Introducción

El lenguaje de consulta estructurado (SQL) es un lenguaje de base de datos normalizado, utilizado por los diferentes motores de bases de datos para realizar determinadas operaciones sobre los datos o sobre la estructura de los mismos. Pero como sucede con cualquier sistema de normalización hay excepciones para casi todo; de hecho, cada motor de bases de datos tiene sus peculiaridades y lo hace diferente de otro motor, por lo tanto, el lenguaje SQL normalizado (ANSI) no nos servirá para resolver todos los problemas, aunque si se puede asegurar que cualquier sentencia escrita en ANSI será interpretable por cualquier motor de datos.

Breve Historia

La historia de SQL (que se pronuncia deletreando en inglés las letras que lo componen, es decir "ese-cu-ele" y no "siquel" como se oye a menudo) empieza en 1974 con la definición, por parte de Donald Chamberlin y de otras personas que trabajaban en los laboratorios de investigación de IBM, de un lenguaje para la especificación de las características de las bases de datos que adoptaban el modelo relacional. Este lenguaje se llamaba SEQUEL (Structured English Query Language) y se implementó en un prototipo llamado SEQUEL-XRM entre 1974 y 1975. Las experimentaciones con ese prototipo condujeron, entre 1976 y 1977, a una revisión del lenguaje (SEQUEL/2), que a partir de ese momento cambió de nombre por motivos legales, convirtiéndose en SQL. El prototipo (System R), basado en este lenguaje, se adoptó y utilizó internamente en IBM y lo adoptaron algunos de sus clientes elegidos. Gracias al éxito de este sistema, que no estaba todavía comercializado, también otras compañías empezaron a desarrollar sus productos relacionales basados en SQL. A partir de 1981, IBM comenzó a entregar sus productos relacionales y en 1983 empezó a vender DB2. En el curso de los años ochenta, numerosas compañías (por ejemplo Oracle y Sybase, sólo por citar algunos) comercializaron productos basados en SQL, que se convierte en el estándar industrial de hecho por lo que respecta a las bases de datos relacionales.

En 1986, el ANSI adoptó SQL (sustancialmente adoptó el dialecto SQL de IBM) como estándar para los lenguajes relacionales y en 1987 se transfomó en estándar ISO. Esta versión del estándar va con el nombre de SQL/86. En los años siguientes, éste ha sufrido diversas revisiones que han conducido primero a la versión SQL/89 y, posteriormente, a la actual SQL/92.

El hecho de tener un estándar definido por un lenguaje para bases de datos relacionales abre potencialmente el camino a la intercomunicabilidad entre todos los productos que se basan en él. Desde el punto de vista práctico, por desgracia las cosas fueron de otro modo. Efectivamente, en general cada productor adopta e implementa en la propia base de datos sólo el corazón del lenguaje SQL (el así llamado Entry level o al máximo el Intermediate level), extendiéndolo de manera individual según la propia visión que cada cual tenga del mundo de las bases de datos.

Actualmente, está en marcha un proceso de revisión del lenguaje por parte de los comités ANSI e ISO, que debería terminar en la definición de lo que en este momento se conoce como SQL3. Las características principales de esta nueva encarnación de SQL deberían ser su transformación en un lenguaje stand-alone (mientras ahora se usa como lenguaje hospedado

en otros lenguajes) y la introducción de nuevos tipos de datos más complejos que permitan, por ejemplo, el tratamiento de datos multimediales.

Componentes del SQL

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

Comandos

Existen dos tipos de comandos SQL:

- Los DLL que permiten crear y definir nuevas bases de datos, campos e índices.
- Los DML que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Comandos DLL:

Comando	Descripción
CREATE	Utilizado para crear nuevas tablas, campos e índices
DROP	Empleado para eliminar tablas e índices
ALTER	Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

Comandos DML:

Comando	Descripción
SELECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado
INSERT	Utilizado para cargar lotes de datos en la base de datos en una única operación.
UPDATE	Utilizado para modificar los valores de los campos y registros especificados
DELETE	Utilizado para eliminar registros de una tabla de una base de datos

Cláusulas

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

Cláusula	Descripción
FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los registros
WHERE	Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar
GROUP BY	Utilizada para separar los registros seleccionados en grupos específicos
HAVING	Utilizada para expresar la condición que debe satisfacer cada grupo
ORDER BY	Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico

Operadores lógicos

Operador	Uso
AND	Es el "y" lógico. Evalua dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdar si alguna de las dos es cierta.
NOT	Negación lógica. Devuelve el valor contrario de la expresión.

Operadores de Comparación

Operador	Uso
<	Menor que
>	Mayor que
<>	Distinto de
<=	Menor ó Igual que
>=	Mayor ó Igual que
=	Igual que
BETWEEN	Utilizado para especificar un intervalo de valores.
LIKE	Utilizado en la comparación de un modelo
In	Utilizado para especificar registros de una base de datos

Funciones de Agregado

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

Función	Descripción
AVG	Utilizada para calcular el promedio de los valores de un campo determinado
COUNT	Utilizada para devolver el número de registros de la selección
SUM	Utilizada para devolver la suma de todos los valores de un campo determinado
MAX	Utilizada para devolver el valor más alto de un campo especificado
MIN	Utilizada para devolver el valor más bajo de un campo especificado

Consultas de Selección

Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos, esta información es devuelta en forma de conjunto de registros que se pueden almacenar en un objeto recordset. Este conjunto de registros es modificable.

Consultas Básicas

La sintaxis básica de una consulta de selección es la siguiente:

```
SELECT Campos FROM Tabla;
```

En donde campos es la lista de campos que se deseen recuperar y tabla es el origen de los mismos, por ejemplo:

```
SELECT Nombre, Telefono FROM Clientes;
```

Esta consulta devuelve un recordset con el campo nombre y teléfono de la tabla clientes.

Devolver Literales

En determinadas ocasiones nos puede interesar incluir una columna con un texto fijo en una consulta de selección, por ejemplo, supongamos que tenemos una tabla de empleados y deseamos recuperar las tarifas semanales de los electricistas, podríamos realizar la siguiente consulta:

```
SELECT
    Empleados.Nombre, 'Tarifa semanal: ', Empleados.TarifaHora * 40
FROM
    Empleados
WHERE
    Empleados.Cargo = 'Electricista'
```

Ordenar los Registros

Adicionalmente se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la claúsula `ORDER BY Lista de Campos`. En donde `Lista de campos` representa los campos a ordenar. Ejemplo:

```
SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY Nombre;
```

Esta consulta devuelve los campos CodigoPostal, Nombre, Telefono de la tabla Clientes ordenados por el campo Nombre.

Se pueden ordenar los registros por mas de un campo, como por ejemplo:

```
SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY
CodigoPostal, Nombre;
```

Incluso se puede especificar el orden de los registros: ascendente mediante la claúsula `ASC` (se toma este valor por defecto) ó descendente (`DESC`).

```
SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY
CodigoPostal DESC , Nombre ASC;
```

Uso de Indices de las tablas

Si deseamos que la sentencia SQL utilice un índice para mostrar los resultados se puede utilizar la palabra reservada `INDEX` de la siguiente forma:

```
SELECT ... FROM Tabla (INDEX=Indice) ...
```

Normalmente los motores de las bases de datos deciden que indice se debe utilizar para la consulta, para ello utilizan criterios de rendimiento y sobre todo los campos de búsqueda especificados en la cláusula WHERE. Si se desea forzar a no utilizar ningún índice utilizaremos la siguiente sintaxis:

```
SELECT ... FROM Tabla (INDEX=0) ...
```

Consultas con Predicado

El predicado se incluye entre la claúsula y el primer nombre del campo a recuperar, los posibles predicados son:

Predicado	Descripción
ALL	Devuelve todos los campos de la tabla
TOP	Devuelve un determinado número de registros de la tabla
DISTINCT	Omite los registros cuyos campos seleccionados coincidan totalmente
DISTINCTROW	Omite los registros duplicados basandose en la totalidad del registro y no sólo en los campos seleccionados.

ALL

Si no se incluye ninguno de los predicados se asume ALL. El Motor de base de datos selecciona todos los registros que cumplen las condiciones de la instrucción SQL. No se conveniente abusar de este predicado ya que obligamos al motor de la base de datos a analizar la estructura de la tabla para averiguar los campos que contiene, es mucho más rápido indicar el listado de campos deseados.

```
SELECT ALL FROM Empleados;
SELECT * FROM Empleados;
```

TOP

Devuelve un cierto número de registros que entran entre al principio o al final de un rango especificado por una cláusula `ORDER BY`. Supongamos que queremos recuperar los nombres de los 25 primeros estudiantes del curso 1994:

```
SELECT TOP 25 Nombre, Apellido FROM Estudiantes ORDER BY Nota DESC;
```

Si no se incluye la cláusula `ORDER BY`, la consulta devolverá un conjunto arbitrario de 25 registros de la tabla Estudiantes. El predicado `TOP` no elige entre valores iguales. En el ejemplo anterior, si la nota media número 25 y la 26 son iguales, la consulta devolverá 26 registros. Se puede utilizar la palabra reservada `PERCENT` para devolver un cierto porcentaje de registros que caen al principio o al final de un rango especificado por la cláusula `ORDER BY`. Supongamos que en lugar de los 25 primeros estudiantes deseamos el 10 por ciento del curso:

```
SELECT TOP 10 PERCENT Nombre, Apellido FROM Estudiantes
ORDER BY Nota DESC;
```

El valor que va a continuación de `TOP` debe ser un Integer sin signo. `TOP` no afecta a la posible actualización de la consulta.

DISTINCT

Omite los registros que contienen datos duplicados en los campos seleccionados. Para que los valores de cada campo listado en la instrucción `SELECT` se incluyan en la consulta deben ser únicos.

Por ejemplo, varios empleados listados en la tabla Empleados pueden tener el mismo apellido. Si dos registros contienen López en el campo Apellido, la siguiente instrucción SQL devuelve un único registro:

```
SELECT DISTINCT Apellido FROM Empleados;
```

Con otras palabras el predicado `DISTINCT` devuelve aquellos registros cuyos campos indicados en la cláusula `SELECT` posean un contenido diferente. El resultado de una consulta que utiliza `DISTINCT` no es actualizable y no refleja los cambios subsiguientes realizados por otros usuarios.

DISTINCTROW

Este predicado no es compatible con ANSI. Que yo sepa a día de hoy sólo funciona con ACCESS.

Devuelve los registros diferentes de una tabla; a diferencia del predicado anterior que sólo se fijaba en el contenido de los campos seleccionados, éste lo hace en el contenido del registro completo independientemente de los campo indicados en la cláusula `SELECT`.

```
SELECT DISTINCTROW Apellido FROM Empleados;
```

Si la tabla empleados contiene dos registros: Antonio López y Marta López, el ejemplo del predicado `DISTINCT` devuleve un único registro con el valor López en el campo Apellido ya que busca no duplicados en dicho campo. Este último ejemplo devuelve dos registros con el valor López en el apellido ya que se buscan no duplicados en el registro completo.

Alias

En determinadas circunstancias es necesario asignar un nombre a alguna columna determinada de un conjunto devuelto, otras veces por simple capricho o por otras circunstancias. Para resolver todas ellas tenemos la palabra reservada `AS` que se encarga de asignar el nombre que deseamos a la columna deseada. Tomado como referencia el ejemplo anterior podemos hacer que la columna devuelta por la consulta, en lugar de llamarse apellido (igual que el campo devuelto) se llame Empleado. En este caso procederíamos de la siguiente forma:

```
SELECT DISTINCTROW Apellido AS Empleado FROM Empleados;
```

AS no es una palabra reservada de ANSI, existen diferentes sistemas de asignar los alias en función del motor de bases de datos. En ORACLE para asignar un alias a un campo hay que hacerlo de la siguiente forma:

```
SELECT Apellido AS "Empleado" FROM Empleados;
```

También podemos asignar alias a las tablas dentro de la consulta de selección, en esta caso hay que tener en cuenta que en todas las referencias que deseemos hacer a dicha tabla se ha de utilizar el alias en lugar del nombre. Esta técnica será de gran utilidad más adelante cuando se estudien las vinculaciones entre tablas. Por ejemplo:

```
SELECT Apellido AS Empleado FROM Empleados AS Trabajadores;
```

Para asignar alias a las tablas en ORACLE y SQL-SERVER los alias se asignan escribiendo el nombre de la tabla, dejando un espacio en blanco y escribiendo el Alias (se asignan dentro de la cláusula FROM).

```
SELECT Trabajadores.Apellido AS Empleado FROM Empleados Trabajadores;
```

Esta nomenclatura [Tabla].[Campo] se debe utilizar cuando se está recuperando un campo cuyo nombre se repite en varias de las tablas que se utilizan en la sentencia. No obstante cuando en la sentencia se emplean varias tablas es aconsejable utilizar esta nomenclatura para evitar el trabajo que supone al motor de datos averiguar en que tabla está cada uno de los campos indicados en la cláusula SELECT.

Recuperar Información de una base de Datos Externa

Para concluir este capítulo se debe hacer referencia a la recuperación de registros de bases de datos externa. En ocasiones es necesario la recuperación de información que se encuentra contenida en una tabla que no se encuentra en la base de datos que ejecutará la consulta o que en ese momento no se encuentra abierta, esta situación la podemos salvar con la palabra reservada IN de la siguiente forma:

```
SELECT DISTINCTROW Apellido AS Empleado FROM Empleados
IN 'c:\databases\gestion.mdb';
```

En donde c:\databases\gestion.mdb es la base de datos que contiene la tabla Empleados. Esta técnica es muy sencilla y común en bases de datos de tipo ACCESS en otros sistemas como SQL-SERVER u ORACLE, la cosa es más complicada la tener que existir relaciones de confianza entre los servidores o al ser necesaria la vinculación entre las bases de datos. Este ejemplo recupera la información de una base de datos de SQL-SERVER ubicada en otro servidor (se da por supuesto que los servidores están lincados):

```
SELECT Apellido FROM Servidor1.BaseDatos1.dbo.Empleados
```

Criterios de Selección

En el capítulo anterior se vio la forma de recuperar los registros de las tablas, las formas empleadas devolvían todos los registros de la mencionada tabla. A lo largo de este capítulo se estudiarán las posibilidades de filtrar los registros con el fin de recuperar solamente aquellos que cumplan una condiciones preestablecidas.

Antes de comenzar el desarrollo de este apartado hay que recalcar tres detalles de vital importancia. El primero de ellos es que cada vez que se desee establecer una condición referida a un campo de texto la condición de búsqueda debe ir encerrada entre comillas simples; la segunda es que no es posible establecer condiciones de búsqueda en los campos memo y; la tercera y última hace referencia a las fechas. A día de hoy no he sido capaz de encontrar una sintaxis que funcione en todos los sistemas, por lo que se hace necesario particularizarlas según el banco de datos:

Banco de Datos	Sintaxis	Ejemplo (para grabar la fecha 18 de mayo de 1969)
SQL-SERVER	Fecha = #mm-dd-aaaa#	Fecha = #05-18-1969# ó Fecha = 19690518
ORACLE	Fecha = to_date('YYYYDDMM','aaaammdd',)	Fecha = to_date('YYYYDDMM', '19690518')
ACCESS	Fecha = #mm-dd-aaaa#	Fecha = #05-18-1969#

Referente a los valores lógicos `True` o `False` cabe destacar que no son reconocidos en ORACLE, ni en este sistema de bases de datos ni en SQL-SERVER existen los campos de tipo `"SI/NO"` de ACCESS; en estos sistemas se utilizan los campos `BIT` que permiten almacenar valores de `0` ó `1`. Internamente, ACCESS, almacena en estos campos valores de `0` ó `-1`, así que todo se complica bastante, pero aprovechando la coincidencia del `0` para los valores `FALSE`, se puede utilizar la sintaxis siguiente que funciona en todos los casos: si se desea saber si el campo es falso `"... CAMPO = 0"` y para saber los verdaderos `"CAMPO <> 0"`.

Operadores Lógicos

Los operadores lógicos soportados por SQL son: `AND`, `OR`, `XOR`, `Eqv`, `Imp`, `Is` y `Not`. A excepción de los dos últimos todos poseen la siguiente sintaxis:

```
<expresión1>    operador    <expresión2>
```

En donde `expresión1` y `expresión2` son las condiciones a evaluar, el resultado de la operación varía en función del operador lógico. La tabla adjunta muestra los diferentes posibles resultados:

<expresión1>	Operador	<expresión2>	Resultado
Verdad	AND	Falso	Falso
Verdad	AND	Verdad	Verdad
Falso	AND	Verdad	Falso
Falso	AND	Falso	Falso
Verdad	OR	Falso	Verdad
Verdad	OR	Verdad	Verdad
Falso	OR	Verdad	Verdad
Falso	OR	Falso	Falso
Verdad	XOR	Verdad	Falso
Verdad	XOR	Falso	Verdad
Falso	XOR	Verdad	Verdad
Falso	XOR	Falso	Falso
Verdad	Eqv	Verdad	Verdad
Verdad	Eqv	Falso	Falso
Falso	Eqv	Verdad	Falso
Falso	Eqv	Falso	Verdad
Verdad	Imp	Verdad	Verdad
Verdad	Imp	Falso	Falso
Verdad	Imp	Null	Null
Falso	Imp	Verdad	Verdad
Falso	Imp	Falso	Verdad
Falso	Imp	Null	Verdad
Null	Imp	Verdad	Verdad
Null	Imp	Falso	Null
Null	Imp	Null	Null

Si a cualquiera de las anteriores condiciones le anteponemos el operador `NOT` el resultado de la operación será el contrario al devuelto sin el operador `NOT`.

El último operador denominado `IS` se emplea para comparar dos variables de tipo objeto `<Objeto1> Is <Objeto2>`. Este operador devuelve verdad si los dos objetos son iguales.

```
SELECT * FROM Empleados WHERE Edad > 25 AND Edad < 50;
SELECT * FROM Empleados WHERE (Edad > 25 AND Edad < 50) OR Sueldo =
100;
SELECT * FROM Empleados WHERE NOT Estado = 'Soltero';
SELECT * FROM Empleados WHERE (Sueldo > 100 AND Sueldo < 500) OR
(Provincia = 'Madrid' AND Estado = 'Casado');
```

Intervalos de Valores

Para indicar que deseamos recuperar los registros según el intervalo de valores de un campo emplearemos el operador Between cuya sintaxis es:

```
campo [Not] Between valor1 And valor2 (la condición Not es opcional)
```

En este caso la consulta devolvería los registros que contengan en "campo" un valor incluido en el intervalo valor1, valor2 (ambos inclusive). Si anteponemos la condición Not devolverá aquellos valores no incluidos en el intervalo.

```
SELECT * FROM Pedidos WHERE CodPostal Between 28000 And 28999;
(Devuelve los pedidos realizados en la provincia de Madrid)
SELECT IIf(CodPostal Between 28000 And 28999, 'Provincial',
'Nacional')
FROM Editores;
(Devuelve el valor 'Provincial' si el código postal se encuentra en el
intervalo,
'Nacional' en caso contrario)
```

El Operador Like

Se utiliza para comparar una expresión de cadena con un modelo en una expresión SQL. Su sintaxis es:

```
expresión Like modelo
```

En donde expresión es una cadena modelo o campo contra el que se compara expresión. Se puede utilizar el operador `Like` para encontrar valores en los campos que coincidan con el modelo especificado. Por modelo puede especificar un valor completo (Ana María), o se pueden utilizar caracteres comodín como los reconocidos por el sistema operativo para encontrar un rango de valores (`Like An*`).

El operador `Like` se puede utilizar en una expresión para comparar un valor de un campo con una expresión de cadena. Por ejemplo, si introduce `Like C*` en una consulta SQL, la consulta devuelve todos los valores de campo que comiencen por la letra C. En una consulta con parámetros, puede hacer que el usuario escriba el modelo que se va a utilizar.

El ejemplo siguiente devuelve los datos que comienzan con la letra P seguido de cualquier letra entre A y F y de tres dígitos:

```
Like 'P[A-F]###'
```

Este ejemplo devuelve los campos cuyo contenido empiece con una letra de la A a la D seguidas de cualquier cadena.

```
Like '[A-D]*'
```

En la tabla siguiente se muestra cómo utilizar el operador `Like` para comprobar expresiones con diferentes modelos.

Tipo de coincidencia	Modelo Planteado	Coincide	No coincide
Varios caracteres	'a*a'	'aa', 'aBa', 'aBBBa'	'aBC'
Carácter especial	'a[*]a'	'a*a'	'aaa'
Varios caracteres	'ab*'	'abcdefg', 'abc'	'cab', 'aab'
Un solo carácter	'a?a'	'aaa', 'a3a', 'aBa'	'aBBBa'
Un solo dígito	'a#a'	'a0a', 'a1a', 'a2a'	'aaa', 'a10a'
Rango de caracteres	'[a-z]'	'f', 'p', 'j'	'2', '&'
Fuera de un rango	'[!a-z]'	'9', '&', '%'	'b', 'a'
Distinto de un dígito	'[!0-9]'	'A', 'a', '&', '~'	'0', '1', '9'
Combinada	'a[!b-m]#'	'An9', 'az0', 'a99'	'abc', 'aj0'

En determinado motores de bases de datos, esta cláusula, no reconoce el asterisco como carácter comodín y hay que sustituirlo por el carácter tanto por ciento (%). Por ejemplo, en SQL-SERVER:

Ejemplo	Descripción
LIKE 'A%'	Todo lo que comience por A
LIKE '_NG'	Todo lo que comience por cualquier carácter y luego siga NG
LIKE '[AF]%'	Todo lo que comience por A ó F
LIKE '[A-F]%'	Todo lo que comience por cualquier letra comprendida entre la A y la F
LIKE '[A^B]%'	Todo lo que comience por A y la segunda letra no sea una B

El Operador In

Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los en una lista. Su sintaxis es:

```
expresión [Not] In(valor1, valor2, . . .)

SELECT * FROM Pedidos WHERE Provincia In ('Madrid', 'Barcelona',
'Sevilla');
```

La cláusula WHERE

La cláusula `WHERE` puede usarse para determinar qué registros de las tablas enumeradas en la cláusula `FROM` aparecerán en los resultados de la instrucción `SELECT`. Depués de escribir esta cláusula se deben especificar las condiciones expuestas en los dos primeros apartados de este capítulo. Si no se emplea esta cláusula, la consulta devolverá todas las filas de la tabla. `WHERE` es opcional, pero cuando aparece debe ir a continuación de `FROM`.

```
SELECT Apellidos, Salario FROM Empleados WHERE Salario > 21000;

SELECT Id_Producto, Existencias FROM Productos
WHERE Existencias <= Nuevo_Pedido;

SELECT * FROM Pedidos WHERE Fecha_Envio = #5/10/94#;

SELECT Apellidos, Nombre FROM Empleados WHERE Apellidos = 'King';

SELECT Apellidos, Nombre FROM Empleados WHERE Apellidos Like 'S*';

SELECT Apellidos, Salario FROM Empleados WHERE Salario Between 200 And
300;

SELECT Apellidos, Salario FROM Empl WHERE Apellidos Between 'Lon' And
'Tol';

SELECT Id_Pedido, Fecha_Pedido FROM Pedidos WHERE Fecha_Pedido
Between #1-1-94# And  #30-6-94#;

SELECT Apellidos, Nombre, Ciudad FROM Empleados WHERE Ciudad
In ('Sevilla', 'Los Angeles', 'Barcelona');
```

Agrupamiento de Registros y Funciones Agregadas

La cláusula GROUP BY

Combina los registros con valores idénticos, en la lista de campos especificados, en un único registro. Para cada registro se crea un valor sumario si se incluye una función SQL agregada, como por ejemplo `Sum` o `Count`, en la instrucción `SELECT`. Su sintaxis es:

```
SELECT campos FROM tabla WHERE criterio GROUP BY campos del grupo
```

`GROUP BY` es opcional. Los valores de resumen se omiten si no existe una función SQL agregada en la instrucción `SELECT`. Los valores Null en los campos `GROUP BY` se agrupan y no se omiten. No obstante, los valores Null no se evalúan en ninguna de las funciones SQL agregadas.

Se utiliza la cláusula `WHERE` para excluir aquellas filas que no desea agrupar, y la cláusula `HAVING` para filtrar los registros una vez agrupados.

A menos que contenga un dato Memo u Objeto OLE , un campo de la lista de campos `GROUP BY` puede referirse a cualquier campo de las tablas que aparecen en la cláusula `FROM`, incluso si el campo no esta incluido en la instrucción `SELECT`, siempre y cuando la instrucción `SELECT` incluya al menos una función SQL agregada.

Todos los campos de la lista de campos de `SELECT` deben o bien incluirse en la cláusula `GROUP BY` o como argumentos de una función SQL agregada.

```
SELECT Id_Familia, Sum(Stock) FROM Productos GROUP BY Id_Familia;
```

Una vez que `GROUP BY` ha combinado los registros, `HAVING` muestra cualquier registro agrupado por la cláusula `GROUP BY` que satisfaga las condiciones de la cláusula `HAVING`.

`HAVING` es similar a WHERE, determina qué registros se seleccionan. Una vez que los registros se han agrupado utilizando `GROUP BY`, `HAVING` determina cuales de ellos se van a mostrar.

```
SELECT Id_Familia Sum(Stock) FROM Productos GROUP BY Id_Familia
HAVING Sum(Stock) > 100 AND NombreProducto Like BOS*;
```

AVG (Media Aritmética)

Calcula la media aritmética de un conjunto de valores contenidos en un campo especificado de una consulta. Su sintaxis es la siguiente:

```
Avg(expr)
```

En donde `expr` representa el campo que contiene los datos numéricos para los que se desea calcular la media o una expresión que realiza un cálculo utilizando los datos de dicho campo. La media calculada por `Avg` es la media aritmética (la suma de los valores dividido por el número de valores). La función `Avg` no incluye ningún campo Null en el cálculo.

```
SELECT Avg(Gastos) AS Promedio FROM Pedidos WHERE Gastos > 100;
```

Count (Contar Registros)

Calcula el número de registros devueltos por una consulta. Su sintaxis es la siguiente:

```
Count(expr)
```

En donde `expr` contiene el nombre del campo que desea contar. Los operandos de `expr` pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL). Puede contar cualquier tipo de datos incluso texto.

Aunque `expr` puede realizar un cálculo sobre un campo, `Count` simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan en los registros. La función `Count` no cuenta los registros que tienen campos null a menos que `expr` sea el carácter comodín asterisco (*). Si utiliza un asterisco, Count calcula el número total de registros, incluyendo aquellos que contienen campos null. `Count(*)` es considerablemente más rápida que `Count(Campo)`. No se debe poner el asterisco entre dobles comillas ('*').

```
SELECT Count(*) AS Total  FROM Pedidos;
```

Si `expr` identifica a múltiples campos, la función `Count` cuenta un registro sólo si al menos uno de los campos no es Null. Si todos los campos especificados son Null, no se cuenta el registro. Hay que separar los nombres de los campos con ampersand (&).

```
SELECT Count(FechaEnvío & Transporte) AS Total FROM Pedidos;
```

Podemos hacer que el gestor cuente los datos diferentes de un determinado campo

```
SELECT Count(DISTINCT Localidad) AS Total FROM Pedidos;
```

Max y Min (Valores Máximos y Mínimos)

Devuelven el mínimo o el máximo de un conjunto de valores contenidos en un campo especifico de una consulta. Su sintaxis es:

```
Min(expr)
Max(expr)
```

En donde `expr` es el campo sobre el que se desea realizar el cálculo. Expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

```
SELECT Min(Gastos) AS ElMin FROM Pedidos WHERE Pais = 'España';
SELECT Max(Gastos) AS ElMax FROM Pedidos WHERE Pais  = 'España';
```

StDev y StDevP (Desviación Estándar)

Devuelve estimaciones de la desviación estándar para la población (el total de los registros de la tabla) o una muestra de la población representada (muestra aleatoria) . Su sintaxis es:

```
StDev(expr)
StDevP(expr)
```

En donde `expr` representa el nombre del campo que contiene los datos que desean evaluarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de `expr` pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

`StDevP` evalúa una población, y `StDev` evalúa una muestra de la población. Si la consulta contiene menos de dos registros (o ningún registro para `StDevP`), estas funciones devuelven un valor Null (el cual indica que la desviación estándar no puede calcularse).

```
SELECT StDev(Gastos) AS Desviacion FROM Pedidos WHERE Pais = 'España';
SELECT StDevP(Gastos) AS Desviacion FROM Pedidos WHERE Pais= 'España';
```

Sum (Sumar Valores)

Devuelve la suma del conjunto de valores contenido en un campo especifico de una consulta. Su sintaxis es:

```
Sum(expr)
```

En donde `expr` respresenta el nombre del campo que contiene los datos que desean sumarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de `expr` pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

```
SELECT Sum(PrecioUnidad * Cantidad) AS Total FROM DetallePedido;
```

Var y VarP (Varianza)

Devuelve una estimación de la varianza de una población (sobre el total de los registros) o una muestra de la población (muestra aleatoria de registros) sobre los valores de un campo. Su sintaxis es:

```
Var(expr)
VarP(expr)
```

`VarP` evalúa una población, y `Var` evalúa una muestra de la población. `Expr` el nombre del campo que contiene los datos que desean evaluarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

Si la consulta contiene menos de dos registros, `Var` y `VarP` devuelven Null (esto indica que la varianza no puede calcularse). Puede utilizar `Var` y `VarP` en una expresión de consulta o en una Instrucción SQL.

```
SELECT Var(Gastos) AS Varianza FROM Pedidos WHERE Pais = 'España';
SELECT VarP(Gastos) AS Varianza FROM Pedidos WHERE Pais = 'España';
```

COMPUTE de SQL-SERVER

Esta cláusula añade una fila en el conjunto de datos que se está recuperando, se utiliza para realizar cálculos en campos numéricos. `COMPUTE` actúa siempre sobre un campo o expresión del conjunto de resultados y esta expresión debe figurar exactamente igual en la cláusula `SELECT` y siempre se debe ordenar el resultado por la misma o al menos agrupar el resultado. Esta expresión no puede utilizar ningún `ALIAS`.

```
SELECT IdCliente, Count(IdPedido) FROM Pedidos
GROUP BY IdPedido HAVING Count(IdPedido) > 20 COMPUTE
Sum(Count(IdPedido))

SELECT IdPedido, (PrecioUnidad * Cantidad - Descuento) FROM [Detalles
de Pedidos]
ORDER BY IdPedido
COMPUTE  Sum((PrecioUnidad * Cantidad - Descuento)) // Calcula el
Total
BY IdPedido // Calcula el Subtotal
```

Consultas de Acción

Las consultas de acción son aquellas que no devuelven ningún registro, son las encargadas de acciones como añadir y borrar y modificar registros.

DELETE

Crea una consulta de eliminación que elimina los registros de una o más de las tablas listadas en la cláusula `FROM` que satisfagan la cláusula `WHERE`. Esta consulta elimina los registros completos, no es posible eliminar el contenido de algún campo en concreto. Su sintaxis es:

```
DELETE FROM Tabla WHERE criterio
```

Una vez que se han eliminado los registros utilizando una consulta de borrado, no puede deshacer la operación. Si desea saber qué registros se eliminarán, primero examine los resultados de una consulta de selección que utilice el mismo criterio y después ejecute la consulta de borrado. Mantenga copias de seguridad de sus datos en todo momento. Si elimina los registros equivocados podrá recuperarlos desde las copias de seguridad.

```
DELETE * FROM Empleados WHERE Cargo = 'Vendedor';
```

INSERT INTO

Agrega un registro en una tabla. Se la conoce como una consulta de datos añadidos. Esta consulta puede ser de dos tipos: Insertar un único registro ó Insertar en una tabla los registros contenidos en otra tabla.

Insertar un único Registro

En este caso la sintaxis es la siguiente:

```
INSERT INTO Tabla (campo1, campo2, .., campoN)
VALUES (valor1, valor2, ..., valorN)
```

Esta consulta graba en el campo1 el valor1, en el campo2 y valor2 y así sucesivamente. Hay que prestar especial atención a acotar entre comillas simples (') los valores literales (cadenas de caracteres) y las fechas indicarlas en formato mm-dd-aa y entre caracteres de almohadillas (#).

Para seleccionar registros e insertarlos en una tabla nueva

En este caso la sintaxis es la siguiente:

```
    SELECT campo1, campo2, ..., campoN INTO nuevatabla
FROM tablaorigen [WHERE criterios]
```

Se pueden utilizar las consultas de creación de tabla para archivar registros, hacer copias de seguridad de las tablas o hacer copias para exportar a otra base de datos o utilizar en informes que muestren los datos de un periodo de tiempo concreto. Por ejemplo, se podría crear un informe de Ventas mensuales por región ejecutando la misma consulta de creación de tabla cada mes.

Insertar Registros de otra Tabla

En este caso la sintaxis es:

```
INSERT INTO Tabla [IN base_externa] (campo1, campo2, ..., campoN)
SELECT TablaOrigen.campo1, TablaOrigen.campo2, ..., TablaOrigen.campoN
FROM TablaOrigen
```

En este caso se seleccionarán los campos 1,2, ..., n de la tabla origen y se grabarán en los campos 1,2,.., n de la Tabla. La condición `SELECT` puede incluir la cláusula `WHERE` para filtrar los registros a copiar. Si Tabla y TablaOrigen poseen la misma estrucutra podemos simplificar la sintaxis a:

```
INSERT INTO Tabla  SELECT TablaOrigen.* FROM TablaOrigen
```

De esta forma los campos de TablaOrigen se grabarán en Tabla, para realizar esta operación es necesario que todos los campos de TablaOrigen estén contenidos con igual nombre en Tabla. Con otras palabras que Tabla posea todos los campos de TablaOrigen (igual nombre e igual tipo).

En este tipo de consulta hay que tener especial atención con los campos contadores o autonuméricos puesto que al insertar un valor en un campo de este tipo se escribe el valor que contenga su campo homólogo en la tabla origen, no incrementandose como le corresponde.

Se puede utilizar la instrucción `INSERT INTO` para agregar un registro único a una tabla, utilizando la sintaxis de la consulta de adición de registro único tal y como se mostró anteriormente. En este caso, su código específica el nombre y el valor de cada campo del registro. Debe especificar cada uno de los campos del registro al que se le va a asignar un valor así como el valor para dicho campo. Cuando no se especifica dicho campo, se inserta el valor predeterminado o Null. Los registros se agregan al final de la tabla.

También se puede utilizar `INSERT INTO` para agregar un conjunto de registros pertenecientes a otra tabla o consulta utilizando la cláusula `SELECT ... FROM` como se mostró anteriormente en la sintaxis de la consulta de adición de múltiples registros. En este caso la cláusula `SELECT` especifica los campos que se van a agregar en la tabla destino especificada.

La tabla destino u origen puede especificar una tabla o una consulta. Si la tabla destino contiene una clave principal, hay que segurarse que es única, y con valores no-Null ; si no es así, no se agregarán los registros. Si se agregan registros a una tabla con un campo Contador , no se debe incluir el campo Contador en la consulta. Se puede emplear la cláusula `IN` para agregar registros a una tabla en otra base de datos.

Se pueden averiguar los registros que se agregarán en la consulta ejecutando primero una consulta de selección que utilice el mismo criterio de selección y ver el resultado. Una consulta de adición copia los registros de una o más tablas en otra. Las tablas que contienen los registros que se van a agregar no se verán afectadas por la consulta de adición. En lugar de agregar registros existentes en otra tabla, se puede especificar los valores de cada campo en un nuevo registro utilizando la cláusula `VALUES`. Si se omite la lista de campos, la cláusula `VALUES` debe incluir un valor para cada campo de la tabla, de otra forma fallará `INSERT`.

```
INSERT INTO Clientes SELECT Clientes_Viejos.* FROM Clientes_Nuevos;

SELECT Empleados.* INTO Programadores FROM  Empleados
WHERE Categoria = 'Programador'

INSERT INTO Empleados (Nombre, Apellido, Cargo)
VALUES ('Luis', 'Sánchez', 'Becario');

INSERT INTO Empleados SELECT Vendedores.* FROM Vendedores
WHERE Fecha_Contratacion < Now() - 30;
```

UPDATE

Crea una consulta de actualización que cambia los valores de los campos de una tabla especificada basándose en un criterio específico. Su sintaxis es:

```
UPDATE Tabla SET Campo1=Valor1, Campo2=Valor2, ... CampoN=ValorN
WHERE Criterio;
```

UPDATE es especialmente útil cuando se desea cambiar un gran número de registros o cuando éstos se encuentran en múltiples tablas. Puede cambiar varios campos a la vez. El ejemplo siguiente incrementa los valores Cantidad pedidos en un 10 por ciento y los valores Transporte en un 3 por ciento para aquellos que se hayan enviado al Reino Unido:

```
UPDATE Pedidos SET Pedido = Pedidos * 1.1, Transporte = Transporte *
1.03
WHERE PaisEnvío = 'ES';
UPDATE
```

no genera ningún resultado. Para saber qué registros se van a cambiar, hay que examinar primero el resultado de una consulta de selección que utilice el mismo criterio y después ejecutar la consulta de actualización.

```
UPDATE Empleados SET Grado = 5 WHERE Grado = 2;

UPDATE Productos SET Precio = Precio  * 1.1 WHERE Proveedor = 8 AND
Familia = 3;
```

Si en una consulta de actualización suprimimos la cláusula

```
WHERE
```

todos los registros de la tabla señalada serán actualizados.

```
UPDATE Empleados SET Salario = Salario * 1.1
```

- Tipos de datos

Tipos de datos

Los tipos de datos SQL se clasifican en 13 tipos de datos primarios y de varios sinónimos válidos reconocidos por dichos tipos de datos.

Tipos de datos primarios:

Tipo de Datos	Longitud	Descripción
BINARY	1 byte	Para consultas sobre tabla adjunta de productos de bases de datos que definen un tipo de datos Binario.
BIT	1 byte	Valores Si/No ó True/False
BYTE	1 byte	Un valor entero entre 0 y 255.
COUNTER	4 bytes	Un número incrementado automáticamente (de tipo Long)
CURRENCY	8 bytes	Un entero escalable entre 922.337.203.685.477,5808 y 922.337.203.685.477,5807.
DATETIME	8 bytes	Un valor de fecha u hora entre los años 100 y 9999.
SINGLE	4 bytes	Un valor en punto flotante de precisión simple con un rango de $-3.402823*10^{38}$ a $-1.401298*10^{-45}$ para valores negativos, $1.401298*10^{-45}$ a $3.402823*10^{38}$ para valores positivos, y 0.
DOUBLE	8 bytes	Un valor en punto flotante de doble precisión con un rango de $-1.79769313486232*10^{308}$ a $-4.94065645841247*10^{-324}$ para valores negativos, $4.94065645841247*10^{-324}$ a $1.79769313486232*10^{308}$ para valores positivos, y 0.
SHORT	2 bytes	Un entero corto entre -32,768 y 32,767.
LONG	4 bytes	Un entero largo entre -2,147,483,648 y 2,147,483,647.
LONGTEXT	1 byte por carácter	De cero a un máximo de 1.2 gigabytes.
LONGBINARY	Según se necesite	De cero 1 gigabyte. Utilizado para objetos OLE.
TEXT	1 byte por caracter	De cero a 255 caracteres.

La siguiente tabla recoge los sinonimos de los tipos de datos definidos:

Tipo de Dato	Sinónimos
BINARY	VARBINARY
BIT	BOOLEAN LOGICAL LOGICAL1 YESNO
BYTE	INTEGER1
COUNTER	AUTOINCREMENT
CURRENCY	MONEY
DATETIME	DATE TIME TIMESTAMP
SINGLE	FLOAT4 IEEESINGLE REAL
DOUBLE	FLOAT FLOAT8 IEEEDOUBLE NUMBER NUMERIC
SHORT	INTEGER2 SMALLINT
LONG	INT INTEGER INTEGER4
LONGBINARY	GENERAL OLEOBJECT
LONGTEXT	LONGCHAR MEMO NOTE
TEXT	ALPHANUMERIC CHAR CHARACTER STRING VARCHAR
VARIANT (No Admitido)	VALUE

Subconsultas

Una subconsulta es una instrucción `SELECT` anidada dentro de una instrucción `SELECT`, `SELECT...INTO`, `INSERT...INTO`, `DELETE`, o `UPDATE` o dentro de otra subconsulta.

Puede utilizar tres formas de sintaxis para crear una subconsulta:

```
comparación [ANY | ALL | SOME] (instrucción sql)
expresión [NOT] IN (instrucción sql)
[NOT] EXISTS (instrucción sql)
```

En donde:

comparación
: Es una expresión y un operador de comparación que compara la expresión con el resultado de la subconsulta.

expresión
: Es una expresión por la que se busca el conjunto resultante de la subconsulta.

instrucción sql
: Es una instrucción `SELECT`, que sigue el mismo formato y reglas que cualquier otra instrucción `SELECT`. Debe ir entre paréntesis.

Se puede utilizar una subconsulta en lugar de una expresión en la lista de campos de una instrucción `SELECT` o en una cláusula `WHERE` o `HAVING`. En una subconsulta, se utiliza una instrucción `SELECT` para proporcionar un conjunto de uno o más valores especificados para evaluar en la expresión de la cláusula `WHERE` o `HAVING`.

Se puede utilizar el predicado `ANY` o `SOME`, los cuales son sinónimos, para recuperar registros de la consulta principal, que satisfagan la comparación con cualquier otro registro recuperado en la subconsulta. El ejemplo siguiente devuelve todos los productos cuyo precio unitario es mayor que el de cualquier producto vendido con un descuento igual o mayor al 25 por ciento:

```
SELECT * FROM Productos WHERE PrecioUnidad > ANY
(SELECT PrecioUnidad FROM DetallePedido WHERE Descuento >= 0 .25);
```

El predicado `ALL` se utiliza para recuperar únicamente aquellos registros de la consulta principal que satisfacen la comparación con todos los registros recuperados en la subconsulta. Si se cambia `ANY` por `ALL` en el ejemplo anterior, la consulta devolverá únicamente aquellos productos cuyo precio unitario sea mayor que el de todos los productos vendidos con un descuento igual o mayor al 25 por ciento. Esto es mucho más restrictivo.

El predicado `IN` se emplea para recuperar únicamente aquellos registros de la consulta principal para los que algunos registros de la subconsulta contienen un valor igual. El ejemplo siguiente devuelve todos los productos vendidos con un descuento igual o mayor al 25 por ciento:

```
SELECT * FROM Productos WHERE IDProducto IN
(SELECT IDProducto FROM DetallePedido WHERE Descuento >=  0.25);
```

Inversamente se puede utilizar `NOT IN` para recuperar únicamente aquellos registros de la consulta principal para los que no hay ningún registro de la subconsulta que contenga un valor igual.

El predicado EXISTS (con la palabra reservada NOT opcional) se utiliza en comparaciones de verdad/falso para determinar si la subconsulta devuelve algún registro. Supongamos que deseamos recuperar todos aquellos clientes que hayan realizado al menos un pedido:

```
SELECT Clientes.Compañía, Clientes.Teléfono FROM Clientes WHERE EXISTS
(SELECT FROM Pedidos WHERE Pedidos.IdPedido = Clientes.IdCliente)
```

Esta consulta es equivalente a esta otra:

```
SELECT Clientes.Compañía, Clientes.Teléfono FROM Clientes WHERE
IdClientes IN
(SELECT Pedidos.IdCliente FROM Pedidos)
```

Se puede utilizar también alias del nombre de la tabla en una subconsulta para referirse a tablas listadas en la cláusula FROM fuera de la subconsulta. El ejemplo siguiente devuelve los nombres de los empleados cuyo salario es igual o mayor que el salario medio de todos los empleados con el mismo título. A la tabla Empleados se le ha dado el alias T1:

```
SELECT Apellido, Nombre, Titulo, Salario FROM Empleados AS T1
WHERE Salario >= (SELECT Avg(Salario) FROM Empleados
WHERE T1.Titulo = Empleados.Titulo) ORDER BY Titulo;
```

En el ejemplo anterior , la palabra reservada AS es opcional. Otros ejemplos:

```
SELECT Apellidos, Nombre, Cargo, Salario FROM Empleados
WHERE Cargo LIKE "Agente Ven*" AND Salario > ALL (SELECT Salario FROM
Empleados WHERE (Cargo LIKE "*Jefe*") OR (Cargo LIKE "*Director*"));
```

Obtiene una lista con el nombre, cargo y salario de todos los agentes de ventas cuyo salario es mayor que el de todos los jefes y directores.

```
SELECT DISTINCTROW NombreProducto, Precio_Unidad FROM Productos
WHERE (Precio_Unidad = (SELECT Precio_Unidad FROM Productos WHERE
Nombre_Producto = "Almíbar anisado");
```

Obtiene una lista con el nombre y el precio unitario de todos los productos con el mismo precio que el almíbar anisado.

```
SELECT DISTINCTROW Nombre_Contacto, Nombre_Compañia, Cargo_Contacto,
Telefono FROM Clientes WHERE (ID_Cliente IN (SELECT DISTINCTROW
ID_Cliente FROM Pedidos WHERE Fecha_Pedido >= #04/1/93# <#07/1/93#);
```

Obtiene una lista de las compañías y los contactos de todos los clientes que han realizado un pedido en el segundo trimestre de 1993.

```
SELECT Nombre, Apellidos FROM Empleados AS E WHERE EXISTS
(SELECT * FROM Pedidos AS O WHERE O.ID_Empleado = E.ID_Empleado);
```

Selecciona el nombre de todos los empleados que han reservado al menos un pedido.

```
SELECT DISTINCTROW Pedidos.Id_Producto, Pedidos.Cantidad,
(SELECT DISTINCTROW Productos.Nombre FROM Productos WHERE
Productos.Id_Producto = Pedidos.Id_Producto) AS ElProducto FROM
Pedidos WHERE Pedidos.Cantidad > 150 ORDER BY Pedidos.Id_Producto;
```

Recupera el Código del Producto y la Cantidad pedida de la tabla pedidos, extrayendo el nombre del producto de la tabla de productos.

```
SELECT NumVuelo, Plazas FROM Vuelos
WHERE Origen = 'Madrid' AND Exists
(SELECT T1.NumVuelo FROM Vuelos AS T1
 WHERE T1.PlazasLibres > 0 AND T1.NumVuelo=Vuelos.NumVuelo)
```

Recupera números de vuelo y capacidades de aquellos vuelos con destino Madrid y plazas libres.

Supongamos ahora que tenemos una tabla con los identificadores de todos nuestros productos y el stock de cada uno de ellos. En otra tabla se encuentran todos los pedidos que tenemos pendientes de servir. Se trata de averiguar que productos no se podemos servir por falta de stock.

```
SELECT PedidosPendientes.Nombre FROM PedidosPendientes
GROUP BY PedidosPendientes.Nombre
HAVING SUM(PedidosPendientes.Cantidad <
(SELECT Productos.Stock FROM Productos
 WHERE Productos.IdProducto = PedidosPendientes.IdProducto));
```

Supongamos que en nuestra tabla de empleados deseamos buscar todas las mujeres cuya edad sea mayor a la de cualquier hombre:

```
SELECT Empleados.Nombre FROM Empleados
WHERE Sexo = 'M' AND Edad > ANY
(SELECT Empleados.Edad FROM Empleados WHERE Sexo ='H')
```

ó lo que sería lo mismo:

```
SELECT Empleados.Nombre FROM Empleados WHERE Sexo = 'M' AND Edad >
(SELECT Max( Empleados.Edad )FROM Empleados WHERE Sexo ='H')
```

La siguiente tabla muestra algún ejemplo del operador `ANY` y `ALL`

Valor 1	Operador	Valor 2	Resultado
3	> ANY	(2,5,7)	Cierto
3	= ANY	(2,5,7)	Falso
3	= ANY	(2,3,5,7)	Cierto
3	> ANY	(2,5,7)	Falso
3	< ANY	(5,6,7)	Falso

El operacion `=ANY` es equivalente al operador `IN`, ambos devuelven el mismo resultado.

Consultas de Unión Internas

Consultas de Combinación entre tablas

Las vinculaciones entre tablas se realiza mediante la cláusula `INNER` que combina registros de dos tablas siempre que haya concordancia de valores en un campo común. Su sintaxis es:

```
SELECT campos FROM tb1 INNER JOIN tb2 ON tb1.campo1 comp tb2.campo2
```

En donde:

tb1, tb2
: Son los nombres de las tablas desde las que se combinan los registros.

campo1, campo2
: Son los nombres de los campos que se combinan. Si no son numéricos, los campos deben ser del mismo tipo de datos y contener el mismo tipo de datos, pero no tienen que tener el mismo nombre.

comp
: Es cualquier operador de comparación relacional : =, <, >, <=, >=, o <>.

Se puede utilizar una operación `INNER JOIN` en cualquier cláusula `FROM`. Esto crea una combinación por equivalencia, conocida también como unión interna. Las combinaciones Equi son las más comunes; éstas combinan los registros de dos tablas siempre que haya concordancia de valores en un campo común a ambas tablas. Se puede utilizar `INNER JOIN` con las tablas Departamentos y Empleados para seleccionar todos los empleados de cada departamento. Por el contrario, para seleccionar todos los departamentos (incluso si alguno de ellos no tiene ningún empleado asignado) se emplea `LEFT JOIN` o todos los empleados (incluso si alguno no está asignado a ningún departamento), en este caso `RIGHT JOIN`.

Si se intenta combinar campos que contengan datos Memo u Objeto OLE, se produce un error. Se pueden combinar dos campos numéricos cualesquiera, incluso si son de diferente tipo de datos. Por ejemplo, puede combinar un campo Numérico para el que la propiedad Size de su objeto Field está establecida como Entero, y un campo Contador.

El ejemplo siguiente muestra cómo podría combinar las tablas Categorías y Productos basándose en el campo IDCategoria:

```
SELECT Nombre_Categoría, NombreProducto
FROM Categorias INNER JOIN Productos
ON Categorias.IDCategoria = Productos.IDCategoria;
```

En el ejemplo anterior, IDCategoria es el campo combinado, pero no está incluido en la salida de la consulta ya que no está incluido en la instrucción `SELECT`. Para incluir el campo combinado, incluir el nombre del campo en la instrucción `SELECT`, en este caso, Categorias.IDCategoria.

También se pueden enlazar varias cláusulas `ON` en una instrucción `JOIN`, utilizando la sintaxis siguiente:

```
SELECT campos
FROM tabla1 INNER JOIN tabla2
ON tb1.campo1 comp tb2.campo1 AND
ON tb1.campo2 comp tb2.campo2) OR
ON tb1.campo3 comp tb2.campo3)];
```

También puede anidar instrucciones `JOIN` utilizando la siguiente sintaxis:

```
SELECT campos
FROM tb1 INNER JOIN
(tb2 INNER JOIN [( ]tb3
[INNER JOIN [( ]tablax [INNER JOIN ...)]
ON tb3.campo3 comp tbx.campox)]
ON tb2.campo2 comp tb3.campo3)
ON tb1.campo1 comp tb2.campo2;
```

Un `LEFT JOIN` o un `RIGHT JOIN` puede anidarse dentro de un `INNER JOIN`, pero un `INNER JOIN` no puede anidarse dentro de un `LEFT JOIN` o un `RIGHT JOIN`.

Por ejemplo:

```
SELECT DISTINCTROW Sum([Precio unidad] * [Cantidad]) AS [Ventas],
[Nombre] & " " & [Apellidos] AS [Nombre completo] FROM [Detalles de
pedidos],
Pedidos, Empleados, Pedidos INNER JOIN [Detalles de pedidos] ON
Pedidos.
[ID de pedido] = [Detalles de pedidos].[ID de pedido], Empleados INNER
JOIN
Pedidos ON Empleados.[ID de empleado] = Pedidos.[ID de empleado] GROUP
BY
[Nombre] & " " & [Apellidos];
```

Crea dos combinaciones equivalentes: una entre las tablas Detalles de pedidos y Pedidos, y la otra entre las tablas Pedidos y Empleados. Esto es necesario ya que la tabla Empleados no contiene datos de ventas y la tabla Detalles de pedidos no contiene datos de los empleados. La consulta produce una lista de empleados y sus ventas totales.

Si empleamos la cláusula `INNER` en la consulta se seleccionarán sólo aquellos registros de la tabla de la que hayamos escrito a la izquierda de `INNER JOIN` que contengan al menos un registro de la tabla que hayamos escrito a la derecha. Para solucionar esto tenemos dos cláusulas que sustituyen a la palabra clave `INNER`, estas cláusulas son `LEFT` y `RIGHT`. `LEFT` toma todos los registros de la tabla de la izquierda aunque no tengan ningún registro en la tabla de la izquierda. `RIGHT` realiza la misma operación pero al contrario, toma todos los registros de la tabla de la derecha aunque no tenga ningún registro en la tabla de la izquierda.

La sintaxis expuesta anteriormente pertenece a ACCESS, en donde todas las sentencias con la sintaxis funcionan correctamente. Los manuales de SQL-SERVER dicen que esta sintaxis es incorrecta y que hay que añadir la palabra reservada OUTER: `LEFT OUTER JOIN` y `RIGHT OUTER JOIN`. En la práctica funciona correctamente de una u otra forma.

No obstante, los `INNER JOIN` ORACLE no es capaz de interpretarlos, pero existe una sintaxis en formato ANSI para los `INNER JOIN` que funcionan en todos los sistemas. Tomando como referencia la siguiente sentencia:

```
SELECT Facturas.*, Albaranes.*
FROM Facturas INNER JOIN Albaranes ON Facturas.IdAlbaran =
Albaranes.IdAlbaran
WHERE Facturas.IdCliente = 325
```

La transformación de esta sentencia a formato ANSI sería la siguiente:

```
SELECT Facturas.*, Albaranes.* FROM Facturas, Albaranes
WHERE Facturas.IdAlbaran = Albaranes.IdAlbaran AND  Facturas.IdCliente
= 325
```

Como se puede observar los cambios realizados han sido los siguientes:

1. Todas las tablas que intervienen en la consulta se especifican en la cláusula `FROM`.
2. Las condiciones que vinculan a las tablas se especifican en la cláusula `WHERE` y se vinculan mediante el operador lógico `AND`.

Referente a los `OUTER JOIN`, no funcionan en ORACLE y además no conozco una sintaxis que funcione en los tres sistemas. La sintaxis en ORACLE es igual a la sentencia anterior pero añadiendo los caracteres `(+)` detrás del nombre de la tabla en la que deseamos aceptar valores nulos, esto equivale a un `LEFT JOIN`:

```
SELECT Facturas.*, Albaranes.* FROM Facturas, Albaranes
WHERE Facturas.IdAlbaran = Albaranes.IdAlbaran (+) AND
Facturas.IdCliente = 325
```

Y esto a un `RIGHT JOIN`:

```
SELECT Facturas.*, Albaranes.* FROM Facturas, Albaranes
WHERE Facturas.IdAlbaran (+) = Albaranes.IdAlbaran AND
Facturas.IdCliente = 325
```

En SQL-SERVER se puede utilizar una sintaxis parecida, en este caso no se utiliza los caracteres `(+)` sino los caracteres `=*` para el `LEFT JOIN` y `*=` para el `RIGHT JOIN`.

Consultas de Autocombinación

La autocombinación se utiliza para unir una tabla consigo misma, comparando valores de dos columnas con el mismo tipo de datos. La sintaxis en la siguiente:

```
SELECT alias1.columna, alias2.columna, ...
FROM tabla1 as alias1, tabla2 as alias2
WHERE alias1.columna = alias2.columna
AND otras condiciones
```

Por ejemplo, para visualizar el número, nombre y puesto de cada empleado, junto con el número, nombre y puesto del supervisor de cada uno de ellos se utilizaría la siguiente sentencia:

```
SELECT t.num_emp, t.nombre, t.puesto, t.num_sup,s.nombre, s.puesto
FROM empleados AS t, empleados AS s WHERE t.num_sup = s.num_emp
```

Consultas de Combinaciones no Comunes

La mayoría de las combinaciones están basadas en la igualdad de valores de las columnas que son el criterio de la combinación. Las no comunes se basan en otros operadores de combinación, tales como `NOT`, `BETWEEN`, `<>`, etc.

Por ejemplo, para listar el grado salarial, nombre, salario y puesto de cada empleado ordenando el resultado por grado y salario habría que ejecutar la siguiente sentencia:

```
SELECT grados.grado,empleados.nombre, empleados.salario,
empleados.puesto
FROM empleados, grados
WHERE empleados.salario
BETWEEN grados.salarioinferior AND grados.salariosuperior
ORDER BY grados.grado, empleados.salario
```

Para listar el salario medio dentro de cada grado salarial habría que lanzar esta otra sentencia:

```
SELECT grados.grado, AVG(empleados.salario) FROM empleados, grados
WHERE empleados.salario
BETWEEN grados.salarioinferior AND grados.salariosuperior
GROUP BY grados.grado
```

CROSS JOIN (SQL-SERVER)

Se utiliza en SQL-SERVER para realizar consultas de unión. Supongamos que tenemos una tabla con todos los autores y otra con todos los libros. Si deseáramos obtener un listado combinar ambas tablas de tal forma que cada autor apareciera junto a cada título, utilizaríamos la siguiente sintaxis:

```
SELECT Autores.Nombre, Libros.Titulo
FROM Autores CROSS JOIN Libros
```

SELF JOIN

`SELF JOIN` es una técnica empleada para conseguir el producto cartesiano de una tabla consigo misma. Su utilización no es muy frecuente, pero pongamos algún ejemplo de su utilización.

Supongamos la siguiente tabla (El campo autor es numérico, aunque para ilustrar el ejemplo utilice el nombre):

Código (Código del libro)	Autor (Nombre del Autor)
B0012	1. Francisco López
B0012	2. Javier Alonso
B0012	3. Marta Rebolledo
C0014	1. Francisco López
C0014	2. Javier Alonso
D0120	2. Javier Alonso
D0120	3. Marta Rebolledo

Queremos obtener, para cada libro, parejas de autores:

```
SELECT A.Codigo, A.Autor, B.Autor FROM Autores A, Autores B
WHERE  A.Codigo = B.Codigo
```

El resultado es el siguiente:

Código	Autor	Autor
B0012	1. Francisco López	1. Francisco López
B0012	1. Francisco López	2. Javier Alonso
B0012	1. Francisco López	3. Marta Rebolledo
B0012	2. Javier Alonso	2. Javier Alonso
B0012	2. Javier Alonso	1. Francisco López
B0012	2. Javier Alonso	3. Marta Rebolledo
B0012	3. Marta Rebolledo	3. Marta Rebolledo
B0012	3. Marta Rebolledo	2. Javier Alonso
B0012	3. Marta Rebolledo	1. Francisco López
C0014	1. Francisco López	1. Francisco López
C0014	1. Francisco López	2. Javier Alonso
C0014	2. Javier Alonso	2. Javier Alonso
C0014	2. Javier Alonso	1. Francisco López
D0120	2. Javier Alonso	2. Javier Alonso
D0120	2. Javier Alonso	3. Marta Rebolledo
D0120	3. Marta Rebolledo	3. Marta Rebolledo
D0120	3. Marta Rebolledo	2. Javier Alonso

Como podemos observar, las parejas de autores se repiten en cada uno de los libros, podemos omitir estas repeticiones de la siguiente forma

```
SELECT A.Codigo, A.Autor, B.Autor FROM Autores A, Autores B
WHERE A.Codigo = B.Codigo AND A.Autor < B.Autor
```

El resultado ahora es el siguiente:

Código	Autor	Autor
B0012	1. Francisco López	2. Javier Alonso
B0012	1. Francisco López	3. Marta Rebolledo
C0014	1. Francisco López	2. Javier Alonso
D0120	2. Javier Alonso	3. Marta Rebolledo

Ahora tenemos un conjunto de resultados en formato Autor - CoAutor.

Si en la tabla de empleados quisiéramos extraer todas las posibles parejas que podemos realizar, utilizaríamos la siguiente sentencia:

```
SELECT Hombres.Nombre, Mujeres.Nombre
FROM Empleados Hombre, Empleados Mujeres
WHERE Hombre.Sexo = 'Hombre' AND  Mujeres.Sexo = 'Mujer'
AND Hombres.Id <>Mujeres.Id
```

Para concluir supongamos la tabla siguiente:

Id	Nombre	SuJefe
1	Marcos	6
2	Lucas	1
3	Ana	2
4	Eva	1
5	Juan	6
6	Antonio	

Queremos obtener un conjunto de resultados con el nombre del empleado y el nombre de su jefe:

```
SELECT Emple.Nombre, Jefes.Nombre FROM Empleados Emple, Empleados Jefe
WHERE Emple.SuJefe = Jefes.Id
```

Consultas de Unión Externas

Se utiliza la operación `UNION` para crear una consulta de unión, combinando los resultados de dos o más consultas o tablas independientes. Su sintaxis es:

```
[TABLE] consulta1 UNION [ALL] [TABLE]
consulta2 [UNION [ALL] [TABLE] consultan [ ... ]]
```

En donde:

consulta1, consulta2, consultan
: Son instrucciones `SELECT`, el nombre de una consulta almacenada o el nombre de una tabla almacenada precedido por la palabra clave `TABLE`.

Puede combinar los resultados de dos o más consultas, tablas e instrucciones `SELECT`, en cualquier orden, en una única operación `UNION`. El ejemplo siguiente combina una tabla existente llamada Nuevas Cuentas y una instrucción `SELECT`:

```
TABLE [Nuevas Cuentas] UNION ALL SELECT * FROM Clientes
WHERE [Cantidad pedidos] > 1000;
```

Si no se indica lo contrario, no se devuelven registros duplicados cuando se utiliza la operación `UNION`, no obstante puede incluir el predicado `ALL` para asegurar que se devuelven todos los registros. Esto hace que la consulta se ejecute más rápidamente. Todas las consultas en una operación `UNION` deben pedir el mismo número de campos, no obstante los campos no tienen porqué tener el mismo tamaño o el mismo tipo de datos.

Se puede utilizar una cláusula `GROUP BY` y/o `HAVING` en cada argumento consulta para agrupar los datos devueltos. Puede utilizar una cláusula `ORDER BY` al final del último argumento consulta para visualizar los datos devueltos en un orden específico.

```
SELECT [Nombre de compañía], Ciudad FROM Proveedores WHERE
País = 'Brasil' UNION SELECT [Nombre de compañía], Ciudad FROM
Clientes
WHERE País = "Brasil"
```
Recupera los nombres y las ciudades de todos proveedores y clientes de Brasil

```
SELECT [Nombre de compañía], Ciudad FROM Proveedores WHERE País =
'Brasil'
UNION SELECT [Nombre de compañía], Ciudad FROM Clientes WHERE País =
'Brasil' ORDER BY Ciudad
```
Recupera los nombres y las ciudades de todos proveedores y clientes radicados en Brasil, ordenados por el nombre de la ciudad

```
SELECT [Nombre de compañía], Ciudad FROM Proveedores WHERE País =
'Brasil'
UNION SELECT [Nombre de compañía], Ciudad FROM Clientes WHERE País =
'Brasil' UNION SELECT [Apellidos], Ciudad FROM Empleados WHERE Región
=
'América del Sur'
```
Recupera los nombres y las ciudades de todos los proveedores y clientes de brasil y los apellidos y las ciudades de todos los empleados de América del Sur

```
TABLE [Lista de clientes] UNION TABLE [Lista de proveedores]
```
Recupera los nombres y códigos de todos los proveedores y clientes

Estructuras de las Tablas

Una base de datos en un sistema relacional está compuesta por un conjunto de tablas, que corresponden a las relaciones del modelo relacional. En la terminología usada en SQL no se alude a las relaciones, del mismo modo que no se usa el término atributo, pero sí la palabra columna, y no se habla de tupla, sino de línea.

Creación de Tablas Nuevas

```
CREATE TABLE tabla (campo1 tipo (tamaño) índice1 ,
campo2 tipo (tamaño) índice2 , ...,
índice multicampo , ... )
```

En donde:

Parte	Descripción
tabla	Es el nombre de la tabla que se va a crear.
campo1 campo2	Es el nombre del campo o de los campos que se van a crear en la nueva tabla. La nueva tabla debe contener, al menos, un campo.
tipo	Es el tipo de datos de campo en la nueva tabla. (Ver Tipos de Datos)
tamaño	Es el tamaño del campo sólo se aplica para campos de tipo texto.
índice1 índice2	Es una cláusula `CONSTRAINT` que define el tipo de indice a crear. Esta cláusula en opcional.
índice multicampos	Es una cláusula `CONSTRAINT` que define el tipo de indice multicampos a crear. Un índice multi campo es aquel que está indexado por el contenido de varios campos. Esta cláusula en opcional.

```
CREATE TABLE Empleados (Nombre TEXT (25) , Apellidos TEXT (50));
```

Crea una nueva tabla llamada Empleados con dos campos, uno llamado Nombre de tipo texto y longutid 25 y otro llamado apellidos con longitud 50.

```
CREATE TABLE Empleados (Nombre TEXT (10), Apellidos TEXT,
Fecha_Nacimiento DATETIME) CONSTRAINT IndiceGeneral UNIQUE
([Nombre], [Apellidos], [Fecha_Nacimiento]);
```

Crea una nueva tabla llamada Empleados con un campo Nombre de tipo texto y longitud 10, otro con llamado Apellidos de tipo texto y longitud predeterminada (50) y uno más llamado Fecha_Nacimiento de tipo Fecha/Hora. También crea un índice único (no permite valores repetidos) formado por los tres campos.

```
CREATE TABLE Empleados (ID INTEGER CONSTRAINT IndicePrimario PRIMARY,
Nombre TEXT, Apellidos TEXT, Fecha_Nacimiento DATETIME);
```

Crea una tabla llamada Empleados con un campo Texto de longitud predeterminada (50) llamado Nombre y otro igual llamado Apellidos, crea otro campo llamado Fecha_Nacimiento de tipo Fecha/Hora y el campo ID de tipo entero el que establece como clave principal.

La cláusula CONSTRAINT

Se utiliza la cláusula `CONSTRAINT` en las instrucciones `ALTER TABLE` y `CREATE TABLE` para crear o eliminar índices. Existen dos sintaxis para esta cláusula dependiendo si desea Crear ó Eliminar un índice de un único campo o si se trata de un campo multiíndice. Si se utiliza el

motor de datos de Microsoft, sólo podrá utilizar esta cláusula con las bases de datos propias de dicho motor.

Para los índices de campos únicos:

```
CONSTRAINT nombre {PRIMARY KEY | UNIQUE | REFERENCES tabla externa
[(campo externo1, campo externo2)]}
```

Para los índices de campos múltiples:

```
CONSTRAINT nombre {PRIMARY KEY (primario1[, primario2 [, ...]]) |
UNIQUE (único1[, único2 [, ...]]) |
FOREIGN KEY (ref1[, ref2 [, ...]]) REFERENCES tabla externa [(campo
externo1
[,campo externo2 [, ...]])]}
```

Parte	Descripción
nombre	Es el nombre del índice que se va a crear.
primarioN	Es el nombre del campo o de los campos que forman el índice primario.
únicoN	Es el nombre del campo o de los campos que forman el índice de clave única.
refN	Es el nombre del campo o de los campos que forman el índice externo (hacen referencia a campos de otra tabla).
tabla externa	Es el nombre de la tabla que contiene el campo o los campos referenciados en refN
campos externos	Es el nombre del campo o de los campos de la tabla externa especificados por ref1, ref2, ..., refN

Si se desea crear un índice para un campo cuando se esta utilizando las instrucciones `ALTER TABLE` o `CREATE TABLE` la cláusula `CONSTRAINT` debe aparecer inmediatamente después de la especificación del campo indexado.

Si se desea crear un índice con múltiples campos cuando se está utilizando las instrucciones `ALTER TABLE` o `CREATE TABLE` la cláusula `CONSTRAINT` debe aparecer fuera de la cláusula de creación de tabla.

Tipo de Indice	Descripción
UNIQUE	Genera un índece de clave única. Lo que implica que los registros de la tabla no pueden contener el mismo valor en los campos indexados.
PRIMARY KEY	Genera un índice primario el campo o los campos especificados. Todos los campos de la clave principal deben ser únicos y no nulos, cada tabla sólo puede contener una única clave principal.
FOREIGN KEY	Genera un índice externo (toma como valor del índice campos contenidos en otras tablas). Si la clave principal de la tabla externa consta de más de un campo, se debe utilizar una definición de índice de múltiples campos, listando todos los campos de referencia, el nombre de la tabla externa, y los nombres de los campos referenciados en la tabla externa en el mismo orden que los campos de referencia listados. Si los campos referenciados son la clave principal de la tabla externa, no tiene que especificar los campos referenciados, predeterminado por valor, el motor Jet se comporta como si la clave principal de la tabla externa fueran los campos referenciados.

Creación de Indices

Si se utiliza el motor de datos Jet de Microsoft sólo se pueden crear índices en bases de datos del mismo motor. La sintaxis para crear un índice en una tabla ya definida en la siguiente:

```
CREATE [ UNIQUE ] INDEX índice
ON tabla (campo [ASC|DESC][, campo [ASC|DESC], ...])
[WITH { PRIMARY | DISALLOW NULL | IGNORE NULL }]
```

En donde:

Parte	Descripción
índice	Es el nombre del índice a crear.
tabla	Es el nombre de una tabla existentes en la que se creará el índice.
campo	Es el nombre del campo o lista de campos que consituyen el índice.
ASC|DESC	Indica el orden de los valores de lso campos ASC indica un orden ascendente (valor predeterminado) y DESC un orden descendente.
UNIQUE	Indica que el indice no puede contener valores duplicados.
DISALLOW NULL	Prohibe valores nulos en el índice
IGNORE NULL	Excluye del índice los valores nulos incluidos en los campos que lo componen.
PRIMARY	Asigna al índice la categoría de clave principal, en cada tabla sólo puede existir un único indice que sea "Clave Principal". Si un índice es clave principal implica que que no puede contener valores nulos ni duplicados.

En el caso de ACCESS, se puede utilizar `CREATE INDEX` para crear un pseudo índice sobre una tabla adjunta en una fuente de datos ODBC tal como SQL Server que no tenga todavía un índice. No necesita permiso o tener acceso a un servidor remoto para crear un pseudo índice, además la base de datos remota no es consciente y no es afectada por el pseudo índice. Se utiliza la misma sintaxis para las tabla adjunta que para las originales. Esto es especialmente útil para crear un índice en una tabla que sería de sólo lectura debido a la falta de un índice.

```
CREATE INDEX MiIndice ON Empleados (Prefijo, Telefono);
```
Crea un índice llamado MiIndice en la tabla empleados con los campos Prefijo y Telefono.

```
CREATE UNIQUE INDEX MiIndice ON Empleados (ID) WITH DISALLOW NULL;
```
Crea un índice en la tabla Empleados utilizando el campo ID, obligando que el campo ID no contenga valores nulos ni repetidos.

Modificar el Diseño de una Tabla

`ALTER TABLE` modifica el diseño de una tabla ya existente, se puden modificar los campos o los índices existentes. Su sintaxis es:

```
ALTER TABLE tabla {ADD {COLUMN tipo de campo[(tamaño)] [CONSTRAINT
índice]
CONSTRAINT índice multicampo} |
DROP {COLUMN campo I CONSTRAINT nombre del índice} }
```

En donde:

Parte	Descripción
tabla	Es el nombre de la tabla que se desea modificar.
campo	Es el nombre del campo que se va a añadir o eliminar.
tipo	Es el tipo de campo que se va a añadir.
tamaño	El el tamaño del campo que se va a añadir (sólo para campos de texto).
índice	Es el nombre del índice del campo (cuando se crean campos) o el nombre del índice de la tabla que se desea eliminar.
índice multicampo	Es el nombre del índice del campo multicampo (cuando se crean campos) o el nombre del índice de la tabla que se desea eliminar.

Operación	Descripción
ADD COLUMN	Se utiliza para añadir un nuevo campo a la tabla, indicando el nombre, el tipo de campo y opcionalmente el tamaño (para campos de tipo texto).
ADD	Se utliza para agregar un índice de multicampos o de un único campo.
DROP COLUMN	Se utliza para borrar un campo. Se especifica únicamente el nombre del campo.
DROP	Se utiliza para eliminar un índice. Se especifica únicamente el nombre del índice a continuación de la palabra reservada CONSTRAINT.

```
ALTER TABLE Empleados ADD COLUMN Salario CURRENCY;
```
Agrega un campo Salario de tipo Moneda a la tabla Empleados.

```
ALTER TABLE Empleados DROP COLUMN Salario;
```
Elimina el campo Salario de la tabla Empleados.

```
ALTER TABLE Pedidos ADD CONSTRAINT RelacionPedidos FOREIGN KEY
(ID_Empleado) REFERENCES Empleados (ID_Empleado);
```
Agrega un indice externo a la tabla Pedidos. El índice externo se basa en el campo ID_Empleado y se refiere al campo ID_Empleado de la tabla Empleados. En este ejemplo no es necesario indicar el campo junto al nombre de la tabla en la cláusula REFERENCES, pues ID_Empleado es la clave principal de la tabla Empleados.

```
ALTER TABLE Pedidos DROP CONSTRAINT RelacionPedidos;
```
Elimina el índide de la tabla Pedidos.

Cursores

En algunos SGDB es posible la abertura de cursores de datos desde el propio entorno de trabajo, para ello se utilizan, normalmente procedimientos almacenados. La sintaxis para definir un cursor es la siguiente:

```
DECLARE nombre-cursor FOR especificacion-consulta [ORDER BY]
```

Por ejemplo:

```
DECLARE
    Mi_Cursor
FOR
    SELECT num_emp, nombre, puesto, salario
    FROM empleados
    WHERE num_dept = 'informatica'
```

Este comando es meramente declarativo, simplemente especifica las filas y columnas que se van a recuperar. La consulta se ejecuta cuando se abre o se activa el cursor. La cláusula [ORDER BY] es opcional y especifica una ordenación para las filas del cursor; si no se especifica, la ordenación de las filas es definida el gestor de SGBD.

Para abrir o activar un cursor se utiliza el comando OPEN del SQL, la sintaxis en la siguiente:

```
OPEN nombre-cursor [USING lista-variables]
```

Al abrir el cursor se evalúa la consulta que aparece en su definición, utilizando los valores actuales de cualquier parámetro referenciado en la consulta, para producir una colección de filas. El puntero se posiciona delante de la primera fila de datos (registro actual), esta sentencia no recupera ninguna fila.

Una vez abierto el cursos se utiliza la cláusula FETCH para recuperar las filas del cursor, la sintaxis es la siguiente:

```
FETCH nombre-cursor  INTO  lista-variables
```

Lista - variables son las variables que van a contener los datos recuperados de la fila del cursor, en la definición deben ir separadas por comas. En la lista de variables se deben definir tantas variables como columnas tenga la fila a recuperar.

Para cerrar un cursor se utiliza el comando CLOSE, este comando hace desaparecer el puntero sobre el registro actual. La sintaxis es:

```
CLOSE nombre-cursor
```

Por último, y para eliminar el cursor se utiliza el comando DROP CURSOR. Su sintaxis es la siguiente:

```
DROP CURSOR nombre-cursor
```

Ejemplo (sobre SQL-SERVER):

```
'Abrir un cursor y recorrelo
DECLARE Employee_Cursor CURSOR FOR
SELECT LastName, FirstName
FROM Northwind.dbo.Employees
WHERE LastName like 'B%'

OPEN Employee_Cursor

FETCH NEXT FROM Employee_Cursor

WHILE @@FETCH_STATUS = 0
BEGIN
    FETCH NEXT FROM Employee_Cursor
END
CLOSE Employee_Cursor
DEALLOCATE Employee_Cursor

'Abrir un cursor e imprimir su contenido
SET NOCOUNT ON
DECLARE
@au_id varchar(11),
@au_fname varchar(20),
@au_lname varchar(40),
@message varchar(80),
@title varchar(80)

PRINT "-------- Utah Authors report --------"

DECLARE authors_cursor CURSOR FOR
SELECT au_id, au_fname, au_lname
FROM authors
WHERE state = "UT"
ORDER BY au_id

OPEN authors_cursor
FETCH NEXT FROM authors_cursor
INTO @au_id, @au_fname, @au_lname

WHILE @@FETCH_STATUS = 0
BEGIN
    PRINT " "
    SELECT
        @message = "----- Books by Author: " +
        @au_fname + " " + @au_lname
    PRINT @message

    DECLARE titles_cursor CURSOR FOR
    SELECT t.title
    FROM titleauthor ta, titles t
    WHERE ta.title_id = t.title_id AND ta.au_id = au_id

    OPEN titles_cursor
    FETCH NEXT FROM titles_cursor INTO @title
    IF @@FETCH_STATUS <> 0
        PRINT " <<No Books>>"
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SELECT @message = " " + @title
        PRINT @message
        FETCH NEXT FROM titles_cursor INTO @title
    END
```

```
    CLOSE titles_cursor
    DEALLOCATE titles_cursor

    FETCH NEXT FROM authors_cursor
    INTO @au_id, @au_fname, @au_lname
END
CLOSE authors_cursor
DEALLOCATE authors_cursor
GO


'Recorrer un cursor
USE pubs
GO
DECLARE authors_cursor CURSOR FOR
SELECT au_lname
FROM authors
WHERE au_lname LIKE "B%"
ORDER BY au_lname

OPEN authors_cursor
FETCH NEXT FROM authors_cursor
WHILE @@FETCH_STATUS = 0
BEGIN
    FETCH NEXT FROM authors_cursor
END
CLOSE authors_cursor
DEALLOCATE authors_cursor


'Recorrer un cursor guardando los valores en variables
USE pubs
GO
DECLARE @au_lname varchar(40)
DECLARE @au_fname varchar(20)

DECLARE authors_cursor CURSOR FOR
SELECTau_lname, au_fname
FROM authors
WHERE au_lname LIKE "B%"
ORDER BY au_lname, au_fname

OPEN authors_cursor
FETCH NEXT FROM authors_cursor INTO @au_lname, @au_fname
WHILE @@FETCH_STATUS = 0
BEGIN
    PRINT "Author: " + @au_fname + " " + @au_lname
    FETCH NEXT FROM authors_cursor
    INTO @au_lname, @au_fname
END
CLOSE authors_cursor
DEALLOCATE authors_cursor
```

- Consulta de Referencias Cruzadas (Access)

Consultas de Referencias Cruzadas (Access)

Una consulta de referencias cruzadas es aquella que nos permite visualizar los datos en filas y en columnas, estilo tabla, por ejemplo:

Producto / Año	1996	1997
Pantalones	1.250	3.000
Camisas	8.560	1.253
Zapatos	4.369	2.563

Si tenemos una tabla de productos y otra tabla de pedidos, podemos visualizar en total de productos pedidos por año para un artículo determinado, tal y como se visualiza en la tabla anterior. La sintaxis para este tipo de consulta es la siguiente:

```
TRANSFORM función agregada instrucción select PIVOT campo pivot
[IN (valor1[, valor2[, ...]])]
```

En donde:

función agregada
: Es una función SQL agregada que opera sobre los datos seleccionados.

instrucción select
: Es una instrucción `SELECT`.

campo pivot
: Es el campo o expresión que desea utilizar para crear las cabeceras de la columna en el resultado de la consulta.

valor1, valor2
: Son valores fijos utilizados para crear las cabeceras de la columna.

Para resumir datos utilizando una consulta de referencia cruzada, se seleccionan los valores de los campos o expresiones especificadas como cabeceras de columnas de tal forma que pueden verse los datos en un formato más compacto que con una consulta de selección.

`TRANSFORM` es opcional pero si se incluye es la primera instrucción de una cadena SQL. Precede a la instrucción `SELECT` que especifica los campos utilizados como encabezados de fila y una cláusula `GROUP BY` que especifica el agrupamiento de las filas. Opcionalmente puede incluir otras cláusulas como por ejemplo `WHERE`, que especifica una selección adicional o un criterio de ordenación.

Los valores devueltos en campo pivot se utilizan como encabezados de columna en el resultado de la consulta. Por ejemplo, al utilizar las cifras de ventas en el mes de la venta como pivot en una consulta de referencia cruzada se crearían 12 columnas. Puede restringir el campo pivot para crear encabezados a partir de los valores fijos (valor1, valor2) listados en la cláusula opcional `IN`.

También puede incluir valores fijos, para los que no existen datos, para crear columnas adicionales.

Ejemplos

```
TRANSFORM Sum(Cantidad) AS Ventas SELECT Producto, Cantidad FROM
```

```
Pedidos WHERE Fecha Between #01-01-98# And  #12-31-98# GROUP BY
Producto
ORDER BY Producto PIVOT DatePart("m", Fecha);
```

Crea una consulta de tabla de referencias cruzadas que muestra las ventas de productos por mes para un año específico. Los meses aparecen de izquierda a derecha como columnas y los nombres de los productos aparecen de arriba hacia abajo como filas.

```
TRANSFORM Sum(Cantidad) AS Ventas SELECT Compania FROM Pedidos
WHERE Fecha Between #01-01-98# And  #12-31-98# GROUP BY Compania
ORDER BY Compania PIVOT "Trimestre " & DatePart("q", Fecha) In
('Trimestre1',
'Trimestre2', 'Trimestre 3', 'Trimestre 4');
```

Crea una consulta de tabla de referencias cruzadas que muestra las ventas de productos por trimestre de cada proveedor en el año indicado. Los trimestres aparecen de izquierda a derecha como columnas y los nombres de los proveedores aparecen de arriba hacia abajo como filas.

Un caso práctico:

Se trata de resolver el siguiente problema: tenemos una tabla de productos con dos campos, el código y el nombre del producto, tenemos otra tabla de pedidos en la que anotamos el código del producto, la fecha del pedido y la cantidad pedida. Deseamos consultar los totales de producto por año, calculando la media anual de ventas.

Estructura y datos de las tablas:

1. Artículos:

ID	Nombre
1	Zapatos
2	Pantalones
3	Blusas

2. Pedidos:

Id	Fecha	Cantidad
1	11/11/1996	250
2	11/11/1996	125
3	11/11/1996	520
1	12/10/1996	50
2	04/05/1996	250
3	05/08/1996	100
1	01/01/1997	40
2	02/08/1997	60
3	05/10/1997	70
1	12/12/1997	8
2	15/12/1997	520
3	17/10/1997	1250

Para resolver la consulta planteamos la siguiente consulta:

```
TRANSFORM Sum(Pedidos.Cantidad) AS Resultado SELECT Nombre AS
Producto,
Pedidos.Id AS Código, Sum(Pedidos.Cantidad) AS TOTAL,
Avg(Pedidos.Cantidad)
AS Media FROM Pedidos INNER JOIN Artículos ON Pedidos.Id =
Artículos.Id
GROUP BY Pedidos.Id, Artículos.Nombre PIVOT Year(Fecha);
```

y obtenemos el siguiente resultado:

Producto	Código	TOTAL	Media	1996	1997
Zapatatos	1	348	87	300	48
Pantalones	2	955	238,75	375	580
Blusas	3	1940	485	620	1320

Comentarios a la consulta:

La clásula `TRANSFORM` indica el valor que deseamos visualizar en las columnas que realmente pertenecen a la consulta, en este caso 1996 y 1997, puesto que las demás columnas son opcionales.

`SELECT` especifica el nombre de las columnas opcionales que deseamos visualizar, en este caso Producto, Código, Total y Media, indicando el nombre del campo que deseamos mostrar en cada columna o el valor de la misma. Si incluimos una función de cálculo el resultado se hará en base a los datos de la fila actual y no al total de los datos.

`FROM` especifica el origen de los datos. La primera tabla que debe figurar es aquella de donde deseamos extraer los datos, esta tabla debe contener al menos tres campos, uno para los títulos de la fila, otros para los títulos de la columna y otro para calcular el valor de las celdas.

En este caso en concreto se deseaba visualizar el nombre del producto, como el tabla de pedidos sólo figuraba el código del mismo se añadió una nueva columna en la cláusula select llamada Producto que se corresponda con el campo Nombre de la tabla de artículos. Para vincular el código del artículo de la tabla de pedidos con el nombre del misma de la tabla artículos se insertó la cláusula `INNER JOIN`.

La cláusula `GROUP BY` especifica el agrupamiento de los registros, contrariamente a los manuales de instrucción esta cláusula no es opcional ya que debe figurar siempre y debemos agrupar los registros por el campo del cual extraemos la información. En este caso existen dos campos del cual extraemos la información: pedidos.cantidad y artículos.nombre, por ellos agrupamos por los campos.

Para finalizar la cláusula `PIVOT` indica el nombre de las columnas no opcionales, en este caso 1996 y 1997 y como vamos al dato que aparecerá en las columnas, en este caso empleamos el año en que se produjo el pedido, extrayéndolo del campo pedidos.fecha.

Otras posibilidades de fecha de la cláusula `PIVOT` son las siguientes:

1. Para agrupamiento por Trimestres

```
PIVOT "Tri " & DatePart("q",[Fecha]);
```

2. Para agrupamiento por meses (sin tener en cuenta el año)

```
3.  PIVOT Format([Fecha],"mmm") In ("Ene", "Feb", "Mar", "Abr",
"May",
"Jun", "Jul", "Ago", "Sep", "Oct", "Nov", "Dic");
```

4. Para agrupar por días

```
PIVOT Format([Fecha],"Short Date");
```

Full Text (SQL Server)

Consultas e índices de texto

El principal requisito de diseño de los índices, consultas y sincronización de texto es la presencia de una columna de clave exclusiva de texto (o clave principal de columna única) en todas las tablas que se registren para realizar búsquedas de texto. Un índice de texto realiza el seguimiento de las palabras significativas que más se usan y dónde se encuentran.

Por ejemplo, imagine un índice de texto para una tabla llamada DevTools. Un índice de texto puede indicar que la palabra "Microsoft" se encuentra en la palabra número 423 y en la palabra 982 de la columna Abstract para la fila asociada con el ProductID igual a 6. Esta estructura de índices admite una búsqueda eficiente de todos los elementos que contengan palabras indizadas y operaciones de búsqueda avanzadas, como búsquedas de frases y búsquedas de proximidad.

Para impedir que los índices de texto se inunden con palabras que no ayudan en la búsqueda, las palabras innecesarias (vacías de significado), como "un", "y", "es" o "el", se pasan por alto. Por ejemplo, especificar la frase "los productos pedidos durante estos meses de verano" es lo mismo que especificar la frase "productos pedidos durante meses verano". Se devuelven las filas que contengan alguna de las cadenas.

En el directorio `\Mssql7\Ftdata\Sqlserver\Config` se proporcionan listas de palabras que no son relevantes en las búsquedas de muchos idiomas. Este directorio se crea, y los archivos de palabras no relevantes se instalan, cuando se instala SQL Server con la funcionalidad de búsqueda de texto. Los archivos de palabras no relevantes se pueden modificar. Por ejemplo, los administradores del sistema de las empresas de alta tecnología podrían agregar la palabra "sistema" a su lista de palabras no relevantes. (Si modifica un archivo de palabras no relevantes, debe volver a rellenar los catálogos de texto para que los cambios surtan efecto). Consulte la ayuda de SQL-SERVER para conocer los correspondientes ficheros.

Cuando se procesa una consulta de texto, el motor de búsqueda devuelve a SQL Server los valores de clave de las filas que coinciden con los criterios de búsqueda. Imagine una tabla CienciaFicción en la que la columna NúmLibro es la columna de clave principal:

NúmLibro	Escritor	Título
A025	Asimov	Los límites de la fundación
A027	Asimov	Fundación e imperio
C011	Clarke	El fin de la infancia
V109	Verne	La isla misteriosa

Suponga que desea usar una consulta de recuperación de texto para buscar los títulos de los libros que incluyen la palabra Fundación. En este caso, del índice de texto se obtienen los valores A025 y A027. SQL Server usa, a continuación, estas claves y el resto de la información de los campos para responder a la consulta.

Componentes de las consultas de texto de Transact-SQL

SQL Server proporciona estos componentes de Transact-SQL para las consultas de texto:

Predicados de Transact-SQL:

- CONTAINS
- FREETEXT

Los predicados `CONTAINS` y `FREETEXT` se pueden usar en cualquier condición de búsqueda (incluida una cláusula `WHERE`) de una instrucción `SELECT`.

Funciones de conjuntos de filas de Transact-SQL:

- CONTAINSTABLE
- FREETEXTTABLE

Las funciones `CONTAINSTABLE` y `FREETEXTTABLE` se pueden usar en la cláusula `FROM` de una instrucción `SELECT`.

Propiedades de texto de Transact-SQL:

Éstas son algunas de las propiedades que se usan con las consultas de texto y las funciones que se utilizan para obtenerlas:

- La propiedad `IsFullTextEnabled` indica si una base de datos está habilitada para texto y se encuentra disponible mediante la función `DatabaseProperty`.
- La propiedad `TableHasActiveFulltextIndex` indica si una tabla está habilitada para texto y se encuentra disponible mediante la función `ObjectProperty`.
- La propiedad `IsFullTextIndexed` indica si una columna está habilitada para texto y se encuentra disponible mediante la función `ColumnProperty`.
- La propiedad `TableFullTextKeyColumn` proporciona el identificador de la columna de clave exclusiva de texto y se encuentra disponible mediante la función `ObjectProperty`.

Procedimientos de texto almacenados del sistema de Transact-SQL:

- Los procedimientos almacenados que definen los índices de texto e inician el relleno de los índices de texto, como, por ejemplo, `sp_fulltext_catalog`, `sp_fulltext_table` y `sp_fulltext_column`.
- Los procedimientos almacenados que consultan los metadatos de los índices de texto que se han definido mediante los procedimientos almacenados del sistema mencionados anteriormente, como, por ejemplo, `sp_help_fulltext_catalogs`, `sp_help_fulltext_tables`, `sp_help_fulltext_columns`, y una variación de éstos que permite utilizar cursores sobre los conjuntos de resultados devueltos.

Estos procedimientos almacenados se pueden usar en conjunción con la escritura de una consulta. Por ejemplo, puede usarlos para buscar los nombres de las columnas indizadas de texto de una tabla y el identificador de una columna de clave única de texto antes de especificar una consulta.

Funciones de conjunto de filas CONTAINSTABLE y FREETEXTTABLE

Las funciones CONTAINSTABLE y FREETEXTTABLE se usan para especificar las consultas de texto que devuelve la clasificación por porcentaje de aciertos de cada fila. Estas funciones son muy similares a los predicados de texto CONTAINS y FREETEXT, pero se utilizan de forma diferente.

Aunque tanto los predicados de texto como las funciones de conjunto de filas de texto se usan para las consultas de texto y la instrucción TRANSACT-SQL usada para especificar la condición de búsqueda de texto es la misma en los predicados y en las funciones, hay importantes diferencias en la forma en la que éstas se usan:

CONTAINS y FREETEXT devuelven ambos el valor TRUE o FALSE, con lo que normalmente se especifican en la cláusula WHERE de una instrucción SELECT. Sólo se pueden usar para especificar los criterios de selección, que usa SQL SERVER para determinar la pertenencia al conjunto de resultados.

CONTAINSTABLE y FREETEXTTABLE devuelven ambas una tabla de cero, una o más filas, con lo que deben especificarse siempre en la cláusula FROM. Se usan también para especificar los criterios de selección. La tabla devuelta tiene una columna llamada KEY que contiene valores de claves de texto. Cada tabla de texto registrada tiene una columna cuyos valores se garantizan como únicos. Los valores devueltos en la columna KEY de CONTAINSTABLE o FREETEXTTABLE son los valores únicos, procedentes de la tabla de texto registrada, de las filas que coinciden con los criterios de selección en la condición de búsqueda de texto.

Además, la tabla que producen CONTAINSTABLE y FREETEXTTABLE tiene una columna denominada RANK, que contiene valores de 0 a 1000. Estos valores se utilizan para ordenar las filas devueltas de acuerdo al nivel de coincidencia con los criterios de selección.

Las consultas que usan las funciones CONTAINSTABLE y FREETEXTTABLE son más complejas que las que usan los predicados CONTAINS y FREETEXT porque las filas que cumplen los criterios y que son devueltas por las funciones deben ser combinadas explícitamente con las filas de la tabla original de SQL SERVER.

CONTAINSTABLE (T-SQL)

Devuelve una tabla con cero, una o más filas para aquellas columnas de tipos de datos carácter que contengan palabras o frases en forma precisa o "aproximada" (menos precisa), la proximidad de palabras medida como distancia entre ellas, o coincidencias medidas. A CONTAINSTABLE se le puede hacer referencia en una cláusula FROM de una instrucción SELECT como si fuera un nombre de tabla normal.

Las consultas que utilizan CONTAINSTABLE especifican consultas de texto contenido que devuelven un valor de distancia (RANK) por cada fila. La función CONTAINSTABLE utiliza las mismas condiciones de búsqueda que el predicado CONTAINS.

Sintaxis

```
CONTAINSTABLE (tabla, {columna | *}, '<condiciónBúsquedaContenido>')
<condiciónBúsqueda> ::=
{
| <términoGeneración>
| <términoPrefijo>
```

```
| <términoProximidad>
| <términoSimple>
| <términoPeso>
}
| { (<condiciónBúsqueda>)
{AND | AND NOT | OR} <condiciónBúsqueda> [...n]
}
<términoPeso> ::=
ISABOUT
( { {
<términoGeneración>)
| <términoPrefijo>)
| <términoProximidad>)
| <términoSimple>)
}
[WEIGHT (valorPeso)]
} [,...n]
)
<términoGeneración> ::=
FORMSOF (INFLECTIONAL, <términoSimple> [,...n] )
<términoPrefijo> ::=
{ "palabra * " | "frase * " }
<términoProximidad> ::=
{<términoSimple> | <términoPrefijo>}
{ {NEAR | ~} {<términoSimple> | <términoPrefijo>} } [_n]
<términoSimple> ::=
palabra | " frase "
```

Argumentos

tabla
Es el nombre de la tabla que ha sido registrada para búsquedas de texto. `tabla` puede ser el nombre de un objeto de una base de datos de una sola parte o el nombre de un objeto de una base de datos con varias partes. Para obtener más información, consulte Convenciones de sintaxis de Transact-SQL.

columna
Es el nombre de la columna que se va a examinar, que reside en tabla. Las columnas de tipos de datos de cadena de caracteres son columnas válidas para búsquedas de texto.

*****
Especifica que todas las columnas de la tabla que se hayan registrado para búsquedas de texto se deben utilizar en las condiciones de búsqueda.

<condiciónBúsqueda>
Especifica el texto que se va a buscar en columna. En la condición de búsqueda no se puede utilizar variables.

palabra
Es una cadena de caracteres sin espacios ni signos de puntuación.

frase
Es una o varias palabras con espacios entre cada una de ellas.

Nota: Algunos idiomas, como los orientales, pueden tener frases que contengan una o varias palabras sin espacios entre ellas.

<términoPeso>
Especifica que las filas coincidentes (devueltas por la consulta) coincidan con una lista de palabras y frases a las que se asigna opcionalmente un valor de peso.

ISABOUT
Especifica la palabra clave `<términoPeso>`.

WEIGHT (valorPeso)

Especifica el valor de peso como número entre 0,0 y 1,0. Cada componente de `<términoPeso>` puede incluir un `valorPeso`. `valorPeso` es una forma de modificar cómo varias partes de una consulta afectan al valor de distancia asignado a cada fila de la consulta. El peso hace una medida diferente de la distancia de un valor porque todos los componentes de `<términoPeso>` se utilizan para determinar la coincidencia. Se devuelven las filas que contengan una coincidencia con cualquiera de los parámetros `ISABOUT`, aunque no tengan un peso asignado.

AND | AND NOT | OR

Especifica una operación lógica entre dos condiciones de búsqueda. Cuando `<condiciónBúsqueda>` contiene grupos entre paréntesis, dichos grupos entre paréntesis se evalúan primero. Después de evaluar los grupos entre paréntesis, se aplican las reglas siguientes cuando se utilizan estos operadores lógicos con condiciones de búsqueda:

- `NOT` se aplica antes que `AND`.
- `NOT` sólo puede estar a continuación de AND, como en `AND NOT`. No se acepta el operador `OR NOT`. No se puede especificar `NOT` antes del primer término (por ejemplo, `CONTAINS(mycolumn, 'NOT "fraseBuscada" ' )`).
- `AND` se aplica antes que `OR`.
- Los operadores booleanos del mismo tipo (`AND`, `OR`) son asociativos y, por tanto, se pueden aplicar en cualquier orden.

<términoGeneración>

Especifica la coincidencia de palabras cuando los términos simples incluyen variaciones de la palabra original que se busca.

INFLECTIONAL

Especifica que se acepten las coincidencias de las formas plurales y singulares de los nombres y los distintos tiempos verbales. Un `<términoSimple>` dado dentro de un `<términoGeneración>` no coincide con nombres y verbos a la vez.

<términoPrefijo>

Especifica la coincidencia de palabras o frases que comiencen con el texto especificado. Enmarque el prefijo entre comillas dobles (`""`) y un asterisco (`*`) antes de la segunda comilla doble. Coincide todo el texto que comience por el término simple especificado antes del asterisco. El asterisco representa cero, uno o varios caracteres (de la palabra o palabras raíz de la palabra o la frase). Cuando `<términoPrefijo>` es una frase, todas las palabras de dicha frase se consideran prefijos. Por tanto, una consulta que especifique el prefijo `"local wine *"` hace que se devuelvan todas las filas que contengan el texto `"local winery"`, `"locally wined and dined"`, etc.

<términoProximidad>

Especifica la coincidencia de palabras o frases que estén cercanas entre ellas. `<términoProximidad>` opera de forma similar al operador `AND`: ambos requieren que existan varias palabras o frases en la columna examinada. Cuanto más próximas estén las palabras de `<términoProximidad>`, mejor será la coincidencia.

NEAR | ~

Indica que la palabra o frase del lado izquierdo del operador `NEAR` o `~` tiene que estar bastante cerca de la palabra o frase del lado derecho del operador `NEAR` o `~`. Se pueden encadenar varios términos de proximidad, por ejemplo:

```
a NEAR b NEAR c
```

Esto significa que la palabra o frase `a` tiene que estar cerca de la palabra o frase `b`, que, a su vez, tiene que estar cerca de la palabra o frase `c`.

SQL Server mide la distancia entre la palabra o frase izquierda y derecha. Un valor de distancia bajo (por ejemplo, 0) indica una distancia grande entre las dos. Si las palabras o frases especificadas están lejos unas de las otras, satisfacen la condición de la consulta; sin embargo, la consulta tiene un valor de distancia muy bajo (0). Sin embargo, si `<condiciónBúsqueda>` sólo consta de uno o varios términos de proximidad `NEAR`, SQL Server no devuelve filas con un valor de distancia de 0.

<términoSimple>

Especifica la coincidencia con una palabra exacta (uno o varios caracteres sin espacios o signos de puntuación en idiomas con caracteres de un solo byte) o una frase (una o varias palabras consecutivas separadas por espacios y signos de puntuación opcionales en idiomas con caracteres de un solo byte). Ejemplos de términos simples válidos son "blue berry", blueberry y "Microsoft SQL Server". Las frases tienen que ir entre comillas dobles (""). Las palabras de una frase tienen que aparecer en la columna de la base de datos en el mismo orden que el especificado en <condiciónBúsqueda>. La búsqueda de caracteres en la palabra o la frase distingue entre mayúsculas y minúsculas. Las palabras de una sola sílaba (como "un" o "la") de las columnas de texto indizadas no se almacenan en los índices de los textos. Si únicamente se utiliza una de estas palabras en una búsqueda, SQL Server devuelve un mensaje de error indicando que en la consulta sólo hay monosílabos. SQL Server incluye una lista estándar de palabras monosílabas en el directorio \Mssql7\Ftdata\Sqlserver\Config.

Los signos de puntuación se omiten. Por lo tanto, el valor "¿Dónde está mi equipo? El fallo de la búsqueda sería grave." satisface la condición CONTAINS(testing, "fallo del equipo").

n

Es un marcador de posición que indica que se pueden especificar varias condiciones y términos de búsqueda.

Observaciones

CONTAINS no se reconoce como palabra clave si el nivel de compatibilidad es menor de 70.

La tabla devuelta por la funcion CONTAINSTABLE tiene una columna llamada KEY que contiene valores de claves de texto. Todas las tablas con textos indizados tienen una columna cuyos valores se garantizan que son únicos y los valores devueltos en la columna KEY son los valores de claves de textos de las filas que satisfacen los criterios de selección especificados en la condición de búsqueda. La propiedad TableFulltextKeyColumn, obtenida mediante la función OBJECTPROPERTY, proporciona la identidad de esta columna de clave única. Para obtener las filas de la tabla original que desee, especifique una combinación con las filas de CONTAINSTABLE. La forma típica de la cláusula FROM de una instrucción SELECT que utilice CONTAINSTABLE es:

```
SELECT select_list
FROM table AS FT_TBL INNER JOIN
CONTAINSTABLE(table, column, contains_search_condition) AS KEY_TBL
ON FT_TBL.unique_key_column = KEY_TBL.[KEY]
```

La tabla que produce CONTAINSTABLE incluye una columna llamada RANK. La columna RANK es un valor (entre 0 y 1000) que para cada fila indica lo bien que cada una de ellas satisface los criterios de selección. Este valor de distancia se suele utilizar en las instrucciones SELECT de una de estas maneras:

- En la cláusula ORDER BY, para devolver las filas de mayor valor al principio.
- En la lista de selección, para ver el valor de distancia asignado a cada fila.
- En la cláusula WHERE, para filtrar las filas con valores de distancia bajos.

CONTAINSTABLE no se reconoce como palabra clave si el nivel de compatibilidad es menor de 70. Para obtener más información, consulte sp_dbcmptlevel.

Ejemplos

A. Devolver valores de distancia mediante CONTAINSTABLE
Este ejemplo busca todos los nombres de productos que contengan las palabras "breads", "fish" o "beers", y los distintos pesos asignados a cada palabra. Por cada fila devuelta que cumpla los criterios de la búsqueda, se muestra la precisión relativa (valor de distancia) de la coincidencia. Además, las filas de mayor valor de distancia se devuelven primero.

```
USE Northwind
GO
SELECT FT_TBL.CategoryName, FT_TBL.Description, KEY_TBL.RANK
FROM Categories AS FT_TBL INNER JOIN
CONTAINSTABLE(Categories, Description,
'ISABOUT (breads weight (.8),
fish weight (.4), beers weight (.2) )' ) AS KEY_TBL
ON FT_TBL.CategoryID = KEY_TBL.[KEY]
ORDER BY KEY_TBL.RANK DESC
GO
```

B. Devolver valores de distancia mayores que uno especificado mediante CONTAINSTABLE
Este ejemplo devuelve la descripción y el nombre de la categoría de todas las categorías de alimentos en las que la columna Description contenga las palabras "sweet" y "savory" cerca de la palabra "sauces" o de la palabra "candies". Todas las filas cuya categoría sea "Seafood" no se devuelven. Sólo se devuelven las filas cuyo grado de coincidencia sea igual o superior a 2.

```
USE Northwind
GO
SELECT FT_TBL.Description,
FT_TBL.CategoryName,
KEY_TBL.RANK
FROM Categories AS FT_TBL INNER JOIN
CONTAINSTABLE (Categories, Description,
'("sweet and savory" NEAR sauces) OR
("sweet and savory" NEAR candies)'
) AS KEY_TBL
ON FT_TBL.CategoryID = KEY_TBL.[KEY]
WHERE KEY_TBL.RANK > 2
AND FT_TBL.CategoryName <> 'Seafood'
ORDER BY KEY_TBL.RANK DESC
```

C. Utilizar CONTAINS con <términoSimple>
Este ejemplo busca todos los productos cuyo precio sea $15,00 que contengan la palabra "bottles".

```
USE Northwind
GO
SELECT ProductName
FROM Products
WHERE UnitPrice = 15.00
AND CONTAINS(QuantityPerUnit, 'bottles')
GO
```

D. Utilizar CONTAINS y una frase en <términoSimple>
Este ejemplo devuelve todos los productos que contengan la frase "sasquatch ale" o "steeleye stout".

```
USE Northwind
GO
SELECT ProductName
FROM Products
WHERE CONTAINS(ProductName, ' "Sasquatch ale" OR "steeleye stout" ')
```

```
GO
```

E. Utilizar CONTAINS con <términoPrefijo>
Este ejemplo devuelve todos los nombres de productos que tengan al menos una palabra que empiece por el prefijo "choc" en la columna ProductName.

```
USE Northwind
GO
SELECT ProductName
FROM Products
WHERE CONTAINS(ProductName, ' "choc*" ')
GO
```

F. Utilizar CONTAINS y OR con <términoPrefijo>
Este ejemplo devuelve todas las descripciones de categorías que contengan las cadenas "sea" o "bread".

```
USE Northwind
SELECT CategoryName
FROM Categories
WHERE CONTAINS(Description, '"sea*" OR "bread*"')
GO
```

G. Utilizar CONTAINS con <términoProximidad>
Este ejemplo devuelve todos los nombres de los productos que tengan la palabra "Boysenberry" cerca de la palabra "spread".

```
USE Northwind
GO
SELECT ProductName
FROM Products
WHERE CONTAINS(ProductName, 'spread NEAR Boysenberry')
GO
```

H. Utilizar CONTAINS con <términoGeneración>
Este ejemplo busca todos los productos que tengan palabras derivadas de "dry": "dried", "drying", etc.

```
USE Northwind
GO
SELECT ProductName
FROM Products
WHERE CONTAINS(ProductName, ' FORMSOF (INFLECTIONAL, dry) ')
GO
```

I. Utilizar CONTAINS con <términoPeso>
Este ejemplo busca todos los nombres de productos que contengan las palabras "spread", "sauces" o "relishes", y los distintos pesos asignados a cada palabra.

```
USE Northwind
GO
SELECT CategoryName, Description
FROM Categories
WHERE CONTAINS(Description, 'ISABOUT (spread weight (.8),
sauces weight (.4), relishes weight (.2) )' )
GO
```

FREETEXTTABLE

Devuelve una tabla de cero, una o varias filas cuyas columnas contienen datos de tipo carácter cuyos valores coinciden con el significado, no literalmente, con el texto especificado en cadenaTexto. Se puede hacer referencia a FREETEXTTABLE en las cláusula FROM de las instrucciones SELECT como a otro nombre de tabla normal.

Las consultas que utilizan FREETEXTTABLE especifican consultas de texto que devuelven el valor de coincidencia (RANK) de cada fila.

Sintaxis

```
FREETEXTTABLE (tabla, {columna | *}, 'cadenaTexto')
```

Argumentos

tabla
Es el nombre de la tabla que se ha marcado para búsquedas de texto. tabla puede ser el nombre de un objeto de una base de datos de una sola parte o el nombre de un objeto de una base de datos con varias partes.

columna
Es el nombre de la columna de tabla en la que se va a buscar. Las columnas cuyos datos sean del tipo de cadena de caracteres son columnas válidas para buscar texto.

*****
Especifica que todas las columnas que hayan sido registradas para la búsqueda de texto se tienen que utilizar para buscar la cadenaTexto dada.

cadenaTexto
Es el texto que se va a buscar en la columna especificada. No se pueden utilizar variables.

Observaciones

FREETEXTTABLE utiliza las mismas condiciones de búsqueda que el predicado FREETEXT. Al igual que en CONTAINSTABLE, la tabla devuelta tiene columnas llamadas KEY y RANK, a las que se hace referencia en la consulta para obtener las filas apropiadas y utilizar los valores de distancia. FREETEXTTABLE no se reconoce como palabra clave si el nivel de compatibilidad es menor que 70.

Ejemplos

En este ejemplo se devuelve el nombre y la descripción de todas las categorías relacionadas con _sweet_, _candy_, _bread_, _dry_ y _meat_.

```
USE Northwind
SELECT FT_TBL.CategoryName,
FT_TBL.Description,
KEY_TBL.RANK
FROM Categories AS FT_TBL INNER JOIN
FREETEXTTABLE(Categories, Description,
'sweetest candy bread and dry meat') AS KEY_TBL
ON FT_TBL.CategoryID = KEY_TBL.[KEY]
GO
```

Utilizar el predicado CONTAINS

Puede usar el predicado CONTAINS para buscar una determinada frase en una base de datos. Por supuesto, dicha consulta puede escribirse con el predicado LIKE. Sin embargo, algunas formas de CONTAINS proporcionan mayor variedad de consultas de texto que la que se puede

obtener con `LIKE`. Además, al contrario que cuando se utiliza el predicado `LIKE`, una búsqueda con `CONTAINS` no distingue entre mayúsculas y minúsculas.

Nota.. Las consultas de búsqueda de texto se comportan de forma que no distinguen entre mayúsculas y minúsculas en aquellos idiomas (mayoritariamente los latinos) en los que tiene sentido distinguir entre mayúsculas y minúsculas. Sin embargo, en japonés, hay muchas ortografías fonéticas en las que el concepto de normalización ortográfica implica no distinguir las mayúsculas de las minúsculas (por ejemplo, las letras kana no tienen mayúsculas y minúsculas). Este tipo de normalización ortográfica no se admite.

Suponga que desea buscar en la base de datos Northwind la frase "bean curd". Si usa el predicado `CONTAINS`, ésta es una consulta bastante fácil.

```
USE Northwind USE Northwind
GO
SELECT Description
FROM Categories
WHERE Description LIKE '%bean curd%'
GO
```

O, con `CONTAINS`:

```
USE Northwind
GO
SELECT Description
FROM Categories
WHERE CONTAINS(Description, ' "bean curd" ')
GO
```

El predicado `CONTAINS` usa una notación funcional en la que el primer parámetro es el nombre de la columna que se está buscando y el segundo parámetro es una condición de búsqueda de texto. La condición de búsqueda, en este caso "bean curd", puede ser bastante compleja y está formada por uno o más elementos, que se describen posteriormente.

El predicado `CONTAINS` admite una sintaxis compleja para buscar en las columnas basadas en caracteres:

- Una o más palabras y frases específicas (términos simples). Una palabra está compuesta por uno o más caracteres sin espacios ni signos de puntuación. Una frase válida consta de varias palabras con espacios y con o sin signos de puntuación entre ellas. Por ejemplo, croissant es una palabra y café au lait es una frase. Las palabras y frases como éstas se llaman términos simples.
- Forma no flexionada de una palabra determinada (término de generación). Por ejemplo, buscar la forma no flexionada de la palabra "conducir". Si hay varias filas en la tabla que incluyen las palabras "conducir", "conduce", "condujo", "conduciendo" y "conducido", todas estarían en el conjunto de resultados porque cada una de estas palabras se puede generar de forma inflexiva a partir de la palabra "conducir".
- Una palabra o frase en la que las palabras empiezan con un texto determinado (término prefijo). En el caso de una frase, cada palabra de la frase se considera un prefijo. Por ejemplo, el término "tran* auto" coincide con "transmisión automática" y "transductor de automóvil".
- Palabras o frases que usan valores ponderados (término ponderado). Por ejemplo, podría desear encontrar una palabra que tuviera un peso designado superior a otra palabra. Devuelve resultados de consulta clasificados.
- Una palabra o frase que esté cerca de otra palabra o frase (término de proximidad). Por ejemplo, podría desear encontrar las filas en las que la palabra "hielo" aparece cerca de la palabra "hockey" o en las que la frase "patinaje sobre hielo" se encuentra próxima a la frase "hockey sobre hielo".

Un predicado CONTAINS puede combinar varios de estos términos si usa AND y OR, por ejemplo, podría buscar todas las filas con "leche" y "café al estilo de Toledo" en la misma columna de base datos habilitada para texto . Además, los términos se pueden negar con el uso de AND NOT, por ejemplo, "pastel AND NOT queso de untar".

Cuando use CONTAINS, recuerde que SQL Server rechaza las palabras vacías de los criterios de búsqueda. Las palabras irrelevantes son aquellas como "un", "y", "es" o "el", que aparecen con frecuencia pero que, en realidad, no ayudan en la búsqueda de un texto determinado.

Utilizar el predicado FREETEXT

Con un predicado FREETEXT, puede escribir cualquier conjunto de palabras o frases, e incluso una frase completa. El motor de consultas de texto examina este texto, identifica todas las palabras y frases de nombres significativas y construye internamente una consulta con esos términos. En este ejemplo se usa un predicado FREETEXT en una columna llamada description.

```
FREETEXT (description, ' "The Fulton County Grand Jury said Friday
an investigation of Atlanta's recent primary election produced no
evidence that any
irregularities took place." ')
```

El motor de búsqueda identifica palabras y frases nominales tales como las siguientes:

Palabras:
Fulton, county, grand, jury, Friday, investigation, Atlanta, recent, primary, election, produce, evidence, irregularities

Frases:
Fulton county grand jury, primary election, grand jury, Atlanta's recent primary election

Las palabras y frases de la cadena FREETEXT (y sus variaciones generadas de forma inflexiva) se combinan internamente en una consulta, ponderada para clasificarla adecuadamente y, a continuación, se realiza la búsqueda real.

Funciones de conjunto de filas CONTAINSTABLE y FREETEXTTABLE

Las funciones CONTAINSTABLE y FREETEXTTABLE se usan para especificar las consultas de texto que devuelve la clasificación por porcentaje de aciertos de cada fila. Estas funciones son muy similares a los predicados de texto CONTAINS y FREETEXT, pero se utilizan de forma diferente.

Los predicados de texto de las funciones

Aunque tanto los predicados de texto como las funciones de conjunto de filas de texto se usan para las consultas de texto y la instrucción TRANSACT-SQL usada para especificar la condición de búsqueda de texto es la misma en los predicados y en las funciones, hay importantes diferencias en la forma en la que éstas se usan:

- CONTAINS y FREETEXT devuelven ambos el valor TRUE o FALSE, con lo que normalmente se especifican en la cláusula WHERE de una instrucción SELECT.
- CONTAINSTABLE y FREETEXTTABLE devuelven ambas una tabla de cero, una o más filas, con lo que deben especificarse siempre en la cláusula FROM.
- CONTAINS y FREETEXT sólo se pueden usar para especificar los criterios de selección, que usa SQL SERVER para determinar la pertenencia al conjunto de resultados.
- CONTAINSTABLE y FREETEXTTABLE se usan también para especificar los criterios de selección. La tabla devuelta tiene una columna llamada KEY que contiene valores de claves de texto. Cada tabla de texto registrada tiene una columna cuyos valores se garantizan como únicos. Los valores devueltos en la columna KEY de CONTAINSTABLE o FREETEXTTABLE son los valores únicos, procedentes de la tabla de texto registrada,

de las filas que coinciden con los criterios de selección en la condición de búsqueda de texto.

- Además, la tabla que producen CONTAINSTABLE y FREETEXTTABLE tiene una columna denominada RANK, que contiene valores de 0 a 1000. Estos valores se utilizan para ordenar las filas devueltas de acuerdo al nivel de coincidencia con los criterios de selección.

Las consultas que usan las funciones CONTAINSTABLE y FREETEXTTABLE son más complejas que las que usan los predicados CONTAINS y FREETEXT porque las filas que cumplen los criterios y que son devueltas por las funciones deben ser combinadas explícitamente con las filas de la tabla original de SQL SERVER.

Este ejemplo devuelve la descripción y el nombre de categoría de todas las categorías de alimentos en las que la columna Description contenga las palabras "sweet and savory" cerca de la palabra "sauces" o de la palabra "candies". Todas las filas cuyo nombre de categoría sea "Seafood" no se devuelven. Sólo se devuelven las filas cuyo valor de distancia sea igual o superior a 2.

```
USE Northwind
GO
SELECT FT_TBL.Description, FT_TBL.CategoryName, KEY_TBL.RANK
FROM Categories AS FT_TBL INNER JOIN
CONTAINSTABLE (Categories, Description,
'("sweet and savory" NEAR sauces) OR
("sweet and savory" NEAR candies)') AS KEY_TBL
ON FT_TBL.CategoryID = KEY_TBL.[KEY]
WHERE KEY_TBL.RANK > 2 AND FT_TBL.CategoryName <> 'Seafood'
ORDER BY KEY_TBL.RANK DESC
```

Este ejemplo devuelve la descripción y el nombre de categoría de las 10 categorías superiores de alimentos donde la columna Description contenga las palabras "sweet and savory" cerca de la palabra "sauces" o de la palabra "candies".

```
SELECT FT_TBL.Description, FT_TBL.CategoryName, KEY_TBL.RANK
FROM Categories AS FT_TBL INNER JOIN
CONTAINSTABLE (Categories, Description,
'("sweet and savory" NEAR sauces) OR
("sweet and savory" NEAR candies)', 10) AS KEY_TBL
ON FT_TBL.CategoryID = KEY_TBL.[KEY]
```

Comparación entre CONTAINSTABLE y CONTAINS

La función CONTAINSTABLE y el predicado CONTAINS utilizan condiciones de búsqueda similares.

Sin embargo, en CONTAINSTABLE se especifica la tabla en la que tendrá lugar la búsqueda de texto, la columna (o todas las columnas) de la tabla en las que se buscará y la condición de búsqueda. Un cuarto parámetro, opcional, hace posible que el usuario indique que se devuelva sólo el número más alto especificado de coincidencias. Para obtener más información, consulte la sección Limitar los conjuntos de resultados.

CONTAINSTABLE devuelve una tabla que incluye una columna denominada RANK. Esta columna RANK contiene un valor para cada fila que indica el grado de coincidencia de cada fila con los criterios de selección.

En esta consulta se especifica la utilización de CONTAINSTABLE para devolver un valor de clasificación por cada fila.

```
USE Northwind
GO
SELECT K.RANK, CompanyName, ContactName, Address
FROM Customers AS C
INNER JOIN
CONTAINSTABLE(Customers,Address,
'ISABOUT ("des*", Rue WEIGHT(0.5), Bouchers WEIGHT(0.9))') AS K
ON C.CustomerID = K.[KEY]
```

Comparación entre FREETEXTTABLE y FREETEXT

En la consulta siguiente se amplía una consulta FREETEXTTABLE para que devuelva primero las filas con clasificación superior y agregue la clasificación de cada fila a la lista de selección. Para especificar la consulta, debe saber que CategoryID es la columna de clave única de la tabla Categories.

```
USE Northwind
GO
SELECT KEY_TBL.RANK, FT_TBL.Description
FROM Categories AS FT_TBL
INNER JOIN
FREETEXTTABLE(Categories, Description,
'How can I make my own beers and ales?') AS KEY_TBL
ON FT_TBL.CategoryID = KEY_TBL.[KEY]
ORDER BY KEY_TBL.RANK DESC
GO
```

La única diferencia en la sintaxis de FREETEXTTABLE y FREETEXT es la inserción del nombre de la tabla como el primer parámetro.

Esto es una ampliación de la misma consulta que sólo devuelve las filas con un valor de clasificación de 10 o superior:

```
USE Northwind
GO
SELECT KEY_TBL.RANK, FT_TBL.Description
FROM Categories FT_TBL
INNER JOIN
FREETEXTTABLE (Categories, Description,
'How can I make my own beers and ales?') AS KEY_TBL
ON FT_TBL.CategoryID = KEY_TBL.[KEY]
WHERE KEY_TBL.RANK >= 10
ORDER BY KEY_TBL.RANK DESC
GO
```

Identificación del nombre de la columna de la clave única

Las consultas que usan funciones que toman valores de conjuntos de filas son complicadas porque es necesario saber el nombre de la columna de clave exclusiva. Cada tabla habilitada para texto tiene la propiedad TableFulltextKeyColumn que contiene el número de ID de la columna que ha sido seleccionada para tener filas únicas en la tabla. En este ejemplo se muestra cómo se puede obtener el nombre de la columna de clave y usarse en la programación.

```
USE Northwind
GO
DECLARE @key_column sysname
SET @key_column = Col_Name(Object_Id('Categories'),
ObjectProperty(Object_id('Categories'),
```

```
'TableFulltextKeyColumn')
)
print @key_column
EXECUTE ('SELECT Description, KEY_TBL.RANK
FROM Categories FT_TBL
INNER JOIN
FreetextTable (Categories, Description,
''How can I make my own beers and ales?'') AS KEY_TBL
ON FT_TBL.'
+ @key_column
+' = KEY_TBL.[KEY]
WHERE KEY_TBL.RANK >= 10
ORDER BY KEY_TBL.RANK DESC
')
GO
```

Puede evitar la complejidad de la utilización de CONTAINSTABLE y FREETEXTTABLE si escribe procedimientos almacenados que acepten unos cuantos supuestos acerca de la consulta y, a continuación, creen y ejecuten la consulta adecuada. A continuación se muestra un procedimiento simplificado que emite una consulta FREETEXTTABLE. La tabla muestra los parámetros del procedimiento (todas las entradas).

Parámetros	Opcional	Descripción
@additional_predicates	Opcional	Si hay algún predicado adicional, éste se agrega con AND detrás del predicado FREETEXT. KEY_TBL.RANK se puede usar en expresiones.
@freetext_column	SI	
@freetext_search	SI	Condición de Búsqueda
@from_table	SI	
@order_by_list	Opcional	KEY_TBL.RANK puede ser una de las columnas especificadas.
@select_list	SI	KEY_TBL.RANK puede ser una de las columnas especificadas.

El código del procedimiento es el siguiente:

```
CREATE PROCEDURE freetext_rank_proc
@select_list nvarchar(1000),
@from_table nvarchar(517),
@freetext_column sysname,
@freetext_search nvarchar(1000),
@additional_predicates nvarchar(500) = '',
@order_by_list nvarchar(500) = ''
AS
BEGIN
DECLARE @table_id integer,
@unique_key_col_name sysname,
@add_pred_var nvarchar(510),
@order_by_var nvarchar(510)

-- Get the name of the unique key column for this table.
SET @table_id = Object_Id(@from_table)
SET @unique_key_col_name =
Col_Name( @table_id,
ObjectProperty(@table_id, 'TableFullTextKeyColumn') )

-- If there is an additional_predicate, put AND() around it.
IF @additional_predicates <> ''
SET @add_pred_var = 'AND (' + @additional_predicates + ')'
ELSE
```

```
SET @add_pred_var = ''

-- Insert ORDER BY, if needed.
IF @order_by_list <> ''
SET @order_by_var = 'ORDER BY ' + @order_by_var
ELSE
SET @order_by_var = ''

-- Execute the SELECT statement.
EXECUTE ( 'SELECT '
+ @select_list
+ ' FROM '
+ @from_table
+ ' AS FT_TBL, FreetextTable('
+ @from_table
+ ','
+ @freetext_column
+ ','''
+ @freetext_search
+ ''') AS KEY_TBL '
+ 'WHERE FT_TBL.'
+ @unique_key_col_name
+ ' = KEY_TBL.[KEY] '
+ @add_pred_var
+ ' '
+ @order_by_var
)
END
```

Este procedimiento se puede usar para emitir la consulta:

```
USE Northwind
GO
EXECUTE freetext_rank_proc
'Description, KEY_TBL.RANK', -- Select list
'Categories', -- From
'Description', -- Column
'How can I make my own beers and ales?', -- Freetext search
'KEY_TBL.RANK >= 10', -- Additional predicate
'KEY_TBL.RANK DESC' -- Order by
GO
```

Limitar los conjuntos de resultados

En muchas consultas de texto, el número de elementos que coinciden con la condición de búsqueda es muy grande. Para evitar que las consultas devuelvan demasiadas coincidencias, utilice el argumento opcional, top_n_by_rank, en `CONTAINSTABLE` y `FREETEXTTABLE` para especificar el número de coincidencias, ordenadas, que desea que se devuelvan.

Con esta información, SQL SERVER ordena las coincidencias y devuelve sólo hasta completar el número especificado. Esta opción puede aumentar significativamente el rendimiento. Por ejemplo, una consulta que por lo general devolvería 100.000 filas de una tabla de 1 millón se procesará de forma más rápida si sólo se piden las 100 primeras filas.

Si sólo se desea que se devuelvan las 3 coincidencias mayores del ejemplo anterior, mediante `CONTAINSTABLE`, la consulta tendrá esta forma:

```
USE Northwind
GO
```

```
SELECT K.RANK, CompanyName, ContactName, Address
FROM Customers AS C
INNER JOIN
CONTAINSTABLE(Customers,Address, 'ISABOUT ("des*",
Rue WEIGHT(0.5),
Bouchers WEIGHT(0.9))', 3) AS K
ON C.CustomerID = K.[KEY]
```

Buscar palabras o frases con valores ponderados (término ponderado)

Puede buscar palabras o frases y especificar un valor ponderado. El peso, un número entre 0,0 y 1,0, indica el grado de importancia de cada palabra o frase en un conjunto de palabras y frases. El valor 0,0 es el peso más pequeño disponible, y el valor 1,0 es el peso más grande. Por ejemplo, en esta consulta se buscan todas las direcciones de los clientes, con valores ponderados, en los que cualquier texto que comience con la cadena "des" esté cerca de Rue o Bouchers. SQL SERVER da una clasificación superior a aquellas filas que contienen la mayor cantidad de palabras especificadas. Por tanto, SQL SERVER da una clasificación superior a una fila que contiene des Rue Bouchers que a una fila que contiene des Rue.

```
USE Northwind
GO
SELECT CompanyName, ContactName, Address
FROM Customers
WHERE CONTAINS(Address, 'ISABOUT ("*des*",
Rue WEIGHT(0.5),
Bouchers WEIGHT(0.9)
) ' )
GO
```

Un término ponderado se puede usar en conjunción con cualquiera de los otros cuatro tipos de términos.

Combinar predicados de texto con otros predicados de TRANSACT-SQL

Los predicados `CONTAINS` y `FREETEXT` se pueden combinar con el resto de predicados de TRANSACT-SQL, como, por ejemplo, `LIKE` y `BETWEEN`; también se pueden usar en una subconsulta. En este ejemplo se buscan descripciones cuya categoría no sea Seafood y que contengan la palabra "sauces" y la palabra "seasonings".

```
USE Northwind
GO
SELECT Description
FROM Categories
WHERE CategoryName <> 'Seafood' AND
CONTAINS(Description, ' sauces AND seasonings ')
GO
```

En la siguiente consulta se usa `CONTAINS` dentro de una subconsulta. Con la base de datos pubs, la consulta obtiene el valor del título de todos los libros de la tabla titles del publicador que se encuentra próximo al platillo volante de Moonbeam, Ontario. (Esta información acerca del publicador se encuentra en la columna pr_info de la tabla pub_info y sólo hay uno de estos publicadores.)

```
USE pubs
GO
-- Add some interesting rows to some tables.
INSERT INTO publishers
```

```
VALUES ('9970', 'Penumbra Press', 'Moonbeam', 'ON', 'Canada')
INSERT INTO pub_info (pub_id, pr_info)
VALUES ('9970',
'Penumbra press is located in the small village of Moonbeam. Moonbeam
is well known as the flying saucer capital of Ontario. You will often
find one
or more flying saucers docked close to the tourist information centre
on the
north side of highway 11.')

INSERT INTO titles
VALUES ('FP0001', 'Games of the World', 'crafts', '9970', 9.85,
0.00, 20, 213, 'A crafts book! A sports book! A history book! The fun
and
excitement of a world at play - beautifully described and lavishly
illustrated',
'1977/09/15')
GO
-- Given the full-text catalog for these tables is pubs_ft_ctlg,
-- repopulate it so new rows are included in the full-text indexes.
sp_fulltext_catalog 'pubs_ft_ctlg', 'start_full'
WAITFOR DELAY '00:00:30' -- Wait 30 seconds for population.
GO
-- Issue the query.
SELECT T.title, P.pub_name
FROM publishers P,
titles T
WHERE P.pub_id = T.pub_id
AND P.pub_id = (SELECT pub_id
FROM pub_info
WHERE CONTAINS (pr_info,
' moonbeam AND
ontario AND
"flying saucer" '))
GO
```

Utilizar predicados de texto para consultar columnas de tipo IMAGE

Los predicados `CONTAINS` y `FREETEXT` pueden utilizarse para buscar columnas `IMAGE` indizadas.

En una sola columna `IMAGE` es posible almacenar muchos tipos de documentos. SQL SERVER admite ciertos tipos de documento y proporciona filtros para los mismos. Esta versión proporciona filtros para documentos de Office, archivos de texto y archivos HTML.

Cuando una columna `IMAGE` participa en un índice de texto, el servicio de texto comprueba las extensiones de los documentos de la columna `IMAGE` y aplica el filtro correspondiente, para interpretar los datos binarios y extraer la información de texto necesaria para la indización y la consulta.

Así, cuando configure la indización de texto sobre una columna `IMAGE` de una tabla, deberá crear una columna separada para que contenga la información relativa al documento. Esta columna de tipo debe ser de cualquier tipo de datos basado en caracteres y contendrá la extensión del archivo, como por ejemplo DOC para los documentos de Microsoft Word. Si el tipo de columna es NULL, el servicio de texto asumirá que el documento es un archivo de texto.

- En el Asistente para indización de texto, si selecciona una columna `IMAGE` para la indización, deberá especificar también una Columna de enlace para que contenga el tipo de documento.

- El procedimiento almacenado sp_fulltext_column acepta también un argumento para la columna que contendrá los tipos de documento.
- El procedimiento almacenado sp_help_fulltext_columns devuelve también el nombre de columna y el Id. de columna de la columna de tipo de documento.

Una vez indizada, podrá consultar la columna `IMAGE` como lo haría con cualquier otra columna de la tabla, mediante los predicados `CONTAINS` y `FREETEXT`.

Acceso a Bases de Datos Externas (Access)

Para el acceso a bases de datos externas se utiliza la cláusula IN. Se puede acceder a base de datos dBase, Paradox o Btrieve. Esta cláusula sólo permite la conexión de una base de datos externa a la vez. Una base de datos externa es una base de datos que no sea la activa. Aunque para mejorar los rendimientos es mejor adjuntarlas a la base de datos actual y trabajar con ellas.

Para especificar una base de datos que no pertenece a Access Basic, se agrega un punto y coma (;) al nombre y se encierra entre comillas simples. También puede utilizar la palabra reservada DATABASE para especificar la base de datos externa. Por ejemplo, las líneas siguientes especifican la misma tabla:

```
FROM Tabla IN  '[dBASE IV; DATABASE=C:\DBASE\DATOS\VENTAS;]';
FROM Tabla IN 'C:\DBASE\DATOS\VENTAS' 'dBASE IV;'
```

Acceso a una base de datos externa de Microsoft Access:

```
SELECT IDCliente FROM Clientes IN MISDATOS.MDB WHERE IDCliente Like
'A*';
```

En donde MISDATOS.MDB es el nombre de una base de datos de Microsoft Access que contiene la tabla Clientes.

Acceso a una base de datos externa de dBASE III o IV:

```
SELECT IDCliente FROM Clientes IN 'C:\DBASE\DATOS\VENTAS' 'dBASE IV';
WHERE IDCliente Like 'A*';
```

Para recuperar datos de una tabla de dBASE III+ hay que utilizar `'dBASE III+;'` en lugar de `'dBASE IV;'`.

Acceso a una base de datos de Paradox 3.x o 4.x:

```
SELECT IDCliente FROM Clientes IN 'C:\PARADOX\DATOS\VENTAS'
'Paradox 4.x;' WHERE IDCliente Like 'A*';
```

Para recuperar datos de una tabla de Paradox versión 3.x, hay que sustituir `'Paradox 4.x;'` por `'Paradox 3.x;'`.

Acceso a una base de datos de Btrieve:

```
SELECT IDCliente FROM Clientes IN 'C:\BTRIEVE\DATOS\VENTAS\FILE.DDF'
'Btrieve;' WHERE IDCliente Like 'A*';
```

C:\BTRIEVE\DATOS\VENTAS\FILE.DDF es la ruta de acceso y nombre de archivo del archivo de definición de datos de Btrieve.

- Parámetros (Access)

Parámetros (Access)

Las consultas con parámetros son aquellas cuyas condiciones de búsqueda se definen mediante parámetros. Si se ejecutan directamente desde la base de datos donde han sido definidas aparecerá un mensaje solicitando el valor de cada uno de los parámetros. Si deseamos ejecutarlas desde una aplicación hay que asignar primero el valor de los parámetros y después ejecutarlas. Su sintaxis es la siguiente:

```
PARAMETERS nombre1 tipo1, nombre2 tipo2, ... , nombreN tipoN Consulta
```

En donde:

Parte	Descripción
nombre	Es el nombre del parámetro
tipo	Es el tipo de datos del parámetro
consulta	Una consulta SQL

Puede utilizar nombre pero no tipo de datos en una cláusula `WHERE` o `HAVING`.

```
PARAMETERS Precio_Minimo Currency, Fecha_Inicio DateTime;
SELECT IDPedido, Cantidad FROM Pedidos WHERE Precio > Precio_Minimo
AND FechaPedido >= Fecha_Inicio;
```

Omitir los permisos de acceso (Access)

En entornos de bases de datos con permisos de seguridad para grupos de trabajo se puede utilizar la cláusula `WITH OWNERACCESS OPTION` para que el usuario actual adquiera los derechos de propietario a la hora de ejecutar la consulta. Su sintaxis es:

```
instrucción sql WITH OWNERACCESS OPTION
SELECT Apellido, Nombre, Salario FROM Empleados ORDER BY Apellido
WITH OWNERACCESS OPTION;
```

Esta opción requiere que esté declarado el acceso al fichero de grupo de trabajo (generalmente system.mda ó system .mdw) de la base de datos actual.

- Cláusula Procedure (Access)

Cláusula Procedure (Access)

Esta cláusula es poco usual y se utiliza para crear una consulta a la misma vez que se ejecuta, opcionalmente define los parámetros de la misma. Su sintaxis es la siguiente:

```
PROCEDURE NombreConsulta Parámetro1 tipo1, .... , ParámetroN tipon
ConsultaSQL
```

En donde:

Parte	Descripción
NombreConsulta	Es el nombre con se guardará la consulta en la base de datos.
Parámetro	Es el nombre de parámetro o de los parámetros de dicha consulta.
tipo	Es el tipo de datos del parámetro.
ConsultaSQL	Es la consulta que se desea grabar y ejecutar.

```
PROCEDURE Lista_Categorias; SELECT DISTINCTROW Nombre_Categoria,
ID_Categoría FROM Categorias ORDER BY Nombre_Categoria;
```

Asigna el nombre Lista_de_categorías a la consulta y la ejecuta.

```
PROCEDURE Resumen Fecha_Inicio DateTime, Fecha_Final DateTime; SELECT
DISTINCTROW Fecha_Envio, ID_Pedido, Importe_Pedido,
Format(Fecha_Envio, "yyyy")
AS Año FROM Pedidos WHERE Fecha_Envio Between Fecha_Inicio And
Fecha_Final;
```

Asigna el nombre Resumen a la consulta e incluye dos parámetros.

Problemas resueltos

Búsqueda de registros duplicados

Para generar este tipo de consultas lo más sencillo es utilizar el asistente de consultas de Access, editar la sentencia SQL de la consulta y pegarla en nuestro código. No obstante este tipo de consulta se consigue de la siguiente forma:

```
SELECT DISTINCTROW Lista de Campos a Visualizar FROM Tabla
WHERE CampoDeBusqueda In (SELECT CampoDeBusqueda FROM Tabla As
psudónimo
GROUP BY CampoDeBusqueda HAVING Count(*)>1 ) ORDER BY CampoDeBusqueda;
```

Un caso práctico, si deseamos localizar aquellos empleados con igual nombre y visualizar su código correspondiente, la consulta sería la siguiente:

```
SELECT DISTINCTROW Empleados.Nombre, Empleados.IdEmpleado
FROM Empleados WHERE Empleados.Nombre In (SELECT Nombre FROM
Empleados As Tmp GROUP BY Nombre HAVING Count(*)>1)
ORDER BY Empleados.Nombre;
```

Búsqueda de registros no relacionados

Este tipo de consulta se emplea en situaciones tales como saber qué productos no se han vendido en un determinado periodo de tiempo.

```
SELECT DISTINCTROW Productos.IdProducto, Productos.Nombre FROM
Productos LEFT JOIN Pedidos ON Productos.IdProducto =
Pedidos.IdProduct WHERE (Pedidos.IdProducto Is Null) AND
(Pedidos.Fecha Between #01-01-98# And #01-30-98#);
```

La sintaxis es sencilla, se trata de realizar una unión interna entre dos tablas seleccionadas mediante un `LEFT JOIN`, estableciendo como condición que el campo relacionado de la segunda sea Null.

Optimizar consultas

El lenguaje SQL es no procedimental, es decir, en las sentencias se indica que queremos conseguir y no como lo tiene que hacer el interprete para conseguirlo. Esto es pura teoría, pues en la práctica a todos los gestores de SQL hay que especificar sus propios truquitos para optimizar el rendimiento.

Por tanto, muchas veces no basta con especificar una sentencia SQL correcta, sino que además, hay que indicarle como tiene que hacerlo si queremos que el tiempo de respuesta sea el mínimo. En este apartado veremos como mejorar el tiempo de respuesta de nuestro interprete ante unas determinadas situaciones:

Diseño de las tablas

- Normaliza las tablas, al menos hasta la tercera forma normal, para asegurar que no hay duplicidad de datos y se aprovecha al máximo el almacenamiento en las tablas. Si hay que desnormalizar alguna tabla piensa en la ocupación y en el rendimiento antes de proceder.
- Los primeros campos de cada tabla deben ser aquellos campos requeridos y dentro de los requeridos primero se definen los de longitud fija y después los de longitud variable.
- Ajusta al máximo el tamaño de los campos para no desperdiciar espacio.
- Es muy habitual dejar un campo de texto para observaciones en las tablas. Si este campo se va a utilizar con poca frecuencia o si se ha definido con gran tamaño, por si acaso, es mejor crear una nueva tabla que contenga la clave primaria de la primera y el campo para observaciones.

Gestión y elección de los índices

Los índices son campos elegidos arbitrariamente por el constructor de la base de datos que permiten la búsqueda a partir de dicho campo a una velocidad notablemente superior. Sin embargo, esta ventaja se ve contrarrestada por el hecho de ocupar mucha más memoria (el doble más o menos) y de requerir para su inserción y actualización un tiempo de proceso superior.

Evidentemente, no podemos indexar todos los campos de una tabla extensa ya que doblamos el tamaño de la base de datos. Igualmente, tampoco sirve de mucho el indexar todos los campos en una tabla pequeña ya que las selecciones pueden efectuarse rápidamente de todos modos.

Un caso en el que los índices pueden resultar muy útiles es cuando realizamos peticiones simultáneas sobre varias tablas. En este caso, el proceso de selección puede acelerarse sensiblemente si indexamos los campos que sirven de nexo entre las dos tablas.

Los índices pueden resultar contraproducentes si los introducimos sobre campos triviales a partir de los cuales no se realiza ningún tipo de petición ya que, además del problema de memoria ya mencionado, estamos ralentizando otras tareas de la base de datos como son la edición, inserción y borrado. Es por ello que vale la pena pensarselo dos veces antes de indexar un campo que no sirve de criterio para búsquedas o que es usado con muy poca frecuencia por razones de mantenimiento.

Campos a Seleccionar

- En la medida de lo posible hay que evitar que las sentencias SQL estén embebidas dentro del código de la aplicación. Es mucho más eficaz usar vistas o procedimientos almacenados por que el gestor los guarda compilados. Si se trata de una sentencia embebida el gestor debe compilarla antes de ejecutarla.
- Seleccionar exclusivamente aquellos que se necesiten
- No utilizar nunca `SELECT *` por que el gestor debe leer primero la estructura de la tabla antes de ejecutar la sentencia
- Si utilizas varias tablas en la consulta especifica siempre a que tabla pertenece cada campo, le ahorras al gestor el tiempo de localizar a que tabla pertenece el campo. En lugar de `SELECT Nombre, Factura FROM Clientes, Facturacion WHERE IdCliente = IdClienteFacturado`, usa: `SELECT Clientes.Nombre, Facturacion.Factura WHERE Clientes.IdCliente = Facturacion.IdClienteFacturado`.

Campos de Filtro

- Se procurará elegir en la cláusula `WHERE` aquellos campos que formen parte de la clave del fichero por el cual interrogamos. Además se especificarán en el mismo orden en el que estén definidos en la clave.
- Interrogar siempre por campos que sean clave.
- Si deseamos interrogar por campos pertenecientes a indices compuestos es mejor utilizar todos los campos de todos los indices. Supongamos que tenemos un índice formado por el campo NOMBRE y el campo APELLIDO y otro índice formado por el campo EDAD. La sentencia `WHERE NOMBRE='Juan' AND APELLIDO Like '%' AND EDAD = 20` sería más optima que `WHERE NOMBRE = 'Juan' AND EDAD = 20` por que el gestor, en este segundo caso, no puede usar el primer índice y ambas sentencias son equivalentes porque la condición `APELLIDO Like '%'` devolvería todos los registros.

Orden de las Tablas

- Cuando se utilizan varias tablas dentro de la consulta hay que tener cuidado con el orden empleado en la clausula `FROM`. Si deseamos saber cuantos alumnos se matricularon en el año 1996 y escribimos: `FROM Alumnos, Matriculas WHERE Alumno.IdAlumno = Matriculas.IdAlumno AND Matriculas.Año = 1996` el gestor recorrerá todos los alumnos para buscar sus matriculas y devolver las correspondientes. Si escribimos `FROM Matriculas, Alumnos WHERE Matriculas.Año = 1996 AND Matriculas.IdAlumno = Alumnos.IdAlumnos`, el gestor filtra las matrículas y después selecciona los alumnos, de esta forma tiene que recorrer menos registros.