Las funciones matemáticas realizan operaciones con expresiones numéricas y retornan un resultado, operan con tipos de datos numéricos.

Microsoft SQL Server tiene algunas funciones para trabajar con números. Aquí presentamos algunas.

-abs(x): retorna el valor absoluto del argumento "x". Ejemplo:

```
select abs(-20);
```

retorna 20.

-ceiling(x): redondea hacia arriba el argumento "x". Ejemplo:

```
select ceiling(12.34);
```

retorna 13.

-floor(x): redondea hacia abajo el argumento "x". Ejemplo:

```
select floor(12.34);
```

retorna 12.

- %: %: devuelve el resto de una división. Ejemplos:

```
select 10%3;
```

retorna 1.

```
select 10%2;
```

retorna 0.

-power(x,y): retorna el valor de "x" elevado a la "y" potencia. Ejemplo:

```
select power(2,3);
```

retorna 8.

-round(numero,longitud): retorna un número redondeado a la longitud especificada. "longitud" debe ser tinyint, smallint o int. Si "longitud" es positivo, el número de decimales es redondeado según "longitud"; si es negativo, el número es redondeado desde la parte entera según el valor de "longitud". Ejemplos:

```
select round(123.456,1);
```

retorna "123.400", es decir, redondea desde el primer decimal.

```
select round(123.456,2);
```

retorna "123.460", es decir, redondea desde el segundo decimal.

```
select round(123.456,-1);
```

retorna "120.000", es decir, redondea desde el primer valor entero (hacia la izquierda).

select round(123.456,-2);

retorna "100.000", es decir, redondea desde el segundo valor entero (hacia la izquierda).

-sign(x): si el argumento es un valor positivo devuelve 1;-1 si es negativo y si es 0, 0.

-square(x): retorna el cuadrado del argumento. Ejemplo:

select square(3); retorna 9.

-srqt(x): devuelve la raiz cuadrada del valor enviado como argumento.

SQL Server dispone de funciones trigonométricas que retornan radianes.

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

Servidor de SQL Server instalado en forma local.

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

```
if object_id ('libros') is not null
  drop table libros;
create table libros(
  codigo int identity,
  titulo varchar(40) not null,
  autor varchar(20) default 'Desconocido',
  editorial varchar(20),
  precio decimal(6,2),
  primary key (codigo)
);
go
insert into libros (titulo,autor,editorial,precio)
  values('El aleph','Borges','Emece',25.33);
insert into libros
  values('Java en 10 minutos','Mario Molina','Siglo XXI',50.65);
insert into libros (titulo,autor,editorial,precio)
  values('Alicia en el pais de las maravillas','Lewis Carroll','Emece',19.95);
```

-- Vamos a mostrar los precios de los libros redondeando el valor hacia abajo y hacia arriba:

```
select titulo,autor,precio,
  floor(precio) as abajo,
  ceiling(precio) as arriba
  from libros;
```

Funciones de fecha y hora

Microsoft SQL Server ofrece algunas funciones para trabajar con fechas y horas. Estas son algunas:

- getdate(): retorna la fecha y hora actuales. Ejemplo:

```
 select getdate();
```

- datepart(partedefecha,fecha): retorna la parte específica de una fecha, el año, trimestre, día, hora, etc.

Los valores para "partedefecha" pueden ser: year (año), quarter (cuarto), month (mes), day (dia), week (semana), hour (hora), minute (minuto), second (segundo) y millisecond (milisegundo). Ejemplos:

```
 select datepart(month,getdate());
```

retorna el número de mes actual;

```
 select datepart(day,getdate());
```

retorna el día actual;

```
 select datepart(hour,getdate());
```

retorna la hora actual;

- datename(partedefecha,fecha): retorna el nombre de una parte específica de una fecha. Los valores para "partedefecha" pueden ser los mismos que se explicaron anteriormente. Ejemplos:

```
 select datename(month,getdate());
```

retorna el nombre del mes actual;

```
 select datename(day,getdate());
```

- dateadd(partedelafecha,numero,fecha): agrega un intervalo a la fecha especificada, es decir, retorna una fecha adicionando a la fecha enviada como tercer argumento, el intervalo de tiempo indicado por el primer parámetro, tantas veces como lo indica el segundo parámetro. Los valores para el primer argumento pueden ser: year (año), quarter (cuarto), month (mes), day (dia), week (semana), hour (hora), minute (minuto), second (segundo) y millisecond (milisegundo). Ejemplos:

```
 select dateadd(day,3,'1980/11/02');
```

retorna "1980/11/05", agrega 3 días.

select dateadd(month,3,'1980/11/02');

retorna "1981/02/02", agrega 3 meses.

select dateadd(hour,2,'1980/11/02');

retorna "1980/02/02 2:00:00", agrega 2 horas.

select dateadd(minute,16,'1980/11/02');

retorna "1980/02/02 00:16:00", agrega 16 minutos.

- datediff(partedelafecha,fecha1,fecha2): calcula el intervalo de tiempo (según el primer argumento) entre las 2 fechas. El resultado es un valor entero que corresponde a fecha2-fecha1. Los valores de "partedelafecha) pueden ser los mismos que se especificaron anteriormente. Ejemplos:

select datediff (day,'2005/10/28','2006/10/28');

retorna 365 (días).

select datediff(month,'2005/10/28','2006/11/29');

retorna 13 (meses).

- day(fecha): retorna el día de la fecha especificada. Ejemplo:

select day(getdate());

- month(fecha): retorna el mes de la fecha especificada. Ejemplo:

select month(getdate());

- year(fecha): retorna el año de la fecha especificada. Ejemplo:

select year(getdate());

Se pueden emplear estas funciones enviando como argumento el nombre de un campo de tipo datetime o smalldatetime.

Servidor de SQL Server instalado en forma local.

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

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

create table libros(
  titulo varchar(40) not null,
  autor varchar(20) default 'Desconocido',
```

```
  editorial varchar(20),
  edicion datetime,
  precio decimal(6,2)
);
go
set dateformat ymd;
insert into libros
  values('El aleph','Borges','Emece','1980/10/10',25.33);
insert into libros
  values('Java en 10 minutos','Mario Molina','Siglo XXI','2000/05/05',50.65);
insert into libros
values('Alicia en el pais de las maravillas','Lewis Carroll','Emece','2000/08/09',19.95);
insert into libros
  values('Aprenda PHP','Mario Molina','Siglo XXI','2000/02/04',45);
-- Mostramos el título del libro y el año de edición:
select titulo, datepart (year,edicion) from libros;
-- Mostramos el título del libro y el nombre del mes de edición:
select titulo, datename (month,edicion) from libros;

-- Mostramos el título del libro y los años que tienen de editados:
select titulo, datediff(year,edicion,getdate()) from libros;

-- Muestre los títulos de los libros que se editaron el día 9, de cualquier mes de cualquier año:
 select titulo from libros
  where datepart(day,edicion)=9;
```

Ordenar registros (order by)

Podemos ordenar el resultado de un "select" para que los registros se muestren ordenados por algún campo, para ello usamos la cláusula "order by".

La sintaxis básica es la siguiente:

```
select * from NOMBRETABLA
 order by CAMPO;
```

Por ejemplo, recuperamos los registros de la tabla "libros" ordenados por el título:

```
select *from libros
 order by titulo;
```

Aparecen los registros ordenados alfabéticamente por el campo especificado.

También podemos colocar el número de orden del campo por el que queremos que se ordene en lugar de su nombre, es decir, referenciar a los campos por su posición en la lista de selección. Por ejemplo, queremos el resultado del "select" ordenado por "precio":

```
select titulo,autor,precio
 from libros order by 3;
```

Por defecto, si no aclaramos en la sentencia, los ordena de manera ascendente (de menor a mayor). Podemos ordenarlos de mayor a menor, para ello agregamos la palabra clave "desc":

```
select * libros
 order by editorial desc;
```

También podemos ordenar por varios campos, por ejemplo, por "titulo" y "editorial":

```
select * from libros
 order by titulo,editorial;
```

Incluso, podemos ordenar en distintos sentidos, por ejemplo, por "titulo" en sentido ascendente y "editorial" en sentido descendente:

```
select * from libros
 order by titulo asc, editorial desc;
```

Debe aclararse al lado de cada campo, pues estas palabras claves afectan al campo inmediatamente anterior.

Es posible ordenar por un campo que no se lista en la selección.

Se permite ordenar por valores calculados o expresiones.

La cláusula "order by" no puede emplearse para campos text, ntext e image.

Servidor de SQL Server instalado en forma local.

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

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

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

insert into libros (titulo,autor,editorial,precio)
  values('El aleph','Borges','Emece',25.33);
insert into libros
  values('Java en 10 minutos','Mario Molina','Siglo XXI',50.65);
insert into libros (titulo,autor,editorial,precio)
  values('Alicia en el pais de las maravillas','Lewis Carroll','Emece',19.95);
insert into libros (titulo,autor,editorial,precio)
  values('Alicia en el pais de las maravillas','Lewis Carroll','Planeta',15);

-- Recuperamos los registros ordenados por el título:
select * from libros
 order by titulo;

-- Ordenamos los registros por el campo "precio", referenciando el campo
```

```
-- por su posición en la lista de selección:
select titulo,autor,precio
 from libros order by 3;

-- Los ordenamos por "editorial", de mayor a menor empleando "desc":
select * from libros
 order by editorial desc;

-- Ordenamos por dos campos:
 select * from libros
 order by titulo,editorial;

-- Ordenamos en distintos sentidos:
 select * from libros
 order by titulo asc, editorial desc;

-- Ordenamos por un campo que no se lista en la selección:
 select titulo, autor
 from libros
 order by precio;

-- Ordenamos por un valor calculado:
 select titulo, autor, editorial,
       precio+(precio*0.1) as 'precio con descuento'
  from libros
  order by 4;
```

Operadores lógicos (and - or - not)

Hasta el momento, hemos aprendido a establecer una condición con "where" utilizando operadores relacionales. Podemos establecer más de una condición con la cláusula "where", para ello aprenderemos los operadores lógicos.

Son los siguientes:

```
- and, significa "y",
- or, significa "y/o",
- not, significa "no", invierte el resultado
- (), paréntesis
```

Los operadores lógicos se usan para combinar condiciones.

Si queremos recuperar todos los libros cuyo autor sea igual a "Borges" y cuyo precio no supere los 20 pesos, necesitamos 2 condiciones:

```
select * from libros
 where (autor='Borges') and
 (precio<=20);
```

Los registros recuperados en una sentencia que une 2 condiciones con el operador "and", cumplen con las 2 condiciones.

Queremos ver los libros cuyo autor sea "Borges" y/o cuya editorial sea "Planeta":

```
select * from libros
 where autor='Borges' or
 editorial='Planeta';
```

En la sentencia anterior usamos el operador "or"; indicamos que recupere los libros en los cuales el valor del campo "autor" sea "Borges" y/o el valor del campo "editorial" sea "Planeta", es decir, seleccionará los registros que cumplan con la primera condición, con la segunda condición o con ambas condiciones.

Los registros recuperados con una sentencia que une 2 condiciones con el operador "or", cumplen 1 de las condiciones o ambas.

Queremos recuperar los libros que NO cumplan la condición dada, por ejemplo, aquellos cuya editorial NO sea "Planeta":

```
select * from libros
 where not editorial='Planeta';
```

El operador "not" invierte el resultado de la condición a la cual antecede.

Los registros recuperados en una sentencia en la cual aparece el operador "not", no cumplen con la condición a la cual afecta el "NOT".

Los paréntesis se usan para encerrar condiciones, para que se evalúen como una sola expresión.
Cuando explicitamos varias condiciones con diferentes operadores lógicos (combinamos "and", "or") permite establecer el orden de prioridad de la evaluación; además permite diferenciar las expresiones más claramente.

Por ejemplo, las siguientes expresiones devuelven un resultado diferente:

```
select* from libros
 where (autor='Borges') or
 (editorial='Paidos' and precio<20);

select * from libros
 where (autor='Borges' or editorial='Paidos') and
 (precio<20);
```

Si bien los paréntesis no son obligatorios en todos los casos, se recomienda utilizarlos para evitar confusiones.

El orden de prioridad de los operadores lógicos es el siguiente: "not" se aplica antes que "and" y "and" antes que "or", si no se especifica un orden de evaluación mediante el uso de

paréntesis.
El orden en el que se evalúan los operadores con igual nivel de precedencia es indefinido, por ello se recomienda usar los paréntesis.

Entonces, para establecer más de una condición en un "where" es necesario emplear operadores lógicos. "and" significa "y", indica que se cumplan ambas condiciones; "or" significa "y/o", indica que se cumpla una u otra condición (o ambas); "not" significa "no", indica que no se cumpla la condición especificada.

Servidor de SQL Server instalado en forma local.

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

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

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

go

insert into libros
  values('El aleph','Borges','Emece',15.90);
insert into libros
  values('Antología poética','Borges','Planeta',39.50);
insert into libros
  values('Java en 10 minutos','Mario Molina','Planeta',50.50);
insert into libros
  values('Alicia en el pais de las maravillas','Lewis
Carroll','Emece',19.90);
insert into libros
  values('Martin Fierro','Jose Hernandez','Emece',25.90);
insert into libros
  values('Martin Fierro','Jose Hernandez','Paidos',16.80);
insert into libros
  values('Aprenda PHP','Mario Molina','Emece',19.50);
insert into libros
  values('Cervantes y el quijote','Borges','Paidos',18.40);

-- Recuperamos los libros cuyo autor sea igual a "Borges" y
```

```
-- cuyo precio no supere los 20 pesos:
select * from libros
  where (autor='Borges') and
  (precio<=20);

-- Seleccionamos los libros cuyo autor es "Borges" y/o cuya editorial es
"Planeta":
select * from libros
  where autor='Borges' or
  editorial='Planeta';

-- Recuperamos los libros cuya editorial NO es "Planeta":
 select * from libros
  where not editorial='Planeta';

-- Veamos cómo el uso de paréntesis hace que SQL Server evalúe en forma
diferente
-- ciertas consultas aparentemente iguales:
 select * from libros
  where (autor='Borges') or
  (editorial='Paidos' and precio<20);

 select * from libros
  where (autor='Borges' or editorial='Paidos') and
  (precio<20);
```

- Otros operadores relacionales (is null)

Hemos aprendido los operadores relacionales "=" (igual), "<>" (distinto), ">" (mayor), "<" (menor), ">=" (mayor o igual) y "<=" (menor o igual). Dijimos que no eran los únicos.

Existen otro operador relacional "is null".

Se emplea el operador "is null" para recuperar los registros en los cuales esté almacenado el valor "null" en un campo específico:

```
 select * from libros
  where editorial is null;
```

Para obtener los registros que no contiene "null", se puede emplear "is not null", esto mostrará los registros con valores conocidos.

Siempre que sea posible, emplee condiciones de búsqueda positivas ("is null"), evite las negativas ("is not null") porque con ellas se evalúan todos los registros y esto hace más lenta la recuperación de los datos.

Servidor de SQL Server instalado en forma local.

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

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

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

go

insert into libros
  values('El aleph','Borges','Emece',15.90);
insert into libros
  values('Cervantes y el
quijote','Borges','Paidos',null);
insert into libros
  values('Alicia en el pais de las maravillas','Lewis
Carroll',null,19.90);
insert into libros
  values('Martin Fierro','Jose
Hernandez','Emece',25.90);
insert into libros (titulo,autor,precio)
  values('Antología poética','Borges',25.50);
insert into libros (titulo,autor,precio)
  values('Java en 10 minutos','Mario Molina',45.80);
insert into libros (titulo,autor)
  values('Martin Fierro','Jose Hernandez');
insert into libros (titulo,autor)
  values('Aprenda PHP','Mario Molina');

select * from libros
  where editorial is null;

select * from libros
  where editorial is not null;
```

31 - Otros operadores relacionales (between)

Hemos visto los operadores relacionales: = (igual), <> (distinto), > (mayor), < (menor), >= (mayor o igual), <= (menor o igual), is null/is not null (si un valor es NULL o no).

Otro operador relacional es "between", trabajan con intervalos de valores.

Hasta ahora, para recuperar de la tabla "libros" los libros con precio mayor o igual a 20 y menor o igual a 40, usamos 2 condiciones unidas por el operador lógico "and":

```
select * from libros
 where precio>=20 and
 precio<=40;
```

Podemos usar "between" y así simplificar la consulta:

```
select * from libros
 where precio between 20 and 40;
```

Averiguamos si el valor de un campo dado (precio) está entre los valores mínimo y máximo especificados (20 y 40 respectivamente).

"between" significa "entre". Trabaja con intervalo de valores.
Este operador se puede emplear con tipos de datos numéricos y money (en tales casos incluyen los valores mínimo y máximo) y tipos de datos fecha y hora (incluye sólo el valor mínimo).

No tiene en cuenta los valores "null".

Si agregamos el operador "not" antes de "between" el resultado se invierte, es decir, se recuperan los registros que están fuera del intervalo especificado. Por ejemplo, recuperamos los libros cuyo precio NO se encuentre entre 20 y 35, es decir, los menores a 15 y mayores a 25:

```
select *from libros
 where precio not between 20 and 35;
```

Siempre que sea posible, emplee condiciones de búsqueda positivas ("between"), evite las negativas ("not between") porque hace más lenta la recuperación de los datos.

Entonces, se puede usar el operador "between" para reducir las condiciones "where".

Servidor de SQL Server instalado en forma local.

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

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

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

go

insert into libros
  values('El aleph','Borges','Emece',15.90);
insert into libros
  values('Cervantes y el quijote','Borges','Paidos',null);
insert into libros
  values('Alicia en el pais de las maravillas','Lewis
Carroll',null,19.90);
insert into libros
  values('Martin Fierro','Jose Hernandez','Emece',25.90);
insert into libros (titulo,autor,precio)
  values('Antología poética','Borges',32);
insert into libros (titulo,autor,precio)
  values('Java en 10 minutos','Mario Molina',45.80);
insert into libros (titulo,autor,precio)
  values('Martin Fierro','Jose Hernandez',40);
insert into libros (titulo,autor,precio)
  values('Aprenda PHP','Mario Molina',56.50);

-- Recuperamos los registros cuyo precio esté entre 20 y 40 empleando
"between":
select * from libros
  where precio between 20 and 40;

-- Para seleccionar los libros cuyo precio NO esté entre un intervalo de
valores
-- antecedemos "not" al "between":
select * from libros
  where precio not between 20 and 35;
```

32 - Otros operadores relacionales (in)

Se utiliza "in" para averiguar si el valor de un campo está incluido en una lista de valores especificada.

En la siguiente sentencia usamos "in" para averiguar si el valor del campo autor está incluido en la lista de valores especificada (en este caso, 2 cadenas).

Hasta ahora, para recuperar los libros cuyo autor sea 'Paenza' o 'Borges' usábamos 2 condiciones:

```
select *from libros
 where autor='Borges' or autor='Paenza';
```

Podemos usar "in" y simplificar la consulta:

```
select * from libros
 where autor in('Borges','Paenza');
```

Para recuperar los libros cuyo autor no sea 'Paenza' ni 'Borges' usábamos:

```
select * from libros
 where autor<>'Borges' and
 autor<>'Paenza';
```

También podemos usar "in" anteponiendo "not":

```
select * from libros
 where autor not in ('Borges','Paenza');
```

Empleando "in" averiguamos si el valor del campo está incluido en la lista de valores especificada; con "not" antecediendo la condición, invertimos el resultado, es decir, recuperamos los valores que no se encuentran (coindicen) con la lista de valores.

Los valores "null" no se consideran.

Recuerde: siempre que sea posible, emplee condiciones de búsqueda positivas ("in"), evite las negativas ("not in") porque con ellas se evalún todos los registros y esto hace más lenta la recuperación de los datos.

Servidor de SQL Server instalado en forma local.

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

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

create table libros(
  codigo int identity,
  titulo varchar(40) not null,
  autor varchar(20),
  editorial varchar(20),
  precio decimal(6,2),
);

go

insert into libros
  values('El aleph','Borges','Emece',15.90);
insert into libros
  values('Cervantes y el
quijote','Borges','Paidos',null);
insert into libros
  values('Alicia en el pais de las maravillas','Lewis
Carroll',null,19.90);
insert into libros
  values('Matematica estas ahi','Paenza','Siglo
XXI',15);
insert into libros (titulo,precio)
  values('Antología poética',32);
insert into libros (titulo,autor,precio)
  values('Martin Fierro','Jose Hernandez',40);
insert into libros (titulo,autor,precio)
  values('Aprenda PHP','Mario Molina',56.50);

select * from libros
  where autor in('Borges','Paenza');

select * from libros
  where autor not in ('Borges','Paenza');
```

33 - Búsqueda de patrones (like - not like)

Existe un operador relacional que se usa para realizar comparaciones exclusivamente de cadenas, "like" y "not like".

Hemos realizado consultas utilizando operadores relacionales para comparar cadenas. Por ejemplo, sabemos recuperar los libros cuyo autor sea igual a la cadena "Borges":

```
select *from libros
 where autor='Borges';
```

El operador igual ("=") nos permite comparar cadenas de caracteres, pero al realizar la comparación, busca coincidencias de cadenas completas, realiza una búsqueda exacta.

Imaginemos que tenemos registrados estos 2 libros:

```
"El Aleph", "Borges";
"Antologia poetica", "J.L. Borges";
```

Si queremos recuperar todos los libros de "Borges" y especificamos la siguiente condición:

```
select * from libros
where autor='Borges';
```

sólo aparecerá el primer registro, ya que la cadena "Borges" no es igual a la cadena "J.L. Borges".

Esto sucede porque el operador "=" (igual), también el operador "<>" (distinto) comparan cadenas de caracteres completas. Para comparar porciones de cadenas utilizamos los operadores "like" y "not like".

Entonces, podemos comparar trozos de cadenas de caracteres para realizar consultas. Para recuperar todos los registros cuyo autor contenga la cadena "Borges" debemos tipear:

```
select * from libros
 where autor like "%Borges%";
```

El símbolo "%" (porcentaje) reemplaza cualquier cantidad de caracteres (incluyendo ningún caracter). Es un caracter comodín. "like" y "not like" son operadores de comparación que señalan igualdad o diferencia.

Para seleccionar todos los libros que comiencen con "M":

```
 select * from libros
  where titulo like 'M%';
```

Note que el símbolo "%" ya no está al comienzo, con esto indicamos que el título debe tener como primera letra la "M" y luego, cualquier cantidad de caracteres.

Para seleccionar todos los libros que NO comiencen con "M":

```
 select * from libros
  where titulo not like 'M%';
```

Así como "%" reemplaza cualquier cantidad de caracteres, el guión bajo "_" reemplaza un caracter, es otro caracter comodín. Por ejemplo, queremos ver los libros de "Lewis Carroll" pero no recordamos si se escribe "Carroll" o "Carrolt", entonces tipeamos esta condición:

```
 select * from libros
  where autor like "%Carrol_";
```

Otro caracter comodín es [] reemplaza cualquier carácter contenido en el conjunto especificado dentro de los corchetes.

Para seleccionar los libros cuya editorial comienza con las letras entre la "P" y la "S" usamos la siguiente sintaxis:

```
 select titulo,autor,editorial
  from libros
  where editorial like '[P-S]%';
```

Ejemplos:

```
... like '[a-cf-i]%': busca cadenas que comiencen con
a,b,c,f,g,h o i;
... like '[-acfi]%': busca cadenas que comiencen con -
,a,c,f o i;
... like 'A[_]9%': busca cadenas que comiencen con
'A_9';
... like 'A[nm]%': busca cadenas que comiencen con 'An'
o 'Am'.
```

El cuarto caracter comodín es [^] reemplaza cualquier caracter NO presente en el conjunto especificado dentro de los corchetes.

Para seleccionar los libros cuya editorial NO comienza con las letras "P" ni "N" tipeamos:

```
 select titulo,autor,editorial
  from libros
  where editorial like '[^PN]%';
```

"like" se emplea con tipos de datos char, nchar, varchar, nvarchar o datetime. Si empleamos "like" con tipos de datos que no son caracteres, SQL Server convierte (si es posible) el tipo de dato a caracter. Por ejemplo, queremos buscar todos los libros cuyo precio se encuentre entre 10.00 y 19.99:

```
 select titulo,precio from libros
  where precio like '1_.%';
```

Queremos los libros que NO incluyen centavos en sus precios:

```
 select titulo,precio from libros
  where precio like '%.00';
```

Para búsquedas de caracteres comodines como literales, debe incluirlo dentro de corchetes, por ejemplo, si busca:

```
... like '%[%]%': busca cadenas que contengan el signo
'%';
... like '%[_]%': busca cadenas que contengan el signo
'_';
... like '%[[]%': busca cadenas que contengan el signo
'[';
```

Servidor de SQL Server instalado en forma local.

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

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

create table libros(
  codigo int identity,
```

```
  titulo varchar(40) not null,
  autor varchar(20) default 'Desconocido',
  editorial varchar(20),
  precio decimal(6,2),
  primary key(codigo)
);

go

insert into libros
  values('El aleph','Borges','Emece',15.90);
insert into libros
  values('Antología poética','J. L.
Borges','Planeta',null);
insert into libros
  values('Alicia en el pais de las maravillas','Lewis
Carroll',null,19.90);
insert into libros
  values('Matematica estas ahi','Paenza','Siglo
XXI',15);
insert into libros
  values('Martin Fierro','Jose Hernandez',default,40);
insert into libros
  values('Aprenda PHP','Mario Molina','Nuevo
siglo',56.50);

-- Recuperamos todos los libros que contengan en el
campo "autor" la cadena "Borges":
select * from libros
  where autor like '%Borges%';

-- Seleccionamos los libros cuyos títulos comienzan con
la letra "M":
select * from libros
  where titulo like 'M%';

-- Seleccionamos todos los títulos que NO comienzan con
"M":
select * from libros
  where titulo not like 'M%';

-- Si queremos ver los libros de "Lewis Carroll" pero no
recordamos si se escribe
```

```
-- "Carroll" o "Carrolt", podemos emplear el comodín "_"
(guión bajo) y establecer
-- la siguiente condición:
select * from libros
  where autor like '%Carrol_';

-- Buscamos los libros cuya editorial comienza con las
letras entre la "P" y la "S":
select titulo,autor,editorial
  from libros
  where editorial like '[P-S]%';

-- Seleccionamos los libros cuya editorial NO comienza
con las letras "P" ni "N":
select titulo,autor,editorial
  from libros
  where editorial like '[^PN]%';

-- Recuperamos todos los libros cuyo precio se encuentra
entre 10.00 y 19.99:
select titulo,precio from libros
  where precio like '1_.%';

-- Recuperamos los libros que NO incluyen centavos en
sus precios:
select titulo,precio from libros
  where precio like '%.00';
```

34 - Contar registros (count)

Existen en SQL Server funciones que nos permiten contar registros, calcular sumas, promedios, obtener valores máximos y mínimos. Estas funciones se denominan funciones de agregado y operan sobre un conjunto de valores (registros), no con datos individuales y devuelven un único valor.

Imaginemos que nuestra tabla "libros" contiene muchos registros. Para averiguar la cantidad sin necesidad de contarlos manualmente usamos la función "count()":

```
 select count(*)
  from libros;
```

La función "count()" cuenta la cantidad de registros de una tabla, incluyendo los que tienen valor nulo.

También podemos utilizar esta función junto con la cláusula "where" para una consulta más específica. Queremos saber la cantidad de libros de la editorial "Planeta":

```
 select count(*)
  from libros
  where editorial='Planeta';
```

Para contar los registros que tienen precio (sin tener en cuenta los que tienen valor nulo), usamos la función "count()" y en los paréntesis colocamos el nombre del campo que necesitamos contar:

```
 select count(precio)
  from libros;
```

Note que "count(*)" retorna la cantidad de registros de una tabla (incluyendo los que tienen valor "null") mientras que "count(precio)" retorna la cantidad de registros en los cuales el campo "precio" no es nulo. No es lo mismo. "count(*)" cuenta registros, si en lugar de un asterisco colocamos como argumento el nombre de un campo, se contabilizan los registros cuyo valor en ese campo NO es nulo.

Servidor de SQL Server instalado en forma local.

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

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

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

go
```

```
insert into libros
  values('El aleph','Borges','Emece',15.90);
insert into libros
  values('Antología poética','J. L.
Borges','Planeta',null);
insert into libros
  values('Alicia en el pais de las maravillas','Lewis
Carroll',null,19.90);
insert into libros
  values('Matematica estas ahi','Paenza','Siglo
XXI',15);
insert into libros
  values('Martin Fierro','Jose Hernandez',default,40);
insert into libros
  values('Aprenda PHP','Mario Molina','Nuevo
siglo',null);
insert into libros
  values('Uno','Richard Bach','Planeta',20);

-- Averiguemos la cantidad de libros usando la función
"count()":
select count(*)
  from libros;

-- Contamos los libros de editorial "Planeta":
select count(*)
  from libros
  where editorial='Planeta';

-- Contamos los registros que tienen precio (sin tener
en cuenta
-- los que tienen valor nulo),
select count(precio)
  from libros;
```

35 - Contar registros (count_big)

Retorna la cantidad de registros. Es similar a la función "count (*)", la diferencia es que "count_big" retorna un valor "bigint" y "count", un "int".

"count_big (*)" cuenta la cantidad de registros de una tabla, incluidos los valores nulos y duplicados.

"count_big (CAMPO)" retorna la cantidad de registros cuyo valor en el campo especificado entre paréntesis no es nulo.

"count_big (distinto CAMPO)" retorna la cantidad de registros cuyo valor en el campo especificado no es nulo, sin considerar los repetidos.

Averiguemos la cantidad de libros usando la función "count_big ()":

```
 seleccione count_big (*)
  de libros;
```

Note que incluye todos los libros aunque tengan valor nulo en algún campo.

Contamos los libros de editorial "Planeta":

```
 seleccione count_big (*)
  de libros
  donde editorial = 'Planeta';
```

Contamos los registros que tienen precio (sin tener en cuenta los que tienen valor nulo):

```
 seleccione count_big (precio)
  de libros;
```

Contamos las editoriales (sin repetir):

```
 seleccione count_big (editorial distinta)
  de libros;
```

Servidor de SQL Server instalado en forma local.

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

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

crear libros de tabla
  codigo int identidad,
  titulo varchar (40) no nulo,
  autor varchar (20) predeterminado 'Desconocido',
  editorial varchar (20),
  precio decimal (6,2),
```

```
  clave primaria (codigo)
);

insertar en libros valores ('El aleph', 'Borges',
'Emece', 15,90);
insertar en libros valores ('Antología poética',
'Borges', 'Planeta', nulo);
insertar en libros valores ('Alicia en el país de las
maravillas', 'Lewis Carroll', nulo, 19,90);
insertar en libros valores ('Matematica estas ahi',
'Paenza', 'Siglo XXI', 15);
insertar en los libros valores ('Martín Fierro', 'José
Hernández', predeterminado, 40);
insertar en libros valores ('Aprenda PHP', 'Mario
Molina', 'Nuevo siglo', nulo);
insertar en libros valores ('Uno', 'Richard Bach',
'Planeta', 20);

seleccione count_big (*)
  de libros;

seleccione count_big (*)
  de libros
  donde editorial = 'Planeta';

seleccione count_big (precio)
  de libros;

- Contamos las editoriales (sin repetir):
seleccione count_big (editorial distinta)
  de libros;
```

36 - Funciones de agrupamiento (count - sum - min - max - avg)

Hemos visto que SQL Server tiene funciones que nos permiten contar registros, calcular sumas, promedios, obtener valores máximos y mínimos, las funciones de agregado.

Ya hemos aprendido una de ellas, "count()", veamos otras.

Se pueden usar en una instrucción "select" y combinarlas con la cláusula "group by".

Todas estas funciones retornan "null" si ningún registro cumple con la condición del "where", excepto "count" que en tal caso retorna cero.

El tipo de dato del campo determina las funciones que se pueden emplear con ellas.

Las relaciones entre las funciones de agrupamiento y los tipos de datos es la siguiente:

- count: se puede emplear con cualquier tipo de dato.

- min y max: con cualquier tipo de dato.

- sum y avg: sólo en campos de tipo numérico.

La función "sum()" retorna la suma de los valores que contiene el campo especificado. Si queremos saber la cantidad total de libros que tenemos disponibles para la venta, debemos sumar todos los valores del campo "cantidad":

```
select sum(cantidad)
 from libros;
```

Para averiguar el valor máximo o mínimo de un campo usamos las funciones "max()" y "min()" respectivamente.
Queremos saber cuál es el mayor precio de todos los libros:

```
select max(precio)
 from libros;
```

Entonces, dentro del paréntesis de la función colocamos el nombre del campo del cuál queremos el máximo valor.

La función "avg()" retorna el valor promedio de los valores del campo especificado. Queremos saber el promedio del precio de los libros referentes a "PHP":

```
select avg(precio)
 from libros
 where titulo like '%PHP%';
```

Ahora podemos entender porque estas funciones se denominan "funciones de agrupamiento", porque operan sobre conjuntos de registros, no con datos individuales.

Tratamiento de los valores nulos:

Si realiza una consulta con la función "count" de un campo que contiene 18 registros, 2 de los cuales contienen valor nulo, el resultado devuelve un total de 16 filas porque no considera aquellos con valor nulo.

Todas las funciones de agregado, excepto "count(*)", excluye los valores nulos de los campos. "count(*)" cuenta todos los registros, incluidos los que contienen "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;

create table libros(
  codigo int identity,
  titulo varchar(40) not null,
  autor varchar(30) default 'Desconocido',
  editorial varchar(15),
  precio decimal(5,2),
  cantidad tinyint,
  primary key(codigo)
);

go

insert into libros
  values('El aleph','Borges','Planeta',15,null);
insert into libros
  values('Martin Fierro','Jose
Hernandez','Emece',22.20,200);
insert into libros
  values('Antologia poetica','J.L.
Borges','Planeta',null,150);
insert into libros
  values('Aprenda PHP','Mario
Molina','Emece',18.20,null);
insert into libros
  values('Cervantes y el quijote','Bioy Casares- J.L.
Borges','Paidos',null,100);
```

```
insert into libros
  values('Manual de PHP', 'J.C. Paez', 'Siglo
XXI',31.80,120);
insert into libros
  values('Harry Potter y la piedra filosofal','J.K.
Rowling',default,45.00,90);
insert into libros
  values('Harry Potter y la camara secreta','J.K.
Rowling','Emece',46.00,100);
insert into libros (titulo,autor,cantidad)
  values('Alicia en el pais de las maravillas','Lewis
Carroll',220);
insert into libros (titulo,autor,cantidad)
  values('PHP de la A a la Z',default,0);

-- Cantidad total de libros, sumamos las cantidades de
cada uno:
 select sum(cantidad)
  from libros;

-- Para conocer cuántos libros tenemos de la editorial
"Emece":
 select sum(cantidad)
  from libros
  where editorial='Emece';

-- Queremos saber cuál es el libro más costoso:
 select max(precio)
  from libros;

-- Para conocer el precio mínimo de los libros de
"Rowling":
 select min(precio)
  from libros
  where autor like '%Rowling%';

 -- Queremos saber el promedio del precio de los libros
referentes a "PHP":
 select avg(precio)
  from libros
  where titulo like '%PHP%';
```

37 - Agrupar registros (group by)

Hemos aprendido que las funciones de agregado permiten realizar varios cálculos operando con conjuntos de registros.

Las funciones de agregado solas producen un valor de resumen para todos los registros de un campo. Podemos generar valores de resumen para un solo campo, combinando las funciones de agregado con la cláusula "group by", que agrupa registros para consultas detalladas.

Queremos saber la cantidad de libros de cada editorial, podemos tipear la siguiente sentencia:

```
select count(*) from libros
 where editorial='Planeta';
```

y repetirla con cada valor de "editorial":

```
select count(*) from libros
 where editorial='Emece';
select count(*) from libros
 where editorial='Paidos';
...
```

Pero hay otra manera, utilizando la cláusula "group by":

```
select editorial, count(*)
 from libros
 group by editorial;
```

La instrucción anterior solicita que muestre el nombre de la editorial y cuente la cantidad agrupando los registros por el campo "editorial". Como resultado aparecen los nombres de las editoriales y la cantidad de registros para cada valor del campo.

Los valores nulos se procesan como otro grupo.

Entonces, para saber la cantidad de libros que tenemos de cada editorial, utilizamos la función "count()", agregamos "group by" (que agrupa registros) y el campo por el que deseamos que se realice el agrupamiento, también colocamos el nombre del campo a recuperar; la sintaxis básica es la siguiente:

```
select CAMPO, FUNCIONDEAGREGADO
```

```
 from NOMBRETABLA
 group by CAMPO;
```

También se puede agrupar por más de un campo, en tal caso, luego del "group by" se listan los campos, separados por comas. Todos los campos que se especifican en la cláusula "group by" deben estar en la lista de selección.

```
select CAMPO1, CAMPO2, FUNCIONDEAGREGADO
 from NOMBRETABLA
 group by CAMPO1,CAMPO2;
```

Para obtener la cantidad libros con precio no nulo, de cada editorial utilizamos la función "count()" enviándole como argumento el campo "precio", agregamos "group by" y el campo por el que deseamos que se realice el agrupamiento (editorial):

```
select editorial, count(precio)
 from libros
 group by editorial;
```

Como resultado aparecen los nombres de las editoriales y la cantidad de registros de cada una, sin contar los que tienen precio nulo.

Recuerde la diferencia de los valores que retorna la función "count()" cuando enviamos como argumento un asterisco o el nombre de un campo: en el primer caso cuenta todos los registros incluyendo los que tienen valor nulo, en el segundo, los registros en los cuales el campo especificado es no nulo.

Para conocer el total en dinero de los libros agrupados por editorial:

```
select editorial, sum(precio)
 from libros
 group by editorial;
```

Para saber el máximo y mínimo valor de los libros agrupados por editorial:

```
select editorial,
 max(precio) as mayor,
 min(precio) as menor
 from libros
 group by editorial;
```

Para calcular el promedio del valor de los libros agrupados por editorial:

```
 select editorial, avg(precio)
  from libros
  group by editorial;
```

Es posible limitar la consulta con "where".

Si incluye una cláusula "where", sólo se agrupan los registros que cumplen las condiciones.

Vamos a contar y agrupar por editorial considerando solamente los libros cuyo precio sea menor a 30 pesos:

```
 select editorial, count(*)
  from libros
  where precio<30
  group by editorial;
```

Note que las editoriales que no tienen libros que cumplan la condición, no aparecen en la salida. Para que aparezcan todos los valores de editorial, incluso los que devuelven cero o "null" en la columna de agregado, debemos emplear la palabra clave "all" al lado de "group by":

```
 select editorial, count(*)
  from libros
  where precio<30
  group by all editorial;
```

Entonces, usamos "group by" para organizar registros en grupos y obtener un resumen de dichos grupos. SQL Server produce una columna de valores por cada grupo, devolviendo filas por cada grupo especificado.

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),
  precio decimal(5,2),
  cantidad tinyint,
  primary key(codigo)
);

insert into libros
  values('El aleph','Borges','Planeta',15,null);
insert into libros
  values('Martin Fierro','Jose
Hernandez','Emece',22.20,200);
insert into libros
  values('Antologia poetica','J.L.
Borges','Planeta',null,150);
insert into libros
  values('Aprenda PHP','Mario
Molina','Emece',18.20,null);
insert into libros
  values('Cervantes y el quijote','Bioy Casares- J.L.
Borges','Paidos',null,100);
insert into libros
  values('Manual de PHP', 'J.C. Paez', 'Siglo
XXI',31.80,120);
insert into libros
  values('Harry Potter y la piedra filosofal','J.K.
Rowling',default,45.00,90);
insert into libros
  values('Harry Potter y la camara secreta','J.K.
Rowling','Emece',null,100);
insert into libros
  values('Alicia en el pais de las maravillas','Lewis
Carroll','Paidos',22.50,200);
insert into libros
  values('PHP de la A a la Z',null,null,null,0);

-- Cantidad de libros de cada editorial:
select editorial, count(*)
  from libros
  group by editorial;

-- Cantidad libros con precio no nulo de cada editorial:
select editorial, count(precio)
  from libros
```

```
  group by editorial;

-- Total en dinero de los libros agrupados por
editorial:
select editorial, sum(precio)
  from libros
  group by editorial;

-- Máximo y mínimo valor de los libros agrupados por
editorial:
select editorial,
  max(precio) as mayor,
  min(precio) as menor
  from libros
  group by editorial;

-- Promedio del valor de los libros agrupados por
editorial:
select editorial, avg(precio)
  from libros
  group by editorial;

-- Contar y agrupar por editorial considerando solamente
los libros cuyo precio es menor a 30 pesos:
select editorial, count(*)
  from libros
  where precio<30
  group by editorial;

-- Todos los valores de editorial, incluso los que
devuelven cero o "null" en la columna de agregado,
--  debemos emplear la palabra clave "all" al lado de
"group by":
select editorial, count(*)
  from libros
  where precio<30
  group by all editorial;
```

38 - Seleccionar grupos (having)

Así como la cláusula "where" permite seleccionar (o rechazar) registros individuales; la cláusula "having" permite seleccionar (o rechazar) un grupo de registros.

Si queremos saber la cantidad de libros agrupados por editorial usamos la siguiente instrucción ya aprendida:

```
select editorial, count(*)
 from libros
 group by editorial;
```

Si queremos saber la cantidad de libros agrupados por editorial pero considerando sólo algunos grupos, por ejemplo, los que devuelvan un valor mayor a 2, usamos la siguiente instrucción:

```
select editorial, count(*) from libros
 group by editorial
 having count(*)>2;
```

Se utiliza "having", seguido de la condición de búsqueda, para seleccionar ciertas filas retornadas por la cláusula "group by".

Veamos otros ejemplos. Queremos el promedio de los precios de los libros agrupados por editorial, pero solamente de aquellos grupos cuyo promedio supere los 25 pesos:

```
select editorial, avg(precio) from libros
 group by editorial
 having avg(precio)>25;
```

En algunos casos es posible confundir las cláusulas "where" y "having". Queremos contar los registros agrupados por editorial sin tener en cuenta a la editorial "Planeta".
Analicemos las siguientes sentencias:

```
select editorial, count(*) from libros
 where editorial<>'Planeta'
 group by editorial;
select editorial, count(*) from libros
 group by editorial
 having editorial<>'Planeta';
```

Ambas devuelven el mismo resultado, pero son diferentes. La primera, selecciona todos los registros rechazando los de editorial "Planeta" y luego los agrupa para contarlos. La segunda, selecciona todos los registros, los agrupa para contarlos y finalmente rechaza fila con la cuenta correspondiente a la editorial "Planeta".

No debemos confundir la cláusula "where" con la cláusula "having"; la primera establece condiciones para la selección de registros de un "select"; la segunda establece condiciones para la selección de registros de una salida "group by".

Veamos otros ejemplos combinando "where" y "having". Queremos la cantidad de libros, sin considerar los que tienen precio nulo, agrupados por editorial, sin considerar la editorial "Planeta":

```
select editorial, count(*) from libros
 where precio is not null
 group by editorial
 having editorial<>'Planeta';
```

Aquí, selecciona los registros rechazando los que no cumplan con la condición dada en "where", luego los agrupa por "editorial" y finalmente rechaza los grupos que no cumplan con la condición dada en el "having".

Se emplea la cláusula "having" con funciones de agrupamiento, esto no puede hacerlo la cláusula "where". Por ejemplo queremos el promedio de los precios agrupados por editorial, de aquellas editoriales que tienen más de 2 libros:

```
select editorial, avg(precio) from libros
 group by editorial
 having count(*) > 2;
```

En una cláusula "having" puede haber hasta 128 condiciones. Cuando utilice varias condiciones, tiene que combinarlas con operadores lógicos (and, or, not).

Podemos encontrar el mayor valor de los libros agrupados y ordenados por editorial y seleccionar las filas que tengan un valor menor a 100 y mayor a 30:

```
select editorial, max(precio) as 'mayor'
 from libros
 group by editorial
 having min(precio)<100 and
 min(precio)>30
 order by editorial;
```

Entonces, usamos la claúsula "having" para restringir las filas que devuelve una salida "group by". Va siempre después de la cláusula "group by" y antes de la cláusula "order by" si la hubiere.

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),
  precio decimal(5,2),
  cantidad tinyint,
  primary key(codigo)
);

go

insert into libros
  values('El aleph','Borges','Planeta',35,null);
insert into libros
  values('Martin Fierro','Jose
Hernandez','Emece',22.20,200);
insert into libros
  values('Martin Fierro','Jose
Hernandez','Planeta',40,200);
insert into libros
  values('Antologia poetica','J.L.
Borges','Planeta',null,150);
insert into libros
  values('Aprenda PHP','Mario Molina','Emece',18,null);
insert into libros
  values('Manual de PHP', 'J.C. Paez', 'Siglo
XXI',56,120);
insert into libros
  values('Cervantes y el quijote','Bioy Casares- J.L.
Borges','Paidos',null,100);
insert into libros
```

```
  values('Harry Potter y la piedra filosofal','J.K.
Rowling',default,45.00,90);
insert into libros
  values('Harry Potter y la camara secreta','J.K.
Rowling','Emece',null,100);
insert into libros
  values('Alicia en el pais de las maravillas','Lewis
Carroll','Paidos',42,80);
insert into libros
  values('PHP de la A a la Z',null,null,110,0);
insert into libros
  values('Uno','Richard Bach','Planeta',25,null);

--  Cantidad de libros agrupados por editorial pero
considerando sólo algunos grupos,
--  los que devuelvan un valor mayor a 2
select editorial, count(*) from libros
  group by editorial
  having count(*)>2;

-- Promedio de los precios de los libros agrupados por
editorial, pero solamente de
-- aquellos grupos cuyo promedio supere los 25 pesos:
 select editorial, avg(precio) from libros
  group by editorial
  having avg(precio)>25;

-- Cantidad de libros, sin considerar los que tienen
precio nulo (where), agrupados por
-- editorial (group by), sin considerar la editorial
"Planeta" (having):
 select editorial, count(*) from libros
  where precio is not null
  group by editorial
  having editorial<>'Planeta';

-- Promedio de los precios agrupados por editorial,
-- de aquellas editoriales que tienen más de 2 libros:
 select editorial, avg(precio) from libros
  group by editorial
  having count(*) > 2;

-- Mayor valor de los libros agrupados y ordenados por
editorial y seleccionamos las
```

```
-- filas que tienen un valor menor a 100 y mayor a 30:
 select editorial, max(precio) as 'mayor'
  from libros
  group by editorial
  having max(precio)<100 and
  max(precio)>30
  order by editorial;
```

39 - Modificador del group by (with rollup)

Podemos combinar "group by" con los operadores "rollup" y "cube" para generar valores de resumen a la salida.

El operador "rollup" resume valores de grupos. representan los valores de resumen de la precedente.

Tenemos la tabla "visitantes" con los siguientes campos: nombre, edad, sexo, domicilio, ciudad, telefono, montocompra.

Si necesitamos la cantidad de visitantes por ciudad empleamos la siguiente sentencia:

```
 select ciudad,count(*) as cantidad
  from visitantes
  group by ciudad;
```

Esta consulta muestra el total de visitantes agrupados por ciudad; pero si queremos además la cantidad total de visitantes, debemos realizar otra consulta:

```
  select count(*) as total
   from visitantes;
```

Para obtener ambos resultados en una sola consulta podemos usar "with rollup" que nos devolverá ambas salidas en una sola consulta:

```
 select ciudad,count(*) as cantidad
  from visitantes
  group by ciudad with rollup;
```

La consulta anterior retorna los registros agrupados por ciudad y una fila extra en la que la primera columna contiene "null" y la columna con la cantidad muestra la cantidad total.

La cláusula "group by" permite agregar el modificador "with rollup", el cual agrega registros extras al resultado de una consulta, que muestran operaciones de resumen.

Si agrupamos por 2 campos, "ciudad" y "sexo":

```
select ciudad,sexo,count(*) as cantidad
 from visitantes
 group by ciudad,sexo
 with rollup;
```

La salida muestra los totales por ciudad y sexo y produce tantas filas extras como valores existen del primer campo por el que se agrupa ("ciudad" en este caso), mostrando los totales para cada valor, con la columna correspondiente al segundo campo por el que se agrupa ("sexo" en este ejemplo) conteniendo "null", y 1 fila extra mostrando el total de todos los visitantes (con las columnas correspondientes a ambos campos conteniendo "null"). Es decir, por cada agrupación, aparece una fila extra con el/ los campos que no se consideran, seteados a "null".

Con "rollup" se puede agrupar hasta por 10 campos.

Es posible incluir varias funciones de agrupamiento, por ejemplo, queremos la cantidad de visitantes y la suma de sus compras agrupados por ciudad y sexo:

```
select ciudad,sexo,
 count(*) as cantidad,
 sum(montocompra) as total
 from visitantes
 group by ciudad,sexo
 with rollup;
```

Entonces, "rollup" es un modificador para "group by" que agrega filas extras mostrando resultados de resumen de los subgrupos. Si se agrupa por 2 campos SQL Server genera tantas filas extras como valores existen del primer campo (con el segundo campo seteado a "null") y una fila extra con ambos campos conteniendo "null".

Con "rollup" se puede emplear "where" y "having", pero no es compatible con "all".

Servidor de SQL Server instalado en forma local.

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

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

create table visitantes(
  nombre varchar(30),
  edad tinyint,
  sexo char(1),
  domicilio varchar(30),
  ciudad varchar(20),
  telefono varchar(11),
  montocompra decimal(6,2) not null
);

go

insert into visitantes
  values ('Susana
Molina',28,'f',null,'Cordoba',null,45.50);
insert into visitantes
  values ('Marcela Mercado',36,'f','Avellaneda
345','Cordoba','4545454',22.40);
insert into visitantes
  values ('Alberto Garcia',35,'m','Gral. Paz 123','Alta
Gracia','03547123456',25);
insert into visitantes
  values ('Teresa Garcia',33,'f',default,'Alta
Gracia','03547123456',120);
insert into visitantes
  values ('Roberto Perez',45,'m','Urquiza
335','Cordoba','4123456',33.20);
insert into visitantes
  values ('Marina Torres',22,'f','Colon 222','Villa
Dolores','03544112233',95);
insert into visitantes
  values ('Julieta Gomez',24,'f','San Martin 333','Alta
Gracia',null,53.50);
insert into visitantes
  values ('Roxana Lopez',20,'f','null','Alta
Gracia',null,240);
insert into visitantes
```

```
  values ('Liliana Garcia',50,'f','Paso
999','Cordoba','4588778',48);
insert into visitantes
  values ('Juan Torres',43,'m','Sarmiento
876','Cordoba',null,15.30);

-- Cantidad de visitantes por ciudad y el total de
visitantes
select ciudad,
  count(*) as cantidad
  from visitantes
  group by ciudad with rollup;

-- Filas de resumen cuando agrupamos por 2 campos,
"ciudad" y "sexo":
 select ciudad,sexo,
  count(*) as cantidad
  from visitantes
  group by ciudad,sexo
  with rollup;

-- Para conocer la cantidad de visitantes y la suma de
sus compras agrupados
-- por ciudad y sexo,
 select ciudad,sexo,
  count(*) as cantidad,
  sum(montocompra) as total
  from visitantes
  group by ciudad,sexo
  with rollup;
```

40 - Modificador del group by (with cube)

Hemos aprendido el modificador "rollup", que agrega filas extras mostrando resultados de resumen por cada grupo y subgrupo.

Por ejemplo, tenemos una tabla llamada "empleados" que contiene, entre otros, los campos "sexo", "estadocivil" y "seccion".

Si se agrupa por esos tres campos (en ese orden) y se emplea "rollup":

```
 select sexo,estadocivil,seccion,
```

```
  count(*) from empleados
  group by sexo,estadocivil,seccion
  with rollup;
```

SQL Server genera varias filas extras con información de resumen para los siguientes subgrupos:

```
- sexo y estadocivil (seccion seteado a "null"),
- sexo (estadocivil y seccion seteados a "null") y
- total (todos los campos seteados a "null").
```

Si se emplea "cube":

```
 select sexo,estadocivil,seccion,
  count(*) from empleados
  group by sexo,estadocivil,seccion
  with cube;
```

retorna más filas extras además de las anteriores:

```
- sexo y seccion (estadocivil seteado a "null"),
- estadocivil y seccion (sexo seteado a "null"),
- seccion (sexo y estadocivil seteados a "null") y
- estadocivil (sexo y seccion seteados a "null"),
```

Es decir, "cube" genera filas de resumen de subgrupos para todas las combinaciones posibles de los valores de los campos por los que agrupamos.

Se pueden colocar hasta 10 campos en el "group by".

Con "cube" se puede emplear "where" y "having", pero no es compatible con "all".

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),
  sexo char(1),
  estadocivil char(1),--c=casado, s=soltero,v=viudo
  seccion varchar(20),
  primary key (documento)
);

go

insert into empleados
  values ('22222222','Alberto
Lopez','m','c','Sistemas');
insert into empleados
  values ('23333333','Beatriz
Garcia','f','c','Administracion');
insert into empleados
  values ('24444444','Carlos
Fuentes','m','s','Administracion');
insert into empleados
  values ('25555555','Daniel
Garcia','m','s','Sistemas');
insert into empleados
  values ('26666666','Ester Juarez','f','c','Sistemas');
insert into empleados
  values ('27777777','Fabian
Torres','m','s','Sistemas');
insert into empleados
  values ('28888888','Gabriela
Lopez','f','c','Sistemas');
insert into empleados
  values ('29999999','Hector
Garcia','m','c','Administracion');
insert into empleados
  values ('30000000','Ines
Torres','f','c','Administracion');
insert into empleados
  values ('11111111','Juan
Garcia','m','v','Administracion');
insert into empleados
  values ('12222222','Luisa
Perez','f','v','Administracion');
insert into empleados
  values ('31111111','Marcela
Garcia','f','s','Administracion');
```

```
insert into empleados
  values ('32222222','Nestor
Fuentes','m','c','Sistemas');
insert into empleados
  values ('33333333','Oscar Garcia','m','s','Sistemas');
insert into empleados
  values ('34444444','Patricia
Juarez','f','c','Administracion');
insert into empleados
  values ('35555555','Roberto
Torres','m','c','Sistemas');
insert into empleados
  values ('36666666','Susana
Torres','f','c','Administracion');

select sexo,estadocivil,seccion,
  count(*) from empleados
  group by sexo,estadocivil,seccion
  with rollup;

select sexo,estadocivil,seccion,
  count(*) from empleados
  group by sexo,estadocivil,seccion
  with cube;
```

41 - Función grouping

La función "grouping" se emplea con los operadores "rollup" y "cube" para distinguir los valores de detalle y de resumen en el resultado. Es decir, permite diferenciar si los valores "null" que aparecen en el resultado son valores nulos de las tablas o si son una fila generada por los operadores "rollup" o "cube".

Con esta función aparece una nueva columna en la salida, una por cada "grouping"; retorna el valor 1 para indicar que la fila representa los valores de resumen de "rollup" o "cube" y el valor 0 para representar los valores de campo.

Sólo se puede emplear la función "grouping" en los campos que aparecen en la cláusula "group by".

Si tenemos una tabla "visitantes" con los siguientes registros almacenados:

```
Nombre            sexo  ciudad
------------------------------
Susana Molina     f     Cordoba
Marcela Mercado   f     Cordoba
Roberto Perez     f     null
Alberto Garcia    m     Cordoba
Teresa Garcia     f     Alta Gracia
```

y contamos la cantidad agrupando por ciudad (note que hay un valor nulo en dicho campo) empleando "rollup":

```
 select ciudad,
  count(*) as cantidad
  from visitantes
  group by ciudad
  with rollup;
```

aparece la siguiente salida:

```
ciudad            cantidad
-------------------------
NULL        1
Alta Gracia 1
Cordoba           3
NULL        5
```

La última fila es la de resumen generada por "rollup", pero no es posible distinguirla de la primera fila, en la cual "null" es un valor del campo. Para diferenciarla empleamos "grouping":

```
 select ciudad,
  count(*) as cantidad,
  grouping(ciudad) as resumen
  from visitantes
  group by ciudad
  with rollup;
```

aparece la siguiente salida:

```
ciudad            cantidad    resumen
---------------------------------------
NULL        1           0
Alta Gracia 1           0
Cordoba           3           0
```

```
NULL        5           1
```

La última fila contiene en la columna generada por "grouping" el valor 1, indicando que es la fila de resumen generada por "rollup"; la primera fila, contiene en dicha columna el valor 0, que indica que el valor "null" es un valor del campo "ciudad".

Entonces, si emplea los operadores "rollup" y "cube" y los campos por los cuales agrupa admiten valores nulos, utilice la función "grouping" para distinguir los valores de detalle y de resumen en el resultado.

Servidor de SQL Server instalado en forma local.

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

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

create table visitantes(
  nombre varchar(30),
  sexo char(1),
  ciudad varchar(20)
);

go

insert into visitantes values('Susana Molina', 'f',
'Cordoba');
insert into visitantes values('Marcela Mercado',
'f','Cordoba');
insert into visitantes values('Roberto Perez','f',null);
insert into visitantes values('Alberto
Garcia','m','Cordoba');
insert into visitantes values('Teresa Garcia','f','Alta
Gracia');

-- Contamos la cantidad de visitantes agrupando por
ciudad y empleando "rollup":
select ciudad,
  count(*) as cantidad
  from visitantes
  group by ciudad
  with rollup;
```

```
-- Contamos la cantidad de visitantes agrupando por
ciudad y empleando "rollup"
-- empleando grouping:
select ciudad,
  count(*) as cantidad,
  grouping(ciudad) as resumen
  from visitantes
  group by ciudad
  with rollup;
```

42 - Cláusulas compute y compute by

La cláusula compute y compute by están discontinuadas en las versiones de SQL Server 2012 y sucesivas.
No utilice esta característica en nuevos proyectos y modifique lo antes posible las aplicaciones que actualmente la utilizan. En su lugar, utilice rollup.

Las cláusulas "compute" y "compute by" generan totales que aparecen en columnas extras al final del resultado.

Produce filas de detalle y un valor único para una columna.

Se usa con las funciones de agrupamiento: avg(), count(), max(), min(), sum().

La sintaxis básica y general es la siguiente:

```
select CAMPOS
 from TABLA
 compute FUNCION(CAMPO);
```

El campo que se coloque en la cláusula "compute" debe estar incluida en la lista de campos del "select".

Para ver todos los datos de los visitantes y el promedio del monto de compra de nuestra tabla "visitantes":

```
select *from visitantes
compute avg(montocompra);
```

Produce la misma salida que las siguientes 2 sentencias:

```
select *from visitantes;
select avg(montocompra) from visitantes;
```

En una misma instrucción se pueden colocar varias cláusulas "compute":

```
select edad,ciudad,montocompra
 from visitantes
 compute avg(edad),sum(montocompra);
```

"Compute by" genera cortes de control y subtotales. Se generan filas de detalle y varios valores de resumen cuando cambian los valores del campo.

Con "compute by" se DEBE usar también la cláusula "order by" y los campos que se incluyan luego de "by" deben estar en el "order by". Listando varios campos luego del "by" corta un grupo en subgrupos y aplica la función de agregado en cada nivel de agrupamiento:

```
select nombre,ciudad,provincia
 from visitantes
 order by provincia
 compute count(provincia)
 by provincia;

select nombre,ciudad,provincia
 from visitantes
 order by provincia,ciudad
 compute count(provincia)
 by provincia,ciudad;
```

Los campos que aparecen luego de la cláusula "compute by" DEBEN ser idénticos a un subconjunto de los campos que aparecen después de "order by" y estar en el mismo orden. Si la cláusula "order by" tiene los siguientes campos:

```
... order by a,b,c...
```

la cláusula "compute by" puede incluir los siguientes subconjuntos de campos:

```
... compute ...
    by a...
o
... compute ...
    by a,b...
o
... compute ...
    by a,b,c...
```

En una misma instrucción se pueden colocar varias cláusulas "compute" combinadas con varias cláusulas "compute by":

```
 select *from visitantes
  order by provincia,ciudad
  compute avg(edad), sum(montocompra)
  compute avg(montocompra),count(provincia)
  by provincia,ciudad;
```

El resultado de la consulta anterior muestra el promedio de la compra y la cantidad al final de cada subgrupo de provincia y ciudad (compute by) y el promedio de las edades y el total del monto de compras de todos (compute).

Los tipos de datos ntext, text e image no se pueden incluir en una cláusula "compute" o "compute by".

43 - Registros duplicados (distinct)

Con la cláusula "distinct" se especifica que los registros con ciertos datos duplicados sean obviadas en el resultado. Por ejemplo, queremos conocer todos los autores de los cuales tenemos libros, si utilizamos esta sentencia:

```
 select autor from libros;
```

Aparecen repetidos. Para obtener la lista de autores sin repetición usamos:

```
 select distinct autor from libros;
```

También podemos tipear:

```
select autor from libros
 group by autor;
```

Note que en los tres casos anteriores aparece "null" como un valor para "autor"· Si sólo queremos la lista de autores conocidos, es decir, no queremos incluir "null" en la lista, podemos utilizar la sentencia siguiente:

```
select distinct autor from libros
 where autor is not null;
```

Para contar los distintos autores, sin considerar el valor "null" usamos:

```
select count(distinct autor)
 from libros;
```

Note que si contamos los autores sin "distinct", no incluirá los valores "null" pero si los repetidos:

```
select count(autor)
 from libros;
```

Esta sentencia cuenta los registros que tienen autor.

Podemos combinarla con "where". Por ejemplo, queremos conocer los distintos autores de la editorial "Planeta":

```
select distinct autor from libros
 where editorial='Planeta';
```

También puede utilizarse con "group by" para contar los diferentes autores por editorial:

```
select editorial, count(distinct autor)
 from libros
 group by editorial;
```

La cláusula "distinct" afecta a todos los campos presentados. Para mostrar los títulos y editoriales de los libros sin repetir títulos ni editoriales, usamos:

```
select distinct titulo,editorial
 from libros
 order by titulo;
```

Note que los registros no están duplicados, aparecen títulos iguales pero con editorial diferente, cada registro es diferente.

Entonces, "distinct" elimina registros duplicados.

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),
  primary key(codigo)
);

go

insert into libros
  values('El aleph','Borges','Planeta');
insert into libros
  values('Martin Fierro','Jose Hernandez','Emece');
insert into libros
  values('Martin Fierro','Jose Hernandez','Planeta');
insert into libros
  values('Antologia poetica','Borges','Planeta');
insert into libros
  values('Aprenda PHP','Mario Molina','Emece');
insert into libros
  values('Aprenda PHP','Lopez','Emece');
insert into libros
  values('Manual de PHP', 'J. Paez', null);
insert into libros
  values('Cervantes y el quijote',null,'Paidos');
insert into libros
  values('Harry Potter y la piedra filosofal','J.K.
Rowling','Emece');
insert into libros
```

```
  values('Harry Potter y la camara secreta','J.K.
Rowling','Emece');
insert into libros
  values('Alicia en el pais de las maravillas','Lewis
Carroll','Paidos');
insert into libros
  values('Alicia en el pais de las maravillas','Lewis
Carroll','Planeta');
insert into libros
  values('PHP de la A a la Z',null,null);
insert into libros
  values('Uno','Richard Bach','Planeta');

-- Para obtener la lista de autores sin repetición
select distinct autor from libros;

-- Para obtener la lista de autores conocidos, es decir,
no incluyendo "null"
-- en la lista:
select distinct autor from libros
  where autor is not null;

-- Contamos los distintos autores
select count(distinct autor)
  from libros;

-- Nombres de las editoriales sin repetir:
select distinct editorial from libros;

--  Cantidad de editoriales distintas:
select count(distinct editorial) from libros;

-- Distintos autores de la editorial "Planeta":
select distinct autor from libros
  where editorial='Planeta';

-- Distintos autores que tiene cada editorial empleando
"group by":
select editorial,count(distinct autor)
  from libros
  group by editorial;

-- Mostramos los títulos y editoriales de los libros sin
repetir
```

```
-- títulos ni editoriales:
select distinct titulo,editorial
  from libros
  order by titulo;
```

44 - Cláusula top

La palabra clave "top" se emplea para obtener sólo una cantidad limitada de registros, los primeros n registros de una consulta.

Con la siguiente consulta obtenemos todos los datos de los primeros 2 libros de la tabla:

```
 select top 2 * from libros;
```

Es decir, luego del "select" se coloca "top" seguido de un número entero positivo y luego se continúa con la consulta.

Se puede combinar con "order by":

```
 select top 3 titulo,autor
  from libros
  order by autor;
```

En la consulta anterior solicitamos los títulos y autores de los 3 primeros libros, ordenados por autor.

Cuando se combina con "order by" es posible emplear también la cláusula "with ties". Esta cláusula permite incluir en la selección, todos los registros que tengan el mismo valor del campo por el que se ordena en la última posición. Es decir, si el valor del campo por el cual se ordena el último registro retornado (el número n) está repetido en los siguientes registros (es decir, el n+1 tiene el mismo valor que n, y el n+2, etc.), lo incluye en la selección.

Veamos un ejemplo:

```
 select top 3 with ties
  * from libros
  order by autor;
```

Esta consulta solicita el retorno de los primeros 3 registros; en caso que el registro número 4 (y los posteriores), tengan el mismo valor en "autor" que el último registro retornado (número 3), también aparecerán en la selección.

Otro argumento posible cuando utilizamos la cláusula 'top' es 'percent' indicando el porcentaje de registros a recuperar y no la cantidad, por ejemplo:

```
 select top 50 percent
  * from libros
  order by autor;
```

Se recuperan la mitad de registros de la tabla 'libros'.

Los valores fraccionarios se redondean al siguiente valor entero.

Si colocamos un valor para "top" que supera la cantidad de registros de la tabla, SQL Server muestra todos los registros.

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(20),
  editorial varchar(20)
);

go

insert into libros values ('Uno','Richard
Bach','Planeta');
insert into libros values ('El aleph','Borges','Emece');
insert into libros values ('Alicia en el
pais...','Carroll','Planeta');
insert into libros values ('Aprenda PHP','Mario
Molina','Siglo XXI');
```

```
insert into libros values ('Java en 10 minutos','Mario
Molina','Siglo XXI');
insert into libros values ('Java desde cero','Mario
Molina','Emece');
insert into libros values ('Ilusiones','Richard
Bach','Planeta');

-- Obtenemos todos los datos de los primeros 2 libros de
la tabla:
select top 2 * from libros;

-- Mostramos los títulos y autores de los 3 primeros
libros ordenados por autor:
select top 3 titulo,autor
  from libros
  order by autor;

-- Realizamos la misma consulta anterior pero empleamos
la cláusula "with ties",
-- con lo cual incluiremos en la selección, todos los
registros que tengan el
-- mismo autor que el último registro retornado, aunque
pasemos de 3:
select top 3 with ties titulo,autor
  from libros
  order by autor;

-- El 50% de los registros de la tabla libros:
select top 50 percent
  * from libros
  order by autor;
```

45 - Clave primaria compuesta

Las claves primarias pueden ser simples, formadas por un solo campo o compuestas, más de un campo.

Recordemos que una clave primaria identifica 1 solo registro en una tabla.

Para un valor del campo clave existe solamente 1 registro. Los valores no se repiten ni pueden ser nulos.

Existe una playa de estacionamiento que almacena cada día los datos de los vehículos que ingresan en la tabla llamada "vehiculos" con los siguientes campos:

```
- patente char(6) not null,
- tipo char (1), 'a'= auto, 'm'=moto,
- horallegada datetime,
- horasalida datetime,
```

Necesitamos definir una clave primaria para una tabla con los datos descriptos arriba. No podemos usar solamente la patente porque un mismo auto puede ingresar más de una vez en el día a la playa; tampoco podemos usar la hora de entrada porque varios autos pueden ingresar a una misma hora.
Tampoco sirven los otros campos.

Como ningún campo, por si sólo cumple con la condición para ser clave, es decir, debe identificar un solo registro, el valor no puede repetirse, debemos usar 2 campos.

Definimos una clave compuesta cuando ningún campo por si solo cumple con la condición para ser clave.

En este ejemplo, un auto puede ingresar varias veces en un día a la playa, pero siempre será a distinta hora.

Usamos 2 campos como clave, la patente junto con la hora de llegada, así identificamos unívocamente cada registro.

Para establecer más de un campo como clave primaria usamos la siguiente sintaxis:

```
create table vehiculos(
 patente char(6) not null,
 tipo char(1),--'a'=auto, 'm'=moto
 horallegada datetime,
 horasalida datetime,
 primary key(patente,horallegada)
);
```

Nombramos los campos que formarán parte de la clave separados por comas.

Al ingresar los registros, SQL Server controla que los valores para los campos establecidos como clave primaria no estén repetidos en la tabla; si estuviesen repetidos, muestra un mensaje y la inserción no se realiza. Lo mismo sucede si realizamos una actualización.

Entonces, si un solo campo no identifica unívocamente un registro podemos definir una clave primaria compuesta, es decir formada por más de un campo.

Servidor de SQL Server instalado en forma local.

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

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

create table vehiculos(
  patente char(6) not null,
  tipo char(1),--'a'=auto, 'm'=moto
  horallegada datetime,
  horasalida datetime,
  primary key(patente,horallegada)
);

go

insert into vehiculos
values('AIC124','a','8:05','12:30');
insert into vehiculos values('CAA258','a','8:05',null);
insert into vehiculos
values('DSE367','m','8:30','18:00');
insert into vehiculos values('FGT458','a','9:00',null);
insert into vehiculos values('AIC124','a','16:00',null);
insert into vehiculos
values('LOI587','m','18:05','19:55');

-- Si intentamos ingresar un registro con clave primaria
repetida:
insert into vehiculos values('LOI587','m','18:05',null);

-- Si ingresamos un registro repitiendo el valor de uno
de los campos
-- que forman parte de la clave, si lo acepta:
```

```
insert into vehiculos values('LOI587','m','21:30',null);

-- Si intentamos actualizar un registro repitiendo la
clave primaria
-- (se genera un error):
update vehiculos set horallegada='8:05'
  where patente='AIC124' and horallegada='16:00';

-- Recordemos que los campos que forman parte de la
clave primaria no
-- aceptan valores nulos, aunque no se haya aclarado en
la definición
-- de la tabla (genera un error):
insert into vehiculos values('HUO690','m',null,null);

-- mostramos la estructura de la tabla
-- (los campos que forman parte de la clave primaria
(patente y horallegada)
--  tienen "NO" en la columna "IS_NULLABLE", es decir,
no admiten valores nulos.):
exec sp_columns vehiculos;
```

46 - Integridad de los datos

Es importante, al diseñar una base de datos y las tablas que contiene, tener en cuenta la integridad de los datos, esto significa que la información almacenada en las tablas debe ser válida, coherente y exacta.

Hasta el momento, hemos controlado y restringido la entrada de valores a un campo mediante el tipo de dato que le definimos (cadena, numéricos, etc.), la aceptación o no de valores nulos, el valor por defecto. También hemos asegurado que cada registro de una tabla sea único definiendo una clave primaria y empleando la propiedad identity.

SQL Server ofrece más alternativas, además de las aprendidas, para restringir y validar los datos, las veremos ordenadamente y al finalizar haremos un resumen de las mismas.

Comenzamos por las restricciones.

Las restricciones (constraints) son un método para mantener la integridad de los datos, asegurando que los valores ingresados sean válidos y que las

relaciones entre las tablas se mantenga. Se establecen a los campos y las tablas.

Pueden definirse al crear la tabla ("create table") o agregarse a una tabla existente (empleando "alter table") y se pueden aplicar a un campo o a varios. Se aconseja crear las tablas y luego agregar las restricciones.

Se pueden crear, modificar y eliminar las restricciones sin eliminar la tabla y volver a crearla.

El procedimiento almacenado del sistema "sp_helpconstraint" junto al nombre de la tabla, nos muestra información acerca de las restricciones de dicha tabla.

Cuando se agrega una restricción a una tabla, SQL Server comprueba los datos existentes.
Hay varios tipos de restricciones.

47 - Restricción default

La restricción "default" especifica un valor por defecto para un campo cuando no se inserta explícitamente en un comando "insert".

Anteriormente, para establecer un valor por defecto para un campo empleábamos la cláusula "default" al crear la tabla, por ejemplo:

```
create table libros(
 ...
 autor varchar(30) default 'Desconocido',
 ...
);
```

Cada vez que establecíamos un valor por defecto para un campo de una tabla, SQL Server creaba automáticamente una restricción "default" para ese campo de esa tabla.

Dicha restricción, a la cual no le dábamos un nombre, recibía un nombre dado por SQL Server que consiste "DF" (por default), seguido del nombre de la tabla, el nombre del campo y letras y números aleatorios.

Podemos agregar una restricción "default" a una tabla existente con la sintaxis básica siguiente:

```
 alter table NOMBRETABLA
 add constraint NOMBRECONSTRAINT
 default VALORPORDEFECTO
 for CAMPO;
```

En la sentencia siguiente agregamos una restricción "default" al campo autor de la tabla existente "libros", que almacena el valor "Desconocido" en dicho campo si no ingresamos un valor en un "insert":

```
 alter table libros
 add constraint DF_libros_autor
 default 'Desconocido'
 for autor;
```

Por convención, cuando demos el nombre a las restricciones "default" emplearemos un formato similar al que le da SQL Server: "DF_NOMBRETABLA_NOMBRECAMPO".

Solamente se permite una restricción "default" por campo y no se puede emplear junto con la propiedad "identity". Una tabla puede tener varias restricciones "default" para sus distintos campos.

La restricción "default" acepta valores tomados de funciones del sistema, por ejemplo, podemos establecer que el valor por defecto de un campo de tipo datetime sea "getdate()".

Podemos ver información referente a las restriciones de una tabla con el procedimiento almacenado "sp_helpcontraint":

```
 exec sp_helpconstraint libros;
```

aparecen varias columnas con la siguiente información:

```
- constraint_type: el tipo de restricción y sobre qué
campo está establecida
  (DEFAULT on column autor),
- constraint_name: el nombre de la restricción
(DF_libros_autor),
- delete_action y update_action: no tienen valores para
este tipo de restricción.
- status_enabled y status_for_replication: no tienen
valores para este tipo
  de restricción.
- constraint_keys: el valor por defecto (Desconocido).
```

Entonces, la restricción "default" especifica un valor por defecto para un campo cuando no se inserta explícitamente en un "insert", se puede establecer uno por campo y no se puede emplear junto con la propiedad "identity".

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) default 'Desconocido',
  editorial varchar(15),
  precio decimal(6,2)
);

go

insert into libros (titulo,editorial) values('Martin
Fierro','Emece');
insert into libros (titulo,editorial) values('Aprenda
PHP','Emece');

-- Veamos que SQL Server creó automáticamente una
restricción "default"
-- para el campo "autor":
exec sp_helpconstraint libros;

drop table libros;

create table libros(
  codigo int identity,
  titulo varchar(40),
  autor varchar(30),
  editorial varchar(15),
  precio decimal(6,2)
);

go
```

```
-- Agregamos una restricción "default" empleando "alter
table"
-- para que almacene el valor "Desconocido" en el campo
"autor":
alter table libros
  add constraint DF_libros_autor
  default 'Desconocido'
  for autor;

-- Veamos la restrición agregada anteriormente con el
procedimiento
-- almacenado "sp_helpcontraint":
exec sp_helpconstraint libros;

insert into libros (titulo,editorial) values('Martin
Fierro','Emece');
insert into libros default values;

-- Veamos cómo se almacenaron los registros sin valor
explícito
-- para el campo con restricción "default":
select * from libros;

-- Agregamos otra restricción "default" para el campo
"precio"
-- para que almacene el valor 0 en dicho campo:
alter table libros
  add constraint DF_libros_precio
  default 0
  for precio;

exec sp_helpconstraint libros;
```

48 - Restricción check

La restricción "check" especifica los valores que acepta un campo, evitando que se ingresen valores inapropiados.

La sintaxis básica es la siguiente:

```
 alter table NOMBRETABLA
 add constraint NOMBRECONSTRAINT
```

```
check CONDICION;
```

Trabajamos con la tabla "libros" de una librería que tiene los siguientes campos: codigo, titulo, autor, editorial, preciomin (que indica el precio para los minoristas) y preciomay (que indica el precio para los mayoristas).

Los campos correspondientes a los precios (minorista y mayorista) se definen de tipo decimal(5,2), es decir, aceptan valores entre -999.99 y 999.99. Podemos controlar que no se ingresen valores negativos para dichos campos agregando una restricción "check":

```
alter table libros
add constraint CK_libros_precio_positivo
check (preciomin>=0 and preciomay>=0);
```

Este tipo de restricción verifica los datos cada vez que se ejecuta una sentencia "insert" o "update", es decir, actúa en inserciones y actualizaciones.

Si la tabla contiene registros que no cumplen con la restricción que se va a establecer, la restricción no se puede establecer, hasta que todos los registros cumplan con dicha restricción.

La condición puede hacer referencia a otros campos de la misma tabla. Por ejemplo, podemos controlar que el precio mayorista no sea mayor al precio minorista:

```
alter table libros
add constraint CK_libros_preciominmay
check (preciomay<=preciomin);
```

Por convención, cuando demos el nombre a las restricciones "check" seguiremos la misma estructura: comenzamos con "CK", seguido del nombre de la tabla, del campo y alguna palabra con la cual podamos identificar fácilmente de qué se trata la restricción, por si tenemos varias restricciones "check" para el mismo campo.

Un campo puede tener varias restricciones restricciones "check" y una restricción "check" puede incluir varios campos.

Las condiciones para restricciones "check" también pueden pueden incluir un patrón o una lista de valores. Por ejemplo establecer que cierto campo conste de 4 caracteres, 2 letras y 2 dígitos:

```
 ...
 check (CAMPO like '[A-Z][A-Z][0-9][0-9]');
```

O establecer que cierto campo asuma sólo los valores que se listan:

```
 ...
 check (CAMPO in ('lunes','miercoles','viernes'));
```

No se puede aplicar esta restricción junto con la propiedad "identity".

Si un campo permite valores nulos, "null" es un valor aceptado aunque no esté incluido en la condición de restricción.

Si intentamos establecer una restricción "check" para un campo que entra en conflicto con otra restricción "check" establecida al mismo campo, SQL Server no lo permite.

Pero si establecemos una restricción "check" para un campo que entra en conflicto con una restricción "default" establecida para el mismo campo, SQL Server lo permite; pero al intentar ingresar un registro, aparece un mensaje de error.

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),
  preciomin decimal(5,2),
  preciomay decimal(5,2)
);

go

insert into libros values
('Uno','Bach','Planeta',22,20);
```

```
insert into libros values ('El
quijote','Cervantes','Emece',15,13);
insert into libros values ('Aprenda PHP','Mario
Molina','Siglo XXI',48,53);
insert into libros values ('Java en 10
minutos','Garcia','Siglo XXI',35,40);

-- Agregamos una restricción "check" para asegurar que
los valores de los campos
-- correspondientes a precios no puedan ser negativos:
alter table libros
  add constraint CK_libros_precios_positivo
  check (preciomin>=0 and preciomay>=0);

update libros set preciomay=48
  where titulo='Aprenda PHP';

delete from libros where titulo='Java en 10 minutos';

-- Agregamos la restricción "check" que impida que se
ingresen valores
-- para "preciomay" superiores a "preciomin":
alter table libros
 add constraint CK_libros_preciominmay
 check (preciomay<=preciomin);

exec sp_helpconstraint libros;

insert into libros default values;

select * from libros;

-- Genera un error si intentamos ingresar un preciomin
negativo
insert into libros values ('Java
Total','Martinez','Cuello',-1,40);
```

49 - Deshabilitar restricciones (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 condición de la restricción, si algún registro no la cumple, la restricción no se establecece.

Es posible deshabilitar esta comprobación en caso de restricciones "check".

Podemos hacerlo cuando agregamos la restricción "check" a una tabla para que SQL Server acepte los valores ya almacenados que infringen la restricción. Para ello debemos incluir la opción "with nocheck" en la instrucción "alter table":

```
alter table libros
 with nocheck
 add constraint CK_libros_precio
 check (precio>=0);
```

La restricción no se aplica en los datos existentes, pero si intentamos ingresar un nuevo valor que no cumpla la restricción, SQL Server no lo permite.

Entonces, para evitar la comprobación de datos existentes al crear la restricción, la sintaxis básica es la siguiente:

```
alter table TABLA
 with nocheck
 add constraint NOMBRERESTRICCION
 check (CONDICION);
```

Por defecto, si no especificamos, la opción es "with check".

También podemos deshabilitar las restricciones para agregar o actualizar datos sin comprobarla:

```
alter table libros
 nocheck constraint CK_libros_precio;
```

En el ejemplo anterior deshabilitamos la restricción "CK_libros_precio" para poder ingresar un valor negativo para "precio".

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

```
 alter table libros
  check constraint CK_libros_precio;
```

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

Para habilitar o deshabilitar restricciones la comprobación de datos en inserciones o actualizaciones, la sintaxis básica es:

```
 alter table NOMBRETABLA
  OPCIONdeRESTRICCION constraint NOMBRERESTRICCION;
```

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

Entonces, las cláusulas "check" y "nocheck" permiten habilitar o deshabilitar restricciones "check" (también las restricciones "foreign key" que veremos más adelante), a las demás se las debe eliminar ("default" y las que veremos posteriormente).

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),
  precio decimal(5,2)
);

go

insert into libros values ('Uno','Bach','Planeta',22);
insert into libros values ('El quijote','Cervantes','Emece',15);
insert into libros values ('Aprenda PHP','Mario Molina','Siglo XXI',-40);

-- Agregamos una restricción "check" para asegurar que los precios no
puedan ser
```

```
-- negativos, pero como ya tenemos almacenado un precio que infringe la
restricción,
-- vamos a especificar que no haya comprobación de datos existentes:
alter table libros
   with nocheck
   add constraint CK_libros_precio_positivo
   check (precio>=0);

-- Si intentamos ingresar un registro con precio negativo, no lo permite.
-- Para que lo permita, debemos dehabilitar la comprobación:
alter table libros
  nocheck constraint CK_libros_precio_positivo;

-- Ingresemos un registro con precio negativo (si lo permite):
insert into libros values('Java en 10 minutos',default,'Siglo XXI',-1);

-- Veamos si la restricción está o no habilitada:
-- La columna "status_enabled" nos informa que está deshabilitada
(Disabled)
exec sp_helpconstraint libros;

-- Habilitamos la restricción.
-- Si ahora intentamos ingresar un precio negativo SQL Server no lo
permitirá.
alter table libros
  check constraint CK_libros_precio_positivo;

select * from libros;
```

50 - Restricción primary key

Hemos visto las restricciones que se aplican a los campos, "default" y "check".

Ahora veremos las restricciones que se aplican a las tablas, que aseguran valores únicos para cada registro.

Hay 2 tipos: 1) primary key y 2) unique.

Anteriormente, para establecer una clave primaria para una tabla empleábamos la siguiente sintaxis al crear la tabla, por ejemplo:

```
 create table libros(
  codigo int not null,
  titulo varchar(30),
  autor varchar(30),
  editorial varchar(20),
  primary key(codigo)
 );
```

Cada vez que establecíamos la clave primaria para la tabla, SQL Server creaba automáticamente una restricción "primary key" para dicha tabla. Dicha restricción, a la cual no le dábamos un nombre, recibía un nombre dado por SQL Server que comienza con "PK" (por primary key), seguido del nombre de la tabla y una serie de letras y números aleatorios.

Podemos agregar una restricción "primary key" a una tabla existente con la sintaxis básica siguiente:

```
alter table NOMBRETABLA
add constraint NOMBRECONSTRAINT
primary key (CAMPO,...);
```

En el siguiente ejemplo 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);
```

Con esta restricción, si intentamos ingresar un registro con un valor para el campo "codigo" que ya existe o el valor "null", aparece un mensaje de error, porque no se permiten valores duplicados ni nulos. Igualmente, si actualizamos.

Por convención, cuando demos el nombre a las restricciones "primary key" seguiremos el formato "PK_NOMBRETABLA_NOMBRECAMPO".

Sabemos que cuando agregamos una restricción a una tabla que contiene información, SQL Server controla los datos existentes para confirmar que cumplen las exigencias de la restricción, si no los cumple, la restricción no se aplica y aparece un mensaje de error. Por ejemplo, si intentamos definir la restricción "primary key" para "libros" y hay registros con códigos repetidos o con un valor "null", la restricción no se establece.

Cuando establecíamos una clave primaria al definir la tabla, automáticamente SQL Server redefinía el campo como "not null"; pero al agregar una restricción "primary key", los campos que son clave primaria DEBEN haber sido definidos "not null" (o ser implícitamente "not null" si se definen identity).

SQL Server permite definir solamente una restricción "primary key" por tabla, que asegura la unicidad de cada registro de una tabla.

Si ejecutamos el procedimiento almacenado "sp_helpconstraint" junto al nombre de la tabla, podemos ver las restricciones "primary key" (y todos los tipos de restricciones) de dicha tabla.

Un campo con una restricción "primary key" puede tener una restricción "check".

Un campo "primary key" también acepta una restricción "default" (excepto si es identity), pero no tiene sentido ya que el valor por defecto solamente podrá ingresarse una vez; si intenta ingresarse cuando otro registro ya lo tiene almacenado, aparecerá un mensaje de error indicando que se intenta duplicar la clave.

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),
  primary key (codigo)
);

go

-- Veamos la restricción "primary key" que creó
automáticamente SQL Server:
exec sp_helpconstraint libros;

-- Vamos a eliminar la tabla y la crearemos nuevamente,
sin establecer la
-- clave primaria:
drop table libros;

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

```
  editorial varchar(15)
);

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);

-- Veamos la información respecto a ella:
exec sp_helpconstraint libros;
```