- luego de "references" indicamos el nombre de la tabla referenciada y el campo que es clave primaria en la misma, a la cual hace referencia la clave foránea. La tabla referenciada debe tener definida una restricción "primary key" o "unique"; si no la tiene, aparece un mensaje de error.

Para agregar una restricción "foreign key" al campo "codigoeditorial" de "libros", tipeamos:

```
 alter table libros
 add constraint FK_libros_codigoeditorial
  foreign key (codigoeditorial)
  references editoriales(codigo);
```

En el ejemplo implementamos una restricción "foreign key" para asegurarnos que el código de la editorial de la tabla "libros" ("codigoeditorial") esté asociada con un código válido en la tabla "editoriales" ("codigo").

Cuando agregamos cualquier restricción a una tabla que contiene información, SQL Server controla los datos existentes para confirmar que cumplen con la restricción, si no los cumple, la restricción no se aplica y aparece un mensaje de error. Por ejemplo, si intentamos agregar una restricción "foreign key" a la tabla "libros" y existe un libro con un valor de código para editorial que no existe en la tabla "editoriales", la restricción no se agrega.

Actúa en inserciones. Si intentamos ingresar un registro (un libro) con un valor de clave foránea (codigoeditorial) que no existe en la tabla referenciada (editoriales), SQL server muestra un mensaje de error. Si al ingresar un registro (un libro), no colocamos el valor para el campo clave foránea (codigoeditorial), almacenará "null", porque esta restricción permite valores nulos (a menos que se haya especificado lo contrario al definir el campo).

Actúa en eliminaciones y actualizaciones. Si intentamos eliminar un registro o modificar un valor de clave primaria de una tabla si una clave foránea hace referencia a dicho registro, SQL Server no lo permite (excepto si se permite la acción en cascada, tema que veremos posteriormente). Por ejemplo, si intentamos eliminar una editorial a la que se hace referencia en "libros", aparece un mensaje de error.

Esta restricción (a diferencia de "primary key" y "unique") no crea índice automaticamente.

La cantidad y tipo de datos de los campos especificados luego de "foreign key" DEBEN coincidir con la cantidad y tipo de datos de los campos de la cláusula "references".

Esta restricción se puede definir dentro de la misma tabla (lo veremos más adelante) o entre distintas tablas.

Una tabla puede tener varias restricciones "foreign key".

No se puede eliminar una tabla referenciada en una restricción "foreign key", aparece un mensaje de error.

Una restriccion "foreign key" no puede modificarse, debe eliminarse y volverse a crear.

Para ver información acerca de esta restricción podemos ejecutar el procedimiento almacenado "sp_helpconstraint" junto al nombre de la tabla. Nos muestra el tipo, nombre, la opción para eliminaciones y actualizaciones, el estado (temas que veremos más adelante), el nombre del campo y la tabla y campo que referencia.

También informa si la tabla es referenciada por una clave foránea.

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),
  autor varchar(30),
```

codigoeditorial tinyint

```
);
create table editoriales(
  codigo tinyint,
  nombre varchar(20),
  primary key (codigo)
```

```
);

go

insert into editoriales values(1,'Emece');
insert into editoriales values(2,'Planeta');
insert into editoriales values(3,'Siglo XXI');

insert into libros values('El aleph','Borges',1);
insert into libros values('Martin Fierro','Jose
Hernandez',2);
insert into libros values('Aprenda PHP','Mario
Molina',2);

-- Agregamos una restricción "foreign key" a la tabla
"libros":
alter table libros
  add constraint FK_libros_codigoeditorial
  foreign key (codigoeditorial)
  references editoriales(codigo);

insert into libros default values;

exec sp_helpconstraint libros;

exec sp_helpconstraint editoriales;
```

77 - Restricciones foreign key en la misma tabla

La restricción "foreign key", que define una referencia a un campo con una restricción "primary key" o "unique" se puede definir entre distintas tablas (como hemos aprendido) o dentro de la misma tabla.

Veamos un ejemplo en el cual definimos esta restricción dentro de la misma tabla.

Una mutual almacena los datos de sus afiliados en una tabla llamada "afiliados". Algunos afiliados inscriben a sus familiares. La tabla contiene un campo que hace referencia al afiliado que lo incorporó a la mutual, del cual dependen.

La estructura de la tabla es la siguiente:

```
 create table afiliados(
  numero int identity not null,
  documento char(8) not null,
  nombre varchar(30),
  afiliadotitular int,
  primary key (documento),
  unique (numero)
 );
```

En caso que un afiliado no haya sido incorporado a la mutual por otro afiliado, el campo "afiliadotitular" almacenará "null".

Establecemos una restricción "foreign key" para asegurarnos que el número de afiliado que se ingrese en el campo "afiliadotitular" exista en la tabla "afiliados":

```
 alter table afiliados
  add constraint FK_afiliados_afiliadotitular
  foreign key (afiliadotitular)
  references afiliados (numero);
```

La sintaxis es la misma, excepto que la tabla se autoreferencia.

Luego de aplicar esta restricción, cada vez que se ingrese un valor en el campo "afiliadotitular", SQL Server controlará que dicho número exista en la tabla, si no existe, mostrará un mensaje de error.

Si intentamos eliminar un afiliado que es titular de otros afiliados, no se podrá hacer, a menos que se haya especificado la acción en cascada (próximo tema).

Servidor de SQL Server instalado en forma local.

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

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

create table afiliados(
  numero int identity not null,
  documento char(8) not null,
  nombre varchar(30),
  afiliadotitular int,
```

```
  primary key (documento),
  unique (numero)
);

go

alter table afiliados
  add constraint FK_afiliados_afiliadotitular
  foreign key (afiliadotitular)
  references afiliados (numero);

insert into afiliados values('22222222','Perez
Juan',null);
insert into afiliados values('23333333','Garcia
Maria',null);
insert into afiliados values('24444444','Lopez
Susana',null);
insert into afiliados values('30000000','Perez
Marcela',1);
insert into afiliados values('31111111','Morales
Luis',1);
insert into afiliados values('32222222','Garcia
Maria',2);

delete from afiliados where numero=5;

exec sp_helpconstraint afiliados;
```

79 - Restricciones foreign key deshabilitar y eliminar (with check - nocheck)

Sabemos que si agregamos una restricción a una tabla que contiene datos, SQL Server los controla para asegurarse que cumplen con la restricción; es posible deshabilitar esta comprobación.

Podemos hacerlo al momento de agregar la restricción a una tabla con datos, incluyendo la opción "with nocheck" en la instrucción "alter table"; si se emplea esta opción, los datos no van a cumplir la restricción.

Se pueden deshabilitar las restricciones "check" y "foreign key", a las demás se las debe eliminar.

La sintaxis básica al agregar la restriccción "foreign key" es la siguiente:

```
alter table NOMBRETABLA1
 with OPCIONDECHEQUEO
 add constraint NOMBRECONSTRAINT
 foreign key (CAMPOCLAVEFORANEA)
 references NOMBRETABLA2 (CAMPOCLAVEPRIMARIA)
 on update OPCION
 on delete OPCION;
```

La opción "with OPCIONDECHEQUEO" especifica si se controlan los datos existentes o no con "check" y "nocheck" respectivamente. Por defecto, si no se especifica, la opción es "check".

En el siguiente ejemplo agregamos una restricción "foreign key" que controla que todos los códigos de editorial tengan un código válido, es decir, dicho código exista en "editoriales". La restricción no se aplica en los datos existentes pero si en los siguientes ingresos, modificaciones y actualizaciones:

```
alter table libros
with nocheck
add constraint FK_libros_codigoeditorial
foreing key (codigoeditorial)
references editoriales(codigo);
```

La comprobación de restricciones se puede deshabilitar para modificar, eliminar o agregar datos a una tabla sin comprobar la restricción. La sintaxis general es:

```
alter table NOMBRETABLA
OPCIONDECHEQUEO constraint NOMBRERESTRICCION;
```

En el siguiente ejemplo deshabilitamos la restricción creada anteriormente:

```
alter table libros
nocheck constraint FK_libros_codigoeditorial;
```

Para habilitar una restricción deshabilitada se ejecuta la misma instrucción pero con la cláusula "check" o "check all":

```
alter table libros
 check constraint FK_libros_codigoeditorial;
```

Si se emplea "check constraint all" no se coloca nombre de restricciones, habilita todas las restricciones que tiene la tabla nombrada ("check" y "foreign key").

Para saber si una restricción está habilitada o no, podemos ejecutar el procedimiento almacenado "sp_helpconstraint" y entenderemos lo que informa la columna "status_enabled".

Entonces, las cláusulas "check" y "nocheck" permiten habilitar o deshabilitar restricciones "foreign key" (y "check"). Pueden emplearse para evitar la comprobación de datos existentes al crear la restricción o para deshabilitar la comprobación de datos al ingresar, actualizar y eliminar algún registro que infrinja la restricción.

Podemos eliminar una restricción "foreign key" con "alter table". La sintaxis básica es la misma que para cualquier otra restricción:

```
 alter table TABLA
  drop constraint NOMBRERESTRICCION;
```

Eliminamos la restricción de "libros":

```
 alter table libros
  drop constraint FK_libros_codigoeditorial;
```

No se puede eliminar una tabla si una restricción "foreign key" hace referencia a ella.

Cuando eliminamos una tabla que tiene una restricción "foreign key", la restricción también se elimina.

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 not null,
  titulo varchar(40),
```

```
  autor varchar(30),
  codigoeditorial tinyint,
  primary key (codigo)
);
create table editoriales(
  codigo tinyint not null,
  nombre varchar(20),
  primary key (codigo)
);

go

insert into editoriales values(1,'Planeta');
insert into editoriales values(2,'Emece');
insert into editoriales values(3,'Paidos');

insert into libros values(1,'Uno','Richard Bach',1);
insert into libros values(2,'El aleph','Borges',2);
insert into libros values(3,'Aprenda PHP','Mario
Molina',5);

-- Agregamos una restricción "foreign key" a la tabla
"libros" para evitar que se ingresen
-- códigos de editoriales inexistentes en "editoriales".
-- Incluimos la opción "with nocheck" para evitar la
comprobación de la restricción en los
-- datos existentes (note que hay un libro que tiene un
código de editorial inválido):
alter table libros
 with nocheck
 add constraint FK_libros_codigoeditorial
 foreign key (codigoeditorial)
 references editoriales(codigo);

-- Para poder ingresar, modificar o eliminar datos a una
tabla sin que SQL Server
-- compruebe la restricción debemos deshabilitarla:
alter table libros
 nocheck constraint FK_libros_codigoeditorial;

-- Veamos si la restricción está habilitada o no:
exec sp_helpconstraint libros;

-- Veamos las restricciones de "editoriales":
```

```
exec sp_helpconstraint editoriales;

-- Ahora podemos ingresar un registro en "libros" con
código inválido:
insert into libros values(4,'Ilusiones','Richard
Bach',6);

-- También podemos modificar:
update editoriales set codigo=8 where codigo=1;

-- También realizar eliminaciones:
delete from editoriales where codigo=2;

-- Habilitamos la restricción:
alter table libros
  check constraint FK_libros_codigoeditorial;

-- Veamos si la restricción está habilitada o no:
exec sp_helpconstraint libros;

-- Eliminamos la restricción:
alter table libros
  drop constraint FK_libros_codigoeditorial;

exec sp_helpconstraint libros;

exec sp_helpconstraint editoriales;
```

80 - Restricciones foreign key (información)

El procedimiento almacenado "sp_helpconstraint" devuelve las siguientes columnas:

- constraint_type: tipo de restricción. Si es una restricción de campo (default o check) indica sobre qué campo fue establecida. Si es de tabla (primary key o unique) indica el tipo de índice creado. Si es una "foreign key" lo indica.

- constraint_name: nombre de la restricción.

- delete_action: solamente es aplicable para restricciones de tipo "foreign key". Indica si la acción de eliminación actúa, no actúa o es en cascada.

Indica "n/a" en cualquier restricción para la que no se aplique; "No Action" si no actúa y "Cascade" si es en cascada.

- update_action: sólo es aplicable para restricciones de tipo "foreign key". Indica si la acción de actualización es: No Action, Cascade, or n/a. Indica "n/a" en cualquier restricción para la que no se aplique.

- status_enabled: solamente es aplicable para restricciones de tipo "check" y "foreign key". Indica si está habilitada (Enabled) o no (Disabled). Indica "n/a" en cualquier restricción para la que no se aplique.

- status_for_replication: solamente es aplicable para restricciones de tipo "check" y "foreign key". Indica "n/a" en cualquier restricción para la que no se aplique.

- constraint_keys: Si es una restricción "default" muestra la condición de chequeo; si es una restricción "default", el valor por defecto; si es una "primary key", "unique" o "foreign key" muestra el/ los campos a los que se aplicaron la restricción. En caso de valores predeterminados y reglas, el texto que lo define.

81 - Restricciones al crear la tabla

Hasta el momento hemos agregado restricciones a tablas existentes con "alter table" (manera aconsejada), también pueden establecerse al momento de crear una tabla (en la instrucción "create table").

Podemos aplicar restricciones a nivel de campo (restricción de campo) o a nivel de tabla (restricción de tabla).

En el siguiente ejemplo creamos la tabla "libros" con varias restricciones:

```
create table libros(
 codigo int identity,
 titulo varchar(40),
 codigoautor int not null,
 codigoeditorial tinyint not null,
 precio decimal(5,2)
  constraint DF_precio default (0),
 constraint PK_libros_codigo
  primary key clustered (codigo),
 constraint UQ_libros_tituloautor
```

```
    unique (titulo,codigoautor),
  constraint FK_libros_editorial
   foreign key (codigoeditorial)
   references editoriales(codigo)
   on update cascade,
  constraint FK_libros_autores
   foreign key (codigoautor)
   references autores(codigo)
   on update cascade,
  constraint CK_precio_positivo check (precio>=0)
);
```

En el ejemplo anterior creamos:

- una restricción "default" para el campo "precio" (restricción a nivel de campo);

- una restricción "primary key" con índice agrupado para el campo "codigo" (a nivel de tabla);

- una restricción "unique" con índice no agrupado (por defecto) para los campos "titulo" y "codigoautor" (a nivel de tabla);

- una restricción "foreign key" para establecer el campo "codigoeditorial" como clave externa que haga referencia al campo "codigo" de "editoriales y permita actualizaciones en cascada y no eliminaciones (por defecto "no action");

- una restricción "foreign key" para establecer el campo "codigoautor" como clave externa que haga referencia al campo "codigo" de "autores" y permita actualizaciones en cascada y no eliminaciones;

- una restricción "check" para el campo "precio" que no admita valores negativos;

Si definimos una restricción "foreign key" al crear una tabla, la tabla referenciada debe existir.

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

create table editoriales(
  codigo tinyint not null,
  nombre varchar(30),
  constraint PK_editoriales primary key (codigo)
);

create table autores(
  codigo int not null
   constraint CK_autores_codigo check (codigo>=0),
  nombre varchar(30) not null,
  constraint PK_autores_codigo
   primary key (codigo),
  constraint UQ_autores_nombre
    unique (nombre),
);

create table libros(
  codigo int identity,
  titulo varchar(40),
  codigoautor int not null,
  codigoeditorial tinyint not null,
  precio decimal(5,2)
   constraint DF_libros_precio default (0),
  constraint PK_libros_codigo
   primary key clustered (codigo),
  constraint UQ_libros_tituloautor
    unique (titulo,codigoautor),
  constraint FK_libros_editorial
   foreign key (codigoeditorial)
   references editoriales(codigo)
   on update cascade,
  constraint FK_libros_autores
   foreign key (codigoautor)
   references autores(codigo)
   on update cascade,
  constraint CK_libros_precio_positivo check (precio>=0)
);
```

```
go

exec sp_helpconstraint editoriales;

exec sp_helpconstraint autores;

exec sp_helpconstraint libros;
```

82 - Unión

El operador "union" combina el resultado de dos o más instrucciones "select" en un único resultado.

Se usa cuando los datos que se quieren obtener pertenecen a distintas tablas y no se puede acceder a ellos con una sola consulta.

Es necesario que las tablas referenciadas tengan tipos de datos similares, la misma cantidad de campos y el mismo orden de campos en la lista de selección de cada consulta. No se incluyen las filas duplicadas en el resultado, a menos que coloque la opción "all".

Se deben especificar los nombres de los campos en la primera instrucción "select".

Puede emplear la cláusula "order by".

Puede dividir una consulta compleja en varias consultas "select" y luego emplear el operador "union" para combinarlas.

Una academia de enseñanza almacena los datos de los alumnos en una tabla llamada "alumnos" y los datos de los profesores en otra denominada "profesores".
La academia necesita el nombre y domicilio de profesores y alumnos para enviarles una tarjeta de invitación.

Para obtener los datos necesarios de ambas tablas en una sola consulta necesitamos realizar una unión:

```
 select nombre, domicilio from alumnos
  union
   select nombre, domicilio from profesores;
```

El primer "select" devuelve el nombre y domicilio de todos los alumnos; el segundo, el nombre y domicilio de todos los profesores.

Los encabezados del resultado de una unión son los que se especifican en el primer "select".

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

create table profesores(
  documento varchar(8) not null,
  nombre varchar (30),
  domicilio varchar(30),
  primary key(documento)
);
create table alumnos(
  documento varchar(8) not null,
  nombre varchar (30),
  domicilio varchar(30),
  primary key(documento)
);

go

insert into alumnos values('30000000','Juan
Perez','Colon 123');
insert into alumnos values('30111111','Marta
Morales','Caseros 222');
insert into alumnos values('30222222','Laura
Torres','San Martin 987');
insert into alumnos values('30333333','Mariano
Juarez','Avellaneda 34');
insert into alumnos values('23333333','Federico
Lopez','Colon 987');
insert into profesores values('22222222','Susana
Molina','Sucre 345');
```

```
insert into profesores values('23333333','Federico
Lopez','Colon 987');

-- Nombre y domicilio de profesores y alumnos
select nombre, domicilio from alumnos
  union
    select nombre, domicilio from profesores;
-- Mostrar las filas duplicadas de ambas tablas
-- (existe un profesor que también está presente en la
tabla "alumnos")
select nombre, domicilio from alumnos
  union all
    select nombre, domicilio from profesores;

-- Ordenamos por domicilio:
select nombre, domicilio from alumnos
  union
    select nombre, domicilio from profesores
  order by domicilio;

-- agregar una columna extra a la consulta con el
encabezado "condicion"
-- en la que aparezca el literal "profesor" o "alumno"
según si la persona
-- es uno u otro:
select nombre, domicilio, 'alumno' as condicion from
alumnos
  union
    select nombre, domicilio,'profesor' from profesores
  order by condicion;
```

83 - Agregar y eliminar campos (alter table - add - drop)

"alter table" permite modificar la estructura de una tabla.
Podemos utilizarla para agregar, modificar y eliminar campos de una tabla.

Para agregar un nuevo campo a una tabla empleamos la siguiente sintaxis básica:

```
 alter table NOMBRETABLA
  add NOMBRENUEVOCAMPO DEFINICION;
```

En el siguiente ejemplo agregamos el campo "cantidad" a la tabla "libros", de tipo tinyint, que acepta valores nulos:

```
alter table libros
 add cantidad tinyint;
```

Puede verificarse la alteración de la estructura de la tabla ejecutando el procedimiento almacenado "sp_columns".

SQL Server no permite agregar campos "not null" a menos que se especifique un valor por defecto:

```
alter table libros
 add autor varchar(20) not null default 'Desconocido';
```

En el ejemplo anterior, se agregó una restricción "default" para el nuevo campo, que puede verificarse ejecutando el procedimiento almacenado "sp_helpconstraint".

Al agregar un campo puede especificarse que sea "identity" (siempre que no exista otro campo identity).

Para eliminar campos de una tabla la sintaxis básica es la siguiente:

```
alter table NOMBRETABLA
 drop column NOMBRECAMPO;
```

En el siguiente ejemplo eliminamos el campo "precio" de la tabla "libros":

```
alter table libros
 drop column precio;
```

No pueden eliminarse los campos que son usados por un índice o tengan restricciones. No puede eliminarse un campo si es el único en la tabla.

Podemos eliminar varios campos en una sola sentencia:

```
alter table libros
 drop column editorial,edicion;
```

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),
  editorial varchar(15),
  edicion datetime,
  precio decimal(6,2)
);

go

insert into libros (titulo,editorial,precio)
  values ('El aleph','Emece',25.50);

-- Agregamos el campo "cantidad" a la tabla "libros", de
tipo tinyint,
-- que acepta valores nulos:
alter table libros
  add cantidad tinyint;

exec sp_columns libros;

-- Agregamos un campo "codigo" a la tabla "libros", de
tipo int con el
-- atributo "identity":
alter table libros
  add codigo int identity;

-- Intentamos agregar un campo llamado "autor" de tipo
varchar(30)
-- que NO acepte valores nulos (genera error):
alter table libros
  add autor varchar(30) not null;

-- Agregar un campo llamado "autor" de tipo varchar(20)
pero con
-- un valor por defecto:
alter table libros
  add autor varchar(20) not null default 'Desconocido';

-- Eliminamos el campo "precio" de la tabla "libros":
alter table libros
  drop column precio;
```

```
exec sp_columns libros;

-- Intentamos eliminar un campo con restricciones
(genera error) :
alter table libros
  drop column autor;

-- Eliminamos varios campos en una sola sentencia:
alter table libros
  drop column editorial,edicion;
```

84 - Alterar campos (alter table - alter)

Hemos visto que "alter table" permite modificar la estructura de una tabla. También podemos utilizarla para modificar campos de una tabla.

La sintaxis básica para modificar un campo existente es la siguiente:

```
 alter table NOMBRETABLA
  alter column CAMPO NUEVADEFINICION;
```

Modificamos el campo "titulo" extendiendo su longitud y para que NO admita valores nulos:

```
 alter table libros
  alter column titulo varchar(40) not null;
```

En el siguiente ejemplo alteramos el campo "precio" de la tabla "libros" que fue definido "decimal(6,2) not null" para que acepte valores nulos:

```
 alter table libros
  alter column precio decimal(6,2) null;
```

SQL Server tiene algunas excepciones al momento de modificar los campos. No permite modificar:

- campos de tipo text, image, ntext y timestamp.

- un campo que es usado en un campo calculado.

- campos que son parte de índices o tienen restricciones, a menos que el cambio no afecte al índice o a la restricción, por ejemplo, se puede ampliar la longitud de un campo de tipo caracter.

- agregando o quitando el atributo "identity".

- campos que afecten a los datos existentes cuando una tabla contiene registros (ejemplo: un campo contiene valores nulos y se pretende redefinirlo como "not null"; un campo int guarda un valor 300 y se pretende modificarlo a tinyint, etc.).

Servidor de SQL Server instalado en forma local.

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

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

create table libros(
  codigo int identity,
  titulo varchar(30),
  autor varchar(30),
  editorial varchar(15),
  precio decimal(6,2) not null default 0
);

go

insert into libros
  values('El aleph','Borges','Planeta',20);
insert into libros
  values('Java en 10 minutos',null,'Siglo XXI',30);
insert into libros
  values('Uno','Richard Bach','Planeta',15);
insert into libros
  values('Martin Fierro','Jose Hernandez',null,30);
insert into libros
  values('Aprenda PHP','Mario Molina','Emece',25);

-- Modificamos el campo "titulo" para que acepte una
cadena más larga y
-- no admita valores nulos:
alter table libros
```

```
  alter column titulo varchar(40) not null;

exec sp_columns libros;

-- Eliminamos registro que tienen en el campo autor el
valor null
-- y realizamos la modificación del campo:
delete from libros where autor is null;
alter table libros
  alter column autor varchar(30) not null;

exec sp_columns libros;

-- Intentamos quitar el atributo "identity" del campo
"codigo" y
-- lo redefinimos como "smallint" (no se produce el
cambio):
alter table libros
  alter column codigo smallint;

exec sp_columns libros;

alter table libros
  alter column precio decimal(6,2) null;

exec sp_columns libros;
```

85 - Agregar campos y restricciones (alter table)

Podemos agregar un campo a una tabla y en el mismo momento aplicarle una restricción.
Para agregar un campo y establecer una restricción, la sintaxis básica es la siguiente:

```
 alter table TABLA
  add CAMPO DEFINICION
  constraint NOMBRERESTRICCION TIPO;
```

Agregamos a la tabla "libros", el campo "titulo" de tipo varchar(30) y una restricción "unique" con índice agrupado:

```
 alter table libros
  add titulo varchar(30)
```

```
  constraint UQ_libros_autor unique clustered;
```

Agregamos a la tabla "libros", el campo "codigo" de tipo int identity not null y una restricción "primary key" con índice no agrupado:

```
 alter table libros
  add codigo int identity not null
  constraint PK_libros_codigo primary key nonclustered;
```

Agregamos a la tabla "libros", el campo "precio" de tipo decimal(6,2) y una restricción "check":

```
 alter table libros
  add precio decimal(6,2)
  constraint CK_libros_precio check (precio>=0);
```

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(
  autor varchar(30),
  editorial varchar(15)
);

go

-- Agregamos el campo "titulo" de tipo varchar(30) y una
-- restricción "unique" con índice agrupado:
alter table libros
  add titulo varchar(30)
  constraint UQ_libros_autor unique clustered;

exec sp_columns libros;

-- Agregamos el campo "codigo" de tipo int identity not
null
-- y en la misma sentencia una restricción "primary key"
con índice no agrupado:
alter table libros
  add codigo int identity not null
```

```
  constraint PK_libros_codigo primary key nonclustered;

-- Agregamos el campo "precio" de tipo decimal(6,2) y
una restricción "check"
-- que no permita valores negativos para dicho campo:
alter table libros
  add precio decimal(6,2)
  constraint CK_libros_precio check (precio>=0);

exec sp_helpconstraint libros;
```

86 - Campos calculados

Un campo calculado es un campo que no se almacena físicamente en la tabla. SQL Server emplea una fórmula que detalla el usuario al definir dicho campo para calcular el valor según otros campos de la misma tabla.

Un campo calculado no puede:

- definirse como "not null".

- ser una subconsulta.

- tener restricción "default" o "foreign key".

- insertarse ni actualizarse.

Puede ser empleado como llave de un índice o parte de restricciones "primary key" o "unique" si la expresión que la define no cambia en cada consulta.

Creamos un campo calculado denominado "sueldototal" que suma al sueldo básico de cada empleado la cantidad abonada por los hijos (100 por cada hijo):

```
 create table empleados(
  documento char(8),
  nombre varchar(10),
  domicilio varchar(30),
  sueldobasico decimal(6,2),
  cantidadhijos tinyint default 0,
  sueldototal as sueldobasico + (cantidadhijos*100)
 );
```

También se puede agregar un campo calculado a una tabla existente:

```
 alter table NOMBRETABLA
  add NOMBRECAMPOCALCULADO as EXPRESION;

 alter table empleados
  add sueldototal as sueldo+(cantidadhijos*100);
```

Los campos de los cuales depende el campo calculado no pueden eliminarse, se debe eliminar primero el campo calculado.

Servidor de SQL Server instalado en forma local.

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

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

create table empleados(
  documento char(8),
  nombre varchar(10),
  domicilio varchar(30),
  sueldobasico decimal(6,2),
  hijos tinyint not null default 0,
  sueldototal as sueldobasico + (hijos*100)
);

go

-- No puede ingresarse valor para el campo sueldototal:
insert into empleados values('22222222','Juan
Perez','Colon 123',300,2);
insert into empleados values('23333333','Ana
Lopez','Sucre 234',500,0);

select * from empleados;

-- Actualizamos un registro:
update empleados set hijos=1 where documento='23333333';

select * from empleados;

-- Agregamos un campo calculado:
```

```
alter table empleados
  add salariofamiliar as hijos*100;

exec sp_columns empleados;

select * from empleados;
```

87 - Tipo de dato definido por el usuario (crear - informacion)

Cuando definimos un campo de una tabla debemos especificar el tipo de datos, sabemos que los tipos de datos especifican el tipo de información (caracteres, números, fechas) que pueden almacenarse en un campo. SQL Server proporciona distintos tipos de datos del sistema (char, varchar, int, decimal, datetime, etc.) y permite tipos de datos definidos por el usuario siempre que se basen en los tipos de datos existentes.

Se pueden crear y eliminar tipos de datos definidos por el usuario.
Se emplean cuando varias tablas deben almacenar el mismo tipo de datos en un campo y se quiere garantizar que todas tengan el mismo tipo y longitud.

Para darle un nombre a un tipo de dato definido por el usuario debe considerar las mismas reglas que para cualquier identificador. No puede haber dos objetos con igual nombre en la misma base de datos.

Para crear un tipo de datos definido por el usuario se emplea el procedimiento almacenado del sistema "sp_addtype". Sintaxis básica:

```
 exec sp_addtype NOMBRENUEVOTIPO,
'TIPODEDATODELSISTEMA', 'OPCIONNULL';
```

Creamos un tipo de datos definido por el usuario llamado "tipo_documento" que admite valores nulos:

```
 exec sp_addtype tipo_documento, 'char(8)', 'null';
```

Ejecutando el procedimiento almacenado "sp_help" junto al nombre del tipo de dato definido por el usuario se obtiene información del mismo (nombre, el tipo de dato en que se basa, la longitud, si acepta valores nulos, si tiene valor por defecto y reglas asociadas).

También podemos consultar la tabla "systypes" en la cual se almacena información de todos los tipos de datos:

```
 select name from systypes;
```

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;

if exists (select *from systypes
  where name = 'tipo_documento')
  exec sp_droptype tipo_documento;

exec sp_addtype tipo_documento, 'char(8)', 'null';

exec sp_help tipo_documento;

create table alumnos(
  documento tipo_documento,
  nombre varchar(30)
);

go

insert into alumnos values('12345678','Ana Acosta');

select * from alumnos;
```

88 - Tipo de dato definido por el usuario (asociación de reglas)

Se puede asociar una regla a un tipo de datos definido por el usuario. Luego de crear la regla se establece la asociación; la sintaxis es la siguiente:

```
 exec sp_bindrule NOMBREREGLA,
'TIPODEDATODEFINIDOPORELUSUARIO', 'futureonly';
```

El parámetro "futureonly" es opcional, especifica que si existen campos (de cualquier tabla) con este tipo de dato, no se asocien a la regla; si creamos una nueva tabla con este tipo de dato, si deberán cumplir la regla. Si no se especifica este parámetro, todos los campos de este tipo de dato, existentes o que se creen posteriormente (de cualquier tabla), quedan asociados a la regla.

Recuerde que SQL Server NO controla los datos existentes para confirmar que cumplen con la regla, si no los cumple, la regla se asocia igualmente; pero al ejecutar una instrucción "insert" o "update" muestra un mensaje de error.

Si asocia una regla a un tipo de dato definido por el usuario que tiene otra regla asociada, esta última la reemplaza.

Para quitar la asociación, empleamos el mismo procedimiento almacenado que aprendimos cuando quitamos asociaciones a campos, ejecutamos el procedimiento almacenado "sp_unbindrule" seguido del nombre del tipo de dato al que está asociada la regla:

```
 exec sp_unbindrule 'TIPODEDATODEFINIDOPORELUSUARIO';
```

Si asocia una regla a un campo cuyo tipo de dato definido por el usuario ya tiene una regla asociada, la nueva regla se aplica al campo, pero el tipo de dato continúa asociado a la regla. La regla asociada al campo prevalece sobre la asociada al tipo de dato. Por ejemplo, tenemos un campo "precio" de un tipo de dato definido por el usuario "tipo_precio", este tipo de dato tiene asociada una regla "RG_precio0a99" (precio entre 0 y 99), luego asociamos al campo "precio" la regla "RG_precio100a500" (precio entre 100 y 500); al ejecutar una instrucción "insert" admitirá valores entre 100 y 500, es decir, tendrá en cuenta la regla asociada al campo, aunque vaya contra la regla asociada al tipo de dato.

Un tipo de dato definido por el usuario puede tener una sola regla asociada.

Cuando obtenemos información del tipo da dato definido por el usuario ejecutando "sp_help", en la columna "rule_name" se muestra el nombre de la regla asociada a dicho tipo de dato; muestran "none" cuando no tiene regla asociada.

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

if exists (select *from systypes
```

```
  where name = 'tipo_documento')
  exec sp_droptype tipo_documento;

-- Creamos un tipo de dato definido por el usuario
llamado
-- "tipo_documento" basado en el tipo "char" que permita
8 caracteres
-- y valores nulos:
exec sp_addtype tipo_documento, 'char(8)', 'null';

exec sp_help tipo_documento;

create table alumnos(
  documento tipo_documento,
  nombre varchar(30)
);

go

if object_id ('RG_documento') is not null
   drop rule RG_documento;

go

-- Creamos la regla que permita 8 caracteres que
solamente pueden ser
-- dígitos del 0 al 5 para el primer dígito y de 0 al 9
para los siguientes:
create rule RG_documento
  as @documento like '[0-5][0-9][0-9][0-9][0-9][0-9][0-
9][0-9]';

go

-- Asociamos la regla al tipo de datos "tipo_documento"
especificando que
-- solamente se aplique a los futuros campos de este
tipo:
exec sp_bindrule RG_documento, 'tipo_documento',
'futureonly';

exec sp_helpconstraint alumnos;

create table docentes(
```

```
  documento tipo_documento,
  nombre varchar(30)
);

-- Verificamos que se aplicó la regla en la nueva tabla:
exec sp_helpconstraint docentes;

-- Ingresamos un registro en "alumnos" con valores para
documento que infrinjan la regla,
-- Lo acepta porque en esta tabla no se aplica la regla.
-- Pero no podríamos ingresar un valor como el anterior
en la tabla "docentes"
-- la cual si tiene asociada la regla.
insert into alumnos values('a111111','Ana Acosta');

-- Quitamos la asociación:
exec sp_unbindrule 'tipo_documento';

-- Volvemos a asociar la regla, ahora sin el parámetro
"futureonly":
exec sp_bindrule RG_documento, 'tipo_documento';

-- Verificamos que se aplicó la regla en ambas tablas:
exec sp_helpconstraint docentes;
exec sp_helpconstraint alumnos;

-- Eliminamos si existe, la regla "RG_documento2":
if object_id ('RG_documento2') is not null
   drop rule RG_documento2;

go

-- Creamos la regla llamada "RG_documento2" que permita
8 caracteres
-- que solamente pueden ser dígitos del 0 al 9 para
todas las posiciones:
create rule RG_documento2
  as @documento like '[0-9][0-9][0-9][0-9][0-9][0-9][0-
9][0-9]';

go

-- Asociamos la regla al tipo de datos "tipo_documento"
(ya tiene una regla asociada):
```

```
exec sp_bindrule RG_documento2, 'tipo_documento';

--Veamos si la asociación fue reemplazada en el tipo de
datos:
exec sp_help tipo_documento;

-- Veamos si la asociación fue reemplazada en las
tablas:
exec sp_helpconstraint alumnos;
exec sp_helpconstraint docentes;

-- Asociamos la regla "RG_documento" al campo
"documento" de "alumnos":
exec sp_bindrule RG_documento, 'alumnos.documento';

-- Verificamos que "documento" de "alumnos" tiene
asociada la regla "RG_documento":
exec sp_helpconstraint alumnos;

-- Verificamos que el tipo de dato "tipo_documento"
tiene asociada la regla "RG_documento2":
exec sp_help tipo_documento;

-- Intentamos ingresar un valor para "documento"
aceptado por la regla asociada al
-- tipo de dato pero no por la regla asociada al campo
(no lo permite):
insert into alumnos values ('77777777','Juan Lopez');

-- Ingrese un valor para "documento" aceptado por la
regla asociada al campo (si lo permite):
insert into alumnos values ('55555555','Juan Lopez');
```

89 - Tipo de dato definido por el usuario (valores predeterminados)

Se puede asociar un valor predeterminado a un tipo de datos definido por el usuario. Luego de crear un valor predeterminado, se puede asociar a un tipo de dato definido por el usuario con la siguiente sintaxis:

```
 exec sp_bindefault NOMBREVALORPREDETERMINADO,
'TIPODEDATODEFINIDOPORELUSUARIO','futureonly';
```

El parámetro "futureonly" es opcional, especifica que si existen campos (de cualquier tabla) con este tipo de dato, no se asocien al valor predeterminado; si creamos una nueva tabla con este tipo de dato, si estará asociado al valor predeterminado. Si no se especifica este parámetro, todos los campos de este tipo de dato, existentes o que se creen posteriormente (de cualquier tabla), quedan asociados al valor predeterminado.

Si asocia un valor predeterminado a un tipo de dato definido por el usuario que tiene otro valor predeterminado asociado, el último lo reemplaza.

Para quitar la asociación, empleamos el mismo procedimiento almacenado que aprendimos cuando quitamos asociaciones a campos:

```
 sp_unbindefault 'TIPODEDATODEFINIDOPORELUSUARIO';
```

Debe tener en cuenta que NO se puede aplicar una restricción "default" en un campo con un tipo de datos definido por el usuario si dicho campo o tipo de dato tienen asociado un valor predeterminado.

Si un campo de un tipo de dato definido por el usuario tiene una restricción "default" y luego se asocia un valor predeterminado al tipo de dato, el valor predeterminado no queda asociado en el campo que tiene la restricción "default".

Un tipo de dato definido por el usuario puede tener un solo valor predeterminado asociado.

Cuando obtenemos información del tipo da dato definido por el usuario ejecutando "sp_help", en la columna "default_name" se muestra el nombre del valor predeterminado asociado a dicho tipo de dato; muestra "none" cuando no tiene ningún valor predeterminado asociado.

Servidor de SQL Server instalado en forma local.

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

```
-- Eliminamos ambas tablas, si existen:
if object_id('alumnos') is not null
  drop table alumnos;
if object_id('docentes') is not null
  drop table docentes;
-- Queremos definir un nuevo tipo de dato llamado
"tipo_documento". Primero debemos eliminarlo,
```

```
-- si existe para volver a crearlo. Para ello empleamos
esta sentencia que explicaremos próximamente:
if exists (select *from systypes
  where name = 'tipo_documento')
  exec sp_droptype tipo_documento;

-- Creamos un tipo de dato definido por el usuario
llamado "tipo_documento"
-- basado en el tipo "char" que permita 8 caracteres y
valores nulos:
exec sp_addtype tipo_documento, 'char(8)', 'null';

-- Ejecutamos el procedimiento almacenado "sp_help"
junto al nombre del tipo de dato
-- definido anteriormente para obtener información del
mismo:
exec sp_help tipo_documento;

create table alumnos(
  documento tipo_documento,
  nombre varchar(30)
);

-- Eliminamos si existe, el valor predeterminado
"VP_documento0":
 if object_id ('VP_documento0') is not null
   drop default VP_documento0;

go

-- Creamos el valor predeterminado "VP_documento0" que
almacene el valor '00000000':
create default VP_documento0
  as '00000000';

go

-- Asociamos el valor predeterminado al tipo de datos
"tipo_documento"
-- especificando que solamente se aplique a los futuros
campos de este tipo:
exec sp_bindefault VP_documento0, 'tipo_documento',
'futureonly';
```

```
-- Ejecutamos el procedimiento almacenado
"sp_helpconstraint" para verificar que
-- no se aplicó a la tabla "alumnos" porque
especificamos la opción "futureonly":
exec sp_helpconstraint alumnos;

create table docentes(
  documento tipo_documento,
  nombre varchar(30)
);

-- Verificamos que se aplicó el valor predeterminado
creado anteriormente al campo
-- "documento" de la nueva tabla:
exec sp_helpconstraint docentes;

-- Ingresamos un registro en "alumnos" sin valor para
documento y vemos qué se almacenó.
-- En esta tabla no se aplica el valor predeterminado
por ello almacena "null",
-- que es el valor por defecto.
insert into alumnos default values;
select * from alumnos;

-- Si ingresamos en la tabla "docentes" un registro con
valores por defecto,
--  se almacena el valor predeterminado porque está
asociado:
insert into docentes default values;
select * from docentes;

-- Quitamos la asociación:
exec sp_unbindefault 'tipo_documento';

-- Volvemos a asociar el valor predeterminado, ahora sin
el parámetro "futureonly":
exec sp_bindefault VP_documento0, 'tipo_documento';

-- Ingresamos un registro en "alumnos" y en "docentes"
sin valor para documento
-- y vemos qué se almacenó (en ambas se almacenó
'00000000'):
insert into alumnos default values;
select * from alumnos;
```

```
insert into docentes default values;
select * from docentes;

-- Eliminamos si existe, el valor predeterminado
"VP_documentoDesconocido":
if object_id ('VP_documentoDesconocido') is not null
   drop default VP_documentoDesconocido;

go

-- Creamos el valor predeterminado llamado
"VP_documentoDesconocido"
-- que almacene el valor 'SinDatos':
create default VP_documentoDesconocido
  as 'SinDatos';

go

-- Asociamos el valor predeterminado al tipo de datos
"tipo_documento"
-- (ya tiene otro valor predeterminado asociado):
exec sp_bindefault VP_DocumentoDesconocido,
'tipo_documento';

-- Veamos si la asociación fue reemplazada en el tipo de
datos:
exec sp_help tipo_documento;

-- Veamos si la asociación fue reemplazada en la tabla
"alumnos":
exec sp_helpconstraint alumnos;

-- Quitamos la asociación del valor predeterminado:
exec sp_unbindefault 'tipo_documento';

-- Veamos si se quitó de ambas tablas:
exec sp_helpconstraint alumnos;
exec sp_helpconstraint docentes;

-- Ingresamos un registro en "alumnos" y vemos qué se
almacenó en el campo "documento":
insert into alumnos default values;
select * from alumnos;
```

```
-- Agregue a la tabla "docentes" una restricción
"default" para el campo "documento":
alter table docentes
 add constraint DF_docentes_documento
 default '--------'
 for documento;

-- Ingrese un registro en "docentes" con valores por
defecto y vea qué se almacenó en
-- "documento" recuperando los registros:
insert into docentes default values;
select * from docentes;

-- Asocie el valor predeterminado "VP_documento0" al
tipo de datos "tipo_documento":
exec sp_bindefault VP_documento0, 'tipo_documento';

-- Vea qué informa "sp_helpconstraint" acerca de la
tabla "docentes" (Tiene asociado
-- el valor por defecto establecido con la restricción
"default"):
exec sp_helpconstraint docentes;

-- Ingrese un registro en "docentes" con valores por
defecto y vea qué se almacenó
-- en "documento" (note que guarda el valor por defecto
establecido con la restricción):
insert into docentes default values;
select * from docentes;

-- Eliminamos la restricción:
alter table docentes
  drop DF_docentes_documento;

-- Vea qué informa "sp_helpconstraint" acerca de la
tabla "docentes" (no tiene valor
-- por defecto):
exec sp_helpconstraint docentes;

-- Asociamos el valor predeterminado "VP_documento0" al
tipo de datos "tipo_documento":
exec sp_bindefault VP_documento0, 'tipo_documento';
```

```
-- Intente agregar una restricción "default" al campo
"documento" de "docentes"
--(SQL Server no lo permite porque el tipo de dato de
ese campo ya tiene
-- un valor predeterminado asociado):
alter table docentes
 add constraint DF_docentes_documento
 default '--------'
 for documento;
```

90 - Tipo de dato definido por el usuario (eliminar)

Podemos eliminar un tipo de dato definido por el usuario con el procedimiento almacenado "sp_droptype":

```
 exec sp_droptype TIPODEDATODEFINIDOPORELUSUARIO;
```

Eliminamos el tipo de datos definido por el usuario llamado "tipo_documento":

```
 exec sp_droptype tipo_documento;
```

Si intentamos eliminar un tipo de dato inexistente, aparece un mensaje indicando que no existe.

Los tipos de datos definidos por el usuario se almacenan en la tabla del sistema "systypes".
Podemos averiguar si un tipo de dato definido por el usuario existe para luego eliminarlo:

```
 if exists (select *from systypes
  where name = 'NOMBRETIPODEDATODEFINIDOPORELUSUARIO')
  exec sp_droptype TIPODEDATODEFINIDOPORELUSUARIO;
```

Consultamos la tabla "systypes" para ver si existe el tipo de dato "tipo_documento", si es así, lo eliminamos:

```
 if exists (select *from systypes
  where name = 'tipo_documento')
  exec sp_droptype tipo_documento;
```

No se puede eliminar un tipo de datos definido por el usuario si alguna tabla (u otro objeto) hace uso de él; por ejemplo, si una tabla tiene un campo definido con tal tipo de dato.

Si eliminamos un tipo de datos definido por el usuario, desaparecen las asociaciones de las reglas y valores predeterminados, pero tales reglas y valores predeterminados, no se eliminan, siguen existiendo en la base de datos.

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;

-- Definimos un nuevo tipo de dato llamado
"tipo_documento".
-- Primero debemos eliminarlo, si existe, para volver a
crearlo:
if exists (select *from systypes
  where name = 'tipo_documento')
  exec sp_droptype tipo_documento;

--Creamos un tipo de dato definido por el usuario
llamado "tipo_documento"
-- basado en el tipo "char" que permita 8 caracteres y
valores nulos:
exec sp_addtype tipo_documento, 'char(8)', 'null';

-- Eliminamos la regla "RG_documento" si existe:
if object_id ('RG_documento') is not null
   drop rule RG_documento;

go

-- Creamos la regla que permita 8 caracteres que
solamente serán dígitos:
create rule RG_documento
  as @documento like '[0-9][0-9][0-9][0-9][0-9][0-9][0-
9][0-9]';

go
```

```
-- Asociamos la regla al tipo de datos "tipo_documento":
exec sp_bindrule RG_documento, 'tipo_documento';

-- Creamos la tabla "alumnos":
create table alumnos(
  nombre varchar(30),
  documento tipo_documento
);

-- No podemos eliminar el tipo de dato "tipo_documento"
porque hay una
-- tabla "alumnos" que lo utiliza. Entonces eliminamos
la tabla:
drop table alumnos;

-- Ahora podemos eliminar el tipo de datos:
exec sp_droptype tipo_documento;

-- Volvemos a crear el tipo de dato:
exec sp_addtype tipo_documento, 'char(8)', 'null';

-- Note que no tiene reglas asociadas:
exec sp_help tipo_documento;

-- Asociamos la regla nuevamente:
exec sp_bindrule RG_documento, 'tipo_documento';
```

91 - Subconsultas

Una subconsulta (subquery) es una sentencia "select" anidada en otra sentencia "select", "insert", "update" o "delete" (o en otra subconsulta).

Las subconsultas se emplean cuando una consulta es muy compleja, entonces se la divide en varios pasos lógicos y se obtiene el resultado con una única instrucción y cuando la consulta depende de los resultados de otra consulta.

Generalmente, una subconsulta se puede reemplazar por combinaciones y estas últimas son más eficientes.

Las subconsultas se DEBEN incluir entre paréntesis.

Puede haber subconsultas dentro de subconsultas, se admiten hasta 32 niveles de anidación.

Se pueden emplear subconsultas:

- en lugar de una expresión, siempre que devuelvan un solo valor o una lista de valores.

- que retornen un conjunto de registros de varios campos en lugar de una tabla o para obtener el mismo resultado que una combinación (join).

Hay tres tipos básicos de subconsultas:

1. las que retornan un solo valor escalar que se utiliza con un operador de comparación o en lugar de una expresión.
2. las que retornan una lista de valores, se combinan con "in", o los operadores "any", "some" y "all".
3. los que testean la existencia con "exists".

Reglas a tener en cuenta al emplear subconsultas:

- la lista de selección de una subconsulta que va luego de un operador de comparación puede incluir sólo una expresión o campo (excepto si se emplea "exists" y "in").

- si el "where" de la consulta exterior incluye un campo, este debe ser compatible con el campo en la lista de selección de la subconsulta.

- no se pueden emplear subconsultas que recuperen campos de tipos text o image.

- las subconsultas luego de un operador de comparación (que no es seguido por "any" o "all") no pueden incluir cláusulas "group by" ni "having".

- "distinct" no puede usarse con subconsultas que incluyan "group by".

- no pueden emplearse las cláusulas "compute" y "compute by".

- "order by" puede emplearse solamente si se especifica "top" también.

- una vista creada con una subconsulta no puede actualizarse.

- una subconsulta puede estar anidada dentro del "where" o "having" de una consulta externa o dentro de otra subconsulta.

- si una tabla se nombra solamente en un subconsulta y no en la consulta externa, los campos no serán incluidos en la salida (en la lista de selección de la consulta externa).

92 - Subconsultas como expresión

Una subconsulta puede reemplazar una expresión. Dicha subconsulta debe devolver un valor escalar (o una lista de valores de un campo).

Las subconsultas que retornan un solo valor escalar se utiliza con un operador de comparación o en lugar de una expresión:

```
select CAMPOS
 from TABLA
 where CAMPO OPERADOR (SUBCONSULTA);

select CAMPO OPERADOR (SUBCONSULTA)
 from TABLA;
```

Si queremos saber el precio de un determinado libro y la diferencia con el precio del libro más costoso, anteriormente debíamos averiguar en una consulta el precio del libro más costoso y luego, en otra consulta, calcular la diferencia con el valor del libro que solicitamos. Podemos conseguirlo en una sola sentencia combinando dos consultas:

```
select titulo,precio,
 precio-(select max(precio) from libros) as diferencia
 from libros
 where titulo='Uno';
```

En el ejemplo anterior se muestra el título, el precio de un libro y la diferencia entre el precio del libro y el máximo valor de precio.

Queremos saber el título, autor y precio del libro más costoso:

```
select titulo,autor, precio
 from libros
 where precio=
  (select max(precio) from libros);
```

Note que el campo del "where" de la consulta exterior es compatible con el valor retornado por la expresión de la subconsulta.

Se pueden emplear en "select", "insert", "update" y "delete".

Para actualizar un registro empleando subconsulta la sintaxis básica es la siguiente:

```
 update TABLA set CAMPO=NUEVOVALOR
  where CAMPO= (SUBCONSULTA);
```

Para eliminar registros empleando subconsulta empleamos la siguiente sintaxis básica:

```
 delete from TABLA
  where CAMPO=(SUBCONSULTA);
```

Recuerde que la lista de selección de una subconsulta que va luego de un operador de comparación puede incluir sólo una expresión o campo (excepto si se emplea "exists" o "in").

No olvide que las subconsultas luego de un operador de comparación (que no es seguido por "any" o "all") no pueden incluir cláusulas "group by".

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)
);

go

insert into libros values('Alicia en el pais de las
maravillas','Lewis Carroll','Emece',20.00);
insert into libros values('Alicia en el pais de las
maravillas','Lewis Carroll','Plaza',35.00);
```

```
insert into libros values('Aprenda PHP','Mario
Molina','Siglo XXI',40.00);
insert into libros values('El
aleph','Borges','Emece',10.00);
insert into libros values('Ilusiones','Richard
Bach','Planeta',15.00);
insert into libros values('Java en 10 minutos','Mario
Molina','Siglo XXI',50.00);
insert into libros values('Martin Fierro','Jose
Hernandez','Planeta',20.00);
insert into libros values('Martin Fierro','Jose
Hernandez','Emece',30.00);
insert into libros values('Uno','Richard
Bach','Planeta',10.00);

-- Obtenemos el título, precio de un libro específico y
la diferencia entre
-- su precio y el máximo valor:
select titulo,precio,
  precio-(select max(precio) from libros) as diferencia
  from libros
  where titulo='Uno';

-- Mostramos el título y precio del libro más costoso:
select titulo,autor, precio
  from libros
  where precio=
   (select max(precio) from libros);

-- Actualizamos el precio del libro con máximo valor:
update libros set precio=45
  where precio=
   (select max(precio) from libros);

-- Eliminamos los libros con precio menor:
delete from libros
  where precio=
   (select min(precio) from libros);
```

93 - Subconsultas con in

Vimos que una subconsulta puede reemplazar una expresión. Dicha subconsulta debe devolver un valor escalar o una lista de valores de un

campo; las subconsultas que retornan una lista de valores reemplazan a una expresión en una cláusula "where" que contiene la palabra clave "in".

El resultado de una subconsulta con "in" (o "not in") es una lista. Luego que la subconsulta retorna resultados, la consulta exterior los usa.

La sintaxis básica es la siguiente:

```
...where EXPRESION in (SUBCONSULTA);
```

Este ejemplo muestra los nombres de las editoriales que ha publicado libros de un determinado autor:

```
select nombre
 from editoriales
 where codigo in
  (select codigoeditorial
    from libros
    where autor='Richard Bach');
```

La subconsulta (consulta interna) retorna una lista de valores de un solo campo (codigo) que la consulta exterior luego emplea al recuperar los datos.

Podemos reemplazar por un "join" la consulta anterior:

```
select distinct nombre
 from editoriales as e
 join libros
 on codigoeditorial=e.codigo
 where autor='Richard Bach';
```

Una combinación (join) siempre puede ser expresada como una subconsulta; pero una subconsulta no siempre puede reemplazarse por una combinación que retorne el mismo resultado. Si es posible, es aconsejable emplear combinaciones en lugar de subconsultas, son más eficientes.

Se recomienda probar las subconsultas antes de incluirlas en una consulta exterior, así puede verificar que retorna lo necesario, porque a veces resulta difícil verlo en consultas anidadas.

También podemos buscar valores No coincidentes con una lista de valores que retorna una subconsulta; por ejemplo, las editoriales que no han publicado libros de un autor específico:

```
 select nombre
  from editoriales
  where codigo not in
   (select codigoeditorial
     from libros
     where autor='Richard Bach');
```

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 editoriales(
  codigo tinyint identity,
  nombre varchar(30),
  primary key (codigo)
);

create table libros (
  codigo int identity,
  titulo varchar(40),
  autor varchar(30),
  codigoeditorial tinyint,
  primary key(codigo),
 constraint FK_libros_editorial
   foreign key (codigoeditorial)
   references editoriales(codigo)
   on update cascade,
);

go

insert into editoriales values('Planeta');
insert into editoriales values('Emece');
insert into editoriales values('Paidos');
insert into editoriales values('Siglo XXI');

insert into libros values('Uno','Richard Bach',1);
insert into libros values('Ilusiones','Richard Bach',1);
```

```
insert into libros values('Aprenda PHP','Mario
Molina',4);
insert into libros values('El aleph','Borges',2);
insert into libros values('Puente al infinito','Richard
Bach',2);

-- Queremos conocer el nombre de las editoriales que han
publicado
-- libros del autor "Richard Bach":
select nombre
  from editoriales
  where codigo in
   (select codigoeditorial
     from libros
     where autor='Richard Bach');

-- Probamos la subconsulta separada de la consulta
exterior
-- para verificar que retorna una lista de valores de un
solo campo:
select codigoeditorial
  from libros
  where autor='Richard Bach';

-- Podemos reemplazar por un "join" la primera consulta:
select distinct nombre
  from editoriales as e
  join libros
  on codigoeditorial=e.codigo
  where autor='Richard Bach';

-- También podemos buscar las editoriales que no han
publicado
-- libros de "Richard Bach":
select nombre
  from editoriales
  where codigo not in
   (select codigoeditorial
     from libros
     where autor='Richard Bach');
```

94 - Subconsultas any - some - all

"any" y "some" son sinónimos. Chequean si alguna fila de la lista resultado de una subconsulta se encuentra el valor especificado en la condición.

El tipo de datos que se comparan deben ser compatibles.

La sintaxis básica es:

```
...VALORESCALAR OPERADORDECOMPARACION
 ANY (SUBCONSULTA);
```

Queremos saber los títulos de los libros de "Borges" que pertenecen a editoriales que han publicado también libros de "Richard Bach", es decir, si los libros de "Borges" coinciden con ALGUNA de las editoriales que publicó libros de "Richard Bach":

```
select titulo
 from libros
 where autor='Borges' and
 codigoeditorial = any
  (select e.codigo
   from editoriales as e
   join libros as l
   on codigoeditorial=e.codigo
   where l.autor='Richard Bach');
```

La consulta interna (subconsulta) retorna una lista de valores de un solo campo (puede ejecutar la subconsulta como una consulta para probarla), luego, la consulta externa compara cada valor de "codigoeditorial" con cada valor de la lista devolviendo los títulos de "Borges" que coinciden.

"all" también compara un valor escalar con una serie de valores. Chequea si TODOS los valores de la lista de la consulta externa se encuentran en la lista de valores devuelta por la consulta interna.
Sintaxis:

```
 VALORESCALAR OPERADORDECOMPARACION all (SUBCONSULTA);
```

Veamos otro ejemplo con un operador de comparación diferente:

Queremos saber si ALGUN precio de los libros de "Borges" es mayor a ALGUN precio de los libros de "Richard Bach":

```
select titulo,precio
 from libros
```

```
  where autor='Borges' and
  precio > any
   (select precio
    from libros
    where autor='Bach');
```

El precio de cada libro de "Borges" es comparado con cada valor de la lista de valores retornada por la subconsulta; si ALGUNO cumple la condición, es decir, es mayor a ALGUN precio de "Richard Bach", se lista.

Veamos la diferencia si empleamos "all" en lugar de "any":

```
 select titulo,precio
  from libros
  where autor='borges' and
  precio > all
   (select precio
    from libros
    where autor='bach');
```

El precio de cada libro de "Borges" es comparado con cada valor de la lista de valores retornada por la subconsulta; si cumple la condición, es decir, si es mayor a TODOS los precios de "Richard Bach" (o al mayor), se lista.

Emplear "= any" es lo mismo que emplear "in".

Emplear "<> all" es lo mismo que emplear "not in".

Recuerde que solamente las subconsultas luego de un operador de comparación al cual es seguido por "any" o "all") pueden incluir cláusulas "group by".

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 editoriales(
  codigo tinyint identity,
```

```
  nombre varchar(30),
  primary key (codigo)
);

create table libros (
  codigo int identity,
  titulo varchar(40),
  autor varchar(30),
  codigoeditorial tinyint,
  precio decimal(5,2),
  primary key(codigo),
 constraint FK_libros_editorial
   foreign key (codigoeditorial)
   references editoriales(codigo)
   on update cascade,
);

go

insert into editoriales values('Planeta');
insert into editoriales values('Emece');
insert into editoriales values('Paidos');
insert into editoriales values('Siglo XXI');

insert into libros values('Uno','Richard Bach',1,15);
insert into libros values('Ilusiones','Richard
Bach',4,18);
insert into libros values('Puente al infinito','Richard
Bach',2,19);
insert into libros values('Aprenda PHP','Mario
Molina',4,40);
insert into libros values('El aleph','Borges',2,10);
insert into libros values('Antología','Borges',1,20);
insert into libros values('Cervantes y el
quijote','Borges',3,25);

-- Mostramos los títulos de los libros de "Borges" de
editoriales que
-- han publicado también libros de "Richard Bach":
select titulo
  from libros
  where autor like '%Borges%' and
  codigoeditorial = any
   (select e.codigo
```

```
    from editoriales as e
    join libros as l
    on codigoeditorial=e.codigo
    where l.autor like '%Bach%');

-- Mostramos los títulos y precios de los libros
"Borges" cuyo precio
-- supera a ALGUN precio de los libros de "Richard
Bach":
 select titulo,precio
  from libros
  where autor like '%Borges%' and
  precio > any
   (select precio
    from libros
    where autor like '%Bach%');

-- Veamos la diferencia si empleamos "all" en lugar de
"any":
 select titulo,precio
  from libros
  where autor like '%Borges%' and
  precio > all
   (select precio
    from libros
    where autor like '%Bach%');

-- Empleamos la misma subconsulta para eliminación:
 delete from libros
  where autor like '%Borges%' and
  precio > all
   (select precio
    from libros
    where autor like '%Bach%');
```

95 - Subconsultas correlacionadas

Un almacén almacena la información de sus ventas en una tabla llamada "facturas" en la cual guarda el número de factura, la fecha y el nombre del cliente y una tabla denominada "detalles" en la cual se almacenan los distintos items correspondientes a cada factura: el nombre del artículo, el precio (unitario) y la cantidad.

Se necesita una lista de todas las facturas que incluya el número, la fecha, el cliente, la cantidad de artículos comprados y el total:

```
select f.*,
  (select count(d.numeroitem)
    from Detalles as d
    where f.numero=d.numerofactura) as cantidad,
  (select sum(d.preciounitario*cantidad)
    from Detalles as d
    where f.numero=d.numerofactura) as total
from facturas as f;
```

El segundo "select" retorna una lista de valores de una sola columna con la cantidad de items por factura (el número de factura lo toma del "select" exterior); el tercer "select" retorna una lista de valores de una sola columna con el total por factura (el número de factura lo toma del "select" exterior); el primer "select" (externo) devuelve todos los datos de cada factura.

A este tipo de subconsulta se la denomina consulta correlacionada. La consulta interna se evalúa tantas veces como registros tiene la consulta externa, se realiza la subconsulta para cada registro de la consulta externa. El campo de la tabla dentro de la subconsulta (f.numero) se compara con el campo de la tabla externa.

En este caso, específicamente, la consulta externa pasa un valor de "numero" a la consulta interna. La consulta interna toma ese valor y determina si existe en "detalles", si existe, la consulta interna devuelve la suma. El proceso se repite para el registro de la consulta externa, la consulta externa pasa otro "numero" a la consulta interna y SQL Server repite la evaluación.

Servidor de SQL Server instalado en forma local.

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

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

create table facturas(
  numero int not null,
  fecha datetime,
```

```
  cliente varchar(30),
  primary key(numero)
);

create table detalles(
  numerofactura int not null,
  numeroitem int not null,
  articulo varchar(30),
  precio decimal(5,2),
  cantidad int,
  primary key(numerofactura,numeroitem),
   constraint FK_detalles_numerofactura
   foreign key (numerofactura)
   references facturas(numero)
   on update cascade
   on delete cascade,
);

go

-- Seteamos el formato de la fecha: año, mes y día:
set dateformat ymd;

insert into facturas values(1200,'2018-01-15','Juan
Lopez');
insert into facturas values(1201,'2018-01-15','Luis
Torres');
insert into facturas values(1202,'2018-01-15','Ana
Garcia');
insert into facturas values(1300,'2018-01-20','Juan
Lopez');

insert into detalles values(1200,1,'lapiz',1,100);
insert into detalles values(1200,2,'goma',0.5,150);
insert into detalles values(1201,1,'regla',1.5,80);
insert into detalles values(1201,2,'goma',0.5,200);
insert into detalles values(1201,3,'cuaderno',4,90);
insert into detalles values(1202,1,'lapiz',1,200);
insert into detalles values(1202,2,'escuadra',2,100);
insert into detalles values(1300,1,'lapiz',1,300);

--  Listado de todas las facturas que incluya el número,
la fecha,
```

```
-- el cliente, la cantidad de artículos comprados y el
total:
select f.*,
  (select count(d.numeroitem)
    from detalles as d
    where f.numero=d.numerofactura) as cantidad,
  (select sum(d.precio*cantidad)
    from detalles as d
    where f.numero=d.numerofactura) as total
 from facturas as f;
```

96 - Subconsulta - Existe y no existe

Los operadores "existe" y "no existe" se emplean para determinar si hay o no hay datos en una lista de valores.

Estos operadores pueden emplearse con subconsultas correlacionadas para restringir el resultado de una consulta exterior a los registros que cumplen la subconsulta (consulta interior). Estos operadores retornan "true" (si las subconsultas retornan registros) o "false" (si las subconsultas no retornan registros).

Cuando se coloca en una subconsulta el operador "existe", SQL Server analiza si hay datos que coinciden con la subconsulta, no se devuelve ningún registro, es como una prueba de existencia; SQL Server termina la recuperación de registros cuando por lo menos un registro cumple la condición "where" de la subconsulta.

La sintaxis básica es la siguiente:

```
 ... donde existe (SUBCONSULTA);
```

En este ejemplo se usa una subconsulta correlacionada con un operador "existe" en la cláusula "donde" para devolver una lista de clientes que compraron el artículo "lapiz":

```
 seleccionar cliente, numero
  de facturas como f
  donde existe
   (seleccione * de Detalles como d
     donde f.numero = d.numerofactura
     y d.articulo = 'lapiz');
```

Puede obtener el mismo resultado empleando una combinación.

Podemos buscar los clientes que no han adquirido el artículo "lapiz" empleando "no existe":

```
 seleccionar cliente, numero
  de facturas como f
  donde no existe
   (seleccione * de Detalles como d
     donde f.numero = d.numerofactura
     y d.articulo = 'lapiz');
```

Servidor de SQL Server instalado en forma local.

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

```
si object_id ('detalles') no es nulo
  detalles de la mesa abatible;
si object_id ('facturas') no es nulo
  facturas de mesa abatible;

crear tablas de facturas
  numero int no nulo,
  fecha fecha y hora,
  cliente varchar (30),
  clave primaria (numero)
);

crear detalles de mesa
  numerofactura int no nulo,
  numeroitem int no nulo,
  articulo varchar (30),
  precio decimal (5,2),
  cantidad int,
  clave primaria (numerofactura, numeroitem),
   restricción FK_detalles_numerofactura
   clave externa (numerofactura)
   referencias facturas (numero)
   en cascada de actualización
   en borrar cascada
);

ir
```

```
establecer formato de fecha ymd;

insertar en valores de facturas (1200, '2018-01-15',
'Juan Lopez');
insertar en valores de facturas (1201, '2018-01-15',
'Luis Torres');
insertar en facturas valores (1202, '2018-01-15', 'Ana
García');
insertar en facturas valores (1300, '2018-01-20', 'Juan
Lopez');

insertar en los valores de los detalles (1200,1,
'lapiz', 1,100);
insertar en los valores de los detalles (1200,2, 'goma',
0.5,150);
insertar en los detalles los valores (1201,1, 'regla',
1.5,80);
insertar en los valores de los detalles (1201,2, 'goma',
0.5,200);
insertar en los valores de los detalles (1201,3,
'cuaderno', 4,90);
insertar en los valores de los detalles (1202,1,
'lapiz', 1200);
insertar en los valores de detalles (1202,2, 'escuadra',
2,100);
insertar en los valores de los detalles (1300,1,
'lapiz', 1,300);

- Retornar la lista de clientes que compraron el
artículo "lapiz":
seleccionar cliente, numero
  de facturas como f
  donde existe
   (seleccione * de detalles como d
     donde f.numero = d.numerofactura
     y d.articulo = 'lapiz');
- Buscamos los clientes que NO han comprado el artículo
"lapiz":
 seleccionar cliente, numero
  de facturas como f
  donde no existe
   (seleccione * de detalles como d
     donde f.numero = d.numerofactura
     y d.articulo = 'lapiz');
```

97 - Subconsulta simil autocombinación

Algunas sentencias en las cuales la consulta interna y la externa emplean la misma tabla pueden reemplazarse por una autocombinación.

Por ejemplo, queremos una lista de los libros que han sido publicados por distintas editoriales.

```
select distinct l1.titulo
 from libros as l1
 where l1.titulo in
 (select l2.titulo
   from libros as l2
   where l1.editorial <> l2.editorial);
```

En el ejemplo anterior empleamos una subconsulta correlacionada y las consultas interna y externa emplean la misma tabla. La subconsulta devuelve una lista de valores por ello se emplea "in" y sustituye una expresión en una cláusula "where".

Con el siguiente "join" se obtiene el mismo resultado:

```
select distinct l1.titulo
 from libros as l1
 join libros as l2
 on l1.titulo=l2.titulo and
 l1.autor=l2.autor
 where l1.editorial<>l2.editorial;
```

Otro ejemplo: Buscamos todos los libros que tienen el mismo precio que "El aleph" empleando subconsulta:

```
select titulo
 from libros
 where titulo<>'El aleph' and
 precio =
  (select precio
    from libros
    where titulo='El aleph');
```

La subconsulta retorna un solo valor.

Buscamos los libros cuyo precio supere el precio promedio de los libros por editorial:

```
 select l1.titulo,l1.editorial,l1.precio
  from libros as l1
  where l1.precio >
   (select avg(l2.precio)
   from libros as l2
    where l1.editorial= l2.editorial);
```

Por cada valor de l1, se evalúa la subconsulta, si el precio es mayor que el promedio.

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)
);

go

insert into libros values('Alicia en el pais de las
maravillas','Lewis Carroll','Emece',20.00);
insert into libros values('Alicia en el pais de las
maravillas','Lewis Carroll','Plaza',35.00);
insert into libros values('Aprenda PHP','Mario
Molina','Siglo XXI',40.00);
insert into libros values('El
aleph','Borges','Emece',10.00);
insert into libros values('Ilusiones','Richard
Bach','Planeta',15.00);
insert into libros values('Java en 10 minutos','Mario
Molina','Siglo XXI',50.00);
insert into libros values('Martin Fierro','Jose
Hernandez','Planeta',20.00);
insert into libros values('Martin Fierro','Jose
Hernandez','Emece',30.00);
```

```
insert into libros values('Uno','Richard
Bach','Planeta',10.00);

-- Obtenemos la lista de los libros que han sido
publicados por distintas editoriales
--  empleando una consulta correlacionada:
 select distinct l1.titulo
  from libros as l1
  where l1.titulo in
  (select l2.titulo
    from libros as l2
    where l1.editorial <> l2.editorial);

-- El siguiente "join" retorna el mismo resultado:
select distinct l1.titulo
  from libros as l1
  join libros as l2
  on l1.titulo=l2.titulo
  where l1.editorial<>l2.editorial;

-- Buscamos todos los libros que tienen el mismo precio
-- que "El aleph" empleando subconsulta:
select titulo
  from libros
  where titulo<>'El aleph' and
  precio =
   (select precio
     from libros
     where titulo='El aleph');

-- Obtenemos la misma salida empleando "join":
select l1.titulo
  from libros as l1
  join libros  as l2
  on l1.precio=l2.precio
  where l2.titulo='el aleph' and
  l1.titulo<>l2.titulo;

-- Buscamos los libros cuyo precio supera el precio
promedio
-- de los libros por editorial:
select l1.titulo,l1.editorial,l1.precio
  from libros as l1
  where l1.precio >
```

```
   (select avg(l2.precio)
   from libros as l2
    where l1.editorial= l2.editorial);

-- Obtenemos la misma salida pero empleando un "join"
con "having":
 select l1.editorial,l1.titulo,l1.precio
  from libros as l1
  join libros as l2
   on l1.editorial=l2.editorial
   group by l1.editorial, l1.titulo, l1.precio
   having l1.precio > avg(l2.precio);
```

98 - Subconsulta en lugar de una tabla

Se pueden emplear subconsultas que retornen un conjunto de registros de varios campos en lugar de una tabla.

Se la denomina tabla derivada y se coloca en la cláusula "from" para que la use un "select" externo.

La tabla derivada debe ir entre paréntesis y tener un alias para poder referenciarla. La sintaxis básica es la siguiente:

```
 select ALIASdeTABLADERIVADA.CAMPO
 from (TABLADERIVADA) as ALIAS;
```

La tabla derivada es una subsonsulta.

Podemos probar la consulta que retorna la tabla derivada y luego agregar el "select" externo:

```
select f.*,
 (select sum(d.precio*cantidad)
  from Detalles as d
  where f.numero=d.numerofactura) as total
  from facturas as f;
```

La consulta anterior contiene una subconsulta correlacionada; retorna todos los datos de "facturas" y el monto total por factura de "detalles". Esta consulta retorna varios registros y varios campos y será la tabla derivada que emplearemos en la siguiente consulta:

```
 select td.numero,c.nombre,td.total
  from clientes as c
  join (select f.*,
   (select sum(d.precio*cantidad)
    from Detalles as d
    where f.numero=d.numerofactura) as total
  from facturas as f) as td
  on td.codigocliente=c.codigo;
```

La consulta anterior retorna, de la tabla derivada (referenciada con "td") el número de factura y el monto total, y de la tabla "clientes", el nombre del cliente. Note que este "join" no emplea 2 tablas, sino una tabla propiamente dicha y una tabla derivada, que es en realidad una subconsulta.

Servidor de SQL Server instalado en forma local.

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

```
if object_id('detalles') is not null
  drop table detalles;
if object_id('facturas') is not null
  drop table facturas;
if object_id('clientes') is not null
  drop table clientes;

create table clientes(
  codigo int identity,
  nombre varchar(30),
  domicilio varchar(30),
  primary key(codigo)
);

create table facturas(
  numero int not null,
  fecha datetime,
  codigocliente int not null,
  primary key(numero),
  constraint FK_facturas_cliente
   foreign key (codigocliente)
   references clientes(codigo)
   on update cascade
);
```

```
create table detalles(
  numerofactura int not null,
  numeroitem int not null,
  articulo varchar(30),
  precio decimal(5,2),
  cantidad int,
  primary key(numerofactura,numeroitem),
   constraint FK_detalles_numerofactura
   foreign key (numerofactura)
   references facturas(numero)
   on update cascade
   on delete cascade,
);

go

insert into clientes values('Juan Lopez','Colon 123');
insert into clientes values('Luis Torres','Sucre 987');
insert into clientes values('Ana Garcia','Sarmiento
576');

set dateformat ymd;

insert into facturas values(1200,'2018-01-15',1);
insert into facturas values(1201,'2018-01-15',2);
insert into facturas values(1202,'2018-01-15',3);
insert into facturas values(1300,'2018-01-20',1);

insert into detalles values(1200,1,'lapiz',1,100);
insert into detalles values(1200,2,'goma',0.5,150);
insert into detalles values(1201,1,'regla',1.5,80);
insert into detalles values(1201,2,'goma',0.5,200);
insert into detalles values(1201,3,'cuaderno',4,90);
insert into detalles values(1202,1,'lapiz',1,200);
insert into detalles values(1202,2,'escuadra',2,100);
insert into detalles values(1300,1,'lapiz',1,300);

-- Recuperar el número de factura, el código de cliente,
-- la fecha y la suma total de todas las facturas:
 select f.*,
  (select sum(d.precio*cantidad)
    from detalles as d
    where f.numero=d.numerofactura) as total
 from facturas as f;
```

```
-- Ahora utilizaremos el resultado de la consulta
anterior como una tabla
-- derivada que emplearemos en lugar de una tabla para
realizar un "join"
-- y recuperar el número de factura, el nombre del
cliente y
-- el monto total por factura:
 select td.numero,c.nombre,td.total
  from clientes as c
  join (select f.*,
   (select sum(d.precio*cantidad)
    from detalles as d
    where f.numero=d.numerofactura) as total
  from facturas as f) as td
  on td.codigocliente=c.codigo;
```

99 - Subconsulta (update - delete)

Dijimos que podemos emplear subconsultas en sentencias "insert", "update", "delete", además de "select".

La sintaxis básica para realizar actualizaciones con subconsulta es la siguiente:

```
 update TABLA set CAMPO=NUEVOVALOR
  where CAMPO= (SUBCONSULTA);
```

Actualizamos el precio de todos los libros de editorial "Emece":

```
 update libros set precio=precio+(precio*0.1)
  where codigoeditorial=
   (select codigo
     from editoriales
     where nombre='Emece');
```

La subconsulta retorna un único valor. También podemos hacerlo con un join.

La sintaxis básica para realizar eliminaciones con subconsulta es la siguiente:

```
 delete from TABLA
```

```
  where CAMPO in (SUBCONSULTA);
```

Eliminamos todos los libros de las editoriales que tiene publicados libros de "Juan Perez":

```
 delete from libros
  where codigoeditorial in
   (select e.codigo
    from editoriales as e
    join libros
    on codigoeditorial=e.codigo
    where autor='Juan Perez');
```

La subconsulta es una combinación que retorna una lista de valores que la consulta externa emplea al seleccionar los registros para la eliminación.

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 editoriales(
  codigo tinyint identity,
  nombre varchar(30),
  primary key (codigo)
);

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

go

insert into editoriales values('Planeta');
```

```
insert into editoriales values('Emece');
insert into editoriales values('Paidos');
insert into editoriales values('Siglo XXI');

insert into libros values('Uno','Richard Bach',1,15);
insert into libros values('Ilusiones','Richard
Bach',2,20);
insert into libros values('El aleph','Borges',3,10);
insert into libros values('Aprenda PHP','Mario
Molina',4,40);
insert into libros values('Poemas','Juan Perez',1,20);
insert into libros values('Cuentos','Juan Perez',3,25);
insert into libros values('Java en 10 minutos','Marcelo
Perez',2,30);

-- Actualizamos el precio de todos los libros de
editorial "Emece"
-- incrementándolos en un 10%:
update libros set precio=precio+(precio*0.1)
  where codigoeditorial=
   (select codigo
     from editoriales
     where nombre='Emece');

-- Eliminamos todos los libros de las editoriales que
tiene
-- publicados libros de "Juan Perez":
delete from libros
  where codigoeditorial in
   (select e.codigo
    from editoriales as e
    join libros
    on codigoeditorial=e.codigo
    where autor='Juan Perez');
```

100 - Subconsulta (insert)

Aprendimos que una subconsulta puede estar dentro de un "select", "update" y "delete"; también puede estar dentro de un "insert".

Podemos ingresar registros en una tabla empleando un "select".

La sintaxis básica es la siguiente:

```
 insert into TABLAENQUESEINGRESA (CAMPOSTABLA1)
  select (CAMPOSTABLACONSULTADA)
  from TABLACONSULTADA;
```

Un profesor almacena las notas de sus alumnos en una tabla llamada "alumnos". Tiene otra tabla llamada "aprobados", con algunos campos iguales a la tabla "alumnos" pero en ella solamente almacenará los alumnos que han aprobado el ciclo.

Ingresamos registros en la tabla "aprobados" seleccionando registros de la tabla "alumnos":

```
 insert into aprobados (documento,nota)
  select (documento,nota)
   from alumnos;
```

Entonces, se puede insertar registros en una tabla con la salida devuelta por una consulta a otra tabla; para ello escribimos la consulta y le anteponemos "insert into" junto al nombre de la tabla en la cual ingresaremos los registros y los campos que se cargarán (si se ingresan todos los campos no es necesario listarlos).

La cantidad de columnas devueltas en la consulta debe ser la misma que la cantidad de campos a cargar en el "insert".

Se pueden insertar valores en una tabla con el resultado de una consulta que incluya cualquier tipo de "join".

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

create table alumnos(
  documento char(8) not null,
  nombre varchar(30),
  nota decimal(4,2)
   constraint CK_alumnos_nota_valores check (nota>=0 and
nota <=10),
```

```
  primary key(documento),
);

create table aprobados(
  documento char(8) not null,
  nota decimal(4,2)
   constraint CK_aprobados_nota_valores check (nota>=0
and nota <=10),
  primary key(documento),
);

go


insert into alumnos values('30000000','Ana Acosta',8);
insert into alumnos values('30111111','Betina
Bustos',9);
insert into alumnos values('30222222','Carlos
Caseros',2.5);
insert into alumnos values('30333333','Daniel
Duarte',7.7);
insert into alumnos values('30444444','Estela
Esper',3.4);

-- Ingresamos registros en la tabla "aprobados"
seleccionando
-- registros de la tabla "alumnos":
insert into aprobados
  select documento,nota
   from alumnos
   where nota>=4;

select * from aprobados;
```