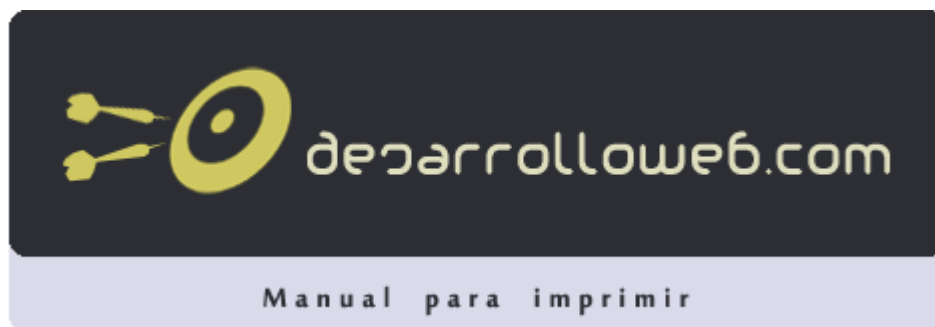


# Tutorial de SQL



## Autores del manual

Este manual ha sido realizado por los siguientes colaboradores de DesarrolloWeb.com:

**Claudio**

<http://personal.lobocom.es/claudio/>  
(18 capítulos)

**Rubén Alvarez**

Redactor de DesarrolloWeb.com  
<http://www.desarrolloweb.com>  
(13 capítulos)

**Agustin Jareño.**

<http://www.levanteweb.com/>  
(12 capítulos)

**Pol Salvat**

<http://www.mistrucos.net>  
(3 capítulos)

**Rosendo Lopez Robles**

(1 capítulo)

**Jonathan Soriano Folch**

(1 capítulo)

**Miguel Angel Alvarez**

Director de DesarrolloWeb.com  
<http://www.desarrolloweb.com>  
(1 capítulo)

**Sara Alvarez**

Equipo DesarrolloWeb.com  
<http://www.desarrolloweb.com>  
(1 capítulo)

## Qué es SQL

Las aplicaciones en red son cada día más numerosas y versátiles. En muchos casos, el esquema básico de operación es una serie de scripts que rigen el comportamiento de una base de datos.

Debido a la diversidad de lenguajes y de bases de datos existentes, la manera de comunicar entre unos y otros sería realmente complicada a gestionar de no ser por la existencia de estándares que nos permiten el realizar las operaciones básicas de una forma universal.

Es de eso de lo que trata el Structured Query Language que no es más que un lenguaje estándar de comunicación con bases de datos. Hablamos por tanto de un lenguaje normalizado que nos permite trabajar con cualquier tipo de lenguaje (ASP o PHP) en combinación con cualquier tipo de base de datos (MS Access, SQL Server, MySQL...).

El hecho de que sea estándar no quiere decir que sea idéntico para cada base de datos. En efecto, determinadas bases de datos implementan funciones específicas que no tienen necesariamente que funcionar en otras.

Aparte de esta universalidad, el SQL posee otras dos características muy apreciadas. Por una parte, presenta una potencia y versatilidad notables que contrasta, por otra, con su accesibilidad de aprendizaje.

[El manual de SQL de desarrolloweb](#) pretende dar a conocer las operaciones básicas que se pueden realizar con SQL y que tienen una aplicación directa con la creación de aplicaciones en red sin profundizar más de lo estrictamente necesario. Buscamos con ello ofrecer al webmaster un manual de referencia práctico y aplicado.

*Artículo por [Rubén Alvarez](#)*

## Tipos de campo

Como sabemos una base de datos esta compuesta de tablas donde almacenamos registros catalogados en función de distintos campos (características).

Un aspecto previo a considerar es la naturaleza de los valores que introducimos en esos campos. Dado que una base de datos trabaja con todo tipo de informaciones, es importante especificarle qué tipo de valor le estamos introduciendo de manera a, por un lado, facilitar la búsqueda posteriormente y por otro, optimizar los recursos de memoria.

Cada base de datos introduce tipos de valores de campo que no necesariamente están presentes en otras. Sin embargo, existe un conjunto de tipos que están representados en la totalidad de estas bases. Estos tipos comunes son los siguientes:

<b>Alfanuméricos</b>	Contienen cifras y letras. Presentan una longitud limitada (255 caracteres)
----------------------	---

<b>Númericos</b>	Existen de varios tipos, principalmente, enteros (sin decimales) y reales (con decimales).
<b>Booleanos</b>	Poseen dos formas: Verdadero y falso (Sí o No)
<b>Fechas</b>	Almacenan fechas facilitando posteriormente su explotación. Almacenar fechas de esta forma posibilita ordenar los registros por fechas o calcular los días entre una fecha y otra...
<b>Memos</b>	Son campos alfanuméricos de longitud ilimitada. Presentan el inconveniente de no poder ser indexados (veremos más adelante lo que esto quiere decir).
<b>Autoincrementales</b>	Son campos numéricos enteros que incrementan en una unidad su valor para cada registro incorporado. Su utilidad resulta más que evidente: Servir de identificador ya que resultan exclusivos de un registro.

Artículo por *Rubén Alvarez*

## Añadir un nuevo registro

Los registros pueden ser introducidos a partir de sentencias que emplean la instrucción Insert.

La sintaxis utilizada es la siguiente:

```
Insert Into nombre_tabla (nombre_campo1, nombre_campo2,...) Values (valor_campo1, valor_campo2...)
```

Un ejemplo sencillo a partir de nuestra tabla modelo es la introducción de un nuevo cliente lo cual se haría con una instrucción de este tipo:

```
Insert Into clientes (nombre, apellidos, direccion, poblacion, codigopostal, email, pedidos) Values ('Perico', 'Palotes', 'Percebe nº13', 'Lepe', '123456', 'perico@desarrolloweb.com', 33)
```

Como puede verse, los campos no numéricos o booleanos van delimitados por apostrofes: '. También resulta interesante ver que el código postal lo hemos guardado como un campo no numérico. Esto es debido a que en determinados países (Inglaterra, como no) los códigos postales contienen también letras.

**Nota:** Si deseamos practicar con una base de datos que está vacía primero debemos crear las tablas que vamos a llenar. Las tablas también se crean con sentencias SQL y [aprendemos a hacerlo en el último capítulo](#).

Aunque, de todos modos, puede que sea más cómodo utilizar un programa con interfaz gráfica, como Access, que nos puede servir para crear las tablas en bases de datos del propio [Access](#) o por ODBC a otras bases de datos como [SQL Server](#) o [MySQL](#), por poner dos ejemplos.

Otra posibilidad en una base de datos como MySQL, sería crear las tablas utilizando un software como [PhpMyAdmin](#).

Por supuesto, no es imprescindible rellenar todos los campos del registro. Eso sí, puede ser que determinados campos sean necesarios. Estos campos necesarios pueden ser definidos cuando construimos nuestra tabla mediante la base de datos.

**Nota:** Si no insertamos uno de los campos en la base de datos se inicializará con el valor por defecto que hayamos definido a la hora de crear la tabla. Si no hay valor por defecto, probablemente se inicialice

como NULL (vacío), en caso de que este campo permita valores nulos. Si ese campo no permite valores nulos (eso se define también al crear la tabla) lo más seguro es que la ejecución de la sentencia SQL nos de un error.

Resulta muy interesante, ya veremos más adelante el por qué, el introducir durante la creación de nuestra tabla un campo autoincrementable que nos permita asignar un único número a cada uno de los registros. De este modo, nuestra tabla clientes presentaría para cada registro un número exclusivo del cliente el cual nos será muy útil cuando consultemos varias tablas simultáneamente.

*Artículo por [Rubén Alvarez](#)*

## Borrar un registro

Para borrar un registro nos servimos de la instrucción Delete. En este caso debemos especificar cual o cuales son los registros que queremos borrar. Es por ello necesario establecer una selección que se llevara a cabo mediante la cláusula Where.

La forma de seleccionar se verá detalladamente en capítulos posteriores. Por ahora nos contentaremos de mostrar cuál es el tipo de sintaxis utilizado para efectuar estas supresiones:

Delete From nombre\_tabla Where condiciones\_de\_selección

**Nota:** Si deseamos practicar con una base de datos que está vacía primero debemos crear las tablas que vamos a llenar. Las tablas también se crean con sentencias SQL y [aprendemos a hacerlo en el último capítulo](#).

Si queremos por ejemplo borrar todos los registros de los clientes que se llamen Perico lo haríamos del siguiente modo:

```
Delete From clientes Where nombre='Perico'
```

Hay que tener cuidado con esta instrucción ya que si no especificamos una condición con Where, lo que estamos haciendo es **borrar toda la tabla**:

**Delete From clientes**

*Artículo por [Rubén Alvarez](#)*

## Actualizar un registro

Update es la instrucción que nos sirve para modificar nuestros registros. Como para el caso de Delete, necesitamos especificar por medio de Where cuáles son los registros en los que queremos hacer efectivas nuestras modificaciones. Además, obviamente, tendremos que especificar cuáles son los nuevos valores de los campos que deseamos actualizar. La sintaxis es de este tipo:

```
Update nombre_tabla Set nombre_campo1 = valor_campo1, nombre_campo2 =
valor_campo2,... Where condiciones_de_selección
```

Un ejemplo aplicado:

```
Update clientes Set nombre='José' Where nombre='Pepe'
```

Mediante esta sentencia cambiamos el nombre Pepe por el de José en todos los registros cuyo nombre sea Pepe.

Aquí también hay que ser cuidadoso de no olvidarse de usar Where, de lo contrario, modificaríamos todos los registros de nuestra tabla.

*Artículo por **Rubén Alvarez***

## **Selección de tablas I**

La selección total o parcial de una tabla se lleva a cabo mediante la instrucción Select. En dicha selección hay que especificar:

- Los campos que queremos seleccionar
- La tabla en la que hacemos la selección

En nuestra tabla modelo de clientes podríamos hacer por ejemplo una selección del nombre y dirección de los clientes con una instrucción de este tipo:

```
Select nombre, dirección From clientes
```

Si quisiésemos seleccionar todos los campos, es decir, **toda la tabla**, podríamos utilizar el comodín \* del siguiente modo:

```
Select * From clientes
```

Resulta también muy útil el filtrar los registros mediante condiciones que vienen expresadas después de la **cláusula Where**. Si quisiésemos mostrar los clientes de una determinada ciudad usaríamos una expresión como esta:

```
Select * From clientes Where poblacion Like 'Madrid'
```

Además, podríamos **ordenar los resultados** en función de uno o varios de sus campos. Para este último ejemplo los podríamos ordenar por nombre así:

```
Select * From clientes Where poblacion Like 'Madrid' Order By nombre
```

Teniendo en cuenta que puede haber más de un cliente con el mismo nombre, podríamos dar un segundo criterio que podría ser el apellido:

```
Select * From clientes Where poblacion Like 'Madrid' Order By nombre, apellido
```

Si invirtiésemos el orden « nombre,apellido » por « apellido, nombre », el resultado sería distinto. Tendríamos los clientes ordenados por apellido y aquellos que tuviesen apellidos idénticos se subclasificarían por el nombre.

Es posible también **clasificar por orden inverso**. Si por ejemplo quisiésemos ver nuestros clientes por orden de pedidos realizados teniendo a los mayores en primer lugar escribiríamos algo así:

```
Select * From clientes Order By pedidos Desc
```

Una opción interesante es la de efectuar **selecciones sin coincidencia**. Si por ejemplo buscásemos el saber en qué ciudades se encuentran nuestros clientes sin necesidad de que para ello aparezca varias veces la misma ciudad usaríamos una sentencia de esta clase:

```
Select Distinct poblacion From clientes Order By poblacion
```

Así evitaríamos ver repetido Madrid tantas veces como clientes tengamos en esa población.

*Artículo por **Rubén Alvarez***

## Selección de tablas II

Hemos querido compilar a modo de tabla ciertos operadores que pueden resultar útiles en determinados casos. Estos operadores serán utilizados después de la cláusula Where y pueden ser **combinados hábilmente mediante paréntesis** para optimizar nuestra selección a muy altos niveles.

Operadores matemáticos:	
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
<>	Distinto
=	Igual

Operadores lógicos
And
Or
Not

Otros operadores	
Like	Selecciona los registros cuyo valor de campo se asemeje, no teniendo en cuenta mayúsculas y minúsculas.

In y Not In	Da un conjunto de valores para un campo para los cuales la condición de selección es (o no) valida
Is Null y Is Not Null	Selecciona aquellos registros donde el campo especificado esta (o no) vacío.
Between...And	Selecciona los registros comprendidos en un intervalo
Distinct	Selecciona los registros no coincidentes
Desc	Clasifica los registros por orden inverso

Comodines	
*	Sustituye a todos los campos
%	Sustituye a cualquier cosa o nada dentro de una cadena
_	Sustituye un solo carácter dentro de una cadena

Veamos a continuación aplicaciones practicas de estos operadores.

En esta sentencia seleccionamos todos los clientes de Madrid cuyo nombre no es Pepe. Como puede verse, empleamos **Like** en lugar de = simplemente para evitar inconvenientes debido al empleo o no de mayúsculas.

```
Select * From clientes Where poblacion Like 'madrid' And Not nombre Like 'Pepe'
```

Si quisiéramos recoger en una selección a los clientes de nuestra tabla cuyo **apellido comienza por A y cuyo número de pedidos esta comprendido entre 20 y 40**:

```
Select * From clientes Where apellidos like 'A%' And pedidos Between 20 And 40
```

El operador **In**, lo veremos más adelante, es muy práctico para consultas en varias tablas. Para casos en una sola tabla es empleado del siguiente modo:

```
Select * From clientes Where poblacion In ('Madrid','Barcelona','Valencia')
```

De esta forma **seleccionamos aquellos clientes que vivan en esas tres ciudades.**

*Artículo por [Rubén Alvarez](#)*

## Selección de tablas III

Una base de datos puede ser considerada como un conjunto de tablas. Estas tablas en muchos casos están relacionadas entre ellas y se complementan unas con otras.

Refiriéndonos a nuestro clásico ejemplo de una base de datos para una aplicación de e-commerce, la tabla clientes de la que hemos estado hablando puede estar perfectamente coordinada con una tabla donde almacenamos los pedidos realizados por cada cliente. Esta

tabla de pedidos puede a su vez estar conectada con una tabla donde almacenamos los datos correspondientes a cada artículo del inventario.

De este modo podríamos fácilmente obtener informaciones contenidas en esas tres tablas como puede ser la designación del artículo más popular en una determinada región donde la designación del artículo sería obtenida de la tabla de artículos, la popularidad (cantidad de veces que ese artículo ha sido vendido) vendría de la tabla de pedidos y la región estaría comprendida obviamente en la tabla clientes.

Este tipo de organización basada en múltiples tablas conectadas nos permite trabajar con tablas mucho más manejables a la vez que nos evita copiar el mismo campo en varios sitios ya que podemos acceder a él a partir de una simple llamada a la tabla que lo contiene.

En este capítulo veremos como, sirviéndonos de lo aprendido hasta ahora, podemos realizar fácilmente selecciones sobre varias tablas. Definamos antes de nada las diferentes tablas y campos que vamos a utilizar en nuestros ejemplos:

<b>Tabla de clientes</b>	
<b>Nombre campo</b>	<b>Tipo campo</b>
id_cliente	Numérico entero
nombre	Texto
apellidos	Texto
direccion	Texto
poblacion	Texto
codigopostal	Texto
telefono	Numérico entero
email	Texto

<b>Tabla de pedidos</b>	
<b>Nombre campo</b>	<b>Tipo campo</b>
id_pedido	Numérico entero
id_cliente	Numérico entero
id_articulo	Numérico entero
fecha	Fecha
cantidad	Numérico entero

<b>Tabla de artículos</b>	
<b>Nombre campo</b>	<b>Tipo campo</b>
id_articulo	Numérico entero
titulo	Alfanumérico
autor	Alfanumérico

editorial	Alfanumérico
precio	Numérico real

Estas tablas pueden ser utilizadas simultáneamente para extraer informaciones de todo tipo. Supongamos que queremos enviar un mailing a todos aquellos que hayan realizado un pedido ese mismo día. Podríamos escribir algo así:

**Select clientes.apellidos, clientes.email From clientes,pedidos Where pedidos.fecha like '25/02/00' And pedidos.id\_cliente= clientes.id\_cliente**

Como puede verse esta vez, después de la cláusula From, introducimos el nombre de las dos tablas de donde sacamos las informaciones. Además, el nombre de cada campo va precedido de la tabla de proveniencia separados ambos por un punto. En los campos que poseen un nombre que solo aparece en una de las tablas, no es necesario especificar su origen aunque a la hora de leer la sentencia puede resultar más claro el precisarlo. En este caso el campo fecha podría haber sido designado como "fecha" en lugar de "pedidos.fecha".

Veamos otro ejemplo más para consolidar estos nuevos conceptos. Esta vez queremos ver el título del libro correspondiente a cada uno de los pedidos realizados:

**Select pedidos.id\_pedido, articulos.titulo From pedidos, articulos Where pedidos.id\_articulo=articulos.id\_articulo**

En realidad la filosofía continua siendo la misma que para la consulta de una única tabla.

*Artículo por **Rubén Alvarez***

## Selección de tablas IV

Además de los criterios hasta ahora explicados para realizar las consultas en tablas, SQL permite también aplicar un conjunto de funciones predefinidas. Estas funciones, aunque básicas, pueden ayudarnos en algunos momentos a expresar nuestra selección de una manera más simple sin tener que recurrir a operaciones adicionales por parte del script que estemos ejecutando.

Algunas de estas funciones son representadas en la tabla siguiente :

Función	Descripción
Sum(campo)	Calcula la suma de los registros del campo especificado
Avg(Campo)	Calcula la media de los registros del campo especificado
Count(*)	Nos proporciona el valor del numero de registros que han sido seleccionados
Max(Campo)	Nos indica cual es el valor máximo del campo
Min(Campo)	Nos indica cual es el valor mínimo del campo

Dado que el campo de la función no existe en la base de datos, sino que lo estamos generando virtualmente, esto puede crear inconvenientes cuando estamos trabajando con nuestros scripts a la hora de tratar su valor y su nombre de campo. Es por ello que el valor de la **función ha de ser recuperada a partir de un alias** que nosotros especificaremos en la sentencia SQL a partir de la instrucción **AS**. La cosa podría quedar así:

### **Select Sum(total) As suma\_pedidos From pedidos**

A partir de esta sentencia calculamos la suma de los valores de todos los pedidos realizados y almacenamos ese valor en un campo virtual llamado suma\_pedidos que podrá ser utilizado como cualquier otro campo por nuestras paginas dinámicas.

Por supuesto, todo lo visto hasta ahora puede ser aplicado en este tipo de funciones de modo que, por ejemplo, podemos establecer condiciones con la cláusula Where construyendo sentencias como esta:

### **Select Sum(cantidad) as suma\_articulos From pedidos Where id\_articulo=6**

Esto nos proporcionaría la cantidad de **ejemplares de un determinado libro que han sido vendidos**.

Otra propiedad interesante de estas funciones es que **permiten realizar operaciones con varios campos dentro de un mismo paréntesis**:

### **Select Avg(total/cantidad) From pedidos**

Esta sentencia da como resultado el **precio medio al que se están vendiendo los libros**. Este resultado no tiene por qué coincidir con el del **precio medio de los libros presentes en el inventario**, ya que, puede ser que la gente tenga tendencia a comprar los libros caros o los baratos:

### **Select Avg(precio) as precio\_venta From articulos**

Una cláusula interesante en el uso de funciones es Group By. Esta cláusula nos permite agrupar registros a los cuales vamos a aplicar la función. Podemos por ejemplo calcular el **dinero gastado por cada cliente**:

### **Select id\_cliente, Sum(total) as suma\_pedidos From pedidos Group By id\_cliente**

O saber el **numero de pedidos que han realizado**:

### **Select id\_cliente, Count(\*) as numero\_pedidos From pedidos Group By id\_cliente**

Las posibilidades como vemos son numerosas y pueden resultar prácticas. Todo queda ahora a disposición de nuestras ocurrencias e imaginación.

**Referencia:** En una de nuestras FAQ se dan otros ejemplos de trabajo con funciones en selecciones SQL. [Seleccionar una parte de una cadena en SQL](#). En general, en las FAQ podemos encontrar respuestas a otras preguntas habituales.

*Artículo por Rubén Alvarez*

## Optimizar prestaciones I

Las bases de datos (BD) cuanto más extensas requieren una mayor atención a la hora de organizar sus contenidos. Cuando se trabaja con tablas de miles o decenas de miles de registros la búsqueda de un determinado dato puede resultar un proceso largo que ralentiza enormemente la creación de nuestra página.

Es por ello importante tener en cuenta una serie de aspectos indispensables para el mejor funcionamiento de la base.

### Gestión y elección de los índices

Los índices son campos elegidos arbitrariamente por el constructor de la BD 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 BD. 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**. En el ejemplo de nuestra librería virtual estos campos serían `id_cliente` e `id_articulo`.

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 pensárselo dos veces antes de indexar un campo que no sirve de criterio para búsquedas de los internautas y que es usado con muy poca frecuencia por razones de mantenimiento.

### Gestión de los nexos entre tablas

El enlace entre tablas es uno de los puntos más peliagudos y que puede llevar a la absoluta ralentización de la base de datos a causa "pequeños" detalles que resultan ser fatales.

Imaginemos que trabajamos con una pequeña BD constituida por dos tablas de 1000 registros cada una. Imaginemos ahora una selección simultánea en la que imponemos la condición de que el valor un campo de la primera sea igual a de una segunda, algo que se realiza con mucha frecuencia. En este tipo de casos, la BD leerá y comparará cada valor de campo de una con cada valor de campo de la otra. Esto representaría un millón de lecturas. Este hecho podría agravarse si consultamos una tercera tabla al mismo tiempo y podría llegar a ser catastrófico si tenemos en cuenta que la BD esta siendo consultada por varios internautas al mismo tiempo.

Para evitar situaciones de colapso, es necesario **indexar cada uno de los campos que sirven de enlace entre esas tablas**. En el ejemplo de nuestra librería virtual, ya lo hemos dicho, estos campos serían `id_cliente` e `id_articulo`. Además, resulta también de vital importancia el **definir esos campos de una forma estrictamente idéntica en cada una**

**de las tablas**, es decir, el campo ha de ser de la misma naturaleza y características. No vale definirlo como real en una tabla y entero en otra o cambiar la longitud máxima para los alfanuméricos o que en una tabla sea de longitud constante y en otra variable...

El gestionar inteligentemente estos aspectos puede solucionarnos muchos quebraderos de cabeza y permitir a los internautas navegar más agradablemente por nuestro sitio.

*Artículo por **Rubén Alvarez***

## **Optimizar prestaciones II**

### **Gestion de los campos**

Ya hemos comentado por encima los diferentes tipos de campo existentes en una base de datos. La elección del tipo de campo apropiado para cada caso puede ayudarnos también a optimizar el tamaño y rapidez de nuestra base de datos.

Las preguntas que hay que hacerse a la hora de elegir la naturaleza y dimensiones del campo son:

**-¿Qué tipo de dato voy a almacenar en el campo? Números, texto, fechas...**

**-¿Cuál es el tamaño máximo que espero que pueda alcanzar alguno de los registros del campo?**

Hay que tener en cuenta que cuanto más margen le demos al valor máximo del campo, más aumentará el tamaño de nuestra base de datos y más tiempo tardará en realizar las consultas. Además, el factor tamaño puede verse agravado si estamos definiendo un campo indexado, para los cuales, el espacio ocupado es aproximadamente del doble.

Un consejo práctico es que las fechas sean almacenadas en formato de fecha ya que ello nos permite reducir el espacio que ocupan en memoria de más del doble y por otro lado, podremos aprovechar las prestaciones que SQL y nuestro lenguaje de servidor nos ofrecen. Podremos calcular la diferencia de días entre dos fechas, ordenar los registros por fecha, mostrar los registros comprendidos en un intervalo de tiempo...

Existe la posibilidad para los campos de texto de fijar una cierta longitud para el campo o dejar que cada registro tenga una longitud variable en función del número de caracteres que posea. Elegir campos de longitud variable nos puede ayudar a optimizar los recursos de memoria de la BD, no obstante, es un arma de doble filo ya que las consultas se realizan más lentamente puesto que obligamos a la tabla a establecer cuál es el tamaño de cada registro que se está comparando en lugar de saberlo de antemano. Es por tanto aconsejable, para los campos indexados de pequeño tamaño, atribuirles una longitud fija.

*Artículo por **Rubén Alvarez***

## Algunos trucos prácticos

### Eliminar llamadas a bases de datos

En páginas tipo portal en las que a los lados se encuentran enlaces que son impresos a partir de bases de datos (distintas secciones, servicios,...) existe siempre un efecto ralentizador debido a que se trata de páginas altamente visitadas que efectúan múltiples llamadas a BD sistemáticamente en cada una de sus páginas.

Una forma de agilizar la visualización de estas páginas es textualizando estos enlaces a partir de scripts internos. Pongamos el ejemplo de DesarrolloWeb:

Como puede verse, a los lados hay secciones como "Vuestras páginas", "Cosecha del 2000", "Manuales" cuyos enlaces están almacenados en bases de datos. Sin embargo, los enlaces que se visualizan en la página no han sido obtenidos por llamadas a bases de datos sino que, cada vez que un nuevo elemento de la sección es añadido, esto actualiza automáticamente, por medio de un script, un archivo texto en el que el nuevo enlace es incluido y el más antiguo es eliminado. Es, de hecho, este archivo texto el que es insertado en el código fuente de la página. De este modo evitamos media docena de llamadas a bases de datos cada vez que una página es vista lo cual permite optimizar recursos de servidor de una manera significativa.

### Eliminar palabras cortas y repeticiones

En situaciones en la que nuestra base de datos tiene que almacenar campos de texto extremadamente largos y dichos campos son requeridos para realizar selecciones del tipo LIKE '%algo%', los recursos de la BD pueden verse sensiblemente mermados. Una forma de ayudar a gestionar este tipo búsquedas es incluyendo un campo adicional.

Este campo adicional puede ser creado automáticamente por medio de scripts y en él incluiríamos el texto original, del cual habremos eliminado palabras triviales como artículos, preposiciones o posesivos. Nos encargaremos además de eliminar las palabras que estén repetidas. De esta forma podremos disminuir sensiblemente el tamaño del campo que va a ser realmente consultado.

Hemos comentado en otros capítulos que los campos texto de mas de 255 caracteres denominados memo no pueden ser indexados. Si aún después de esta primera filtración nuestro campo continua siendo demasiado largo para ser indexado, lo que se puede hacer es cortarlo en trozos de 255 caracteres de manera a que lo almacenemos en distintos campos que podrán ser indexados y por tanto consultados con mayor rapidez.

*Artículo por [Rubén Alvarez](#)*

## Creación de tablas

En general, la mayoría de las bases de datos poseen potentes editores de bases que permiten la creación rápida y sencilla de cualquier tipo de tabla con cualquier tipo de formato.

Sin embargo, una vez la base de datos está alojada en el servidor, puede darse el caso de que queramos introducir una nueva tabla ya sea con carácter temporal (para gestionar un carrito

de compra por ejemplo) o bien permanente por necesidades concretas de nuestra aplicación.

En estos casos, podemos, a partir de una sentencia SQL, crear la tabla con el formato que deseemos lo cual nos puede ahorrar más de un quebradero de cabeza.

Este tipo de sentencias son especialmente útiles para bases de datos como Mysql, las cuales trabajan directamente con comandos SQL y no por medio de editores.

Para crear una tabla debemos especificar diversos datos: El nombre que le queremos asignar, los nombres de los campos y sus características. Además, puede ser necesario especificar cuáles de estos campos van a ser índices y de qué tipo van a serlo.

La sintaxis de creación puede variar ligeramente de una base de datos a otra ya que los tipos de campo aceptados no están completamente estandarizados.

A continuación os explicamos someramente la sintaxis de esta sentencia y os proponemos una serie de ejemplos prácticos:

### Sintaxis

```
Create Table nombre_tabla
(
nombre_campo_1 tipo_1
nombre_campo_2 tipo_2
nombre_campo_n tipo_n
Key(campo_x,...)
)
```

Pongamos ahora como ejemplo la creación de la tabla pedidos que hemos empleado en capítulos previos:

```
Create Table pedidos
(
id_pedido INT(4) NOT NULL AUTO_INCREMENT,
id_cliente INT(4) NOT NULL,
id_articulo INT(4) NOT NULL,
fecha DATE,
cantidad INT(4),
total INT(4), KEY(id_pedido,id_cliente,id_articulo)
)
```

En este caso creamos los campos *id* los cuales son considerados de tipo entero de una longitud especificada por el número entre paréntesis. Para *id\_pedido* requerimos que dicho campo se incremente automáticamente (AUTO\_INCREMENT) de una unidad a cada introducción de un nuevo registro para, de esta forma, automatizar su creación. Por otra parte, para evitar un mensaje de error, es necesario requerir que los campos que van a ser definidos como índices no puedan ser nulos (NOT NULL).

El campo *fecha* es almacenado con formato de fecha (DATE) para permitir su correcta explotación a partir de las funciones previstas a tal efecto.

Finalmente, definimos los índices enumerándolos entre paréntesis precedidos de la palabra

KEY o INDEX.

Del mismo modo podríamos crear la tabla de *artículos* con una sentencia como ésta:

```
Create Table articulos
(
id_articulo INT(4) NOT NULL AUTO_INCREMENT,
titulo VARCHAR(50),
autor VARCHAR(25),
editorial VARCHAR(25),
precio REAL,
KEY(id_articulo)
)
```

En este caso puede verse que los campos alfanuméricos son introducidos de la misma forma que los numéricos. Volvemos a recordar que en tablas que tienen campos comunes es de vital importancia definir estos campos de la misma forma para el buen funcionamiento de la base.

Muchas son las opciones que se ofrecen al generar tablas. No vamos a tratarlas detalladamente pues sale de lo estrictamente práctico. Tan sólo mostraremos algunos de los tipos de campos que pueden ser empleados en la creación de tablas con sus características:

Tipo	Bytes	Descripción
INT o INTEGER	4	Números enteros. Existen otros tipos de mayor o menor longitud específicos de cada base de datos.
DOUBLE o REAL	8	Números reales (grandes y con decimales). Permiten almacenar todo tipo de número no entero.
CHAR	1/caracter	Alfanuméricos de longitud fija predefinida
VARCHAR	1/caracter+1	Alfanuméricos de longitud variable
DATE	3	Fechas, existen multiples formatos específicos de cada base de datos
BLOB	1/caracter+2	Grandes textos no indexables
BIT o BOOLEAN	1	Almacenan un bit de información (verdadero o falso)

Artículo por **Rubén Alvarez**

## Funciones para búsquedas con fechas en Access

Búsquedas con fechas en Access Hemos recibido una pregunta recientemente de un amigo que deseaba realizar búsquedas en Access utilizando, en las condiciones del Where, campos de tipo fecha. Después de varios intentos hemos rescatado un par de notas que pueden ser interesantes para publicar. Seguro que sirven de ayuda a otras personas que tengan que trabajar con fechas en consultas de Access.

La consulta era la siguiente:

*Tengo una tabla con varios campos. Dos de ellos son fechas, que corresponden con un intervalo. Uno de ellos es la fecha de inicio del intervalo (fechadesde) y otro la de final del intervalo (fechahasta)*

*Quisiera saber cómo se puede hacer una consulta SQL en Access para obtener los registros cuyo intervalo de fechas contenga el día de hoy.*

*Es decir, que la fecha desde sea menor que hoy y la fecha hasta sea mayor que hoy.*

Estuvimos primero haciendo un par de pruebas sin éxito, comparando las fechas con operadores aritméticos. En algunos casos obtuvimos la respuesta esperada, pero no siempre funcionaban las sentencias y teníamos problemas al ejecutarlas desde Access o desde el servidor web, porque no devolvían los mismos resultados.

### **Función DateDiff()**

Al final, la respuesta que propusimos pasó por utilizar la función DateDiff, que sirve para obtener la diferencia entre dos fechas. Por ejemplo:

```
DateDiff("y", #06/10/2004#, Now())
```

Nos dice los días que han pasado desde el seis de octubre de 2004.

Nosotros podemos utilizarla como condición en un where de una sentencia SQL. Por ejemplo, para una sentencia como esta:

```
DateDiff("y",A,B)
```

- Si son iguales, la función devolverá cero.
- Si A es una fecha anterior a B, entonces la función devolverá un número de días que será mayor que cero.
- Si A es una fecha posterior a B, entonces devolverá un número de días que será menor que cero.

Tenemos que comparar el día de hoy con las fechas desde y hasta. Hoy tiene que ser mayor que desde y menor que hasta. Nos queda como resultado esta sentencia.

```
SELECT * FROM vuelos WHERE  
DateDiff('y',fechadesde,now())>=0  
and DateDiff('y',fechahasta,nom())<=0
```

**Nota:** Hay que tener cuidado con el idioma de las fechas, pues en castellano se escriben de manera distinta que en inglés. Access intenta interpretar la fecha correctamente, por ejemplo, si introducimos 02/26/04 pensará que está trabajando en fechas en inglés y si introducimos 26/02/04 pensará que estamos escribiendo las fechas en castellano. El problema es con una fecha como 02/02/04 que su valor dependerá de cómo esté configurado el Access, en castellano o inglés.

### **Función DatePart**

Sirve para extraer parte de una fecha. Recibe dos parámetros, el primero indica mediante un string la parte a obtener. El otro parámetro es la fecha con la que se desea trabajar.

```
DatePart("m",fecha)
```

En este caso se está indicando que se desea obtener el mes del año. Otro valor posible para el primer parámetro es, por ejemplo "yyyy", que se utiliza para obtener el año con cuatro dígitos. Un ejemplo de sentencia SQL que utiliza esta función puede ser la siguiente:

```
SELECT DatePart("yyyy",validezdesde) FROM vuelos
```

### Función DateAdd

Esta última función que vamos a ver en el presente artículo sirve para añadir a la fecha, algo como días, meses o años. Para ello la función recibe tres parámetros, el primero corresponde con un string para indicar las unidades de lo que deseamos añadir, por ejemplo, días, meses o años. El segundo parámetro es el número de días meses o años a adicionar y el tercer parámetro es la fecha a la que sumar esos valores. Vemos un ejemplo de su sintaxis:

```
DateAdd("yyyy",10,validezdesde)
```

En este ejemplo la función DateAdd devolvería una fecha diez años posterior a validezdesde. Otros valores para el string del primer parámetro son "d", para añadir días, o "m", para añadir meses.

Un ejemplo del funcionamiento de esta función en una sentencia SQL es el siguiente:

```
SELECT DateAdd("yyyy",10,validezdesde) FROM vuelos
```

*Artículo por **Miguel Angel Alvarez***

## Función en SQL para el cálculo de días laborables

*/\*Primeramente declaramos que vamos a crear una funcion, en este caso se llama Dif Dias y recibe dos parámetros, la fecha inicial del período y la final\*/*

```
CREATE FUNCTION DifDias(@StartDate DATETIME,@EndDate DATETIME)
RETURNS integer
AS
Begin
```

```
//Con esta variable calculamos cuantos días "normales" hay en el rango de fechas
```

```
DECLARE @DaysBetween INT
```

```
//Con esta variable acumulamos los días totales
```

```
DECLARE @BusinessDays INT
```

```
//esta variable nos sirve de contador para saber cuando lleguemos al ultimo día del rango
```

```
DECLARE @Cnt INT
```

```
/*esta variable es la que comparamos para saber si el día que esta calculando es sábado o domingo*/
```

```
DECLARE @EvalDate DATETIME
```

```
/*Esta par de variables sirven para comparar las dos fechas, si son iguales, la funcion nos regresa un 0*/
```

```
DECLARE @ini VARCHAR(10)
```

```
DECLARE @fin VARCHAR(10)

//Inicializamos algunas variables

SELECT @DaysBetween = 0
SELECT @BusinessDays = 0
SELECT @Cnt=0

//Calculamos cuantos dias normales hay en el rango de fechas

SELECT @DaysBetween = DATEDIFF(DAY,@StartDate,@EndDate) + 1

/*Ordenamos el formato de las fechas para que no importando como se proporcionen se comparen igual*/

SELECT @ini = (SELECT CAST((CAST(datepart(dd,@StartDate)AS
VARCHAR(2))+ '/' + CAST(datepart(mm,@StartDate)AS
VARCHAR(2))+ '/' + CAST(datepart(yy,@StartDate)AS VARCHAR(4))) as
varchar(10)))
SELECT @fin = (SELECT CAST((CAST(datepart(dd,@EndDate)AS
VARCHAR(2))+ '/' + CAST(datepart(mm,@EndDate)AS VARCHAR(2))+ '/' +
CAST(datepart(yy,@EndDate)AS VARCHAR(4)))as varchar(10)))

//Se comparan las dos fechas

IF @ini <>@fin
BEGIN

/*Si la diferencia de fechas es igual a dos, es porque solo ha transcurrido un dia, asi que solo se valida que no vaya a
marcar dias de mas*/

IF @DaysBetween = 2
BEGIN
SELECT @BusinessDays = 1
END
ELSE
BEGIN
WHILE @Cnt < @DaysBetween
BEGIN

/*Se Iguala la fecha a que vamos a calcular para saber si es sabado o domingo en la variable @EvalDate sumandole
los dias que marque el contador, el cual no debe ser mayor que el numero total de dias que hay en el rango de
fechas*/

SELECT @EvalDate = @StartDate + @Cnt

/*Utilizando la funcion datepart con el parametro dw que calcula que dia de la semana corresponde una fecha
determinada, determinados que no sea sabado (7) o domingo (1)*/

IF ((datepart(dw,@EvalDate) <> 1) and
(datepart(dw,@EvalDate) <> 7) )
BEGIN

/*Si no es sabado o domingo, entonces se suma uno al total de dias que queremos desplegar*/

SELECT @BusinessDays = @BusinessDays + 1
END

//Se suma un dia mas al contador

SELECT @Cnt = @Cnt + 1
END
END
END
ELSE
BEGIN
```

```
//Si fuese cierto que las fechas eran iguales se despliegue cero  
SELECT @BusinessDays = 0  
END  
  
//Al finalizar el ciclo, la funcion regresa el numero total de dias  
return (@BusinessDays)  
END
```

*Artículo por **Rosendo Lopez Robles***

## SQL con Oracle

### Introducción:

Antes de empezar me gustaría decir que este curso esta basado en Oracle, es decir los ejemplos expuestos y material se han utilizado sobre Oracle. Por otro lado decir que pienso que es interesante saber algo de SQL antes de comenzar con MYSQL, ya que, aunque existen algunos cambios insignificantes, sabiendo manejar SQL sabes manejar MYSQL.

### Algunas características:

#### SQL: Structured query language.

- Permite la comunicación con el sistema gestor de base de datos.
- En su uso se puede especificar que quiere el usuario.
- Permite hacer consulta de datos.

### Tipos de datos:

#### CHAR:

- Tienen una longitud fija.
- Almacena de 1 a 255.
- Si se introduce una cadena de menos longitud que la definida se rellenara con blancos a la derecha hasta quedar completada.
- Si se introduce una cadena de mayor longitud que la fijada nos dará un error.

#### VARCHAR:

- Almacena cadenas de longitud variable.
- La longitud máxima es de 2000 caracteres.
- Si se introduce una cadena de menor longitud que la que esta definida, se almacena con esa longitud y no se rellenara con blancos ni con ningún otro carácter a la derecha hasta completar la longitud definida.
- Si se introduce una cadena de mayor longitud que la fijada, nos dará un error

#### NUMBER:

- Se almacenan tanto enteros como decimales.

- Number (precisión, escala)
- Ejemplo:

X=number (7,2)

X=155'862 à Error ya que solo puede tomar 2 decimales

X= 155'86 à Bien

**Nota:** El rango máximo va de 1 a 38.

### LONG:

- No almacena números de gran tamaño, sino cadenas de caracteres de hasta 2 GB

### DATE:

- Almacena la fecha. Se almacena de la siguiente forma:

Siglo/Año/Mes/Día/Hora/Minutos/Segundos

### RAW:

- Almacena cadenas de Bytes (gráficos, sonidos...)

### LONGRAW:

- Como el anterior pero con mayor capacidad.

### ROWID:

- Posición interna de cada una de las columnas de las tablas.
- Sentencias de consultas de datos

Select:

```
Select [ALL | Distinct] [expresión_columna1, expresión_columna2, ..., | *]
```

```
From [nombre1, nombre_tabla1, ..., nombre_tablan]
```

```
{[Where condición]
```

```
[Order By expresión_columna [Desc | Asc]...]};
```

Vamos a explicar como leer la consulta anterior y así seguir la pauta para todas las demás. Cuando ponemos [] significa que debemos la que va dentro debe existir, y si además ponemos | significa que deberemos elegir un valor de los que ponemos y no mas de uno. En cambio si ponemos {} significa que lo que va dentro de las llaves puede ir o no, es decir es opcional y se pondrá según la consulta.

**Nota:** En el select el valor por defecto entre ALL y DISTINCT es ALL.

- Alias = El nuevo nombre que se le da a una tabla. Se pondrá entre comillas
- Order By = Ordena ascendentemente (Asc) (valor por defecto) o descendientemente (Desc).
- All = Recupera todas las filas de la tabla aunque estén repetidas.
- Distinct = Solo recupera las filas que son distintas.
- Desc Emple; = Nos da un resumen de la tabla y sus columnas. En este caso de la tabla Emple.
- Not Null= Si aparece en una lista de una columna significa que la columna no puede tener valores nulos.
- Null= Si está nulo.

**Nota:** Nótese que cada consulta de SQL que hagamos hemos de terminarla con un punto y coma ";".

Varios ejemplos para verlo mas claro:

```
SELECT JUGADOR_NO, APELLIDO, POSICION, EQUIPO
FROM JUGADORES
WHERE EQUIPO_NO = 'VALENCIA'
ORDER BY APELLIDO;
```

Este ejemplo mostrar el número de jugador (jugador\_no) el apellido (Apellido), la posición en la que juega (Posición), y el equipo (Equipo) al que pertenece.

Seleccionara todos los datos de la tabla jugadores donde (Where) el nombre de equipo (Equipo\_No) sea igual que la palabra 'Valencia' y se ordenara (order by) apellido. Notemos también que no pone ni 'Distinct' ni 'All'. Por defecto generara la sentencia con ALL.

```
SELECT *
FROM JUGADORES
WHERE POSICION = 'DELANTERO'
ORDER BY JUGADOR_NO;
```

Este ejemplo muestra todos los campos de la tabla jugadores donde (Where) la posición sea igual que 'Delantero' y lo ordena por número de jugador. Al no poner nada se presupone que es ascendentemente (Asc).

```
SELECT *
FROM JUGADORES
WHERE EQUIPO_NO = 'VALENCIA' AND POSICION = 'DELANTERO'
ORDER BY APELLIDO DESC, JUGADOR_NO ASC;
```

En este ejemplo selecciona todos los campos de la tabla jugadores donde (Where) el nombre del equipo sea igual a 'Valencia' y la posición de los jugadores sea igual a 'Delantero'. Por ultimo los ordena por 'Apellido' descendentemente y por numero de jugador ascendentemente.

*Artículo por [Agustin Jareño](#).*

## SQL con Oracle. Operadores

### Operadores aritméticos:

+ = Suma  
- = Resta  
\* = Multiplicación  
/ = división

### Operadores de comparación y lógicos:

!> = Distinto  
>= = Mayor o igual que  
<= = Menor o igual que = = Igual que  
Like = Se utiliza para unir cadenas de caracteres. Propiedades:  
% = representa cualquier cadena de caracteres de 0 o mas caracteres.

\_ = representa un único carácter cualquiera.

Not = Negación

And = y

a and b

Cierto si son ciertas a y b.

Or = o

a or b

Cierto si a o b son ciertas

### Veamos un par de ejemplos:

Obtenemos los datos de los jugadores cuyos apellidos empiecen con una "S":

```
SELECT APELLIDO
FROM JUGADORES
WHERE APELLIDO LIKE 'S%';
```

Obtenemos aquellos apellidos que tengan una "R" en la segunda posición:

```
SELECT APELLIDO
FROM JUGADORES
WHERE APELLIDO LIKE '_R*';
```

Obtenemos aquellos apellidos que empiezan por "A" y tiene una "o" en su interior:

```
SELECT APELLIDO
FROM JUGADORES
WHERE APELLIDOS LIKE 'A%O%';
```

### Comprobación con conjuntos de valores:

- In= permite saber si una expresión pertenece o no a un conjunto de valores.
- Between= permite saber si una expresión esta o no entre esos valores:

### Ejemplo:

```
SELECT APELLIDOS
FROM JUGADORES
WHERE JUGADOR_NUM IN (10, 20);
```

Selecciona los apellidos de los jugadores donde el número de jugador (Jugador\_num) sea (In) o 10 o 20

```
SELECT APELLIDOS
FROM JUGADORES
WHERE SALARIO NOT BETWEEN 15000000 AND 20000000;
```

Selecciona los apellidos de los jugadores donde el salario de estos no este entre (Not Between) 15000000 y 20000000.

*Artículo por **Agustín Jareño**.*

## Subconsultas SQL

### Subconsultas:

Consulta que se hace sobre los datos que nos da otra consulta. Su formato es:

```
SELECT _____  
FROM _____  
WHERE CONDICION OPERADOR (SELECT _____  
FROM _____  
WHERE CONDICION OPERADOR); Ejemplo:
```

Obtenemos los jugadores con la misma posición que "Sánchez":

```
SELECT APELLIDO  
FROM EMPLE  
WHERE POSICION = (SELECT OFICIO  
FROM EMPLE  
WHERE APELLIDO LIKE 'GIL');
```

Seleccionamos en todos los campos de la tabla Jugadores cuya sede está en Madrid o Barcelona:

```
SELECT *  
FROM JUGADORES  
WHERE EQUIPO_NOM IN (SELECT EQUIPO_NOM  
FROM SEDE  
WHERE LOC IN ('MADRID', 'BARCELONA');  
FROM SEDE  
WHERE LOC IN ('MADRID', 'BARCELONA');
```

*Artículo por **Agustín Jareño**.*

## Funciones SQL

### Funciones de valores simples:

ABS(n)= Devuelve el valor absoluto de (n).  
CEIL(n)=Obtiene el valor entero inmediatamente superior o igual a "n".  
FLOOR(n) = Devuelve el valor entero inmediatamente inferior o igual a "n".  
MOD (m, n)= Devuelve el resto resultante de dividir "m" entre "n".  
NVL (valor, expresión)= Sustituye un valor nulo por otro valor.  
POWER (m, exponente)= Calcula la potencia de un numero.  
ROUND (numero [, m])= Redondea números con el numero de dígitos de precisión indicados.  
SIGN (valor)= Indica el signo del "valor".  
SQRT(n)= Devuelve la raíz cuadrada de "n".  
TRUNC (numero, [m])= Trunca números para que tengan una cierta cantidad de dígitos de precisión.  
VARIANCE (valor)= Devuelve la varianza de un conjunto de valores.

### Funciones de grupos de valores:

AVG(n)= Calcula el valor medio de "n" ignorando los valores nulos.  
COUNT (\* | Expresión)= Cuenta el numero de veces que la expresión evalúa algún dato con valor no nulo. La opción "\*" cuenta todas las filas seleccionadas.  
MAX (expresión)= Calcula el máximo.  
MIN (expresión)= Calcula el mínimo.  
SUM (expresión)= Obtiene la suma de los valores de la expresión.  
GREATEST (valor1, valor2...)= Obtiene el mayor valor de la lista.  
LEAST (valor1, valor2...)= Obtiene el menor valor de la lista.

### Funciones que devuelven valores de caracteres:

CHR(n) = Devuelve el carácter cuyo valor en binario es equivalente a "n".  
CONCAT (cad1, cad2)= Devuelve "cad1" concatenada con "cad2".  
LOWER (cad)= Devuelve la cadena "cad" en minúsculas.  
UPPER (cad)= Devuelve la cadena "cad" en mayúsculas.  
INITCAP (cad)= Convierte la cadena "cad" a tipo titulo.  
LPAD (cad1, n[,cad2])= Añade caracteres a la izquierda de la cadena hasta que tiene una cierta longitud.  
RPAD (cad1, n[,cad2])= Añade caracteres a la derecha de la cadena hasta que tiene una cierta longitud.  
LTRIM (cad [,set])= Suprime un conjunto de caracteres a la izquierda de la cadena.  
RTRIM (cad [,set])= Suprime un conjunto de caracteres a la derecha de la cadena.  
REPLACE (cad, cadena\_búsqueda [, cadena\_sustitucion])= Sustituye un carácter o caracteres de una cadena con 0 o mas caracteres.  
SUBSTR (cad, m [,n])= Obtiene parte de una cadena.  
TRANSLATE (cad1, cad2, cad3)= Convierte caracteres de una cadena en caracteres diferentes, según un plan de sustitución marcado por el usuario.

### Funciones que devuelven valores numéricos:

ASCII(cad)= Devuelve el valor ASCII de la primera letra de la cadena "cad".  
INSTR (cad1, cad2 [, comienzo [,m]])= Permite una búsqueda de un conjunto de caracteres en una cadena pero no suprime ningún carácter después.  
LENGTH (cad)= Devuelve el numero de caracteres de cad.

### Funciones para el manejo de fechas:

SYSDATE= Devuelve la fecha del sistema.  
ADD\_MONTHS (fecha, n)= Devuelve la fecha "fecha" incrementada en "n" meses.  
LASTDAY (fecha)= Devuelve la fecha del último día del mes que contiene "fecha".  
MONTHS\_BETWEEN (fecha1, fecha2)= Devuelve la diferencia en meses entre las fechas "fecha1" y "fecha2".  
NEXT\_DAY (fecha, cad)= Devuelve la fecha del primer día de la semana indicado por "cad" después de la fecha indicada por "fecha".

### Funciones de conversión:

TO\_CHAR= Transforma un tipo DATE ó NUMBER en una cadena de caracteres.  
TO\_DATE= Transforma un tipo NUMBER ó CHAR en DATE.  
TO\_NUMBER= Transforma una cadena de caracteres en NUMBER.

Artículo por **Agustin Jareño**.

## Agrupación y combinación de elementos con SQL

### Agrupación de elementos. Group by y Having:

Para saber cual es el salario medio de cada departamento de la tabla Jugadores sería:

```
SELECT EQUIPO_NO, AVG (SALARIO) "SALARIO MEDIO"  
FROM JUGADORES  
GROUP BY DEPT_NO;
```

La sentencia "Select" posibilita agrupar uno o más conjuntos de filas. El agrupamiento se lleva a cabo mediante la cláusula "GROUP BY" por las columnas especificadas y en el orden especificado. Formato:

```
SELECT...  
FROM...  
GROUP BY COLUMNA1, COLUMNA2, COLUMNAN...  
HAVING CONDICION  
GROUP BY ...
```

Los datos seleccionados en la sentencia "Select" que lleva el "Group By" deben ser:

- Una constante.
- Una función de grupo (SUM, COUNT, AVG...)
- Una columna expresada en el Group By.

La cláusula Group By sirve para calcular propiedades de uno o más conjuntos de filas. Si se selecciona más de un conjunto de filas, Group By controla que las filas de la tabla original sean agrupadas en un temporal.

La cláusula Having se emplea para controlar cual de los conjuntos de filas se visualiza. Se evalúa sobre la tabla que devuelve el Group By. No puede existir sin Group By.

Having es similar al Where, pero trabajo con grupos de filas; pregunta por una característica de grupo, es decir, pregunta por los resultados de las funciones de grupo, lo cual Where no puede hacer.

### Combinación externa (outer joins):

Nos permite seleccionar algunas filas de una tabla aunque estas no tengan correspondencia con las filas de la otra tabla con la que se combina. Formato:

```
SELECT TABLA1.COLUMNA1, TABLA1.COLUMNA2, TABLA2.COLUMNA1, TABLA2.COLUMNA2  
FROM TABLA1, TABLA2  
WHERE TABLA1.COLUMNA1 = TABLA2.COLUMNA1 (+);
```

Esto selecciona todas las filas de la tabla "tabla1" aunque no tengan correspondencia con las filas de la tabla "tabla2", se utiliza el símbolo +.

El resto de columnas de la tabla "tabla2" se rellena con NULL.

### Union, intersec y minus:

Permite combinar los resultados de varios "Select" para obtener un único resultado. Formato:

```
SELECT... FROM... WHERE...  
OPERADOR_DE_CONJUNTO  
SELECT...FROM...WHERE...
```

UNION= Combina los resultados de dos consultas. Las filas duplicadas que aparecen se reducen a una fila única.

UNION ALL= Como la anterior pero aparecerán nombres duplicados.

INTERSEC= Devuelve las filas que son iguales en ambas consultas. Todas las filas duplicadas serán eliminadas.

MINUS= Devuelve aquellas filas que están en la primera "Select" y no están en la segunda "Select". Las filas duplicadas del primer conjunto se reducirán a una fila única antes de que empiece la comparación con el otro conjunto.

Reglas para la utilización de operadores de conjunto:

- Las columnas de las dos consultas se relacionan en orden, de izquierda a derecha.
- Los nombres de columna de la primera sentencia "Select" no tiene porque ser los mismos que los nombres de la segunda.
- Los "Select" necesitan tener el mismo numero de columnas.
- Los tipos de datos deben coincidir, aunque la longitud no tiene que ser la misma.

*Artículo por **Agustin Jareño**.*

## Manipulación de datos con SQL

### Insert, Update y Delete:

Insert:

Se añaden filas de datos en una tabla:

```
INSERT INTO NOMBRETABLA [(COL [,COL]...)]  
VALUES (VALOR [,VALOR]...);
```

Nombretabla= Es la tabla en la que se van a insertar las filas.

Propiedades:

- Si las columnas no se especifican en la cláusula Insert se consideran, por defecto, todas las columnas de la tabla.
- Las columnas a las que damos valores se identifican por su nombre.
- La asociación columna valor es posicional.
- Los valores que se dan a las columnas deben coincidir con el tipo de dato definido en la columna.
- Los valores constantes de tipo carácter han de ir encerrados entre comillas simples ( ' ' )

(los de tipo fecha también).

Con Select:

Se añaden tantas filas como devuelva la consulta:

```
INSERT INTO NOMBRETABLA [(COL [,COL]...)]
SELECT {COLUMNA [, COLUMNA]... | *}
FROM NOMBRETABLA2 [CLAUSULAS DE SELECT];
```

Update:

Actualiza los valores de las columnas para una o varias filas de una tabla:

```
UPDATE NOMBRETABLA
SET COLUMNA1= VALOR1, ..., COLUMNAN= VALORN
WHERE CONDICION;
```

Set= Indica las columnas que se van a actualizar y sus valores.

Con Select:

Cuando la subconsulta (orden select) forma parte de SET, debe seleccionar el mismo número de columnas, (con tipos de datos adecuados) que los que hay entre paréntesis al lado de SET.

```
UPDATE NOMBRETABLA
SET COLUMNA= VALOR1, COLUMNA2= VALOR2, ...
WHERE COLUMNA3= (SELECT...)
```

Ó

```
UPDATE NOMBRETABLA
SET (COLUMNA1, COLUMNA2, ...)= (SELECT ...)
WHERE CONDICION;
```

Delete:

Elimina una o varias filas de una tabla:

```
DELETE [FROM] NOMBRETABLA
WHERE CONDICION;
```

*Artículo por **Agustin Jareño**.*

## **Claves primarias con SQL con Oracle**

**Rollback:**

Permite ir hasta el último COMMIT hecho o en su defecto hasta el comienzo de las órdenes con lo que estas no se ejecutan.

**Commit:**

Cuando ejecutamos ordenes estas no son creadas en la tabla hasta que ponemos este orden, por tanto los cambios realizados se perderán si al salir del programa no realizamos esta acción. Puede programarse para que lo haga automáticamente.

Algunas ordenes que lleven COMMIT implícito:

- QUIT
- EXIT
- CONNECT
- DISCONNECT
- CREATE TABLE
- CREATE VIEW
- GRANT
- REVOKE
- DROP TABLE
- DROP VIEW
- ALTER
- AUDIT
- NO AUDIT

### Creacion de una tabla:

Su primer carácter debe ser alfabético y el resto pueden ser letras, números y el carácter subrayado.

```
CREATE TABLE NOMBRETABLA
(COLUMNA1 TIPO_DATO {NOT NULL},
COLUMNA2 TIPO_DATO {NOT NULL},
...
) TABLESPACE ESPACIO_DE_TABLA;
```

Características:

- Las definiciones individuales de columnas se separan mediante comas.
- No se pone coma después de la última definición de columna.
- Las mayúsculas y minúsculas son indiferentes.

Los usuarios pueden consultar las tablas creadas por medio de la vista USER\_TABLES.

### Integridad de datos:

La integridad hace referencia al hecho de que los datos de la base de datos han de ajustarse a restricciones antes de almacenarse en ella. Una restricción de integridad será: Una regla que restringe el rango de valores para una o más columnas en la tabla.

### Restricciones en create table:

Usamos la cláusula CONSTRAINT, que puede restringir una sola columna o un grupo de columnas de una misma tabla.

Hay dos modos de especificar restricciones:

- Como parte de la definición de columnas.
- Al final, una vez especificados todas las columnas.

Formato:

```
CREATE TABLE NOMBRE_TABLA
(COLUMNA1 TIPO_DE_DATO
 {CONSTRAINT NOMBRE_RESTRICCION}
 {NOT NULL}
 {UNIQUE}
 {PRIMARY KEY}
 {DEFAULT VALOR}
 {REFERENCES NOMBRETABLA [(COLUMNA, [,COLUMNA)]
 {ON DELETE CASCADE}}}
 {CHECK CONDICION},
 COLUMNA2...
 )
 {TABLESPACE ESPACIO_DE_TABLA} ;
CREATE TABLE NOMBRE_TABLA
(COLUMNA1 TIPO_DATO ,
 COLUMNA2 TIPO_DATO,
 COLUMNA3 TIPO_DATO,
 ...
 {CONSTRAINT NOMBRE_RESTRICCION}
 [{UNIQUE} | {PRIMARY KEY} (COLUMNA [, COLUMNA])],
 {CONSTRAINT NOMBRE_RESTRICCION}
 {FOREIGN KEY (COLUMNA [, COLUMNA])
 REFERENCES NOMBRETABLA {(COLUMNA [,
 COLUMNA])
 {ON DELETE CASCADE}}},
 {CONSTRAINT NOMBRE_RESTRICCION}
 {CHECK (CONDICION)}
 ...
 ) [TABLESPACE ESPACIO_DE_TABLA];
```

*Artículo por **Agustín Jareño**.*

## **Definición de claves para tablas y restricciones**

### **Clave primaria: Primary key**

Es una columna o un conjunto de columnas que identifican unívocamente a cada fila. Debe ser única, no nula y obligatoria. Como máximo, podemos definir una clave primaria por tabla. Esta clave se puede referenciar por una columna o columnas. Cuando se crea una clave primaria, automáticamente se crea un índice que facilita el acceso a la tabla.

Formato de restricción de columna:

```
CREATE TABLE NOMBRE_TABLA
(COL1 TIPO_DATO [CONSTRAINT NOMBRE_RESTRICCION] PRIMARY KEY
 COL2 TIPO_DATO
 ...
 ) [TABLESPACE ESPACIO_DE_TABLA];
```

Formato de restricción de tabla:

```
CREATE TABLE NOMBRE_TABLA
(COL1 TIPO_DATO,
 COL2 TIPO_DATO,
 ...
```

```
[CONSTRAINT NOMBRE Restriccion] PRIMARY KEY (COLUMNA [,COLUMNA]),  
...  
) [TABLESPACE ESPACIO_DE_TABLA];
```

### Claves ajenas: Foreign Key:

Esta formada por una o varias columnas que están asociadas a una clave primaria de otra o de la misma tabla. Se pueden definir tantas claves ajenas como se precise, y pueden estar o no en la misma tabla que la clave primaria. El valor de la columna o columnas que son claves ajenas debe ser: NULL o igual a un valor de la clave referenciada (regla de integridad referencial).

Formato de restricción de columna:

```
CREATE TABLE NOMBRE_TABLA  
(COLUMNA1 TIPO_DATO  
 [CONSTRAINT NOMBRE Restriccion]  
 REFERENCES NOMBRE TABLA [(COLUMNA)] [ON DELETE CASCADE]  
 ...  
) [TABLESPACE ESPACIO_DE_TABLA];
```

Formato de restricción de tabla:

```
CREATE TABLE NOMBRE_TABLA  
(COLUMNA1 TIPO_DATO,  
 COLUMNA2 TIPO_DATO,  
 ...  
 [CONSTRAINT NOMBRE Restriccion]  
 FOREIGN KEY (COLUMNA [,COLUMNA])  
   REFERENCES NOMBRE TABLA [(COLUMNA [,  
   COLUMNA])]  
   [ON DELETE CASCADE],  
) [TABLESPACE ESPACIO_DE_TABLA];
```

Notas:

- En la cláusula REFERENCES indicamos la tabla a la cual remite la clave ajena.
- Hay que crear primero una tabla y después aquella que le hace referencia.
- Hay que borrar primero la tabla que hace referencia a otra tabla y después la tabla que no hace referencia.
- Borrado en cascada (ON DELETE CASCADE): Si borramos una fila de una tabla maestra, todas las filas de la tabla detalle cuya clave ajena sea referenciada se borrarán automáticamente. La restricción se declara en la tabla detalle. El mensaje "n filas borradas" solo indica las filas borradas de la tabla maestra.

NOT NULL: Significa que la columna no puede tener valores nulos.

DEFAULT: Le proporcionamos a una columna un valor por defecto cuando el valor de la columna no se especifica en la cláusula INSERT. En la especificación DEFAULT es posible incluir varias expresiones: constantes, funciones SQL y variables UID y SYSDATE.

Verificación de restricciones: CHECK: Actúa como una cláusula where. Puede hacer referencia a una o más columnas, pero no a valores de otras filas. En una cláusula CHECK no se pueden incluir subconsultas ni las pseudoconsultas SYSDATE, UID y USER.

**Nota:** La restricción NOT NULL es similar a CHECK (NOMBRE\_COLUMNA IS NOT NULL)

UNIQUE: Evita valores repetidos en la misma columna. Puede contener una o varias columnas.

Es similar a la restricción PRIMARY KEY, salvo que son posibles varias columnas UNIQUE definidas en una tabla. Admite valores NULL. Al igual que en PRIMARY KEY, cuando se define una restricción UNIQUE se crea un índice automáticamente.

### **Vistas del diccionario de datos para las restricciones:**

Contienen información general las siguientes:

USER\_CONSTRAINTS: Definiciones de restricciones de tablas propiedad del usuario.

ALL\_CONSTRAINTS: Definiciones de restricciones sobre tablas a las que puede acceder el usuario.

DBA\_CONSTRAINTS: Todas las definiciones de restricciones sobre todas las tablas.

### **Creación de una tabla con datos recuperados en una consulta:**

CREATE TABLE: permite crear una tabla a partir de la consulta de otra tabla ya existente. La nueva tabla contendrá los datos obtenidos en la consulta. Se lleva a cabo esta acción con la cláusula AS colocada al final de la orden CREATE TABLE.

```
CREATE TABLE NOMBRETABLA  
(COLUMNA [,COLUMNA]  
) [TABLESPACE ESPACIO_DE_TABLA]  
AS CONSULTA;
```

No es necesario especificar tipos ni tamaño de las consultas, ya que vienen determinadas por los tipos y los tamaños de las recuperadas en la consulta.

La consulta puede tener una subconsulta, una combinación de tablas o cualquier sentencia select valida.

Las restricciones CON NOMBRE no se crean en una tabla desde la otra, solo se crean aquellas restricciones que carecen de nombre.

*Artículo por **Agustín Jareño**.*

## **Supresión y modificación de tablas con SQL**

### **Supresión de tablas:**

DROP TABLE: suprime una tabla de la base de datos. Cada usuario puede borrar sus propias tablas, pero solo el administrador o algún usuario con el privilegio "DROP ANY TABLE" puede borrar las tablas de otro usuario. Al suprimir una tabla también se suprimen los índices y los privilegios asociados a ella. Las vistas y los sinónimos creados a partir de esta tabla dejan de funcionar pero siguen existiendo en la base de datos por tanto deberíamos eliminarlos.

Ejemplo:

```
DROP TABLE [USUARIO].NOMBRETABLA [CASCADE CONSTRAINTS];
```

TRUNCATE: permite suprimir todas las filas de una tabla y liberar el espacio ocupado para otros usos sin que reaparezca la definición de la tabla de la base de datos. Una orden TRUNCATE no se puede anular, como tampoco activa disparadores DELETE.

```
TRUNCATE TABLE [USUARIO.]NOMBRETABLA [{DROP | REUSE} STORAGE];
```

### Modificación de tablas:

Se modifican las tablas de dos formas: Cambiando la definición de una columna (MODIFY) ó añadiendo una columna a una tabla existente (ADD):

Formato:

```
ALTER TABLE NOMBRETABLA  
{[ADD (COLUMNA [,COLUMNA]...)]  
[MODIFY (COLUMNA [,COLUMNA]...)]  
[ADD CONSTRAINT RESTRICCIÓN]  
[DROP CONSTRAINT RESTRICCIÓN]};
```

ADD= Añade una columna o mas al final de una tabla.

MODIFY= Modifica una o mas columnas existentes en la tabla.

ADD CONSTRAINT= Añade una restricción a la definición de la tabla.

DROP CONSTRAINT= Elimina una restricción de la tabla.

A la hora de añadir una columna a una tabla hay que tener en cuenta:

- Si la columna no esta definida como NOT NULL se le puede añadir en cualquier momento.
- Si la columna esta definida como NOT NULL se pueden seguir estos pasos:
  1. Se añade una columna sin especificar NOT NULL.
  2. Se da valor a la columna para cada una de las filas.
  3. Se modifica la columna NOT NULL.

Al modificar una columna de una tabla se han de tener en cuenta:

- Se puede aumentar la longitud de una columna en cualquier momento.
- Es posible aumentar o disminuir el numero de posiciones decimales en una columna de tipo NUMBER.
- Si la columna es NULL en todas las filas de la tabla, se puede disminuir la longitud y modificar el tipo de dato
- La opción MODIFY... NOT NULL solo será posible cuando la tabla no contenga ninguna fila con valor nulo en la columna que se modifica.

### Adición de restricciones:

Con la orden ALTER TABLE se añaden restricciones a una tabla.

Formato:

```
ALTER TABLE NOMBRETABLA  
ADD CONSTRAINT NOMBRECONSTRAINT...
```

### Borrado de restricciones:

La orden ALTER TABLE con la cláusula DROP CONSTRAINT; con la que se borran las restricciones con nombre y las asignadas por el sistema. Formato:

```
ALTER TABLE NOMBRETABLA  
DROP CONSTRAINT NOMBRE_CONSTRAINT,  
NOMBRE_RESTRICCIÓN;
```

Artículo por Agustin Jareño.

## Gestión de vistas en SQL

### Creación y uso de vistas

No contienen información por si mismas, sino que están basadas en las que contienen otras tablas y refleja los datos de estas.

Si se suprime una tabla la vista asociada se invalida. Formato:

```
CREATE [OR REPLACE] VIEW NOMBREVISTA  
[(COLUMNA [,COLUMNA])]  
AS CONSULTA;
```

AS CONSULTA= Determina las columnas y las tablas que aparecerán en la vista.

[OR REPLACE]= Crea de nuevo la vista si ya existía.

Para consultar la vista creada, USER\_VIEWS:

```
SELECT VIEW_NAME FROM...
```

**Nota:** al borrar las tablas, las vistas de esas tablas no se borran y quedan inutilizadas.

### Borrado de vistas

Se hace con DROP VIEW. Formato:

```
DROP VIEW NOMBREVISTA;
```

### Operaciones sobre vistas

Se pueden realizar las mismas operaciones que se hacen sobre las tablas. Restricciones:

- Actualización Si una vista esta basada en una sola tabla, se pueden modificar las filas de la vista.
- La modificación de la vista cambia la tabla sobre la que esta definida.
- Borrado de filas a través de una vista= Para borrar filas de una tabla a través de una vista, esta se debe crear:
  - Con filas de una sola tabla.
  - Sin utilizar la cláusula GROUP BY ni DISTINCT.
  - Sin usar funciones de grupo o referencias a pseudocolumnas.
- Actualización de filas a través de una vista: Para actualizar filas en una tabla a través de una vista, esta ha de estar definida según las restricciones anteriores y , además, ninguna de las columnas que se va a actualizar se habrá definido como una expresión.
- Inserción de filas a través de una vista: Para insertar filas en una tabla a través de una vista se han de tener en cuenta todas las restricciones anteriores y , además, todas las columnas obligatorias de la tabla asociada deben estar presentes en la vista.
- Manejo de expresiones y de funciones en vistas: Se pueden crear vistas usando funciones, expresiones en columnas y consultas avanzadas pero únicamente se pueden consultar estas vistas. También podemos modificar filas siempre y cuando la columna

que se va a modificar no sea la columna expresada en forma de cálculo o con funciones.

**Nota:** No es posible insertar filas si las columnas de la vista contiene cálculos o funciones.

## Cambios de nombre

RENAME cambia el nombre de una tabla, vista o sinónimo. El nuevo nombre no puede ser una palabra reservada ni el nombre de un objeto que tenga creado el usuario. Las restricciones de integridad, los índices y los permisos dados al objeto se transfieren automáticamente al nuevo objeto.

```
RENAME NOMBRE_ANTERIOR TO NOMBRE_NUEVO;
```

Con esta orden no podemos renombrar columnas de una tabla, estas se renombran mediante CREATE TABLE AS...

*Artículo por **Agustin Jareño**.*

## Usuarios en Oracle

Es un nombre definido en la base de datos que se puede conectar a ella y acceder a determinados objetos según ciertas condiciones que establece el administrador.

Los objetos del diccionario de datos a los que un usuario puede acceder se encuentran en la vista DICTIONARY, que es propiedad del usuario SYS.

```
DESC DICTIONARY;
```

Con la orden:

```
SELECT TABLENAME FROM DICTIONARY;
```

Se visualizan los objetos del diccionario de datos a los que se puede acceder.

### Creación de usuarios:

```
CREATE USER NOMBRE_USUARIO  
IDENTIFIED BY CLAVE_ACCESO  
[DEFAULT TABLESPACE ESPACIO_TABLA]  
[TEMPORARY TABLESPACE ESPACIO_TABLA]  
[QUOTA {ENTERO {K | M} | UNLIMITED } ON ESPACIO_TABLA]  
[PROFILE PERFIL];
```

DEFAULT TABLESPACE= Asigna a un usuario el tablespace por defecto para almacenar los objetos que cree. Si no se asigna ninguno, el tablespace por defecto es SYSTEM.

TEMPORARY TABLESPACE= Especifica el nombre del tablespace para trabajar temporales. Si no se especifica ninguno, el tablespace por defecto es SYSTEM.

QUOTA= Asigna un espacio en megabytes o kilobytes en el tablespace asignado. Si no se especifica esta cláusula, el usuario no tiene cuota asignada y no podrá crear objetos en el tablespace. Para tener espacio y acceso ilimitado a un tablespace es:

```
GRANT UNLIMITED TABLESPACE NOMBRE_TABLESPACE;
```

PROFILE= Asigna un perfil a un usuario.

### Modificación de usuarios:

```
ALTER USER NOMBRE_USUARIO
IDENTIFIED BY CLAVE_ACCESO
[DEFAULT TABLESPACE ESPACIO_TABLA]
[TEMPORARY TABLESPACE ESPACIO_TABLA]
[QUOTA {ENTERO {K | M } | UNLIMITED } ON ESPACIO_TABLA]
[PROFILE PERFIL];
```

### Borrado de usuarios:

```
DROP USER USUARIO [CASCADE];
```

CASCADE= Suprime todos los objetos del usuario antes de borrarlo.

*Artículo por **Agustín Jareño**.*

## Gestión en Oracle con SQL

### Privilegios

Es la capacidad de un usuario dentro de la base de datos a realizar determinadas operaciones o acceder a determinados objetos de otros usuarios.

### Privilegios sobre los objetos

Nos permite acceder y realizar cambios en los datos de otros usuarios. Ejemplo: El privilegio de consultar la tabla de otro usuario es un privilegio sobre objetos.

```
GRANT {PRIV_OBJETO [, PRIV_OBJETO]... | ALL [PRIVILEGES]}
[(COL [,COL]...)]
ON [USUARIO] OBJETO
TO {USUARIO | ROL | PUBLIC} [, {USUARIO | ROL | PUBLIC}...]
[WITH GRANT OPTION];
```

ON= Especifica el objeto sobre el que se dan los privilegios.

TO= Identifica a los usuarios o roles a los que se conceden los privilegios.

ALL= Concede todos los privilegios sobre el objeto especificado.

WITH GRANT OPTION= Permite que el receptor del privilegio o rol se lo asigne a otros usuarios o roles.

PUBLIC= Asigna los privilegios a todos los usuarios actuales y futuros: El propósito principal del grupo PUBLIC es garantizar el acceso a determinados objetos a todos los usuarios de la base de datos.

### Privilegios de sistema

Dan derecho a ejecutar un tipo de comando SQL o a realizar alguna acción sobre objetos de un tipo especificado. Por ejemplo, el privilegio para crear TABLESPACES es un privilegio de

sistema. Formato:

```
GRANT {PRIVILEGIO | ROL} [, {PRIVILEGIO | ROL}, ...]  
TO {USUARIO | ROL | PUBLIC} [, {USUARIO | ROL | PUBLIC}]  
[WITH ADMIN OPTION];
```

WITH ADMIN OPTION= Permite que el receptor del privilegio o rol pueda conceder esos mismos privilegios a otros usuarios o roles.

### Retirada de privilegios de objetos a los usuarios

```
REVOKE {PRIV_OBJETO [,PRIV_OBJETO]... | ALL [PRIVILEGES]}  
ON [USUARIO.]OBJETO  
FROM {USUARIO | ROL | PUBLIC} [, {USUARIO | ROL | PUBLIC}]...;
```

### Retirada de privilegios de sistema o roles a los usuarios

```
REVOKE {PRIV_SISTEMA | ROL} [, {PRIV_SISTEMA | ROL}]...  
FROM {USUARIO | ROL | PUBLIC} [, {USUARIO | ROL | PUBLIC}]...;
```

### Roles

Conjunto de privilegios agrupados. Formato:

```
CREATE ROLE NOMBREROL [IDENTIFIED BY CONTRASEÑA];
```

**Nota:** Un rol puede decidir el acceso de un usuario a un objeto, pero no puede permitir la creación de objetos.

### Supresión de privilegios en los roles

```
REVOKE NOMBREPRIVILEGIO ON NOMBRETABLA FROM NOMBREROL;
```

```
REVOKE NOMBREPRIVILEGIO FROM NOMBREROL;
```

### Supresión de un rol

```
DROP ROLE NOMBREROL;
```

### Establecer un rol por defecto

```
ALTER USER NOMBREUSUARIO  
DEFAULT {[ROLE NOMBRE_ROL] | [NONE]};
```

NONE= Hace que el usuario no tenga rol por defecto.

### Perfiles:

Conjunto de límites a los recursos de la base de datos:

```
CREATE PROFILE NOMBREPERFIL LIMIT  
{NOMBRE DE LOS LÍMITES}  
{ENTERO [K | M] | UNLIMITED | DEFAULT };
```

UNLIMITED= No hay límites sobre un recurso en particular.

DEFAULT= Coge el limite del perfil default.

### Borrado de un perfil:

```
DROP FILE NOMBREPERFIL [CASCADE];
```

### Gestión de tablespaces

Un tablespace es una unidad lógica de almacenamiento d datos representada físicamente por uno o más archivos de datos. Se recomienda no mezclar datos de diferentes aplicaciones en un mismo tablespace.

### Para crear un tablespace

```
CRATE TABLESPACE NOMBRETABLESPACE  
DATAFILE 'NOMBREARCHIVO' [SIZE ENTERO [K | M] [REUSE]  
[DEFAULT STORAGE  
(INITIAL TAMAÑO  
MINEXTENTS TAMAÑO  
MAXEXTENTS TAMAÑO  
PCTINCREASE VALOR  
)]  
[ONLINE | OFFLINE];
```

REUSE= Reutiliza el archivo si ya existe o lo crea si no existe.

DEFAULT STORAGE= Define el almacenamiento por omisión para todos los objetos que se creen en este espacio de la tabla. Fija la cantidad de espacio si no se especifica en la sentencia CREATE TABLE.

### Modificación de tablespaces

```
ALTER TABLESPACE NOMBRETABLESPACE  
{[ADD DATAFILE 'NOMBREARCHIVO' [SIZE ENTERO [K | M] [REUSE]  
[AUTOEXTEND ON... | OFF]  
]  
[REANME DATAFILE 'ARCHIVO' [, 'ARCHIVO']...  
TO 'ARCHIVO' [, 'ARCHIVO']]  
[DEFAULT STORAGE CLAUSULAS_ALMACENAMIENTO]  
[ONLINE | OFFLINE]  
};
```

ADD\_DATAFILE= Añade al tablespace uno o varios archivos.

AUTOEXTEND= Activa o desactiva el crecimiento automático de los archivos de datos del tablespace. Cuando un tablespace se llena podemos usar esta opción para que el tamaño del archivo o archivos de datos asociados crezca automáticamente.

Autoextend off: desactiva el crecimiento automático.

RENAME\_DATAFILE= Cambia el nombre de un archivo existente del tablespace. Este cambio se tiene que hacer desde el sistema operativo y, después, ejecutar la orden SQL.

### Borrado de tablespaces

```
DROP TABLESPACE NOMBRETABLESPACE  
[INCLUDING CONTENTS];
```

INCLUDING CONTENTS= Permite borrar un tablespace que tenga datos. Sin esta opción solo se puede suprimir un tablespace vacío.

Se recomienda poner el tablespace offline antes de borrarlo para asegurarse de que no haya sentencias SQL que estén accediendo a datos del tablespace, en cuyo caso no sería posible borrarlo.

Cuando se borra un tablespace los archivos asociados no se borran del sistema operativo, por lo que tendremos que borrarlos de forma manual.

*Artículo por **Agustin Jareño**.*

## **Optimizar consultas SQL**

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 trucos 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 pensárselo 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 índices compuestos es mejor utilizar todos los campos de todos los índices. 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 óptima 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 por que 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 cláusula 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 matrículas 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.

*Artículo por **Claudio***

## Consultas de selección

Este conjunto de registros puede ser 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, Teléfono
FROM
  Clientes
```

Esta sentencia devuelve un conjunto de resultados 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 cláusula 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 cláusula (ASC - se toma este valor por defecto) ó descendente (DESC)

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

## Uso de Índices 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 índice 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 cláusula y el primer nombre del campo a recuperar, los posibles predicados son:

<b>Predicado</b>	<b>Descripción</b>
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
DISTINCTOW	Omite los registros duplicados basándose 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 y devuelve todos y cada uno de sus campos. No es 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 de 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 entero 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 campos 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 devuelve 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 porque estamos recuperando datos de diferentes tablas y resultan tener un campo con igual nombre. 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 (1) AS Empleado
FROM
  Empleados Trabajadores
```

(1) 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 externas. Es 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
  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
```

*Artículo por **Claudio***

## Criterios de selección en SQL

En el apartado 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 apartado se estudiarán las posibilidades de filtrar los registros con el fin de recuperar solamente aquellos que cumplan unas 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
SQL-SERVER	Fecha = #mm-dd-aaaa#
ORACLE	Fecha = to_date('YYYYDDMM','aaaammdd',)

ACCESS Fecha = #mm-dd-aaaa#

### Ejemplo

Banco de Datos Ejemplo (para grabar la fecha 18 de mayo de 1969)

SQL-SERVER Fecha = #05-18-1969# ó  
 Fecha = 19690518  
 ORACLE Fecha = to\_date('YYYYDDMM', '19690518')  
 ACCESS 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 antepone 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 antepone 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)
```

## 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 puede utilizar una cadena de 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.

### ACCESS

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'

### 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

En determinados motores de bases de datos, esta cláusula, no reconoce el asterisco como carácter comodín y hay que sustituirlo por el carácter tanto por ciento (%).

## 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. Después de escribir esta cláusula se deben especificar las condiciones expuestas en los apartados anteriores. 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
  IdProducto, Existencias
FROM
  Productos
WHERE
  Existencias <= NuevoPedido
```

```
SELECT *
FROM
  Pedidos
WHERE
  FechaEnvio = #05-30-1994#
```

```
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
  Empleados
WHERE
  Apellidos Between 'Lon' And 'Tol'
```

```
SELECT
  IdPedido, FechaPedido
FROM
  Pedidos
WHERE
  FechaPedido Between #01-01-1994# And #12-31-1994#
```

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

Artículo por **Claudio**

## **Criterios de selección en SQL II**

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 está 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
  IdFamilia, Sum(Stock) AS StockActual
FROM
  Productos
GROUP BY
  IdFamilia
```

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
  IdFamilia, Sum(Stock) AS StockActual
FROM
  Productos
GROUP BY
  IdFamilia
HAVING
  StockActual > 100
AND
  NombreProducto Like BOS*
```

## AVG

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

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, Min

Devuelven el mínimo o el máximo de un conjunto de valores contenidos en un campo específico 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, StDevP

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 Desviación
FROM
  Pedidos
WHERE
  País = 'España'
```

```
SELECT
  StDevP(Gastos) AS Desviación
FROM
  Pedidos
WHERE
  País = 'España'
```

## Sum

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

Sum(expr)

En donde expr representa 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, VarP

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
  País = 'España'
```

```
SELECT
  VarP(Gastos) AS Varianza
FROM
  Pedidos
WHERE
  País = '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
```

*Artículo por **Claudio***

## 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. Tanto las sentencias de actualización como las de borrado desencadenarán (según el motor de datos) las actualizaciones en cascada, borrados en cascada, restricciones y valores por defecto definidos para los diferentes campos o tablas afectadas por la consulta.

### 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.

#### Para 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.

#### 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.

### Para 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 Tabla Origen
```

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 Tabla Origen poseen la misma estructura podemos simplificar la sintaxis a:

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

De esta forma los campos de Tabla Origen se grabarán en Tabla, para realizar esta operación es necesario que todos los campos de Tabla Origen estén contenidos con igual nombre en Tabla. Con otras palabras que Tabla posea todos los campos de Tabla Origen (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 incrementándose 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 especifica 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 asegurarse que es única, y con valores no nulos; 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.

## Ejemplos

```
INSERT INTO
  Clientes
SELECT
  ClientesViejos.*
FROM
  ClientesNuevos
```

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

Esta consulta crea una tabla nueva llamada programadores con igual estructura que la tabla empleado y copia aquellos registros cuyo campo categoria se programador

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

```
INSERT INTO
  Empleados
SELECT
  Vendedores.*
FROM
  Vendedores
WHERE
  Provincia = 'Madrid'
```

## 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
```

*Artículo por **Claudio***

## **Tipos de datos SQL**

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

<b>Tipo de Dato</b>	<b>Longitud</b>	<b>Descripción</b>

<b>s</b>		
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.
DATE TIME	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*1038 a -1.401298*10-45 para valores negativos, 1.401298*10-45 a 3.402823*1038 para valores positivos, y 0.
DOUBLE	8 bytes	Un valor en punto flotante de doble precisión con un rango de -1.79769313486232*10308 a -4.94065645841247*10-324 para valores negativos, 4.94065645841247*10-324 a 1.79769313486232*10308 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.
LONG TEXT	1 byte por carácter	De cero a un máximo de 1.2 gigabytes.
LONG BINARY	Según se necesite	De cero 1 gigabyte. Utilizado para objetos OLE.
TEXT	1 byte por carácter	De cero a 255 caracteres.

La siguiente tabla recoge los sinónimos 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 IEEE SINGLE REAL
DOUBLE	FLOAT FLOAT8 IEEE DOUBLE NUMBER NUMERIC
SHORT	INTEGER2 SMALLINT
LONG	INT INTEGER INTEGER4
LONG BINARY	GENERAL OLEOBJECT
LONG TEXT	LONGCHAR MEMO NOTE
TEXT	ALPHANUMERIC CHAR - CHARACTER STRING - VARCHAR
VARIANT (No Admitido)	VALUE

Artículo por **Claudio**

## Subconsultas en SQL

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 SQL, rodeada por 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.

```
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 DISTINCT
NombreProducto, Precio_Unidad
FROM
Productos
WHERE
PrecioUnidad =
(
SELECT
PrecioUnidad
FROM
Productos
WHERE
NombreProducto = '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 DISTINCT
NombreContacto, NombreCompania, CargoContacto, Telefono
FROM
Clientes
WHERE
IdCliente IN (
SELECT DISTINCT IdCliente
FROM Pedidos
WHERE FechaPedido <#07/01/1993#
)
```

(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.IdEmpleado = E.IdEmpleado
)
```

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

```
SELECT DISTINCT
Pedidos.Id_Producto, Pedidos.Cantidad,
(
```

```
SELECT
  Productos.Nombre
FROM
  Productos
WHERE
  Productos.IdProducto = Pedidos.IdProducto
) 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	>	ALL (2,5,7)	Falso
3	<	ALL (5,6,7)	Falso

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

*Artículo por **Claudio***

## Consultas SQL de Unión Internas

### Consultas de Combinación entre tablas

Las vinculaciones entre tablas se realizan 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.
campo 1, campo 2	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: =, <, <>, <=, =>, ó >.

Se puede utilizar una operación INNER JOIN en cualquier cláusula FROM. Esto crea una combinación por equivalencia, conocida también como unión interna. Las combinaciones equivalentes 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
    NombreCategoria, NombreProducto
FROM
    Categorías
INNER JOIN
    Productos
ON
    Categorías.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, Categorías.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.campo3])
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.

Ejemplo:

```
SELECT DISTINCT
    Sum(PrecioUnitario * Cantidad) AS Sales,
    (Nombre + ' ' + Apellido) AS Name
FROM
    Empleados
INNER JOIN(
    Pedidos
INNER JOIN
    DetallesPedidos
ON
    Pedidos.IdPedido = DetallesPedidos.IdPedido)
ON
```

```
Empleados.IdEmpleado = Pedidos.IdEmpleado
GROUP BY
Nombre + ' ' + Apellido
```

(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 derecha. 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 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):

Autores	
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
    Hombres.Sexo = 'Hombre' AND
    Mujeres.Sexo = 'Mujer' AND
    Hombres.Id <> Mujeres.Id
```

Para concluir supongamos la tabla siguiente:

Empleados		
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
```

*Artículo por **Claudio***

## Consultas SQL 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:

consulta 1, consulta 2, consulta n
--

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
  NuevasCuentas
UNION ALL
SELECT *
FROM
  Clientes
WHERE
  CantidadPedidos > 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
  NombreCompania, Ciudad
FROM
  Proveedores
WHERE
  Pais = 'Brasil'
UNION
  SELECT NombreCompania, Ciudad
  FROM Clientes
  WHERE Pais = 'Brasil'
```

(Recupera los nombres y las ciudades de todos proveedores y clientes de Brasil)

```
SELECT
  NombreCompania, Ciudad
FROM
  Proveedores
WHERE
  Pais = 'Brasil'
UNION
  SELECT NombreCompania, Ciudad
  FROM Clientes
  WHERE Pais = '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
  NombreCompania, Ciudad
FROM
  Proveedores
WHERE
  Pais = 'Brasil'
UNION
SELECT NombreCompania, Ciudad
FROM Clientes
WHERE Pais = 'Brasil'
UNION
SELECT Apellidos, Ciudad
FROM Empleados
WHERE Region = '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_Clientes
UNION TABLE
  ListaProveedores
```

(Recupera los nombres y códigos de todos los proveedores y clientes)

*Artículo por **Claudio***

## Estructuras de las tablas en SQL

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:

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. ( <a href="#">Ver Tipos de Datos</a> )
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 índice a crear. Esta cláusula es opcional.

índice multicampos	Es una cláusula CONSTRAINT que define el tipo de índice multicampos a crear. Un índice multicampo es aquel que está indexado por el contenido de varios campos. Esta cláusula es 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 longitud 25 y otro llamado apellidos con longitud 50).

```
CREATE TABLE
  Empleados (
    Nombre TEXT (10),
    Apellidos TEXT,
    FechaNacimiento DATETIME
  )
CONSTRAINT
  IndiceGeneral
  UNIQUE (
    Nombre, Apellidos, FechaNacimiento
  )
```

(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 FechaNacimiento de tipo Fecha/Hora. También crea un índice único - no permite valores repetidos - formado por los tres campos.)

```
CREATE TABLE
  Empleados (
    IdEmpleado INTEGER CONSTRAINT IndicePrimario PRIMARY,
    Nombre TEXT,
    Apellidos TEXT,
    FechaNacimiento 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 FechaNacimiento de tipo Fecha/Hora y el campo IdEmpleado 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 [,...]])}

En donde:

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 está 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.

o que implica que los registros de la tabla no pueden contener el mismo valor en los campos indexados.

o los campos especificados. Todos los campos de la clave principal deben ser únicos y no nulos, cada tabla sólo puede

como valor del índice campos contenidos en otras tablas). Si la clave principal de la tabla externa consta de más de un campo todos los campos de referencia, el nombre de la tabla externa, y los nombres de los campos referenciados en la tabla externa. Si los campos referenciados son la clave principal de la tabla externa, no tiene que especificar los campos referenciados, sino el nombre de la tabla externa estuviera formada por los campos referenciados.

## Creación de Índices

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 es la siguiente:

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

En donde:

índice	Es el nombre del índice a crear.
tabla	Es el nombre de una tabla existente en la que se creará el índice.
campo	Es el nombre del campo o lista de campos que constituyen el índice.
ASC	Indica el orden de los valores de los campos ASC indica un orden ascendente

DESC	(valor predeterminado) y DESC un orden descendente.
UNIQUE	Indica que el índice no puede contener valores duplicados.
DISALLOW NULL	Prohíbe 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 índice que sea "Clave Principal". Si un índice es clave principal implica 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 tablas adjuntas 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 Teléfono.)
```

```
CREATE UNIQUE INDEX
    MiIndice
ON
    Empleados (IdEmpleado)
    WITH DISALLOW NULL
```

(Crea un índice en la tabla Empleados utilizando el campo IdEmpleado, obligando que el campo IdEmpleado no contenga valores nulos ni repetidos.)

## Modificar el Diseño de una Tabla

Modifica el diseño de una tabla ya existente, se pueden 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:

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	Es 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.

ripción

utiliza para añadir un nuevo campo a la tabla, indicando el nombre, el tipo de campo y opcionalmente el tamaño (para o ).

utiliza para agregar un índice de multicampos o de un único campo.

utiliza para borrar un campo. Se especifica únicamente el nombre del campo.

utiliza para eliminar un índice. Se especifica únicamente el nombre del índice a continuación de la palabra reservada CO

```
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
(IdEmpleado)
REFERENCES
Empleados (IdEmpleado)
(Agrega un índice externo a la tabla Pedidos. El índice externo se basa en el campo IdEmpleado y se refiere al campo IdEmpleado 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 índice de la tabla Pedidos.)
```

*Artículo por **Claudio***

## **Cursores en SQL**

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 recorrello**

```
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
SELECT au_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
```

*Artículo por **Claudio***

## **Referencias Cruzadas en SQL**

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 #01-01-1998# And #12-31-1998#
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 #01-01-1998# And #12-31-1998#
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:

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, Artículos
WHERE
  Pedidos.Id = Artículos.Id
GROUP BY
  Pedidos.Id, Artículos.Nombre
PIVOT
  Year(Fecha)
```

Y obtenemos el siguiente resultado:

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

Comentarios a la consulta:

La cláusula 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 basándose en 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 en la 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 mismo 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 de los que extraemos la información: pedidos.cantidad y artículos.nombre, por ello 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 a el 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)  
PIVOT Format([Fecha],"mmm") In ("Ene", "Feb", "Mar", "Abr", "May", "Jun", "Jul", "Ago", "Sep", "Oct", "Nov", "Dic");
3. Para agrupar por días  
PIVOT Format([Fecha],"Short Date");

*Artículo por **Claudio***

## **Introducción a freetext y contains en SQL-Server**

### **FULL TEXT SEARCH**

Se utilizan en campos de texto de gran tamaño utilizando unos índices denominados catálogos. Estos catálogos sólo se pueden utilizar con tablas que tengan definidas claves primarias y almacenan todas las palabras del contenido de las tablas a excepción de los artículos, preposiciones, etc.

Los catálogos no se actualizan automáticamente ni se guardan junto con la base de datos y cada tabla puede tener un único catálogo.

Para la utilización de estos catálogos dentro de una consulta, podemos utilizar dos métodos, el primero consiste en incluir los criterios dentro de una cláusula WHERE (CONTAINS ó FREETEXT) y la segunda es utilizando una tabla temporal que contiene el ratio de acierto en la consulta (CONTAINSTABLE ó FREETEXTTABLE).

### **El predicado CONTAINS**

Se utiliza este predicado para buscar un texto específico en una tabla. Su funcionamiento es similar al predicado LIKE, a diferencia que éste no puede realizar búsquedas en los campos grandes de texto. CONTAINS no diferencia entre mayúsculas y minúsculas.

### **Sintaxis:**

```
SELECT <Campos> FROM <Tabla>  
WHERE CONTAINS(<Campo>,<Cadena>) OR/AND CONTAINS(<Campo>,<Cadena>)
```

El predicado CONTAINS soporta sintaxis complejas para buscar:

- Una o más palabras utilizando los operadores lógicos AND/OR.
- Familias de palabras
- Una palabra o una frase que comiencen por un determinado texto.
- Palabras o frases que estén unas cerca de otra.

Para buscar una palabra en un campo:

```
SELECT title_id, title, notes FROM titles
```

```
WHERE CONTAINS(notes,'business')
```

Para localizar una frase en un campo:

```
SELECT title_id, titles, notes FROM titles
WHERE CONTAINS(notes, 'common business applications' )
```

Para localizar una frase en todos los campos habilitados:

```
SELECT title_id, titles, notes FROM titles
WHERE CONTAINS(*, 'common business applications' )
```

Utilizando AND, OR y NOT

```
SELECT title, notes FROM titles
WHERE CONTAINS(notes, 'favorite recipes' OR 'gourmet recipes' )
```

```
SELECT titles, notes FROM titles
WHERE CONTAINS(notes, 'cooking AND NOT ("computer*") ')
```

```
SELECT titles, notes FROM titles
WHERE CONTAINS(notes, 'beer AND ales ')
```

```
SELECT titles, notes FROM titles
WHERE CONTAINS(*, ('ice skating' OR hockey) AND NOT olympics')
```

Utilizando caracteres comodines

```
SELECT titles, notes FROM titles
WHERE CONTAINS(notes, 'ice*' )
```

```
SELECT titles, notes FROM titles
WHERE CONTAINS(notes, 'light bread*' )
```

Búsqueda de palabras o frases indicando la importancia de las palabras:

Esta búsqueda permite indicar el peso que tendrá cada una de las palabras o frases que se buscan sobre el resultado de la búsqueda, el peso oscila entre el valor más bajo 0.0 y el valor más alto 1.0.

```
SELECT Cliente, Nombre, Direccion FROM Cliente
WHERE CONTAINS (Direccion, 'ISABOUT ( "Calle*", Velazquez WEIGHT(0.5), Serrano(0.9)')
```

(Se encontrarán todos aquellos registros que en el campo dirección exista la cadena calle seguida de cualquier valor, ordenando primero los de "Calle Serrano", luego los de "Calle Velázquez" y después el resto.

Búsqueda de palabras cercanas:

Podemos realizar búsquedas por dos palabras e indicar que se encuentren próximas una de otra. El orden de las palabras no altera el resultado de la búsqueda.

```
SELECT titulo, notas FROM libros
WHERE CONTAINS (notas, "usuario NEAR computadora")
```

```
SELECT titulo, notas FROM libros
WHERE CONTAINS (notas, "usuario ~ computadora")
```

Se pueden indicar tres palabras, de tal forma que la segunda y la primera deben estar

próximas al igual que la segunda y la tercera.

```
SELECT titulo, notas FROM libros
WHERE CONTAINS (notas, "usuario ~ principiante ~ computadora")
```

Búsquedas con frases:

```
... WHERE CONTAINS(Descripcion, " salsas ~ "mezcl*" ")
```

```
... WHERE CONTAINS(Descripcion, " "carne*" ~ "salsa empanada*" ")
```

## El predicado FREETEXT

Al utilizar este predicado se analizan todas las palabras de las frases y nos devuelve como resultado, aquellos registros que contiene la frase completa o algún fragmento de la misma. La sintaxis es igual que el predicado CONTAINS.

```
... FREETEXT(descripcion, ` "En un lugar de la mancha de cuyo nombre no quiero acordarme" `)
```

## El predicado CONTAINSTABLE

Este predicado tiene igual funcionamiento y sintaxis que CONTAINS a diferencia que en este caso devuelve una tabla con dos columnas, la primera llamada [KEY] contiene el valor de la clave primaria de la tabla que la que buscamos, la segunda llamada RANK devuelve un valor indicando el porcentaje de acierto en la búsqueda para cada registro.

```
SELECT Preguntas.Pregunta, Preguntas.Respuesta, Resultado.RANK
FROM Preguntas, CONTAINSTABLE(Respuesta, " SQL Server") AS Resultado
WHERE Preguntas.IdPregunta = Resultado.[KEY]
ORDER BY Resultado.RANK Desc
```

```
SELECT Preguntas.Pregunta, Preguntas.Respuesta, Resultado.RANK
FROM Preguntas INNER JOIN CONTAINSTABLE (Respuesta, "SQL Server") AS Resultado
ON Preguntas.IdPregunta = Resultados.KEY
```

## El predicado FREETEXTTABLE

Es el equivalente a CONTAINSTABLE pero realizando búsquedas de FREETEXT.

*Artículo por **Claudio***

# Consultas e índices de texto I

## 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 la tabla 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 indexadas 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\Sqllserver\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 Microsoft® 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 Microsoft 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

Microsoft® 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 Microsoft® 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 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 &lt;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.

Microsoft® 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 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, y, 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? satisface la condición CONTAINS(testing, "fallo del equipo") El fallo de la búsqueda sería grave."

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. Para obtener más información, consulte sp\_dbcmptlevel.

La tabla devuelta por la función 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_dbcmtlevel`.

## 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.AI 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. Para obtener más información, consulte sp\_dbcmptlevel.

### 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
```

*Artículo por **Claudio***

## Consultas e índices de texto II

### 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
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 Microsoft® 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, Microsoft® 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. Microsoft® 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. Microsoft® 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.

*Artículo por **Claudio***

## Acceso a base de datos externas

Para el acceso a bases de datos externas se utiliza la cláusula IN. Se puede acceder a bases 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 'C:\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:\BTREIVE\DATOS\VENTAS\FILE.DDF' 'Btrieve;'
WHERE
  IDCliente Like 'A*'
```

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

*Artículo por **Claudio***

## Consultas con parámetros y omisión de permisos

### Consultas con parámetros

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:

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

Se pueden utilizar nombres pero no tipos de datos en una cláusula WHERE o HAVING.

```
PARAMETERS
```

```
PrecioMinimo Currency,  
FechaInicio DateTime;  
SELECT  
  IdPedido, Cantidad  
FROM  
  Pedidos  
WHERE  
  Precio = PrecioMinimo  
AND  
  FechaPedido = FechaInicio
```

## Omitir los permisos de acceso

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.

*Artículo por **Claudio***

## Procedures y búsqueda de registros duplicados en SQL

### Clausula Procedure

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:

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
```

```
ListaCategorias;
SELECT DISTINCTROW
  NombreCategoria, IdCategoria
FROM
  Categorias
ORDER BY
  NombreCategoria
(Asigna el nombre Lista_de_categorías a la consulta y la ejecuta.)
```

```
PROCEDURE
  Resumen
  FechaInicio DATETIME,
  FechaFinal DATETIME;
SELECT DISTINCTROW
  FechaEnvio, IdPedido, ImportePedido, Format(FechaEnvio, "yyyy") AS Año
FROM
  Pedidos
WHERE
  FechaEnvio Between FechaInicio And FechaFinal
(Asigna el nombre Resumen a la consulta e incluye dos parámetros.)
```

## 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 DISTINCT 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 DISTINCT
  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
```

*Artículo por **Claudio***

## Crear una cadena de conexión para SQL Server

En nuestro caso, crearemos una conexión para SQL Server 2005 EXPRESS

Sobre el escritorio, crearemos un nuevo documento de texto que le cambiaremos el nombre, por ejemplo: cadena.udl (es preciso que podamos ver las extensiones de los ficheros)

Una vez creado el fichero, pulsaremos sobre él para que se abra. Si tienes problemas para crear el fichero puedes descargarlo pulsando aquí.

En la pestaña de Proveedor seleccionaremos el Proveedor "Microsoft OLE DB Provider for SQL Server" y le daremos a siguiente.

En la pestaña de conexión escribiremos el nombre del servidor, y si precisa de autenticación específica le introduciremos el nombre de usuario y contraseña.

También podemos especificar en la cadena la base de datos.

Ahora ya podemos probar la conexión, pulsando el botón de "Probar conexión"

Si la prueba de conexión fue satisfactoria ya podemos cerrar el fichero y abrirlo con el notepad, para ver toda la cadena de conexión.

```
Provider=SQLOLEDB.1;Persist Security Info=False;User ID=sa;Initial Catalog=BDInicial;Data Source=Server
```

*Artículo por **Pol Salvat***

## **Problemas de Conexión con SQL Server 2005**

Después de instalar SQL Server 2005 EXPRESS y querer atacar con mi aplicación al servidor de SQL Server 2005 EXPRESS me daba error de conexión o acceso denegado al servidor.

Investigando he encontrado que el problema es de configuración del servidor, ya que por defecto no admite conexiones TCP/IP para conexiones remotas.

Para habilitar las conexiones remotas: Inicio, Todos los programas, Microsoft SQL Server 2005, Configuration Tools, SQL Server Configuration Manager.

De la lista de SQL Server Configuration Manager seleccionar SQL Server 2005 Network Configuration. Aparecerá Protocols for SQLEXPRESS (SQLEXPRESS porque es la version que hemos instalado).

En la derecha aparecen los protocolos para SQLEXPRESS:

Seleccionar el protocolo TCP/IP y darle doble clic. Hay que ponerlo Enabled = Yes.

Pulsamos en aplicar, y aparece el mensaje que es necesario reiniciar el servicio para que tenga efecto las modificaciones del protocolo TCP/IP.

Aceptamos y reiniciamos el servicio SQL SERVER EXPRESS

*Artículo por **Pol Salvat***

## La función datepart() en Access

Hace unos días tenía que hacer una consulta sobre fechas en Access: "Obtener el nombre de las empresas cuya fecha de alta coincidía con "x" año" y leí un artículo publicado en la esta web con el título: "Funciones para búsquedas con fechas en Access".

Tras leer este artículo supe de la existencia de la función DatePart() pero la forma de ponerla en la práctica tal y como el problema me lo planteaba, no es nada sencillo, o en ese momento, no lo vi claro. De hecho, haciendo una búsqueda, todo era bastante lioso implementando programitas en VBA (Visual Basic Applications). (Importante: El formato de fecha en Access es el formato americano: mm/dd/aaaa, pero en este caso el formato americano y el de la tabla en cuanto al año se refieren coinciden.)

Si realizamos la siguiente consulta:

```
SELECT DatePart("yyyy",FECHA_ALTA) FROM tabla1;
```

Obtenemos:

Hasta aquí todo bien, el problema surge cuando quieres obtener sólo un tipo de fecha en la que el año coincida con uno dado. Por ejemplo, quiero obtener todas las empresas dadas de alta en el año 2003 y dispones de 1000 empresas con 1000 fechas de alta... Para ello hay que hacer lo siguiente:

```
SELECT *
FROM Tabla1
WHERE DatePart("yyyy",FECHA_ALTA)="2003";
```

Artículo por **Jonathan Soriano Folch**

## Emular un Cursor SQL con un Bucle

Gracias a este truco de SQL Server 2000 podrás emular el funcionamiento de un cursor con un bucle.

Para ello crearemos una tabla temporal donde le pondremos los elementos que queremos iterar en el bucle para poderlos tratar.

```
DECLARE
@Anuncios
TABLE
(
pk_id numeric(18, 0) NOT NULL IDENTITY (1, 1),
Idtruco numeric(18,0),
IdUsuario numeric(18,0),
Alias nvarchar(255),
usuario nvarchar(255)
)
```

Creamos dos variables para poder iterar en el bucle

```
DECLARE
@Rows numeric,
@i numeric(18,0)
```

```
SET @Rows=0
SET @i=1
```

Insertamos los datos en la tabla temporal @anuncios

```
INSERT INTO
@Anuncios
(
Idtruco,
IdUsuario,
Alias,
Usuario
)
SELECT
a.ARTID,
a.ARTUSR,
p.Alias,
p.LonUsr
FROM
TABLA_ANUNCIOS a
INNER JOIN
TABLA_USUARIOS p
ON a.ARTUSR=p.LONID
```

Asignamos a la variable contadora de filas totales el total de la tabla @anuncios

```
Set @Rows=(SELECT TOP 1 PK_ID FROM @Anuncios order BY PK_ID DESC)
```

Iteramos con el while. De esta manera podemos emular el funcionamiento de un cursor sin ser un cursor, pudiéndolo ejecutar las veces que queramos a la vez.

```
WHILE @i <= @Rows
BEGIN
Declare
@Idtruco numeric(18,0),
@IdUsuario numeric(18,0),
@Alias nvarchar(255),
@Usuario nvarchar(255)
```

```
SELECT
@Idtruco=Idtruco,
@IdUsuario=IdUsuario,
@Alias=Alias,
@Usuario=Usuario
FROM
@Anuncios
WHERE
pk_id=@i
```

Realizar todas las acciones!

```
SET @i=@i + 1
END
```

*Artículo por **Pol Salvat***

## Tipos de sentencias SQL y sus componentes sintácticos

En SQL tenemos bastantes sentencias que se pueden utilizar para realizar diversas tareas.

Dependiendo de las tareas, estas sentencias se pueden clasificar en tres grupos principales (DML, DDL, DCL), aunque nos quedaría otro grupo que a mi entender no está dentro del lenguaje SQL sino del PLSQL.

SENTENCIA	DESCRIPCIÓN	
	DML	Manipulación de datos
		SELECT INSERT DELETE UPDATE
		Recupera datos de la base de datos. Añade nuevas filas de datos a la base de datos. Suprime filas de datos de la base de datos. Modifica datos existentes en la base de datos.
	DDL	Definición de datos
		CREATE TABLE DROP TABLE ALTER TABLE CREATE VIEW DROP VIEW

	CREATE INDEX DROP INDEX CREATE SYNOYM DROP SYNONYM
	Añade una nueva tabla a la base de datos. Suprime una tabla de la base de datos. Modifica la estructura de una tabla existente. Añade una nueva vista a la base de datos. Suprime una vista de la base de datos. Construye un índice para una columna. Suprime el índice para una columna. Define un alias para un nombre de tabla. Suprime un alias para un nombre de tabla. <b>DCL</b>   <b>Control de acceso</b> GRANT REVOKE <b>Control de transacciones</b> COMMIT ROLLBACK
	Concede privilegios de acceso a usuarios. Suprime privilegios de acceso a usuarios  Finaliza la transacción actual. Aborata la transacción actual. <b>PLSQL</b>   <b>SQL Programático</b> DECLARE OPEN FETCH CLOSE
	Define un cursor para una consulta. Abre un cursor para recuperar resultados de consulta. Recupera una fila de resultados de consulta. Cierra un cursor.

## Componentes sintácticos

La mayoría de sentencias SQL tienen la misma estructura.

Todas comienzan por un verbo (select, insert, update, create), a continuación le sigue una o más cláusulas que nos dicen los datos con los que vamos a operar (from, where), algunas de estas son opcionales y otras obligatorias como es el caso del from.

Artículo por **Sara Alvarez**

