51 - Restricción unique

Hemos visto que las restricciones aplicadas a tablas aseguran valores únicos para cada registro.
Anteriormente aprendimos la restricción "primary key", otra restricción para las tablas es "unique".

La restricción "unique" impide la duplicación de claves alternas (no primarias), es decir, especifica que dos registros no puedan tener el mismo valor en un campo. Se permiten valores nulos. Se pueden aplicar varias restricciones de este tipo a una misma tabla, y pueden aplicarse a uno o varios campos que no sean clave primaria.

Se emplea cuando ya se estableció una clave primaria (como un número de legajo) pero se necesita asegurar que otros datos también sean únicos y no se repitan (como número de documento).

La sintaxis general es la siguiente:

```
alter table NOMBRETABLA
add constraint NOMBRERESTRICCION
unique (CAMPO);
```

Ejemplo:

```
alter table alumnos
 add constraint UQ_alumnos_documento
 unique (documento);
```

En el ejemplo anterior se agrega una restricción "unique" sobre el campo "documento" de la tabla "alumnos", esto asegura que no se pueda ingresar un documento si ya existe. Esta restricción permite valores nulos, asi que si se ingresa el valor "null" para el campo "documento", se acepta.

Por convención, cuando demos el nombre a las restricciones "unique" seguiremos la misma estructura: "UQ_NOMBRETABLA_NOMBRECAMPO". Quizá parezca innecesario colocar el nombre de la tabla, pero cuando empleemos varias tablas verá que es útil identificar las restricciones por tipo, tabla y campo.

Recuerde que cuando agregamos una restricción a una tabla que contiene información, SQL Server controla los datos existentes para confirmar que cumplen la condición de la restricción, si no los cumple, la restricción no se aplica y aparece un mensaje de error. En el caso del ejemplo anterior, si la tabla contiene números de documento duplicados, la restricción no podrá establecerse; si podrá establecerse si tiene valores nulos.

SQL Server controla la entrada de datos en inserciones y actualizaciones evitando que se ingresen valores duplicados.

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(
  legajo char(4) not null,
  apellido varchar(20),
  nombre varchar(20),
  documento char(8)
);

go

-- Agregamos una restricción "primary" para el campo "legajo":
alter table alumnos
 add constraint PK_alumnos_legajo
 primary key(legajo);

-- Agregamos una restricción "unique" para el campo "documento":
alter table alumnos
 add constraint UQ_alumnos_documento
 unique (documento);

insert into alumnos values('A111','Lopez','Ana','22222222');
insert into alumnos values('A123','Garcia','Maria','23333333');

exec sp_helpconstraint alumnos;
```

52 - Información de restricciones (sp_helpconstraint)

El procedimiento almacenado "sp_helpconstraint" seguido del nombre de una tabla muestra la información referente a todas las restricciones establecidas en dicha tabla, 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 (tema que veremos posteriormente).

- constraint_name: nombre de la restricción.

- delete_action: solamente es aplicable para restricciones de tipo "foreign key" (la veremos posteriormente).

- update_action: sólo es aplicable para restricciones de tipo "foreign key" (la veremos posteriormente).

- 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 "check" muestra la condición de chequeo; si es una restricción "default", el valor por defecto; si es una "primary key" o "unique" muestra el/ los campos a los que se aplicaron la restricción.

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(
  legajo char(4) not null,
  apellido varchar(20),
  nombre varchar(20),
  documento char(8),
  domicilio varchar(30),
  ciudad varchar(30),
  notafinal decimal(4,2)
);

go

-- Agregamos una restricción "primary" para el campo
"legajo":
alter table alumnos
 add constraint PK_alumnos_legajo
 primary key(legajo);

-- Agregamos una restricción "unique" para el campo
"documento"
```

```
alter table alumnos
 add constraint UQ_alumnos_documento
 unique (documento);

-- Agregamos una restricción "check" para que el campo
"notafinal"
-- admita solamente valores entre 0 y 10:
alter table alumnos
 add constraint CK_alumnos_nota
 check (notafinal>=0 and notafinal<=10);

-- Agregamos una restricción "default" para el campo
"ciudad":
alter table alumnos
 add constraint DF_alumnos_ciudad
 default 'Cordoba'
 for ciudad;

 -- Veamos las restricciones:
exec sp_helpconstraint alumnos;

-- Deshabilitamos la restricción "check":
alter table alumnos
  nocheck constraint CK_alumnos_nota;

 -- Veamos las restricciones:
exec sp_helpconstraint alumnos;
```

53 - Eliminar restricciones (alter table - drop)

Para eliminar una restricción, la sintaxis básica es la siguiente:

```
 alter table NOMBRETABLA
  drop NOMBRERESTRICCION;
```

Para eliminar la restricción "DF_libros_autor" de la tabla libros tipeamos:

```
 alter table libros
  drop DF_libros_autor;
```

Pueden eliminarse varias restricciones con una sola instrucción separándolas por comas.

Cuando eliminamos una tabla, todas las restricciones que fueron establecidas en ella, se eliminan también.

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 not null,
  titulo varchar(40),
  autor varchar(30),
  editorial varchar(15),
  precio decimal(6,2)
);

go

-- Definimos una restricción "primary key" para nuestra
tabla "libros" para asegurarnos
-- que cada libro tendrá un código diferente y único:
alter table libros
 add constraint PK_libros_codigo
 primary key(codigo);

-- Definimos una restricción "check" para asegurarnos
que el precio no será negativo:
alter table libros
 add constraint CK_libros_precio
 check (precio>=0);

-- Definimos una restricción "default" para el campo
"autor" para que almacene "Desconocido":
alter table libros
 add constraint DF_libros_autor
 default 'Desconocido'
 for autor;

-- Definimos una restricción "default" para el campo
"precio" para que almacene 0:
alter table libros
```

```
 add constraint DF_libros_precio
 default 0
 for precio;

-- Vemos las restricciones:
exec sp_helpconstraint libros;

-- Eliminamos la restricción "DF_libros_autor":
alter table libros
  drop DF_libros_autor;

-- Eliminamos la restricción "PK_libros_codigo":
alter table libros
  drop PK_libros_codigo;

exec sp_helpconstraint libros;
```

54 - Crear y asociar reglas (create rule - sp_bindrule)

Vimos que SQL Server ofrece varias alternativas para asegurar la integridad de datos, mediante el uso de:

1. RESTRICCIONES (constraints), que se establecen en tablas y campos y son controlados automáticamente por SQL Server. Hay 3 tipos:

I) DE LOS CAMPOS (hace referencia a los valores válidos para un campo determinado). Pueden ser:

a) DEFAULT: especifica un valor por defecto para un campo cuando no se inserta explícitamente en un comando "insert".

b) CHECK: especifica un rango de valores que acepta un campo, se emplea en inserciones y actualizaciones ("insert" y "update").

II) DE LA TABLA (asegura un identificador único para cada registro de una tabla). Hay 2 tipos:

a) PRIMARY KEY: identifica unívocamente cada uno de los registros; asegura que no haya valores duplicados ni valores nulos. Se crea un índice automáticamente.

b) UNIQUE: impide la duplicación de claves alternas (no primarias). Se permiten valores nulos. Se crea un índice automáticamente.

III) REFERENCIAL: lo veremos más adelante.

2. REGLAS (rules) y
3. VALORES PREDETERMINADOS (defaults).

Veamos las reglas.

Las reglas especifican los valores que se pueden ingresar en un campo, asegurando que los datos se encuentren en un intervalo de valores específico, coincidan con una lista de valores o sigan un patrón.

Una regla se asocia a un campo de una tabla (o a un tipo de dato definido por el usuario, tema que veremos posteriormente).

Un campo puede tener solamente UNA regla asociado a él.

Sintaxis básica es la siguiente:

```
create rule NOMBREREGLA
as @VARIABLE CONDICION
```

Entonces, luego de "create rule" se coloca el nombre de la regla, luego la palabra clave "as" seguido de una variable (a la cual la precede el signo arroba) y finalmente la condición.

Por convención, nombraremos las reglas comenzando con "RG", el nombre del campo al que se asocia y alguna palabra que haga referencia a la condición.

La variable puede tener cualquier nombre, pero debe estar precedido por el signo arroba (@), dicha variable será reemplazada por el valor del campo cuando se asocie.

La condición se refiere a los valores permitidos para inserciones y actualizaciones y puede contener cualquier expresión válida para una cláusula "where"; no puede hacer referencia a los campos de una tabla.

Creamos una regla para restringir los valores que se pueden ingresar en un campo "sueldo" de una tabla llamada "empleados", estableciendo un intervalo de valores:

```
 create rule RG_sueldo_intervalo
 as @sueldo between 100 and 1000
```

Luego de crear la regla, debemos asociarla a un campo ejecutando un procedimiento almacenado del sistema empleando la siguiente sintaxis básica:

```
 exec sp_bindrule NOMBREREGLA, 'TABLA.CAMPO';
```

Asociamos la regla creada anteriormente al campo "sueldo" de la tabla "empleados":

```
 exec sp_bindrule RG_sueldo_intervalo,
'empleados.sueldo';
```

Si intentamos agregar (o actualizar) un registro con valor para el campo "sueldo" que no esté en el intervalo de valores especificado en la regla, aparece un mensaje de error indicando que hay conflicto con la regla y la inserción (o actualización) no se realiza.

SQL Server NO controla los datos existentes para confirmar que cumplen con la regla como lo hace al aplicar restricciones; si no los cumple, la regla se asocia igualmente; pero al ejecutar una instrucción "insert" o "update" muestra un mensaje de error, es decir, actúa en inserciones y actualizaciones.

La regla debe ser compatible con el tipo de datos del campo al cual se asocia; si esto no sucede, SQL Server no lo informa al crear la regla ni al asociarla, pero al ejecutar una instrucción "insert" o "update" muestra un mensaje de error.

No se puede crear una regla para campos de tipo text, image, o timestamp.

Si asocia una nueva regla a un campo que ya tiene asociada otra regla, la nueva regla reeemplaza la asociación anterior; pero la primera regla no desaparece, solamente se deshace la asociación.

La sentencia "create rule" no puede combinarse con otras sentencias en un lote.

La función que cumple una regla es básicamente la misma que una restricción "check", las siguientes características explican algunas diferencias entre ellas:

- podemos definir varias restricciones "check" sobre un campo, un campo solamente puede tener una regla asociada a él;

- una restricción "check" se almacena con la tabla, cuando ésta se elimina, las restricciones también se borran. Las reglas son objetos diferentes e independientes de las tablas, si eliminamos una tabla, las asociaciones desaparecen, pero las reglas siguen existiendo en la base de datos;

- una restricción "check" puede incluir varios campos; una regla puede asociarse a distintos campos (incluso de distintas tablas);

- una restricción "check" puede hacer referencia a otros campos de la misma tabla, una regla no.

Un campo puede tener reglas asociadas a él y restricciones "check". Si hay conflicto entre ellas, SQL Server no lo informa al crearlas y/o asociarlas, pero al intentar ingresar un valor que alguna de ellas no permita, aparece un mensaje de error.

Con "sp_helpconstraint" podemos ver las reglas asociadas a los campos de una tabla.

Con "sp_help" podemos ver todos los objetos de la base de datos activa, incluyendo las reglas, en tal caso en la columna "Object_type" aparece "rule".

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 varchar(8) not null,
  nombre varchar(30),
  seccion varchar(20),
  fechaingreso datetime,
  fechanacimiento datetime,
```

```
  hijos tinyint,
  sueldo decimal(6,2)
);

go

-- Recuerde que las reglas son objetos independientes de
las tablas (no se eliminan al
-- borrar la tabla), así que debemos eliminarlas con las
siguientes intrucciones:
if object_id ('RG_documento_patron') is not null
  drop rule RG_documento_patron;
if object_id ('RG_empleados_seccion') is not null
  drop rule RG_empleados_seccion;
if object_id ('RG_empleados_fechaingreso') is not null
  drop rule RG_empleados_fechaingreso;
if object_id ('RG_hijos') is not null
  drop rule RG_hijos;
if object_id ('RG_empleados_sueldo') is not null
  drop rule RG_empleados_sueldo;
if object_id ('RG_empleados_sueldo2') is not null
  drop rule RG_empleados_sueldo2;

go

insert into empleados
   values('22222222','Ana Acosta','Contaduria','1990-10-
10','1972-10-10',2,700);
insert into empleados
   values('23333333','Carlos Costa','Contaduria','1990-
12-10','1972-05-04',0,750);
insert into empleados
   values('24444444','Daniel Duarte','Sistemas','1995-
05-05','1975-10-06',1,880);
insert into empleados
   values('25555555','Fabiola
Fuentes','Secretaria','1998-02-11','1978-02-08',3,550);
insert into empleados
   values('26666666','Gaston Garcia','Secretaria','1999-
05-08','1981-01-01',3,670);
insert into empleados
   values('27777777','Ines Irala','Gerencia','2000-04-
10','1985-12-12',0,6000);
```

```
go

-- Creamos una regla que establezca un patrón para el
documento:
create rule RG_documento_patron
   as @documento like '[0-9][0-9][0-9][0-9][0-9][0-9][0-
9][0-9]';

go

-- Ejecutamos el procedimiento almacenado del sistema
"sp_help" para
-- ver si la regla creada anteriormente fue creada:
exec sp_help;

-- Ejecutamos el procedimiento almacenado del sistema
"sp_helpconstraint" para ver si está asociada la regla
-- a algún campo de "empleados" (No aparece porque aún
no la asociamos):
exec sp_helpconstraint empleados;

-- Si ingresamos un registro con un documento que no
cumpla la regla, SQL Server lo acepta porque la regla
-- aún no está asociada al campo:
insert into empleados values('ab888888','Juan
Juarez','Secretaria','2001-04-11','1986-11-12',0,600);

-- Asociamos la regla "RG_documento_patron" al campo
"documento":
exec sp_bindrule RG_documento_patron,
'empleados.documento';

-- Volvemos a ejecutar "sp_helpconstraint" (aparece la
regla):
exec sp_helpconstraint empleados;

go

-- Creamos una regla para restringir los valores que se
pueden ingresar en un campo "seccion":
create rule RG_empleados_seccion
 as @seccion in
('Secretaria','Contaduria','Sistemas','Gerencia');
```

```
go

-- La asociamos al campo "seccion":
exec sp_bindrule RG_empleados_seccion,
'empleados.seccion';

go

-- Creamos una regla para restringir los valores que se
pueden ingresar en el campo "fechaingreso",
-- para que no sea posterior a la fecha actual:
create rule RG_empleados_fechaingreso
 as @fecha <= getdate();

go

-- Asociamos la regla anteriormente creada a los campos
"fechaingreso" y "fechanacimiento":
exec sp_bindrule RG_empleados_fechaingreso,
'empleados.fechaingreso';
exec sp_bindrule RG_empleados_fechaingreso,
'empleados.fechanacimiento';

go

-- Creamos una regla para restringir los valores que se
pueden ingresar en el campo "hijos":
create rule RG_hijos
 as @hijos between 0 and 20;

go

-- La asociamos al campo "hijos":
exec sp_bindrule RG_hijos, 'empleados.hijos';

go

-- Creamos una regla para restringir los valores que se
pueden ingresar en un campo "sueldo":
create rule RG_empleados_sueldo
 as @sueldo>0 and @sueldo<= 5000;

go
```

```
-- La asociamos al campo "sueldo":
exec sp_bindrule RG_empleados_sueldo,
'empleados.sueldo';

go

-- Creamos otra regla para restringir los valores que se
pueden ingresar en un campo "sueldo":
create rule RG_empleados_sueldo2
 as @sueldo>0 and @sueldo<= 7000;

go

-- La asociamos al campo "sueldo" (la nueva regla
reeemplaza la asociación anterior):
exec sp_bindrule RG_empleados_sueldo2,
'empleados.sueldo';

insert into empleados values('29999999','Luis
Lopez','Secretaria','2002-03-03','1990-09-09',0,6000);

exec sp_help;

exec sp_helpconstraint empleados;
```

55 - Eliminar y dasasociar reglas (sp_unbindrule - drop rule)

Para eliminar una regla, primero se debe deshacer la asociación, ejecutando el procedimiento almacenado del sistema "sp_unbindrule":

```
 exec sp_unbindrule 'TABLA.CAMPO';
```

No es posible eliminar una regla si está asociada a un campo. Si intentamos hacerlo, aparece un mensaje de error y la eliminación no se realiza.

Con la instrucción "drop rule" eliminamos la regla:

```
 drop rule NOMBREREGLA;
```

Quitamos la asociación de la regla "RG_sueldo_intervalo" con el campo "sueldo" de la tabla "empleados" tipeando:

```
 exec sp_unbindrule 'empleados.sueldo';
```

Luego de quitar la asociación la eliminamos:

```
 drop rule RG_sueldo_100a1000;
```

Si eliminamos una tabla, las asociaciones de reglas de sus campos desaparecen, pero las reglas siguen existiendo.

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 ('RG_sueldo_100a1000') is not null
   drop rule RG_sueldo_100a1000;

create table empleados(
  documento char(8),
  nombre varchar(30) not null,
  seccion varchar(20),
  sueldo decimal(6,2),
  primary key(documento)
);

go

-- Creamos una regla para restringir los valores que se
pueden ingresar
-- en un campo "sueldo":
create rule RG_sueldo_100a1000
   as @sueldo between 100 and 1000;

go

-- Asociamos la regla creada anteriormente al campo
"sueldo":
exec sp_bindrule RG_sueldo_100a1000, 'empleados.sueldo';

-- Vemos si la regla está asociada a algún campo de
"empleados":
exec sp_helpconstraint empleados;
```

```
-- Quitamos la asociación:
exec sp_unbindrule 'empleados.sueldo';

-- Ahora que hemos quitado la asociación, podemos
ingresar el valor
-- "1200" en el campo "sueldo":
insert into empleados values ('30111222','Pedro
Torres','Contaduria',1200);

-- Vemos si la regla está asociada a algún campo de
"empleados":
exec sp_helpconstraint empleados;

-- Ejecutamos el procedimiento "sp_help" para verificar
que la regla aún existe:
exec sp_help;

-- Ahora si podemos borrar la regla:
drop rule RG_sueldo_100a1000;
```

56 - Información de reglas (sp_help - sp_helpconstraint)

Podemos utilizar el procedimiento almacenado "sp_help" con el nombre del objeto del cual queremos información, en este caso el nombre de una regla:

```
 exec sp_help NOMBREREGLA;
```

muestra nombre, propietario, tipo y fecha de creación.

Con "sp_help", no sabemos si las reglas existentes están o no asociadas a algún campo.

"sp_helpconstraint" retorna una lista de todas las restricciones que tiene una tabla. Podemos ver las reglas asociadas a una tabla con este procedimiento almacenado:

```
 exec sp_helpconstraint NOMBRETABLA;
```

muestra la siguiente información:

- constraint_type: indica que es una regla con "RULE", nombrando el campo al que está asociada.

- constraint_name: nombre de la regla.

- constraint_keys: muestra el texto de la regla.

Para ver el texto de una regla empleamos el procedimiento almacenado "sp_helptext" seguido del nombre de la regla:

```
 exec sp_helptext NOMBREREGLA;
```

También se puede consultar la tabla del sistema "sysobjects", que nos muestra el nombre y varios datos de todos los objetos de la base de datos actual. La columna "xtype" indica el tipo de objeto, en caso de ser una regla aparece el valor "R":

```
 select * from sysobjects;
```

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

```
 select * from sysobjects
  where xtype='R' and-- tipo regla
  name like 'RG%';--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 ('RG_sueldo') is not null
   drop rule RG_sueldo;
if object_id ('RG_seccion_lista') is not null
   drop rule RG_seccion_lista;

create table empleados(
  documento char(8) not null,
  nombre varchar(30) not null,
  seccion varchar(20),
  sueldo decimal(6,2),
  primary key(documento)
);

go
```

```
-- Creamos una regla para el campo "sueldo":
create rule RG_sueldo
  as @sueldo between 100 and 1000;

go

-- Asociamos la regla creada anteriormente al campo
"sueldo":
exec sp_bindrule RG_sueldo, 'empleados.sueldo';

go

-- Creamos una regla con una lista de valores para
"seccion":
create rule RG_seccion_lista
 as @seccion in ('Sistemas','Secretaria','Contaduria');

go

exec sp_help RG_sueldo;

exec sp_helpconstraint empleados;

exec sp_bindrule RG_seccion_lista, 'empleados.seccion';

exec sp_helpconstraint empleados;

exec sp_helptext "RG_seccion_lista";

-- Deshacemos la asociación de la regla "RG_sueldo" y la
eliminamos:
exec sp_unbindrule 'empleados.sueldo';
drop rule RG_sueldo;

exec sp_help RG_sueldo;

select * from sysobjects
  where xtype='R' and
  name like '%seccion%';
```

57 - Valores predeterminados (create default)

Hemos visto que para mantener la integridad declarativa se emplean restricciones, reglas (que hemos estudiado en secciones anteriores) y valores predeterminados.

Veamos los valores predeterminados.

Los valores predeterminados se asocian con uno o varios campos (o tipos de datos definidos por el usuario); se definen una sola vez y se pueden usar muchas veces.

Si no se coloca un valor cuando se ingresan datos, el valor predeterminado especifica el valor del campo al que está asociado.

Sintaxis básica:

```
create default NOMBREVALORPREDETERMINADO
 as VALORPREDETERMINADO;
```

"VALORPREDETERMINADO" no puede hacer referencia a campos de una tabla (u otros objetos) y debe ser compatible con el tipo de datos y longitud del campo al cual se asocia; si esto no sucede, SQL Server no lo informa al crear el valor predeterminado ni al asociarlo, pero al ejecutar una instrucción "insert" muestra un mensaje de error.

En el siguiente ejemplo creamos un valor predeterminado llamado "VP_datodesconocido' con el valor "Desconocido":

```
create default VP_datodesconocido
 as 'Desconocido'
```

Luego de crear un valor predeterminado, debemos asociarlo a un campo (o a un tipo de datos definido por el usuario) ejecutando el procedimiento almacenado del sistema "sp_bindefault":

```
exec sp_bindefault NOMBRE, 'NOMBRETABLA.CAMPO';
```

La siguiente sentencia asocia el valor predeterminado creado anteriormente al campo "domicilio" de la tabla "empleados":

```
exec sp_bindefault VP_datodesconocido,
'empleados.domicilio';
```

Podemos asociar un valor predeterminado a varios campos. Asociamos el valor predeterminado "VP_datodesconocido" al campo "barrio" de la tabla "empleados":

```
 exec sp_bindefault VP_datodesconocido,
'empleados.barrio';
```

La función que cumple un valor predeterminado es básicamente la misma que una restricción "default", las siguientes características explican algunas semejanzas y diferencias entre ellas:

- un campo solamente puede tener definida UNA restricción "default", un campo solamente puede tener UN valor predeterminado asociado a él,

- una restricción "default" se almacena con la tabla, cuando ésta se elimina, las restricciones también. Los valores predeterminados son objetos diferentes e independientes de las tablas, si eliminamos una tabla, las asociaciones desaparecen, pero los valores predeterminados siguen existiendo en la base de datos.

- una restricción "default" se establece para un solo campo; un valor predeterminado puede asociarse a distintos campos (inclusive, de diferentes tablas).

- una restricción "default" no puede establecerse sobre un campo "identity", tampoco un valor predeterminado.

No se puede asociar un valor predeterminado a un campo que tiene una restricción "default".

Un campo con un valor predeterminado asociado puede tener reglas asociadas a él y restricciones "check". Si hay conflicto entre ellas, SQL Server no lo informa al crearlas y/o asociarlas, pero al intentar ingresar un valor que alguna de ellas no permita, aparece un mensaje de error.

La sentencia "create default" no puede combinarse con otra sentencia en un mismo lote.

Si asocia a un campo que ya tiene asociado un valor predeterminado otro valor predeterminado, la nueva asociación reemplaza a la anterior.

Veamos otros ejemplos.
Creamos un valor predeterminado que inserta el valor "0" en un campo de tipo numérico:

```
 create default VP_cero
  as 0;
```

En el siguiente creamos un valor predeterminado que inserta ceros con el formato válido para un número de teléfono:

```
 create default VP_telefono
 as '(0000)0-000000';
```

Con "sp_helpconstraint" podemos ver los valores predeterminados asociados a los campos de una tabla.

Con "sp_help" podemos ver todos los objetos de la base de datos activa, incluyendo los valores predeterminados, en tal caso en la columna "Object_type" aparece "default".

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 ('VP_cero') is not null
   drop default VP_cero;
if object_id ('VP_100') is not null
   drop default VP_100;
if object_id ('VP_datodesconocido') is not null
   drop default VP_datodesconocido;
if object_id ('VP_telefono') is not null
   drop default VP_telefono;

create table empleados(
  nombre varchar(30),
  domicilio varchar(30),
  barrio varchar(15),
  telefono char(14),
  sueldo decimal(6,2)
);

go

insert into empleados default values;
```

```
select * from empleados;

go

-- Creamos un valor predeterminado con el valor
"Desconocido":
create default VP_datodesconocido
  as 'Desconocido';

go

-- Lo asociamos al campo "domicilio":
exec sp_bindefault VP_datodesconocido,
'empleados.domicilio';

-- Lo asociamos al campo "barrio":
exec sp_bindefault VP_datodesconocido,
'empleados.barrio';

insert into empleados default values;

select * from empleados;

go

-- Creamos un valor predeterminado que inserta el valor
"0":
create default VP_cero
  as 0;

go

-- Lo asociamos al campo "sueldo":
exec sp_bindefault VP_cero, 'empleados.sueldo';

insert into empleados default values;

select * from empleados;

go

-- Creamos un valor predeterminado que inserta el valor
"100":
```

```
create default VP_100
  as 100;

go

-- Lo asociamos al campo "sueldo"
-- Recuerde que si asociamos a un campo que ya tiene
asociado un valor
-- predeterminado otro valor predeterminado, la nueva
asociación reemplaza a la anterior
exec sp_bindefault VP_100, 'empleados.sueldo';

insert into empleados default values;

select * from empleados;

exec sp_helpconstraint empleados;

exec sp_help;

go

-- Creamos un valor predeterminado que inserta ceros con
el formato válido
-- para un campo número de teléfono:
create default VP_telefono
 as '(0000)0-000000';

go

-- La asociamos al campo "telefono" de la tabla
"empleados":
exec sp_bindefault VP_telefono,'empleados.telefono';

insert into empleados default values;

select * from empleados;

exec sp_helpconstraint empleados;
```

58 - Desasociar y eliminar valores predeterminados

Un valor predeterminado no puede eliminarse si no se ha desasociado previamente.

Para deshacer una asociación empleamos el procedimiento almacenado "sp_unbindefault" seguido de la tabla y campo al que está asociado:

```
exec sp_unbindefault 'TABLA.CAMPO';
```

Quitamos la asociación al campo "sueldo" de la tabla "empleados":

```
exec sp_unbindefault 'empleados.sueldo';
```

Con la instrucción "drop default" podemos eliminar un valor predeterminado:

```
drop default NOMBREVALORPREDETERMINADO;
```

Eliminamos el valor predeterminado llamado "VP_cero":

```
drop default VP_cero;
```

Si eliminamos una tabla, las asociaciones de valores predeterminados de sus campos desaparecen, pero los valores predeterminados siguen existiendo.

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 ('VP_cero') is not null
   drop default VP_cero;
if object_id ('VP_datodesconocido') is not null
   drop default VP_datodesconocido;

create table empleados(
  nombre varchar(30),
  domicilio varchar(30),
  barrio varchar(15),
  sueldo decimal(6,2)
```

```
);

go

-- Creamos un valor predeterminado que inserta el valor "0":
create default VP_cero
  as 0;

go

-- Lo asociamos al campo "sueldo":
exec sp_bindefault VP_cero, 'empleados.sueldo';

go

-- Creamos un valor predeterminado con el valor "Desconocido":
create default VP_datodesconocido
  as 'Desconocido';

go

-- Lo asociamos al campo "domicilio" y al campo "barrio":
exec sp_bindefault VP_datodesconocido, 'empleados.domicilio';
exec sp_bindefault VP_datodesconocido, 'empleados.barrio';

-- Veamos los valores predeterminados asociados a los campos de la tabla
"empleados":
exec sp_helpconstraint empleados;

-- Quitamos la asociación al campo "barrio":
exec sp_unbindefault 'empleados.barrio';

exec sp_helpconstraint empleados;

exec sp_help;

-- Aun no podemos eliminarlo porque está asociado al campo "domicilio",
-- quitamos la asociación y luego lo eliminamos:
exec sp_unbindefault 'empleados.domicilio';

drop default VP_datodesconocido;
```

59 - Información de valores predeterminados

Para obtener información de los valores predeterminados podemos emplear los mismos procedimientos almacenados que usamos para las reglas.

Si empleamos "sp_help", vemos todos los objetos de la base de datos activa (incluyendo los valores predeterminados); en la columna "Object_type" (tipo de objeto) muestra "default".

Si al procedimiento almacenado "sp_help" le agregamos el nombre de un valor predeterminado, nos muestra el nombre, propietario, tipo y fecha de creación:

```
exec sp_help NOMBREVALORPREDETERMINADO;
```

Con "sp_help", no sabemos si los valores predeterminados existentes están o no asociadas a algún campo.

"sp_helpconstraint" retorna una lista de todas las restricciones que tiene una tabla. También los valores predeterminados asociados; muestra la siguiente información:

- constraint_type: indica que es un valor predeterminado con "DEFAULT", nombrando el campo al que está asociado.

- constraint_name: nombre del valor predeterminado.

- constraint_keys: muestra el texto del valor predeterminado.

Con "sp_helptext" seguido del nombre de un valor predeterminado podemos ver el texto de cualquier valor predeterminado:

```
exec sp_helptext NOMBREVALORPREDETERMINADO;
```

También se puede consultar la tabla del sistema "sysobjects", que nos muestra el nombre y varios datos de todos los objetos de la base de datos actual. La columna "xtype" indica el tipo de objeto, en caso de ser un valor predeterminado aparece el valor "D":

```
select * from sysobjects;
```

Si queremos ver todos los valores predeterminados creados por nosotros, podemos tipear:

```
select * from sysobjects
 where xtype='D' and-- tipo valor predeterminado
 name like 'VP%';--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 ('VP_sueldo') is not null
   drop default VP_sueldo;
if object_id ('VP_seccion') is not null
   drop default Vp_seccion;

create table empleados(
  documento char(8) not null,
  nombre varchar(30) not null,
  seccion varchar(20),
  sueldo decimal(6,2),
  primary key(documento)
);

go

-- Creamos un valor predeterminado para el campo
"sueldo":
create default VP_sueldo
 as 500;

go

-- Asociamos el valor predeterminado creado
anteriormente al campo "sueldo":
exec sp_bindefault VP_sueldo, 'empleados.sueldo';

go

-- Creamos un valor predeterminado para "seccion":
create default VP_seccion
 as 'Secretaria';

 go

-- Veamos todos los objetos de la base de datos activa:
exec sp_help;

-- Si agregamos al procedimiento almacenado "sp_help" el
nombre
-- del valor predeterminado del cual queremos
información:
exec sp_help VP_sueldo;
```

```
-- Para ver los valores predeterminados asociados a la
tabla "empleados" tipeamos:
exec sp_helpconstraint empleados;

-- Asociamos el valor predeterminado a la tabla:
exec sp_bindefault VP_seccion, 'empleados.seccion';

exec sp_helpconstraint empleados;

exec sp_helptext VP_seccion;

-- Deshacemos la asociación del valor predeterminado
"VP_sueldo" y lo eliminamos:
exec sp_unbindefault'empleados.sueldo';
drop default VP_sueldo;

exec sp_help VP_sueldo;

-- Vemos si el valor predeterminado "VP_seccion" existe
consultando la tabla "sysobjects":
select * from sysobjects
  where xtype='D' and
  name like '%seccion%';
```

60 - Indices

SQL Server accede a los datos de dos maneras:

1. recorriendo las tablas; comenzando el principio y extrayendo los registros que cumplen las condiciones de la consulta.
2. empleando índices; recorriendo la estructura de árbol del índice para localizar los registros y extrayendo los que cumplen las condiciones de la consulta.

Los índices se emplean para facilitar la obtención de información de una tabla. El indice de una tabla desempeña la misma función que el índice de un libro: permite encontrar datos rápidamente; en el caso de las tablas, localiza registros.

Una tabla se indexa por un campo (o varios).

Un índice posibilita el acceso directo y rápido haciendo más eficiente las búsquedas. Sin índice, SQL Server debe recorrer secuencialmente toda la tabla para encontrar un registro.

El objetivo de un indice es acelerar la recuperación de información. La indexación es una técnica que optimiza el acceso a los datos, mejora el rendimiento acelerando las consultas y otras operaciones. Es útil cuando la tabla contiene miles de registros, cuando se realizan operaciones de ordenamiento y agrupamiento y cuando se combinan varias tablas (tema que veremos más adelante).

La desventaja es que consume espacio en el disco y genera costo de mantenimiento (tiempo y recursos).

Los índices más adecuados son aquellos creados con campos que contienen valores únicos.

Es importante identificar el o los campos por los que sería útil crear un índice, aquellos campos por los cuales se realizan búsqueda con frecuencia: claves primarias, claves externas o campos que combinan tablas.

No se recomienda crear índices por campos que no se usan con frecuencia en consultas o no contienen valores únicos.

SQL Server permite crear dos tipos de índices: 1) agrupados y 2) no agrupados.

61 - Indices agrupados y no agrupados (clustered y nonclustered)

Dijimos que SQL Server permite crear dos tipos de índices: 1) agrupados (clustered) y 2) no agrupados (nonclustered).

1) Un INDICE AGRUPADO es similar a una guía telefónica, los registros con el mismo valor de campo se agrupan juntos. Un índice agrupado determina la secuencia de almacenamiento de los registros en una tabla. Se utilizan para campos por los que se realizan busquedas con frecuencia o se accede siguiendo un orden.
Una tabla sólo puede tener UN índice agrupado.
El tamaño medio de un índice agrupado es aproximadamente el 5% del tamaño de la tabla.

2) Un INDICE NO AGRUPADO es como el índice de un libro, los datos se almacenan en un lugar diferente al del índice, los punteros indican el lugar de almacenamiento de los elementos indizados en la tabla.
Un índice no agrupado se emplea cuando se realizan distintos tipos de busquedas frecuentemente, con campos en los que los datos son únicos.
Una tabla puede tener hasta 249 índices no agrupados.

Si no se especifica un tipo de índice, de modo predeterminado será no agrupado.

Los campos de tipo text, ntext e image no se pueden indizar.

Es recomendable crear los índices agrupados antes que los no agrupados, porque los primeros modifican el orden físico de los registros, ordenándolos secuencialmente.

La diferencia básica entre índices agrupados y no agrupados es que los registros de un índice agrupado están ordenados y almacenados de forma secuencial en función de su clave.

SQL Server crea automaticamente índices cuando se crea una restricción "primary key" o "unique" en una tabla.
Es posible crear índices en las vistas.

Resumiendo, los índices facilitan la recuperación de datos, permitiendo el acceso directo y acelerando las búsquedas, consultas y otras operaciones que optimizan el rendimiento general.

62 - Creación de índices

Para crear índices empleamos la instrucción "create index".

La sintaxis básica es la siguiente:

```
create TIPODEINDICE index NOMBREINDICE
on TABLA(CAMPO);
```

"TIPODEINDICE" indica si es agrupado (clustered) o no agrupado (nonclustered). Si no especificamos crea uno No agrupado.
Independientemente de si es agrupado o no, también se puede especificar que sea "unique", es decir, no haya valores repetidos. Si se intenta crear un

índice unique para un campo que tiene valores duplicados, SQL Server no lo permite.

En este ejemplo se crea un índice agrupado único para el campo "codigo" de la tabla "libros":

```
create unique clustered index I_libros_codigo
on libros(codigo);
```

Para identificar los índices fácilmente, podemos agregar un prefijo al nombre del índice, por ejemplo "I" y luego el nombre de la tabla y/o campo.

En este ejemplo se crea un índice no agrupado para el campo "titulo" de la tabla "libros":

```
create nonclustered index I_libros_titulo
on libros(titulo);
```

Un índice puede tener más de un campo como clave, son índices compuestos. Los campos de un índice compuesto tienen que ser de la misma tabla (excepto cuando se crea en una vista - tema que veremos posteriormente).

Creamos un índice compuesto para el campo "autor" y "editorial":

```
create index I_libros_autoreditorial
on libros(autor,editorial);
```

SQL Server crea automáticamente índices cuando se establece una restricción "primary key" o "unique" en una tabla. Al crear una restricción "primary key", si no se especifica, el índice será agrupado (clustered) a menos que ya exista un índice agrupado para dicha tabla. Al crear una restricción "unique", si no se especifica, el índice será no agrupado (non-clustered).

Ahora podemos entender el resultado del procedimiento almacenado "sp_helpconstraint" cuando en la columna "constraint_type" mostraba el tipo de índice seguido de las palabras "clustered" o "non_clustered".

Puede especificarse que un índice sea agrupado o no agrupado al agregar estas restricciones.
Agregamos una restricción "primary key" al campo "codigo" de la tabla "libros" especificando que cree un índice NO agrupado:

```
 alter table libros
  add constraint PK_libros_codigo
  primary key nonclustered (codigo);
```

Para ver los indices de una tabla:

```
 exec sp_helpindex libros;
```

Muestra el nombre del índice, si es agrupado (o no), primary (o unique) y el campo por el cual se indexa.

Todos los índices de la base de datos activa se almacenan en la tabla del sistema "sysindexes", podemos consultar dicha tabla tipeando:

```
 select name from sysindexes;
```

Para ver todos los índices de la base de datos activa creados por nosotros podemos tipear la siguiente consulta:

```
 select name from sysindexes
  where name like 'I_%';
```

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(15)
);

go

-- Creamos un índice agrupado único para el campo
"codigo" de la tabla "libros":
create unique clustered index I_libros_codigo
 on libros(codigo);

-- Creamos un índice no agrupado para el campo "titulo":
```

```
create nonclustered index I_libros_titulo
 on libros(titulo);

-- Veamos los indices de "libros":
exec sp_helpindex libros;

-- Creamos una restricción "primary key" al campo
"codigo" especificando
-- que cree un índice NO agrupado:
alter table libros
  add constraint PK_libros_codigo
  primary key nonclustered (codigo);

-- Verificamos que creó un índice automáticamente:
exec sp_helpindex libros;

-- Analicemos la información que nos muestra
"sp_helpconstraint":
exec sp_helpconstraint libros;

-- Creamos un índice compuesto para el campo "autor" y
"editorial":
create index I_libros_autoreditorial
 on libros(autor,editorial);

-- Consultamos la tabla "sysindexes":
select name from sysindexes;

-- Veamos los índices de la base de datos activa creados
por nosotros
-- podemos tipear la siguiente consulta:
select name from sysindexes
  where name like 'I_%';
```

63 - Regenerar índices

Vimos que para crear índices empleamos la instrucción "create index".

Empleando la opción "drop_existing" junto con "create index" permite regenerar un índice, con ello evitamos eliminarlo y volver a crearlo. La sintaxis es la siguiente:

```
 create TIPODEINDICE index NOMBREINDICE
```

```
 on TABLA(CAMPO)
 with drop_existing;
```

También podemos modificar alguna de las características de un índice con esta opción, a saber:

- tipo: cambiándolo de no agrupado a agrupado (siempre que no exista uno agrupado para la misma tabla). No se puede convertir un índice agrupado en No agrupado.

- campo: se puede cambiar el campo por el cual se indexa, agregar campos, eliminar algún campo de un índice compuesto.

- único: se puede modificar un índice para que los valores sean únicos o dejen de serlo.

En este ejemplo se crea un índice no agrupado para el campo "titulo" de la tabla "libros":

```
 create nonclustered index I_libros
 on libros(titulo);
```

Regeneramos el índice "I_libros" y lo convertimos a agrupado:

```
 create clustered index I_libros
 on libros(titulo)
 with drop_existing;
```

Agregamos un campo al índice "I_libros":

```
 create clustered index I_libros
 on libros(titulo,editorial)
 with drop_existing;
```

Esta opción no puede emplearse con índices creados a partir de una restricción "primary key" o "unique".

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(15)
);

go

-- Creamos un índice no agrupado para el campo "titulo":
create nonclustered index I_libros_titulo
 on libros(titulo);

exec sp_helpindex libros;

-- Vamos a agregar el campo "autor" al índice
"I_libros_titulo"
-- y vemos si se modificó:
create index I_libros_titulo
 on libros(titulo,autor)
 with drop_existing;

exec sp_helpindex libros;

-- Lo convertimos en agrupado:
create clustered index I_libros_titulo
 on libros(titulo,autor)
 with drop_existing;

exec sp_helpindex libros;

-- Quitamos un campo "autor":
create clustered index I_libros_titulo
 on libros(titulo)
 with drop_existing;

exec sp_helpindex libros;
```

64 - Eliminar índices

Los índices creados con "create index" se eliminan con "drop index"; la siguiente es la sintaxis básica:

```
 drop index NOMBRETABLA.NOMBREINDICE;
```

Eliminamos el índice "I_libros_titulo":

```
 drop index libros.I_libros_titulo;
```

Los índices que SQL Server crea automáticamente al establecer una restricción "primary key" o "unique" no pueden eliminarse con "drop index", se eliminan automáticamente cuando quitamos la restricción.

Podemos averiguar si existe un índice para eliminarlo, consultando la tabla del sistema "sysindexes":

```
 if exists (select name from sysindexes
  where name = 'NOMBREINDICE')
   drop index NOMBRETABLA.NOMBREINDICE;
```

Eliminamos el índice "I_libros_titulo" si existe:

```
 if exists (select *from sysindexes
  where name = 'I_libros_titulo')
   drop index libros.I_libros_titulo;
```

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(15)
);

go

-- Creamos un índice para el campo "titulo":
create index I_libros_titulo
 on libros(titulo);
```

```
exec sp_helpindex libros;

-- Eliminamos el índice "I_libros_titulo":
drop index libros.I_libros_titulo;

exec sp_helpindex libros;

-- Solicitamos que se elimine el índice
"I_libros_titulo" si existe:
if exists (select name from sysindexes
  where name = 'I_libros_titulo')
   drop index libros.I_libros_titulo;
```

65 - Trabajar con varias tablas

Hasta el momento hemos trabajado con una sola tabla, pero generalmente, se trabaja con más de una.

Para evitar la repetición de datos y ocupar menos espacio, se separa la información en varias tablas. Cada tabla almacena parte de la información que necesitamos registrar.

Por ejemplo, los datos de nuestra tabla "libros" podrían separarse en 2 tablas, una llamada "libros" y otra "editoriales" que guardará la información de las editoriales.
En nuestra tabla "libros" haremos referencia a la editorial colocando un código que la identifique.

Veamos:

```
 create table libros(
  codigo int identity,
  titulo varchar(40) not null,
  autor varchar(30) not null default 'Desconocido',
  codigoeditorial tinyint not null,
  precio decimal(5,2),
  primary key (codigo)
 );

 create table editoriales(
  codigo tinyint identity,
  nombre varchar(20) not null,
  primary key(codigo)
```

```
);
```

De esta manera, evitamos almacenar tantas veces los nombres de las editoriales en la tabla "libros" y guardamos el nombre en la tabla "editoriales"; para indicar la editorial de cada libro agregamos un campo que hace referencia al código de la editorial en la tabla "libros" y en "editoriales".

Al recuperar los datos de los libros con la siguiente instrucción:

```
select * from libros;
```

vemos que en el campo "codigoeditorial" aparece el código, pero no sabemos el nombre de la editorial.
Para obtener los datos de cada libro, incluyendo el nombre de la editorial, necesitamos consultar ambas tablas, traer información de las dos.

Cuando obtenemos información de más de una tabla decimos que hacemos un "join" (combinación).

Veamos un ejemplo:

```
select * from libros
 join editoriales
 on libros.codigoeditorial=editoriales.codigo;
```

Resumiendo: si distribuimos la información en varias tablas evitamos la redundancia de datos y ocupamos menos espacio físico en el disco. Un join es una operación que relaciona dos o más tablas para obtener un resultado que incluya datos (campos y registros) de ambas; las tablas participantes se combinan según los campos comunes a ambas tablas.

Hay tres tipos de combinaciones. En los siguientes capítulos explicamos cada una de ellas.

66 - Combinación interna (inner join)

Un join es una operación que relaciona dos o más tablas para obtener un resultado que incluya datos (campos y registros) de ambas; las tablas participantes se combinan según los campos comunes a ambas tablas.

Hay tres tipos de combinaciones:

1. combinaciones internas (inner join o join),
2. combinaciones externas y
3. combinaciones cruzadas.

También es posible emplear varias combinaciones en una consulta "select", incluso puede combinarse una tabla consigo misma.

La combinación interna emplea "join", que es la forma abreviada de "inner join". Se emplea para obtener información de dos tablas y combinar dicha información en una salida.

La sintaxis básica es la siguiente:

```
select CAMPOS
 from TABLA1
 join TABLA2
 on CONDICIONdeCOMBINACION;
```

Ejemplo:

```
select * from libros
 join editoriales
 on codigoeditorial=editoriales.codigo;
```

Analicemos la consulta anterior.

- especificamos los campos que aparecerán en el resultado en la lista de selección;

- indicamos el nombre de la tabla luego del "from" ("libros");

- combinamos esa tabla con "join" y el nombre de la otra tabla ("editoriales"); se especifica qué tablas se van a combinar y cómo;

- cuando se combina información de varias tablas, es necesario especificar qué registro de una tabla se combinará con qué registro de la otra tabla, con "on". Se debe especificar la condición para enlazarlas, es decir, el campo por el cual se combinarán, que tienen en común.
"on" hace coincidir registros de ambas tablas basándose en el valor de tal campo, en el ejemplo, el campo "codigoeditorial" de "libros" y el campo "codigo" de "editoriales" son los que enlazarán ambas tablas. Se emplean campos comunes, que deben tener tipos de datos iguales o similares.

La condicion de combinación, es decir, el o los campos por los que se van a combinar (parte "on"), se especifica según las claves primarias y externas.

Note que en la consulta, al nombrar el campo usamos el nombre de la tabla también. Cuando las tablas referenciadas tienen campos con igual nombre, esto es necesario para evitar confusiones y ambiguedades al momento de referenciar un campo. En el ejemplo, si no especificamos "editoriales.codigo" y solamente tipeamos "codigo", SQL Server no sabrá si nos referimos al campo "codigo" de "libros" o de "editoriales" y mostrará un mensaje de error indicando que "codigo" es ambiguo.

Entonces, si las tablas que combinamos tienen nombres de campos iguales, DEBE especificarse a qué tabla pertenece anteponiendo el nombre de la tabla al nombre del campo, separado por un punto (.).

Si una de las tablas tiene clave primaria compuesta, al combinarla con la otra, en la cláusula "on" se debe hacer referencia a la clave completa, es decir, la condición referenciará a todos los campos clave que identifican al registro.

Se puede incluir en la consulta join la cláusula "where" para restringir los registros que retorna el resultado; también "order by", "distinct", etc..

Se emplea este tipo de combinación para encontrar registros de la primera tabla que se correspondan con los registros de la otra, es decir, que cumplan la condición del "on". Si un valor de la primera tabla no se encuentra en la segunda tabla, el registro no aparece.

Para simplificar la sentencia podemos usar un alias para cada tabla:

```
select l.codigo,titulo,autor,nombre
 from libros as l
 join editoriales as e
 on l.codigoeditorial=e.codigo;
```

En algunos casos (como en este ejemplo) el uso de alias es para fines de simplificación y hace más legible la consulta si es larga y compleja, pero en algunas consultas es absolutamente necesario.

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) default 'Desconocido',
  codigoeditorial tinyint not null,
  precio decimal(5,2)
);

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

go

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

insert into libros values('El aleph','Borges',2,20);
insert into libros values('Martin Fierro','Jose
Hernandez',1,30);
insert into libros values('Aprenda PHP','Mario
Molina',3,50);
insert into libros values('Java en 10
minutos',default,3,45);

-- Recuperamos los datos de libros:
select * from libros;

-- Realizamos un join para obtener datos de ambas tablas
-- (titulo, autor y nombre de la editorial):
select titulo, autor, nombre
  from libros
  join editoriales
  on codigoeditorial=editoriales.codigo;
```

```
 -- Mostramos el código del libro, título, autor, nombre
de la
 -- editorial y el precio realizando un join y empleando
alias:
select l.codigo,titulo,autor,nombre,precio
  from libros as l
  join editoriales as e
  on codigoeditorial=e.codigo;

-- Realizamos la misma consulta anterior agregando un
"where"
-- para obtener solamente los libros de la editorial
"Siglo XXI":
select l.codigo,titulo,autor,nombre,precio
  from libros as l
  join editoriales as e
  on codigoeditorial=e.codigo
  where e.nombre='Siglo XXI';

-- Obtenemos título, autor y nombre de la editorial,
-- esta vez ordenados por título:
select titulo,autor,nombre
  from libros as l
  join editoriales as e
  on codigoeditorial=e.codigo
  order by titulo;
```

67 - Combinación externa izquierda (left join)

Vimos que una combinación interna (join) encuentra registros de la primera tabla que se correspondan con los registros de la segunda, es decir, que cumplan la condición del "on" y si un valor de la primera tabla no se encuentra en la segunda tabla, el registro no aparece.

Si queremos saber qué registros de una tabla NO encuentran correspondencia en la otra, es decir, no existe valor coincidente en la segunda, necesitamos otro tipo de combinación, "outer join" (combinación externa).

Las combinaciones externas combinan registros de dos tablas que cumplen la condición, más los registros de la segunda tabla que no la cumplen; es decir, muestran todos los registros de las tablas relacionadas, aún cuando no haya valores coincidentes entre ellas.

Este tipo de combinación se emplea cuando se necesita una lista completa de los datos de una de las tablas y la información que cumple con la condición. Las combinaciones externas se realizan solamente entre 2 tablas.

Hay tres tipos de combinaciones externas: "left outer join", "right outer join" y "full outer join"; se pueden abreviar con "left join", "right join" y "full join" respectivamente.

Vamos a estudiar las primeras.

Se emplea una combinación externa izquierda para mostrar todos los registros de la tabla de la izquierda. Si no encuentra coincidencia con la tabla de la derecha, el registro muestra los campos de la segunda tabla seteados a "null".

En el siguiente ejemplo solicitamos el título y nombre de la editorial de los libros:

```
select titulo,nombre
from editoriales as e
left join libros as l
on codigoeditorial = e.codigo;
```

El resultado mostrará el título y nombre de la editorial; las editoriales de las cuales no hay libros, es decir, cuyo código de editorial no está presente en "libros" aparece en el resultado, pero con el valor "null" en el campo "titulo".

Es importante la posición en que se colocan las tablas en un "left join", la tabla de la izquierda es la que se usa para localizar registros en la tabla de la derecha.

Entonces, un "left join" se usa para hacer coincidir registros en una tabla (izquierda) con otra tabla (derecha); si un valor de la tabla de la izquierda no encuentra coincidencia en la tabla de la derecha, se genera una fila extra (una por cada valor no encontrado) con todos los campos correspondientes a la tabla derecha seteados a "null". La sintaxis básica es la siguiente:

```
select CAMPOS
from TABLAIZQUIERDA
left join TABLADERECHA
on CONDICION;
```

En el siguiente ejemplo solicitamos el título y el nombre la editorial, la sentencia es similar a la anterior, la diferencia está en el orden de las tablas:

```
  select titulo,nombre
  from libros as l
  left join editoriales as e
  on codigoeditorial = e.codigo;
```

El resultado mostrará el título del libro y el nombre de la editorial; los títulos cuyo código de editorial no está presente en "editoriales" aparecen en el resultado, pero con el valor "null" en el campo "nombre".

Un "left join" puede tener clausula "where" que restringa el resultado de la consulta considerando solamente los registros que encuentran coincidencia en la tabla de la derecha, es decir, cuyo valor de código está presente en "libros":

```
 select titulo,nombre
  from editoriales as e
  left join libros as l
  on e.codigo=codigoeditorial
  where codigoeditorial is not null;
```

También podemos mostrar las editoriales que NO están presentes en "libros", es decir, que NO encuentran coincidencia en la tabla de la derecha:

```
 select titulo,nombre
  from editoriales as e
  left join libros as l
  on e.codigo=codigoeditorial
  where codigoeditorial is null;
```

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) default 'Desconocido',
  codigoeditorial tinyint not null,
  precio decimal(5,2)
);
create table editoriales(
  codigo tinyint identity,
  nombre varchar(20),
  primary key (codigo)
);

go

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

insert into libros values('El aleph','Borges',1,20);
insert into libros values('Martin Fierro','Jose
Hernandez',1,30);
insert into libros values('Aprenda PHP','Mario
Molina',2,50);
insert into libros values('Java en 10
minutos',default,4,45);

-- Combinación izquierda para obtener los datos de los
libros,
-- incluyendo el nombre de la editorial.
-- Las editoriales de las cuales no hay libros, es
decir, cuyo código
-- de editorial no está presente en "libros" aparece en
el resultado,
-- pero con el valor "null" en el campo "titulo":
select titulo,nombre
  from editoriales as e
  left join libros as l
  on codigoeditorial = e.codigo;

-- Realizamos la misma consulta anterior pero cambiamos
el orden de las tablas:
select titulo,nombre
  from libros as l
  left join editoriales as e
  on codigoeditorial = e.codigo;
```

```
-- Consulta considerando solamente los registros que
encuentran coincidencia en la
-- tabla de la derecha, es decir, cuyo valor de código
está presente en "libros":
select titulo,nombre
  from editoriales as e
  left join libros as l
  on e.codigo=codigoeditorial
  where codigoeditorial is not null;

-- Mostramos las editoriales que no están presentes en
"libros", es decir,
-- que no encuentran coincidencia en la tabla de la
derecha:
select titulo,nombre
  from editoriales as e
  left join libros as l
  on e.codigo=codigoeditorial
  where codigoeditorial is null;
```

68 - Combinación externa derecha (right join

Vimos que una combinación externa izquierda (left join) encuentra registros de la tabla izquierda que se correspondan con los registros de la tabla derecha y si un valor de la tabla izquierda no se encuentra en la tabla derecha, el registro muestra los campos correspondientes a la tabla de la derecha seteados a "null".

Una combinación externa derecha ("right outer join" o "right join") opera del mismo modo sólo que la tabla derecha es la que localiza los registros en la tabla izquierda.

En el siguiente ejemplo solicitamos el título y nombre de la editorial de los libros empleando un "right join":

```
  select titulo,nombre
  from libros as l
  right join editoriales as e
  on codigoeditorial = e.codigo;
```

El resultado mostrará el título y nombre de la editorial; las editoriales de las cuales no hay libros, es decir, cuyo código de editorial no está presente en "libros" aparece en el resultado, pero con el valor "null" en el campo "titulo".

Es FUNDAMENTAL tener en cuenta la posición en que se colocan las tablas en los "outer join". En un "left join" la primera tabla (izquierda) es la que busca coincidencias en la segunda tabla (derecha); en el "right join" la segunda tabla (derecha) es la que busca coincidencias en la primera tabla (izquierda).

En la siguiente consulta empleamos un "left join" para conseguir el mismo resultado que el "right join" anterior":

```
select titulo,nombre
 from editoriales as e
 left join libros as l
 on codigoeditorial = e.codigo;
```

Note que la tabla que busca coincidencias ("editoriales") está en primer lugar porque es un "left join"; en el "right join" precedente, estaba en segundo lugar.

Un "right join" hace coincidir registros en una tabla (derecha) con otra tabla (izquierda); si un valor de la tabla de la derecha no encuentra coincidencia en la tabla izquierda, se genera una fila extra (una por cada valor no encontrado) con todos los campos correspondientes a la tabla izquierda seteados a "null". La sintaxis básica es la siguiente:

```
 select CAMPOS
 from TABLAIZQUIERDA
 right join TABLADERECHA
 on CONDICION;
```

Un "right join" también puede tener cláusula "where" que restringa el resultado de la consulta considerando solamente los registros que encuentran coincidencia en la tabla izquierda:

```
select titulo,nombre
 from libros as l
 right join editoriales as e
 on e.codigo=codigoeditorial
 where codigoeditorial is not null;
```

Mostramos las editoriales que NO están presentes en "libros", es decir, que NO encuentran coincidencia en la tabla de la derecha empleando un "right join":

```
 select titulo,nombre
  from libros as l
  right join editoriales as e
  on e.codigo=codigoeditorial
  where codigoeditorial is null;
```

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) default 'Desconocido',
  codigoeditorial tinyint not null,
  precio decimal(5,2)
);

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

go

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

insert into libros values('El aleph','Borges',1,20);
insert into libros values('Martin Fierro','Jose
Hernandez',1,30);
```

```
insert into libros values('Aprenda PHP','Mario
Molina',2,50);
insert into libros values('Java en 10
minutos',default,4,45);

-- Solicitamos el título y nombre de la editorial de los
libros
-- empleando un "right join":
select titulo,nombre
  from libros as l
  right join editoriales as e
  on codigoeditorial = e.codigo;

-- Realizamos la misma consulta anterior agregando un
"where" que restringa
-- el resultado considerando solamente los registros que
encuentran
-- coincidencia en la tabla izquierda:
 select titulo,nombre
  from libros as l
  right join editoriales as e
  on e.codigo=codigoeditorial
  where codigoeditorial is not null;

-- Mostramos las editoriales que no están presentes en
"libros"
-- (que no encuentran coincidencia en "editoriales"):
 select titulo,nombre
  from libros as l
  right join editoriales as e
  on e.codigo=codigoeditorial
  where codigoeditorial is null;
```

69 - Combinación externa completa (full join)

Vimos que un "left join" encuentra registros de la tabla izquierda que se correspondan con los registros de la tabla derecha y si un valor de la tabla izquierda no se encuentra en la tabla derecha, el registro muestra los campos correspondientes a la tabla de la derecha seteados a "null". Aprendimos también que un "right join" opera del mismo modo sólo que la tabla derecha es la que localiza los registros en la tabla izquierda.

Una combinación externa completa ("full outer join" o "full join") retorna todos los registros de ambas tablas. Si un registro de una tabla izquierda no encuentra coincidencia en la tabla derecha, las columnas correspondientes a campos de la tabla derecha aparecen seteadas a "null", y si la tabla de la derecha no encuentra correspondencia en la tabla izquierda, los campos de esta última aparecen conteniendo "null".

Veamos un ejemplo:

```
 select titulo,nombre
  from editoriales as e
  full join libros as l
  on codigoeditorial = e.codigo;
```

La salida del "full join" precedente muestra todos los registros de ambas tablas, incluyendo los libros cuyo código de editorial no existe en la tabla "editoriales" y las editoriales de las cuales no hay correspondencia en "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;
if object_id('editoriales') is not null
  drop table editoriales;

create table libros(
  codigo int identity,
  titulo varchar(40),
  autor varchar(30) default 'Desconocido',
  codigoeditorial tinyint not null,
  precio decimal(5,2)
);

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

go
```

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

insert into libros values('El aleph','Borges',1,20);
insert into libros values('Martin Fierro','Jose
Hernandez',1,30);
insert into libros values('Aprenda PHP','Mario
Molina',2,50);
insert into libros values('Java en 10
minutos',default,4,45);

-- Combinación externa completa para obtener todos los
registros de ambas tablas,
-- incluyendo los libros cuyo código de editorial no
existe en la tabla "editoriales"
-- y las editoriales de las cuales no hay
correspondencia en "libros":
 select titulo,nombre
  from editoriales as e
  full join libros as l
  on codigoeditorial = e.codigo;
```

70 - Combinaciones cruzadas (cross join)

Vimos que hay tres tipos de combinaciones:
1) combinaciones internas (join), 2) combinaciones externas (left, right y full join) y 3) combinaciones cruzadas.

Las combinaciones cruzadas (cross join) muestran todas las combinaciones de todos los registros de las tablas combinadas. Para este tipo de join no se incluye una condición de enlace. Se genera el producto cartesiano en el que el número de filas del resultado es igual al número de registros de la primera tabla multiplicado por el número de registros de la segunda tabla, es decir, si hay 5 registros en una tabla y 6 en la otra, retorna 30 filas.

La sintaxis básica es ésta:

```
 select CAMPOS
  from TABLA1
  cross join TABLA2;
```

Veamos un ejemplo. Un pequeño restaurante almacena los nombres y precios de sus comidas en una tabla llamada "comidas" y en una tabla denominada "postres" los mismos datos de sus postres.

Si necesitamos conocer todas las combinaciones posibles para un menú, cada comida con cada postre, empleamos un "cross join":

```
 select c.nombre as 'plato principal', p.nombre as
'postre'
  from comidas as c
  cross join postres as p;
```

La salida muestra cada plato combinado con cada uno de los postres.

Como cualquier tipo de "join", puede emplearse una cláusula "where" que condicione la salida.

Servidor de SQL Server instalado en forma local.

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

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

create table comidas(
  codigo tinyint identity,
  nombre varchar(30),
  precio decimal(4,2)
);

create table postres(
  codigo tinyint identity,
  nombre varchar(30),
  precio decimal(4,2)
);

go

insert into comidas values('ravioles',5);
insert into comidas values('tallarines',4);
insert into comidas values('milanesa',7);
```

```
insert into comidas values('cuarto de pollo',6);

insert into postres values('flan',2.5);
insert into postres values('porcion torta',3.5);

-- Combinar los registros de ambas tablas para mostrar
-- los distintos menúes que ofrece. Lo hacemos usando un
"cross join":
select c.nombre as 'plato principal',
  p.nombre as 'postre',
  c.precio+p.precio as 'total'
  from comidas as c
  cross join postres as p;
```

71 - Autocombinación

Dijimos que es posible combinar una tabla consigo misma.

Un pequeño restaurante tiene almacenadas sus comidas en una tabla llamada "comidas" que consta de los siguientes campos:

```
- nombre varchar(20),
- precio decimal (4,2) y
- rubro char(6)-- que indica con 'plato' si es un plato
principal y 'postre' si es postre.
```

Podemos obtener la combinación de platos empleando un "cross join" con una sola tabla:

```
 select c1.nombre as 'plato principal',
  c2.nombre as postre,
  c1.precio+c2.precio as total
  from comidas as c1
  cross join comidas as c2;
```

En la consulta anterior aparecen filas duplicadas, para evitarlo debemos emplear un "where":

```
 select c1.nombre as 'plato principal',
  c2.nombre as postre,
  c1.precio+c2.precio as total
  from comidas as c1
  cross join comidas as c2
```

```
  where c1.rubro='plato' and
  c2.rubro='postre';
```

En la consulta anterior se empleó un "where" que especifica que se combine "plato" con "postre".

En una autocombinación se combina una tabla con una copia de si misma. Para ello debemos utilizar 2 alias para la tabla. Para evitar que aparezcan filas duplicadas, debemos emplear un "where".

También se puede realizar una autocombinación con "join":

```
 select c1.nombre as 'plato principal',
  c2.nombre as postre,
  c1.precio+c2.precio as total
  from comidas as c1
  join comidas as c2
  on c1.codigo<>c2.codigo
  where c1.rubro='plato' and
  c2.rubro='postre';
```

Para que no aparezcan filas duplicadas se agrega un "where".

Servidor de SQL Server instalado en forma local.

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

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

create table comidas(
  codigo int identity,
  nombre varchar(30),
  precio decimal(4,2),
  rubro char(6),-- 'plato'=plato principal',
'postre'=postre
  primary key(codigo)
);

go

insert into comidas values('ravioles',5,'plato');
insert into comidas values('tallarines',4,'plato');
```

```
insert into comidas values('milanesa',7,'plato');
insert into comidas values('cuarto de pollo',6,'plato');
insert into comidas values('flan',2.5,'postre');
insert into comidas values('porcion
torta',3.5,'postre');

-- Realizamos un "cross join"
-- Note que aparecen filas duplicadas, por ejemplo,
"ravioles" se
-- combina con "ravioles" y la combinación "ravioles-
flan"
-- se repite como "flan- ravioles"
select c1.nombre as 'plato principal',
  c2.nombre as postre,
  c1.precio+c2.precio as total
  from comidas as c1
  cross join comidas as c2;

-- Debemos especificar que combine el rubro "plato" con
"postre":
select c1.nombre as 'plato principal',
  c2.nombre as postre,
  c1.precio+c2.precio as total
  from comidas as c1
  cross join comidas as c2
  where c1.rubro='plato' and
  c2.rubro='postre';

-- También se puede realizar una autocombinación con
"join":
select c1.nombre as 'plato principal',
  c2.nombre as postre,
  c1.precio+c2.precio as total
  from comidas as c1
  join comidas as c2
  on c1.codigo<>c2.codigo
  where c1.rubro='plato' and
  c2.rubro='postre';
```

72 - Combinaciones y funciones de agrupamiento

Podemos usar "group by" y las funciones de agrupamiento con combinaciones de tablas.

Para ver la cantidad de libros de cada editorial consultando la tabla "libros" y "editoriales", tipeamos:

```
 select nombre as editorial,
  count(*) as cantidad
  from editoriales as e
  join libros as l
  on codigoeditorial=e.codigo
  group by e.nombre;
```

Note que las editoriales que no tienen libros no aparecen en la salida porque empleamos un "join".

Empleamos otra función de agrupamiento con "left join". Para conocer el mayor precio de los libros de cada editorial usamos la función "max()", hacemos un "left join" y agrupamos por nombre de la editorial:

```
 select nombre as editorial,
  max(precio) as 'mayor precio'
  from editoriales as e
  left join libros as l
  on codigoeditorial=e.codigo
  group by nombre;
```

En la sentencia anterior, mostrará, para la editorial de la cual no haya libros, el valor "null" en la columna calculada.

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 not null,
  precio decimal(5,2)
);
```

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

go

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

insert into libros values('El aleph','Borges',1,20);
insert into libros values('Martin Fierro','Jose
Hernandez',1,30);
insert into libros values('Aprenda PHP','Mario
Molina',3,50);
insert into libros values('Uno','Richard Bach',3,15);
insert into libros values('Java en 10
minutos',default,4,45);

-- Contamos la cantidad de libros de cada editorial
consultando ambas tablas:
select nombre as editorial,
  count(*) as cantidad
  from editoriales as e
  join libros as l
  on codigoeditorial=e.codigo
  group by e.nombre;

-- Buscamos el libro más costoso de cada editorial con
un "left join":
select nombre as editorial,
  max(precio) as 'mayor precio'
  from editoriales as e
  left join libros as l
  on codigoeditorial=e.codigo
  group by nombre;
```

73 - Combinación de más de dos tablas

Podemos hacer un "join" con más de dos tablas.

Cada join combina 2 tablas. Se pueden emplear varios join para enlazar varias tablas. Cada resultado de un join es una tabla que puede combinarse con otro join.

La librería almacena los datos de sus libros en tres tablas: libros, editoriales y autores.
En la tabla "libros" un campo "codigoautor" hace referencia al autor y un campo "codigoeditorial" referencia la editorial.

Para recuperar todos los datos de los libros empleamos la siguiente consulta:

```
 select titulo,a.nombre,e.nombre
  from autores as a
  join libros as l
  on codigoautor=a.codigo
  join editoriales as e  on codigoeditorial=e.codigo;
```

Analicemos la consulta anterior. Indicamos el nombre de la tabla luego del "from" ("autores"), combinamos esa tabla con la tabla "libros" especificando con "on" el campo por el cual se combinarán; luego debemos hacer coincidir los valores para el enlace con la tabla "editoriales" enlazándolas por los campos correspondientes. Utilizamos alias para una sentencia más sencilla y comprensible.

Note que especificamos a qué tabla pertenecen los campos cuyo nombre se repiten en las tablas, esto es necesario para evitar confusiones y ambiguedades al momento de referenciar un campo.

Note que no aparecen los libros cuyo código de autor no se encuentra en "autores" y cuya editorial no existe en "editoriales", esto es porque realizamos una combinación interna.

Podemos combinar varios tipos de join en una misma sentencia:

```
 select titulo,a.nombre,e.nombre
  from autores as a
  right join libros as l
  on codigoautor=a.codigo
  left join editoriales as e  on
codigoeditorial=e.codigo;
```

En la consulta anterior solicitamos el título, autor y editorial de todos los libros que encuentren o no coincidencia con "autores" ("right join") y a ese resultado lo combinamos con "editoriales", encuentren o no coincidencia.

Es posible realizar varias combinaciones para obtener información de varias tablas. Las tablas deben tener claves externas relacionadas con las tablas a combinar.

En consultas en las cuales empleamos varios "join" es importante tener en cuenta el orden de las tablas y los tipos de "join"; recuerde que la tabla resultado del primer join es la que se combina con el segundo join, no la segunda tabla nombrada. En el ejemplo anterior, el "left join" no se realiza entre las tablas "libros" y "editoriales" sino entre el resultado del "right join" y la tabla "editoriales".

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

create table libros(
  codigo int identity,
  titulo varchar(40),
  codigoautor int not null,
  codigoeditorial tinyint not null,
  precio decimal(5,2),
  primary key(codigo)
);

create table autores(
  codigo int identity,
  nombre varchar(20),
  primary key (codigo)
);


create table editoriales(
```

```
  codigo tinyint identity,
  nombre varchar(20),
  primary key (codigo)
);

go

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

insert into autores values ('Richard Bach');
insert into autores values ('Borges');
insert into autores values ('Jose Hernandez');
insert into autores values ('Mario Molina');
insert into autores values ('Paenza');

insert into libros values('El aleph',2,2,20);
insert into libros values('Martin Fierro',3,1,30);
insert into libros values('Aprenda PHP',4,3,50);
insert into libros values('Uno',1,1,15);
insert into libros values('Java en 10 minutos',0,3,45);
insert into libros values('Matematica estas
ahi',0,0,15);
insert into libros values('Java de la A a la Z',4,0,50);

-- Recuperamos todos los datos de los libros consultando
las tres tablas:
select titulo,a.nombre,e.nombre,precio
  from autores as a
  join libros as l
  on codigoautor=a.codigo
  join editoriales as e
  on codigoeditorial=e.codigo;

-- Podemos combinar varios tipos de join en una misma
sentencia:
select titulo,a.nombre,e.nombre,precio
  from autores as a
  right join libros as l
  on codigoautor=a.codigo
  left join editoriales as e
  on codigoeditorial=e.codigo;
```

74 - Combinaciones con update y delete

Las combinaciones no sólo se utilizan con la sentencia "select", también podemos emplearlas con "update" y "delete".

Podemos emplear "update" o "delete" con "join" para actualizar o eliminar registros de una tabla consultando otras tablas.

En el siguiente ejemplo aumentamos en un 10% los precios de los libros de cierta editorial, necesitamos un "join" para localizar los registros de la editorial "Planeta" en la tabla "libros":

```
 update libros set precio=precio+(precio*0.1)
 from libros
 join editoriales as e
 on codigoeditorial=e.codigo
 where nombre='Planeta';
```

Eliminamos todos los libros de editorial "Emece":

```
 delete libros
  from libros
  join editoriales
  on codigoeditorial = editoriales.codigo
  where editoriales.nombre='Emece';
```

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) default 'Desconocido',
  codigoeditorial tinyint not null,
  precio decimal(5,2)
);
create table editoriales(
```

```
  codigo tinyint identity,
  nombre varchar(20),
  primary key (codigo)
);

go

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

insert into libros values('El aleph','Borges',2,20);
insert into libros values('Martin Fierro','Jose
Hernandez',1,30);
insert into libros values('Aprenda PHP','Mario
Molina',3,50);
insert into libros values('Java en 10
minutos',default,3,45);

-- Aumentamos en un 10% los precios de los libros de
editorial "Planeta":
update libros set precio=precio+(precio*0.1)
  from libros
  join editoriales as e
  on codigoeditorial=e.codigo
  where nombre='Planeta';

select titulo,autor,e.nombre,precio
  from libros as l
  join editoriales as e
  on codigoeditorial=e.codigo;

-- Eliminamos todos los libros de editorial "Emece":
delete libros
  from libros
  join editoriales
  on codigoeditorial = editoriales.codigo
  where editoriales.nombre='Emece';

select titulo,autor,e.nombre,precio
  from libros as l
  join editoriales as e
  on codigoeditorial=e.codigo;
```

75 - Clave foránea

Un campo que no es clave primaria en una tabla y sirve para enlazar sus valores con otra tabla en la cual es clave primaria se denomina clave foránea, externa o ajena.

En el ejemplo de la librería en que utilizamos las tablas "libros" y "editoriales" con estos campos:

```
 libros: codigo (clave primaria), titulo, autor,
codigoeditorial, precio y
 editoriales: codigo (clave primaria), nombre.
```

el campo "codigoeditorial" de "libros" es una clave foránea, se emplea para enlazar la tabla "libros" con "editoriales" y es clave primaria en "editoriales" con el nombre "codigo".

Las claves foráneas y las claves primarias deben ser del mismo tipo para poder enlazarse. Si modificamos una, debemos modificar la otra para que los valores se correspondan.

Cuando alteramos una tabla, debemos tener cuidado con las claves foráneas. Si modificamos el tipo, longitud o atributos de una clave foránea, ésta puede quedar inhabilitada para hacer los enlaces.

Entonces, una clave foránea es un campo (o varios) empleados para enlazar datos de 2 tablas, para establecer un "join" con otra tabla en la cual es clave primaria.