precio
   from libros
   where autor like @autor
  select @suma=sum(precio)
   from libros
   where autor like @autor
  select @promedio=avg(precio)
   from libros
   where autor like @autor;
```

Ejecutamos el procedimiento y vemos el contenido de las variables en las que almacenamos los parámetros de salida del procedimiento:

```
 declare @s decimal(6,2), @p decimal(6,2)
 execute pa_autor_sumaypromedio 'Richard Bach', @s
output, @p output
 select @s as total, @p as promedio;
```

Servidor de SQL Server instalado en forma local.

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

```
-- Eliminamos el procedimiento almacenado "pa_promedio",
si existe:
if object_id('pa_promedio') is not null
  drop proc pa_promedio;

go

-- Creamos un procedimiento almacenado al cual le
enviamos
-- 2 números decimales y retorna el promedio:
create procedure pa_promedio
  @n1 decimal(4,2),
  @n2 decimal(4,2),
  @resultado decimal(4,2) output
  as
   select @resultado=(@n1+@n2)/2;
```

```
go

-- Lo ejecutamos enviando diferentes valores:
declare @variable decimal(4,2)
exec pa_promedio 5,6, @variable output
select @variable

exec pa_promedio 5.3,4.7, @variable output
select @variable

exec pa_promedio 9,10, @variable output
select @variable;

-- Trabajamos con la tabla "libros" de una librería.
-- Eliminamos la tabla si existe y la creamos
nuevamente:
if object_id('libros') is not null
  drop table libros;

go

create table libros(
  codigo int identity,
  titulo varchar(40),
  autor varchar(30),
  editorial varchar(20),
  precio decimal(5,2),
  primary key(codigo)
);

go

insert into libros values ('Uno','Richard
Bach','Planeta',15);
insert into libros values ('Ilusiones','Richard
Bach','Planeta',12);
insert into libros values ('El
aleph','Borges','Emece',25);
insert into libros values ('Aprenda PHP','Mario
Molina','Nuevo siglo',50);
insert into libros values ('Matematica estas
ahi','Paenza','Nuevo siglo',18);
insert into libros values ('Puente al infinito','Richard
Bach','Sudamericana',14);
```

```
insert into libros values ('Antología','J. L.
Borges','Paidos',24);
insert into libros values ('Java en 10 minutos','Mario
Molina','Siglo XXI',45);
insert into libros values
('Antología','Borges','Planeta',34);

-- Eliminamos el procedimiento almacenado
"pa_autor_sumaypromedio", si existe:
if object_id('pa_autor_sumaypromedio') is not null
  drop proc pa_autor_sumaypromedio;

go

-- Creamos un procedimiento almacenado que muestre los
títulos, editorial y precio
-- de los libros de un determinado autor (enviado como
parámetro de entrada)
-- y nos retorne la suma y el promedio de los precios de
todos los libros del autor enviado:
create procedure pa_autor_sumaypromedio
  @autor varchar(30)='%',
  @suma decimal(6,2) output,
  @promedio decimal(6,2) output
  as
   select titulo,editorial,precio
   from libros
   where autor like @autor
  select @suma=sum(precio)
   from libros
   where autor like @autor
  select @promedio=avg(precio)
   from libros
   where autor like @autor;

go

-- Ejecutamos el procedimiento enviando distintos
valores:
declare @s decimal(6,2), @p decimal(6,2)
exec pa_autor_sumaypromedio 'Richard Bach', @s output,
@p output
select @s as total, @p as promedio
```

```
exec pa_autor_sumaypromedio 'Borges', @s output, @p
output
select @s as total, @p as promedio

exec pa_autor_sumaypromedio 'Mario Molina', @s output,
@p output
select @s as total, @p as promedio;

go

-- Ejecutamos el procedimiento sin pasar el parámetro
para autor.
-- Recuerde que en estos casos debemos colocar los
nombres de las variables.
declare @s decimal(6,2), @p decimal(6,2)
exec pa_autor_sumaypromedio @suma=@s output,@promedio=
@p output
select @s as total, @p as promedio;
```

124 - Procedimientos almacenados (return)

La instrucción "return" sale de una consulta o procedimiento y todas las instrucciones posteriores no son ejecutadas.

Creamos un procedimiento que muestre todos los libros de un autor determinado que se ingresa como parámetro. Si no se ingresa un valor, o se ingresa "null", se muestra un mensaje y se sale del procedimiento:

```
 create procedure pa_libros_autor
  @autor varchar(30)=null
 as
 if @autor is null
 begin
  select 'Debe indicar un autor'
  return
 end;
 select titulo from  libros where autor = @autor;
```

Si al ejecutar el procedimiento enviamos el valor "null" o no pasamos valor, con lo cual toma el valor por defecto "null", se muestra un mensaje y se sale; en caso contrario, ejecuta la consulta luego del "else".

"return" puede retornar un valor entero.

Un procedimiento puede retornar un valor de estado para indicar si se ha ejecutado correctamente o no.

Creamos un procedimiento almacenado que ingresa registros en la tabla "libros". Los parámetros correspondientes al título y autor DEBEN ingresarse con un valor distinto de "null", los demás son opcionales. El procedimiento retorna "1" si la inserción se realiza, es decir, si se ingresan valores para título y autor y "0", en caso que título o autor sean nulos:

```
 create procedure pa_libros_ingreso
  @titulo varchar(40)=null,
  @autor varchar(30)=null,
  @editorial varchar(20)=null,
  @precio decimal(5,2)=null
 as
 if (@titulo is null) or (@autor is null)
  return 0
 else
 begin
  insert into libros values
(@titulo,@autor,@editorial,@precio)
  return 1
 end;
```

Para ver el resultado, debemos declarar una variable en la cual se almacene el valor devuelto por el procedimiento; luego, ejecutar el procedimiento asignándole el valor devuelto a la variable, finalmente mostramos el contenido de la variable:

```
 declare @retorno int
 exec @retorno=pa_libros_ingreso 'Alicia en el
pais...','Lewis Carroll'
 select 'Ingreso realizado=1' = @retorno
 exec @retorno=pa_libros_ingreso
 select 'Ingreso realizado=1' = @retorno;
```

También podríamos emplear un "if" para controlar el valor de la variable de retorno:

```
 declare @retorno int;
 exec @retorno=pa_libros_ingreso 'El gato con
botas','Anónimo'
 if @retorno=1 select 'Registro ingresado'
```

```
 else select 'Registro no ingresado porque faltan
datos';
```

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),
  autor varchar(30),
  editorial varchar(20),
  precio decimal(5,2),
  primary key(codigo)
);

go

insert into libros values ('Uno','Richard
Bach','Planeta',15);
insert into libros values ('Ilusiones','Richard
Bach','Planeta',12);
insert into libros values ('El
aleph','Borges','Emece',25);
insert into libros values ('Aprenda PHP','Mario
Molina','Nuevo siglo',50);
insert into libros values ('Matematica estas
ahi','Paenza','Nuevo siglo',18);
insert into libros values ('Puente al infinito','Richard
Bach','Sudamericana',14);
insert into libros values ('Antología','J. L.
Borges','Paidos',24);
insert into libros values ('Java en 10 minutos','Mario
Molina','Siglo XXI',45);
insert into libros values
('Antología','Borges','Planeta',34);

-- Eliminamos el procedimiento llamado
"pa_libros_autor", si existe:
if object_id('pa_libros_autor') is not null
  drop procedure pa_libros_autor;
```

```
go

-- Creamos un procedimiento que muestre todos los libros
de un autor determinado
-- que se ingresa como parámetro. Si no se ingresa un
valor, o se ingresa "null",
-- se muestra un mensaje y se sale del procedimiento:
create procedure pa_libros_autor
  @autor varchar(30)=null
 as
 if @autor is null
 begin
  select 'Debe indicar un autor'
  return
 end
 select titulo from  libros where autor = @autor;

go

-- Ejecutamos el procedimiento con parámetro:
exec pa_libros_autor 'Borges';

-- Ejecutamos el procedimiento sin parámetro:
exec pa_libros_autor;

-- Eliminamos el procedimiento "pa_libros_ingreso", si
existe:
if object_id('pa_libros_ingreso') is not null
  drop procedure pa_libros_ingreso;

go

-- Creamos un procedimiento almacenado que ingresa
registros en la tabla "libros".
-- Los parámetros correspondientes al título y autor
DEBEN ingresarse con un valor
-- distinto de "null", los demás son opcionales.
-- El procedimiento retorna "1" si la inserción se
realiza (si se ingresan valores
-- para título y autor) y "0", en caso que título o
autor sean nulos:
create procedure pa_libros_ingreso
  @titulo varchar(40)=null,
```

```
  @autor varchar(30)=null,
  @editorial varchar(20)=null,
  @precio decimal(5,2)=null
 as
 if (@titulo is null) or (@autor is null)
  return 0
 else
 begin
  insert into libros values
(@titulo,@autor,@editorial,@precio)
  return 1
 end;

go

-- Declaramos una variable en la cual almacenaremos el
valor devuelto,
-- ejecutamos el procedimiento enviando los dos
parámetros obligatorios
-- y vemos el contenido de la variable:
declare @retorno int
exec @retorno=pa_libros_ingreso 'Alicia en el
pais...','Lewis Carroll'
select 'Ingreso realizado=1' = @retorno;

select * from libros;

go

-- Ejecutamos los mismos pasos, pero esta vez no
enviamos valores al procedimiento
-- (El procedimiento retornó "0", lo cual indica que el
registro no fue ingresado.):
declare @retorno int
exec @retorno=pa_libros_ingreso
select 'Ingreso realizado=1' = @retorno;

select * from libros;

go

-- Empleamos un "if" para controlar el valor de la
variable de retorno.
```

```
-- Enviando al procedimiento valores para los parámetros
obligatorios:
declare @retorno int
exec @retorno=pa_libros_ingreso 'El gato con
botas','Anónimo'
if @retorno=1 select 'Registro ingresado'
 else select 'Registro no ingresado porque faltan
datos';

select * from libros;

go

declare @retorno int
exec @retorno=pa_libros_ingreso
if @retorno=1 select 'Registro ingresado'
 else select 'Registro no ingresado porque faltan
datos';

select * from libros;
```

125 - Procedimientos almacenados (información)

Los procedimientos almacenados son objetos, así que para obtener información de ellos pueden utilizar los siguientes procedimientos almacenados del sistema y las siguientes tablas:

- "sp_help": sin parámetros nos muestra todos los objetos de la base de datos seleccionados, incluidos los procedimientos. En la columna "Object_type" aparece "procedimiento almacenado" si es un procedimiento almacenado. Si le enviamos como argumento el nombre de un procedimiento, obtenemos la fecha de creación e información sobre sus parámetros.

- "sp_helptext": seguido del nombre de un procedimiento almacenado nos muestra el texto que define el procedimiento, excepto si ha sido encriptado.

- "sp_stored_procedures": muestra todos los procedimientos almacenados, los propietarios, etc. Este procedimiento almacenado puede recibir 3 parámetros: @sp_name (nombre, nvarchar, admite comodines para búsqueda de patrones), @sp_owner (propietario, nvarchar, admite comodines) y @calificador (nombre de la base de datos). Por ejemplo,

podemos ver todos los procedimientos almacenados creados por nosotros con esta sentencia:

```
 exec sp_stored_procedures @ sp_name = 'pa_%';
```

- "sp_depends": seguido del nombre de un objeto, nos devuelve 2 resultados: 1) nombre, tipo, campos, etc. de los objetos de los cuales dependen el objeto enviado y 2) nombre y tipo de los objetos que dependen del objeto nombrado. Por ejemplo, ejecutamos "sp_depends" seguido del nombre de un procedimiento:

```
 exec sp_depends pa_autor_promedio;
```

aparecen las tablas (y demás objetos) de las cuales dependen el procedimiento, es decir, las tablas referenciadas en el mismo. Podemos ejecutar el procedimiento seguido del nombre de una tabla:

```
 exec sp_depends libros;
```

aparecen los procedimientos (y demás objetos) que dependen de ella.

- La tabla del sistema "sysobjects": muestra nombre y varios datos de todos los objetos de la base de datos actual. La columna "xtype" indica el tipo de objeto. Si es un procedimiento almacenado, muestra "P". Ejemplo:

```
 seleccionar * de sysobjects;
```

Si queremos ver todos los procedimientos almacenados creados por nosotros, podemos tipear:

```
 seleccionar * de sysobjects
  donde xtype = 'P' y-- tipo procedimiento
  nombre como 'pa%'; - búsqueda con comodín
```

Servidor de SQL Server instalado en forma local.

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

```
si object_id ('libros') no es nulo
  caída de libros de mesa;

crear libros de tabla
  codigo int identidad,
  titulo varchar (40),
```

```
  autor varchar (30),
  editorial varchar (20),
  precio decimal (5,2),
  clave primaria (codigo)
);

ir

insertar en libros valores ('Uno', 'Richard Bach',
'Planeta', 15);
insertar en libros valores ('Ilusiones', 'Richard Bach',
'Planeta', 12);
insertar en libros valores ('El aleph', 'Borges',
'Emece', 25);
insertar en libros valores ('Aprenda PHP', 'Mario
Molina', 'Nuevo siglo', 50);
insertar en libros valores ('Matematica estas ahi',
'Paenza', 'Nuevo siglo', 18);
insertar en libros valores ('Puente al infinito',
'Richard Bach', 'Sudamericana', 14);
insertar en libros valores ('Antología', 'JL Borges',
'Paidos', 24);
insertar en libros valores ('Java en 10 minutos', 'Mario
Molina', 'Siglo XXI', 45);
insertar en libros valores ('Antología', 'Borges',
'Planeta', 34);

si object_id ('pa_autor_promedio') no es nulo
  drop proc pa_autor_promedio;

ir

- Creamos un procedimiento almacenado para que recibió
el nombre de un autor
- y nos retorne el promedio de los precios de todos los
libros de tal autor:
crear procedimiento pa_autor_promedio
  @autor varchar (30) = '%',
  Salida @promedio decimal (6,2)
  como
  seleccione @ promedio = avg (precio)
   de libros
   donde autor como @autor;
```

```
ir

exec sp_help pa_autor_promedio;

exec sp_helptext pa_autor_promedio;

exec sp_stored_procedures;

exec sp_stored_procedures 'pa_%';

exec sp_depends pa_autor_promedio;

exec sp_depends libros;

seleccionar * de sysobjects;

seleccionar * de sysobjects
  donde xtype = 'P' y-- tipo procedimiento
  nombre como 'pa%' - búsqueda con comodín;

drop proc pa_autor_promedio;

exec sp_depends libros;
```