

*Colección*  
*Recursos Informáticos*


# SQL Server 2012

## SQL, Transact SQL

**Diseño y creación  
de una base de datos**

Descarga  
[www.ediciones-eni.com](http://www.ediciones-eni.com)

**Jérôme GABILLAUD**

 **INFORMÁTICA TÉCNICA**



<b>SQL Server 2012 - SQL, Transact</b>	5
<b>SQL</b>	5
Diseño y creación de una base de datos	5
Preámbulo	6
Introducción	7
Recordatorio sobre el almacenamiento de datos	8
<b>1. Las diferentes categorías de datos</b>	8
a. Los datos básicos	8
b. Los datos cambiantes	8
c. Los datos de trabajo	9
d. Los datos almacenados	9
<b>2. La organización de los datos</b>	9
a. Directa	9
b. Secuencial	9
c. Secuencial indexada	9
d. Base de datos jerárquica	10
e. Base de datos relacional	10
El modelo relacional	10
<b>1. Conceptos y definiciones</b>	11
a. Representación	12
<b>2. Reglas principales</b>	12
<b>3. Normalización del esquema relacional</b>	14
El álgebra relacional	16
<b>1. Operadores</b>	16
<b>2. Etapas de la resolución de un problema</b>	23
Administrar una base de datos	29
<b>1. Administrar el espacio de almacenamiento</b>	30
<b>2. Administrar el objeto DATABASE</b>	32
a. Crear la base de datos	32
b. Modificar el tamaño	35
c. Eliminar la base de datos	38
d. Renombrar una base de datos	38
e. Configurar una base de datos	38
Administrar tablas e índices	43
<b>1. Identificador</b>	43
<b>2. Los tipos de datos</b>	44
a. Tipos de datos de sistema	44
b. Tipos de datos definidos por el usuario	48

<b>3. Administrar las tablas</b>	52
a. Crear una tabla	52
b. Modificar una tabla	55
c. Eliminar una tabla	56
d. Nombre completo de una tabla	56
e. Columnas calculadas	57
Implementación de la integridad de los datos	57
<b>1. Los valores por defecto</b>	58
<b>2. Las reglas</b>	58
<b>3. La propiedad Identity</b>	58
<b>4. Las restricciones de integridad</b>	60
a. NOT NULL	61
b. PRIMARY KEY	61
c. UNIQUE	63
d. REFERENCIAS	64
e. DEFAULT	66
f. CHECK	68
Administrar los índices	70
<b>1. Crear un índice</b>	73
<b>2. Eliminar un índice</b>	77
<b>3. Reconstruir un índice</b>	77
<b>4. Las estadísticas</b>	79
<b>5. Información sobre los índices</b>	81
Monitorizar y verificar las bases de datos y los objetos	83
Los esquemas	84
Aspectos generales	84
<b>1. Expresiones</b>	85
<b>2. Operadores</b>	86
<b>3. Funciones</b>	90
a. Funciones de agregación	90
b. Funciones matemáticas	91
c. Funciones trigonométricas	92
d. Funciones logarítmicas	92
e. Funciones diversas	93
f. Funciones de tipo fecha	93
g. Funciones de tratamiento de cadena de caracteres	95
h. Funciones de sistema	97
El SQL-DML	100

<b>1. Creación de registros</b>	100
<b>2. Modificación de registros</b>	104
<b>3. Eliminar registros</b>	106
<b>4. Extracción de registros</b>	109
<b>5. Operaciones del álgebra relacional</b>	110
a. Selección de columnas	110
b. Restricción	112
c. Cálculos sencillos	114
d. Proyección	115
e. Cálculos agregados	117
f. Producto cartesiano	119
g. Join	121
h. Join externo	124
i. Order By	126
j. Unión	128
k. Except	128
l. Intersect	129
m. Extraer solo los primeros registros	130
<b>6. Consulta de creación de tablas</b>	132
<b>7. Forzar el optimizador de consultas</b>	133
<b>8. Tablas CTE</b>	134
<b>9. Generación de registros estadísticos</b>	135

# SQL Server 2012 - SQL, Transact SQL

## Diseño y creación de una base de datos

Este libro sobre SQL Server 2012 está dirigido tanto a los **estudiantes de informática** que deseen aprender SQL con SQL Server, como a los **informáticos** que deseen actualizar sus competencias sobre SQL Server 2012. De esta manera podrán entender y dominar las funcionalidades que permiten diseñar y construir una base de datos.

El libro detalla **el conjunto de instrucciones** necesarias para definir **tablas** y **manipular los datos**: se presentan e ilustran las diferentes instrucciones SQL y Transact SQL (procedimientos almacenados, funciones y triggers), para entender perfectamente el interés de las funcionalidades que se explican.

Se abordan temas tan interesantes y novedosos como la gestión de datos no estructurados (FILESTREAM), las estructuras jerárquicas y los datos geográficos. También se explican en detalle las principales funcionalidades que permiten gestionar los datos de manera rápida y eficaz, como **XML** o la integración de **código CLR** (Common Language Runtime). Por último, se aborda la biblioteca de programación **SMO** y cómo operar con ella en PowerShell.

**Los capítulos del libro:**  
Preámbulo – El modelo relacional – Implementación de bases de datos – Las órdenes de SQL – Transact SQL: el lenguaje procedimental – Gestión de datos distribuidos – Los tipos avanzados – Common Language Runtime – Anexos

---

Jérôme GABILLAUD

Ingeniero informático industrial y consultor, **Jérôme GABILLAUD** también es responsable pedagógico en un gran centro de formación informática. Especialista de los sistemas de acceso a los datos de Microsoft y Oracle, Jérôme Gabillaud también es autor de numerosas obras sobre este tema, reconocidas por sus cualidades técnicas y pedagógicas

## Preámbulo

Este libro está dirigido a los desarrolladores de aplicaciones y a todas aquellas personas que quieran serlo. La primera parte está dedicada al modelo y álgebra relacionales. El álgebra es el origen del lenguaje SQL (*Structured Query Language*). Una vez que se domina, se puede trabajar de manera eficaz con el lenguaje SQL, independientemente de cuál sea el servidor de base de datos elegido. Después se aborda la parte correspondiente a la gestión de la estructura o SQL DDL (*Data Definition Language*) y de datos con SQL DML (*Data Manipulation Language*). Para terminar, se detalla Transact SQL, que es el lenguaje de programación en SQL Server. Los capítulos siguientes se dedican a descubrir y aprender las diferentes funcionalidades que ofrece SQL Server al desarrollador de aplicaciones para facilitarle el trabajo: integración de código .NET en SQL Server, gestión de datos de tipo XML...

Las tareas relativas a la administración del servidor, tales como la gestión del espacio en disco, la seguridad, las copias de seguridad, la restauración, la replicación, etc. se detallan en otro libro de Ediciones ENI.

Respecto al desarrollo, SQL Server ofrece un conjunto de herramientas y funcionalidades que permiten codificar más rápido. SQL Server también permite reducir la distancia que existe entre el desarrollo y la administración de la base de datos y del servidor.

El objetivo que se persigue es mejorar el rendimiento y permitir la gestión de todos los datos de la empresa con los que los usuarios trabajan habitualmente. SQL Server ofrece muchos tipos de datos especializados, como el tipo jerárquico, la gestión de los datos geográficos o los documentos XML. También ofrece una gestión eficiente de los datos de tipo binario, gracias al almacenamiento en el sistema de archivos.

Como el resto de productos servidor, SQL ofrece una interfaz completa con PowerShell, usando la biblioteca SMO del framework .NET.

## **Introducción**

La puesta en marcha (o implementación) de una base de datos implica una serie de operaciones específicas, que se deben distinguir de las operaciones orientadas a la administración.

Las diferentes funciones del implementador son:

- Diseñar el esquema lógico de los datos.
- Plasmar los datos en el modelo físico.
- Diseñar e implementar las restricciones de integridad.
- Programar el servidor de datos.

## **Recordatorio sobre el almacenamiento de datos**

El almacenamiento de datos es un problema tan antiguo como la informática. A medida que evolucionan las capacidades técnicas del hardware y el volumen de datos que se maneja, también evoluciona la manera de almacenar y organizar estos datos.

En una aplicación de gestión, estos problemas de organización no afectan por igual a todas las categorías de datos.

### **1. Las diferentes categorías de datos**

En primer lugar hay que definir categoría de datos. Esta categorización se realiza planteándose una serie de sencillas preguntas:

- ¿para qué sirven los datos? y
- ¿cuánto tiempo hay que conservar estos datos?

#### **a. Los datos básicos**

Este tipo de datos es la base de cualquier sistema de información. A partir de estos datos es posible trabajar. Por ejemplo, pensemos en una gestión comercial. Los datos básicos están formados por la información de los clientes y productos. Los datos de este tipo se detallan tanto como sea posible. Su vigencia es muy prolongada en el tiempo. Como se trata de datos básicos, es necesario poder acceder a ellos fácil y rápidamente.

#### **b. Los datos cambiantes**

Estos datos se generan a partir de los datos básicos. Al contrario que estos últimos su vigencia está limitada, pero su volumen es mucho más importante. Por ejemplo, siempre en el marco de una gestión comercial, la información relativa a cada pedido se considera como datos cambiantes. El volumen es importante, ya que la empresa espera que cada cliente haga varios pedidos a lo largo de un mismo año contable.



Por el contrario, la vigencia de esta información es menor. Efectivamente, no es necesario conservar accesible este tipo de información varios años, sino que se puede almacenar en otro soporte de almacenamiento menos costoso.

### c. Los datos de trabajo

Son los datos generados con un objetivo determinado, algunas veces con un volumen importante, pero con una vigencia muy corta. Tan pronto como se haga el trabajo, ya no es necesario conservar estos datos. Un ejemplo sería los datos extraídos de la base de datos que sirven para preparar un gráfico. Cuando se han preparado los gráficos, ya no es necesario conservar los datos que fueron extraídos de la base de datos para generarlos.

### d. Los datos almacenados

Se trata de datos muy voluminosos y con una vigencia muy larga, pero se caracterizan por no poder acceder a ellos directamente, sino que se accede en modo lectura. Si pensamos en el mismo ejemplo basado en una gestión comercial, un ejemplo sería los datos de años contables anteriores.

## 2. La organización de los datos

### a. Directa

Sin duda, es la organización más sencilla que se puede utilizar. Los datos se guardan unos detrás de otros en un archivo. Cada conjunto de datos tiene un tamaño fijo y los registros se almacenan unos detrás de otros. Si se conoce el tamaño de un registro, mediante un sencillo cálculo, se puede acceder directamente por ejemplo al décimo registro.

Este tipo de organización es costosa en espacio en disco y no permite extraer fácilmente la información, utilizando criterios diferentes a los de su simple posición en el orden de grabación.

### b. Secuencial

Con la organización secuencial, los datos se graban unos detrás de otros. Se utiliza un carácter especial para marcar la separación entre los diferentes campos y otro para marcar el final de cada registro. Normalmente, los caracteres que se utilizan son la coma (,) y fin de línea (CR). Los archivos que contienen estos separadores se llaman archivos CSV (*Comma Separated Values*).

Este tipo de organización permite optimizar el espacio de almacenamiento que se utiliza. De esta manera se resuelve uno de los problemas principales de los archivos con acceso directo. Por el contrario, como sucede con la organización directa, cuando queremos buscar datos que respondan a criterios de selección precisos, es necesario recorrer todos los datos: esto es más largo cuanto mayor sea el volumen de datos (número de registros).

### c. Secuencial indexada

Los datos siempre se almacenan en formato secuencial, pero para permitir un acceso más rápido a la información, se pueden definir índices para cada archivo.

Dentro de estos índices, los datos se ordenan por orden alfanumérico. El recorrido de un índice es secuencial y permite un acceso directo a la información que se almacena en el archivo de datos.

El recorrido del índice, aunque es secuencial, es rápido ya que el volumen de datos que se maneja es bajo. Además, como los datos están ordenados, no es necesario leer el índice completo.

Por último, es posible definir varios índices para el mismo archivo de datos. Por ejemplo, en un archivo que almacena información de clientes, es posible definir un índice sobre los nombres y otro sobre las ciudades.

Con este tipo de organización lo difícil es mantener los índices actualizados después de operaciones de adición, eliminación o modificación de registros. Además, como sucede con la organización directa y secuencial, los archivos no están relacionados los unos con los otros y no existe contexto de seguridad a nivel de datos. Por ejemplo, a nivel de datos, se puede eliminar un cliente aunque tenga pedidos en curso. De la misma manera, cualquier persona puede trabajar con los datos y acceder a ellos en modo lectura y escritura. Estos inconvenientes plantean más problemas con la organización secuencial indexada, porque se pueden gestionar volúmenes de datos importantes, con muchos usuarios conectados.

Esta solución secuencial indexada se ha adoptado de manera masiva en las aplicaciones pequeñas y medianas, porque facilita los desarrollos y muchos lenguajes de programación ofrecen un motor de gestión que usa este tipo de organización.

#### **d. Base de datos jerárquica**

Con estas bases de datos se resuelven los problemas de seguridad de acceso a los datos y la relación entre ellos. Pero por otra parte, los diferentes fabricantes han desarrollado cada uno su motor de manera independiente. Aprender a utilizar un nuevo motor implica aprender desde el principio (el lenguaje de consulta y el API de acceso a los datos), a lo que hay que añadir una organización compleja de los datos. Estas soluciones propietarias suelen ser muy costosas para la empresa que las elige.

#### **e. Base de datos relacional**

Se basa en una representación lógica de los datos, que respeta el modelo relacional. Las bases de datos relacionales se han sabido imponer, ya que todas utilizan un mismo lenguaje estandarizado y normalizado de acceso a datos: SQL.

### **El modelo relacional**

La organización de los datos dentro del sistema de gestión de bases de datos relacionales (SGBDR o RDBMS en inglés), se basa completamente en el modelo relacional. Edgar Franck Codd, con el apoyo de IBM, desarrolló este modelo en los años 70. Además de este modelo también se creó un álgebra (el álgebra relacional), para poder extraer los datos que se almacenan en este modelo. Entender este modelo y su álgebra permite aprender cómodamente SQL, ya que solo hay que trasladar los conceptos teóricos a la línea de comandos. Además, este trabajo

permitirá adaptarse más fácilmente a las diferentes mejoras que SQL puede incluir en futuras versiones.

El álgebra relacional ha permitido desarrollar SQL, que se ha convertido en el estándar de la gestión de los datos.

El hecho de que los SGBDR respeten el modelo relacional permite trabajar con una estructura lógica de la organización de los datos (tablas, vistas, índices...), independiente de la estructura física (archivos, etc.). Cada SGBDR debe proporcionar una vista lógica al usuario, que garantice un almacenamiento físico de la información.

Esta restricción es, al mismo tiempo, la fortaleza de los SGBDR ya que la gestión de los datos desde un punto de vista lógico, simplifica mucho su uso. De esta manera, los usuarios poco o nada acostumbrados a desarrollar aplicaciones se pueden iniciar sin problemas en SQL.

## 1. Conceptos y definiciones

El modelo relacional se basa en conceptos básicos sencillos (dominio, relación o atributo), a los que se aplican reglas precisas. Un lenguaje declarativo sencillo (no procedimental), basado en una lógica de conjuntos, facilita el desarrollo de una base de datos.

### **Dominio**

Es un conjunto de valores representados por un nombre.

### **Cardinal**

Es el número de elementos de un dominio.

#### Ejemplo

*El diccionario de datos del análisis de una gestión comercial puede tener, entre otros, especificaciones sobre la gestión de los estados del pedido o de los números de pedido que se deben mostrar.*

*El modelo relacional los traducirá de la siguiente manera:*

Estados de los pedidos = {"EC", "LI", "FA", "SO"}; cardinal 4

Números de pedido = {n | 1 ≤ n ≤ 9.999}; cardinal 9.999.

### **Producto cartesiano**

El producto cartesiano P entre varios dominios D1, D2... Dn, que se representa por  $P = D1 \times D2 \times \dots \times Dn$ , es el conjunto de las n-tuplas (d1, d2 ... dn), donde cada di es un elemento del dominio Di.

#### Ejemplo

*Si queremos gestionar dos dominios (códigos y tasas), podríamos obtener tuplas compuestas de un código y una tasa.*

Códigos = {1,2,3,4}

Tasa de IVA = {0,7.0,21.0}

Códigos X Tasas de IVA = { (1,0), (1,7.0), (1,21.0),  
(2,0), (2,7.0), (2,21.0), (3,0), (3,7.0), (3,21.0),  
(4,0), (4,7.0), (4,21.0) }

## Relación

Una relación definida sobre los dominios D1, D2... Dn, es un subconjunto del producto cartesiano de estos dominios, representado por un nombre.

## Atributo

Es una columna de una relación representada por un nombre.

## Grado

Es el número de atributos de una relación.

### Ejemplo

*Para asociar una sola tasa para cada código, solo se ven afectados tres tuplas.*

Relación IVA = { (1,0), (2,7.0), (3,21.0) }

### a. Representación

Se hace en forma de tabla, como una matriz de valores:

IVA	CODIGO	VALOR
	1	0.00
	2	7.00
	3	21.00

O detalle de valores:

IVA (CODIGO:códigos, VALOR:Tasas de IVA)

o

IVA (CODIGO, VALOR)

## 2. Reglas principales

El modelo relacional gestiona un objeto principal, la relación, asociada a los conceptos de dominio y atributo.

Las reglas se aplican a esta relación para respetar las restricciones relacionadas con el análisis. Algunas de estas reglas son:

### Unicidad

Todos los elementos de una relación deben ser distintos.

## Identificador

Atributo o conjunto de atributos que permite caracterizar de manera única a cada elemento de la relación.

## Clave primaria o Primary key

Identificador mínimo de una relación.

## Clave secundaria

Otros identificadores de la relación.

## Integridad referencial

Esta regla obliga a que un atributo o conjunto de atributos de una relación aparezca como clave primaria en otra relación.

## Clave extranjera o Foreign key

Atributo o conjunto de atributos que cumplen la regla de integridad referencial.

### Ejemplo

*El análisis de una gestión comercial nos obliga a gestionar clientes con características (nombre, dirección) y sus pedidos.*

*Podemos tener el siguiente modelo:*

CLIENTES	NUMEROCLI	NOMBRE	DIRECCION
	15	SANCHEZ S.A.	SEVILLA
	20	Mrt Gonz	MADRID
	35	Constr CO	SEVILLA
	138	Constr CO	SANTANDER

FECHA

PEDIDOS	NUMEROPDO	FECHA	NUMEROCLI	ESTADO
	1210	01/08/2012	15	SO
	1230	07/07/2012	35	SO
	1301	05/04/2012	15	EC
	1280	01/03/2011	20	LI
	1150	18/02/2012	15	SO
	1250	02/01/2012	35	EC

CLIENTES (NUMEROCLI, NOMBRE, DIRECCION)

NUMEROCLI identificador clave primaria de CLIENTES

NOMBRE, DIRECCION identificador clave secundaria de CLIENTES

PEDIDOS (NUMEROPDO, FECHA, NUMEROCLI, ESTADO)

NUMEROPDO identificador clave primaria de PEDIDOS

NUMEROCLI clave extranjera de PEDIDOS, que hace referencia a

NUMEROCLI de CLIENTES

## Valor nulo

En el modelo relacional se permite la noción de nulidad. Es un valor que representa un dato desconocido o no aplicable para una columna. Se representa con `_`, `^` o `NULL`.

## Restricción de integridad

Todo valor de una clave primaria debe ser diferente a `NULL`.

### Ejemplo

*En la relación artículo, permitimos que el precio o el IVA puedan ser desconocidos, pero la referencia del artículo (clave primaria), debe estar rellena.*

ARTICULOS	REFART	NOMBRE	PRECIO	IVA	CODIGO CAT
	AB10	Albombra china	1.500	2	IMPORT
	AB22	Alfombra persa	1.750	2	IMPORT
	CD50	Cadena HIFI	735	2	IMPORT
	ZZZZ	Sobrecamas	NULL	NULL	VARIOS
	AA00	Cuadro	0	NULL	VARIOS
	AB03	Marco pared	75	2	OFERTA
	AB	Tapiz	NULL	2	VARIOS
	ZZ01	Silla	250	2	VARIOS

## 3. Normalización del esquema relacional

Cuando el esquema relacional se define para responder a las necesidades de todos los usuarios, es necesario normalizarlo para evitar cualquier redundancia de datos o estructura no conforme con el modelo relacional. Cuando se hace esta operación, el esquema se podrá desnormalizar, aunque normalmente esta no será la mejor operación. Si el desarrollador desnormaliza el esquema, también debe poner en marcha el conjunto de mecanismos que permitan mantener la coherencia de los datos. El modelo relacional, y por tanto los SGBDR, solo pueden garantizar la coherencia de los datos en modelos normalizados.

Las formas normales permiten garantizar que el esquema respeta el modelo relacional. Teóricamente, existen cinco formas normales, pero en la práctica solo se usan las tres primeras.

La aplicación de las formas normales hace necesario dominar el concepto de dependencia funcional. Un dato depende funcionalmente de otro cuando, conociendo el segundo, podemos determinar el valor del primero. Por ejemplo, es posible decir que en una aplicación de gestión comercial, existe una dependencia funcional entre un código IVA y la tasa de IVA o entre la referencia de un artículo y su representación.

**Primera forma normal:** se dice que una tabla está en primera forma normal, cuando todas las columnas contienen valores simples.

Por ejemplo, si una tabla de clientes contiene un campo Teléfonos en el que se almacenan los diferentes números de teléfono de un cliente, entonces esta tabla no está en primera forma normal. Es necesario definir las columnas Oficina y Móvil para estructurar mejor los datos.

Cientes	Número	Apellido	Nombre	Teléfonos	Pedido	Fecha
	1	Sánchez	Ángel	01 02 03 04 05 06 07 08 09 10	1350	01/01/2012
	1	Sánchez	Ángel	01 02 03 04 05 06 07 08 09 10	1352	15/01/2012
	2	González	María	01 03 05 07 09	1351	02/01/2012

La tabla anterior no respeta la primera forma normal.

Cientes	Número	Apellido	Nombre	Oficina	Móvil	Pedido	Fecha
	1	Sánchez	Ángel	01 02 03 04 05	06 07 08 09 10	1350	01/01/2012
	1	Sánchez	Ángel	01 02 03 04 05	06 07 08 09 10	1352	15/01/2012
	2	González	María	01 03 05 07 09		1351	02/01/2012

Esta tabla respeta la primera forma normal.

**Segunda forma normal:** se dice que una tabla está en segunda forma normal si está en primera forma normal y todas las columnas que no pertenecen a ninguna clave, dependen funcionalmente de la clave primaria.

Volviendo al ejemplo anterior, en principio es posible admitir que la clave de la tabla de clientes está formada por las columnas Número y Pedido.

En este caso, los valores de las columnas Apellido, Nombre, Oficina y Móvil dependen solo del número, mientras que la columna Fecha está relacionada con el número del pedido. Por tanto, la tabla no está en segunda forma normal. Es necesario definir dos tablas: clientes y pedidos.

Cientes	Numero	Apellido	Nombre	Oficina	Móvil
	1	Sánchez	Ángel	01 02 03 04 05	06 07 08 09 10
	2	González	María	01 03 05 07 09	

Pedidos	Número	Fecha	Cliente
	1350	01/01/2012	1
	1352	15/01/2012	1
	1351	02/01/2012	2

Las dos tablas anteriores cumplen la segunda forma normal.

**Tercera forma normal:** se dice que una tabla está en tercera forma normal si está en segunda forma normal y no hay dependencia funcional entre dos columnas que no pertenecen a ninguna clave.

Por ejemplo, si en la tabla de clientes se añaden las columnas Tratamiento y Sexo, de la siguiente manera:

Cientes	Número	Apellido	Nombre	Oficina	Móvil	Tratamiento	Sexo
	1	Sánchez	Ángel	01 02 03 04 05	06 07 08 09 10	Sr	M
	2	González	María	01 03 05 07 09		Srta.	F

Es posible decir que existe una dependencia funcional entre el sexo y el tratamiento. Efectivamente, el hecho de conocer el tratamiento (Srta., Sra. o Sr.), permite deducir el sexo. Por tanto, la tabla de clientes no respeta la tercera forma normal. Se define la tabla de tratamientos para obtener el siguiente esquema:

Cientes	Número	Apellidos	Nombre	Oficina	Móvil	Tratamiento
	1	Sánchez	Ángel	01 02 03 04 05	06 07 08 09 10	Sr
	2	González	María	01 03 05 07 09		Srta.
Tratamientos				Valor	Sexo	
				Srta.	F	
				Sra.	F	
				Sr.	M	

Las dos tablas anteriores cumplen la tercera forma normal.

## El álgebra relacional

Es un método de extracción que permite manipular tablas y columnas. Su principio se basa en la creación de nuevas tablas (tablas resultantes) a partir de las existentes. Estas nuevas tablas se convierten en objetos que se pueden usar inmediatamente.

Los operadores del álgebra relacional que permiten crear las tablas resultantes, se basan en la teoría de conjuntos.

La sintaxis y los elementos de notación que se utilizan aquí son los más habituales.

### 1. Operadores

#### Unión

La unión entre dos relaciones de una misma estructura (grado y dominios), da como resultado una tabla con la misma estructura, cuyos elementos son el conjunto de elementos distintos de las dos relaciones iniciales.

Notación:  $R_x = R_1 \cup R_2$

#### Ejemplos

Tomamos las tablas *CLIOESTE* et *CLICENTRO*:



CLIOESTE	NUMCLI	NOMCLI	DIRECCION
	15	Sánchez S.A.	SEVILLA
	35	CONstr Con	SEVILLA
	152	Julious	CORUÑA

CLICENTRO	NUMCLI	NOMCLI	DIRECCION
	152	Julious	CORUÑA
	20	Mrt Gonz	MADRID
	138	Larioiss	SANTANDER
	36	INCCRT	BARCELONA

*Cientes de las dos regiones:*

CLIENTES=CLIOESTE U CLICENTRO

CLIENTES	NUMCLI	NOMCLI	DIRECCION
	15	Sánchez S.A.	SEVILLA
	35	CONstr Con	SEVILLA
	152	Julious	CORUÑA
	20	Mrt Gonz	MADRID
	138	Larioiss	SANTANDER
	36	INCCRT	BARCELONA

## Intersección

La intersección entre dos relaciones de una misma estructura (grado y dominios), da como resultado una tabla con la misma estructura, cuyos elementos son el conjunto de elementos comunes a las dos relaciones iniciales.

Notación:  $R_x = R_1 \cap R_2$

Ejemplo

*Cientes comunes a las dos regiones:*

CLICOMUN=CLIOESTE  $\cap$  CLICENTRO

Cientes comunes a las dos regiones

CLICOMUN = CLIOESTE  $\cap$  CLICENTRO

CLICOMUN	NUMCLI	NOMCLI	DIRECCION
	152	Julious	CORUÑA

## Diferencia

La diferencia entre dos relaciones de una misma estructura (grado y dominios), da como resultado una tabla con la misma estructura, cuyos elementos son el conjunto de los elementos de la primera relación, que no están en la segunda.

Notación:  $R_x = R_1 - R_2$

Ejemplo

*Cientes solo de la región OESTE:*

CLISOLOESTE=CLIOESTE - CLICENTRO

CLISOLOESTE	NUMCLI	NOMCLI	DIRECCION
	15	Sánchez S.A.	SEVILLA
	35	Constr Con	SEVILLA

## División

La división entre dos relaciones es posible con la condición de que la relación divisora esté completamente incluida en la relación dividendo. El cociente de la división corresponde a los datos presentes en el dividendo y no en el divisor.

También es posible definir la división de la siguiente manera:

Sea  $R_1$  y  $R_2$  relaciones, tales que  $R_2$  está completamente incluida en  $R_1$ .

El cociente  $R_1 \div R_2$  está formado por las tuplas  $t$  para las que, para toda tupla  $t'$  definida en  $R_2$ , existe la tupla  $t.t'$  definida en  $R_1$ .

Notación:  $R_x = R_1 \div R_2$

### Ejemplo

La relación personas que contiene los datos de los individuos:

PERSONAS	NUMCLI	NOMCLI	DIRECCION	TELEFONO
	15	Sánchez S.A	SEVILLA	0034987425888
	35	CONstr Con	SEVILLA	0034878965289
	204	MARTIN	MADRID	0034548798599
	152	Julious	CORUÑA	0034913572198

La relación clioste:

CLIOESTE	NUMCLI	NOMCLI	DIRECCION
	15	Sánchez S.A.	SEVILLA
	35	CONstr Con	SEVILLA
	152	Julious	CORUÑA

La división entre las dos relaciones, permite aislar la información complementaria de los clientes y presente en la relación individuo:

INTERSECCION	TELEFONO
	0034987425888
	0034878965289
	0034913572198

## Restricción

La restricción se basa en una condición. A partir de una relación, genera una relación del mismo esquema pero solo con los elementos de la relación inicial que cumplen la condición.

Notación:  $R_x = \sigma (\text{condición}) R_1$

La condición se expresa como:

[NO] [(] atributo operador valor [)] [{Y/O}condicion]

operador

Un operador de comparación: =, <>, >, <, >=, <=

valor

Una constante u otro atributo.

### Ejemplos

*Cientes de SEVILLA:*

CLI44= $\sigma$  (DIRECCION="SEVILLA") CLIOESTE

CLI44	NOMCLI	NUMCLI	DIRECCION
	15	Sánchez S.A.	SEVILLA
	35	Constr Con	SEVILLA

*Artículos de la familia AB:*

ART1= $\sigma$  (REFART>="AB" Y REFART<"AC") ARTICULOS

ART1	REFART	NOMBRE	PRECIO	IVA
	AB10	Albombra china	1.500	2
	AB22	Alfombra persa	1.750	2
	AB03	Marco pared	75	2
	AB	Tapiz	-	2

*Alfombras cuyo precio es inferior a 100:*

ART2= $\sigma$  (PRECIO<=100) ART1

ART2	REFART	NOMBRE	PRECIO	IVA
	AB03	Marco pared	75	2

## Proyección

La proyección de una relación sobre un grupo de atributos da como resultado una relación, cuyo esquema está formado solo por estos atributos y los elementos son las distintas tuplas formadas por los valores asociados de estos atributos.

Notación:  $R_x = \pi R (A_1, A_2 \dots A_n)$ .

### Ejemplo

*Pedidos y estados del pedido:*

PDO=  $\pi$  PEDIDOS (NUMEROPDO, NUMEROCLI, ESTADO)

PDO	NUMEROPDO	NUMEROCLI	ESTADO
	1210	15	SO
	1230	35	SO
	1301	15	EC
	1280	20	LI
	1150	15	SO
	1250	35	EC

*Cientes que tienen pedidos:*

CLIPDO1=  $\pi$  PEDIDOS (NUMEROCLI)

CLIPDO1	NUMEROCLI
	15
	35
	20

agrupa las líneas:  
1210, 1301, 1150  
1230 y 1250

*Cientes y estados del pedido:*

CLIPDO2=  $\pi$  PEDIDOS (NUMEROCLI, ESTADO)

## Producto cartesiano

El producto cartesiano entre dos relaciones, genera una relación cuyo esquema está formado por todos los atributos de las dos relaciones existentes y los elementos son la asociación de cada registro de la primera tabla con cada registro de la segunda.

Notación:  $R_x = S1 \times S2$

Ejemplo

*Tomamos las tablas:*

ALMACENES	CODIGO	NOMBREALM
	NW	SEVILLA
	SE	SANTANDER
	P1	PALENCIA1

ART2	REFART	NOMBRE
	AB	ALFOMBRA
	CD	HIFI

INVENTARIO = ALMACEN  $\times$  ART2

INVENTARIO	CODIGO	NOMBREALM	REFART	NOMBRE
	NW	SEVILLA	AB	ALFOMBRA
	NW	SEVILLA	CD	HIFI
	SE	SANTANDER	AB	ALFOMBRA
	SE	SANTANDER	CD	HIFI
	P1	PALENCIA 1	AB	ALFOMBRA
	P1	PALENCIA 1	CD	HIFI

## Joins

El join entre dos relaciones es un producto cartesiano, sobre el que se aplica una restricción.

Notación: Rx = S1 JOIN (condición) S2.

### Ejemplo

Tomamos las tablas:

PDOEC	NUMEROPDO	NUMEROCLI
	1301	15
	1250	35

LINEAPDO	NUMPDO	NUMLIN	REFART	CTA
	1210	1	AB10	3
	1210	2	CD50	4
	1230	1	AB10	1
	1301	1	AB03	3
	1301	2	AB22	1
	1250	1	CD50	5
	1280	1	AB10	15
	1150	1	AB03	7
	1150	2	AB22	5
	1150	3	AA00	1

LIGCDEEC = PDOEC JOIN (PDOEC.NUMEROPDO =  
LINEASPDO.NUMPDO) LINEASPDO

LIGCDEEC	NUMEROPDO	NUMEROCLI	NUMPDO	NUMLIN	REFART	CTAPDO
	1301	15	1301	1	AB03	3
	1301	15	1301	2	AB22	1
	1250	35	1250	1	CD50	5

Los diferentes tipos de join son:

### **Theta-join**

La condición es una comparación entre dos atributos.

### **Equi-join**

La condición se basa en la igualdad entre dos atributos.

### **Join natural**

Equi-join entre los atributos que tienen el mismo nombre.

### **Cálculos sencillos**

Proyección sobre una relación asociada a un cálculo sobre cada registro, para crear uno o varios atributos nuevos.

Notación: Rx =  $\pi$  S (A1... N1 = expresión calculada,...)

La expresión calculada puede ser:

- Una operación aritmética.
- Una función matemática.
- Una función que opera sobre una cadena.

### Ejemplo

Queremos obtener el importe de una línea de pedido (Precio \* Cantidad).

LINPDO	NUMPDO	NUMLIN	REFART	CTAPDO	PRECIO
	1301	1	AB03	3	150.00
	1301	2	AB22	1	1 250.10
	1250	1	CD50	5	735.40

$LINPDOVALO = \pi LINPDO (NUMPDO, NUMLIN, REFART, VALOLIN = CNTDPDO * PRECIOUNIT)$

LINPDOVALO	NUMPDO	NUMLIN	REFART	VALOLIN
	1301	1	AB03	450.00
	1301	2	AB22	1 250.10
	1250	1	CD50	3 677.00

## Cálculo agregado

Proyección de una relación asociada a uno o varios cálculos estadísticos basados en un atributo, para todos los elementos de la relación o de la agrupación relacionada con la proyección, para crear uno o varios atributos nuevos.

Notación:  $R_x = \pi S (A_1 \dots N_1 = \text{función estadística } (A_x), \dots)$

Las principales funciones estadísticas son:

**COUNT (\*)** número de registros.

**COUNT (atributo)** número de valores no nulos.

**SUM (atributo)** suma de los valores no nulos.

**AVG (atributo)** media de los valores no nulos.

**MAX (atributo)** valor máximo (no nulo).

**MIN (atributo)** valor mínimo (no nulo).

### Ejemplo

Número total de clientes en la tabla.

$NBCLI = \pi CLIENTES (N = COUNT (*))$

NBCLI	N
	6

Total de los importes por pedido:

$PDOVALO = \pi LINPDOVALO (NUMPDO, TOTPDO = SUM (VALOLIN))$

PDOVALO	NUMPDO	TOTPDO
	1301	1 700.10
	1250	3 677.00

Los precios mayores, los más bajos y el precio medio por categoría de artículos:

$STATART = \pi ARTICULOS (CATEGORIA, MASCARO =$

$MAX (PRECIO), MASBARATO = MIN (PRECIO),$

MEDIA=AVG (PRECIO) )

STATART	CATEGORIA	MASCARO	MASBARATO	MEDIA
	IMPORT	1 500.00	735.40	1 161.85
	VARIOS	500.00	0.00	250.00
	OFERTA	150.00	150.00	150.00

## 2. Etapas de la resolución de un problema

A partir de una base de datos conocida (esquemas, dominios, relaciones, elementos), es necesario:

### Analizar los requerimientos

- Transformar los requerimientos de las especificaciones en relaciones resultantes.
- Determinar los atributos y las relaciones que se deben usar.
- Expresar los cálculos sencillos y agregados para crear los atributos que no existen.

### Establecer la "vista"

La vista es una relación intermedia que contiene todos los atributos que permiten realizar la extracción, con sus relaciones de origen, clases de utilidad y operaciones que se deben aplicar.

### Clases de atributo

Clase a: atributo que participa en la relación resultante.

Clase b: atributo que participa en un cálculo.

Clase c: atributo que participa en una restricción.

Clase d: atributo que participa en un join.

### Planificar y expresar las operaciones

#### Caso general

- 1 Relaciones implicadas.
- 2 Restricciones (para eliminar los registros inútiles).
- 3 Joins, productos cartesianos, uniones, intersecciones y diferencias (para asociar los registros restantes).
- 4 Cálculos sencillos (para crear nuevas columnas).
- 5 Cálculos agregados (para las columnas estadísticas).

6 Join entre la tabla obtenida en los cálculos agregados y la tabla inicial en cálculos sencillos (para añadir las columnas estadísticas a las demás).

7 Repetir las etapas de cálculos agregados para el resto de agrupaciones.

8 Restricciones en relación a los atributos calculados.

9 Proyecciones para eliminar las repeticiones.

10 Proyección final para eliminar los atributos inútiles en la tabla resultante.

### **Ejemplo que se trata en el libro**

Base de datos GESCOM

El análisis de la gestión comercial (CLIENTES, STOCKS, PEDIDOS) de una empresa de montaje y venta, proporciona los siguientes esquemas de tabla:

CLIENTES (NUMERO\_CLI, NOMBRE, APELLIDOS, DIRECCION, CODIGOPOSTAL, CIUDAD, TELEFONO)

La tabla CLIENTES contiene un registro por cliente, con todas las características para poder contactar con el cliente o enviarle correo.

#### **Restricciones:**

NUMERO\_CLI es la clave primaria.

NOMBRE obligatorio (no NULL).

CODIGOPOSTAL en formato español.

CATEGORIAS (CODIGO\_CAT, ETIQUETA\_CAT)

Una categoría de artículos es una agrupación estadística interna codificada (01: Micros completos, 02: Paquete de software, etc.).

#### **Restricciones:**

CODIGO\_CAT es la clave primaria.

ARTICULOS (REFERENCIA\_ART, NOMBRE\_ART, PRECIOUNIT\_ART, CODIGO\_CAT)

Esta tabla debe contener un registro para cada artículo, sus características, principalmente su precio y la categoría a la que pertenece.

#### **Restricciones:**

REFERENCIA\_ART es la clave primaria.

CODIGO\_CAT es clave extranjera, que hace referencia a CATEGORIAS.

NOMBRE\_ART es obligatorio (no NULL).



PRECIUNIT\_ART debe ser mayor a cero.

STOCKS (REFERENCIA\_ART, ALMACEN, CNTD\_STK, STOCK\_MINI, STOCK\_MAXI)

La empresa tiene varios almacenes en España que pueden distribuir los materiales. Se debe poder gestionar la cantidad de artículos en cada almacén, así como los valores límite de las cantidades almacenadas.

### **Restricciones:**

El identificador de la clave primaria es la asociación de las columnas REFERENCIA\_ART y ALMACEN.

REFERENCIA\_ART es clave extranjera, que hace referencia a ARTICULOS.

STOCK\_MINI debe ser menor o igual a STOCK\_MAXI.

CNTD\_STK valor comprendido entre -100.000 y +100.000.

PEDIDOS (NUMERO\_PDO, FECHA\_PDO, TASA\_DESCUENTO, NUMERO\_CLI, ESTADO\_PDO)

Cuando un cliente hace un pedido, este se identifica por un número único. Podemos aplicar un descuento global a todo el pedido. Un identificador (ESTADO\_PDO) nos permite saber si el pedido está en fase de entrega, en fase de facturación o entregado.

### **Restricciones:**

NUMERO\_PDO es la clave primaria, los números se deben asignar en orden de creación.

ESTADO\_PDO solo puede tomar los valores: EC (en curso), LP (entregado parcialmente), LI (entregado) y SO (finalizado).

NUMERO\_CLI es clave extranjera, que hace referencia a CLIENTES.

TASA\_DESCUENTO no puede ser superior al 50%.

LINEAS\_PDO (NUMERO\_PDO, NUMERO\_LIN, REFERENCIA\_ART, CNTD\_PDO)

Como mínimo, cada pedido tiene un artículo con la cantidad solicitada.

### **Restricciones:**

El identificador de la clave primaria es la asociación de las columnas NUMERO\_PDO y NUMERO\_LIN.

NUMERO\_PDO es clave extranjera, que hace referencia a PEDIDOS.

REFERENCIA\_ART es clave extranjera, que hace referencia a ARTICULOS.

Todas las zonas son obligatorias (no NULL).

HISTO\_FAC (NUMERO\_FAC, FECHA\_FAC, NUMERO\_PDO, BASE\_IMPONIBLE, ESTADO\_FAC)

La interfaz con la contabilidad debe proporcionar la información de las (una o varias) facturas asociadas a cada pedido, en particular el importe total sin impuestos y el estado de la factura (parcialmente pagada, totalmente pagada o no pagada).

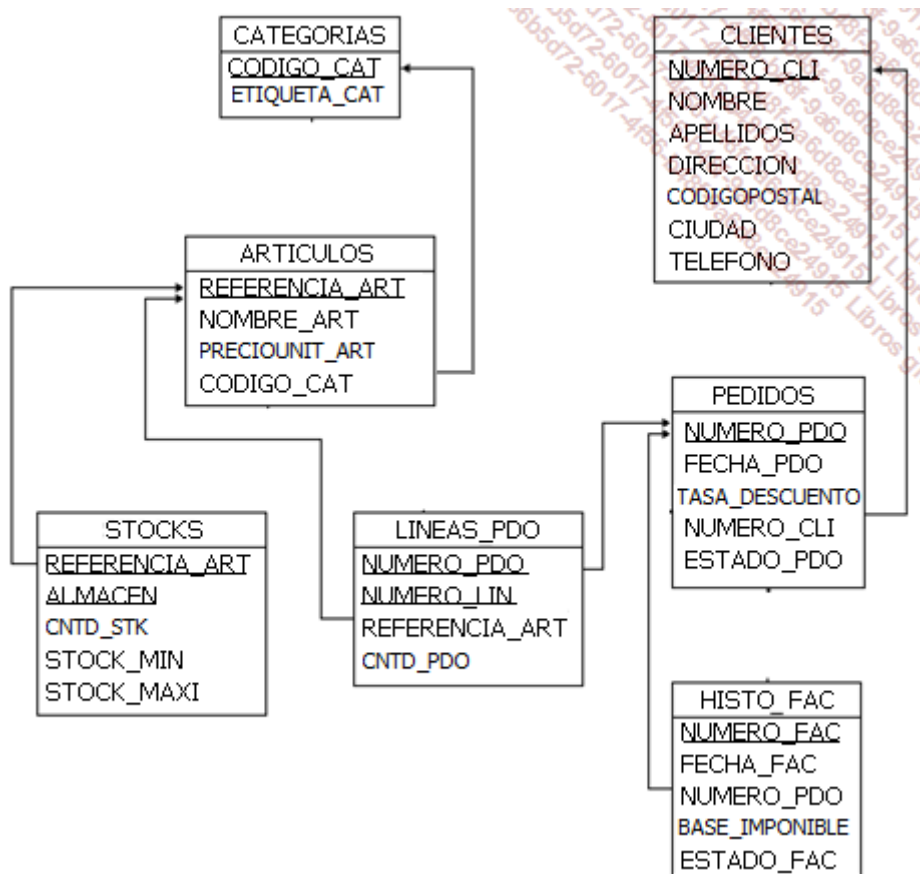
Cuando un pedido tiene todas las facturas pagadas, se considera finalizado.

### Restricciones:

NUMERO\_FAC es la clave primaria.

NUMERO\_PDO es clave extranjera, que hace referencia a PEDIDOS.

ESTADO\_FAC puede tomar los valores: NP (no pagada), PP (parcialmente pagada) y TP (totalmente pagada).



### Ejemplo

Podemos obtener la confirmación del pedido n° 1301. Modelo de documento:

el (fecha)		<u>Confirmación del pedido</u>		n° (N° del pedido) de (fecha del pedido)	
<u>CLIENTE:</u>		(Nombre del cliente) (Dirección del cliente)			
<b>Artículo</b>	<b>Precio</b>	<b>Cantidad</b>	<b>Precio total</b>		
(referencia)	(nombre)	(Precio unitario)	(Cantidad solicitada)	(precio * cntd)	
.	.	.	.	.	
.	.	.	.	.	
.	.	.	.	.	
.	.	.	.	.	
			Total sin impuestos: (Suma de cantidades brutas)		

### *Análisis del modelo:*

Fecha actual: información que se puede introducir cuando se rellene la página.

N° de pedido: NUMERO\_PDO de PEDIDOS

Fecha del pedido: FECHA\_PDO de PEDIDOS

Nombre del cliente: NOMBRE de CLIENTES

Dirección del cliente: DIRECCION de CLIENTES

Referencia: REFERENCIA\_ART de LINEAS\_PDO

Descripción: NOMBRE\_ART de ARTICULOS

Cantidad solicitada: CNTD\_PDO de LINEAS\_PDO

Precio unitario: PRECIUNIT\_ART de ARTICULOS

Importe: zona que se debe calcular

Base imponible: zona que se debe calcular

*Por lo tanto, para generar el documento es necesaria la siguiente tabla:*

CONFPDO (NUMERO\_PDO, FECHA\_PDO, NOMBRE, DIRECCION,  
REFERENCIA\_ART, NOMBRE\_ART, PRECIUNIT\_ART, CNTD\_PDO, IMPORTE,  
BASE\_IMP)

con  $IMPORTE = PRECIUNIT\_ART * CNTD\_PDO$  por cada pedido y

$BASE\_IMP = SUM(IMPORTE)$

para el pedido.

### **Vista**

Atributo	Tabla	Clase	Operación
NUMERO_PDO	PEDIDOS LINEAS_PDO	a,c,d	Restricción sobre NUMERO_CODIGO=1301 Join natural entre PEDIDOS y LINEAS_PDO

FECHA_PDO	PEDIDOS	A	
NUMERO_CLI	PEDIDOS CLIENTES	d	Join natural entre PEDIDOS y CLIENTES
NOMBRE	CLIENTES	a	
DIRECCION	CLIENTES	a	
REFERENCIA_ART	ARTICULOS LINEAS_PDO	a,d	Join natural entre LINEAS_PDO y ARTICULOS
CNTD_PDO	LINEAS_PDO	a,b	Cálculo de IMPORTE
NOMBRE_ART	ARTICULOS	a	
PRECIUNIT_ART	ARTICULOS	a,b	Cálculo de IMPORTE
IMPORTE		a,b	Cálculo de BASE_IMP
BASE_IMP		a	

## Operaciones

*Restricción sobre el número de pedido:*

$T1 = \sigma (\text{NUMERO\_PDO} = 1301) \text{ PEDIDOS}$

T1	NUMERO_PDO	FECHA_PDO	TASA_DESCUENTO	NUMERO_CLI	ESTADO_PDO
	1301	08/02/2001	0	15	EC

*Join natural entre PEDIDOS y CLIENTES:*

$T2 = T1 \text{ JOIN } (T1.\text{NUMERO\_CLI} = \text{CLIENTE}.\text{NUMERO\_CLI}) \text{ CLIENTES}$

T2	NUMERO_PDO	FECHA_PDO	NUMERO_CLI	NOMBRE	DIRECCION	...
	1301	08/02/2001	15	Sánchez	Islas Pitiusas	...

*Join natural entre PEDIDOS y LINEAS\_PDO:*

$T3 = T2 \text{ JOIN } (T2.\text{NUMERO\_PDO} = \text{LINEAS\_PDO}.\text{NUMERO\_PDO}) \text{ LINEAS\_PDO}$

T3	NUMERO_PDO	FECHA_PDO	NOMBRE	DIRECCION	REFERENCIA_ART	CNTD_PDO	...
	1301	08/02/2001	Sánchez	Islas Pitiusas	AB03	3	...
	1301	08/02/2001	Sánchez	Islas Pitiusas	AB22	1	...

*Join natural entre LINEAS\_PDO y ARTICULOS:*

$T4 = T3 \text{ JOIN } (T3.\text{REFERENCIA\_ART} = \text{ARTICULOS}.\text{REFERENCIA\_ART}) \text{ ARTICULOS}$

T4	NUMERO_PDO	FECHA_PDO	NOMBRE	DIRECCION	REFERENCIA_ART	CNTD_PDO	NOMBRE_ART	PRECIO_UNIT_ART	...
	1301	08/02/2001	Sánchez	Islas Pitiusas	AB03	3	BICICLETA	3500	...
	1301	08/02/2001	Sánchez	Islas Pitiusas	AB22	1	CASCO	200	...

*Proyección de cálculo sencillo de IMPORTE y eliminación de las columnas inútiles:*

T5=πT4 (NUMERO\_PDO, FECHA\_PDO, NOMBRE, DIRECCION,  
REFERENCIA\_ART,CNTD\_PDO,  
NOMBRE\_ART, PRECIOUNIT\_ART, IMPORTE=PRECIOUNIT\_ART\*CNTD\_PDO)

T5	NUMERO_PDO	FECHA_PDO	NOMBRE	DIRECCION	REFERENCIA_ART	CNTD_PDO	NOMBRE_ART	PRECIOUNIT_ART	IMPORTE
	1301	08/02/2001	Sánchez	Islas Pitiusas	AB03	3	BICICLETA	3500	10500
	1301	08/02/2001	Sánchez	Islas Pitiusas	AB22	1	CASCO	200	200

*Proyección de cálculo agregado para BASEIMP:*

T6=πT5 (BASEIMP=SUM ( IMPORTE ) )

T6	BASEIMP
	10700

*Producto cartesiano para tener todas las columnas en la tabla resultante:*

T7=T5xT6

T7	NUMERO_PDO	FECHA_PDO	NOMBRE	DIRECCION	REFERENCIA_ART	CNTD_PDO	NOMBRE_ART	PRECIOUNIT_ART	IMPORTE	BASE_IMP
	1301	08/02/01	Sánchez	Islas Pitiusas	AB03	3	BICICLETA	3500	10500	10700
	1301	08/02/01	Sánchez	Islas Pitiusas	AB22					

## Administrar una base de datos

La creación y mantenimiento de una base de datos SQL Server implica una serie de operaciones que pertenecen a varios dominios, como por ejemplo:

- La gestión del espacio de almacenamiento.
- La configuración de la base de datos.
- La gestión de los objetos de la base de datos.
- La traducción de las restricciones del análisis.
- La gestión de la seguridad de acceso.
- Las copias de seguridad.

Algunos de estos dominios se extienden al ámbito de la administración y se estudiarán más adelante. La gestión y configuración de SQL Server se puede hacer

de dos maneras; usando Transact SQL, es decir, usando scripts, o de manera interactiva con Microsoft SQL Server Management Studio.

En SQL Server 2012 hay tres tipos de base de datos:

- Las bases OLTP (*OnLine Transaction Processing*), es decir, las bases de datos que van a soportar las transacciones de los usuarios. Este es el tipo de base de datos que hay en entornos de producción. Las principales características de este tipo de base de datos son que, a pesar de su importante volumen de datos y de la gran cantidad de usuarios conectados, los tiempos de respuesta deben ser óptimos. Afortunadamente, los usuarios trabajan con transacciones cortas y cada transacción maneja pocos datos.
- Las bases OLAP (*OnLine Analytical Processing*), es decir, las bases de datos que van a permitir almacenar un máximo de información, para crear consultas de ayuda en la toma de decisiones. Este libro no trata estas bases de datos decisionales.
- Las bases de datos de tipo snapshot, que son réplicas más o menos completas de la base de datos original. Sirven, por ejemplo, para acceder de manera rápida a datos remotos. Este tipo de base de datos no se trata en este libro.

Aquí solo se trata la noción de base de datos de tipo usuario. La gestión de las bases de datos de sistema no se menciona.

## **1. Administrar el espacio de almacenamiento**

SQL Server usa un conjunto de archivos para almacenar la información relativa a una base de datos.

### **Archivo principal**

Solo hay un archivo de este tipo por cada base de datos. Es el punto de entrada. Este archivo tiene extensión \*.mdf.

### **Archivos secundarios**

Por cada base de datos puede haber varios ficheros de este tipo. Tienen extensión \*.ndf.

### **Archivos de traza**

Estos archivos (puede haber varios) contienen la traza de las transacciones. Su extensión es \*.ldf.

### **Los grupos de archivos**

Es posible indicar un grupo de archivos cuando se establecen los archivos. Estos grupos tienen la ventaja de balancear la carga de trabajo entre los diferentes discos del sistema. Los datos se escriben de manera equitativa en los diferentes archivos del grupo.

### **Estructura de los archivos de datos**

Los archivos de datos se dividen en páginas de 8 KB. Los datos se almacenan en el interior de las páginas y el tamaño máximo de un registro es 8060 bits, sin contar los datos de tipo texto e imagen. Estos 8 KB de tamaño permiten:

- Mejores tiempos de respuesta en las operaciones de lectura/escritura.
- Soportar registros de datos más grandes y por tanto, hacer menos llamadas a datos de tipo texto e imagen.
- Mejorar la gestión de las bases de gran tamaño.

Estas páginas se agrupan en extensiones. Las extensiones están formadas por ocho páginas contiguas (64 KB). Representan la unidad de asignación de espacio para las tablas e índices. Para evitar perder espacio en disco, existen dos tipos de extensiones:

- Uniforme

Reservado a un único objeto.

- Mixto

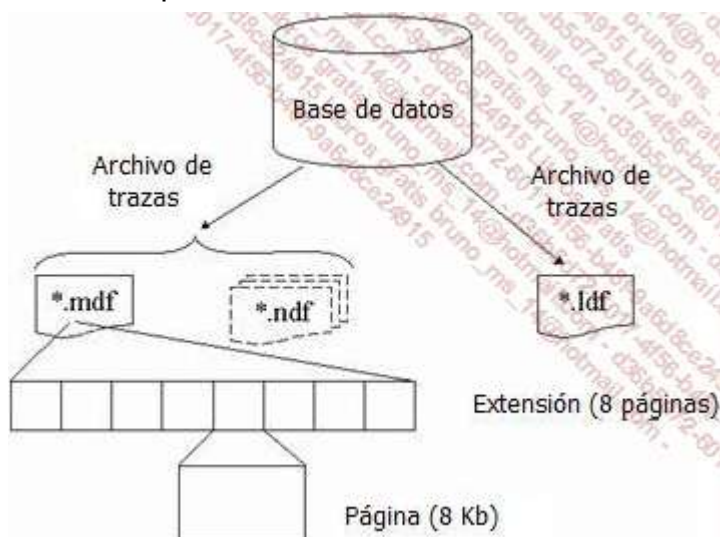
Compartido entre varios objetos, 8 como máximo.

Cuando se crea una tabla, las páginas se asignan con una extensión mixta. Cuando los datos representan ocho páginas, se asignan a la tabla extensiones uniformes.

## Los archivos de datos

Los archivos de datos se pueden redimensionar de manera dinámica o manual. Cuando se crea un archivo, hay que indicar:

- El nombre lógico del archivo, para manipularlo con el lenguaje Transact SQL.
- El nombre físico, para indicar la ubicación del archivo.
- Un tamaño inicial.
- Un tamaño máximo.
- Un paso de incremento.



## 2. Administrar el objeto DATABASE

Un DATABASE contiene el resto de objetos:

- El catálogo de la base de datos.
- Los objetos de usuario (tablas, vistas, triggers y procedimientos).
- Los índices, los tipos de datos y las restricciones.
- El archivo de traza de las transacciones.

La persona que crea la base de datos debe tener permisos suficientes y se convierte en propietario de la base de datos. SQL Server es capaz de administrar 32.767 bases de datos.

Los archivos que forman una base de datos no se deben ubicar en un sistema de archivos comprimido o en una ubicación de red compartida.

Cuando se crean los archivos, se inicializan con ceros para eliminar todos los datos existentes. Esta operación implica una sobrecarga de trabajo durante la creación de los archivos, pero permite optimizar el tiempo de respuesta cuando la base de datos está en producción.

Es posible asignar espacio en disco a la base de datos sin que sea necesario inicializarla con 0. Esta operación se conoce como inicialización instantánea. Los datos antiguos presentes en disco se eliminan a medida que lo necesita la base de datos.

SQL Server es capaz de utilizar particiones sin formatear para la creación de los archivos de base de datos. Sin embargo, en la mayor parte de los casos, el mejor método es crear los archivos en una partición NTFS.

Efectivamente, el uso de una partición sin formatear no permite indicar al sistema operativo, y por tanto al administrador, que SQL Server está utilizando esta partición. El espacio que no se usa desde un punto de vista del sistema, se puede utilizar fácilmente para extender una partición o crear una nueva. Usando particiones sin formatear, el riesgo de manipulaciones erróneas aumenta considerablemente en relación al beneficio, que es poco significativo.

Para terminar, solo es posible crear un único archivo de datos en cada partición sin formatear.

Los archivos creados en particiones NTFS soportan sin problemas la compactación NTFS y, eventualmente, se puede establecer un conjunto de archivos en modo solo lectura. Estas consideraciones solo se aplican a las bases de datos de usuario y no se pueden aplicar a los archivos de las bases de datos de sistema.

### a. Crear la base de datos

Para crear una base de datos hay que estar conectado como administrador de sistema o tener permisos para usar CREATE DATABASE, y estar en la base de datos de sistema maestro.



En primer lugar se debe crear el objeto DATABASE. Una base de datos contiene el resto de objetos:

- El catálogo de base de datos.
- Los objetos de usuario (tablas, vistas, triggers y procedimientos).
- Los índices, tipos de datos y restricciones de integridad.
- El archivo de traza de las transacciones.

El nombre de la base de datos debe ser único en una instancia SQL Server. Este nombre está limitado a 128 caracteres y debe respetar las reglas de construcción de identificadores. Esta longitud máxima se reduce a 123 caracteres si no se especifica el nombre del archivo de traza durante la creación de la base de datos.

## Sintaxis

```
CREATE DATABASE nombreBase[ ON [PRIMARY]
[( [ NAME = nombreLogico, ]
FILENAME = 'nombreFisico'
[, SIZE = tamaño]
[, MAXSIZE = { tamañoMaximo | UNLIMITED } ]
[, FILEGROWTH = valorIncremento] ) [,...]]
[ LOG ON { archivo } ]
[COLLATE nombreClasificacion]
[ FOR ATTACH | FOR ATTACH_REBUILD_LOG ]
NAME
```

Nombre lógico del archivo.

FILENAME

Ubicación y nombre físico del archivo.

SIZE

Tamaño inicial del archivo en megabytes (MB) o kilobytes (KB). Si no se especifica ningún tamaño, SQL Server considera que el tamaño se expresa en megabytes (MB). En cualquier caso, el tamaño se expresa con un valor entero.

MAXSIZE

Tamaño máximo del archivo en kilo o megabytes (por defecto megabytes). Si no se especifica ningún valor, el tamaño del archivo estará limitado por el espacio disponible en disco.

UNLIMITED

Sin tamaño máximo, el límite es el espacio libre en disco.

## FILEGROWTH

Indica el paso de incremento para el tamaño del archivo, que nunca podrá ser superior al valor máximo. Este paso se puede expresar como un porcentaje o de manera estática en kilo o megabytes. Las extensiones tienen un tamaño de 64 KB. Por lo tanto, hay que fijar el valor mínimo del paso.

Si este valor no se especifica, el incremento es de 1 megabyte para los archivos de datos y del 10% para los archivos de traza.

## LOG ON

Ubicación del archivo de traza de las transacciones. El archivo de trazas de las transacciones almacena las modificaciones que se realizan sobre los datos. Por cada INSERT, UPDATE o DELETE, se registra una traza en el archivo de trazas antes de escribir en la base de datos. La validación de las transacciones también se registra en el archivo de trazas. Este archivo de trazas sirve para recuperar los datos en caso de error.

## COLLATE

Indica la clasificación por defecto de la base de datos. El nombre de la clasificación puede ser una clasificación SQL o Windows. Si no se indica, se usa la clasificación por defecto de la instancia SQL Server.

## FOR ATTACH

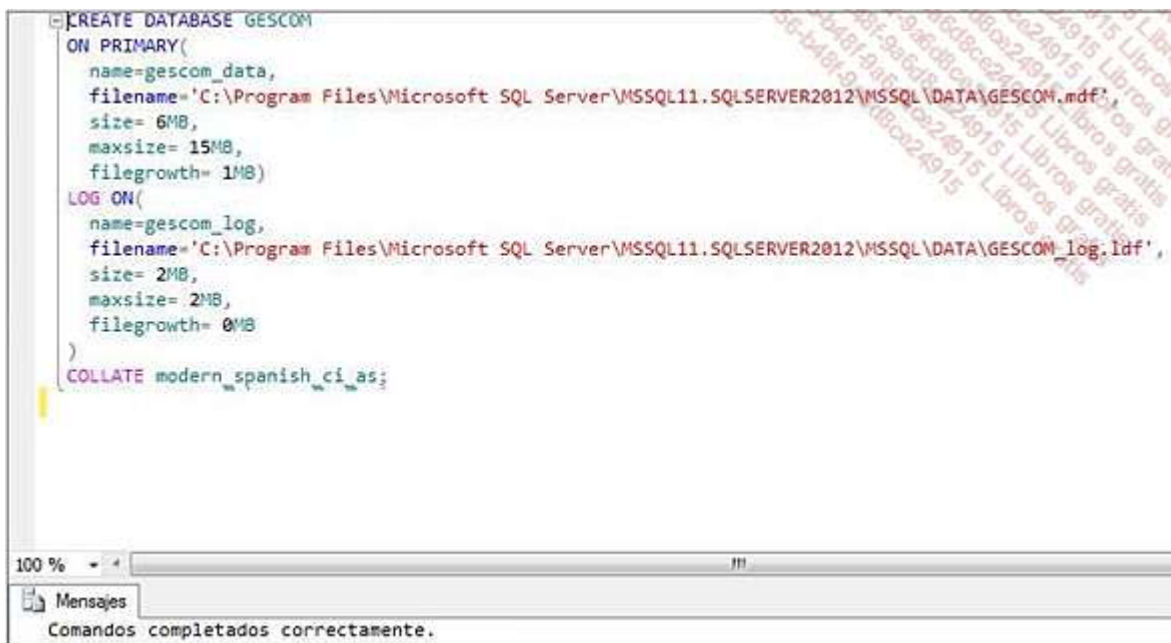
Permite crear una base de datos usando los archivos ya creados. La base de datos que se debe adjuntar debe ser de un nivel de compatibilidad 90 (es decir, SQL 2005), como mínimo.

## FOR ATTACH\_REBUILD\_LOG

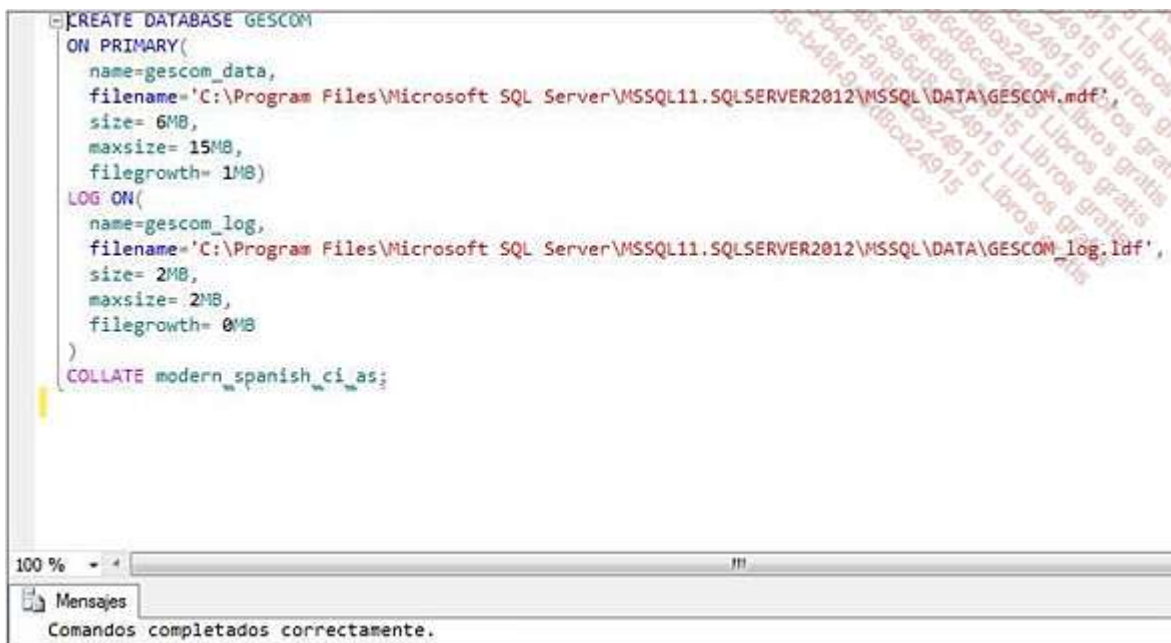
Con esta opción, es posible crear la base de datos adjuntándole los archivos de datos (mdf y ndf), pero no necesariamente los archivos de traza. En este caso, los archivos de traza se crean vacíos. Si se adjunta una base de datos de esta manera, es importante hacer rápidamente una copia de seguridad completa de la base de datos y planificar todos los procesos de copia de seguridad. No es posible utilizar copias de seguridad que se hayan hecho antes de adjuntar los archivos, ya que las secuencias de traza ya no corresponden.

## Ejemplo

*Creación de la base de datos Gescom (6 MB) con el archivo de trazas de transacción (2 MB).*



Por supuesto, también es posible hacer esta operación desde la consola gráfica SQL Server Management Studio. Para hacerlo, después de haber seleccionado el nodo Bases de datos desde el explorador, es necesario seleccionar **Nueva base de datos** desde el menú contextual. Aparecerá la siguiente pantalla.



Desde este cuadro de diálogo, es posible definir las diferentes opciones de creación de la base de datos.

## b. Modificar el tamaño

Es posible incrementar o disminuir el tamaño de los archivos, automática o manualmente.

### Incrementar el tamaño

Si se especifican un paso de incremento (FILEGROWTH) y un tamaño máximo durante la creación del archivo, el archivo cambiará de tamaño en función de las necesidades.

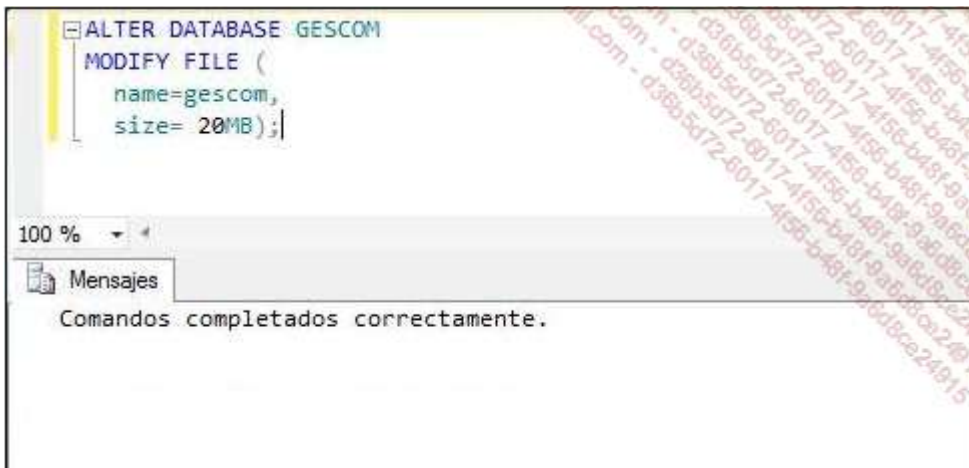
Es posible modificar manualmente el tamaño, el tamaño máximo y la tasa de incremento de un archivo de datos, usando el comando ALTER DATABASE.

```
ALTER DATABASE nombre  
MODIFY FILE
```

```
(NAME=nombreLogico  
[, SIZE=tamaño]  
[, MAXSIZE=tamañoMaximo]  
[FILEGROWTH=valorIncremento])
```

### Ejemplo

*Incrementar el tamaño de un archivo existente:*

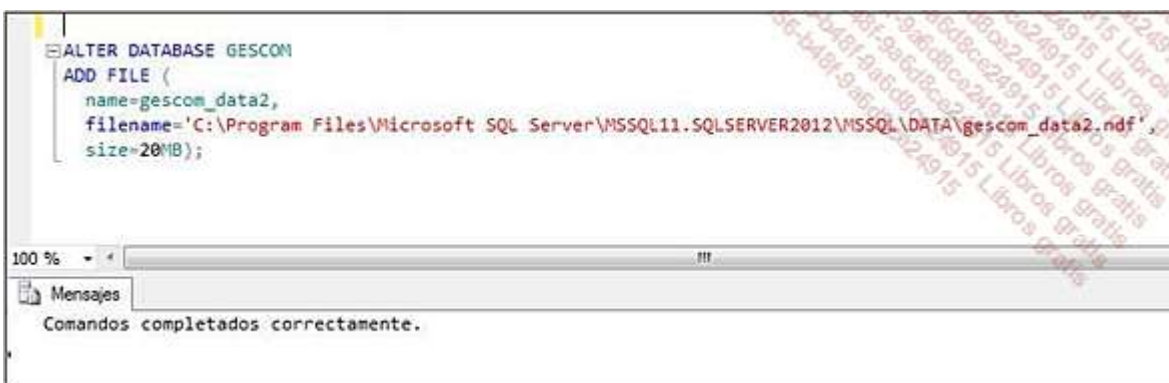


También es posible añadir archivos.

```
ALTER DATABASE nombre  
ADD FILE (  
NAME = nombreLogico,  
FILENAME = 'nombreFisico'  
[, SIZE = tamaño]  
[, MAXSIZE = { tamañoMaximo | UNLIMITED } ]  
[, FILEGROWTH = valorIncremento] )
```

### Ejemplo

*Añadir un segundo archivo a la base de datos GESCOM:*



El comando ALTER DATABASE permite una acción mucho más importante sobre la base de datos que la simple modificación del tamaño de los archivos. También es posible añadir y eliminar archivos y grupos de archivos, modificar el nombre de la base de datos, indicar el modo de fin por defecto de las transacciones actuales y cambiar la clasificación de la base de datos.

Para conocer las diferentes clasificaciones disponibles en el servidor, hay que ejecutar la siguiente consulta:

```
SELECT * FROM ::fn_helpcollations()
```

## Reducir el tamaño

El tamaño de los archivos se puede reducir de manera automática, si se ha seteado la opción **auto-shrink** en la base de datos.

Los comandos DBCC SHRINKFILE y DBCC SHRINKDATABASE permiten ejecutar manualmente la reducción del tamaño. DBCC SHRINKFILE solo se aplica sobre un archivo concreto, mientras que DBCC SHRINKDATABASE va a recorrer todos los archivos de la base de datos.

La operación de reducción de los archivos, siempre empieza por el final del archivo. Por ejemplo, si la base de datos tiene un archivo de 500 MB que deseamos reducir a 400 MB, los últimos 100 megabytes del archivo se van a reorganizar antes de liberarlo para evitar el bloqueo de datos.

Cuando se activa la opción auto-shrink, no permite controlar el desarrollo de las operaciones de reducción de tamaño. Por lo tanto, es prudente tener en cuenta este aspecto antes de activar la opción porque puede dificultar las operaciones de administración preventiva (como el incremento de archivos), que el administrador haya puesto en marcha.

La cantidad de espacio que se libera realmente, es función del tamaño ideal que se ha fijado como argumento en el comando DBCC SHRINKFILE y de la realidad de los datos. Si en el ejemplo anterior el archivo contiene 450 MB de extensiones utilizadas, entonces solo se podría liberar 50 MB de espacio.

## Sintaxis

```
DBCC SHRINKFILE (nombreArchivo { [ , EMPTYFILE ] | [ , nuevoTamaño ]  
[ , NOTRUNCATE | TRUNCATEONLY ] } ) [ WITH NO_INFOMSGS ]  
DBCC SHRINKDATABASE (nombreBase [ , nuevoPorcentaje ]  
[ , { NOTRUNCATE | TRUNCATEONLY } ] )  
[ WITH NO_INFOMSGS ]
```

DBCC SHINKFILE

Reduce el tamaño del archivo de datos o del archivo de trazas para la base de datos que se especifica.

DBCC SHRINKDATABASE

Reduce el tamaño de los archivos de datos en la base de datos que se especifica.

nuevoTamaño

Indica el tamaño que queremos que tenga el archivo, después de la reducción de tamaño.

nuevoPorcentaje

Indica el porcentaje de espacio libre que queremos obtener en el archivo de datos, después de la reducción de tamaño de la base de datos.

EMPTYFILE

Permite indicar al comando DBCC\_SHRINKFILE que transfiera todos los datos presentes en este archivo de datos, a otro archivo del mismo grupo. Cuando esté vacío, el archivo se podrá eliminar con un comando ALTER TABLE.

NOTRUNCATE

Reorganiza el archivo situando las páginas ocupadas al inicio del archivo, pero no se reduce el tamaño.

TRUNCATEONLY

Corta el archivo sin reorganizarlo.

WITH NO\_INFOMSGS

No se muestran los mensajes de una gravedad inferior a 10.

### c. Eliminar la base de datos

El comando DROP DATABASE permite eliminar la base de datos. Los archivos físicos también se eliminan.

Si se separan algunos archivos de la base de datos antes de eliminarla, estos no se eliminan y será necesario hacerlo manualmente desde el explorador de archivos.

Para terminar, no se puede eliminar la base de datos si hay usuarios conectados. Para forzar la desconexión de estos usuarios y permitir la eliminación de la base de datos, es necesario cambiar a modo SINGLE\_USER con la instrucción ALTER DATABASE.

La instrucción DROP DATABASE solo se puede ejecutar si el modo autocommit está activado (es el caso por defecto). No es posible eliminar las bases de datos de sistema.

### d. Renombrar una base de datos

Es posible renombrar una base de datos con la instrucción ALTER DATABASE.

#### Sintaxis

```
ALTER DATABASE {nombreBase | CURRENT} MODIFY NAME = nuevoNombreBase
```

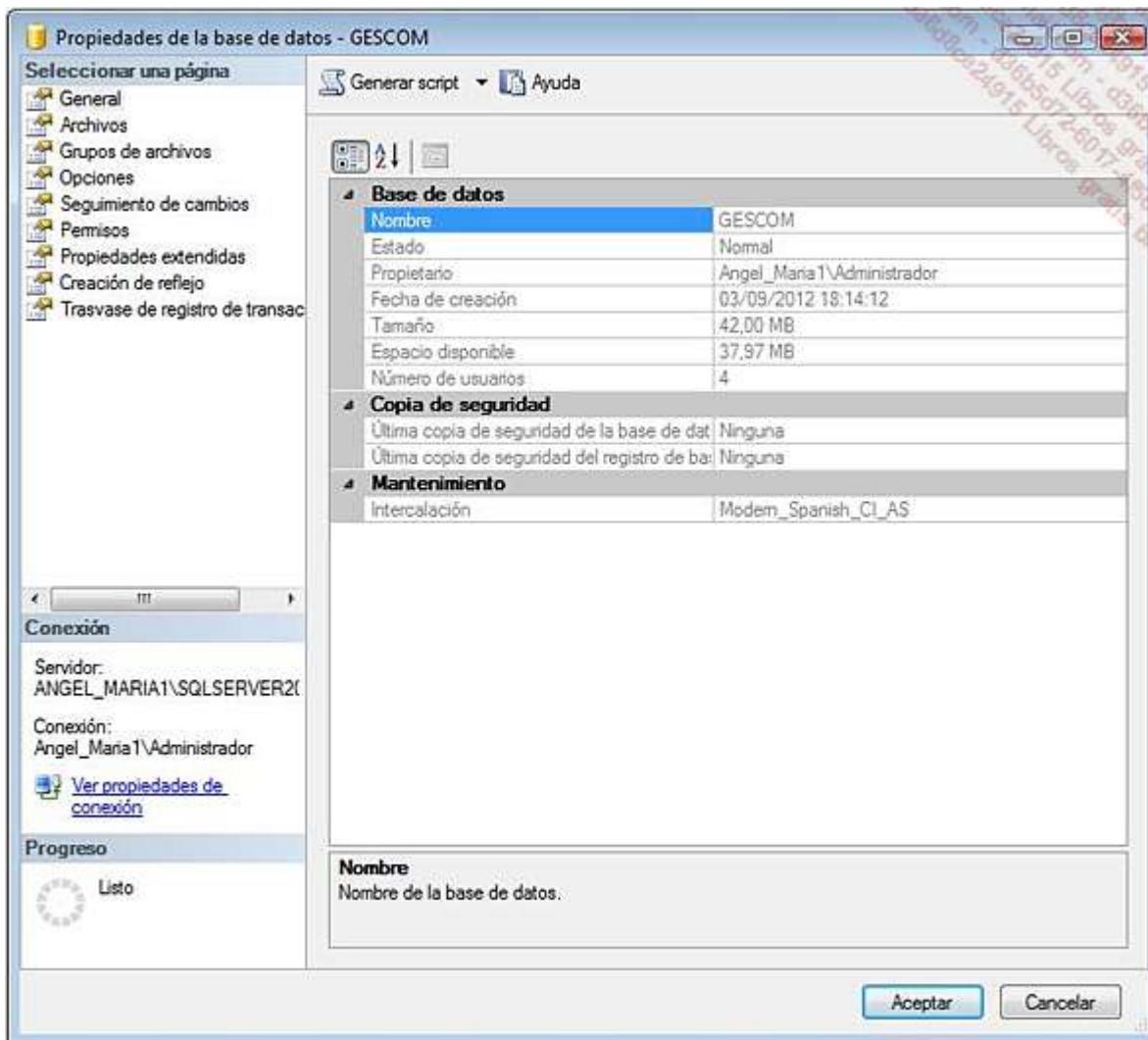
CURRENT

Esta palabra clave permite indicar que la instrucción ALTER DATABASE se ejecuta en la base de datos desde la que se ha ejecutado la instrucción.

Se mantiene el procedimiento **sp\_renamedb** por razones de compatibilidad hacia atrás. Hay que tener cuidado para no volver a usarlo en SQL Server 2012.

### e. Configurar una base de datos

Es posible configurar una base de datos para fijar un determinado número de opciones y obtener el comportamiento deseado de la base de datos, en función de las necesidades de los usuarios. Podemos acceder a estas diferentes opciones de manera gráfica usando SQL Server Management Studio. Para ello, hay que situarse en la base de datos y llamar a la ventana de propiedades con la tecla [F4], mediante el menú contextual asociado a la base de datos o incluso usando el menú **Ver - Ventana propiedades** del menú general de SQL Server Management Studio.



El procedimiento **sp\_dboption**, que todavía existía en SQL Server 2008 por razones de compatibilidad hacia atrás, ya no está disponible en SQL Server 2012. Es necesario usar la instrucción ALTER DATABASE para definir las opciones de configuración de la base de datos.

La instrucción ALTER DATABASE tiene muchas opciones. A continuación se listan las principales.

## Sintaxis

```
ALTER DATABASE {nombreBase | CURRENT} SET opción;
```

## Estado de la base de datos

ONLINE

Permite hacer visible de nuevo la base de datos.

OFFLINE

Permite hacer inaccesible la base de datos. La base de datos se para y se cierra correctamente. No es posible hacer operaciones de mantenimiento sobre una base de datos offline.

EMERGENCY

La base de datos solo es accesible en modo lectura, el registro de las trazas está desactivado y su acceso solo está permitido a los administradores del servidor.

## Acceso

SINGLE\_USER

Acceso limitado a un único usuario.

RESTRICTED\_USER

Sólo se pueden conectar a la base de datos los miembros con roles db\_owner, dbcreator o sysadmin.

MULTI\_USER

Es el modo predeterminado y permite acceder a la información a todos los usuarios con permisos suficientes.

## Operaciones posibles

READ\_ONLY

La base de datos solo está disponible en modo lectura.

READ\_WRITE

La base de datos solo está disponible en modo lectura/escritura.

## Configuración

ANSI\_NULL\_DEFAULT

Permite indicar el comportamiento de las columnas con valor NULL. Si se activa este parámetro (valor ON), por defecto las columnas aceptan valores NULL y los prohíben en caso contrario (valor OFF).

RECURSIVE\_TRIGGERS

Autoriza la recursividad de los triggers.

TORN\_PAGE\_DETECTION

Permite detectar páginas incompletas.

AUTO\_CLOSE

La base de datos se detiene y se liberan los recursos cuando se desconecta el último usuario.

AUTO\_SHRINK

Los archivos de la base de datos se podrán reducir de tamaño automáticamente.

AUTO\_CREATE\_STATISTICS

Se crean todas las estadísticas que no existen cuando se optimiza una consulta. Por defecto, esta opción está activada (valor ON).

Este tipo de configuración corresponde a la mayor parte de las bases de datos.

AUTO\_UPDATE\_STATISTICS

Se recalculan todas las estadísticas obsoletas para la correcta optimización de una consulta.

Por defecto esta opción está activada (valor ON). Así se garantiza la validez de las estadísticas.



## AUTO\_UPDATE\_STATISTICS\_ASYNC

Se actualizan de manera asíncrona las estadísticas que permiten representar la pertinencia de los índices. La consulta que provoca la actualización de las estadísticas no espera que las estadísticas estén actualizadas para ejecutarse. Son las futuras consultas las que se beneficiarán de esta actualización. Por defecto, esta opción no está activada (valor OFF).

## QUOTED\_IDENTIFIERS

Los identificadores delimitados se pueden poner entre comillas.

## ANSI\_NULLS

Si el parámetro está activado (valor ON), todas las comparaciones con un valor NULL se evalúan como desconocido. Si el parámetro no está activo (valor OFF), las comparaciones con valores NULL y con valores no Unicode, se evalúan como verdadero si los dos valores son NULL.

Este valor también se puede configurar en SQL Server 2012, pero en versiones futuras este parámetro se activará por defecto (valor ON).

## ANSI\_WARNINGS

Permite dar mensajes de error o advertencia (warnings) cuando se cumplen algunas condiciones.

## ARITHABORT

Permite detener el tratamiento del lote de instrucciones cuando se supera la capacidad o hay una división por cero.

## CONCAT\_NULL\_YIELDS\_NULL

El resultado es NULL si uno de los dos operadores en una operación de concatenación es NULL.

## CURSOR\_CLOSE\_ON\_COMMIT

Permite cerrar todos los cursores durante la definición de una transacción o durante la terminación de una transacción.

## CURSOR\_DEFAULT

Las declaraciones de cursor tienen el valor por defecto LOCAL.

Permite especificar si la declaración del cursor tiene un alcance limitado al lote Transact SQL en el que se define (LOCAL) o si el cursor es visible desde cualquier lote Transact SQL, que se ejecute en la misma conexión.

## NUMERIC\_ROUNDABORT

Se genera un error si hay pérdida de precisión durante un cálculo.

## PAGE\_VERIFY

Esta opción permite validar la calidad de la información almacenada en cada página. SQL Server recomienda la opción por defecto CHECKSUM.

## PARAMETERIZATION

En modo SIMPLE, por defecto las consultas se configuran en función de las reglas en vigor en el servidor. Con el modo FORCED, SQL Server configura todas las consultas, antes de aplicar el plan de ejecución.

## Modo de registro de trazas

### RECOVERY SIMPLE

Permite minimizar el espacio que ocupa el archivo de trazas, ya que SQL Server libera el espacio tan pronto como la información deja de ser útil en el marco de la restauración automática (el archivo de trazas se trunca en cada punto de sincronización).

### RECOVERY BULK\_LOGGED

Permite optimizar el tamaño del archivo de trazas para permitir la restauración de los datos después de un problema en un archivo, y restaurar las operaciones que afectan a los datos, como los índices.

### RECOVERY FULL

Permite conservar en el archivo de trazas toda la información necesaria para autorizar la restauración, después de un problema en un archivo de datos.

## Gestión de transacciones

### ROLLBACK AFTER nombre

La anulación de las transacciones es efectiva después de unos segundos de espera.

### ROLLBACK IMMEDIATE

La anulación de la transacción es inmediata.

### NO\_WAIT

Si la transacción no accede inmediatamente a los recursos que necesita, se anula.

### ANSI\_PADDING

Para los datos de tipo carácter, permite especificar si se tienen que eliminar o no los espacios a la derecha.

### COMPATIBILITY

Permite fijar el nivel de compatibilidad de la base de datos: 90 en SQL Server 2005, 100 en SQL Server 2008 y 110 en SQL Server 2012.

### DATE\_CORRELATION\_OPTIMIZATION

Con esta opción, SQL Server se encarga de mantener la correlación de las estadísticas entre dos tablas relacionadas por una restricción de clave extranjera y que tienen las dos una columna de tipo datetime.

## Acceso externo

### DB\_CHAINING

Permite administrar los contextos de seguridad durante el acceso a la base de datos desde otra base de datos.

### TRUSTWORTHY

Los módulos internos (procedimientos almacenados y funciones), pueden acceder a los recursos externos al servidor, usando un contexto de impersonalización.

## Service Broker

ENABLE\_BROKER

Activa el service Broker.

DISABLE\_BROKER

Desactiva el service Broker.

NEW\_BROKER

Permite indicar que la base de datos debe recibir un nuevo identificador Service Broker.

ERROR\_BROKER\_CONVERSATIONS

Las conversaciones actuales van a recibir un mensaje de error y se van a cerrar.

## **Snapshot (Captura instantánea)**

ALLOW\_SNAPSHOT\_ISOLATION

Cuando este modo está activado, todas las transacciones pueden trabajar con una captura instantánea (snapshot) de la base de datos, tal y como es justo antes del inicio de la transacción.

READ\_COMMITTED\_SNAPSHOT

Cuando este modo está activado, todas las instrucciones ven los datos, tal y como eran antes del inicio de la instrucción.

# **Administrar tablas e índices**

## **1. Identificador**

Todos los elementos que se crean en SQL Server, se identifican perfectamente con un nombre que se usa como identificador. Efectivamente, dos objetos de un mismo tipo no pueden tener el mismo nombre, si se definen al mismo nivel. Por ejemplo, en una instancia de SQL Server, no es posible tener dos bases de datos con el mismo nombre, pero es posible si las bases de datos se definen en dos instancias distintas de SQL Server. De la misma manera, dentro de una base de datos, no es posible tener dos tablas con el mismo nombre. Los objetos se pueden manipular con SQL, gracias al identificador. Por lo tanto, es importante definir correctamente estos identificadores.

Los identificadores se componen desde 1 hasta 128 caracteres. Siempre empiezan por una letra o por uno de los siguientes caracteres: \_, @, #.

Los siguientes caracteres son caracteres alfanuméricos.

Por supuesto, ningún identificador puede ser igual a una palabra clave de Transact SQL.

Existen dos categorías de identificadores: regulares y delimitados.

### **Los identificadores regulares**

Esta categoría de identificadores es la que más se utiliza y es la que se debe usar con preferencia. Efectivamente, este tipo de identificadores está presente en todas las bases de datos y son muy sencillos a nivel de la escritura de las consultas, ya que no distingue entre mayúsculas y minúsculas.

### Ejemplo

EdicionesENI, RecursosInformaticos

## Los identificadores delimitados

Esta categoría de identificadores permite conservar los caracteres especiales en los identificadores, como los caracteres acentuados, los espacios, etc. y conservar la distinción entre mayúsculas y minúsculas. Estos identificadores se usan entre corchetes [ ] o entre comillas ". El uso de este tipo de identificadores raramente permite ganar en claridad, ya que la escritura de las consultas es más pesada. Por tanto, es mejor usar los identificadores regulares.

### Ejemplo

[Ediciones ENI], "Recursos Informaticos",...

## 2. Los tipos de datos

Durante la definición de una columna, especificamos el formato de uso del dato, así como el modo de almacenamiento para el tipo de columna.

### a. Tipos de datos de sistema

Estos tipos están disponibles para todas las bases de datos estándares.

#### Caracteres

`char (n)`

Cadena de caracteres de longitud fija, de n caracteres como máximo. Por defecto 1 y 8.000 como máximo.

`varchar (n|max)`

Cadena de caracteres de longitud variable, de n caracteres como máximo. Por defecto 1 y 8.000 caracteres como máximo. Si se indica max, la variable puede contener datos de tipo texto hasta  $2^{31}$  caracteres.

El espacio que ocupa en disco corresponde al número de caracteres introducidos más dos bytes.

`nchar (n)`

Cadena de caracteres unicode, con un máximo de 4.000.

`nvarchar (n|max)`

Cadena de caracteres unicode, con un máximo de 4.000. Si se indica max, la variable puede contener datos de tipo texto con  $2^{31}$  bytes como máximo.

El tipo sysname, que aparece cuando trabajamos con las tablas de sistema, se utiliza para hacer referencia a los nombres de objetos. Este tipo es idéntico a un `nvarchar(128)` con la particularidad de que los valores nulos no están permitidos.

#### Numéricos

`decimal (p[,d])`

Numérico exacto con precisión p (número total de cifras), con d cifras a la derecha de la coma.

p está comprendido entre 1 y 38, 18 por defecto.

d está comprendido entre 1 y p, 0 por defecto.

Ejemplo: para un valor decimal (8,3) el intervalo permitido será de -99.999,999 a +99.999,999.

Los valores gestionados van de  $-10^{38}$  a  $10^{38} - 1$ .

`numeric [(p[,d))]`

Idéntico a decimal. Para el tipo decimal, algunas veces la precisión podría ser mayor de lo necesario.

`bigint`

Tipo de datos entero codificado con 8 bytes. Los valores almacenados con este tipo de datos están comprendidos entre  $-2^{63}$  (-9.223.372.036.854.775.808) y  $2^{63}-1$  (9.223.372.036.854.775.807).

`int`

Número entero entre  $-2^{31}$  (-2.147.783.648) y  $+2^{31}-1$  (+2.147.483.647). El tipo de datos **int** es específico de SQL Server y su sinónimo **integer** es compatible con ISO.

`smallint`

Número entero entre  $-2^{15}$  (-32.768) y  $2^{15}-1$  (+32.767).

`tinyint`

Número entero positivo entre 0 y 255.

`float[(n)]`

Numérico aproximado de n cifras, con n entre de 1 y 53.

`real`

Idéntico a float(24).

`money`

Numérico en formato moneda, comprendido entre -922.337.203.685.477,5.808 y +922.337.203.685.477,5.807 (8 bytes).

`smallmoney`

Numérico en formato moneda, comprendido entre -214.748,3.648 y +214.748,3.647 (4 bytes).

## Binarios

`binary[(n)]`

Dato binario de n bytes (de 1 a 8.000), la longitud es fija.

`varbinary (n|max)`

Dato binario de longitud variable de n bytes (de 1 a 8.000). La opción max permite reservar un espacio de 231 bytes como máximo.

## Fecha

Para la gestión de los datos de tipos fecha y hora, SQL Server 2012 ofrece tipos de datos para optimizar el almacenamiento. Estos tipos de datos permiten una gestión más fina de los datos de tipos fecha y hora. En primer lugar, existen tipos particulares para almacenar datos de tipo hora y otros para almacenar datos de tipo

fecha. Esta separación es beneficiosa porque permite dar mayor precisión, limitando el espacio utilizado por los datos de uno u otro tipo. Por ejemplo, ¿es necesario conservar los datos de tipo hora, minutos y segundos cuando vamos a almacenar la fecha de nacimiento de un cliente? Sin lugar a dudas, la respuesta es no. El simple hecho de conservar esta información puede provocar errores más adelante, durante las operaciones de cálculo. Por lo tanto, es más razonable y tiene mejor rendimiento que este dato sea de un tipo que sólo guarde la fecha.

Estos tipos de datos, para los datos de tipo fecha y hora, permiten una mayor compatibilidad con otros sistemas de gestión de datos y facilita las operaciones de recuperación de datos.

Con los tipos `datetime2` y `datetimeoffset`, SQL Server ofrece la posibilidad de conservar la información de tipo fecha y hora de manera simultánea.

El tipo `datetimeoffset` permite almacenar información de tipo fecha y hora con una precisión que puede llegar a 100 nanosegundos y además, se guarda la hora en formato UTC y la diferencia (en número de horas) entre esta hora UTC y la zona horaria con la que trabaja el usuario que introduce el dato en la base de datos.

Los tipos `datetime` y `smalldatetime` siempre están presentes en SQL Server, pero es mejor utilizar los tipos `time`, `date`, `datetime2` y `datetimeoffset` en los nuevos desarrollos. Efectivamente, estos tipos ofrecen mayor precisión y respetan los estándares SQL.

`datetime`

Permite almacenar una fecha y una hora con 8 bytes. 4 para el número de días desde el 1 de enero de 1.900, 4 para el número de milisegundos desde medianoche. Las fechas se gestionan desde el 1 de enero de 1.753 hasta el 31 diciembre de 9.999. Las horas se gestionan con una precisión de 3,33 milisegundos.

`smalldatetime`

Permite almacenar una fecha y una hora con 4 bytes. Las fechas se gestionan desde el 1 de enero de 1.900 hasta el 6 junio de 2.079, con precisión de un minuto.

`datetime2`

Más preciso que el tipo `datetime`, permite almacenar un dato de tipo fecha y hora entre el 01/01/0001 y el 31/12/9.999, con una precisión de 100 nanosegundos.

`datetimeoffset`

Permite almacenar un dato de tipo fecha y hora entre el 01/01/0001 y el 31/12/9.999, con una precisión de 100 nanosegundos. La información horaria se almacena en formato UTC y la diferencia horaria se conserva para encontrar la hora local indicada inicialmente.

`date`

Permite almacenar una fecha entre el 01/01/0001 y el 31/12/9.999, con una precisión de un día.

`time`

Permite almacenar un dato positivo de tipo hora inferior a 24:00, con una precisión de 100 nanosegundos.

## **Especiales**

`bit`

Valor entero que puede tomar los valores 0, 1 o nulo.

Si una tabla tiene hasta 8 columnas de tipo bit, solo se usa un byte de tamaño en disco.

timestamp

Dato cuyo valor se actualiza automáticamente cuando el registro se modifica o inserta .

uniqueidentifier

Permite crear un identificador único basándose en la función NEWID().

sql\_variant

El tipo de datos **sql\_variant** permite almacenar cualquier tipo de datos, excepto los de tipo **text**, **ntext**, **timestamp** y **sql\_variant**. Si una columna usa este tipo de datos, los diferentes registros de la tabla pueden almacenar en esta columna datos de diferente tipo. Una columna de tipo **sql\_variant** puede tener una longitud máxima de 8.016 bytes. Antes de usar en una operación un valor almacenado en formato **sql\_variant**, es necesario convertir los datos a su formato inicial. Las columnas que usan el tipo **sql\_variant**, pueden participar en restricciones de clave primaria, extranjera o de unicidad, pero los datos contenidos en la clave de un registro no pueden superar los 900 bytes (límite impuesto por los índices). **sql\_variant** no se puede usar en las funciones CONTAINSTABLE y FREETEXTTABLE.

table

Es un tipo de datos particular que permite almacenar y devolver un conjunto de valores para usarlos en el futuro. El modo principal de uso de este tipo de datos es la creación de una tabla temporal.



xml

Este tipo permite almacenar un documento XML en una columna de una tabla relacional.

El uso del tipo XML se detalla en el capítulo Los tipos avanzados - Trabajar con el formato XML.

Hierarchyid

Este tipo de datos permite almacenar una estructura jerárquica en una tabla relacional. El uso de datos de este tipo se presenta en el capítulo Los tipos avanzados - Las estructuras jerárquicas.

Cursor

Este tipo de datos es específico de Transact SQL porque permite hacer referencia a un cursor Transact SQL. El uso de los cursores se detalla en el capítulo Transact SQL: el lenguaje procedimental - Gestión de cursores.

Los tipos `text`, `ntext` e `images` se mantienen por razones de compatibilidad hacia atrás. Ahora es mejor utilizar `varchar(max)`, `nvarchar(max)` y `varbinary(max)`.

SQL Server también ofrece sinónimos para sus propios tipos básicos. Los sinónimos normalmente están presentes para asegurar la compatibilidad con el estándar ISO.

Sinónimo	Tipo SQL Server
Caracteres char varying character( <i>n</i> ) character varying( <i>n</i> ) national character( <i>n</i> ) national char( <i>n</i> ) national character varying( <i>n</i> ) national char varying( <i>n</i> ) national text	varchar char( <i>n</i> ) varchar( <i>n</i> ) nchar( <i>n</i> ) nchar( <i>n</i> ) nvarchar( <i>n</i> ) nvarchar( <i>n</i> ) ntext
Numéricos dec double precision integer	decimal float int
Binarios binary varying	varbinary
Otros rowversion	timestamp

b. Tipos de datos definidos por el usuario

Es posible definir sus propios tipos de datos con SQL Server Management Studio o usando

el comando `CREATE TYPE`.

Los procedimientos almacenados **sp\_addtype** y **sp\_droptype** se mantienen por razones de compatibilidad hacia atrás. Microsoft recomienda no seguidos utilizando porque se eliminarán en futuras versiones de SQL Server.

## Sintaxis (creación)

```
CREATE TYPE nombreTipo
{FROM tipoBasico [ ( longitud [ , precision ] ) ]
  [ NULL | NOT NULL ]
  | EXTERNAL NAME nombreAssembly [ .nombreClase]
} [ ; ]
```

Es posible definir un tipo de datos basándose en la definición de una clase. Esta opción está relacionada con la integración de CLR en SQL Server.

Para poder integrar en SQL Server un tipo definido en SQL Server, se deben cumplir varias condiciones. En primer lugar, se debe configurar SQL Server para ejecutar código CLR (*Common Language Runtime*). Para esto, hay que activar la opción **clr\_enabled** con el procedimiento **sp\_configure**. Para que el servidor tenga mejor rendimiento, esta opción está desactivada por defecto. Después el assembly generado por Visual Studio se debe integrar en SQL Server. Para esto se usa la instrucción **CREATE ASSEMBLY**.

## Sintaxis (eliminación)

```
DROP TYPE [ schema_name. ] type_name [ ; ]
```

Un tipo no se podrá eliminar si se está usando en una tabla de la base de datos en la que se creó.



## Ejemplos


Creación de un tipo para las columnas como el nombre del cliente, nombre del proveedor, etc.:



```
CREATE TYPE tipoNombre
FROM VARCHAR(30);
```

Mensajes  
Comandos completados correctamente.

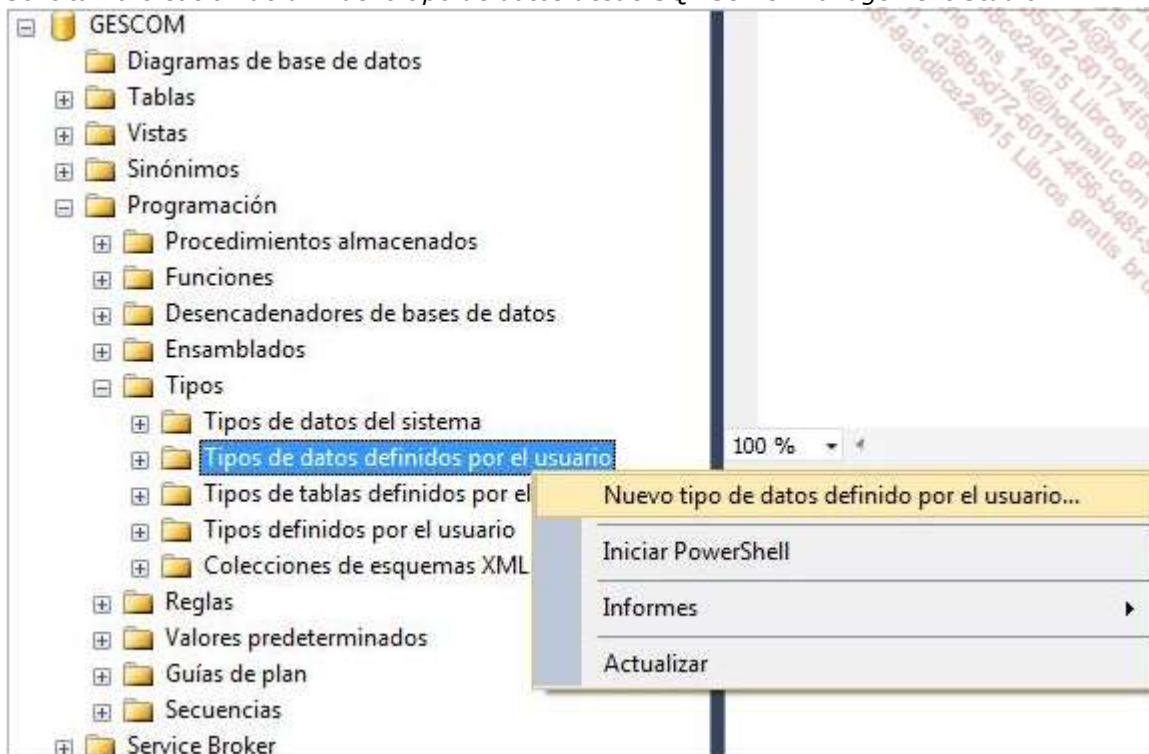
Creación de un tipo para los valores entre -999 y +999:



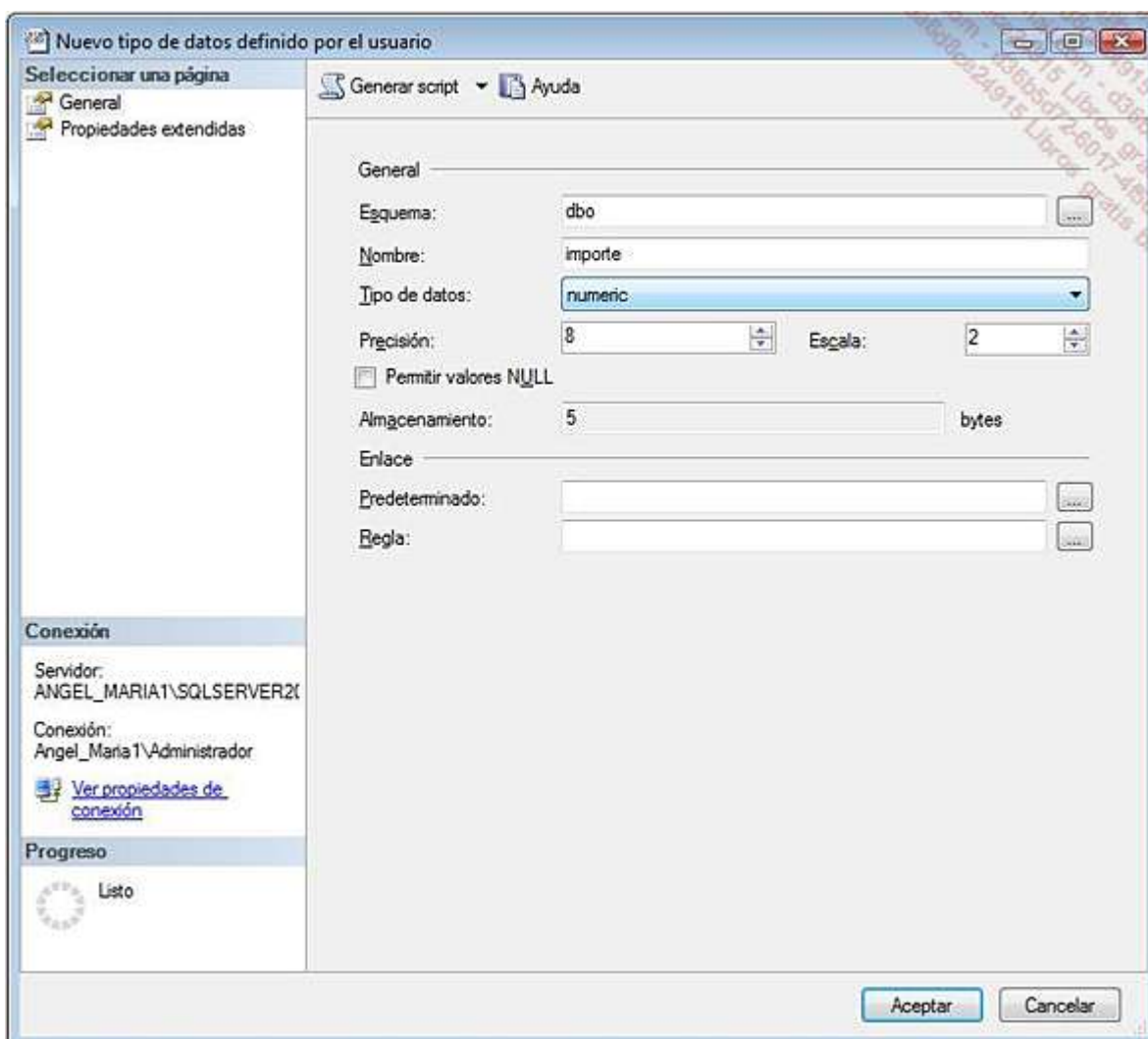
```
CREATE TYPE entero_3
FROM numeric(3) not null;
```

Mensajes  
Comandos completados correctamente.

Solicitar la creación de un nuevo tipo de datos desde SQL Server Management Studio:



Definición del nuevo tipo de datos "importe" desde SQL Server Management Studio:



A nivel Transact SQL, la instrucción CREATE TYPE también permite crear tipos compuestos por varios campos. Estos tipos normalmente se llaman "tipos estructurados" en los lenguajes de programación.

En lo que respecta a SQL Server, la instrucción CREATE TYPE permite crear un tipo TABLE o tabla. Cada columna que participa de la definición de este nuevo tipo, se define de la misma manera que una columna en una tabla. De esta manera es posible definir las restricciones de integridad de clave primaria (PRIMARY KEY), unicidad (UNIQUE), validación (CHECK) y no nulidad. Estas restricciones se pueden definir a nivel de columna o de tabla. También es posible definir una columna de tipo identidad.

La instrucción CREATE TYPE permite crear tipos llamados fuertemente tipados, porque las restricciones de integridad permiten una definición más precisa del posible formato de los datos.

La introducción de este nuevo tipo va a permitir definir los argumentos de funciones o procedimientos de tipo tabla. Hablaremos de una tabla `value parameter`.

## Sintaxis

```
CREATE TYPE nombreTipo AS TABLE (  
columna tipoColumna [restriccionColumna], ...)
```

`nombreTipo`

Nombre del tipo creado.

`Columna`

Nombre de la columna que participa en la definición de este nuevo tipo. Se pueden definir varias columnas.

`tipoColumnas`

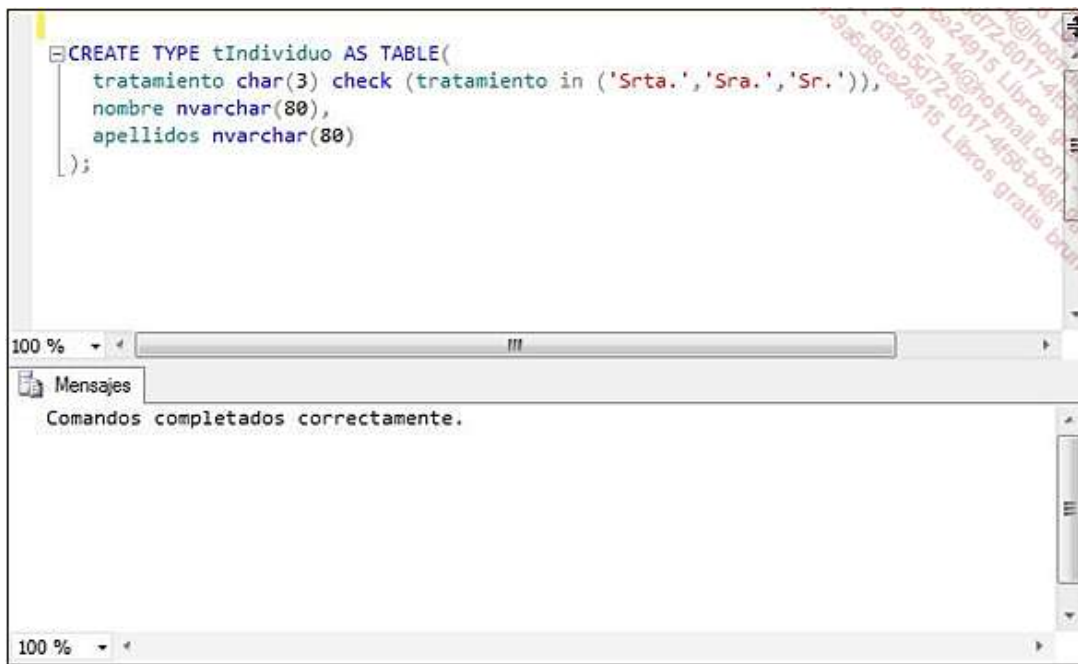
Tipo de datos Transact SQL sobre el que se define la columna. No todas las columnas de un mismo tipo de tabla se definen forzosamente con el mismo tipo, ni con la misma precisión.

`restriccionColumnas`

Definición de la restricción de integridad asociada a la columna.

### Ejemplo

*En el siguiente ejemplo, se define un tipo que representa a un individuo. Este tipo está formado por los campos tratamiento, nombre y apellidos. El campo tratamiento solo puede tomar algunos valores determinados.*



### 3. Administrar las tablas

Una tabla representa una estructura lógica que se utiliza para guardar datos. Para permitir una buena organización de la información, cada tabla está formada por columnas para estructurar los datos. Cada columna se identifica perfectamente por su nombre, que es único dentro de la tabla y por su tipo de datos. Una aplicación usa varias tablas diferentes y los datos se guardan en estas diferentes tablas. Para garantizar la coherencia de los datos, es necesario definir restricciones a nivel de la estructura de las tablas. Estas se llaman restricciones de integridad.

Las tres operaciones de gestión de tabla son: la creación (CREATE TABLE), la modificación (ALTER TABLE) y la eliminación (DROP TABLE). Estas operaciones se pueden hacer en Transact SQL o en SQL Server Management Studio. Estas operaciones sobre las tablas afectan directamente a la estructura de la base de datos. Es necesario tener permisos SQL Server suficientes, como ser miembro del rol de base de datos db\_owner (lo que equivale a ser el propietario de la base de datos) o tener permisos para ejecutar la instrucción CREATE TABLE en la base de datos actual. El administrador del servidor SQL Server puede crear tantas tablas como desee.

#### a. Crear una tabla

La etapa de creación de las tablas es una etapa importante del diseño de la base de datos, porque los datos se organizan en relación a las tablas. Esta operación es puntual y normalmente la realiza el administrador (DBA: *DataBase Administrator*) o la persona encargada de la gestión de la base de datos. La creación de una tabla permite definir las columnas (nombre y tipo de datos) que la componen, así como las restricciones de integridad. Además, es posible definir columnas calculadas, una ordenación específica para la columna y la ubicación de los datos de tipo texto o imagen.

La estructura de la base de datos tendrá un impacto directo sobre el rendimiento general de la aplicación, ya que será posible o no escribir consultas sencillas o indexar algunas columnas para mejorar el rendimiento. Por lo tanto, es importante diseñar el esquema de los datos teniendo en cuenta cómo utilizarlos.

Toda la información relativa a la estructura de las tablas se almacena en las tablas de sistema. Por lo tanto, es posible ir a estas tablas y leer esta información para, por ejemplo, conocer la estructura de una o varias tablas. SQL Server Management Studio consulta estas tablas para ofrecer una interfaz gráfica. Es posible consultar estas tablas de manera indirecta usando el procedimiento **sp\_help**. El uso de este procedimiento evita consultar directamente las tablas y vistas de sistema.

A continuación se muestra una versión simplificada de la instrucción CREATE TABLE.

#### Sintaxis

```
CREATE TABLE [nombreEsquema.] nombreTabla
```

```
( nombreColumna {tipoColumna|AS expresionCalculada}  
[,nombreColumna ... ][,restricciones...])  
[ON grupoArchivo]  
[TEXTIMAGE_ON grupoArchivo]
```

nombreEsquema

Nombre del esquema en el que se va a definir la tabla.

nombreTabla

Puede tener el formato nombreBase.nombreEsquema.nombreTabla.

Usando el nombre completo de la tabla, es decir nombreBase.nombreEsquema.nombreTabla, es posible crear una tabla en una base de datos diferente a aquella en la que se ejecuta el script.

nombreColumna

Nombre de la columna, que debe ser única en la tabla.

tipoColumna

Tipo de sistema o tipo definido por el usuario.

restricciones

Reglas de integridad (se verán más adelante en este libro).

grupoArchivo

Grupo de archivos en el que se va a crear la tabla.

AS expresionCalculada

Es posible definir una regla de cálculo para las columnas que contienen los datos calculados. Estas columnas solo son accesibles en modo solo lectura, y no es posible insertar o actualizar los datos de una determinada columna.

Para las columnas calculadas, es posible usar la palabra clave **PERSISTED** para persistir el resultado del cálculo. Este tipo de opción va a ocupar más espacio en disco y será necesaria una actualización cada vez que los datos básicos del cálculo se actualicen, pero va a permitir acelerar el tratamiento de las consultas de extracción de datos.

TEXTIMAGE\_ON

Permite indicar el grupo de archivos de destino para los datos de tipo texto e imagen.

El número máximo de columnas para una tabla es 1.024. La longitud máxima de un registro es 8.060 bytes (sin contar los datos de tipo texto o imagen).

### Ejemplos

*Creación de la tabla ARTICULOS:*

```

CREATE TABLE ARTICULOS(
    REFERENCIA_ART nvarchar(16),
    NOMBRE_ART nvarchar(200),
    PRECIOUNIT_ART decimal(10,2),
    CODIGO_CAT int
);

```

Mensajes

Comandos completados correctamente.

Ver la información de la tabla con el procedimiento almacenado **sp\_help**:

exec sp\_help ARTICULOS

100 %

Resultados Mensajes

Name	Owner	Type	Created_datetime
1 ARTICULOS	dbo	usertable	2012-09-03 18:48:43.663

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	Trim	TrailingBlanks	FixedLen
1	REFERENCIA_ART	nvarchar	no	32			yes	(n/a)		(n/a)
2	NOMBRE_ART	nvarchar	no	400			yes	(n/a)		(n/a)
3	PRECIOUNIT_ART	decimal	no	9	10	2	yes	(n/a)		(n/a)
4	CODIGO_CAT	int	no	4	10	0	yes	(n/a)		(n/a)

Identity	Seed	Increment	Not For Replication
1 No identity column defined.	NULL	NULL	NULL

RowGuidCol
1 No rowguidcol column defined.

Data_located_on_filegroup
1 PRIMARY

Creación de la tabla **CLIENTES** (desde la interfaz gráfica):

ANGEL\_MARIA1\SQL...OM - dbo.Table\_1\* 283a.sql - ANGEL\_M...dministrador (53))\*

Nombre de columna	Tipo de datos	Permitir val...
numero	int	<input type="checkbox"/>
nombre	nvarchar(30)	<input type="checkbox"/>
apellido	nvarchar(30)	<input type="checkbox"/>
direccion	nvarchar(30)	<input checked="" type="checkbox"/>
codigo_postal	char(5)	<input checked="" type="checkbox"/>
ciudad	nvarchar(30)	<input checked="" type="checkbox"/>
telefono	char(14)	<input checked="" type="checkbox"/>

Propiedades de columna

(General)

(Nombre) numero

Permitir valores NULL No

Tipo de datos int

Valor o enlace predeterminado

Diseñador de tablas

(General)

Propiedades

[Tab] dbo.Clientes

(Nombre) Clientes

Descripción

Esquema dbo

Nombre de la base GESCO.M

Nombre del servidor angel.maria1\sqlserver2

Diseñador de tablas

Columna de GUID

Columna de iden

Especificación de PRIMARY

Extensión de bloc Tabla

Grupo de archivo PRIMARY

Indicizable Si

Replicada No

(Nombre)

## b. Modificar una tabla

La modificación de una tabla se hace con el comando ALTER TABLE o utilizando la interfaz gráfica de SQL Server Management Studio. Cuando se modifica una tabla, es posible añadir y eliminar columnas y restricciones, modificar la definición de una columna (tipo de datos, clasificación y comportamiento respecto al valor NULL), activar o desactivar las restricciones de integridad y los triggers. Para mantener un tiempo de tratamiento coherente en la importación masiva de datos en la base de datos, este último punto puede resultar útil.

A continuación se presenta una sintaxis resumida de las principales opciones de la instrucción ALTER TABLE.

### Sintaxis

```
ALTER TABLE [nombreEsquema.] nombreTabla
{ [ ALTER COLUMN nombreColumna
  { nuevoTipoDatos [ ( longitud [ , precision ] ) ]
  [ COLLATE clasificacion ] [ NULL | NOT NULL ] } ]
| ADD nuevaColumna
| [ WITH CHECK | WITH NOCHECK ] ADD restriccionTabla
| DROP { [ CONSTRAINT ] nombreRestriccion | COLUMN nombreColumna }
| { CHECK | NOCHECK } CONSTRAINT { ALL | nombreRestriccion }
| { ENABLE | DISABLE } TRIGGER { ALL | nombreTrigger } }
```

nombreEsquema

Nombre del esquema en el que se va a definir la tabla.

WITH NOCHECK

Permite establecer una restricción de integridad sobre la tabla sin que esta restricción se verifique para los registros ya existentes en la tabla.

COLLATE

Permite definir una clasificación para la columna, diferente de la clasificación de la base de datos.

NULL, NOT NULL

Permiten definir una restricción de nulidad o no nulidad sobre una columna existente de la tabla.

CHECK, NOCHECK

Permiten activar y desactivar las restricciones de integridad.

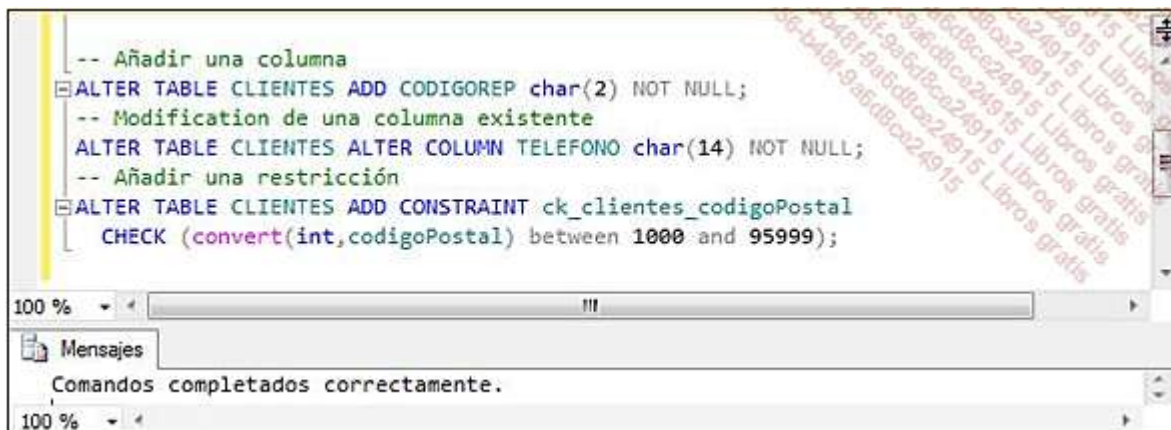
ENABLE, DISABLE

Permiten activar y desactivar la ejecución de los triggers asociados a la tabla.

### Ejemplo

*Añadir una columna:*





```
-- Añadir una columna
ALTER TABLE CLIENTES ADD CODIGOREP char(2) NOT NULL;
-- Modificación de una columna existente
ALTER TABLE CLIENTES ALTER COLUMN TELEFONO char(14) NOT NULL;
-- Añadir una restricción
ALTER TABLE CLIENTES ADD CONSTRAINT ck_clientes_codigoPostal
CHECK (convert(int,codigoPostal) between 1000 and 95999);
```

Mensajes

Comandos completados correctamente.

### c. Eliminar una tabla

La eliminación de una tabla implica la eliminación de todos los datos presentes en ella. Los triggers e índices asociados a la tabla también se eliminan, así como los permisos de uso de la tabla. Por el contrario, las vistas, procedimientos y funciones que hacen referencia a la tabla, no se eliminan. Si hacen referencia a la tabla eliminada, se produce un error durante la ejecución.

No es posible eliminar una tabla que esté siendo referenciada en una restricción de clave extranjera, por otra tabla.

### Sintaxis

```
DROP TABLE [nombreEsquema.] nombreTabla [,nombreTabla...]
```

La eliminación de una tabla eliminará los datos e índices asociados.

*Eliminar una tabla:*



```
DROP TABLE LINEAS_PDO;
```

Mensajes

Comandos completados correctamente.

### d. Nombre completo de una tabla

En función de la ubicación desde donde se hace referencia a la tabla y, de manera más general, al objeto al que se hace referencia, es necesario usar un nombre más o menos concreto. El nombre completo de una tabla, y por tanto de un objeto, es:

```
nombreBase.nombreEsquema.nombreObjeto
```

Sin embargo, como normalmente los objetos a los que se hace referencia están presentes en la base de datos actual, es posible omitir el nombre de la base de datos.

El nombre del esquema también se puede omitir. En este caso, el motor de base de datos buscará el objeto en el esquema asociado al usuario y, en caso de no encontrarlo, en el esquema dbo.



## e. Columnas calculadas

SQL Server permite crear columnas calculadas en las tablas. El hecho de poder definir columnas como estas, permite encontrar la información directamente en la tabla sin tener que rehacer el cálculo de manera sistemática. Este tipo de columna solo está justificado cuando el cálculo se haga de manera regular o si los datos se deben filtrar u ordenar. Efectivamente, estas columnas pueden estar indexadas, pero para esto la columna se tiene que crear de tipo PERSISTED, es decir, que el resultado del cálculo se almacene físicamente en la tabla, lo que no es el caso por defecto.

Para poder definir los datos de la base de datos de cálculo, deben estar presentes directamente en el registro de datos. No es posible ir a buscarlos a otra tabla.

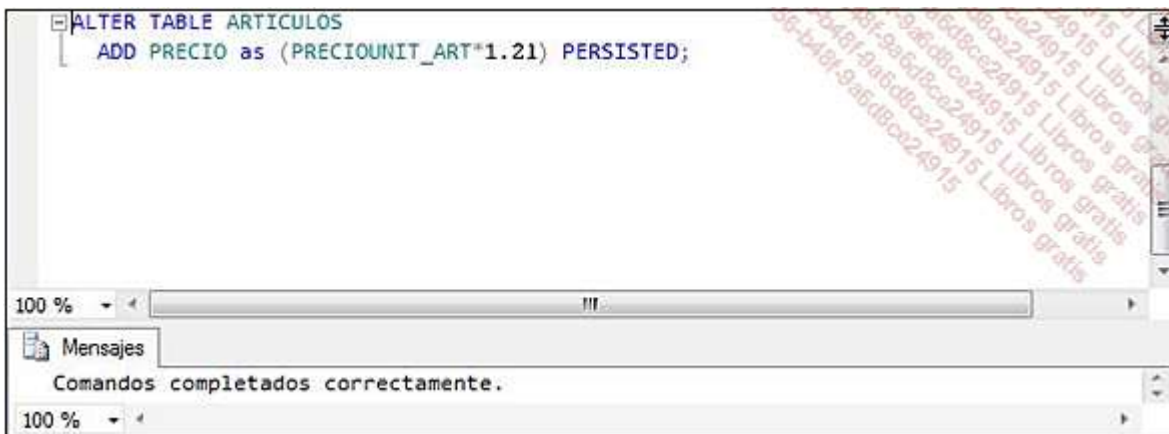
Este tipo de columna se puede definir durante la creación de la tabla o cuando se modifica la tabla con la instrucción ALTER TABLE.

### Sintaxis

```
ALTER TABLE nombreTabla  
    ADD nombreColumna AS (calcula) [PERSISTED];
```

### Ejemplo

En el siguiente ejemplo se añade la columna precio a la tabla de artículos. Para este ejemplo, el IVA queda fijado en el 21%.



## Implementación de la integridad de los datos

Para asegurar la coherencia de los datos en la base de datos, es posible administrar a nivel del servidor, un conjunto de funcionalidades que permiten centralizar los controles y las reglas de funcionamiento que se especifican en el análisis.

La implementación de la integridad de los datos se puede hacer de manera procedimental usando triggers (TRIGGER) o declarativa, usando restricciones (CONSTRAINT).

Cuando se construye la tabla, también es posible definir valores por defecto (DEFAULT) o especificar un contador asociado a una columna (IDENTITY). Estas 2 opciones no permiten garantizar la integridad de los datos, pero contribuyen de manera significativa a esta integridad.

La integridad de los datos traduce las reglas del modelo relacional, reglas de coherencia (integridad de dominio), existencia de valores nulos, regla de unicidad (integridad de entidad) y claves extranjeras (integridad referencial).

En la medida de lo posible, es preferible implementar la integridad usando restricciones, ya que la restricción forma parte integrante de la estructura de la tabla. El respeto de la restricción es efectivo para todos los registros y la comprobación es mucho más rápida.

## 1. Los valores por defecto

Desde SQL Server 2005, ya no se usan los objetos DEFAULT y no se deben aplicar en los nuevos desarrollos. Efectivamente, este tipo de objeto no es conforme a la norma de SQL.

Incluso si las instrucciones CREATE DEFAULT, DROP DEFAULT y los procedimientos almacenados **sp\_bindefault** y **sp\_unbindefault** siguen estando presentes, es preferible definir el valor por defecto durante la creación de la tabla (CREATE TABLE) o usar una instrucción de modificación de tabla (ALTER TABLE). Como siempre, estas operaciones se pueden ejecutar en un script o con SQL Server Management Studio.

La manera en la que se debe definir un valor por defecto en SQL Server 2012, se presenta en la sección DEFAULT de este capítulo.

## 2. Las reglas

Para ofrecer una gestión más uniforme de los diferentes elementos de la base de datos, generalizando el uso de las instrucciones CREATE, ALTER y DROP, y para estar más cerca de cumplir la norma, SQL Server 2012 ya no ofrece la gestión de las reglas como objetos independientes. Las restricciones de integridad, que se pueden expresar como reglas, se deben definir durante la creación de la tabla usando la instrucción CREATE TABLE. También se pueden añadir/eliminar de una tabla existente con la instrucción ALTER TABLE.

Para asegurar la continuidad de los scripts, SQL Server sigue interpretando correctamente las instrucciones CREATE RULE, DROP RULE, **sp\_bindrule** y **sp\_unbindrule**.

## 3. La propiedad Identity

Esta propiedad se puede asignar a una columna numérica de tipo entero, durante la creación o modificación de la tabla y permite que el sistema administre los valores para esta columna. Los valores se gestionarán en la creación del registro, partiendo sucesivamente del valor inicial especificado (por defecto 1) y aumentando o disminuyendo registro a registro un incremento (por defecto 1).

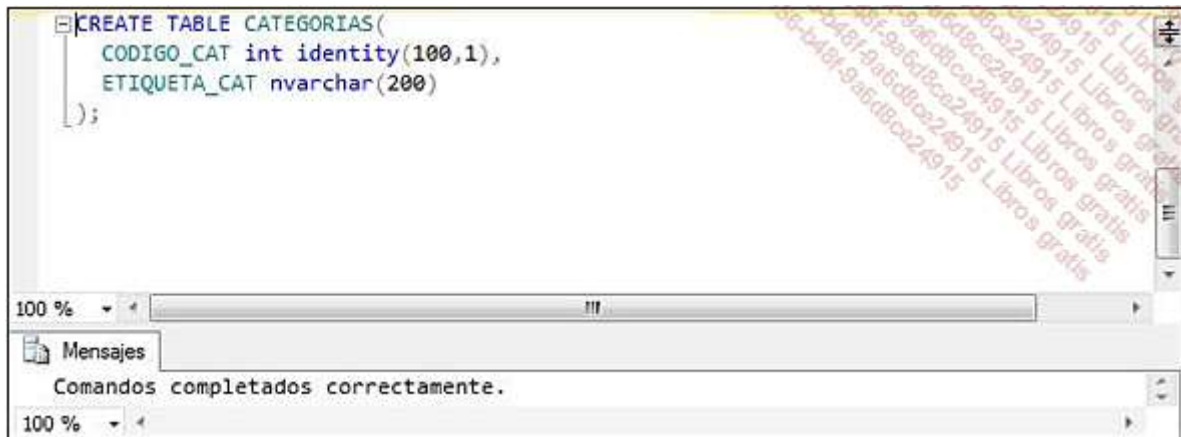
### Sintaxis

```
CREATE TABLE nombre (columna tipoEntero IDENTITY [(inicio, incremento)],  
...)
```

Solo puede haber una columna IDENTITY por cada tabla.

La propiedad IDENTITY se debe definir al mismo tiempo que la columna con la que se relaciona. La definición de una columna identity puede intervenir en un comando CREATE TABLE o ALTER TABLE.

### Ejemplo



Durante la creación de un registro (INSERT), no se especifica ningún valor para CODIGO\_CAT. La primera inserción asignará el CODIGO\_CAT 100, la segunda el CODIGO\_CAT 101, etc.

La palabra clave IDENTITYCOL se podrá usar en una cláusula WHERE, en lugar del nombre de la columna.

La variable global @@IDENTITY almacena el último valor asignado por una identidad durante la sesión en curso. La función SCOPE\_IDENTITY permite hacer lo mismo, pero limitando el alcance de la visibilidad al lote de instrucciones actual. La función IDENT\_CURRENT permite conocer el último valor de identidad generado para la tabla que se especifica como argumento, independientemente de las sesiones.

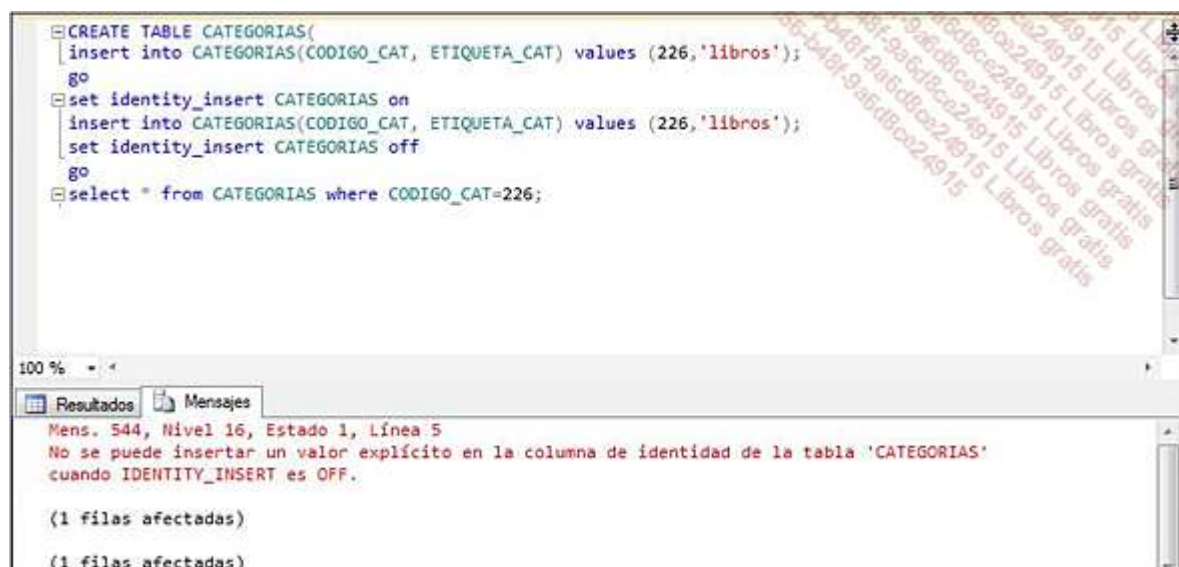
Para poder insertar datos sin usar la propiedad IDENTITY y, por tanto, la numeración automática, hay que llamar a la instrucción IDENTITY\_INSERT de la siguiente manera:

```
SET IDENTITY_INSERT nombreTabla ON
```

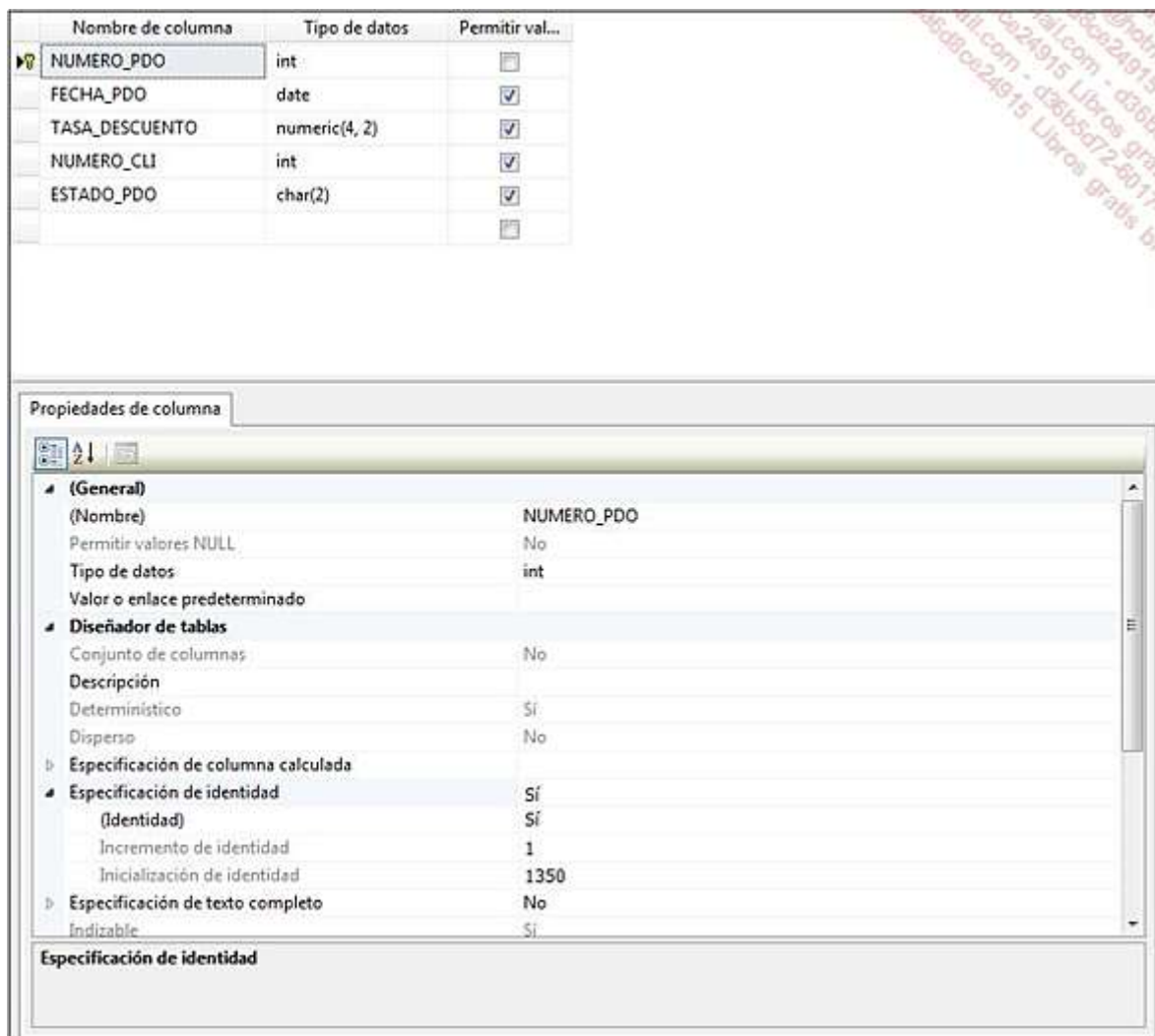
El argumento ON permite desactivar el uso de la propiedad IDENTITY, mientras que la misma instrucción con el argumento OFF, vuelve a activar la propiedad.

### Ejemplo

En el siguiente ejemplo, se añade una categoría nueva. La primera inserción termina con error, porque la propiedad IDENTITY está activada. Después de desactivarla con la instrucción SET IDENTITY\_INSERT categorías ON, es posible insertar el registro de datos.



Es posible definir la propiedad identity desde SQL Server Management Studio, en la pantalla de modificación o creación de una tabla (**Creación**, desde el menú contextual asociado a la tabla).



Es posible usar las siguientes funciones para obtener más información sobre los tipos de identidad:

- IDENT\_INCR para saber el paso de incremento del valor identity.
- IDENT\_SEED para saber el valor de inicio fijado durante la creación del tipo identity.

El objetivo de todas estas funciones es permitir al programador controlar mejor el valor generado, para poder recuperarlo cuando sea la clave primaria.

## 4. Las restricciones de integridad

Las restricciones permiten implementar la integridad declarativa, definiendo los controles de valor a nivel de la estructura de la tabla en sí misma.

La definición de las restricciones se hace con un script, usando las instrucciones CREATE y ALTER TABLE. También es posible definir las desde SQL Server Management Studio.

Cuando es posible, se recomienda usar las restricciones de integridad en lugar de triggers de base de datos, ya que las restricciones de integridad están normalizadas y limitan la codificación, por lo tanto, el riesgo de error. Están integradas en la definición de la estructura de la tabla y su verificación es más rápida que la ejecución de un trigger.

### Sintaxis

```
ALTER TABLE nombreTabla
```

```
{ADD|DROP} CONSTRAINT nombreRestriccion ...[;]
```

Es posible obtener la información de las restricciones de integridad definidas, consultando **sys.key\_constraints** o usando los procedimientos almacenados, **sp\_help** y **sp\_helpconstraint**.

Las restricciones se asociarán a una tabla o a una columna de la tabla, en función del caso.

Es posible verificar la integridad de las restricciones con DBCC CHECKCONSTRAINT. Sobre todo, esta herramienta tiene sentido cuando se desactivan las restricciones de una tabla y después se vuelven a activar sin verificar los datos de la tabla.

## a. NOT NULL

SQL Server considera la restricción de nulidad como una propiedad de columna. La sintaxis es la siguiente:

```
CREATE TABLE nombreTabla (nombreColumna tipo
[{NULL | NOT NULL}] [?...])
```

NOT NULL

Indica que la columna se debe validar durante la creación o modificación.

Es preferible indicar sistemáticamente NULL o NOT NULL, porque los valores por defecto de esta propiedad dependen de muchos factores:

- Para un tipo de datos definido por el usuario, es el valor que se especifica cuando se crea el tipo.
- Los tipos bit y timestamp solo aceptan NOT NULL.
- Los parámetros de sesión ANSI\_NULL\_DFLT\_ON o ANSI\_NULL\_DFLT\_OFF se pueden activar con el comando SET.
- Se puede activar el argumento de base de datos ANSI NULL.

Desde SQL Server 2005, es posible modificar la restricción de NULL/NOT NULL con el comando ALTER TABLE, para una columna que ya existe. Por supuesto, los datos ya presentes en la tabla deben respetar estas restricciones.

## b. PRIMARY KEY

Esta restricción permite definir un identificador como clave primaria, decir una o varias columnas que solo aceptan valores únicos en la tabla (regla de unicidad o restricción de integridad).

### Sintaxis

```
[CONSTRAINT nombreRestriccion] PRIMARY KEY [CLUSTERED | NONCLUSTERED]
(nombreColumna[, ...]) [WITH FILLFACTOR=x] [ON
grupoArchivos]
```

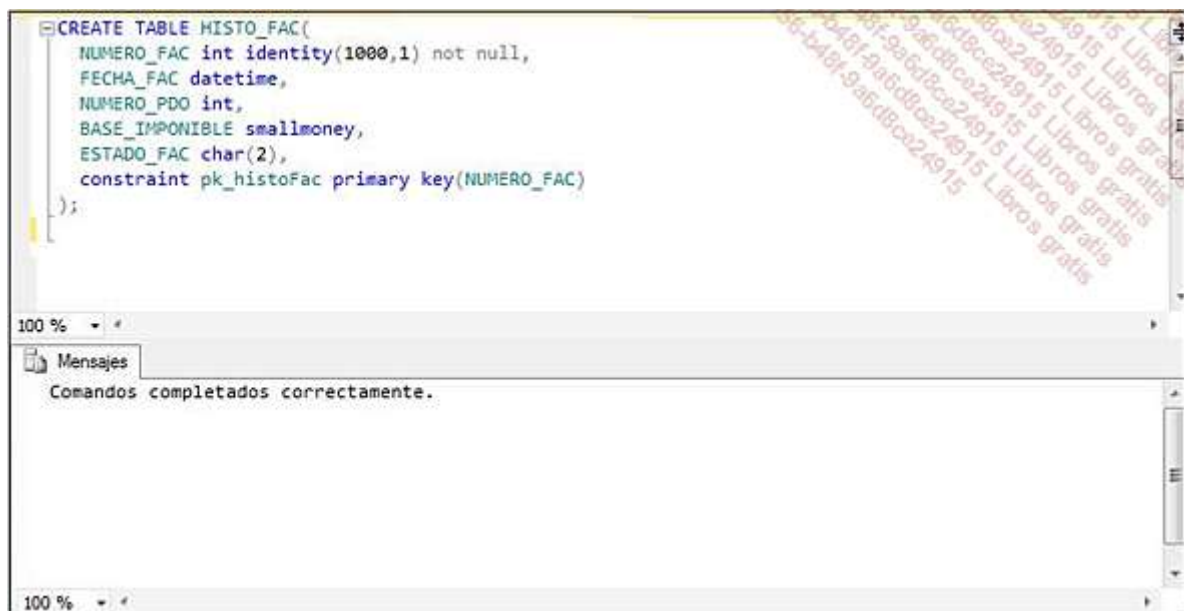
nombreRestriccion

nombreRestriccion permite identificar la restricción en las tablas de sistema. Por defecto, SQL Server dará un nombre poco manejable.

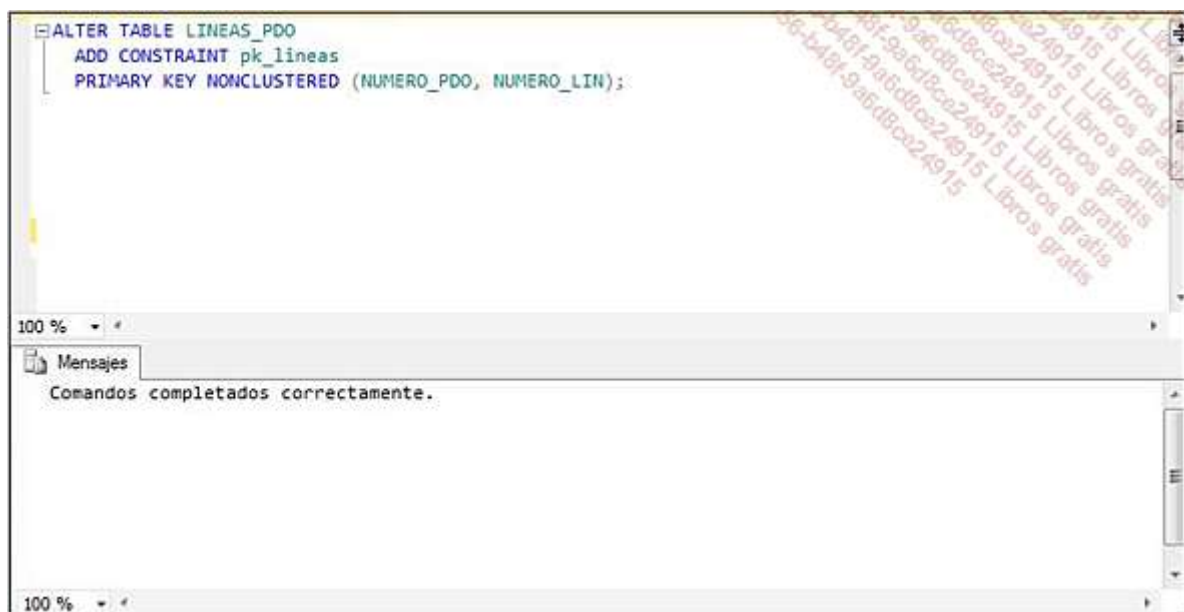
Esta restricción va a crear automáticamente un índice único, ordenado por defecto, con el nombre de la restricción y las opciones NONCLUSTERED y FILLFACTOR. Una clave primaria puede contener hasta 16 columnas. Solo puede haber una clave primaria por cada tabla. Las columnas que la componen deben ser NOT NULL.

### Ejemplos

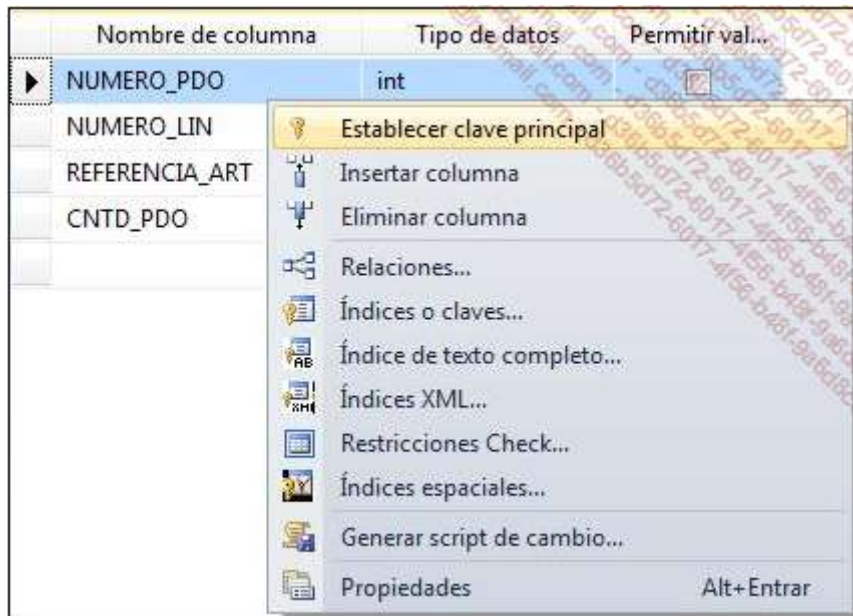
Se define la tabla HISTO\_FAC y se establece la clave primaria pk\_histoFac.



Se añade la clave primaria a la tabla LINEAS\_PDO (se definirá un índice ordenado sobre la columna numero\_pdo):



Gestión de la clave primaria con SQL Server Management Studio:



No es posible eliminar la clave primaria si:

- Está siendo referenciada por una clave extranjera.
- Se define un índice XML principal en la tabla.

### c. UNIQUE

Esta restricción permite traducir la regla de unicidad para el resto de claves únicas de la tabla o identificadores clave secundarios.

Esta restricción tiene las mismas características que PRIMARY KEY, salvo dos excepciones:

- Es posible tener varias restricciones UNIQUE para cada tabla.
- Las columnas que se usan pueden ser NULL (aunque no es recomendable).

Durante la adición de una restricción de unicidad en una tabla existente, SQL Server se asegura de respetar esta restricción en los registros existentes, antes de validar la restricción.

La gestión de esta restricción se asegura mediante un índice de tipo UNIQUE. No es posible eliminar este índice con el comando DROP INDEX. Hay que eliminar la restricción con la instrucción ALTER TABLE.

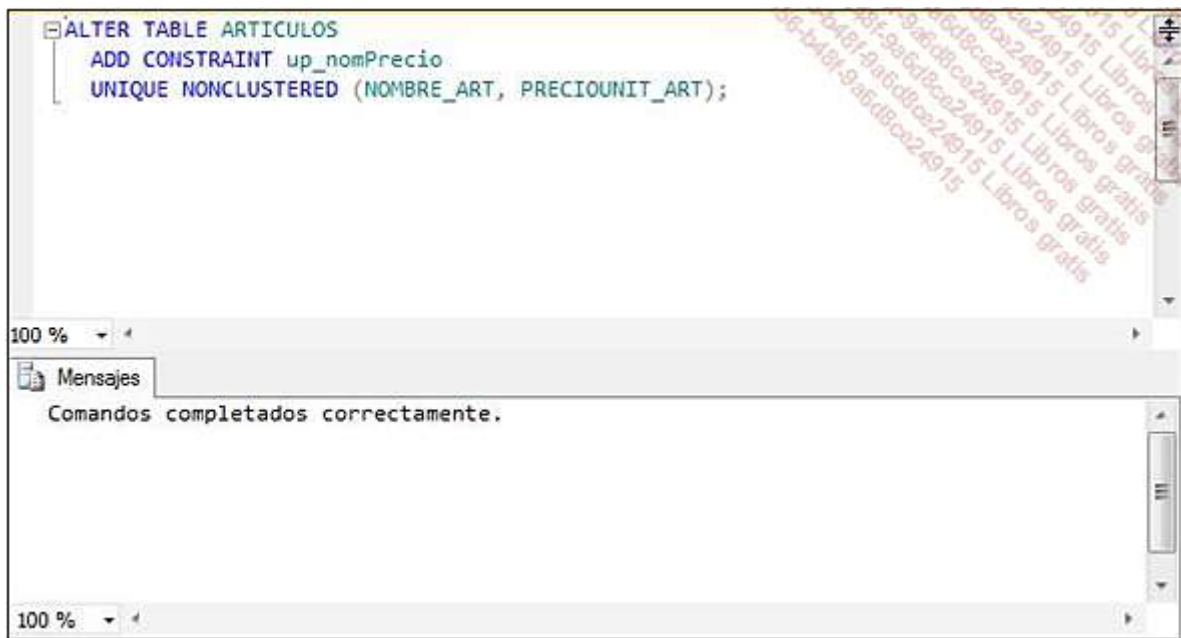
### Sintaxis

```
[CONSTRAINT nombreRestriccion] UNIQUE [CLUSTERED | NONCLUSTERED ]
(nombreColumna [,...]) [WITH FILLFACTOR=x] [ON grupoArchivos]
```

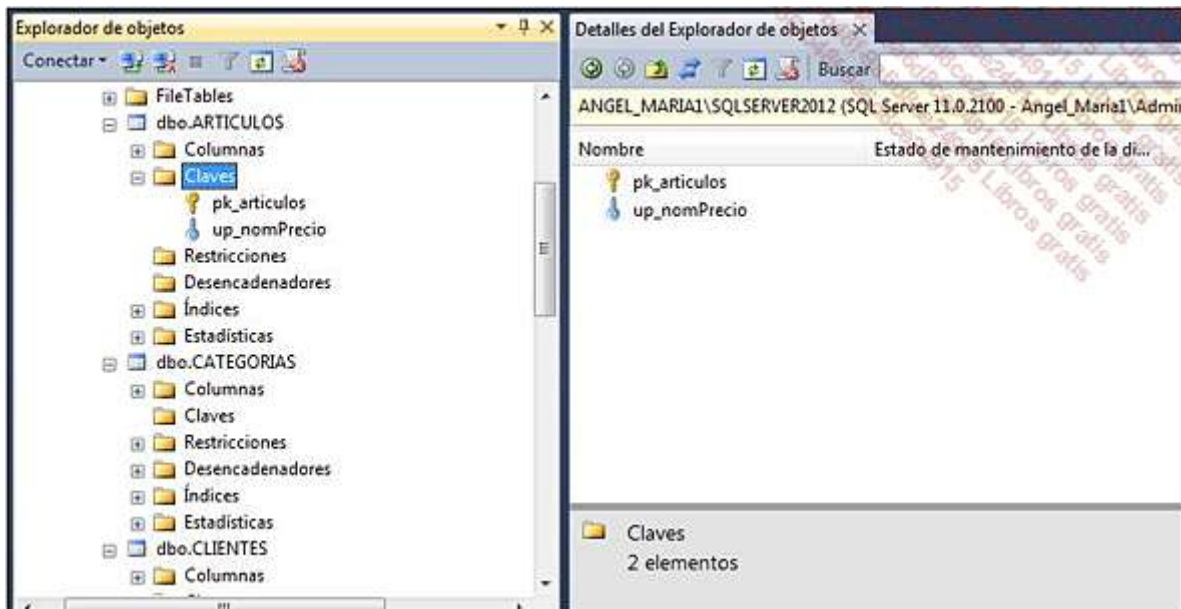
### Ejemplo

*La asociación de las columnas nombre\_art y preciounit\_art debe ser única en la tabla ARTICULOS:*





Gestión de las claves secundarias con SQL Server Management Studio:



#### d. REFERENCIAS

Esta restricción traduce la integridad referencial entre una clave extranjera de una tabla y una clave primaria o secundaria de otra tabla.

#### Sintaxis

```
CONSTRAINT nombreRestriccion
[FOREIGN KEY (columna[,...])]
REFERENCIAS tabla [ ( columna [,... ] ) ]
[ ON DELETE { CASCADE | NO ACTION | SET NULL | SET DEFAULT } ]
[ ON UPDATE { CASCADE | NO ACTION | SET NULL | SET DEFAULT } ]
```

La cláusula FOREIGN KEY es obligatoria cuando usamos una sintaxis de restricción de tabla para añadir la restricción.



La opción cascade permite indicar el comportamiento que debe adoptar SQL Server cuando el usuario actualiza o intenta eliminar una columna que está siendo referenciada. Durante la definición de una restricción referencial con las instrucciones CREATE TABLE o ALTER TABLE, es posible indicar las cláusulas ON DELETE y ON UPDATE.

NO ACTION

Valor por defecto de estas opciones. Permite obtener un comportamiento idéntico al de las versiones anteriores de SQL Server.

ON DELETE CASCADE

Permite indicar que, en caso de eliminación de un registro cuya clave primaria está siendo referenciada por uno o varios registros, todos los registros que contienen la clave extranjera que hacen referencia a la clave primaria, también se eliminan. Por ejemplo, usando esta opción, si se elimina un registro de la tabla PEDIDOS, se eliminan todos los registros de la tabla LINEAS\_PDO.

ON UPDATE CASCADE

Permite pedir a SQL Server que actualice los valores contenidos en las columnas de las claves extranjeras, cuando se actualiza el valor de clave primaria referenciada.

SET NULL

Cuando el registro correspondiente a la clave primaria en la tabla referenciada se elimina, la clave extranjera toma valor nulo.

SET DEFAULT

Cuando se elimina el registro correspondiente a la clave primaria de la tabla referenciada, la clave extranjera toma el valor por defecto que se define a nivel de la columna.

No es posible una acción en cascada sobre las tablas que tienen un trigger instead of.

Se prohíbe cualquier referencia circular en los triggers que se ejecutan en modo cascada.

La restricción referencial no crea índices. Se recomienda crearlos a continuación.

Aunque SQL Server no tiene límite máximo respecto al número de restricciones de claves extranjeras definidas para una tabla, Microsoft recomienda no sobrepasar las 253. Este límite se debe respetar en lo referente al número de claves extranjeras definidas en la tabla y el número de claves extranjeras que hacen referencia a la tabla. Más allá de este límite, sería recomendable volver a diseñar la base de datos para obtener un esquema mejor.

### Ejemplo

*Creación de la clave extranjera codigo\_cat en la tabla ARTICULOS:*

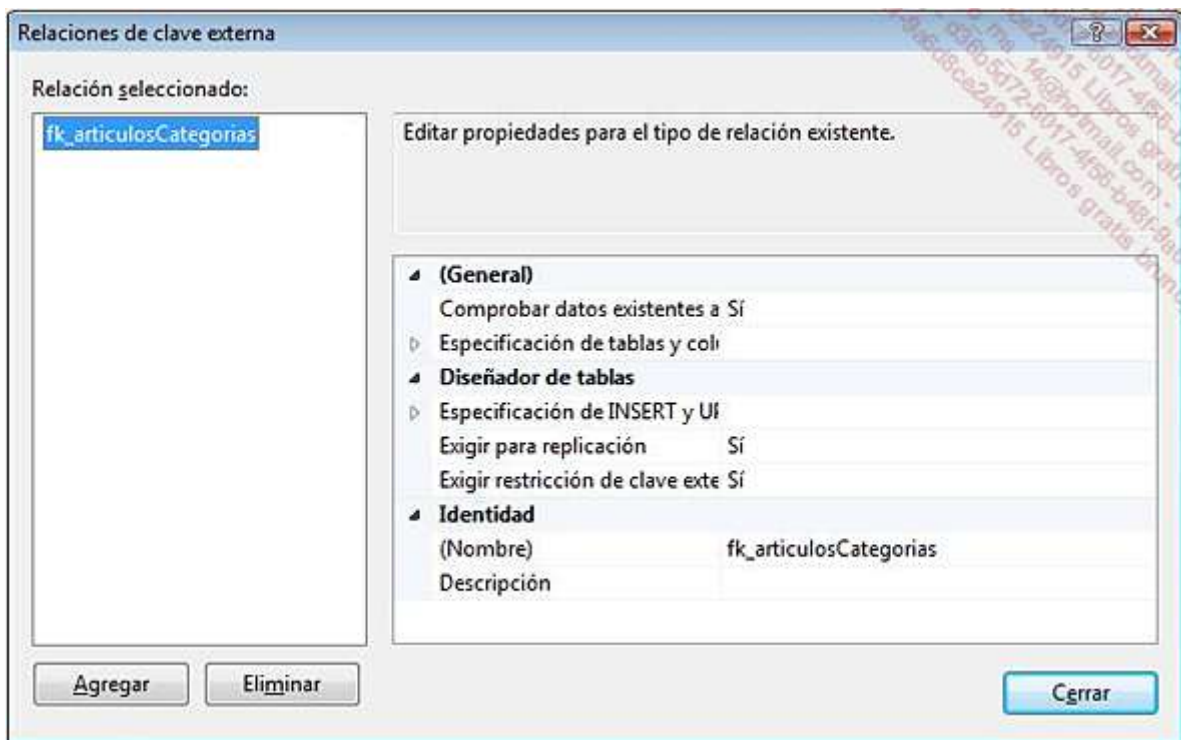


Desde SQL Server Management Studio, es posible crear nuevas claves extranjeras seleccionando **Nueva clave extranjera**, en el menú contextual del nodo **Claves** de la tabla que se muestra en el explorador de objetos.

También es posible ver el detalle de la definición de una clave extranjera existente, haciendo doble clic sobre el nombre de la clave, tal y como aparece en el nodo **Claves**.

### Ejemplo

Ver las propiedades de la clave extranjera `fk_articulosCategorias` desde SQL Server Management Studio.



### e. DEFAULT

El valor por defecto permite indicar el valor que se va a insertar en la columna, en caso de que no se indique ningún dato cuando se inserta el registro.

Los valores por defecto se utilizan mucho cuando la columna no acepta valores NULL, ya que garantizan la existencia de un valor.

Siempre es necesario ser conscientes de que el valor por defecto se usa solo cuando no se especifica ningún valor para la columna en la instrucción INSERT. No es posible completar o eliminar un valor introducido por el usuario. Para hacer este tipo de operaciones, es necesario crear un trigger de base de datos.

Se puede definir un valor por defecto para todas las columnas, excepto para las columnas de tipo timestamp o las que tienen un tipo identity.

## Sintaxis

```
[CONSTRAINT nombreRestriccion] DEFAULT valor
```

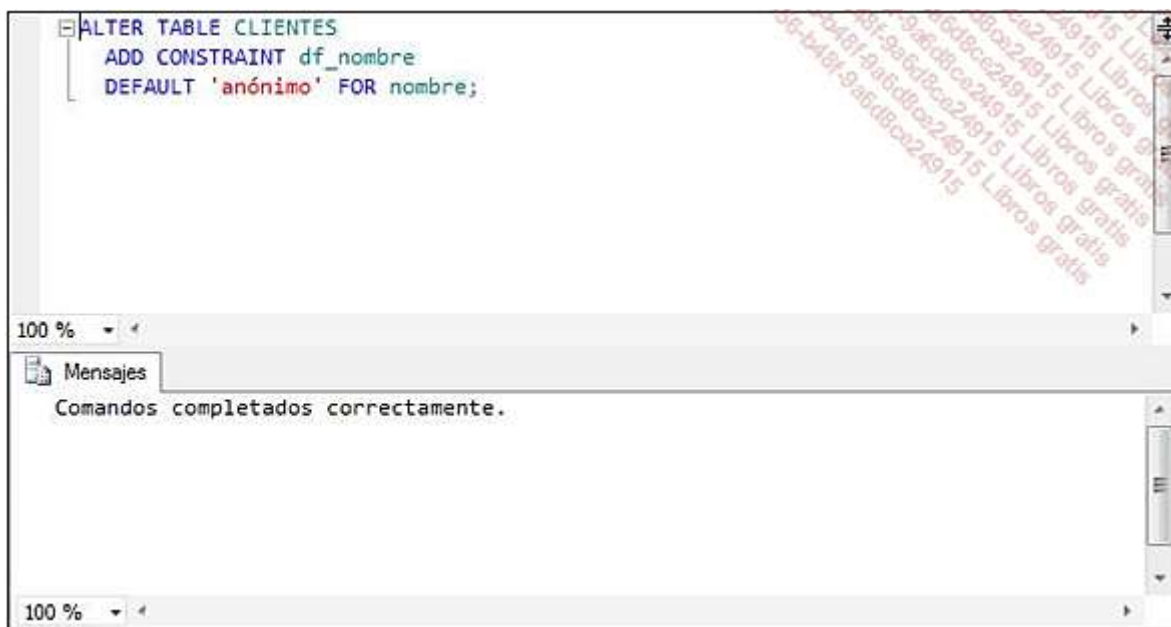
```
[FOR nombreColumna]
```

valor

El valor debe ser exactamente del mismo tipo que el que se usa para definir la columna. Este valor puede ser una constante, una función escalar (como por ejemplo: USER, CURRENT\_USER, SESSION\_USER, SYSTEM\_USER...) o el valor NULL.

### Ejemplo

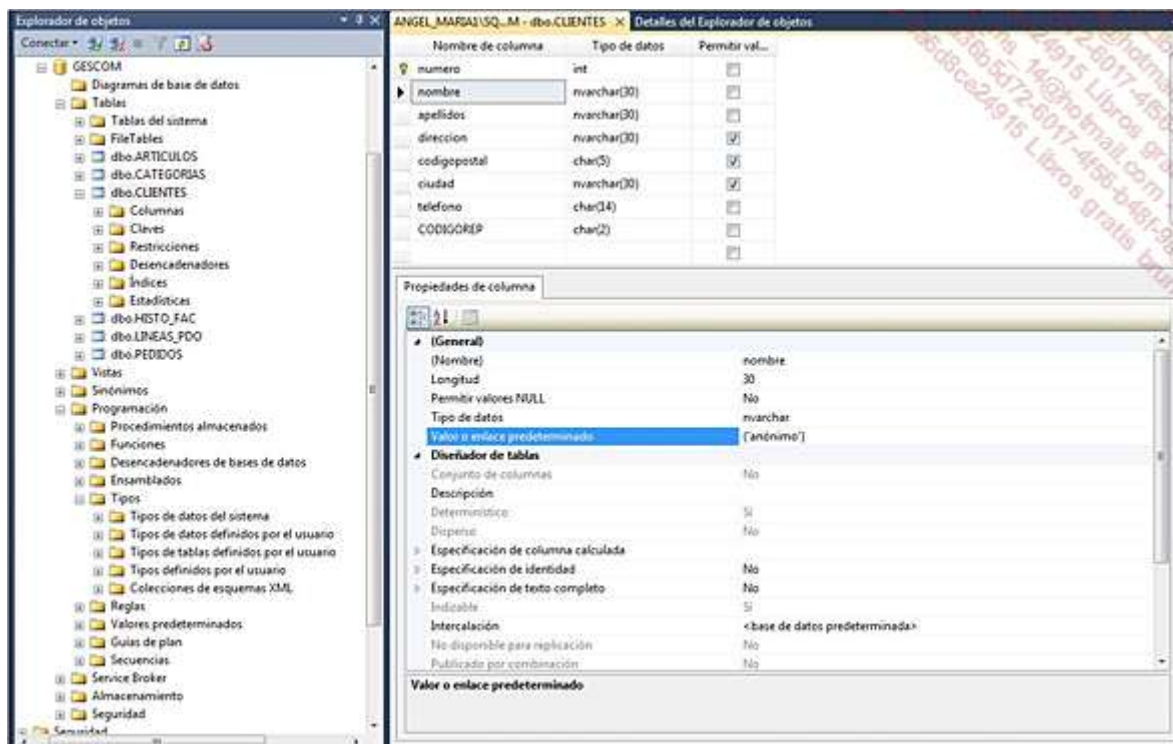
*Valor por defecto para el nombre del cliente:*



Desde SQL Server Management Studio, para establecer o modificar un valor por defecto, es necesario ver la tabla en modo creación, seleccionando **Diseño** desde el menú contextual del nodo que representa a la tabla, en el explorador de objetos.

### Ejemplo

*Definición de un valor por defecto desde SQL Server Management Studio.*



## f. CHECK

La restricción de validación o restricción CHECK permite definir las reglas de validación, relacionando los valores en cuestión de diferentes columnas del mismo registro. Este tipo de restricción también asegura que los datos respetan un formato concreto durante su inserción y actualización en la tabla. Para terminar, con una restricción de validación es posible garantizar que el valor de la columna pertenece a un dominio concreto de valores.

### Sintaxis

```
[CONSTRAINT nombreRestriccion] CHECK [NOT FOR REPLICATION]
```

```
(expresión booleana)
```

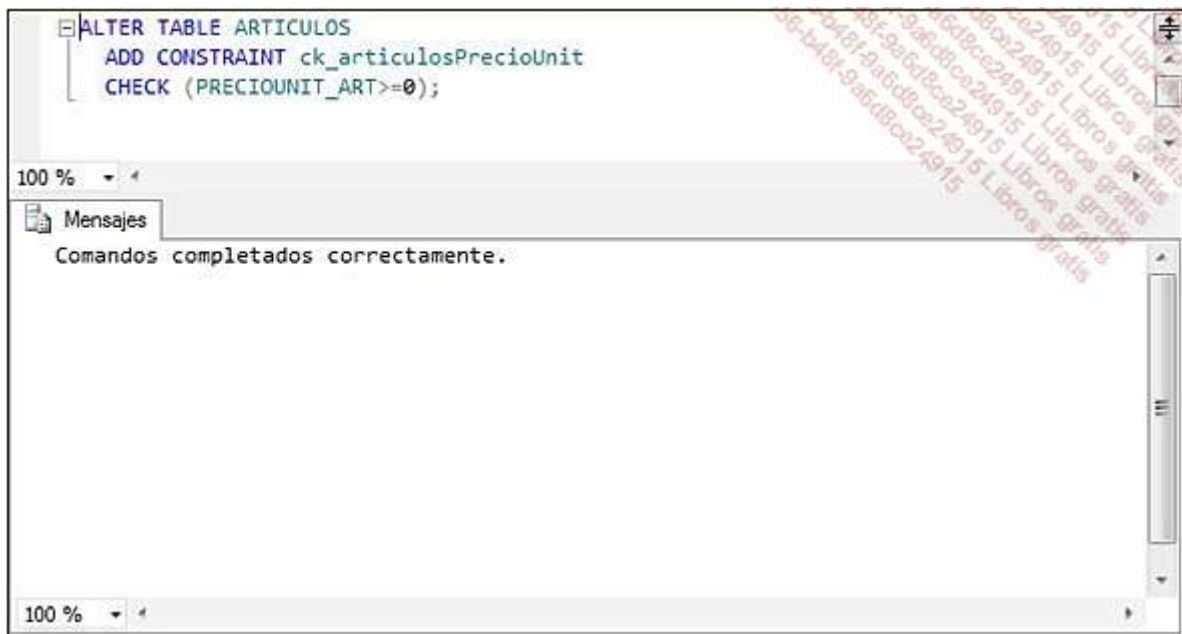
```
NOT FOR REPLICATION
```

Permite impedir la aplicación de la restricción durante la replicación.

La restricción CHECK se asocia automáticamente a la columna que se especifica en la expresión de la condición.

### Ejemplo

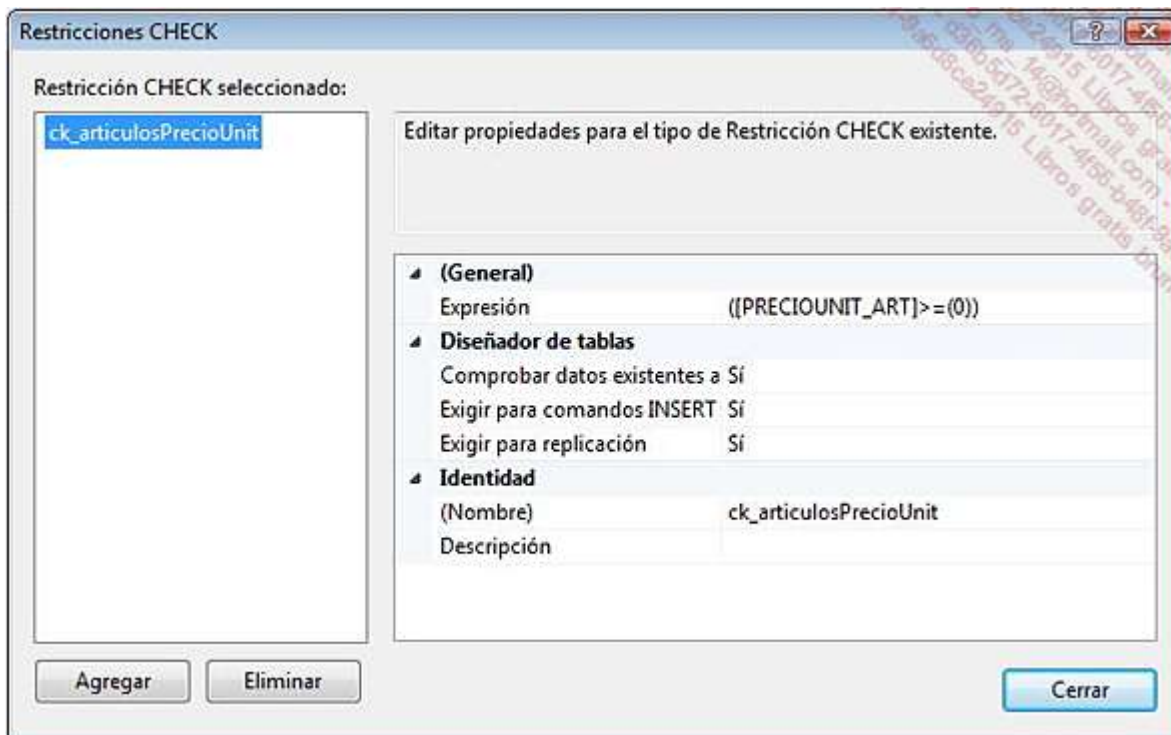
*Implementación del control de precio positivo para un artículo:*



Para aplicar o definir una restricción de validación desde el explorador de objetos de SQL Server Management Studio, hay que desplegar el nodo **Restricciones** asociado a la tabla. Haciendo doble clic en el nombre de la restricción existente, se muestra la ventana de propiedades de la restricción y se pueden modificar. Para definir una nueva restricción de validación, hay que seleccionar **Nueva restricción**, desde el menú contextual del nodo **Restricciones**.

#### Ejemplo

*Gestión de las restricciones CHECK usando SQL Server Management Studio:*



# Administrar los índices

## Uso de los índices o por qué indexar

El objetivo de los índices es permitir un acceso más rápido a la información durante las extracciones (SELECT) o actualizaciones (INSERT, UPDATE, DELETE) de datos, reduciendo el tiempo necesario para localizar un registro. Sin embargo, los índices son costosos cuando se actualiza el valor de una columna indexada.

Una buena estrategia de indexación debe considerar diferentes puntos. Se pueden deducir dos reglas básicas:

- Es mejor tener pocos índices que demasiados. En caso de múltiples índices, el tiempo que se gana en acceder a la información se pierde en actualizarlos.
- Los índices deben ser lo más "largos" posible, para poder usarse en varias consultas.

Para terminar, hay que estar seguros de que las consultas usan los índices que se han definido.

Las consultas también se deben escribir de la manera más explícita posible para manipular el mínimo de información.

Por ejemplo, en caso de una proyección, es preferible listar las columnas que se deben mostrar en lugar de usar el carácter genérico \*.

Para las restricciones, es mejor comparar entre constantes y el valor de una columna.

Por ejemplo, si la tabla ARTICULOS contiene el precio sin IVA de cada artículo, para extraer la lista de los artículos cuyo precio con IVA es inferior o igual a 100 €, es mejor escribir la siguiente condición `preciosinIVA <= 100 / 1.21`, en lugar de `precioconiva * 1.21 <= 100`. Efectivamente, en el segundo caso, el cálculo se hace para cada artículo, mientras que en el primero, el cálculo se hace una vez para todos.

En el ejemplo anterior, todos los artículos usan un IVA del 21%.

También hay que tener en cuenta que los datos se almacenan en los índices y por tanto, van a ocupar un espacio en disco importante. El nivel hoja de un índice ordenado (clustered), contiene el conjunto de los datos. Para un índice no ordenado, el nivel hoja del índice contiene una referencia directa al registro de datos relacionado con la clave del índice.

Los otros niveles del índice se usan para navegar en el índice y llegar, de esta manera, muy rápidamente a los datos.

Es posible añadir la información a nivel de las hojas del índice, sin que estas columnas se tengan en cuenta en el índice. Esta técnica es práctica cuando se define un índice para las consultas. Por ejemplo, es necesario extraer la lista de códigos postales y ciudades de la tabla CLIENTES. Para esto, se define un índice no ordenado respecto a la columna de códigos postales y la columna que representa el nombre de la ciudad, se añade a nivel hoja. De esta manera, el índice cubre la consulta, que es capaz de generar el resultado sin acceder a la tabla.

## ¿Qué es un índice?

La noción de índice ya es conocida por todos. Todos hemos utilizado alguna vez el índice de un libro para acceder directamente a la página o páginas que contienen la información que buscamos. Puede ser que haya utilizado el índice de este libro para acceder directamente a esta explicación, recorriendo el índice buscando la palabra clave "índices".

Hablamos de índice único si junto a cada palabra clave del índice, aparece un único número de página.

Los índices que SQL Server ofrece se asemejan mucho a los índices que podemos encontrar en los libros.

Es posible recorrer el índice para encontrar toda la información, de la misma manera que es posible leer un libro a partir del índice en lugar de seguir el orden propuesto en la tabla de contenidos.

También es posible usar el índice para acceder directamente a una información concreta. Para garantizar tiempos de acceso a la información homogéneos, SQL Server estructura la información de forma arborescente alrededor de la propiedad indexada. De hecho, es el enfoque que adoptamos cuando recorremos un índice tras localizar una primera aparición de la información, en relación al primer carácter. Después, recorremos secuencialmente para localizar la palabra clave que buscamos.

Ahora, imagine un libro en el que es posible definir varios índices, por ejemplo, en relación a las palabras clave, temas, tipo de operaciones que queremos hacer, etc. Esta variedad de índices es lo que ofrece SQL Server, dando la posibilidad de crear varios índices para una tabla.

Entre todos los índices del libro, hay uno en concreto que representa la estructura del libro: se trata de la tabla de contenidos, que se puede ver como un índice de temas. De la misma manera, SQL Server ofrece la estructura física de los datos en relación a un índice. Se trata del índice CLUSTERED.

SQL Server ofrece muchas opciones de indexación, como la posibilidad de indexar datos de tipo XML, geográficos o vistas. A este nivel, solo vamos a ver la parte correspondiente a la indexación de las tablas.

## **¿Organizar o no los datos?**

SQL Server ofrece dos tipos de índice: los índices organizados, de los que solo puede haber uno por tabla, que organizan físicamente la tabla y los índices no organizados.

La creación o eliminación de un índice no organizado no tiene ningún impacto sobre la organización de los datos de la tabla. Por el contrario, la creación o eliminación de un índice organizado tendrá consecuencias sobre la estructura de los índices no organizados.

## **Tabla sin índice organizado**

Si una tabla solo tiene este tipo de índices, la información se almacena gradualmente sin seguir una organización determinada.

Esta es la mejor opción cuando:

- La tabla almacena información que se va a particionar.
- La información se va a truncar.
- Los índices se crean y eliminan frecuentemente.
- Cuando ha tenido lugar una carga de datos en bloque.
- Los índices se crean después de la carga de los datos y su creación puede ser paralela.
- Los datos se modifican raramente (UPDATE) para conservar una estructura sólida.
- El espacio en disco que utiliza el índice se ha reducido, lo que permite definir índices a bajo coste.

Esta solución tiene buen rendimiento para la indexación de una tabla que no está en un servidor OLTP, como por ejemplo un servidor dedicado al análisis de los datos.

## **Los índices organizados**

Solo se puede tener índice de este tipo en cada tabla. Este tipo de índice permite organizar físicamente los datos de la tabla, siguiendo un criterio concreto. Por ejemplo, se puede definir un índice organizado sobre la clave primaria.

Este tipo de índice es costoso para el servidor en términos de tiempo y espacio en disco, durante su construcción o reconstrucción.

Si este tipo de índices se definen en una tabla que ya contenga valores, su construcción será larga. Será incluso más larga cuanto más largos sean los índices no ordenados que existan.



De manera ideal, para evitar un mantenimiento demasiado costoso del índice ordenado, se define sobre una columna que contenga datos estáticos y que ocupe un espacio limitado, como por ejemplo la clave primaria.

La definición de un índice organizado sobre una columna no estable, como el nombre de una persona o su dirección, conduce de manera irremediable a una degradación significativa del rendimiento.

## Los índices no organizados

Dependiendo de si están definidas en una tabla con o sin un índice organizado, las hojas del índice no organizado no hacen referencia de la misma manera al o los registros de datos.

En caso de que la tabla no tenga un índice organizado, el RID (*Row IDentifier*) del registro de datos se almacena a nivel de las hojas del índice. Este RID corresponde a la dirección física (en sentido SQL Server) del registro.

Por el contrario, si la tabla tiene un índice organizado, la clave del registro de datos que se busca se almacena a nivel de la hoja del índice no organizado. Esta clave corresponde al criterio de búsqueda del índice organizado.

Los índices no organizados son la mejor opción para definir un índice que cubra una o varias consultas. Con este tipo de índices, la consulta encuentra en el índice todos los datos que necesita y evita tener que acceder inútilmente a la tabla, porque solo se manipula el índice. El rendimiento es considerablemente mejor, ya que el volumen de datos manipulado es mucho menor.

Cada consulta se puede optimizar con esta técnica, pero este no es el objetivo que se persigue, ya que es costoso mantener tanta diversidad de índices.

## Índice de recubrimiento

SQL Server ofrece índices llamados de recubrimiento, en el sentido de que contienen todos los datos necesarios para una o varias consultas y evitan, de esta manera, un acceso completo a la tabla. Para estos índices solo se indexa una o varias columnas, por ejemplo aquella(s) sobre la(s) que se definen los criterios de ordenación. Por el contrario, a nivel de las hojas de este índice y solo a este nivel, se añade una copia de los valores contenidos en una o varias columnas de la tabla. El rendimiento se puede mejorar de manera considerable, solo recorriendo el índice sin acceder a la tabla.

## Índice filtrado

Un índice filtrado es necesariamente un índice no organizado y que solo cubre una parte de la tabla. Durante la definición de estos índices, se añade una cláusula WHERE para limitar el número de registros que participan en el índice. Es importante que los datos que participan en esta cláusula de restricción sean estables, ya que un cambio de valor puede implicar la inclusión del registro en el índice o bien su exclusión.

## Índice y columnas calculadas

SQL Server ofrece la posibilidad de definir columnas calculadas a nivel de tabla. Si esta columna se define con la propiedad PERSISTED, es posible incluirla en un índice, como cualquier otra columna de la tabla.

## Índice y cálculo agregado

En las tablas voluminosas se tienen que implementar índices para asegurar que las consultas actuales no provocan un recorrido completo de tabla, sino que se basan en los índices. Este punto es particularmente importante para los cálculos agregados, que necesitan una operación de ordenación antes de poder realizar el cálculo. Si un índice permite limitar la ordenación que hay que hacer, la ganancia de rendimiento es mayor.

Siempre con el objetivo de optimizar el rendimiento al acceder a los datos, es posible definir el índice sobre las columnas de una vista, incluso si el resultado presente en la columna es el resultado de un cálculo agregado.

El índice que se define sobre una vista está limitado a 16 columnas y 900 bytes de datos, para cada entrada del índice.

Al contrario que con la inclusión de columnas no indexadas en las hojas del índice, los índices definidos sobre una vista pueden contener el resultado de un cálculo agregado.



## 1. Crear un índice

Un índice se puede crear en cualquier momento, haya o no datos en la tabla.

Sin embargo, si se tienen que importar los datos, es mejor importarlos primero y después definir los índices. En caso contrario (los índices se definen antes de una importación importante de datos), es necesario reconstruir los índices para garantizar un reparto equilibrado de los datos en el índice.

Las principales opciones y argumentos del comando de creación del índice son las siguientes:

### Sintaxis

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX nombreIndice
ON { nombreTabla | nombreVista } ( columna [ ASC | DESC ] [ ,...n ] )
[ INCLUDE (columna[ ,...n ] ) ]
[ WHERE restriccion ]
[ WITH [ PAD_INDEX = {ON | OFF} ]
      [, FILLFACTOR = x]
      [, IGNORE_DUP_KEY = {ON | OFF} ]
      [, DROP_EXISTING = {ON | OFF} ]
      [, ONLINE = {ON | OFF} ]
      [, STATISTICS_NORECOMPUTE = {ON | OFF} ]
      [, SORT_IN_TEMPDB = {ON | OFF} ]
[ DATA_COMPRESSION =
  { NONE | ROW | PAGE } ]
[ ON grupoArchivo ]

UNIQUE
```

Indica que la(s) columna(s) indexada(s), no puede(n) tener el mismo valor en varios registros de la tabla.

CLUSTERED o NONCLUSTERED

Con un índice CLUSTERED (ordenado), el orden físico de los registros de las páginas de datos es idéntico al orden del índice. Solo se puede tener uno por tabla y se tiene que crear en primer lugar. El valor por defecto es NONCLUSTERED.

INCLUDE

Con esta opción es posible duplicar la información para incluir directamente en el índice, una copia de las columnas que se especifican como argumento. Esta posibilidad está limitada a los índices no organizados y los datos se almacenan a nivel hoja. Es posible incluir en el índice de 1 a 1.026 columnas de cualquier tipo, excepto varchar(max), varbinary(max) y nvarchar(max). Esta opción INCLUDE permite definir índices que cubran la consulta, es decir, que la consulta va a recorrer solo el índice para encontrar todo lo que necesita.

WHERE

Esta opción permite limitar el número de registros implicados en el índice, definiendo una restricción. Si se usa esta opción, el índice se llama índice filtrado. Un índice como este es obligatoriamente no organizado (NONCLUSTERED).

FILLFACTOR=x

Indica el porcentaje de ocupación de las páginas del índice, a nivel hoja. Esto permite mejorar el rendimiento, evitando tener valores de índice consecutivos que nunca estarán físicamente contiguos. El valor por defecto es 0 o se fija con **sp\_configure**. Para x = 0, el nivel hoja se ocupa al 100% y se reserva espacio a nivel no hoja.

Para x entre 1 y 99, el porcentaje de ocupación a nivel hoja es del x% y se reserva espacio a nivel no hoja. Para x = 100, se ocupan los niveles hoja y no hoja.

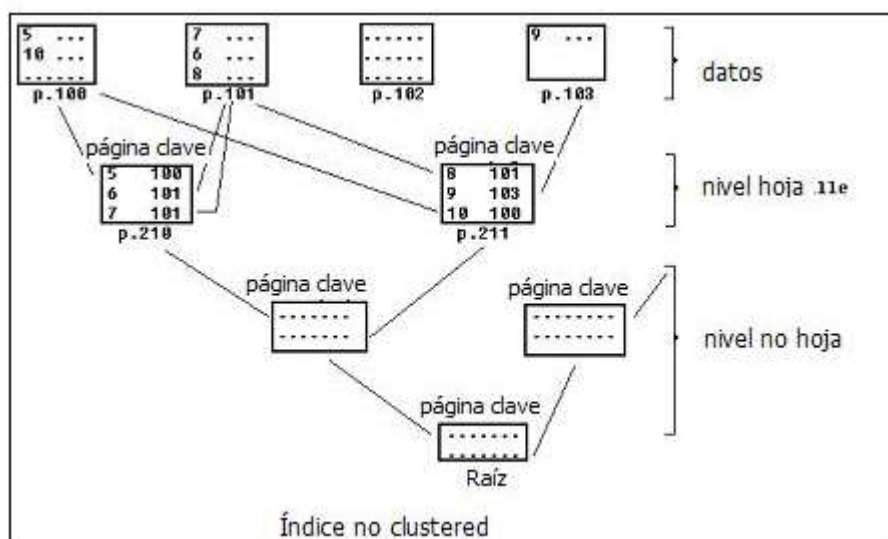
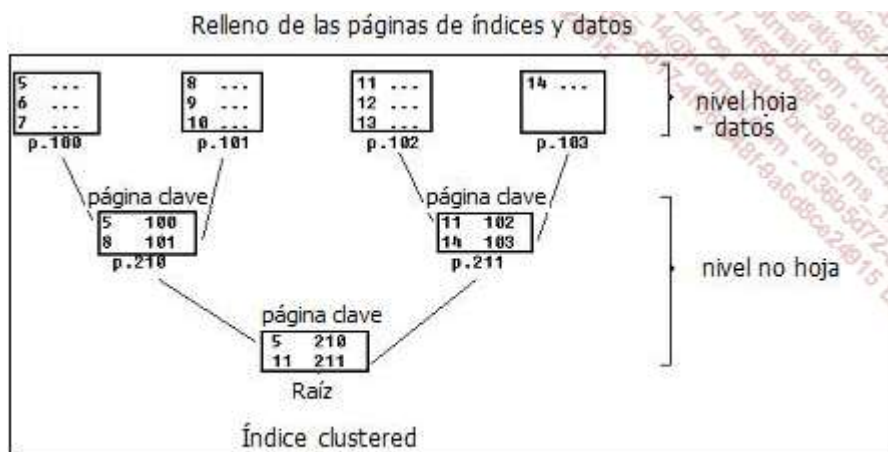
PAD\_INDEX

Indica la tasa de ocupación de las páginas no hoja del índice. Esta opción solo se puede utilizar con FILLFACTOR, cuyo valor se recupera.

grupoArchivo

Grupo de archivos sobre el que se crea el índice.

Los índices particionados no se tienen en cuenta a este nivel. Los elementos de sintaxis solo se aplican en los índices definidos completamente sobre un único grupo de archivos.



IGNORE\_DUP\_KEY

Esta opción autoriza entradas dobles en los índices únicos. Si se activa esta opción, se genera un mensaje de aviso cuando se inserta un dato duplicado y SQL Server ignora la inserción del registro. En caso contrario, se genera un error.

DROP\_EXISTING

Indica que se debe eliminar el índice ya existente.

#### ONLINE

Cuando esta opción está activada (ON) al generar el índice, todavía se puede acceder a los datos de la tabla en modo lectura y modificación. Esta opción está disponible a partir de SQL Server 2005 y está desactivada por defecto.

#### STATISTICS\_NORECOMPUTE

Las estadísticas de índice obsoletas no se recalculan y será necesario usar el comando UPDATE STATISTICS para actualizarlas.

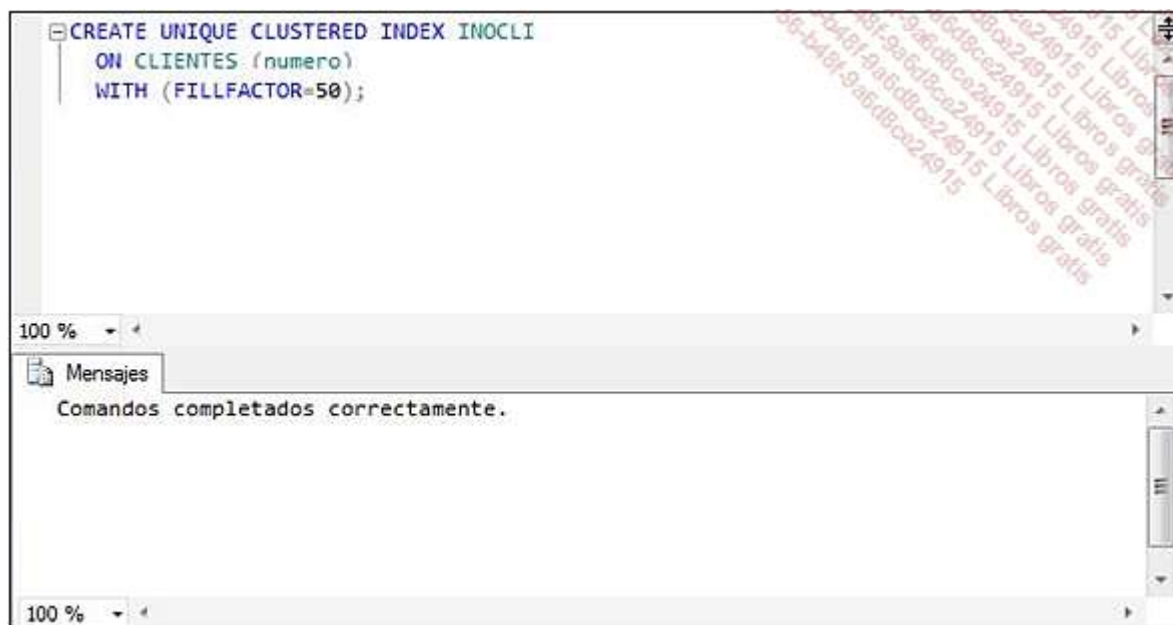
Si el servidor sobre el que se ejecuta SQL Server tiene varios procesadores, la creación del índice se puede paralelizar para ganar tiempo en la construcción del índice sobre las tablas de grandes dimensiones. La implantación de un plan de ejecución en paralelo para la construcción de un índice tiene en cuenta el número de procesadores del servidor, que se fija en la opción de configuración **max degree of parallelism (sp\_configure)** y el número de procesadores que no están sobrecargados por los threads SQL Server.

#### DATA\_COMPRESSION

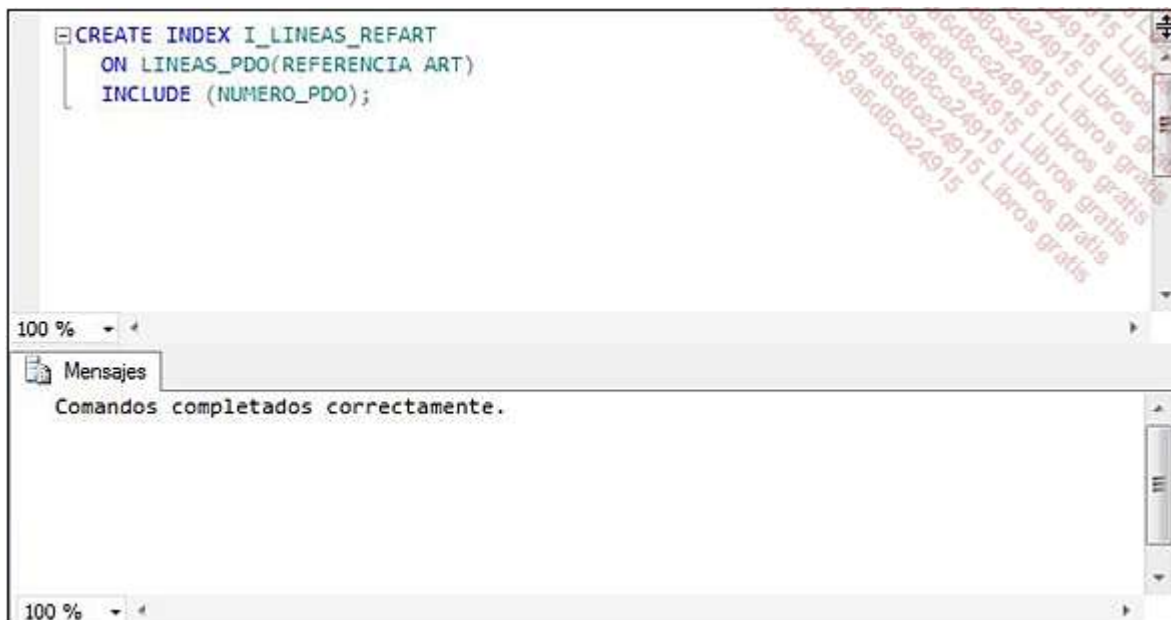
Esta opción permite indicar si el índice se va a comprimir o no, ya que si bien el índice organizado (CLUSTERED) hereda la propiedad de compresión de la tabla, este no es el caso de los índices no organizados (NONCLUSTERED). La compresión es a nivel registro (ROW) o página (PAGE) y solo afecta a las páginas a nivel hoja.

### Ejemplos

*Creación de un índice ordenado:*



*Creación de un índice que cubre la columna referencia\_art de la tabla LINEAS\_PDO. Este índice incluye a nivel hoja la columna numero\_pdo.*



*Creación de un índice filtrado sobre la columna etiqueta de la tabla ARTICULOS, pero solo para los artículos de las categorías 1 y 2.*



*Creación de un índice sobre una columna calculada, lo que es posible porque la columna calculada se ha creado con la propiedad PERSISTED.*



## 2. Eliminar un índice

Solo se pueden eliminar los índices definidos con la instrucción CREATE INDEX, usando DROP INDEX. Un índice se puede eliminar cuando su presencia no mejora el rendimiento significativamente, en comparación con el coste de mantenimiento.

Si la eliminación se hace durante una reconstrucción del índice, es mejor activar la opción DROP\_EXISTING de la instrucción CREATE INDEX, ya que ofrece mejor rendimiento.

### Sintaxis

```
DROP INDEX nombreIndice ON nombreTabla;
```

La antigua sintaxis DROP INDEX nombreTabla.nombreIndice se mantiene por razones de compatibilidad, pero no se debe utilizar en nuevos desarrollos.

## 3. Reconstruir un índice

El comando DBCC DBREINDEX todavía está disponible por razones de compatibilidad. Es preferible usar el comando ALTER INDEX, que permite mantener los índices.

El comando ALTER INDEX permite reconstruir un índice en particular o todos los índices asociados a una tabla. Durante la reconstrucción del índice, es posible indicar el factor de ocupación de las páginas hojas.

### Sintaxis

```
ALTER INDEX { nombreIndice | ALL }  
ON nombreTabla  
REBUILD  
[WITH(  
[PAD_INDEX = { ON | OFF },]  
[FILLFACTOR = x ,]  
[SORT_IN_TEMPDB = { ON | OFF },]  
[IGNORE_DUP_KEY = { ON | OFF },]  
[STATISTICS_NORECOMPUTE = { ON | OFF }]  
[ONLINE = { ON | OFF },])]  
[;]
```

## FILLFACTOR

Permite especificar el porcentaje de ocupación de las páginas a nivel hoja del índice.

## PAD\_INDEX

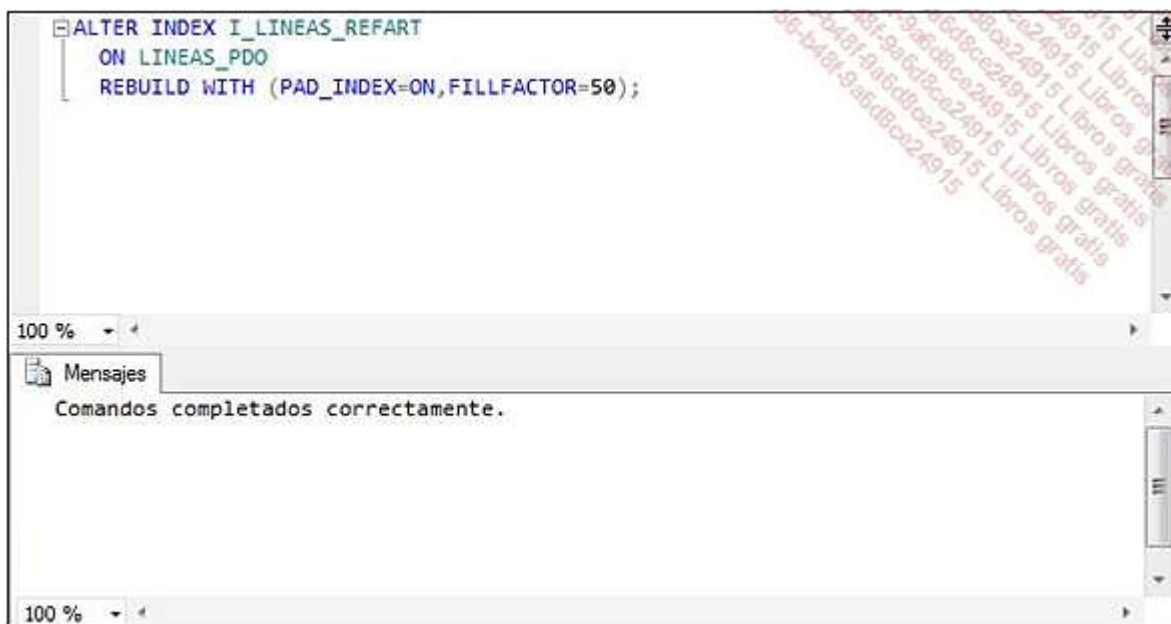
Permite aplicar en las páginas no hoja del índice, el mismo nivel de ocupación que el indicado con FILLFACTOR.

Las otras opciones de la instrucción ALTER INDEX tienen el mismo significado que las que se usan con la instrucción CREATE INDEX. Observe que la opción ONLINE tiene todo su sentido en este caso concreto, ya que permite reconstruir el índice, dejando a los usuarios trabajar con los datos de la tabla subyacente. Algunas operaciones serán más lentas, pero no estarán bloqueadas. Si la reconstrucción del índice se planifica en un momento en el que el servidor no está muy cargado, incluso puede llegar a ser transparente para los usuarios que estén trabajando con la base de datos en ese momento.

*El siguiente ejemplo muestra la reconstrucción de todos los índices de una tabla:*



*Este segundo ejemplo muestra la reconstrucción de un índice, especificando un valor para la opción FILLFACTOR y aplicando un factor de ocupación en las funciones no hoja.*



## 4. Las estadísticas

SQL Server usa los datos de la distribución de los valores de las claves para optimizar las consultas. Esta información se debe actualizar después de las modificaciones importantes de datos.

Aunque aquí se describa el procedimiento manual de creación y actualización de las estadísticas, es mucho mejor configurar la base de datos para que lo haga automáticamente. Es habitual que la degradación del rendimiento de un servidor se deba en parte o totalmente, a las estadísticas no actualizadas.

Las estadísticas se crean automáticamente para optimizar las consultas durante la creación del índice, pero en el caso de eliminación de estas estadísticas con la instrucción **DROP INDEX**, hay que ejecutar la instrucción **CREATE INDEX** para aplicarlas.

### Sintaxis

```
UPDATE STATISTICS nombreTabla [,nombreIndice]
[WITH {FULLSCAN|SAMPLE n {PERCENT|ROWS}|RESAMPLE}]
```

Si se omite el nombre del índice, se tiene en cuenta todos los índices.

**FULLSCAN**

Las estadísticas se crean a partir de un recorrido completo de la tabla, es decir, del 100% de los registros.

**SAMPLE n{PERCENT|ROWS}**

Las estadísticas derivan de una muestra representativa de los datos de la tabla. Esta muestra se puede expresar como un porcentaje o un número de registros. Si el tamaño de la muestra no es suficiente, SQL Server corrige este tamaño, para garantizar un recorrido cercano a las 1.000 páginas de datos. Este es el modo de muestreo de estadísticas por defecto.

**RESAMPLE**

Permite redefinir las estadísticas a partir de un nuevo muestreo.

Las estadísticas también se pueden actualizar automáticamente. Esta opción se debe definir cuando se construye la base de datos con el comando **ALTER DATABASE** o usando el procedimiento almacenado **sp\_autostats**.

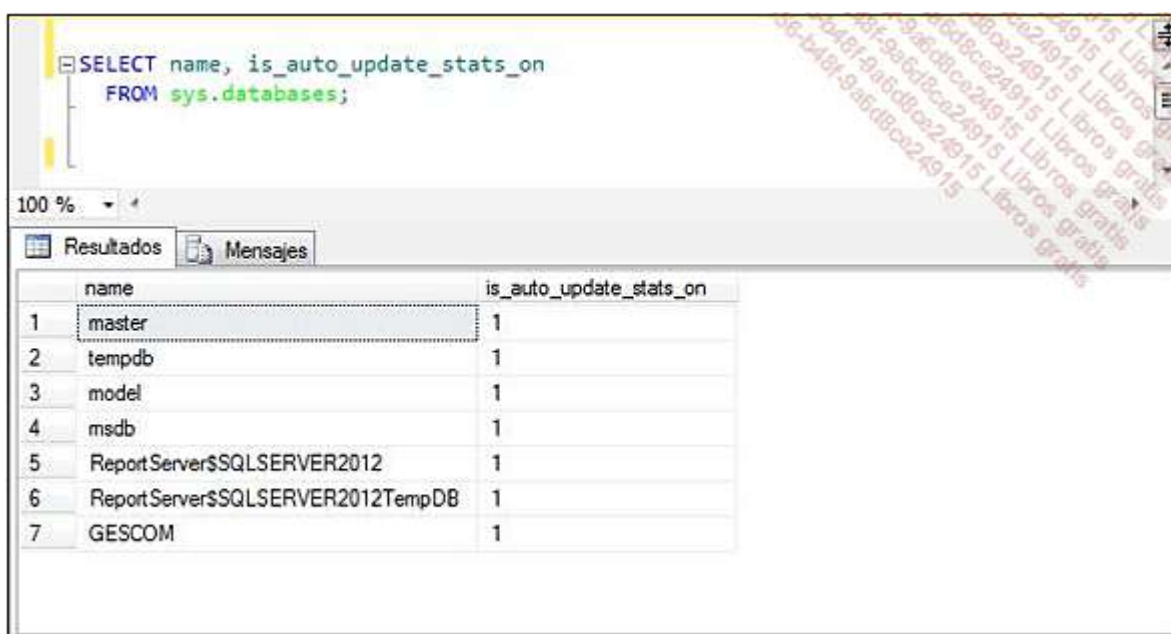
El procedimiento **sp\_createstats** permite definir estadísticas sobre todos los índices de datos de usuario de la base de datos, en una única operación.

*Configuración de la base de datos para una actualización automática de las estadísticas de índices:*



En modo automático, es el motor el que se encarga de calcular las estadísticas que faltan, mantenerlas actualizadas en función de las operaciones que se hacen sobre los datos y eliminar las inútiles.

Es posible saber si una base de datos configura de manera automática las estadísticas, consultando la columna **is\_auto\_update\_stats\_on** de la tabla **sys.databases** o viendo el valor de la propiedad **IsAutoUpdateStatistics** de la función **databasepropertyex**.



*Creación de las estadísticas de todos los índices sobre los datos no de sistema, de la base de datos GESCOM:*



```
exec sp_createstats

100 %

Mensajes

Tabla 'GESCOM.dbo.ARTICULOS'. creando estadísticas para las siguientes columnas:
PRECIOUNIT_ART
CODIGO_CAT

Tabla 'GESCOM.dbo.CLIENTES'. creando estadísticas para las siguientes columnas:
nombre
apellidos
direccion
codigopostal
ciudad
telefono
CODIGOREP

Tabla 'GESCOM.dbo.HISTO_FAC'. creando estadísticas para las siguientes columnas:
NUMERO_FAC
FECHA_FAC
NUMERO_PDO
BASE_IMPONIBLE
ESTADO_FAC

Tabla 'GESCOM.dbo.PEDIDOS'. creando estadísticas para las siguientes columnas:
FECHA_PDO
TASA_DESCUENTO
NUMERO_CLI
ESTADO_PDO

Tabla 'GESCOM.dbo.CATEGORIAS'. creando estadísticas para las siguientes columnas:
ETIQUETA_CAT

Tabla 'GESCOM.dbo.LINEAS_PDO'. creando estadísticas para las siguientes columnas:
NUMERO_LIN
CNTD_PDO

Tabla 'GESCOM.sys.queue_messages_1977058079'. creando estadísticas para las siguientes columnas:
priority
queuing_order
conversation_group_id
conversation_handle
message_sequence_number
message_id
message_type_id
service_id
service_contact_id
```

## 5. Información sobre los índices

La información de la estructura de los índices se puede obtener usando los procedimientos almacenados **sp\_help** o **sp\_helpindex**.

La instrucción DBCC SHOWCONTIG se mantiene solo por razones de compatibilidad hacia atrás. Por tanto, se aconseja no volver a usarla.

La información del tamaño y fragmentación de las tablas e índices se puede obtener con la función **sys.dm\_db\_index\_physical\_stats**.

### Sintaxis

```
dm_db_index_physical_stats (idBase | NULL,  
idObjeto | NULL,  
idIndice | NULL | 0,  
numeroParticion | NULL ,  
modo | NULL | DEFAULT)
```

idBase

Identificador de la base de datos. Es posible usar la función **db\_id()** para conocer el identificador de la base de datos actual. El valor NULL tiene en cuenta el conjunto de bases de datos definidas en el servidor e implica usar el valor NULL para el identificador del objeto, índice y partición.

idObjeto

Identificador del objeto (tabla o vista) del que deseamos la información. Este identificador se puede obtener con la función **object\_id()**. El uso del valor NULL permite indicar que queremos información de todas las tablas y vistas de la base de datos actual. Esto también implica usar el valor NULL para la opción **idIndice** y **numeroParticion**.

`idIndice`

Identificador del índice que se desea analizar. Si se especifica el valor NULL, el análisis se hace sobre todos los índices de la tabla.

`numeroParticion`

Número de la partición implicada. Si se especifica el valor NULL, se tienen en cuenta todas las particiones.

`modo`

Indica el modo para obtener la información: DEFAULT, NULL, LIMITED o DETAILED. El valor por defecto (NULL) es LIMITED.

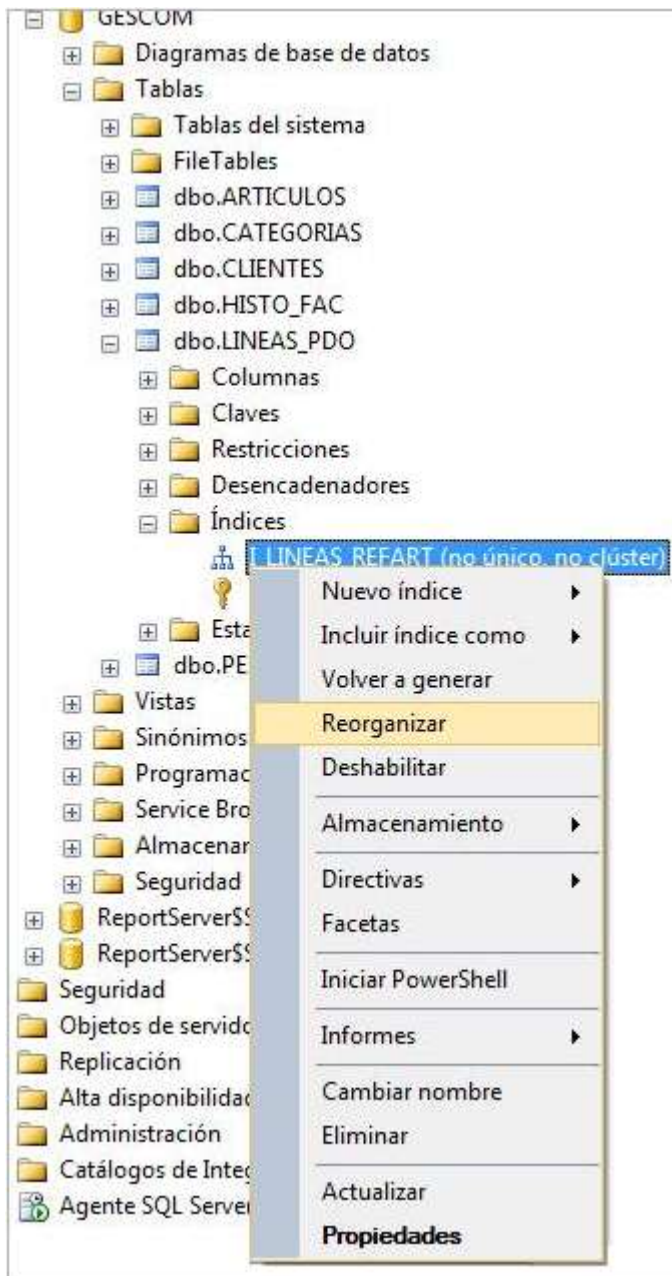
El procedimiento almacenado **sp\_spaceused** permite conocer el espacio en disco que utilizan los índices.

La función **INDEXPROPERTY** permite obtener la información de los índices.

Las funciones **INDEX\_COL** y **INDEXKEY\_PROPERTY** permiten obtener la información relativa a las columnas presentes en los índices.

También es posible consultar directamente las vistas de sistema. Las principales son **sys.index**, **sys.index\_columns** y **sys.stats**.

La manera más sencilla de administrar los índices y obtener la información de ellos de manera puntual es usar SQL Server Management Studio. Desde el explorador de objeto, desplegando una tabla, se presenta el nodo índice. Usando el menú contextual asociado a este nodo o a un índice particular, es posible crear un índice nuevo, ver sus propiedades, eliminarlo, reconstruirlo y reorganizarlo.



## Monitorizar y verificar las bases de datos y los objetos

Después de crear y usar las tablas e índices, a veces es útil verificar la coherencia de los datos y las páginas.

La instrucción DBCC lo permite.

```
DBCC CHECKDB [(nombreBase[,NOINDEX])]
```

Para todas las tablas de la base de datos, verifica la relación entre páginas de datos e índices, los criterios de ordenación y el puntero. También se da información del espacio en disco del archivo de trazas.

Para hacer estas comprobaciones, la ejecución de la instrucción DBCC CHECKDB implica la ejecución automática de las instrucciones DBCC CHECKALLOC y DBCC CHECKCATALOG a nivel de la base de datos y DBCC CHECKTABLE para cada tabla y vista.

La instrucción DBCC CHECKTABLE se puede ejecutar de manera autónoma a nivel de tabla, usando la siguiente sintaxis.

```
DBCC CHECKTABLE (nombreTabla[,NOINDEX|identificadorIndice])
```

Si se indica el identificador del índice, solo se comprobará este último. Si se especifica NOINDEX, los índices no se comprueban.

DBCC CHECKFILEGROUP permite ejecutar comprobaciones sobre un grupo de archivos en particular.

## Los esquemas

En SQL Server, los esquemas representan un conjunto lógico dentro de una base de datos. Permiten organizar mejor de manera lógica las tablas, vistas, procedimientos y funciones. Por defecto, durante la creación de un objeto, este se registra en el esquema del usuario actual. Por lo tanto, el esquema tiene el mismo nombre que el del usuario. Es posible asociar un esquema existente a un usuario, o crear un objeto en un esquema diferente al que tiene asociado el usuario (con la condición de que el administrador de la base de datos lo autorice).

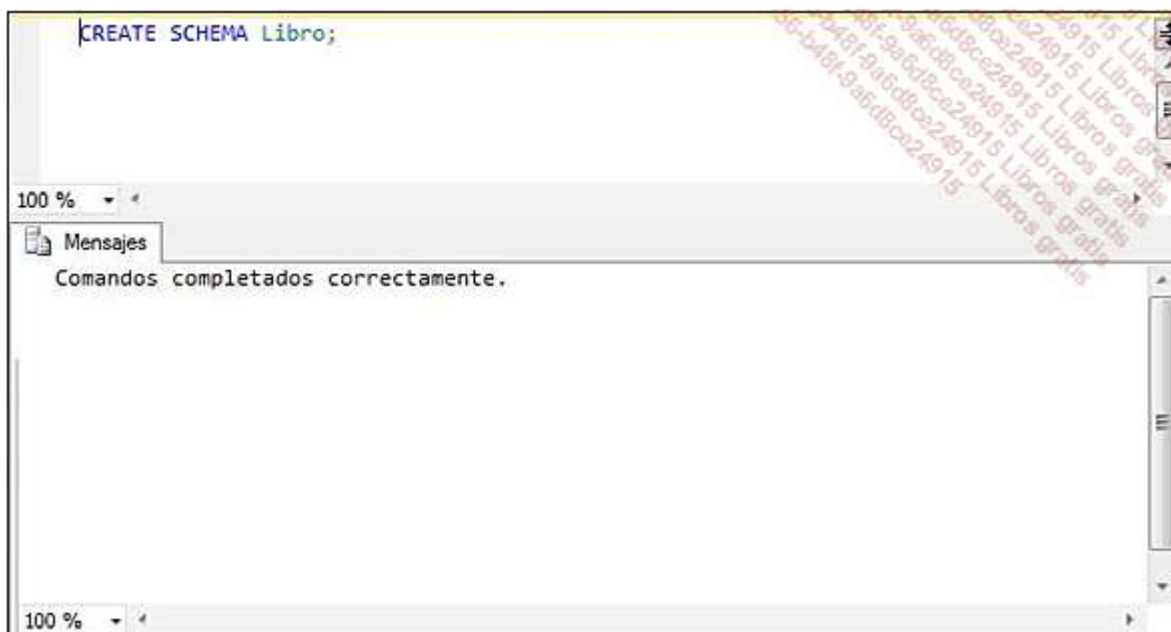
Durante la creación de la base de datos, existe el esquema dbo. Este esquema está presente en todas las bases de datos. Creando y usando otros esquemas, la organización lógica de los datos se mejora, ya que es necesario hacer referencia a los objetos, usando como prefijo el nombre del esquema (de hecho, los objetos ya no están en el esquema por defecto).

El esquema se define con la instrucción CREATE SCHEMA nombreEsquema.

Es posible crear objetos en este esquema, indicando durante la creación de las tablas, vistas, etc. el nombre como *nombreEsquema.nombreObjeto*.

### Ejemplo

En el siguiente ejemplo se crea el esquema Libro.



Las ordenes en sql

## Aspectos generales

Microsoft Transact SQL es un lenguaje de consultas mejorado respecto a SQL. El SQL (*Structured Query Language*) es el lenguaje estándar, creado por IBM en los años 70, para la gestión de los SGBDR (Sistemas de gestión de bases de datos relacionales).

Este lenguaje está compuesto por tres categorías de instrucciones:

- El lenguaje de definición de datos (*Data Description Language* - DDL) que permite la creación, modificación y eliminación de objetos SQL (TABLES, INDEX, VIEWS, PROCEDURES, etc.).
- El lenguaje de manipulación de datos (*Data Manipulation Language* - DML) que proporciona instrucciones de creación, actualización, eliminación y extracción de los datos almacenados.
- El lenguaje de control de acceso (*Data Control Language* - DCL) para la gestión de los accesos a los datos, las transacciones y la configuración de las sesiones y los accesos a las bases de datos.

Además, Transact SQL tiene en cuenta las funcionalidades procedimentales, como la gestión de las variables, las estructuras de control de flujo, los cursores y los lotes de instrucciones. Por lo tanto, es un lenguaje completo que cuenta con instrucciones, manipula los objetos SQL, admite la programación y usa expresiones.

Con Transact SQL, es posible definir funciones y procedimientos que se ejecutan directamente en el servidor de base de datos. Este tipo de procedimientos y funciones son particularmente interesantes cuando las operaciones para generar el resultado se hacen con un volumen de información importante. Así mismo, el desarrollo con Transact SQL se adapta perfectamente a un contexto de funcionalidades compartidas, ya que los procedimientos y funciones que alberga el servidor se pueden ejecutar desde cualquier entorno cliente (.NET, Java...).

Ahora es posible, pero no obligatorio, utilizar el punto y coma como marcador de fin de instrucción.

## 1. Expresiones

En la mayor parte de las sintaxis Transact SQL se pueden utilizar expresiones o combinaciones de expresiones para administrar valores o sacar partido de las capacidades de programación del lenguaje. Las expresiones pueden tener diferentes formas:

### Constantes

#### Ejemplo

Carácter	'QWERTY', 'Escuela nacional de 'Informática'	
Numérico	10, -15.26, 1.235e-5	
Fecha		
constante	fecha	hora
'801215'	5 de Diciembre de 1980	00:00:00:000
'15/12/1980'	ídem	ídem
'15-12-80 8:30'	ídem	8:30:00:000
'8:30:2'	1 de Enero de 1900	08:30:02:000
'15.12.1980 8:30pm'	15 de Diciembre de 1980	20:30:00:000
Binario	0x05, 0xFF, 0x5aef1b	
Nulo	NULL	

### Nombres de columna

Se podrá usar un nombre de columna como expresión, siendo el valor de la expresión el valor "almacenado" de la columna.

### Funciones

Podemos utilizar como expresión cualquier función. El valor de la expresión es el resultado que devuelve la función.

#### Ejemplo

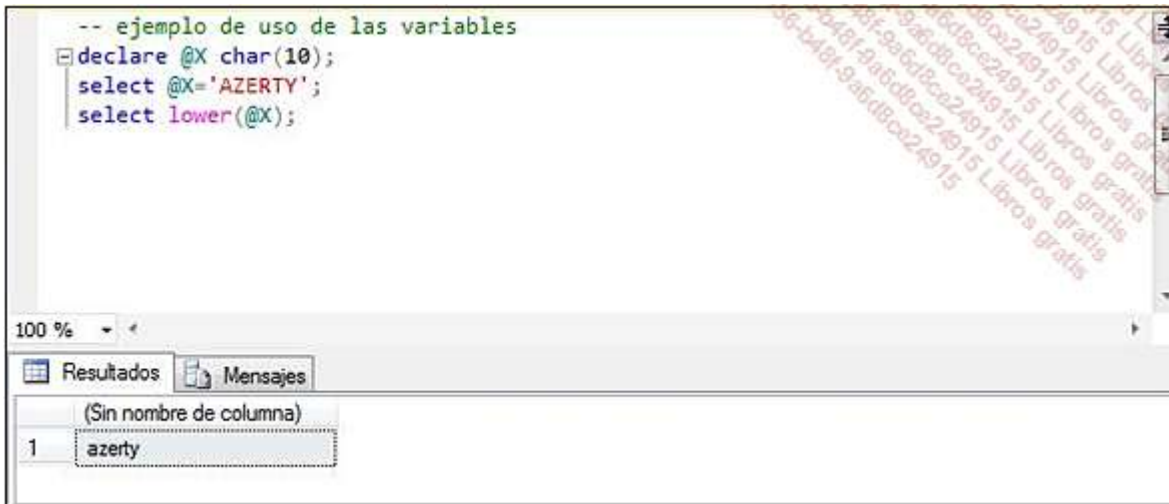
expresión	valor
SQRT(9)	3

```
substring('ABPDOF',2,3)      'BCD'
```

## Variables

Las variables se pueden usar como expresiones o en una expresión, usando la siguiente forma @nombre\_de\_variable o @@nombre\_de\_variable. El valor de la expresión es el valor de la variable.

### Ejemplo



The screenshot shows a SQL query window with the following code:

```
-- ejemplo de uso de las variables
declare @X char(10);
select @X='AZERTY';
select lower(@X);
```

Below the query window, the 'Resultados' (Results) tab is active, displaying a single row with the value 'azerty'.

(Sin nombre de columna)
1 azerty

## Subconsultas

Se puede usar una consulta SELECT entre paréntesis como expresión. El resultado de la consulta puede ser un valor único o un conjunto de valores.

### Ejemplo

Almacenar el número de clientes en una variable:



The screenshot shows a SQL query window with the following code:

```
declare @numero int;
select @numero=count(*) from CLIENTES;
select 'número de clientes'=@numero;
```

Below the query window, the 'Resultados' (Results) tab is active, displaying a single row with the value '27'.

número de clientes
1 27

## Expresiones booleanas

Están destinadas a verificar las condiciones (IF, WHILE, WHERE, etc.). Estas expresiones se forman de la siguiente manera:

```
expresion1 operador expresion2
```

## 2. Operadores

Los operadores van a permitir formar expresiones calculadas, booleanas o combinaciones de expresiones.

## Operadores aritméticos

Permiten realizar cálculos sencillos y devolver un resultado.

+ Suma

- Resta

\* Multiplicación

/ División

% Módulo (resto de la división)

(...) Paréntesis

### Ejemplo



```
DECLARE @TTC float;
DECLARE @TxIVA float;
DECLARE @X int;

SELECT @TxIVA=19.6;
SELECT @TTC=(PRECIOUNIT_ART*(1+(@TxIVA/100)))
FROM ARTICULOS
WHERE REFERENCIA_ART='SOU16';

SELECT @X=COUNT(*)
FROM ARTICULOS;
IF (@X%2)=0 PRINT 'X es par'
ELSE PRINT 'X es impar';
```

Mensajes

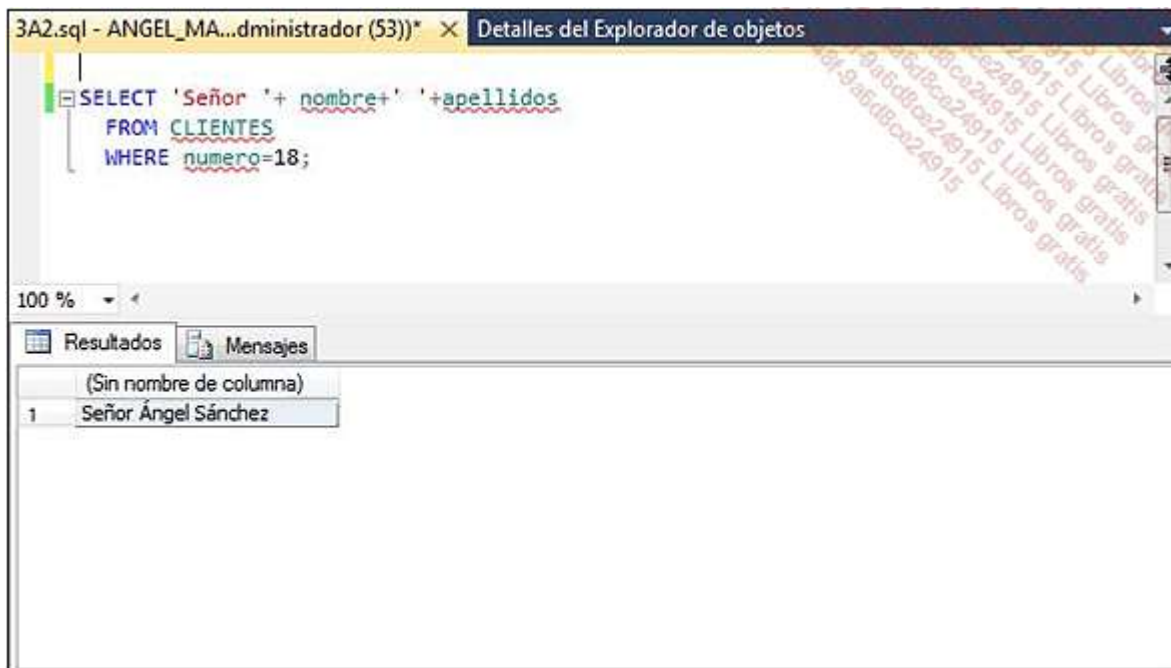
X es par

Los operadores + y - también funcionan con fechas.

## Manipulación de cadenas de caracteres

La concatenación permite formar una única cadena de caracteres, a partir de varias expresiones de tipo carácter. El operador de concatenación es el signo más (+).

### Ejemplo



## Operadores de bit

Permiten operar con enteros traducidos como valores binarios.

& Y

| O

^ O Exclusivo

~ NO

## Operadores de comparación

Permiten constituir expresiones booleanas comparando expresiones. Estas expresiones se pueden poner entre paréntesis.

$exp1 = exp2$

Igual.

$exp1 > exp2$

Superior.

$exp1 \geq exp2$  o  $exp1 \geq exp2$

Superior o igual.

$exp1 < exp2$

Inferior.

$exp1 \leq exp2$  o  $exp1 \leq exp2$

Inferior o igual.

$exp1 \neq exp2$  o  $exp1 \neq exp2$



Diferente.

`exp IN (exp1, exp2, ...)`

Compara cada expresión de la lista.

`exp IS NULL`

Verifica el valor NULL. Para verificar si una variable contiene el valor NULL, es imprescindible utilizar el operador IS NULL.

`exp LIKE 'mascara'`

Filtra la cadena de caracteres o la fecha siguiendo la máscara que se ha especificado.

La máscara puede estar formada por:

—

Un carácter cualquiera.

%

n caracteres cualesquiera.

[ab...]

Un carácter de la lista ab...

[a-z]

Un carácter del intervalo a-z.

[^ab...]

Un carácter fuera de la lista o del intervalo especificado.

ab...

El carácter en sí mismo.

Para usar `_`, `%`, `[` y `^` como caracteres de búsqueda, hay que ponerlos entre corchetes `[]`.

### Ejemplo

máscara	Cadenas correspondientes
'G%'	empieza por "G"
'_X%1'	segundo carácter "X" y último "1"
'%[1-9]'	terminan por una cifra comprendida entre "1" y "9"
'[^XW]%'	no empiezan ni por X ni por W
'LE[_]%'	empiezan por "LE_"

`exp LIKE 'mascara' ESCAPE 'c'`

Cuando la máscara tiene un carácter especial como `_` o `*`, es necesario poner delante un carácter específico. A este carácter específico se le llama carácter de escape. Para adaptarse a todas las situaciones posibles, el carácter de escape es libre y solo hay que especificarlo después de la palabra clave ESCAPE.

`exp BETWEEN min AND max`

Buscar en el intervalo compuesto por los valores min y max (ambos incluidos).

`EXISTS (subconsulta)`

Devuelve VERDADERO si la subconsulta ha devuelto, al menos, un registro.

## Operadores lógicos

Permiten combinar expresiones booleanas (expb). Retornan un valor booleano.

`expb1 OR expb2`

Verdadero si una de las dos expresiones es verdadera.

`expb1 AND expb2`

Verdadero si las dos expresiones son verdaderas.

`NOT expb`

Verdadero si expb es falsa.

## 3. Funciones

Hay muchas funciones disponibles para valorizar las columnas o hacer pruebas. Se darán ejemplos con la instrucción SELECT.

Es posible agrupar por tipo las funciones que SQL propone de manera estándar: rowset, agregación, ranking y escalar. Este último tipo de funciones se clasifica en categorías: matemática, cadena de caracteres, fecha... porque son muy numerosas.

Algunas funciones, más particularmente las funciones que permiten manipular los datos de tipo carácter, tienen en cuenta la clasificación definida a nivel del servidor.

### a. Funciones de agregación

Estas funciones devuelven un valor único como resultado de un cálculo estadístico sobre una selección de registros.

Las funciones agregadas son deterministas, es decir, aplicada a un mismo conjunto de datos, la función siempre devolverá el mismo resultado.

Con excepción de la función COUNT, las funciones agregadas no tienen en cuenta los valores NULL.

`COUNT (*)`

Cuenta los registros seleccionados.

`COUNT ([ALL|DISTINCT] expr)`

Cuenta todas las expresiones no nulas (ALL) o las expresiones no nulas únicas (DISTINCT).

`COUNT_BIG`

Su funcionamiento es idéntico a la función COUNT, pero el resultado devuelto está en formato bigint, en lugar de int como sucede con la función COUNT.

`SUM([ALL|DISTINCT] exprn)`

Suma todas las expresiones no nulas (ALL) o las expresiones no nulas únicas (DISTINCT).

`AVG([ALL|DISTINCT] exprn)`

Media de todas las expresiones no nulas (ALL) o de las expresiones no nulas únicas (DISTINCT).

`MIN(exp) o MAX(exp)`

Valor mínimo o máximo de todas las expresiones.

`STDEV ([ALL|DISTINCT] exprn)`

Desviación típica de todos los valores de la expresión dada, no nulos (ALL) o de las expresiones no nulas únicas (DISTINCT).

`STDEVP ([ALL|DISTINCT] exprn)`

Desviación típica de la población para todos los valores de la expresión dada, no nulos (ALL) o de las expresiones no nulas únicas (DISTINCT).

`VAR ([ALL|DISTINCT] exprn)`

Variancia de todos los valores de la expresión dada, no nulos (ALL) o de las expresiones no nulas únicas (DISTINCT).

`VARP ([ALL|DISTINCT] exprn)`

Varianza de la población para todos los valores de la expresión dada, no nulos (ALL) o de las expresiones no nulas únicas (DISTINCT).

`GROUPING`

Se usa junto con ROLLUP y CUBE. Indica el valor 1 cuando el registro se genera con una instrucción ROLLUP o CUBE. En caso contrario indica el valor 0.

`CHECKSUM (*|exp[, ...])`

Permite calcular un código de control en relación a un registro de la tabla o una lista de expresiones, varias columnas por ejemplo. Esta función permite generar un código hash.

`CHECKSUM_AGG([ALL|DISTINCT] exp)`

Permite calcular un valor hash en función de un grupo de datos. Este código de control permite saber rápidamente si las modificaciones tienen lugar sobre un grupo de datos, ya que el valor que genera esta función ya no será el mismo.

## **b. Funciones matemáticas**

Estas funciones devuelven un valor como resultado de cálculos matemáticos clásicos (álgebra, trigonometría, logaritmos, etc.).

`ABS(exprn)`

Valor absoluto de `exprn`.

`CEILING(exprn)`

Entero más pequeño igual o superior a  $\text{expn}$ .

`FLOOR (expn)`

Entero más grande inferior o igual a  $\text{expn}$ .

`SIGN (expn)`

Devuelve 1 si  $\text{expn}$  es positiva, -1 si es negativa y 0 si es cero.

`SQRT (expn)`

Raíz cuadrada de  $\text{expn}$ .

`POWER (expn, n)`

$\text{expn}$  elevado a la potencia  $n$ .

`SQUARE (expn)`

Calcula el cuadrado de  $\text{expn}$ .

### c. Funciones trigonométricas

`PI ()`

Valor del número PI.

`DEGREES (expn)`

Conversión a grados de  $\text{expn}$  en radianes.

`RADIANS (expn)`

Conversión a radianes de  $\text{expn}$  en grados.

`SIN (expn)` , `TAN (expn)` , `COS (expn)` , `COT (expn)`

Seno, tangente, coseno y cotangente del ángulo  $\text{expn}$  en radianes.

`ACOS (expn)` , `ASIN (expn)` , `ATAN (expn)`

Arco coseno, arco seno y arco tangente de  $\text{expn}$ .

`ATN2 (expn1, expn2)`

Ángulo (radianes) cuya tangente pasa por el punto  $\text{expn1}$ ,  $\text{expn2}$ .

### d. Funciones logarítmicas

`EXP (expn)`

Exponencial de  $\text{expn}$ .

`LOG (expn, [base])`

Calcula el logaritmo neperiano (base=**e**) si el argumento base no tiene valor; en caso contrario, calcula el logaritmo en relación al valor de la base que se ha especificado.

LOG10 (expn)

Logaritmo en base 10 de expn.

## e. Funciones diversas

RAND ([expn])

Número aleatorio comprendido entre 0 y 1. expn representa al valor de inicio.

ROUND (expn, n [, f])

Redondea expn con precisión n. Si n es positivo, representa el número de decimales. Si es cero, redondea al entero más cercano. Si es negativo, redondea a la decena más cercana (-1), a la centena (-2), etc., o devuelve 0 si n es superior al número de cifras enteras de expn. Si se especifica el valor para f, su función es truncar expn. Los valores que pueden tomar f, se interpretan como para n. El valor de f solo se tendrá en cuenta si n es 0.

### Ejemplo

expn	n	f	resultado
1.256	2		1.260
1.256	4		1.256
1.256	0		1.000
11.25	-1		10
11.25	-2		.00
11.25	-3		.00
150.75	0		151
150.75	0	1	150

## f. Funciones de tipo fecha

Las funciones de tipo fecha manipulan expresiones de tipo fecha y usan los formatos que representan la parte de la fecha que se debe administrar.

Estos formatos son:

Formato	Abreviatura	Significado
year	yy, yyyy	Año (de 1.753 a 9.999)
quarter	qq, q	Trimestre (de 1 a 4)
month	mm, m	Mes (de 1 a 12)
day of year	dy, y	Día del año (de 1 a 366)
day	dd, d	Día del mes (de 1 a 31)
weekday	dw, ww	Día de la semana (de 1 Lunes a 7 Domingo)
hour	hh	Hora (de 0 a 23)
minute	mi, n	Minuto (de 0 a 59)
seconds	ss, s	Segundo (de 0 a 59)
millisecond	ms	Milisegundo (de 0 a 999)

GETDATE ()

o  
GETUTCDATE ()

Fecha y hora de sistema.

DATENAME (formato, expd)

Devuelve

la parte fecha en formato texto.

`DATEPART (formato,expd)`

Devuelve el valor de la parte fecha, según el formato.

Es posible configurar el primer día de la semana con la función `SET DATEFIRST(numeroDía)`. Los días se numeran partiendo de 1 para el lunes, hasta 7 para el domingo. Es posible conocer la configuración actual, consultando la función `@@datefirst`.



`DATEDIFF (formato,expd1,expd2)`

Diferencia según el formato entre las dos fechas.

`DATEADD (formato,n,expd)`

Añade n formato a la fecha expd.

`DAY (expd)`

Devuelve el número de día del mes.

`MONTH (expd)`

Devuelve el número del mes.

`YEAR (expd)`

Devuelve el año.

Para las tres funciones anteriores, si el valor del argumento es 0, SQL Server efectúa sus cálculos partiendo del 1 de enero de 1.900.

`SWITCHOFFSET (datetimeoffset, zonaHoraria)`

Convierte el dato de tipo `DATETIMEOFFSET` que se pasa como argumento, a la zona horaria que se pasa en el segundo argumento.

`SYSDATETIME`

Devuelve la fecha y hora actual del servidor en formato **datetime2**. No se incluye el desfase respecto al tiempo GMT (OFFSET). Esta función ofrece más precisión que el valor que retornan GETDATE() o GETUTCDATE().

`SYSDATETIMEOFFSET`

Funciona igual que SYSDATETIME, pero el dato de tipo fecha y hora devuelto está en formato **datetimeoffset** e incluye el desfase en relación a la hora GMT.

`EOMONTH (date [, numeroMes])`

Esta función permite conocer el último día del mes correspondiente a la fecha que se pasa como argumento. Si se indica un número de mes, la función calcula el último día del mes de la fecha que resulta de añadir el número de mes a la fecha que se pasa como argumento.

Para facilitar la construcción de datos de tipo fecha y hora a partir de valores enteros, SQL Server ofrece las funciones que se detallan a continuación:

`DATEFROMPARTS (año, mes, día)`

Devuelve un valor de tipo **date**, formado a partir de números enteros para el año, el mes y el día.

`DATETIME2FROMPARTS (año, mes, día, hora, minuto, segundo, fracción, precisión)`

Devuelve un valor de tipo **datetime2**, formado a partir de los números enteros que se suministran como argumentos para representar los diferentes valores.

`DATETIMEFROMPARTS (año, mes, día, hora, minuto, segundo, milisegundos)`

Devuelve un valor de tipo **datetime**, formado a partir de los números enteros que se suministran como argumentos para representar los diferentes valores.

`DATETIMEOFFSETFROMPARTS(año, mes, día, hora, minuto, segundo, fracción, desfaseHora, desfaseMinuto, precisión)`

Devuelve un valor de tipo **datetimeoffset**, formado a partir de los números enteros que se suministran como argumentos para representar los diferentes valores.

`SMALLDATETIMEFROMPARTS (año, mes, día, hora, minuto)`

Devuelve un valor de tipo **smalldatetime**, formado a partir de los números enteros que se suministran como argumentos para representar los diferentes valores.

`TIMEFROMPARTS (hora, minuto, segundo, fracción, precisión)`

Devuelve un valor de tipo **time**, formado a partir de los números enteros que se suministran como argumentos para representar los diferentes valores.

## g. Funciones de tratamiento de cadena de caracteres

`ASCII (expc)`

Valor del código ASCII del primer carácter de expd.

`UNICODEGO (expc)`

Valor numérico correspondiente al código carácter Unicode de expd.

`CHAR (expn)`

Carácter correspondiente al código ASCII `expd`.

`NCHAR (expn)`

Carácter unicode que corresponde al código numérico `expd`.

`LTRIM (expc) , RTRIM (expc)`

Elimina los espacios no significativos a la derecha (`RTRIM`) o a la izquierda (`LTRIM`) de `expd`.

`STR (expn, [lg[, nbd]])`

Convierte el número `expd` en cadena de longitud total `lg`, con los `nbd` caracteres a la derecha de la marca decimal.

`SPACE (n)`

Devuelve `n` espacios.

`REPLICATE (expc, n)`

Devuelve `n` veces `expd`.

`CHARINDEX ('mascara', expc) , PATINDEX ('%mascara%', expc)`

Devuelve la posición de inicio de la primera expresión '`mascara`' en `expd`. `PATINDEX` permite utilizar caracteres genéricos (ver `LIKE`) y trabajar con los tipos `text`, `char` y `varchar`.

`LOWER (expc) , UPPER (expc)`

Cambia entre mayúsculas y minúsculas. Convierte `expc` a minúsculas o a mayúsculas.

`REVERSE (expc)`

Devuelve `expc` al revés (leído de derecha a izquierda).

`RIGHT (expc, n)`

Devuelve los `n` caracteres más a la derecha de `expc`.

`LEFT (expc, n)`

Devuelve los `n` caracteres más a la izquierda de `expc`.

`SUBSTRING (expc, dp, lg)`

Devuelve `lg` caracteres de `expc` a partir de `dp`.

`STUFF (expc1, dp, lg, expc2)`

Elimina `lg` caracteres de `expc1` a partir de `dp`. Después inserta `expc2` en la posición `dp`.

`SOUNDEX (expc)`

Devuelve el código fonético de `expc`. Este código se forma con la primera letra de `expc` y tres cifras.

`DIFFERENCE (expc1, expc2)`



Compara los SOUNDEX de las dos expresiones. Devuelve un valor entre 1 y 4; 4 significa que las dos expresiones son muy similares.

`LEN (expc)`

Devuelve el número de caracteres de `expc`.

`QUOTENAME (expc [, delimitador])`

Permite transformar `expc` en un identificador válido para SQL Server.

`REPLACE (expc1, expc2, expc3)`

Permite sustituir en `expc1` todas las ocurrencias de `expc2` por `expc3`.

`CONCAT (expc1, expc2 [, ...])`

Devuelve una cadena de caracteres, formada por la concatenación de las diferentes cadenas de caracteres que se pasan como argumento.

`FORMAT (valor, formato [, ubicación])`

Devuelve la representación en forma de cadena de caracteres de la fecha o del valor numérico que se pasa como argumento, respetando el formato deseado. Si deseamos una representación particular, se puede especificar como argumento adicional la ubicación; por ejemplo, la representación de las fechas no es la misma entre hispanohablantes y anglosajones.

## h. Funciones de sistema

`COALESCE (exp1, exp2, ...)`

Devuelve la primera `exp` no NULL.

`COL_LENGTH ('nombreTabla', 'nombreColumna')`

Longitud de la columna.

`COL_NAME (idTabla, idCol)`

Nombre de la columna cuyo número de identificación es `idCol`, en la tabla identificada por `idTabla`.

`DATALENGTH (exp)`

Longitud en bytes de la expresión.

`DB_ID ([ 'nombreBase' ])`

Número de identificación de la base de datos.

`DB_NAME ([idBase])`

Nombre de la base de datos identificada por `idBase`.

`GETANSINULL ([ 'nombreBase' ])`

Devuelve 1 si la opción "ANSI NULL DEFAULT" está activada en la base de datos.

`HOST_ID()`

Número de identificación de la estación de trabajo.

HOST\_NAME()

Nombre de la estación de trabajo.

IDENT\_INCR ('nombreTabla')

Valor del incremento definido para la columna IDENTITY de la tabla o de la vista que afecta a una tabla, con una columna IDENTITY.

IDENT\_SEED ('nombreTabla')

Valor inicial definido para la columna IDENTITY de la tabla o de la vista que afecta a una tabla, con una columna IDENTITY.

IDENT\_CURRENT ('nombreTabla')

Devuelve el último valor de tipo identidad que se usa para esta tabla.

INDEX\_COL ('nombreTabla', idIndices, idClave)

Nombre de la columna indexada que corresponde al índice.

ISDATE(exp)

Devuelve 1 si la expresión de tipo varchar tiene un formato de tipo fecha válido.

ISNULL(exp, valor)

Devuelve valor si exp es NULL.

ISNUMERIC(exp)

Devuelve 1 si la expresión de tipo varchar tiene un formato numérico válido.

NULLIF(exp1, exp2)

Devuelve NULL si exp1 = exp2.

OBJECT\_ID('nombre')

Número de identificación del objeto 'nombre'.

OBJECT\_NAME(id)

Nombre del objeto identificado por id.

STATS\_DATE(idTabla, idIndice)

Fecha de la última actualización del índice.

SUSER\_SID(['nombreAcceso '])

Número de identificación que corresponde al nombre de acceso.

SUSER\_SNAME([id])

Nombre de acceso identificado por id.

`USER_NAME([id])`

Nombre del usuario identificado por id. Solo se usa con la restricción DEFAULT (funciones niládicas).

`CURRENT_TIMESTAMP`

Fecha y hora de sistema, equivalente a GETDATE().

`SYSTEM_USER`

Nombre de acceso.

`CURRENT_USER, USER, SESSION_USER`

Nombre del usuario de la sesión.

`OBJECT_PROPERTY(id,propiedad)`

Permite recuperar las propiedades de un objeto de la base de datos.

`ROW_NUMBER`

Permite conocer el número secuencial de un registro en una partición, de un conjunto de resultados. Esta numeración empieza por 1 para el primer registro de cada partición.

`RANK`

Permite conocer el rango de un registro de una partición de un conjunto de resultados. El rango de un registro es superior en una unidad al rango del registro de la misma partición.

`DENSE_RANK`

Funciona como RANK, pero solo se aplica a los registros presentes en el conjunto de resultados.

`HAS_DBACCESS('nombreBase')`

Permite saber si es posible acceder a la base de datos que se pasa como argumento con el contexto de seguridad actual (devuelve=1) o si no es posible (devuelve=0).

`HAS_PERMS_BY_NAME`

Permite saber, usando programación, si se tiene un determinado permiso o no. Este tipo de función puede resultar interesante en caso de un cambio de contexto.

`KILL`

Esta función, bien conocida por los usuarios de los sistemas Unix/Linux, permite finalizar la sesión de un usuario. Para finalizar una conexión, es necesario pasar como argumento el identificador de la sesión (sessionID o SPID) o el identificador del lote que se está ejecutando actualmente (UOW - *Unit Of Work*). El UOW se puede conocer consultando la columna request\_owner\_guid de la vista sys.dm\_tran\_locks. El identificador de sesión se puede obtener consultando el contenido de la variable @@SPID, ejecutando el procedimiento **sp\_who** o incluso consultando la columna session\_id de las vistas sys.dm\_tran\_locks o sys.dm\_exec\_sessions. Si la anulación se hace sobre una transacción voluminosa, la operación puede ser relativamente larga.

`NEWID()`

Permite generar un valor de tipo UniqueIdentifier.

NEWSEQUENTIALID()

Esta función está destinada a generar un valor por defecto. Permite generar el siguiente valor de tipo UniqueIdentifier. Como el siguiente valor de tipo UniqueIdentifier es predecible, esta función no se debe utilizar en las estructuras que necesiten un alto nivel de seguridad.

## El SQL-DML

El lenguaje de manipulación de datos (*SQL-Data Manipulation Language*), está formado por instrucciones que permiten la gestión, visualización y extracción de los registros de las tablas y vistas.

### 1. Creación de registros

La creación de registros en una tabla o vista que cumplen determinadas condiciones se hace con el comando INSERT.

Se controlarán las restricciones y los triggers se ejecutarán durante la ejecución del comando. La forma "INSERT..... VALUES....." crea un solo registro, mientras que "INSERT..... SELECT....." permite crear varios registros.

#### Sintaxis

```
INSERT [INTO] nombreObjeto [(col,...)] {DEFAULT VALUES|VALUES  
(val,...)|consulta| EXECUTE procedimiento }
```

nombreObjeto

Nombre válido de una tabla o vista.

(col,...)

Lista de columnas que se tienen que valorizar. Las columnas que no se mencionan tomarán el valor NULL. Si la lista se omite, todas las columnas deberán tener un valor.

DEFAULT VALUES

Todas las columnas mencionadas toman su valor por defecto o NULL si no existe.

(val,...)

Lista de valores formada por expresiones constantes, palabras clave NULL o DEFAULT, o variables. Debe tener tantos valores como columnas se deban valorizar, del mismo tipo y en el mismo orden.

consulta

Instrucción SELECT que devuelve tantos valores, en el mismo orden y del mismo tipo, que las columnas que se deben valorizar. Esta forma de sintaxis permite insertar varios registros en una sola operación.

Procedimiento

Nombre de un procedimiento almacenado, local o remoto. Solo los valores de las columnas que retorna la cláusula SELECT del procedimiento valorizarán las columnas implicadas en el INSERT.

#### Ejemplos

*En este primer ejemplo, se indica la lista de todas las columnas de la tabla y se proporcionan los valores. Observe la ausencia de valor para la dirección indicada con la palabra clave null y el uso del valor por defecto para indicar la ciudad.*

```
INSERT INTO CLIENTES(numero,apellidos,nombre,direccion,codigoPostal,ciudad,telefono,CODIGOREP)
VALUES (251,'Sánchez','Ángel',Null,'28000','Predeterminado','02 28 28 25 10','CD');
```

Mensajes

(1 filas afectadas)

Creación de un artículo (precio desconocido):

```
INSERT INTO ARTICULOS (REFERENCIA ART,NOMBRE ART, CODIGO CAT)
VALUES ('SOU26','Microsoft Arc Mouse',40);
```

Mensajes

(1 filas afectadas)

Creación de artículos en stock en el almacén P2 a partir de la tabla Artículos:

```
INSERT INTO STOCKS (REFERENCIA ART, ALMACEN, CNTD STK)
SELECT REFERENCIA ART,'P2',0 FROM ARTICULOS;
```

Mensajes

(6 filas afectadas)

Ejecución de un procedimiento que recupera el esquema y los nombres de las tablas de la base de datos actual:

execute dbo.nombres\_tablas

100 %

Resultados Mensajes

	TABLE_SCHEMA	TABLE_NAME
1	dbo	ARTICULOS
2	dbo	CLIENTES
3	dbo	HISTO_FAC
4	dbo	PEDIDOS
5	dbo	CATEGORIAS
6	dbo	LINEAS_PDO

Aplicación de este procedimiento para valorizar una tabla temporal con las tablas de la base de datos:

```
CREATE TABLE #TBN(elEsquema sysname, elNombre sysname);
go
INSERT INTO #TBN execute dbo.nombres_tablas
```

100 %

Mensajes

(6 filas afectadas)

*Insertión de registros, forzando el valor para una columna de tipo Identity:*

```
USE GESCOM;
GO
insert into CATEGORIAS (CODIGO_CAT, ETIQUETA_CAT) values (60, 'Webcams');
set identity_insert CATEGORIAS on
insert into CATEGORIAS (CODIGO_CAT, ETIQUETA_CAT) values (60, 'Webcams');
set identity_insert CATEGORIAS off
```

Mensajes

Mens. 544, Nivel 16, Estado 1, Línea 1  
No se puede insertar un valor explícito en la columna de identidad de la tabla 'CATEGORIAS' cuando IDENTITY\_INSERT es OFF.  
(2 filas afectadas)

Cuando se asocia la instrucción INSERT a una consulta SELECT, es posible limitar el número de registros que se insertan usando la cláusula TOP.

```
CREATE TABLE #TBN(e1Esquema sysname, e1Nombre sysname);
go
INSERT INTO #tbn
SELECT TOP(5) TABLE_SCHEMA, TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES;
```

Mensajes

(5 filas afectadas)

La instrucción INSERT permite insertar varios registros de datos de manera simultánea, cuando se asocia a una cláusula SELECT. También es posible añadir varios registros indicando directamente los valores de los datos que se desean insertar. Para esto, los datos relativos a cada registro que se va a insertar, se especifican entre paréntesis detrás de la cláusula VALUES y cada conjunto de valores se separa por una coma.

## Sintaxis

```
INSERT INTO nombreTabla[(col, ...)]
VALUES((valorRegistro1, ...), (valorRegistro2, ...), ...);
```

nombreTabla

Nombre de la tabla implicada en la operación de adición de registros.

(col, ...)

Nombre de las diferentes columnas de la tabla para las que se dan los valores. El orden de las columnas que se define aquí, también define el orden en que se suministran los valores.

(valorRegistro1,...)

Valores de los datos que se desean añadir en la tabla. Cada conjunto de valores debe suministrar un dato para cada una de las columnas que se especifican en la lista de columnas.

### Ejemplo

En el siguiente ejemplo se insertan dos valores en la tabla de categorías, con una única instrucción INSERT:



## 2. Modificación de registros

La instrucción UPDATE permite modificar los valores de las columnas de registros existentes. Esta instrucción puede actualizar varias columnas de varios registros de una tabla, a partir de expresiones o valores de otras tablas.

### Sintaxis

```
UPDATE nombreObjeto SET col=exp[,...]
[FROM nombreObjeto[,...]] [WHERE condicion]
```

nombreObjeto

Nombre de tabla o vista.

col

Nombre de la columna que se va a actualizar.

exp

Cualquier tipo de expresión que devuelva un solo valor del mismo tipo que la columna. Están permitidas las palabras clave DEFAULT y NULL.



FROM nombreObjeto

Permite actualizar cada registro a partir de los datos de otras tablas o vistas.

condicion

Expresión booleana que permite limitar los registros que se van a actualizar.

### Ejemplos

*Modificación de los datos NOMBRE, DIRECCION y CIUDAD del Cliente 25:*



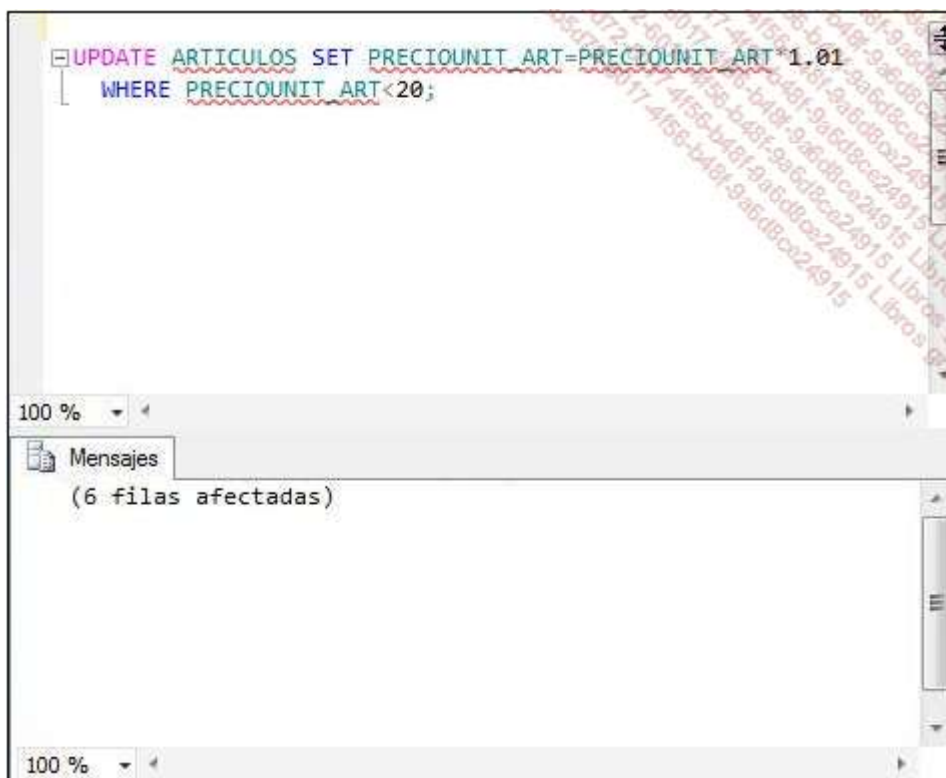
The screenshot shows a SQL query window with the following text:

```
UPDATE CLIENTES SET nombre=DEFAULT, direccion=NULL, ciudad='Madrid'
WHERE numero=25;
SELECT * FROM CLIENTES WHERE numero=25;
```

Below the query window, the 'Resultados' (Results) tab is active, displaying a table with the following data:

numero	nombre	apellidos	direccion	codigopostal	ciudad	telefono	CODIGOREP
25	Ángel	Sánchez	NULL	28290	Madrid	913572199	CD

*Aumenta el 1 % los precios de todos los artículos que cuestan menos de 20 €:*



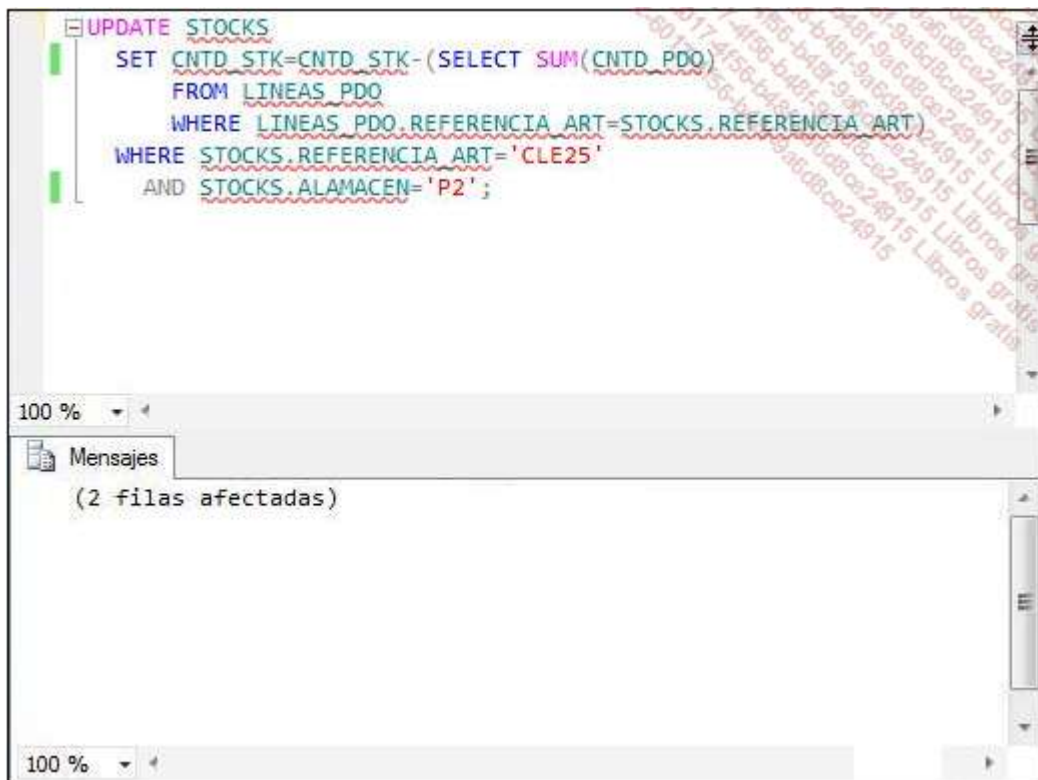
The screenshot shows a SQL query window with the following text:

```
UPDATE ARTICULOS SET PRECIOUNIT ART=PRECIOUNIT ART*1.01
WHERE PRECIOUNIT ART<20;
```

Below the query window, the 'Mensajes' (Messages) pane is active, displaying the message:

(6 filas afectadas)

*Actualización de la cantidad en stock del artículo con referencia CLE25 en el almacén P2, respecto a todas las líneas de pedido de este artículo:*



### 3. Eliminar registros

La instrucción DELETE permite eliminar uno o varios registros de una tabla o vista, en función de una condición o de los datos de otra tabla.

#### Sintaxis

```
DELETE [FROM] nombreObjeto [FROM nombreObjeto [,...]] [WHERE
condicion]
```

nombreObjeto

Nombre de tabla o vista.

FROM nombreObjeto

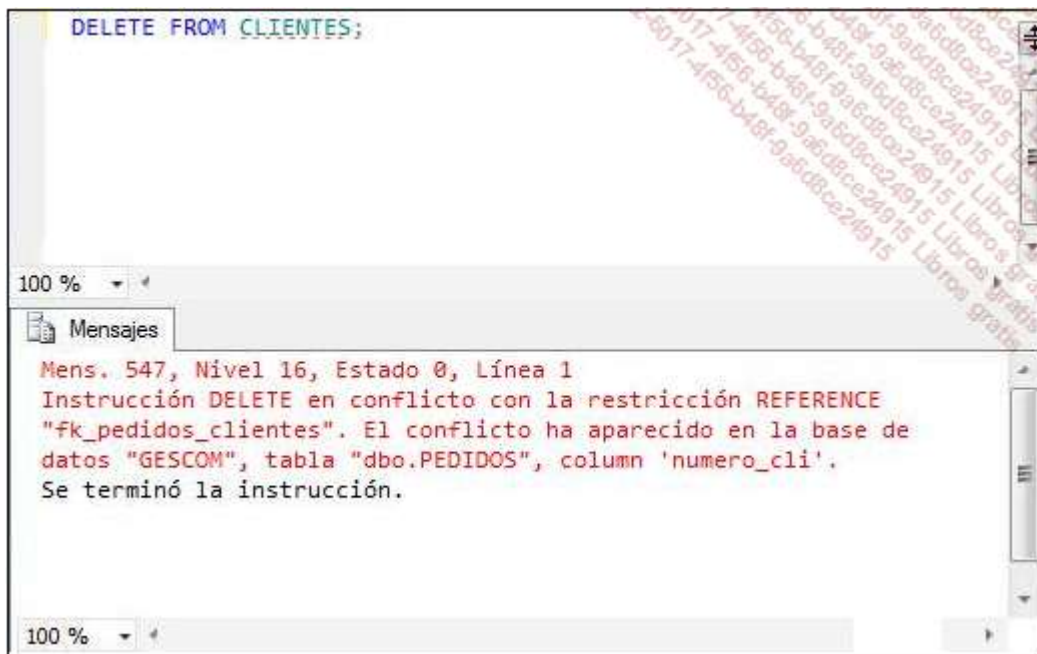
Permite utilizar las columnas de otras tablas o vistas en la cláusula WHERE.

condicion

Expresión booleana que permite restringir los registros que se deben eliminar.

#### Ejemplos

*El intento de eliminar todos los clientes provoca un fallo, ya que al menos uno ha realizado un pedido. La restricción de integridad referencial que existe entre las dos tablas, bloquea la eliminación del cliente en cuestión. La instrucción DELETE, como el resto de instrucciones SQL, se ejecuta correctamente en su totalidad o se cancela completamente. Esto último es lo que sucede en el caso actual.*

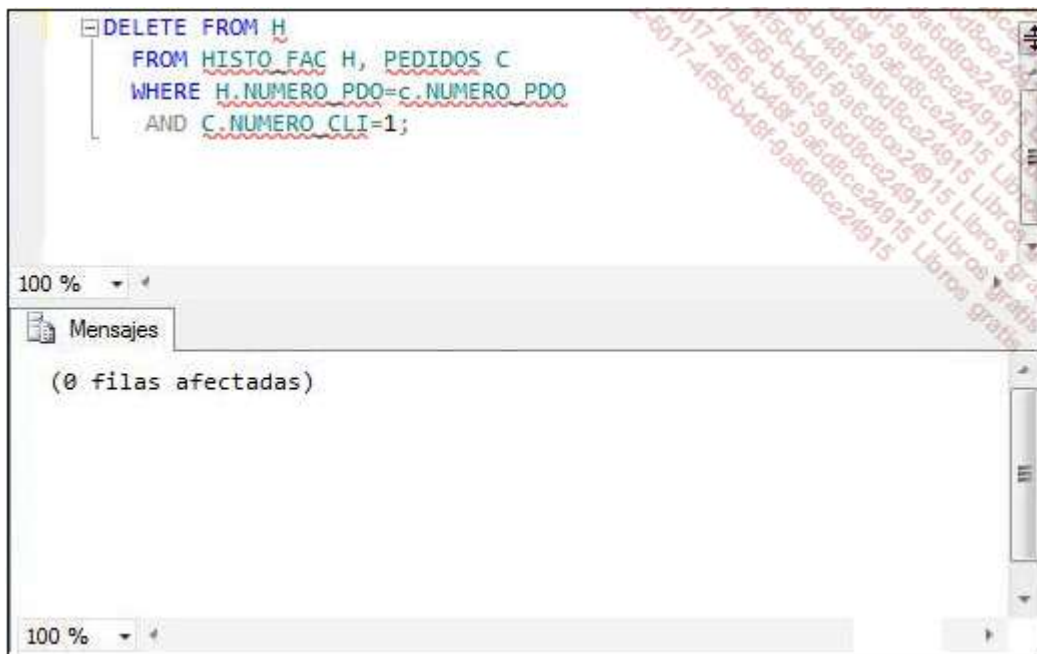


La opción ON DELETE, cuando se crea la restricción referencial, permite evitar este problema.

*Eliminar las facturas cerradas:*



*Eliminar del histórico del cliente 1:*



La variable @@ROWCOUNT contiene el número de registros que ha eliminado la última instrucción DELETE.

La instrucción DELETE también ofrece la cláusula TOP para eliminar solo los n primeros registros (TOP n) del juego de datos. El uso de la cláusula TOP en la instrucción DELETE, funciona igual que cuando se usa con la instrucción SELECT. Por este motivo, la cláusula TOP se detalla a nivel de la instrucción SELECT, más adelante en este capítulo.

La cláusula OUTPUT, que permite conocer los registros eliminados, existe para la instrucción DELETE como para las otras instrucciones SQL DML. Esta cláusula se detalla en el capítulo Transact SQL: el lenguaje procedimental, ya que el tratamiento del resultado llama normalmente al código procedimental.

La instrucción DELETE es una instrucción que se traza, por lo que los datos a los que afecta se guardan en el archivo de trazas. Cuando se eliminan muchos registros, esta traza puede consumir mucho tiempo. De esta manera, cuando es necesario eliminar todos los registros de una tabla, es mejor utilizar la instrucción TRUNCATE TABLE.

## Sintaxis

```
TRUNCATE TABLE nombreTabla;
```

### Ejemplo

Se usa la instrucción TRUNCATE TABLE para eliminar todos los datos de la tabla HISTO\_FAC.



## 4. Extracción de registros

La instrucción SELECT permite visualizar los datos que se almacenan en las bases de datos, hacer cálculos o transformaciones sobre estos datos o valorizar las tablas a partir de otras tablas.

Esta instrucción tiene una sintaxis compleja, que podemos estudiar en tres partes:

- Sintaxis básica, que permite ver y ejecutar las operaciones del álgebra relacional, como las restricciones, proyecciones, productos cartesianos, joins, cálculos sencillos, cálculos agregados y uniones.
- Sintaxis INTO, que permite crear una nueva tabla a partir del resultado de la SELECT.
- Cláusula COMPUTE, que permite generar registros que contienen estadísticas. Esta cláusula no es relacional.

### Sintaxis basica

```
SELECT [ALL|DISTINCT]{*|listaExpresiones}
FROM listaObjetos
[WHERE condición]
[GROUP BY listaExpresiones]
[HAVING condición]
[ORDER BY listaExpresiones]
[UNION[ALL]SELECT...]
```

ALL

Permite la extracción de todos los registros (opción por defecto).

DISTINCT

No muestra las repeticiones, es decir los registros idénticos.

\*

Extrae todas las columnas.

listaExpresiones

Lista formada por nombres de columnas, constantes, funciones, expresiones calculadas o cualquier combinación de expresiones separadas por comas. Cada expresión se podrá completar con un título de columna con el formato: TITULO = exp o exp ALIAS\_COL, que permite modificar el título por defecto del resultado que es nulo o el nombre de la columna de la tabla.

listaObjetos

Lista de tablas o vistas separadas por comas, a partir de las que se extraen los datos. Cada nombre de objeto se podrá acompañar de un nombre de alias que permite hacer referencia a la tabla o vista con otro nombre. Además, se puede orientar el funcionamiento interno de la consulta, colocando directivas de consulta para optimizarla, entre paréntesis, detrás de cada nombre de objeto.

## 5. Operaciones del álgebra relacional

### a. Selección de columnas

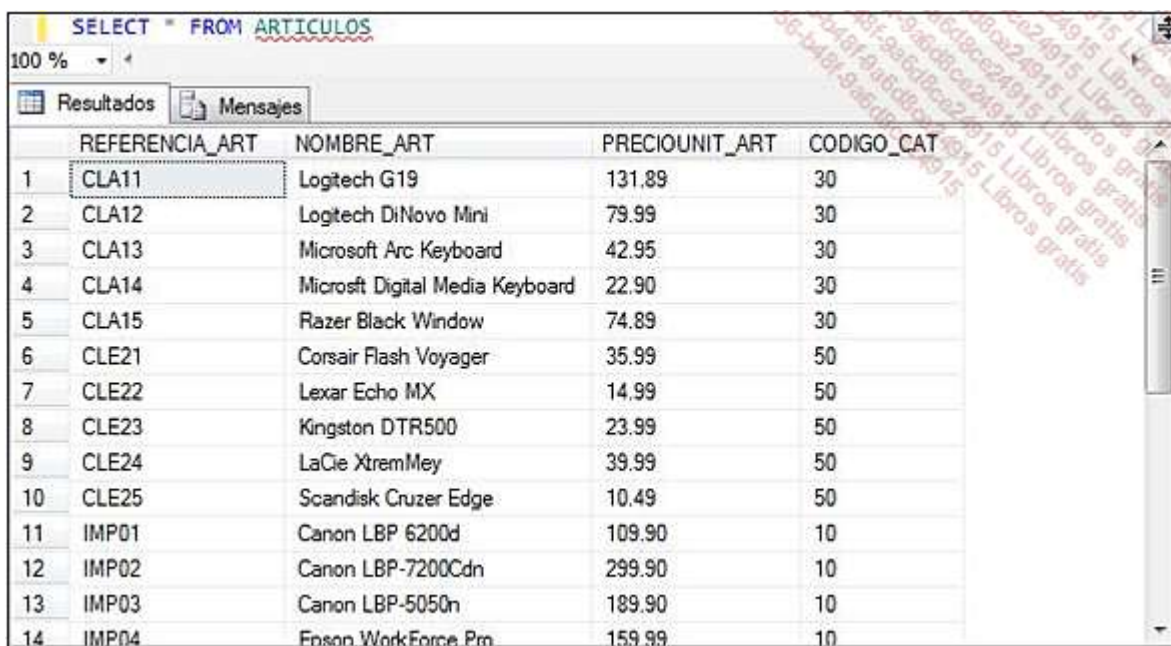
La palabra clave SELECT permite introducir una lista de columnas que se desea extraer de las tablas en el orden elegido o todas las columnas en el orden de creación con \* (o nombreTabla.\*).

En caso de ambigüedad con los nombres de las columnas, podemos utilizar la forma: nombreTabla.nombrecol.

La columna resultante tendrá el mismo nombre que la columna inicial, salvo si usamos un título o un alias de columna.

#### Ejemplos

Ver todas las columnas de todos los registros de la tabla Artículos:



	REFERENCIA_ART	NOMBRE_ART	PRECIOUNIT_ART	CODIGO_CAT
1	CLA11	Logitech G19	131.89	30
2	CLA12	Logitech DiNovo Mini	79.99	30
3	CLA13	Microsoft Arc Keyboard	42.95	30
4	CLA14	Microsoft Digital Media Keyboard	22.90	30
5	CLA15	Razer Black Window	74.89	30
6	CLE21	Corsair Flash Voyager	35.99	50
7	CLE22	Lexar Echo MX	14.99	50
8	CLE23	Kingston DTR500	23.99	50
9	CLE24	LaCie XtremMey	39.99	50
10	CLE25	Scandisk Cruzer Edge	10.49	50
11	IMP01	Canon LBP 6200d	109.90	10
12	IMP02	Canon LBP-7200Cdn	299.90	10
13	IMP03	Canon LBP-5050n	189.90	10
14	IMP04	Epson WorkForce Pro	159.99	10

En el siguiente ejemplo, no todas las columnas participan en la selección, sino solo aquellas a cuyo nombre se hace referencia en la palabra clave SELECT.



SELECT CODIGO\_CAT, REFERENCIA\_ART FROM ARTICULOS;

100 %

Resultados Mensajes

	CODIGO_CAT	REFERENCIA_ART
1	30	CLA11
2	30	CLA12
3	30	CLA13
4	30	CLA14
5	30	CLA15
6	50	CLE21
7	50	CLE22
8	50	CLE23
9	50	CLE24
10	50	CLE25
11	10	IMP01
12	10	IMP02
13	10	IMP03

En este ejemplo, se da un nuevo nombre a las columnas para que el resultado se pueda interpretar más fácilmente. Observe que si queremos incluir caracteres especiales (espacio, apóstrofe...) el nombre de la columna se debe escribir entre delimitadores de cadenas de caracteres de SQL Server.

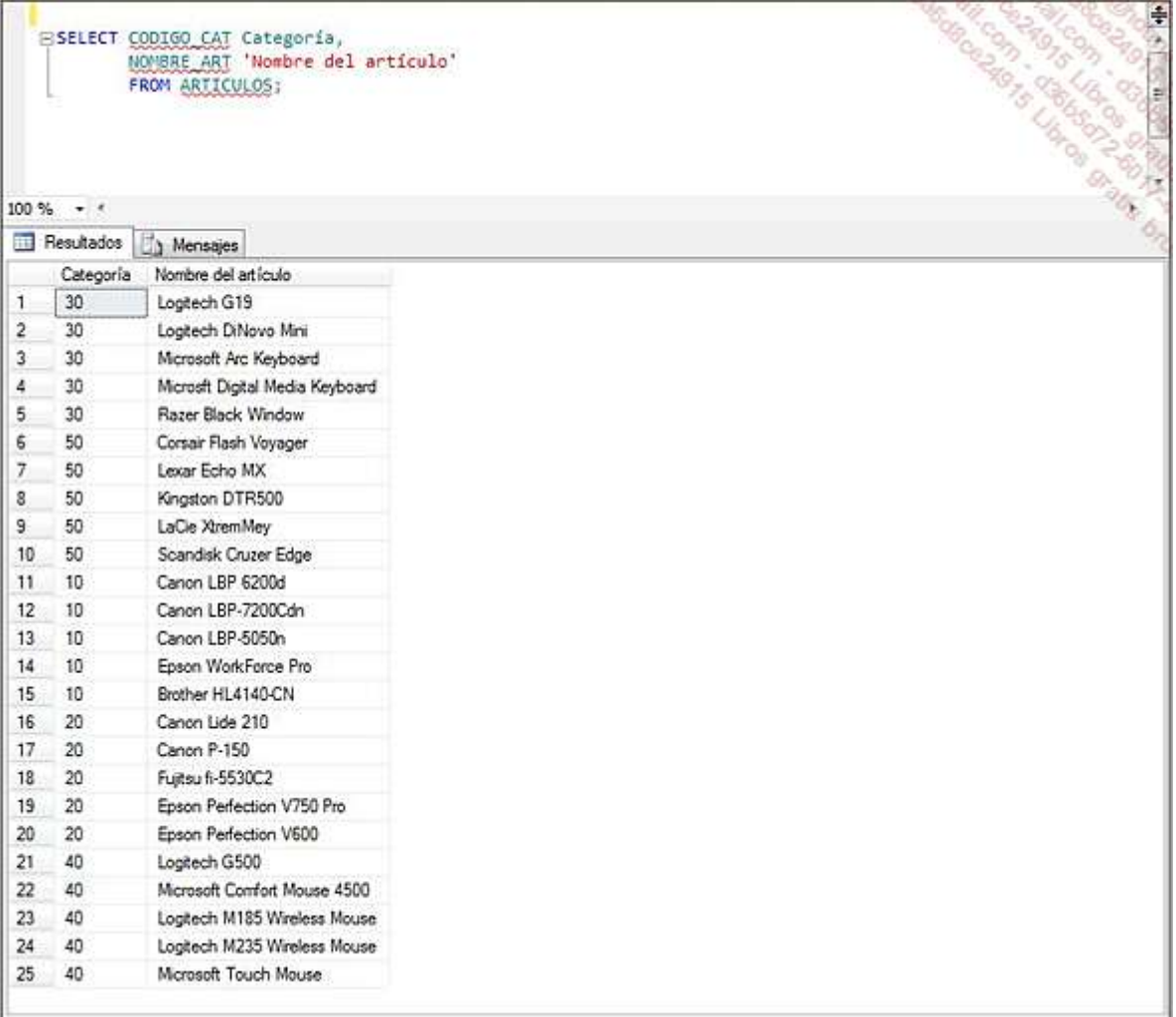
SELECT Categoría=CODIGO\_CAT,  
'Nombre del artículo'=NOMBRE\_ART  
FROM ARTICULOS;

100 %

Resultados Mensajes

	Categoría	Nombre del artículo
1	30	Logitech G19
2	30	Logitech DiNovo Mini
3	30	Microsoft Arc Keyboard
4	30	Microsoft Digital Media Keyboard
5	30	Razer Black Window
6	50	Corsair Flash Voyager
7	50	Lexar Echo MX
8	50	Kingston DTR500
9	50	LaCie XtremMey
10	50	Scandisk Cruzer Edge
11	10	Canon LBP 6200d
12	10	Canon LBP-7200Cdn
13	10	Canon LBP-5050n
14	10	Epson WorkForce Pro
15	10	Brother HL4140-CN
16	20	Canon Lide 210
17	20	Canon P-150
18	20	Fujitsu fi-5530C2
19	20	Epson Perfection V750 Pro
20	20	Epson Perfection V600
21	40	Logitech G500
22	40	Microsoft Comfort Mouse 4500
23	40	Logitech M185 Wireless Mouse
24	40	Logitech M235 Wireless Mouse
25	40	Microsoft Touch Mouse

El nuevo nombre de la columna se puede ver como un alias de columna. En este caso es necesario ponerlo detrás



```
SELECT CODIGO_CAT Categoría,  
NOMBRE_ART 'Nombre del artículo'  
FROM ARTICULOS;
```

	Categoría	Nombre del artículo
1	30	Logitech G19
2	30	Logitech DiNovo Mini
3	30	Microsoft Arc Keyboard
4	30	Microsoft Digital Media Keyboard
5	30	Razer Black Window
6	50	Corsair Flash Voyager
7	50	Lexar Echo MX
8	50	Kingston DTR500
9	50	LaCie XtremMey
10	50	Scandisk Cruzer Edge
11	10	Canon LBP 6200d
12	10	Canon LBP-7200Cdn
13	10	Canon LBP-5050n
14	10	Epson WorkForce Pro
15	10	Brother HL4140-CN
16	20	Canon Lide 210
17	20	Canon P-150
18	20	Fujitsu fi-5530C2
19	20	Epson Perfection V750 Pro
20	20	Epson Perfection V600
21	40	Logitech G500
22	40	Microsoft Comfort Mouse 4500
23	40	Logitech M185 Wireless Mouse
24	40	Logitech M235 Wireless Mouse
25	40	Microsoft Touch Mouse

de la columna.

También es posible renombrar una columna usando la palabra clave **AS** entre el nombre de la columna y su alias en la cláusula SELECT. El resultado será el mismo que el del ejemplo anterior.

## b. Restricción

La restricción consiste en extraer un determinado número de registros que respondan a una o varias condiciones. La cláusula WHERE permite implementar las restricciones. Las condiciones son expresiones booleanas formadas por el nombre de columnas, constantes, funciones, operadores de comparación y operadores lógicos.

### Ejemplos

*Clientes de Barcelona:*



SELECT numero, nombre, apellidos  
FROM CLIENTES  
WHERE ciudad='Barcelona';

100 %

Resultados Mensajes

	numero	nombre	apellidos
1	18	Ramis	Sánchez
2	19	Sandra	Fernández
3	20	Teddy	Trente
4	21	Urbain	Ugnace
5	22	Valeria	Zavala

*Cientes de Madrid:*

SELECT numero, nombre, apellidos, ciudad  
FROM CLIENTES  
WHERE CODIGOPOSTAL BETWEEN 28000 AND 28999;

100 %

Resultados Mensajes

	numero	nombre	apellidos	ciudad
1	14	Naomi	Zavala	MADRID

Esta instrucción funciona aunque la columna CODIGOPOSTAL se defina de tipo carácter y el criterio de restricción use valores numéricos enteros. SQL Server hace la conversión de manera implícita.

*Cientes cuyo apellido empieza por "Sá" y vivan en cualquier ciudad que empiece por "S":*

SELECT numero, nombre, apellidos, ciudad  
FROM CLIENTES  
WHERE (ciudad like 'S%' or ciudad is null)  
and apellidos like 'Sá%'

100 %

Resultados Mensajes

	numero	nombre	apellidos	ciudad
1	0	Ángel	Sánchez	SANTANDER

En este ejemplo, la restricción se hace sobre las fechas para extraer los pedidos que se han hecho en el mes de enero, durante los 3 últimos años. Para expresar el criterio de manera sencilla, son necesarias funciones de manipulación de fechas.

SELECT NUMERO\_PDO, NUMERO\_CLI  
FROM PEDIDOS  
WHERE DATEPART(mm, FECHA\_PDO)=1  
AND DATEDIFF(year, GETDATE(), FECHA\_PDO)<3;

Resultados Mensajes

	NUMERO_PDO	NUMERO_CLI
1	1	1
2	2	1
3	4	3
4	5	4
5	6	5
6	7	6
7	8	7
8	9	8
9	10	9
10	11	1
11	12	2
12	13	3
13	14	4
14	15	5
15	16	6
16	17	7

### c. Cálculos sencillos

El uso de expresiones calculadas gracias a operadores aritméticos o funciones, permite obtener nuevas columnas con el resultado de estos cálculos ejecutado sobre cada registro resultante.

#### Ejemplos

*Simulación de un aumento del 10% de las tarifas:*

SELECT REFERENCIA\_ART,  
'Anterior precio'=PRECIOUNIT\_ART,  
'Nuevo precio'=PRECIOUNIT\_ART\*1.1  
FROM ARTICULOS;

Resultados Mensajes

	REFERENCIA_ART	Anterior precio	Nuevo precio
1	IMP05	269.90	296.890
2	IMP01	109.90	120.890
3	IMP03	189.90	208.890
4	IMP02	299.90	329.890
5	SCA06	83.90	92.290
6	SCA07	269.90	296.890
7	CLE21	35.99	39.589
8	SCA10	269.90	296.890

*Ver la concatenación del nombre y apellidos de los clientes y el número del departamento:*

3B5c.sql - ANGEL\_M...dministrador (63))\* X

```

SELECT RTRIM(nombre)+' '+RTRIM(apellidos) AS NombreCompleto,
SUBSTRING(codigoPostal,1,2) AS Area
FROM CLIENTES;

```

100 %

Resultados Mensajes

	NombreCompleto	Area
44	Pascal Osuma	46
45	Andrés Fernández	46
46	Ramis Sánchez	46
47	Sandra Fernández	46
48	Teddy Trente	46
49	Urbain Ugnace	46
50	Valeria Zavala	46
51	Wilfried Willy	41
52	Javier Alonso	41
53	Yann Yvres	41
54	Zoe Zazou	39

ANGEL\_MARIA1\SQLSERVER2012 ... | Angel\_Maria1\Administr... | GESCOM | 00:00:00 | 54 filas

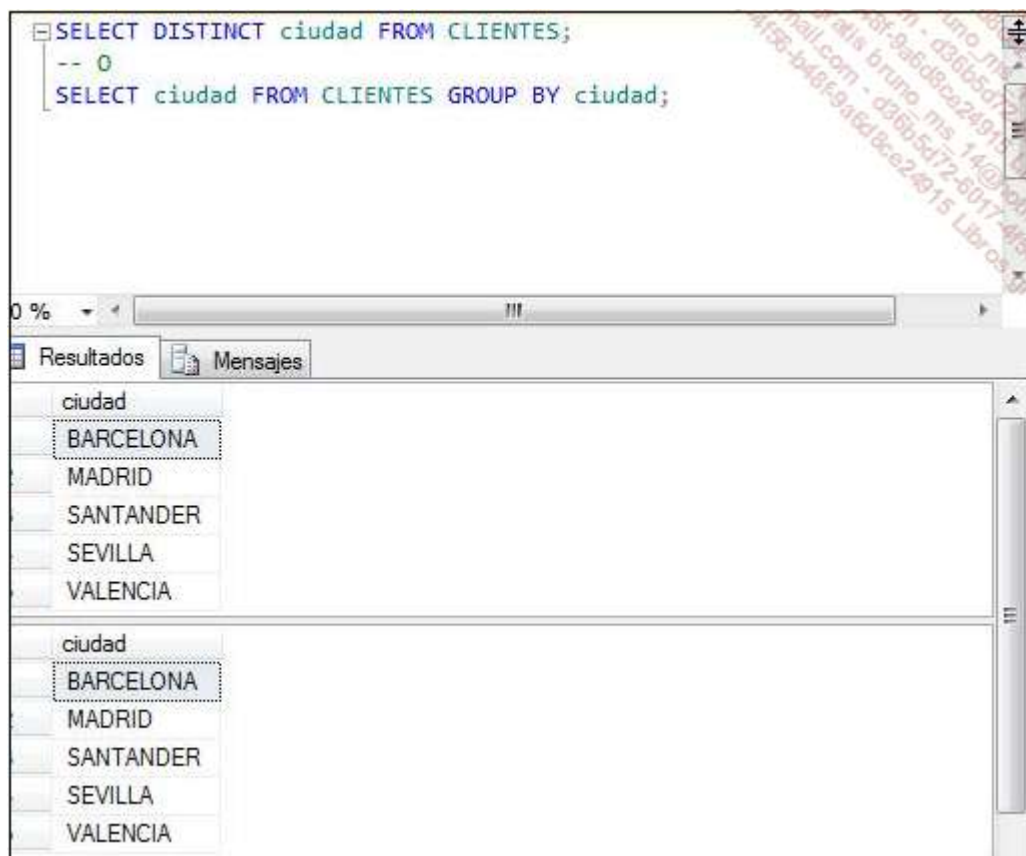
#### d. Proyección

La operación de proyección permite agrupar los registros respecto a columnas. El resultado de esta operación permitirá obtener registros únicos sobre una selección de columnas. La proyección se hace con la opción DISTINCT o la cláusula GROUP BY listaCol. La lista de columnas o de expresiones de agrupación debe ser idéntica a la lista de columnas seleccionadas.

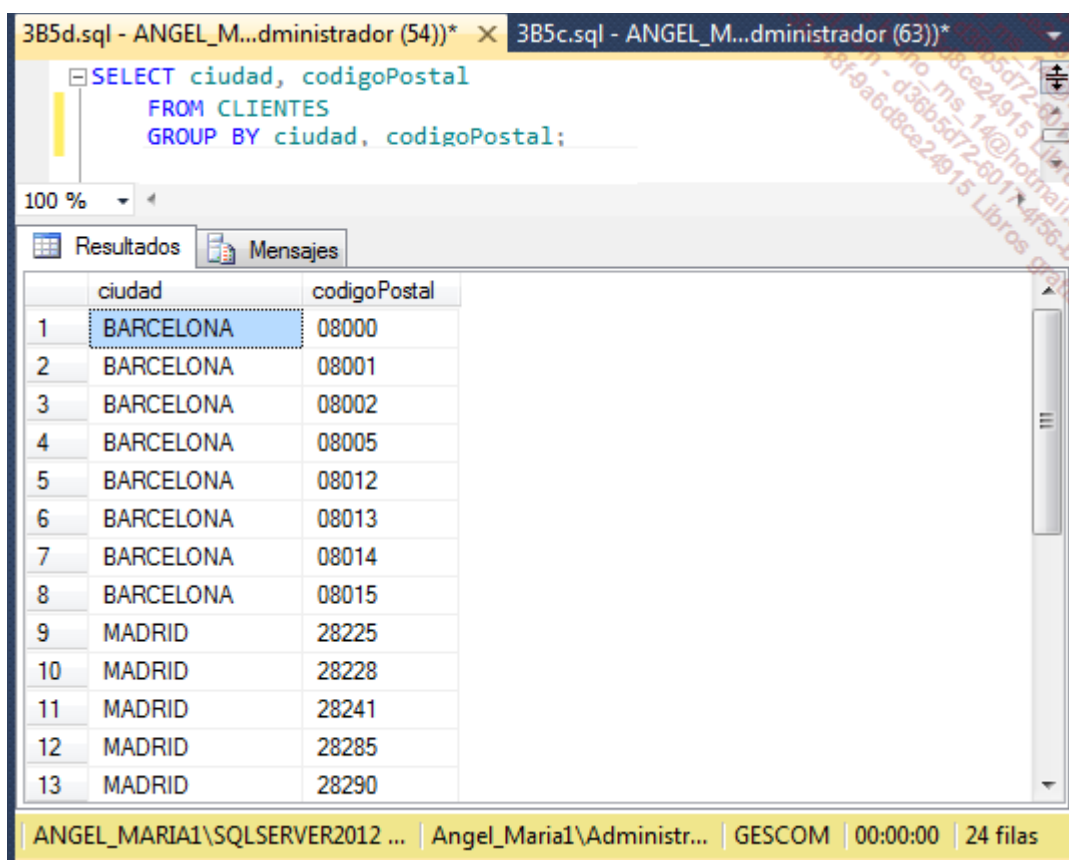
Incluso si estas dos instrucciones permiten obtener un resultado similar, el tratamiento realizado es totalmente diferente. Con la opción DISTINCT los datos se muestran en el orden de extracción de la base de datos y solo se muestran los datos distintos entre sí. En el caso de la cláusula GROUP BY, todos los datos se extraen de la base de datos y después se agrupan (ordenan), siguiendo un criterio de agrupación especificado. De esta manera, se forman subconjuntos y solo se muestran las etiquetas de estos subconjuntos.

#### Ejemplos

*Lista de las ciudades donde viven los clientes:*



Agrupación utilizando los criterios ciudad y código postal (para ver las ciudades que tienen varios códigos postales):



## e. Cálculos agregados

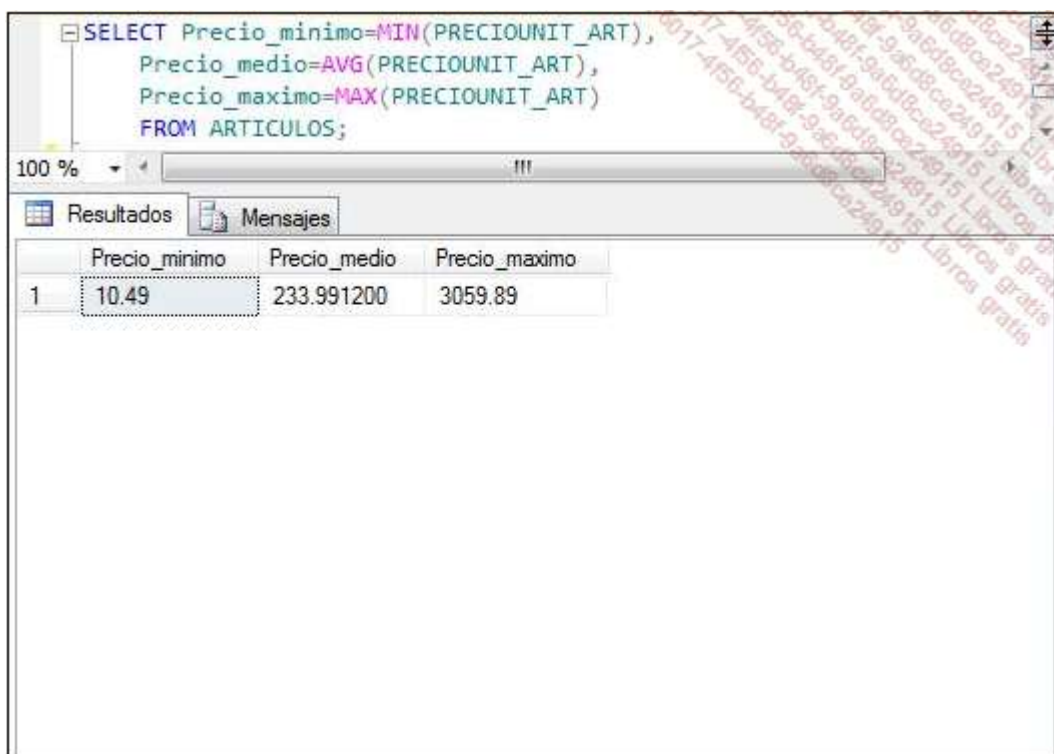
Los cálculos agregados permiten hacer un cálculo sobre un conjunto de registros. Este tipo de cálculo es posible con las funciones de cálculo agregado. Las más habituales son COUNT, SUM, AVG, MIN y MAX. Si el cálculo se hace sobre todos los registros, es suficiente con incluir la o las funciones de cálculo agregado en la cláusula SELECT.

Es frecuente que el cálculo se tenga que hacer para subconjuntos de valores particulares. En este caso, los subconjuntos se definen con la cláusula GROUP BY. Esta cláusula permite construir los subconjuntos antes de hacer el cálculo.

La cláusula WHERE permite establecer restricciones sobre los criterios sencillos y los cálculos sencillos. Para establecer las restricciones sobre los cálculos agregados, es necesario utilizar la cláusula HAVING.

### Ejemplos

*Precio mínimo, medio y máximo de los artículos:*



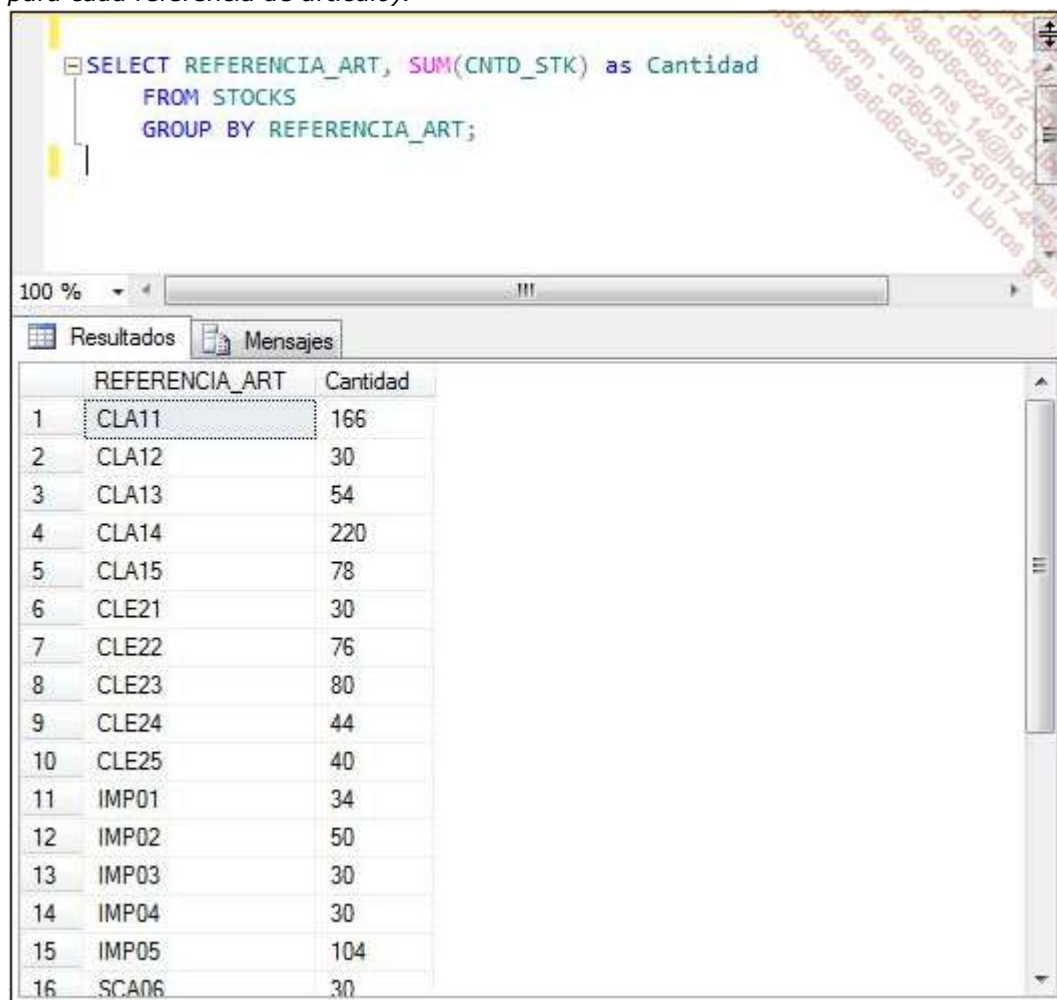
The screenshot shows a SQL query window with the following query:

```
SELECT Precio_minimo=MIN(PRECIOUNIT_ART),  
       Precio_medio=AVG(PRECIOUNIT_ART),  
       Precio_maximo=MAX(PRECIOUNIT_ART)  
FROM ARTICULOS;
```

Below the query, the results are displayed in a table with three columns: Precio\_minimo, Precio\_medio, and Precio\_maximo. The results show a single row with the values 10.49, 233.991200, and 3059.89 respectively.

	Precio_minimo	Precio_medio	Precio_maximo
1	10.49	233.991200	3059.89

Cantidad de artículos en stock, en todos los almacenes: el cálculo se hace para cada artículo (más exactamente, para cada referencia de artículo).



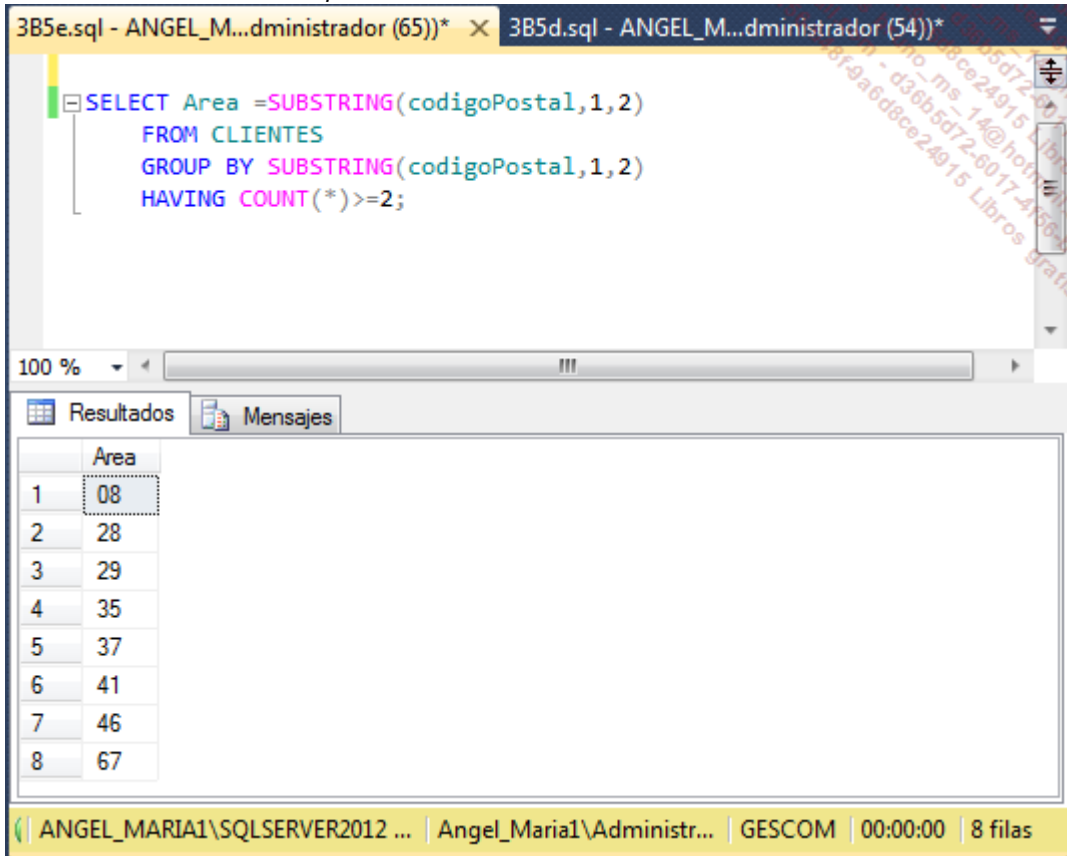
The screenshot shows a SQL query editor at the top with the following query:

```
SELECT REFERENCIA_ART, SUM(CNTD_STK) as Cantidad
FROM STOCKS
GROUP BY REFERENCIA_ART;
```

Below the query editor, the 'Resultados' (Results) tab is active, displaying a table with 16 rows. The first row is highlighted. The table has two columns: 'REFERENCIA\_ART' and 'Cantidad'.

	REFERENCIA_ART	Cantidad
1	CLA11	166
2	CLA12	30
3	CLA13	54
4	CLA14	220
5	CLA15	78
6	CLE21	30
7	CLE22	76
8	CLE23	80
9	CLE24	44
10	CLE25	40
11	IMP01	34
12	IMP02	50
13	IMP03	30
14	IMP04	30
15	IMP05	104
16	SCA06	30

Lista de las aerías que tienen al menos dos clientes:



The screenshot shows a SQL query editor with the following query:

```
SELECT Area =SUBSTRING(codigoPostal,1,2)
FROM CLIENTES
GROUP BY SUBSTRING(codigoPostal,1,2)
HAVING COUNT(*)>=2;
```

Below the query editor, the 'Resultados' (Results) tab is active, displaying a table with 8 rows and 1 column named 'Area'.

	Area
1	08
2	28
3	29
4	35
5	37
6	41
7	46
8	67

The status bar at the bottom indicates: ANGEL\_MARIA1\SQLSERVER2012 ... | Angel\_Maria1\Administr... | GESCOM | 00:00:00 | 8 filas

#### f. Producto cartesiano

##### Sintaxis

*Sintaxis actual:*

```
SELECT listaColumnas FROM listaTablas
```

*Sintaxis ANSI:*

```
SELECT listaColumnas FROM nombreTabla CROSS JOIN nombreTabla [ .... ]
```

El producto cartesiano permite extraer datos de varias tablas, asociando cada registro de cada tabla que se utiliza. Las tablas implicadas tienen que estar separadas por comas detrás del FROM. Si vamos a usar el mismo nombre de columna procedente de dos tablas diferentes, el nombre de estas columnas debe estar precedido por el nombre de tabla o el nombre de alias.

Esta operación se puede usar para simular la asociación de datos o para generar un gran número de registros (el número de registros resultante será el producto del número de registros de cada tabla).

##### Ejemplos

*Asociación de cada artículo con cada categoría:*



SELECT CAT.ETIQUETA\_CAT, ART.NOMBRE\_ART  
FROM CATEGORIAS CAT, ARTICULOS ART;

100 %

Resultados Mensajes

	ETIQUETA_CAT	NOMBRE_ART
1	Impresoras	Logitech G19
2	Impresoras	Logitech DiNovo Mini
3	Impresoras	Microsoft Arc Keyboard
4	Impresoras	Microsft Digital Media Keyboard
5	Impresoras	Razer Black Window
6	Impresoras	Corsair Flash Voyager
7	Impresoras	Lexar Echo MX
8	Impresoras	Kingston DTR500
9	Impresoras	LaCie XtremMey
10	Impresoras	Scandisk Cruzer Edge
11	Impresoras	Canon LBP 6200d
12	Impresoras	Canon LBP-7200Cdn
13	Impresoras	Canon LBP-5050n
14	Impresoras	Epson WorkForce Pro
15	Impresoras	Brother HL4140-CN
16	Impresoras	Canon Lide 210
17	Impresoras	Canon P-150
18	Impresoras	Fujitsu fi-5530C2
19	Impresoras	Epson Perfection V750 Pro

*La misma consulta en sintaxis ANSI:*



	ETIQUETA_CAT	NOMBRE_ART
1	Impresoras	Logitech G19
2	Impresoras	Logitech DiNovo Mini
3	Impresoras	Microsoft Arc Keyboard
4	Impresoras	Microsoft Digital Media Keyboard
5	Impresoras	Razer Black Window
6	Impresoras	Corsair Flash Voyager
7	Impresoras	Lexar Echo MX
8	Impresoras	Kingston DTR500
9	Impresoras	LaCie XtremMey
10	Impresoras	Scandisk Cruzer Edge
11	Impresoras	Canon LBP 6200d
12	Impresoras	Canon LBP-7200Cdn
13	Impresoras	Canon LBP-5050n
14	Impresoras	Epson WorkForce Pro
15	Impresoras	Brother HL4140-CN
16	Impresoras	Canon Lide 210
17	Impresoras	Canon P-150
18	Impresoras	Fujitsu fi-5530C2
19	Impresoras	Epson Perfection V750 Pro

## g. Join

El join es la combinación de un producto cartesiano y una restricción. Permite asociar de manera lógica los registros de diferentes tablas. Los joins se utilizan normalmente para establecer correspondencia entre los datos de un registro que tiene una clave extranjera, con los datos del registro que tiene la clave primaria (join natural).

### Sintaxis

*Sintaxis actual:*

```
SELECT listaColumnas FROM listaTablas WHERE
nombreTabla.nombreColumnas operador nombreTabla.nombreColumna [...]
```

*Sintaxis ANSI:*

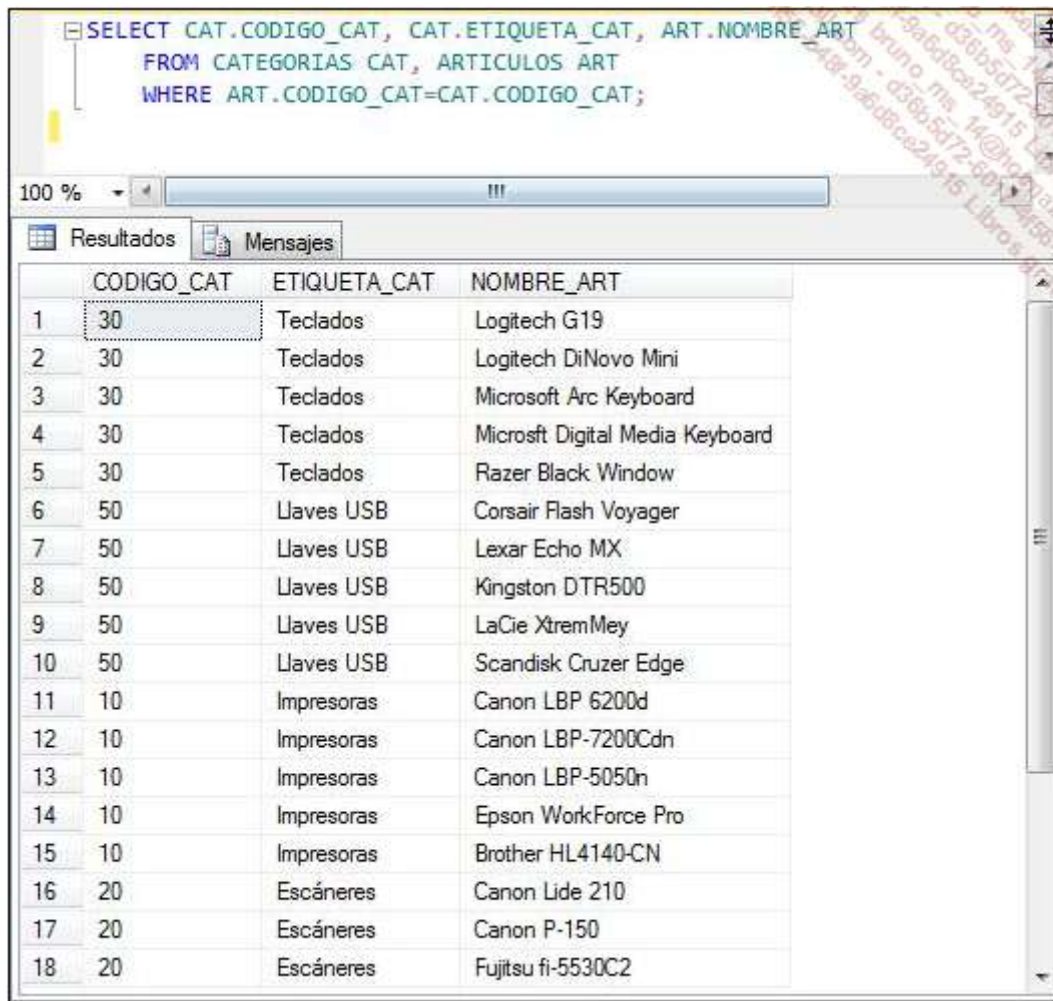
```
SELECT listaColumnas FROM nombreTabla INNER JOIN nombreTabla ON
nombreTabla.nombreColumna operador nombreTabla.nombreColumna [...]
```

La sintaxis ANSI es menos natural y no se utiliza tanto. Sin embargo, el hecho de que esta sintaxis esté normalizada, tiene la ventaja de no mezclar en la cláusula WHERE los criterios correspondientes a las restricciones y los correspondientes al join. Por lo tanto, la consulta que se escribe respetando la sintaxis normalizada será más fácil de construir y leer.

Independientemente de cuál sea su opción, el optimizador de consultas de SQL Server identificará el join y no hay diferencia respecto al tratamiento.

## Ejemplos

Visualización de los datos artículos y categorías:



```
SELECT CAT.CODIGO_CAT, CAT.ETIQUETA_CAT, ART.NOMBRE_ART
FROM CATEGORIAS CAT, ARTICULOS ART
WHERE ART.CODIGO_CAT=CAT.CODIGO_CAT;
```

	CODIGO_CAT	ETIQUETA_CAT	NOMBRE_ART
1	30	Teclados	Logitech G19
2	30	Teclados	Logitech DiNovo Mini
3	30	Teclados	Microsoft Arc Keyboard
4	30	Teclados	Microsoft Digital Media Keyboard
5	30	Teclados	Razer Black Window
6	50	Llaves USB	Corsair Flash Voyager
7	50	Llaves USB	Lexar Echo MX
8	50	Llaves USB	Kingston DTR500
9	50	Llaves USB	LaCie XtremMey
10	50	Llaves USB	Scandisk Cruiser Edge
11	10	Impresoras	Canon LBP 6200d
12	10	Impresoras	Canon LBP-7200Cdn
13	10	Impresoras	Canon LBP-5050n
14	10	Impresoras	Epson WorkForce Pro
15	10	Impresoras	Brother HL4140-CN
16	20	Escáneres	Canon Lide 210
17	20	Escáneres	Canon P-150
18	20	Escáneres	Fujitsu fi-5530C2

En el contexto de esta consulta de extracción, como se muestran todas las columnas de la tabla de CATEGORIAS, es posible utilizar el carácter \* precediendo al alias de la tabla (que se define aquí), es decir: ART.\*.

La misma consulta en sintaxis ANSI:

SELECT CAT.CODIGO\_CAT, CAT.ETIQUETA\_CAT, ART.NOMBRE\_ART  
FROM CATEGORÍAS CAT INNER JOIN ARTICULOS ART  
ON ART.CODIGO\_CAT=CAT.CODIGO\_CAT;

100 %

Resultados Mensajes

	CODIGO_CAT	ETIQUETA_CAT	NOMBRE_ART
1	30	Teclados	Logitech G19
2	30	Teclados	Logitech DiNovo Mini
3	30	Teclados	Microsoft Arc Keyboard
4	30	Teclados	Microsoft Digital Media Keyboard
5	30	Teclados	Razer Black Window
6	50	Llaves USB	Corsair Flash Voyager
7	50	Llaves USB	Lexar Echo MX
8	50	Llaves USB	Kingston DTR500
9	50	Llaves USB	LaCie XtremMey
10	50	Llaves USB	Scandisk Cruzer Edge
11	10	Impresoras	Canon LBP 6200d
12	10	Impresoras	Canon LBP-7200Cdn
13	10	Impresoras	Canon LBP-5050n
14	10	Impresoras	Epson WorkForce Pro
15	10	Impresoras	Brother HL4140-CN
16	20	Escáneres	Canon Lide 210
17	20	Escáneres	Canon P-150
18	20	Escáneres	Fujitsu fi-5530C2

Visualización del pedido:

```

SELECT CDE.NUMERO_PDO,
  CONVERT(char(10), FECHA_PDO,103) as FechaPdo,
  SUBSTRING(nombre,1,10) Nombre,
  ART.REFERENCIA_ART, PRECIOUNIT_ART, CNTD_PDO
FROM CLIENTES CLI INNER JOIN PEDIDOS CDE
  ON CLI.numero=CDE.NUMERO_CLI
INNER JOIN LINEAS_PDO LIG
  ON CDE.NUMERO_PDO=LIG.NUMERO_PDO
INNER JOIN ARTICULOS ART
  ON LIG.REFERENCIA_ART=ART.REFERENCIA_ART;

```

	NUMERO_PDO	FechaPdo	Nombre	REFERENCIA_ART	PRECIOUNIT_ART	CN
1	1	01/01/2012	Juán	SOU19	18.90	6
2	2	01/01/2012	Juán	IMP02	299.90	2
3	2	01/01/2012	Juán	IMP02	299.90	7
4	4	01/01/2012	Paco	SOU16	28.99	8
5	4	01/01/2012	Paco	IMP02	299.90	1
6	4	01/01/2012	Paco	CLE25	10.49	5
7	4	01/01/2012	Paco	IMP02	299.90	2
8	4	01/01/2012	Paco	SOU19	18.90	5
9	4	01/01/2012	Paco	IMP05	269.90	3
10	4	01/01/2012	Paco	CLA12	79.99	1
11	4	01/01/2012	Paco	SCA10	269.90	3
12	5	01/01/2012	David	CLA14	22.99	0

## h. Join externo

Cuando la condición no se cumple, no aparece ningún registro en el resultado. Los joins externos permiten extraer registros de una de las dos tablas implicadas, aunque la condición sea falsa. En este caso, los datos de la segunda tabla tienen valor NULL.

La sintaxis que se utiliza en la condición es:

```

nombreTabla1 LEFT OUTER JOIN nombreTabla2 ON nombreTabla1.col1=
nombreTabla2.col2

```

o

```

nombreTabla1 RIGHT OUTER JOIN nombreTabla2 ON nombreTabla1.col1=
nombreTabla2.col2

```

o

```

nombreTabla1 FULL OUTER JOIN nombreTabla2 ON nombreTabla1.col1=nombreTabla2.col2

```

dependiendo de si queremos ver los registros de la primera (LEFT) o de la segunda tabla (RIGHT).

El join externo completo (FULL OUTER JOIN) permite mostrar los datos de las dos tablas, aunque no se pueda establecer ninguna correspondencia.

## Sintaxis

```

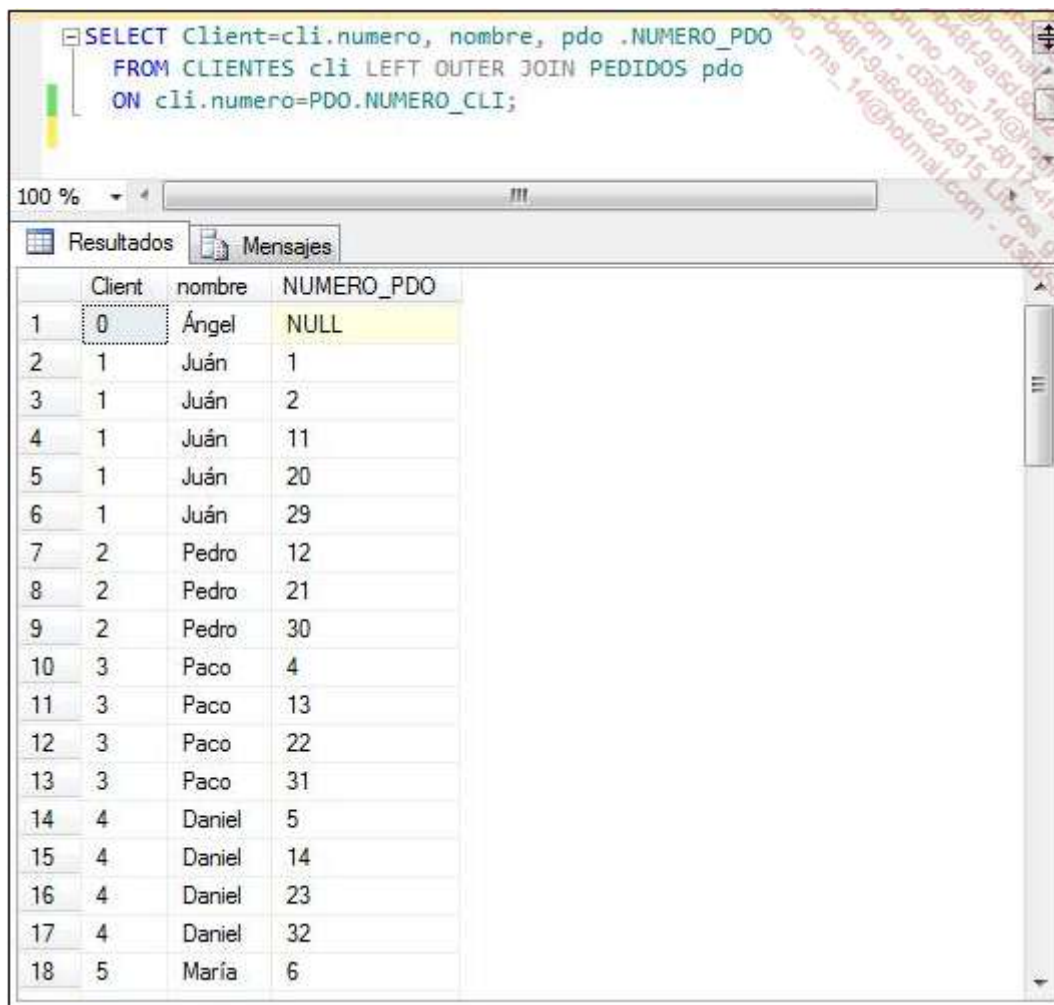
SELECT listaColumnas FROM nombreTabla {LEFT|RIGHT|FULL}OUTER JOIN
nombreTabla ON nombreTabla.nombreColumna operador nombreTabla.nombreColumna [...]

```



### Ejemplo

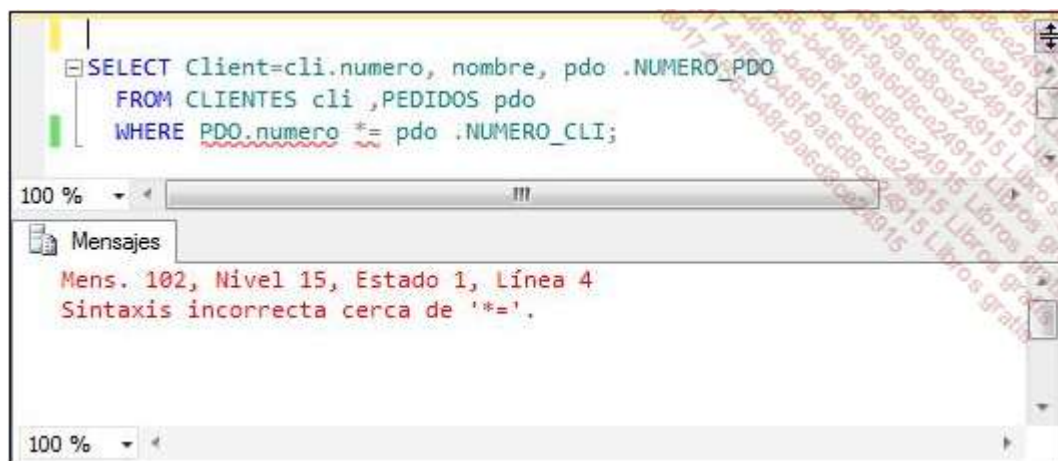
Lista de pedidos por cliente. Los clientes que no tengan pedidos también aparecen:



```
SELECT Client=cli.numero, nombre, pdo .NUMERO_PDO
FROM CLIENTES cli LEFT OUTER JOIN PEDIDOS pdo
ON cli.numero=PDO.NUMERO_CLI;
```

	Client	nombre	NUMERO_PDO
1	0	Ángel	NULL
2	1	Juán	1
3	1	Juán	2
4	1	Juán	11
5	1	Juán	20
6	1	Juán	29
7	2	Pedro	12
8	2	Pedro	21
9	2	Pedro	30
10	3	Paco	4
11	3	Paco	13
12	3	Paco	22
13	3	Paco	31
14	4	Daniel	5
15	4	Daniel	14
16	4	Daniel	23
17	4	Daniel	32
18	5	María	6

La sintaxis SQL Server \*= y =\* para definir los joins externos, ya no se soporta desde SQL Server 2008, como se muestra en el siguiente ejemplo:



Por lo tanto, se debe usar obligatoriamente la sintaxis normalizada para los joins externos. Es mejor adoptar siempre la sintaxis normalizada para expresar los joins.

### **Auto-join**

Es posible asociar registros de una tabla a otros registros de la misma tabla, resultando un auto-join. El uso de alias de tabla es obligatorio en este caso, para evitar ambigüedades en la sintaxis.

Un auto-join tiene lugar entre dos tablas cuando la condición del join es la igualdad y se aplica sobre dos columnas que tienen el mismo nombre.

## i. Order By

La cláusula ORDER BY permite ordenar los datos resultado de una consulta de tipo SELECT. Permite indicar las columnas que queremos utilizar para ordenar los datos. Para cada criterio de ordenación, hay que indicar si usamos el orden ascendente (por defecto) o descendente.

Las columnas se definen en la cláusula SELECT por su nombre o su número de orden.

Manipulando la columna usando su número de orden, es posible ordenar columnas que presentan el resultado de un cálculo sencillo o agregado.

## Sintaxis

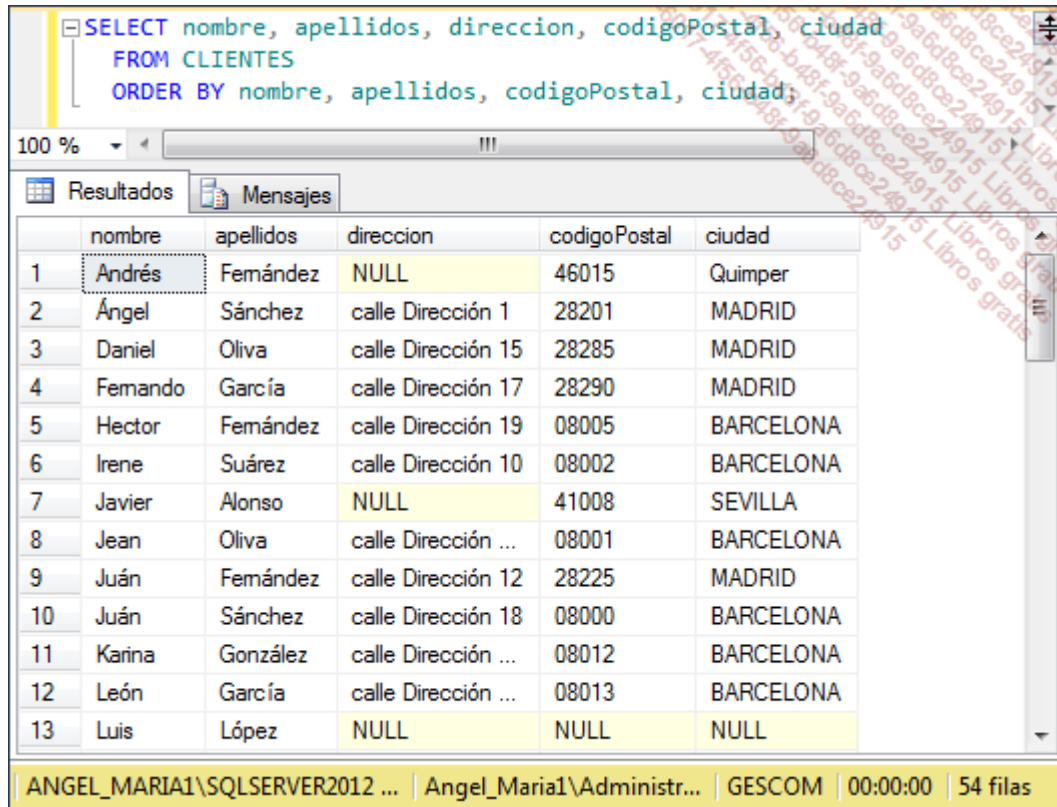
```
ORDER BY columna1[, columna2, ... ] [ASC|DESC]
```

ASC|DESC

Permite indicar si los datos tienen que estar ordenados de manera ascendente (ASC) o descendente. En esta ordenación de los datos, el valor null se considera como el valor más pequeño posible.

## Ejemplo

Se muestran los clientes en orden alfabético ascendente, usando su nombre, apellidos, códigos postales y ciudades.



The screenshot shows a SQL query editor with the following query:

```
SELECT nombre, apellidos, direccion, codigoPostal, ciudad
FROM CLIENTES
ORDER BY nombre, apellidos, codigoPostal, ciudad;
```

Below the query, the results are displayed in a table with 5 columns: nombre, apellidos, direccion, codigoPostal, and ciudad. The results are ordered alphabetically by name and address. The table has 13 rows of data.

	nombre	apellidos	direccion	codigoPostal	ciudad
1	Andrés	Fernández	NULL	46015	Quimper
2	Ángel	Sánchez	calle Dirección 1	28201	MADRID
3	Daniel	Oliva	calle Dirección 15	28285	MADRID
4	Fernando	García	calle Dirección 17	28290	MADRID
5	Hector	Fernández	calle Dirección 19	08005	BARCELONA
6	Irene	Suárez	calle Dirección 10	08002	BARCELONA
7	Javier	Alonso	NULL	41008	SEVILLA
8	Jean	Oliva	calle Dirección ...	08001	BARCELONA
9	Juán	Fernández	calle Dirección 12	28225	MADRID
10	Juán	Sánchez	calle Dirección 18	08000	BARCELONA
11	Karina	González	calle Dirección ...	08012	BARCELONA
12	León	García	calle Dirección ...	08013	BARCELONA
13	Luis	López	NULL	NULL	NULL

At the bottom of the screenshot, the status bar shows: ANGEL\_MARIA1\SQLSERVER2012 ... | Angel\_Maria1\Administr... | GESCOM | 00:00:00 | 54 filas

Aunque es posible indicar el número de las columnas en lugar de su nombre en la cláusula ORDER BY, no es recomendable. La lectura de la consulta es más complicada y un cambio en el orden de las columnas a nivel de la cláusula SELECT, tiene un impacto directo sobre la ordenación.

La cláusula ORDER BY que se usa con las opciones OFFSET y FETCH, permite limitar el tamaño del resultado. El hecho de limitar el tamaño del resultado permite tener consultas más rápidas en tiempo de ejecución y por lo tanto, permite mostrar un resultado más rápido a nivel del programa cliente. Para permitir tener el resultado deseado, es posible indicar el número de registros que se deben ignorar, antes de integrar el registro al resultado con la opción OFFSET, así como el número de registros presentes en el resultado con FETCH.

Las opciones OFFSET y FETCH se activan en relación al juego de datos, ordenado en función de la cláusula ORDER BY.

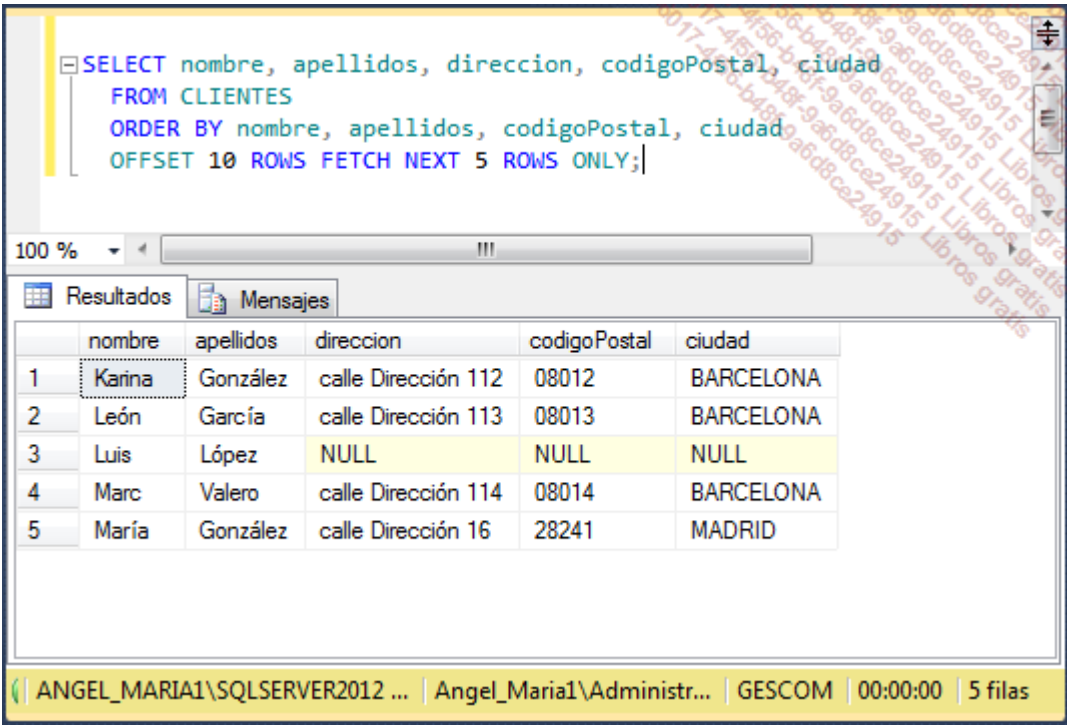
Sintaxis

```
ORDER BY columna1[, columna2, ... ] [ASC|DESC]
OFFSET numeroEntero{ROW|ROWS}
[FETCH {FIRST|NEXT} numeroEntero {ROW|ROWS} ONLY]
```

ROW y ROWS	Estos sinónimos solo se indican por razones de compatibilidad con la norma ANSI.
FIRST y NEXT	Estos sinónimos solo se indican por razones de compatibilidad con la norma ANSI.

Ejemplo

La siguiente consulta muestra 5 clientes después de ignorar los 10 primeros.



La cláusula ORDER BY que se utiliza junto con la instrucción CASE, permite definir una ordenación condicional, es decir que no es la misma en función de criterios definidos con antelación.

La implementación de este tipo de ordenación obliga a que los valores que permiten decidir el orden que se debe adoptar, se conozcan incluso antes de ejecutar la consulta.

Existen dos maneras principales de utilizar esta ordenación condicional. La primera consiste en definir una columna que juega el papel de indicador, para indicar, por ejemplo, si la ordenación es ascendente o descendente sobre una o varias columnas. La segunda consiste en decir que, en función del valor de una columna, la ordenación se hace sobre esa columna o sobre otra.

Sintaxis

ORDER BY CASE columna WHEN valor THEN columnaAOrdenar [, ...]

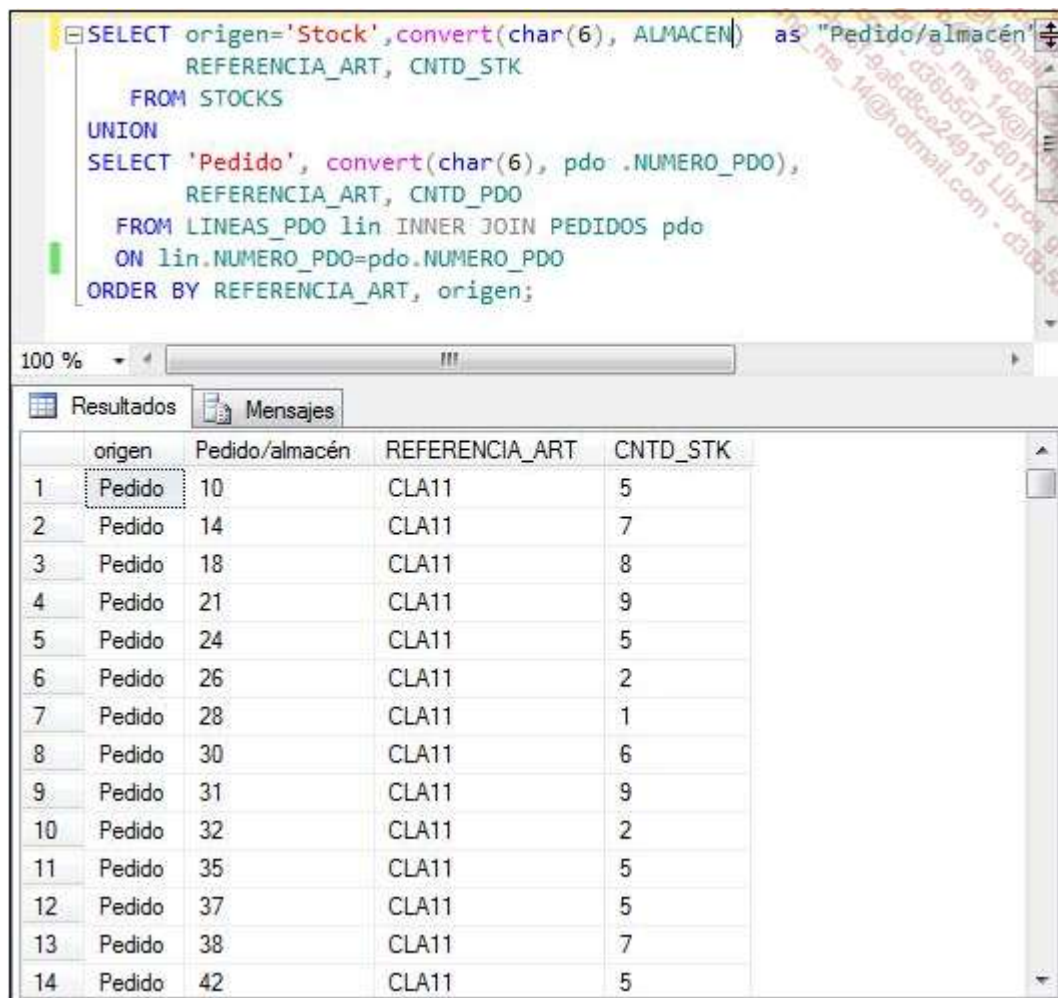
## j. Unión

El operador UNION permite obtener un conjunto de registros que proviene de varias consultas. Todas las consultas tienen que proporcionar el mismo número de columnas, del mismo tipo.

Los operadores de ensamblado que traducen la unión, intersección y diferencia, normalmente se olvidan porque se usan poco. Es importante no ignorarlos, porque algunas veces permiten simplificar mucho algunas consultas complejas.

### Ejemplo

Mostrar los registros de pedido y las cantidades en stock:



```
SELECT origen='Stock',convert(char(6), ALMACEN) as "Pedido/almacén",
REFERENCIA_ART, CNTD_STK
FROM STOCKS
UNION
SELECT 'Pedido', convert(char(6), pdo .NUMERO_PDO),
REFERENCIA_ART, CNTD_PDO
FROM LINEAS_PDO lin INNER JOIN PEDIDOS pdo
ON lin.NUMERO_PDO=pdo.NUMERO_PDO
ORDER BY REFERENCIA_ART, origen;
```

	origen	Pedido/almacén	REFERENCIA_ART	CNTD_STK
1	Pedido	10	CLA11	5
2	Pedido	14	CLA11	7
3	Pedido	18	CLA11	8
4	Pedido	21	CLA11	9
5	Pedido	24	CLA11	5
6	Pedido	26	CLA11	2
7	Pedido	28	CLA11	1
8	Pedido	30	CLA11	6
9	Pedido	31	CLA11	9
10	Pedido	32	CLA11	2
11	Pedido	35	CLA11	5
12	Pedido	37	CLA11	5
13	Pedido	38	CLA11	7
14	Pedido	42	CLA11	5

## k. Except

Este operador permite poner en práctica en SQL Server el operador de diferencia, definido en el álgebra relacional. Permite localizar los registros de datos presentes en un juego de resultados y que no están en otro.

Esta diferencia solo se puede hacer entre dos resultados que tengan la misma estructura, es decir, el mismo número de columnas, definidas en el mismo orden y sobre los mismos tipos de datos para los dos resultados.

### Ejemplo

Mostrar la lista de clientes que no viven en la Comunidad de Madrid (28).



<pre> SELECT * FROM CLIENTES EXCEPT SELECT * FROM CLIENTES WHERE codigoPostal BETWEEN 28000 and 28999; </pre>							
100 %							
Resultados Mensajes							
	numero	nombre	apellidos	direccion	codigopostal	ciudad	telefc
1	107	Juán	Sánchez	calle Dirección 18	08000	BARCELONA	04 0
2	108	Hector	Fernández	calle Dirección 19	08005	BARCELONA	
3	109	Irene	Suárez	calle Dirección 10	08002	BARCELONA	
4	1010	Jean	Oliva	calle Dirección 111	08001	BARCELONA	04 0
5	1011	Karina	González	calle Dirección 112	08012	BARCELONA	02 0
6	1012	León	García	calle Dirección 113	08013	BARCELONA	04 0
7	1013	Marc	Valero	calle Dirección 114	08014	BARCELONA	04 0
8	1014	Naomi	Zavala	calle Dirección 115	08015	BARCELONA	02 0
9	1015	Octavio	Pinto	calle Dirección 116	46001	VALENCIA	02 0
10	1016	Pascal	Osuma	calle Dirección 117	46012	VALENCIA	01 0
11	1017	Andrés	Fernández	NULL	46015	Quimper	02 1
12	1018	Ramis	Sánchez	calle Dirección 118	46018	VALENCIA	02 3
13	1019	Sandra	Fernández	calle Dirección 119	46003	VALENCIA	05 6
14	1020	Teddy	Trente	calle Dirección 120	46007	VALENCIA	02 3

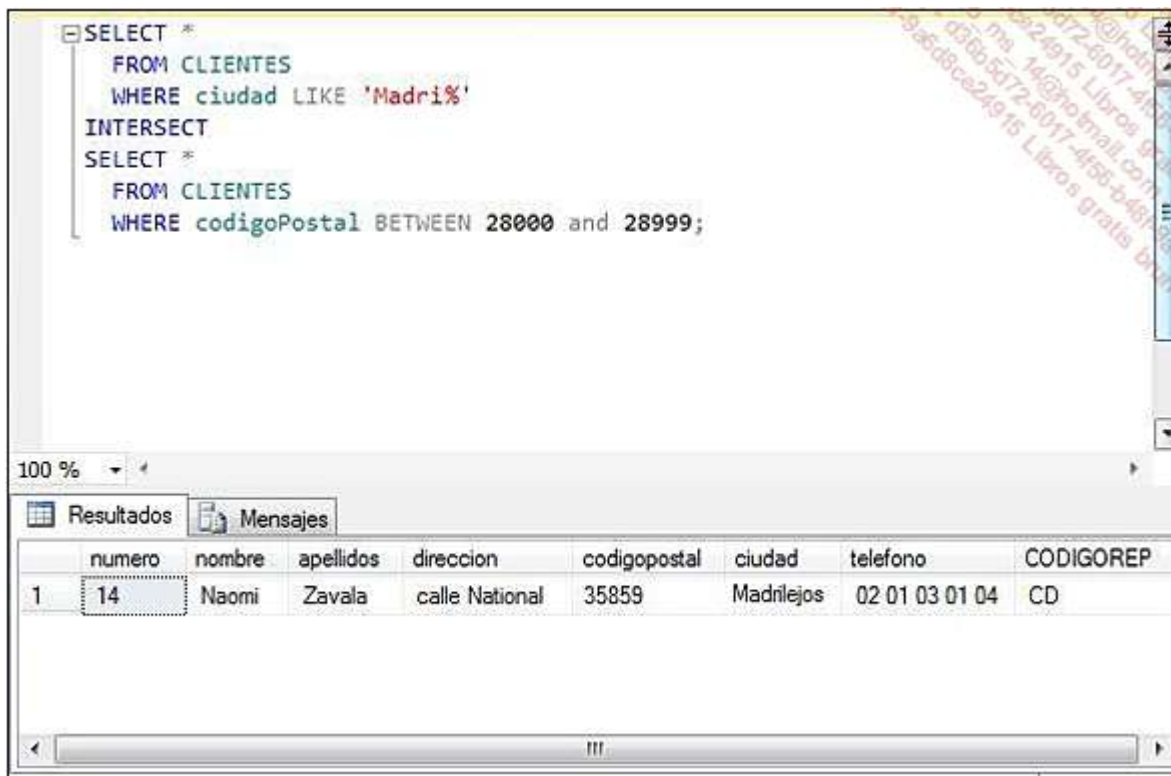
## I. Intersect

Este operador corresponde a la traducción en Transact SQL del operador de ensamblado de intersección. De esta manera, es posible identificar en una sola consulta SQL los registros de datos que están presentes de manera simultánea en dos resultados distintos, pero con la misma estructura.

Como sucede con la unión y la diferencia, la intersección solo se puede implementar entre resultados con la misma estructura.

### Ejemplo

*Mostrar la lista de clientes que viven en una ciudad cuyo nombre empieza por Madri y que no son de la Comunidad de Madrid (28):*



Hay otros medios para obtener el mismo resultado, pero el objetivo es ilustrar simplemente el operador INTERSECT.

### m. Extraer solo los primeros registros

La cláusula TOP permite extraer solo los primeros registros de un resultado. Esta cláusula está disponible para las instrucciones SELECT, INSERT, UPDATE, DELETE y MERGE.

El número de registros devueltos se especifica como argumento de la cláusula TOP. Este número se puede expresar como un valor o un porcentaje. Tanto en un caso como en el otro, es importante no subestimar las nociones de redondeo e igualdad de criterios de ordenación.

En caso de un porcentaje, la cláusula TOP devuelve siempre un número entero de registros. Por lo tanto, el valor calculado del número de registros que se tienen que devolver, siempre se redondea al entero inmediatamente superior.

#### Ejemplo

Si la tabla tiene 511 registros de datos y se usa la cláusula TOP 10 PERCENT, se devolverán 52 registros.

Otro caso particular que hay que tener en cuenta es el de las igualdades. El uso de la cláusula TOP se asocia muy habitualmente a la cláusula ORDER BY para ordenar los datos, según el orden deseado. Como sucede con cualquier ordenación, puede ocurrir que algunos registros contengan valores equivalentes en los criterios de ordenación. Para estar seguros de incluir todos los registros, es mejor añadir la opción WITH TIES cuando se use la cláusula TOP.

### Sintaxis

```
SELECT TOP (numero) [PERCENT] [WITH TIES] listaColumnas
FROM listaTablas...

numero
```

Representa el número de registros devueltos. Se trata de un número entero (bigint) o del porcentaje de registros que se deben devolver. Este porcentaje se puede expresar como un número en coma flotante.

PERCENT

Especifica que el número representa un porcentaje.

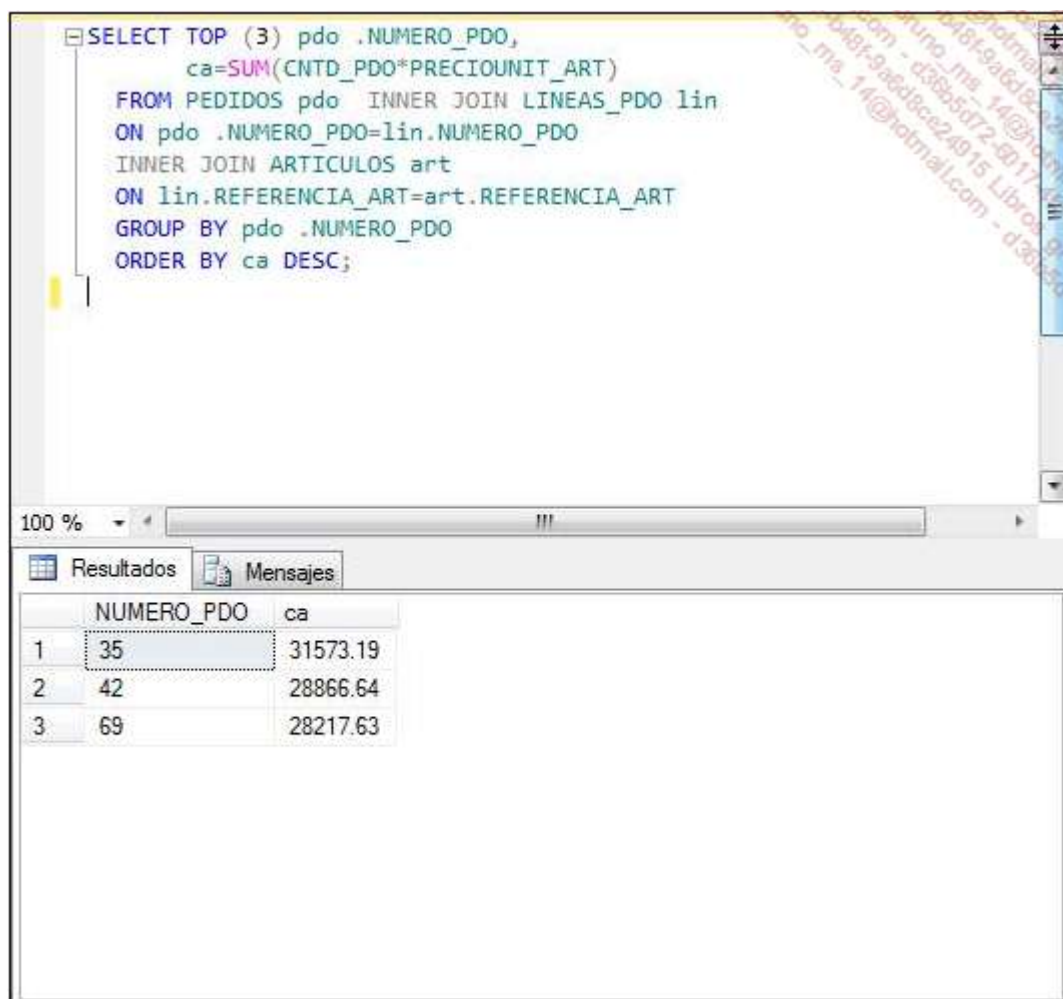
#### WITH TIES

La opción WITH TIES permite incluir en el resultado los registros que son iguales en los criterios de selección. Usando esta opción, puede suceder que el número de registros realmente devuelto sea superior al número de registros que se indica en la cláusula TOP.

En la cláusula SELECT, es opcional especificar entre paréntesis el número de registros o el porcentaje expresado por TOP, pero se recomienda utilizarlos.

#### Ejemplo

*Los pedidos se ordenan por cifras de negocio de manera descendente y solo queremos saber los tres primeros.*



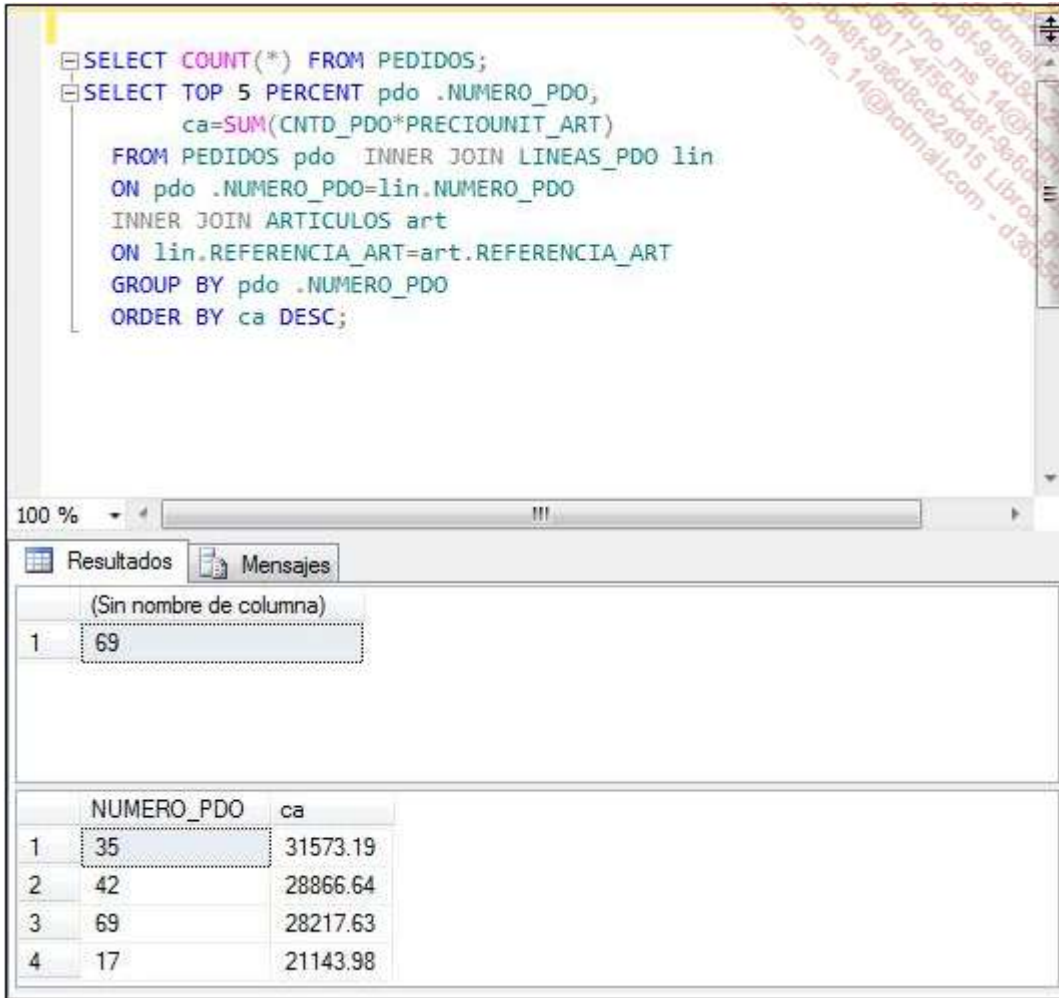
The screenshot shows a SQL query editor with the following query:

```
SELECT TOP (3) pdo .NUMERO_PDO,  
              ca=SUM(CNTD_PDO*PRECIUNIT_ART)  
FROM PEDIDOS pdo INNER JOIN LINEAS_PDO lin  
ON pdo .NUMERO_PDO=lin.NUMERO_PDO  
INNER JOIN ARTICULOS art  
ON lin.REFERENCIA_ART=art.REFERENCIA_ART  
GROUP BY pdo .NUMERO_PDO  
ORDER BY ca DESC;
```

Below the query editor, the results are displayed in a table with two columns: NUMERO\_PDO and ca. The table shows the top 3 results ordered by ca in descending order.

	NUMERO_PDO	ca
1	35	31573.19
2	42	28866.64
3	69	28217.63

En este segundo ejemplo, la cláusula PERCENT permite conocer el 5% del resultado final:



```
SELECT COUNT(*) FROM PEDIDOS;  
SELECT TOP 5 PERCENT pdo .NUMERO_PDO,  
    ca=SUM(CNTD_PDO*PRECIOUNIT_ART)  
FROM PEDIDOS pdo INNER JOIN LINEAS_PDO lin  
ON pdo .NUMERO_PDO=lin.NUMERO_PDO  
INNER JOIN ARTICULOS art  
ON lin.REFERENCIA_ART=art.REFERENCIA_ART  
GROUP BY pdo .NUMERO_PDO  
ORDER BY ca DESC;
```

100 %

Resultados Mensajes

(Sin nombre de columna)

1	69
---	----

	NUMERO_PDO	ca
1	35	31573.19
2	42	28866.64
3	69	28217.63
4	17	21143.98

## 6. Consulta de creación de tablas

Es posible crear una nueva tabla a partir de una consulta, utilizando la siguiente sintaxis:

```
SELECT..... INTO nombreTabla FROM.....
```

La nueva tabla tendrá el esquema que corresponde a las columnas extraídas. En caso de columnas calculadas, se debe indicar un nombre de alias o un título.

Si el nombre de la tabla está precedido de #, la tabla será temporal local. Si se precede de ##, será una tabla temporal global. Estos dos tipos de tablas se almacenan en la base de datos tempdb.

Solo se puede acceder a una tabla temporal en la sesión en la que se ha creado y desaparece después de la desconexión. A una tabla temporal global se puede acceder desde todas las sesiones y se elimina cuando termina la última sesión que la ha utilizado.

Cuando las tablas se crean para permitir solucionar extracciones complejas, es mejor crear una tabla temporal.

El hecho de elegir crear una tabla temporal, no debe ocultar el hecho de que sigue siendo necesario eliminarla lo antes posible con la instrucción DROP TABLE.

Es importante resaltar que es muy habitual que existan maneras más elegantes de resolver las consultas complejas, que crear una tabla intermedia (incluso temporal). Estas soluciones se articulan alrededor de las vistas, las tablas CTE y la escritura correcta de la consulta SELECT.

### Ejemplos

Creación de una nueva tabla en la base de datos actual:



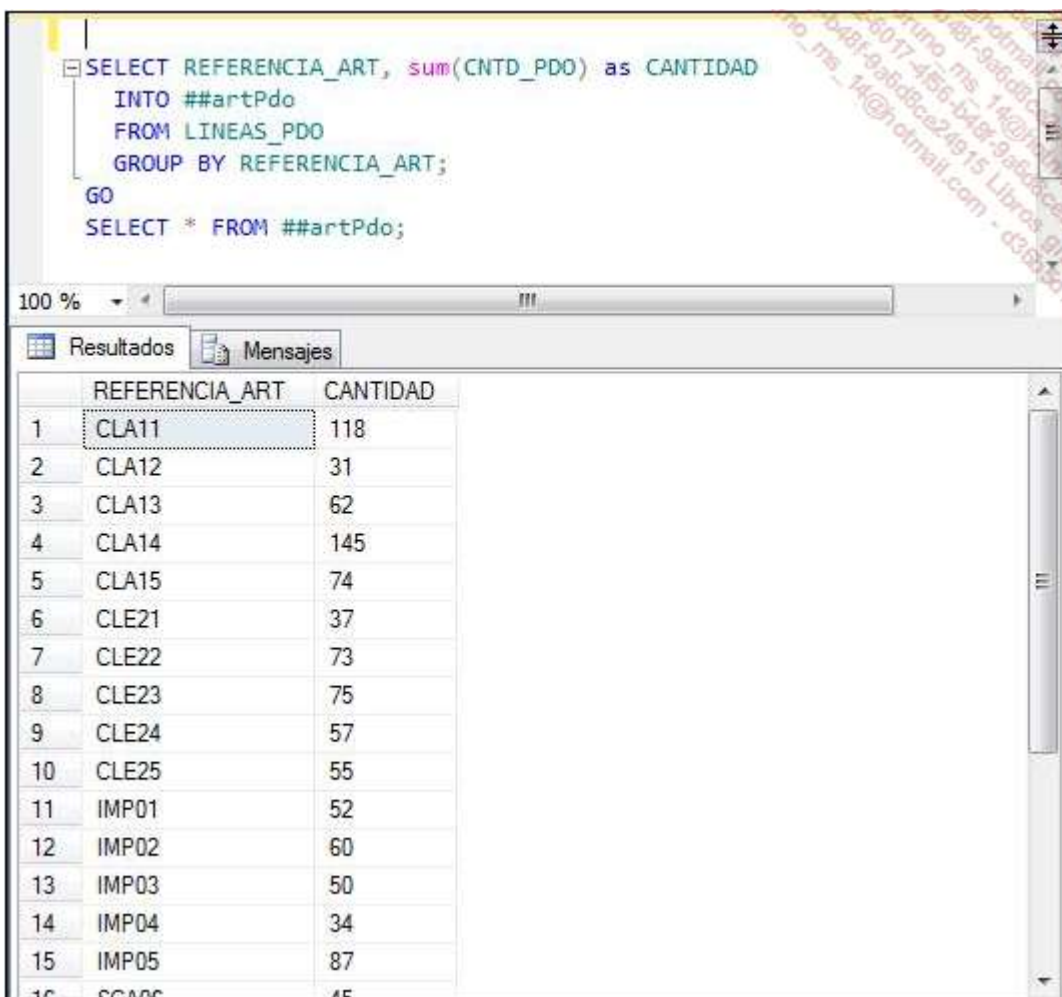
```
SELECT numero, nombre, apellidos
INTO CLIBARCELONA
FROM CLIENTES
WHERE ciudad='Barcelona';
GO
SELECT *
FROM CLIBARCELONA;
```

100 %

Resultados Mensajes

	numero	nombre	apellidos
1	14	Naomi	Zavala

Se establece la lista de artículos y el número de veces que se han pedido. Después se almacenan en la tabla ##artPdo, que es una tabla temporal global.



```
SELECT REFERENCIA_ART, sum(CNTD_PDO) as CANTIDAD
INTO ##artPdo
FROM LINEAS_PDO
GROUP BY REFERENCIA_ART;
GO
SELECT * FROM ##artPdo;
```

100 %

Resultados Mensajes

	REFERENCIA_ART	CANTIDAD
1	CLA11	118
2	CLA12	31
3	CLA13	62
4	CLA14	145
5	CLA15	74
6	CLE21	37
7	CLE22	73
8	CLE23	75
9	CLE24	57
10	CLE25	55
11	IMP01	52
12	IMP02	60
13	IMP03	50
14	IMP04	34
15	IMP05	87
16	SCA06	45

## 7. Forzar el optimizador de consultas

SQL es un lenguaje interpretado, que permite describir el resultado que queremos obtener. Es relativamente fácil obtener una descripción válida del resultado. Para un mismo resultado pueden existir diferentes maneras de hacerlo.

A partir de esta descripción (consulta SELECT), el optimizador decide cuál es el mejor camino que se debe utilizar para calcular el resultado.



La cláusula `OPTION` de la consulta `SELECT`, permite especificar al optimizador de consultas la manera en la que debe definir el plan de ejecución de la consulta. Por supuesto, como el volumen de datos cambia sin parar y se pueden hacer mejoras de estructura (definiendo índice, por ejemplo), esta solución se debe utilizar con moderación. En cualquier caso, es preferible dejar al optimizador de consultas la tarea de definir el plan de ejecución. Efectivamente, casi siempre, el optimizador encuentra el mejor plan de ejecución posible. Las directivas de destino del optimizador de consulta se deben usar como último recurso y las tiene que aplicar un desarrollador experimentado o el administrador de la base de datos, quien tiene una visión global de la organización de los datos en la base de datos.

## 8. Tablas CTE

El objetivo de las tablas CTE (*Common Table Expression*), es simplificar la escritura y, por tanto, la comprensión de las consultas. Una tabla CTE se puede considerar como una tabla temporal y específica de una instrucción SQL DML. Una alternativa al uso de tablas CTE es definir una tabla temporal local (`#MyTable`), antes de la consulta del SQL DML y eliminarla inmediatamente después de la ejecución de la consulta. Evidentemente, esta alternativa es mucho más compleja de administrar y menos apropiada en términos de programación.

Las tablas CTE permiten escribir de manera sencilla las consultas complejas, simplificando considerablemente la escritura de consultas anidadas.

Las tablas CTE se pueden usar en el contexto de una consulta de extracción de datos (`SELECT`) y en consultas de modificación de datos (`INSERT`, `UPDATE` o `DELETE`), aunque este último caso es menos habitual.

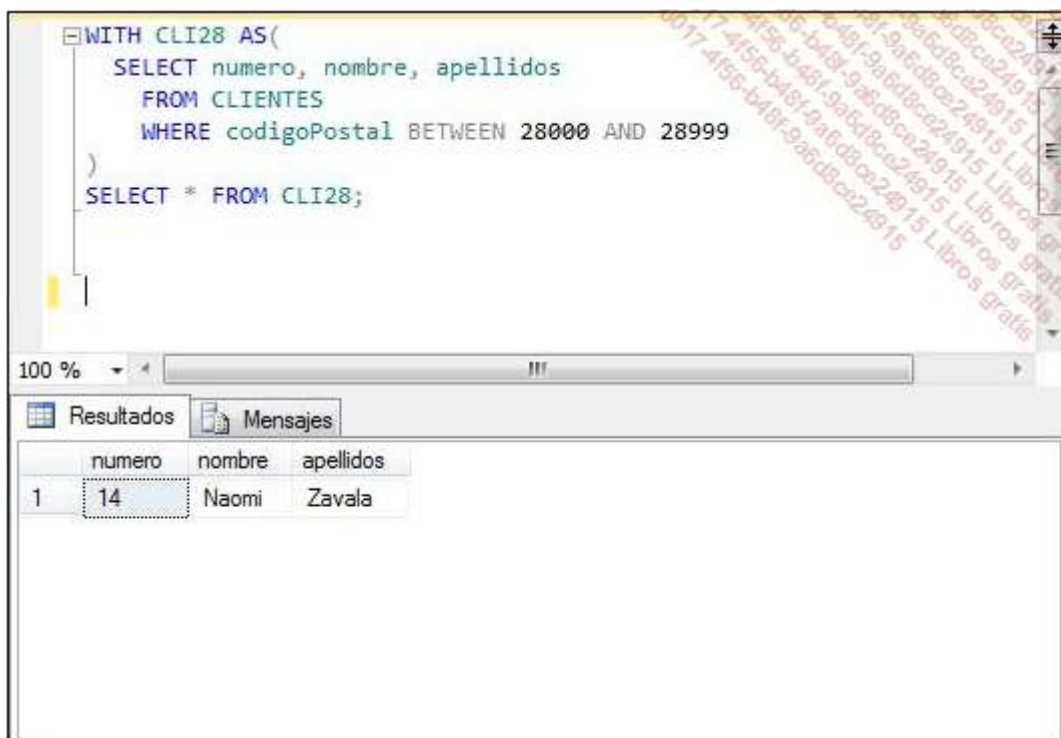
Las CTE son elementos de la norma ANSI SQL 99 o SQL 3.

### Sintaxis

```
WITH nombreTablaCTE(nombreColumna1, nombreColumna2, ...) AS
(
    consultaSelect
)
```

La tabla CTE se crea usando la instrucción `WITH`, seguida del nombre de la tabla acompañada de la lista de columnas. Para terminar, la consulta `SELECT` que sirve de base de datos para la construcción de la tabla CTE, se define después de la palabra clave `AS`.

*Ejemplo de implementación de una tabla CTE*



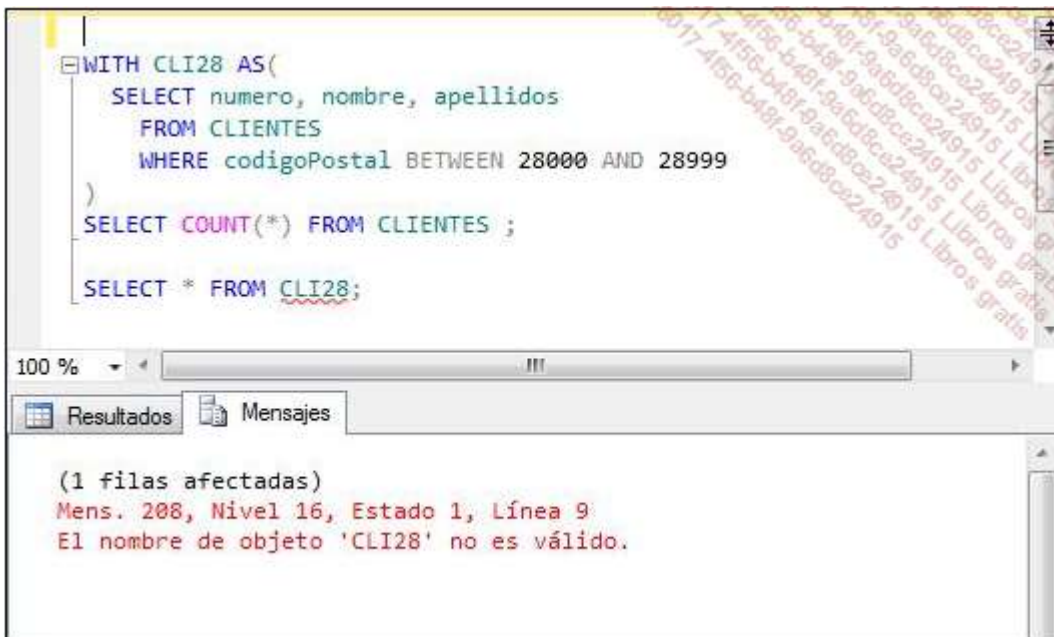
The screenshot shows a SQL query editor with the following code:

```
WITH CLI28 AS(
    SELECT numero, nombre, apellidos
    FROM CLIENTES
    WHERE codigoPostal BETWEEN 28000 AND 28999
)
SELECT * FROM CLI28;
```

Below the query editor, there is a results pane with the following data:

	numero	nombre	apellidos
1	14	Naomi	Zavala

Una vez que se ha definido, la tabla CTE se debe usar inmediatamente. En caso contrario se produce un error.



Las tablas CTE se crean más fácil y rápidamente que las tablas temporales, gracias a una sintaxis particularmente sencilla. Además, la tabla CTE no tiene sentencias DDL. Por el contrario, las restricciones de uso de la tabla CTE son más numerosas que las de una tabla temporal:

- Se tienen que usar inmediatamente después de su definición.
- No es posible utilizar las instrucciones COMPUTE, ORDER BY, INTO, FOR XML y FOR BROWSE.

## 9. Generación de registros estadísticos

### ROLLUP y CUBE

Para mejorar los resultados realizados por los cálculos agregados y permitir el cálculo de sumas intermedias, SQL Server ofrece los ROLLUP y CUBE. Un uso reflexivo de estas instrucciones puede dar la posibilidad de ejecutar todos los cálculos en el servidor, para que el programa cliente que lanza la consulta solo tenga la tarea de formatear la ejecución.

Las cláusulas COMPUTE y COMPUTE BY ya no están disponibles en SQL Server 2012. Por lo tanto es necesario modificar las consultas y hacer referencia usando la cláusula ROLLUP, para obtener los mismos resultados.

Los operadores ROLLUP y CUBE se usan junto con la cláusula GROUP BY y las funciones estadísticas, para obtener los registros adicionales que contienen el cálculo de la función, para agrupaciones combinadas.

La cláusula WITH ROLLUP permite crear registros que tienen resultados estadísticos para las agrupaciones de las columnas del GROUP BY, combinadas de izquierda a derecha. Por ejemplo, si solicitamos la suma para una agrupación sobre las columnas A, B y C, la cláusula WITH ROLLUP proporcionará además la suma para una agrupación sobre A, sobre A y B y la suma total.

La cláusula WITH CUBE permite crear registros adicionales para todas las combinaciones de agrupación de las columnas del GROUP BY. Para el ejemplo anterior, además tendríamos la suma para una agrupación sobre B, sobre C, sobre A y C y sobre B y C.

Podemos utilizar un máximo de 10 expresiones de agrupación, para un tamaño total de 900 bytes.