Lenguaje SQL – TSQL 2008

Marta Zorrilla

UNIVERSIDAD DE CANTABRIA

Tabla de contenidos

Bibliografía

- Básica
  - Cap. 8 y 24. Elmasri, R., Navathe, S.B., Fundamentos de Sistemas de Bases de Datos, 5ª; edición, Pearson Education, 2008.
  - Cap. 8. Mora, E., Zorrilla, M. E., Díaz de Entresotos, J. Iniciación a las bases de datos con Access 2002. Díaz de Santos, 2003.
  - Cap. 7. Pons, O. et al. Introducción a los sistemas de bases de datos. Paraninfo. 2008
  - Cap. 3 y 4. Silberschatz, A., Korth, H.F., Sudarshan, S., Fundamentos de Bases de Datos, 5ª edición, Madrid, 2006.
- Complementaria
  - Coles, Michael. Pro T-SQL 2008 programmer's guide. Apress, cop. 2008
  - Vieira, Robert. Professional Microsoft SQL Server 2008 programming. Wiley, cop. 2009.

Introducción al lenguaje SQL

- Lenguaje declarativo de acceso a los datos.
- Estándar para las bases de datos relacionales.
- Incluye la capacidad de manipular tanto la estructura de la base de datos como sus datos. Aspectos de seguridad.
- Desarrollado en el Laboratorio de investigación de San José de IBM. Fue utilizado por primera vez en 1970.
- En 1986:
  ANSI (American National Standards Institute) e
  ISO (International Standards Organization)
  publicaron una norma, denominada **SQL-86**.

  Ésta ha tenido dos actualizaciones: **SQL-89** y **SQL-92**.

  En la actualidad, se trabaja con el **SQL:1999** y **SQL2003**

Tabla de contenidos

CASO DE ESTUDIO: BD Compras

La base de datos trata de informatizar el proceso de compras de una empresa. Esto es, recoger los pedidos de los artículos, contemplados en su catálogo, que compran a proveedores ya conocidos.
Reglas:
- No puede seleccionar un artículo descatalogado.
- Si el stock alcanza el mínimo establecido, se ha de notificar como un evento.

Pedido num. *2* ***Fecha*** *10/06/2002*

Fecha prevista de entrega *15/08/2002*

Proveedor

Código *P002* *Zar Luna, Ana*
Ercilla 22, 1º
48002 Bilbao

Tipo arancel *UE*

Linea	*Código art.*	*Descripción*	*Uds.*	*Precio compra*	*Dto.*	*Importe*
1	0002	SILLA ERGONOMICA MOD.MX	3,00	120,00 €	2,00	353 €
2	0003	ARMARIO DIPLOMATIC	2,00	300,00 €	3,00	582 €
3	0002	SILLA ERGONOMICA MOD.MX	5,00	120,00 €	0,00	600 €

Total sin IVA 1.535 €

IVA (16 %) 245,57 €

Total con IVA 1.780,37 €

Esquema relacional

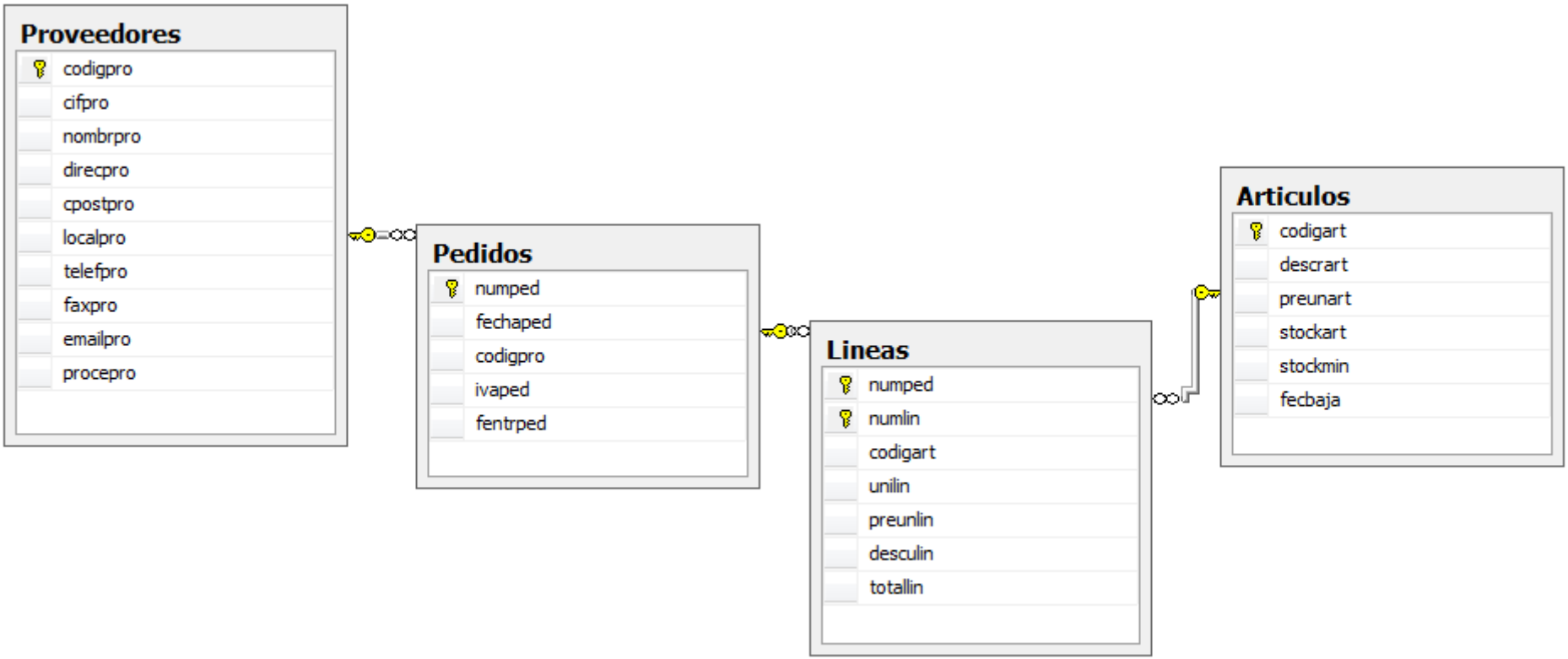


Tabla de contenidos

Consideraciones previas

- Qué tipo de información se va a almacenar. Por ejemplo, no se pueden guardar caracteres en un campo cuyo tipo de datos sea numérico.
- El espacio de almacenamiento necesario (dimensionar el campo).
- Qué tipo de operaciones se van a realizar con los valores del campo. Pues, por ejemplo, no se puede calcular la suma de dos cadenas de texto.
- Si se desea ordenar o indexar por ese campo. Los criterios de ordenación difieren en función del tipo de dato, así, los números almacenados en un campo texto se ordenan según el valor de su código ASCII (1,10,11,2,20,...) que no coincide con la ordenación numérica.

Tipos de Datos (I)

- Numéricos Exactos:

Tipo	Desde	Hasta
bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255
bit	0	1
decimal	-10^38 +1	10^38 –1
numeric	-10^38 +1	10^38 –1
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214,748.3648	+214,748.3647

Tipos de Datos (II)

- Numéricos Aproximados:

Tipo	Desde	Hasta
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

- Fechas / Horas:

Tipo	Desde	Hasta	Precisión
datetime	1 Enero de 1753	31 Diciembre 9999	3.33 ms.
smalldatetime	1 Enero de 1900	6 Junio 2079	1 minuto
time	N/A	N/A	1 ns.
date	1 Enero de 0001	31 Diciembre 9999	1 día
datetime2	1 Enero de 0001	31 Diciembre 9999	*Variable*
datetimeoffset	1 Enero de 0001	31 Diciembre 9999	*Variable*

Tipos de Datos (III)

- Texto

Tipo	Variable	Unicode	Capacidad
char	NO	NO	8000
varchar	SI	NO	8000
varchar(max)	SI	NO	2^{31}
text	SI	NO	2,147,483,647
nchar	NO	SI	4000
nvarchar	SI	SI	4000
nvarchar(max)	SI	SI	2^{30}
ntext	SI	SI	1,073,741,823

Tipos de Datos (IV)

- Binarios

Tipo	Variable	Capacidad
binary	NO	8000
varbinary	SI	8000
varbinary(max)	SI	2^{31} FILESTREAM
image	SI	2,147,483,647

Tipos de Datos (V)

- Otros tipos de datos

Tipo	Comentario
XML	Almacena una instancia de XML
HierarchyID	Representa la posición en una jerarquía. No representa un árbol.
Table	Un tipo de datos especial que se utiliza para almacenar un conjunto de resultados para un proceso posterior.
Rowversion	Un número único para toda la base de datos que se actualiza cada vez que se actualiza una fila. (utilizar rowversion para versiones futuras)
Sql_variant	Un tipo de datos que almacena valores de varios tipos de datos aceptados en SQL Server, excepto text, ntext, rowversion y sql_variant
Uniqueidentifier	Un identificador exclusivo global (GUID), necesario para
Cursor	Una referencia a un cursor.

Tipos de datos autonumérico

***IDENTITY** [(semilla , incremento)]*

- ***semilla**:* valor de inicio.
- ***incremento:*** incremento que se aplica

```
CREATE TABLE dbo.herramientas(
   ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
   Nombre VARCHAR(40) NOT NULL
)
-- insertamos valores
INSERT INTO dbo.herramientas (Nombre ) VALUES ('Martillo')
INSERT INTO dbo.herramientas (Nombre ) VALUES ('Taladro')

-- si borramos, Martillo, se pierde el ID 1. Para reutilizarlo

SET IDENTITY_INSERT dbo.Tool ON
INSERT INTO dbo.herramientas (ID, Nombre) VALUES (1, Serrucho')
```

Tabla de contenidos

Lenguaje de definición de datos

- Las IDD comprenden todas las operaciones necesarias para implantar y mantener un esquema relacional.
- Con ellas, se describen los datos y su agrupamiento formando tablas, así como las restricciones que deben cumplir.
- Las IDD permiten crear, modificar y eliminar tablas, así como todos los componentes que las definen: campos, índices, claves, etc. y las restricciones que sean precisas.
- Principales instrucciones:
  - CREATE DATABASE
  - CREATE TABLE } Cláusula CONSTRAINT
  - ALTER TABLE
  - CREATE INDEX
  - CREATE VIEW
  - CREATE TRIGGER
  - CREATE PROCEDURE / FUNCTION
  - CREATE RULE
  - CREATE SCHEMA
  - DROP “objeto”

CREATE DATABASE

Para crear una base de datos. Su sintaxis es:

```
CREATE DATABASE nombreBD
[ ON
   [PRIMARY][ < fichero > [ ,...n ] ]
   [ , < grupo_fichero > [ ,...n ] ]
]
[ LOG ON { < fichero > [ ,...n ] } ]
[ COLLATE collation_name ]
[FOR ATTACH |
     FOR ATTACH REBUILT_LOG]

< fichero > ::=
 ( [ NAME = logical_file_name , ]
   FILENAME = 'os_file_name'
   [ , SIZE = size ]
   [ , MAXSIZE = { max_size | UNLIMITED } ]
   [ , FILEGROWTH = growth_increment ] ) [ ,...n ]

< grupo_fichero > ::=
FILEGROUP filegroup_name  [CONTAINS FILESTREAM] < fichero >
   [ ,...n ]
```

- ***nombreBD***: nombre de la BD que se va a crear.
- ***collation_name***: mapa de caracteres
- ***logical_file_name*** : nombre lógico del fichero.
- ***os_file_name*** : nombre físico del fichero.
- ***size***: tamaño del fichero.
- ***max_size***: tamaño máximo del fichero.
- ***growth_increment*** : incremento del fichero.
- ***filegroup_name***: nombre grupo de archivos

CREATE DATABASE: ejemplo

```
CREATE DATABASE [compras]
 ON (NAME = N'compras',
    FILENAME = N'C:\data\compras.mdf' ,
    SIZE = 2, MAXSIZE = 3000,FILEGROWTH = 10%)
 LOG ON (NAME = N'compras_log',
    FILENAME = N'C:\data\compras_log.LDF' ,
    SIZE = 1, FILEGROWTH = 10%)
 COLLATE Modern_Spanish_CI_AS
```

CREATE TABLE

```
CREATE TABLE
 [ database_name . [ schema_name ] . | schema_name . ] table_name
   ( { <column_definition> | <computed_column_definition>  |
       <column_set_definition>}
 [ <table_constraint> ] [ ,...n ] )
 [ ON { partition_scheme_name ( partition_column_name ) | filegroup | "default" } ]
 [ { TEXTIMAGE_ON { filegroup | "default" } ]
 [ FILESTREAM_ON { partition_scheme_name | filegroup | "default" } ]
 [ WITH ( <table_option> [ ,...n ] ) ] [ ; ]

<column_definition> ::=
  column_name <data_type> [ FILESTREAM ]   [ COLLATE collation_name ]
  [ NULL | NOT NULL ]
  [    [ CONSTRAINT constraint_name ] DEFAULT constant_expression ]
    | [ IDENTITY [ ( seed ,increment ) ] [ NOT FOR REPLICATION ] ]
  [ ROWGUIDCOL ] [ <column_constraint> [ ...n ] ]
  [ SPARSE ]
```

- ***TEXTIMAGE_ON:*** indica en qué grupo de archivos se almacenan las columnas text, ntext e image.
- ***FILESTREAN_ON:*** indica en qué grupo de archivos se almacenan las columnas varbinary
- ***SPARSE: columna con muchas filas NULAS***
- ***FILESTREAM: especifica almacenamiento filestream (fichero SO)***

CREATE TABLE

<table options> ::=
{ DATA_COMPRESSION = { NONE | ROW | PAGE }
[ON PARTITIONS ({ <partition_number_expression> | <range> } [, ...n])] }

<computed_column_definition> ::=
column_name AS c*omputed_column_expression*
[PERSISTED [NOT NULL]]
[[CONSTRAINT *constraint_name*]
{ PRIMARY KEY | UNIQUE }
[CLUSTERED | NONCLUSTERED]
[WITH FILLFACTOR = *fillfactor* | WITH **(** <index_option> [**,** ...*n*]**)**]
| [FOREIGN KEY] REFERENCES *referenced_table_name* [**(** *ref_column* **)**]
[ON DELETE { NO ACTION | CASCADE }]
[ON UPDATE { NO ACTION }]
[NOT FOR REPLICATION]
| CHECK [NOT FOR REPLICATION] **(** *logical_expression* **)**
[ON { *partition_scheme_name* **(** *partition_column_name* **)** | *filegroup* | "default" }]]

COLUMN CONSTRAINT

Restricción a nivel de columna

```
CONSTRAINT nombre
 { [ NULL | NOT NULL ] |
  [ { PRIMARY KEY | UNIQUE }
       [ CLUSTERED | NONCLUSTERED ]
       [ WITH FILLFACTOR = factor_relleno ]
       [ ON {grupo_ficheros | DEFAULT} ] ]
    ] |
  [ [ FOREIGN KEY ]
       REFERENCES otra_tabla[ (campo_externo1) ]
       [ ON DELETE {NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
       [ ON UPDATE {NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
       [ NOT FOR REPLICATION ]
    ] |
  CHECK [ NOT FOR REPLICATION ]  ( expresión_lógica )
 }
```

- ***nombre***: es el nombre de la restricción que se va a crear.
- ***otra_tabla***: es el nombre de la tabla a la que se hace referencia.
- ***campo_externo1***: son los nombres de los campos de la ***otra_tabla*** a los que se hace referencia.
- ***factor_relleno***: especifica cuánto se debe llenar cada página de índice utilizada para almacenar los datos de índice.Entre 0 y 100. Por defecto 0.
- ***grupo_ficheros***: indica dónde se almacena la tabla
- ***expresión_lógica***: Expresión que devuelve true o false

TABLE CONSTRAINT

Restricción a nivel de tabla

```
CONSTRAINT nombre
 {      [ { PRIMARY KEY | UNIQUE }
           [ CLUSTERED | NONCLUSTERED ]
           { (principal1 [ ASC | DESC ] [ , principal2 [, ...] ] ) }
           [ WITH FILLFACTOR = factor_relleno]
           [ ON { grupo_ficheros | DEFAULT } ]
        ]     |
        FOREIGN KEY
           [ (referencia1 [, referencia2 [, ...] ] ) ]
           REFERENCES otra_tabla [ (campo_externo1 [ ,... campo_externo2 ] ) ]
           [ ON DELETE {NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
           [ ON UPDATE {NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
           [ NOT FOR REPLICATION ]    |
        CHECK [ NOT FOR REPLICATION ]  (expresión_lógica )
 }
```

- ***nombre***: es el nombre de la restricción que se va a crear.
- ***principal1, principal2***: son los nombres de los campos que compondrán la clave principal.
- ***referencia1, referencia2***: son los nombres de los campos que hacen referencia a otros de otra tabla.
- ***otra_tabla***: es el nombre de la tabla a la que se hace referencia.
- ***campo_externo1, campo_externo2***: son los nombres de los campos de la ***otra_tabla*** a los que se hace referencia.
- ***expresión_lógica***: criterio que se ha de cumplir. Devuelve true o false

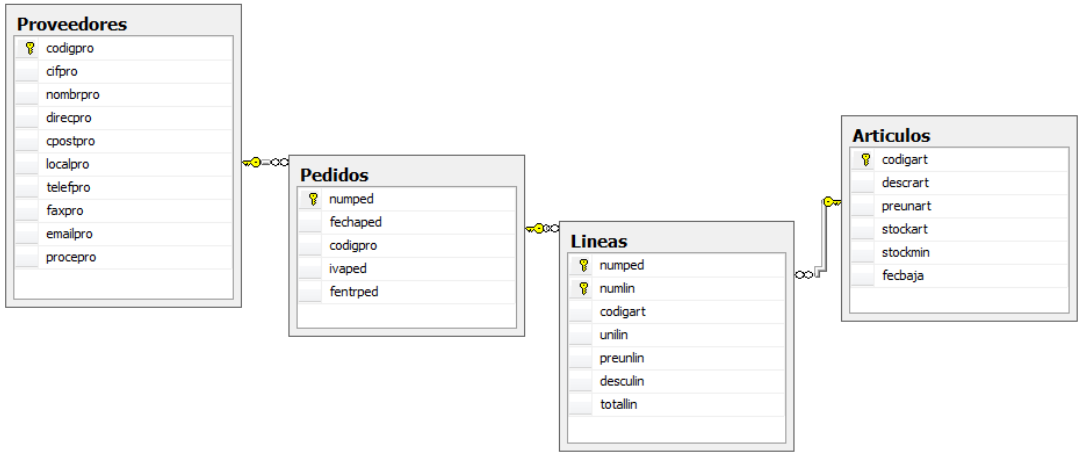
CREATE TABLE: EJEMPLOS


```
CREATE TABLE Proveedores (
  codigpro        CHAR(4)   NOT NULL    CONSTRAINT id_pro PRIMARY KEY,
  cifpro          CHAR(12)  NOT NULL    CONSTRAINT u_cif UNIQUE,
  nombrpro        CHAR(30)   NOT NULL,
  direcpro        CHAR(30)   NOT NULL,
  cpostpro        CHAR(5)     NOT NULL CHECK (cpostpro like '[0-9][0-9][0-9][0-9][0-9]'),
  localpro        CHAR(20)   NOT NULL,
  telefpro        CHAR(17)   NOT NULL,
  faxpro          CHAR(17),
  emailpro        CHAR(25),
  procepro        CHAR(10)  NOT NULL CHECK (procepro in ('UE', 'No UE')))
```

CREATE TABLE: EJEMPLOS (y 2)

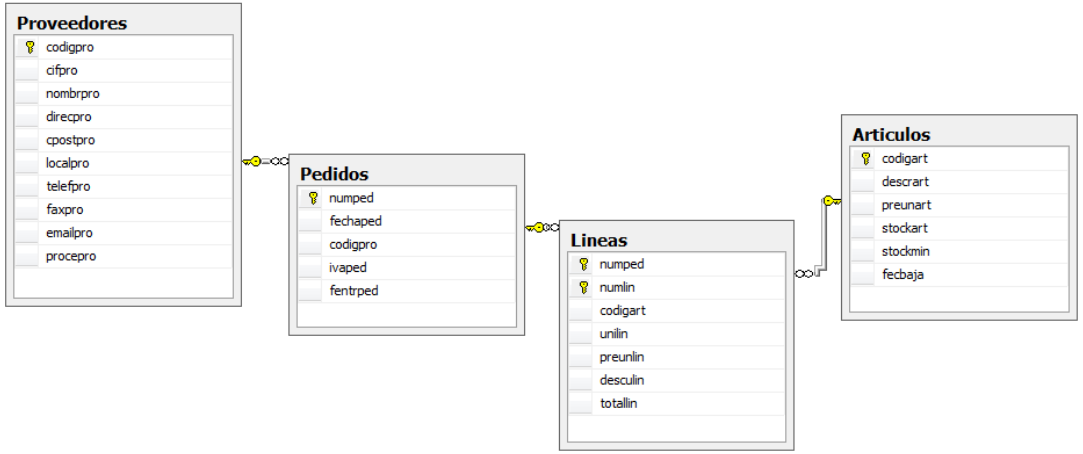

```
CREATE TABLE Articulos (
   codigartCHAR(6)   NOT NULL    CONSTRAINT id_art PRIMARY KEY,
   descrart              CHAR(40) NOT NULL,
   preunart              MONEY    NOT NULL,
   stockart              INTEGER  NOT NULL CHECK (stockart >=0),
   stockmin              INTEGER  NOT NULL CHECK (stockmin>=0),
            fecbaja     DATE)
```

CREATE TABLE: EJEMPLOS (y 3)




```
CREATE TABLE Pedidos (
   numped          INTEGER          NOT NULL   CONSTRAINT id_ped PRIMARY KEY,
   fechaped         DATE            NOT NULL   DEFAULT getdate(),
   codigpro        CHAR(4)          NOT NULL,
   ivaped          FLOAT            NOT NULL CHECK (ivaped>0 and ivaped<100),
   fentrped        DATE             NOT NULL,
         CONSTRAINT f_pro FOREIGN KEY (codigpro) REFERENCES Proveedores (codigpro),
         CONSTRAINT c_fecha CHECK (fechaped<=fentrped))
```

CREATE TABLE: EJEMPLOS (y 4)




```
CREATE TABLE Lineas (
   numped          INTEGER NOT NULL,
   numlin          SMALLINT NOT NULL,
   codigart        CHAR(6)  NOT NULL,
   unilin          FLOAT    NOT NULL,
   preunlin        MONEY    NOT NULL,
   desculin        FLOAT    NOT NULL CHECK (desculin>=0 and desculin<=100),
   totallin     AS ([preunlin] * [unilin] * (1 –[desculin] / 100)),
        CONSTRAINT id_lin PRIMARY KEY (numped, numlin),
        CONSTRAINT f_ped FOREIGN KEY (numped) REFERENCES Pedidos (numped),
        CONSTRAINT f_art FOREIGN KEY (codigart) REFERENCES Articulos (codigart))
```

ALTER TABLE

Para modificar el diseño de una tabla que ya existe en la base de datos. Su sintaxis es:

```
ALTER TABLE tabla
{
 ALTER COLUMN  { campo  tipo [(tamaño)] }  [COLLATE] [ NULL | NOT NULL]  |
 ADD
    {<column_definition> | <computed_definition>|<table_constraint>} [,…n]          |
DROP
    {COLUMN nombre_campo   |   CONSTRAINT nombre_restricción} [,…n]        |
 { ENABLE | DISABLE } TRIGGER  { ALL | nombre_trigger [ ,...n ] }                    |
 { CHECK | NOCHECK } CONSTRAINT { ALL | nombre_restricción[ ,...n ] }}
```

ALTER TABLE: ejemplos

```
ALTER TABLE [dbo].[Proveedores]
   ADD CONSTRAINT [id_pro] PRIMARY KEY  CLUSTERED ( [codigpro] )
   ON [PRIMARY]
GO


ALTER TABLE [dbo].[Pedidos] WITH NOCHECK ADD
  CONSTRAINT [DF_Pedidos_fechaped] DEFAULT (getdate()) FOR [fechaped],
  CONSTRAINT [c_fecha] CHECK ([fechaped] <= [fentrped]),
  CHECK ([ivaped] > 0 and [ivaped] < 100)
GO


ALTER TABLE [dbo].[Proveedores] ADD
   CONSTRAINT [u_cif] UNIQUE  NONCLUSTERED ( [cifpro] )  ON [PRIMARY] ,
   CHECK ([cpostpro] like '[0-9][0-9][0-9][0-9][0-9]'),
   CHECK ([procepro] = 'No UE' or [procepro] = 'UE')
GO
```

CREATE INDEX

Para crear un índice nuevo en una tabla que ya existe en la base de datos. Su sintaxis es:

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX índice
   ON {tabla | vista } (campo1 [ {ASC | DESC }] [, campo2 [{ASC|DESC}], ...])
[ INCLUDE (column_name,…n) ]
[ WHERE filtro ]
[ WITH < opción_índice > [ ,...n] ]
[ ON {grupo_ficheros | particion | defecto } ]

< opción_índice > :: =
  { PAD_INDEX | FILLFACTOR = factor_relleno | IGNORE_DUP_KEY |
    DROP_EXISTING | STATISTICS_NORECOMPUTE |  SORT_IN_TEMPDB |
ALLOW_ROW_LOCKS | ALLOW_PAGE_LOCKS | DATA_COMPRESSION }
```

- ***índice**:* es el nombre del índice que se va a crear.
- ***tabla/vista**:* es el nombre de una tabla/vista existente a la que estará asociado el índice.
- ***campo1, campo2**:* son los nombres de los campos que van a formar el índice. Puede ser un solo campo. Para forzar una ordenación descendente de un campo, hay que utilizar la palabra reservada **DESC**; sino se especifica o se escribe **ASC,** se entiende que la ordenación es ascendente.

CREATE INDEX: ejemplo

```
CREATE NONCLUSTER INDEX FK_Lineas  ON Lineas (codigart)
```

```
CREATE UNIQUE INDEX UX_CIF  ON Proveedores (cifpro)
```

```
CREATE NONCLUSTER INDEX i_Codpost  ON Proveedores (cpostpro)
  INCLUDE ( localpro)
```

- **UNIQUE:** señala que el índice es de unicidad, en caso contrario permitirá repetición de valores.
- **CLUSTER**: el orden de los valores de clave determinan el orden físico de las filas correspondientes de la tabla. NONCLUSTER, el orden lógico es independiente del físico.
- **INCLUDE**: columnas sin clave que se incluyen en el nivel hoja del índice nocluster.
- **WHERE**: valores de filas que serán indizados

Tabla de contenidos

Lenguaje de manipulación de datos

- Las IMD permiten actuar sobre los propios datos.
- Las operaciones básicas de manipulación de datos son: insertar, modificar, borrar y consultar.
  - Las tres primeras permiten alterar el contenido de la base de datos.
  - La última consiste en localizar datos para su observación.
- Principales instrucciones:
  - INSERT
  - UPDATE
  - DELETE
  - SELECT

INSERT

Permite añadir una o más filas en una tabla. La sintaxis para insertar una sola fila es:

INSERT INTO *destino* [(*campo1*[, *campo2*[, ...]])]
VALUES (*valor1*[, *valor2*[, ...]])

La sintaxis para insertar varias filas es:

INSERT INTO *destino* [(*campo1*[, *campo2*[, ...]])]
Instrucción_SELECT

- ***destino****:* es el nombre de la tabla o vista donde se van a añadir filas.
- ***campo1, campo2****:* son los nombres de los campos donde se van a añadir los datos.
- ***valor1, valor2****:* son los valores que tomarán los campos en la nueva fila que se van a insertar. Cada valor se asigna al campo que corresponde a la posición del valor en la lista, así, valor1 se asigna al campo1, valor2 al campo2, y así sucesivamente. Por su frecuencia, cabe mencionar que los valores que se asignen a campos de texto deben escribirse entre comillas simples ('texto').
- ***Instrucción_SELECT***: es la instrucción **SELECT** cuya ejecución proporcionará las filas a insertar.

UPDATE

Permite modificar información existente en una o varias filas de una tabla.
Su sintaxis es:

UPDATE *destino* SET *campo1=nuevovalor1* [, *campo2=nuevovalor2* [, ...]]

[WHERE *condición*]

- ***destino****:* es el nombre de la tabla o vista en la que se desea modificar datos.
- ***campo1, campo2****:* son los nombres de los campos que van a modificar su valor.
- ***nuevovalor1, nuevovalor2****:* son expresiones que determinan los valores que van a tomar los campos de las filas que se van a actualizar.
- ***condición****:* es una expresión lógica que determina qué filas se actualizarán. Sólo se actualizarán las filas que satisfagan la expresión. Si no se incluye cláusula WHERE (no hay condición) se modificarán todas las filas de la tabla.

DELETE

Permite eliminar una o varias filas de una tabla. Su sintaxis es:

DELETE FROM *destino* [WHERE *condición*]

- ***destino**:* es el nombre de la tabla o vista cuyas filas se van a eliminar.
- ***condición**:* es una expresión lógica que determina qué filas se borrarán. Sólo se borrarán las filas que satisfagan la expresión. Si no se incluye cláusula WHERE (no hay condición) se borrarán todas las filas de la tabla.

EJEMPLOS

codigpro	cifpro	nombrpro	direcpro	cpostpro	localpro	telefpro	faxpro	emailpro	procepro
P001	A39184215	Bau Pi, Pablo	Alta 3, 2º	39390	(34) 942 223 345	Santander	(34) 942 223 344	mailto:baupi@eresmas.es	UE
P002	A48162311	Zar Luna, Ana	Ercilla 22, 1º	48002	(34) 947 865 413	Bilbao	(34) 947 865 434	mailto:zarana@yahoo.es	UE
P003	B28373212	Gras León, Luz	Pez 14, 5º dcha.	28119	(34) 916 677 829	Madrid	(34) 916 677 889	NULL	UE
P004	B85392314	Gil Laso, Luis	Uría 18, 2º	85223	(34) 952 345 6632	Oviedo	(34) 952 345 678	NULL	UE

Insertar una nueva fila en la tabla **Proveedores.**

```
INSERT INTO Proveedores (codigpro, cifpro, nombrpro, direcpro, cpostpro, localpro, telefpro, faxpro, procepro)
VALUES ('P005', 'A39144325', 'Angulo Lastra, Antonio', 'Hernán Cortés,18', '39002', 'Santander', '(34) 942 202 022', '(34) 942 202 022', 'UE')
```

Incorporar el e-mail del proveedor Luis Gil Laso, con valor mailto:gil@unican.es.

```
UPDATE Proveedores  SET emailpro='mailto:gil@unican.es‘ WHERE codigpro='P004'
```

Borrar todos los proveedores de Santander.

```
DELETE FROM Proveedores  WHERE localpro='Santander'
```

INSERT con SELECT

• Incorporar a la tabla proveedores, los procedentes de la tabla temporal prov_tmp. Se considera que esta tabla tiene la misma estructura que la tabla proveedores.

```
INSERT INTO proveedores
  SELECT codigpro, cifpro, nombrpro, direcpro, cpostpro, localpro,
         telefpro, faxpro, procepro
  FROM prov_tmp
```

DATOS

codigpro	cifpro	nombrpro	direcpro	cpostpro	localpro	telefpro	faxpro	emailpro	procepro
P001	A39184215	Bau Pi, Pablo	Alta 3, 2º	39390	(34) 942 223 345	Santander	(34) 942 223 344	mailto:baupi@eresmas.es	UE
P002	A48162311	Zar Luna, Ana	Ercilla 22, 1º	48002	(34) 947 865 413	Bilbao	(34) 947 865 434	mailto:zarana@yahoo.es	UE
P003	B28373212	Gras León, Luz	Pez 14, 5º dcha.	28119	(34) 916 677 829	Madrid	(34) 916 677 889	NULL	UE
P004	B85392314	Gil Laso, Luis	Uría 18, 2º	85223	(34) 952 345 6632	Oviedo	(34) 952 345 678	NULL	UE

codigart	descrart	preunart	stockart	stockmin	fecbaja
0001	MESA OFICINA 90x1,80	225,00	100	1	NULL
0002	SILLA ERGONOMIC MOD. MX	120,00	25	1	NULL
0003	ARMARIO DIPLOMATIC	300,00	2	1	NULL
0004	ARCHIVADOR MOD. TR	180,00	3	1	NULL

numped	fechaped	codigpro	ivaped	fentrped
1	2010-05-22	P001	18.0	2010-06-16
2	2010-06-10	P002	18.0	2010-08-15
3	2010-10-15	P003	18.0	2010-12-15
4	2010-08-13	P001	18.0	2010-09-10

numped	numlin	codigart	unilin	preunlin	desculin	totallin
1	1	0001	1	220,00	0	220
1	2	0003	2	295,00	0	590
2	1	0002	3	120,00	2	352,8
2	2	0003	2	300,00	3	582
2	3	0002	5	120,00	0	600
3	1	0002	1	110,00	0	110
4	1	0002	4	120,00	0	480
4	2	0004	10	180,00	0	1800

SELECT

Está dedicada a obtener información de la base de datos. El resultado de su ejecución, si existe, siempre tiene estructura de una tabla y los campos de sus filas responden a la lista de selección. Tiene enormes posibilidades, lo que hace que su sintaxis presente muchas variantes.

SELECT [*predicado*] *Lista_de_selección*

[**INTO** *tabla_temporal*]

FROM *lista_de_tablas*

[**WHERE** *condición*]

[**GROUP BY** *lista_campos_group_by*]

[**HAVING** *condición_group_by*]

[**ORDER BY** *colum1* { [ASC] | DESC } [, *colum2* { [ASC] | DESC }, ...]]

SELECT (y 2)

❑ ***predicado***: puede tomar uno de los siguientes valores: **ALL**, **DISTINCT** o **TOP número_de_filas** (devuelve el número de registros especificado según una cláusula ORDER BY). Puede utilizar el predicado para limitar el número de registros devueltos. Si no especifica ninguno, el valor predeterminado es **ALL**.

❑ ***Lista_de_selección****:* es el conjunto de los elementos que serán aportados como respuesta. Éstos, pueden ser expresiones y funciones separados por comas, aunque generalmente responden a una de las siguientes alternativas:

{ * | *tabla*.* | [*tabla*.]*campo1* [AS *alias1*] [, [*tabla*.]*campo2* [AS *alias2*] , ...] | *funciones*}

- o * especifica que se seleccionan todos los campos de la tabla o tablas a las que se accede.
- o *tabla:* es el nombre de la tabla que contiene los campos de la que se van a seleccionar los registros.
- o *campo1, campo2:* son los nombres de los campos que contienen los datos que desea recuperar.
- o *alias1, alias2:* Los nombres que se van a utilizar como encabezados de columnas en vez de los nombres de columnas originales en tabla.
- o *funciones:* funciones definidas por el usuario, anteponer el propietario.

❑ ***tabla_temporal****:* es el nombre de la tabla que se creará para almacenar los registros obtenidos.

❑ ***lista_de_tablas****:* representa el nombre de la tabla o las tablas que contienen los datos a los que se desea acceder.

❑ ***condición****:* es una expresión lógica con los criterios de selección de registros.

❑ ***lista_campos_group_by****:* son los nombres de los campos que se van a utilizar para agrupar los registros.

❑ ***condición_group_by****:* son las condiciones que se imponen sobre el criterio de agrupamiento.

❑ ***colum1*, *colum2***: son nombres de elementos de la lista de selección o la posición que ocupan en ella. ASC quiere decir ordenación ascendente (es la opción por defecto) y DESC significa ordenación descendente.

SELECT: búsquedas sencillas

• Obtener el contenido de la tabla Articulos.

```
SELECT * FROM Articulos
```

codigart	descrart	preunart	stockart	stockmin	fecbaja
0001	MESA OFICINA 90x1,80	225,00	100	1	NULL
0002	SILLA ERGONOMIC MOD. MX	120,00	25	1	NULL
0003	ARMARIO DIPLOMATIC	300,00	2	1	NULL
0004	ARCHIVADOR MOD. TR	180,00	3	1	NULL

• Listar el nombre y el teléfono de todos los Proveedores.

```
SELECT nombrpro as Nombre, telefpro as Telf
FROM Proveedores
```

nombrpro	telefpro
Bau Pi, Pablo	(34) 942 223 345
Zar Luna, Ana	(34) 947 865 413
Gras León, Luz	(34) 916 677 829
Gil Laso, Luis	(34) 952 345 6632

SELECT: Búsquedas cualificadas: condiciones de comparación

Las condiciones de comparación son expresiones lógicas que permiten comparar una columna o expresión con otra columna, expresión o lista de columnas. Pueden adoptar una de las formas siguientes:

exp operador_de_comparación *exp*
exp [NOT] BETWEEN *exp* AND *exp*
exp [NOT] IN (lista de valores)
campo [NOT] LIKE 'cadena_de_caracteres'
campo IS [NOT] NULL

• Encontrar los artículos cuyo precio unitario sea superior a 180 € y su stock sea inferior o igual a 100.

```
SELECT *    FROM Articulos
WHERE preunart > 180 AND stockart <= 100
```

codigart	descrart	preunart	stockart	stockmin	fecbaja
0001	MESA OFICINA 90x1,80	225,00	100	1	NULL
0003	ARMARIO DIPLOMATIC	300,00	2	1	NULL

SELECT: Búsquedas cualificadas: condiciones de comparación

• Listar los artículos cuyo precio unitario esté comprendido entre 180 € y 300 €.

```
SELECT *    FROM Articulos
WHERE preunart BETWEEN 180 AND 300
```

codigart	descrart	preunart	stockart	stockmin	fecbaja
0001	MESA OFICINA 90x1,80	225,00	100	1	NULL
0003	ARMARIO DIPLOMATIC	300,00	2	1	NULL
0004	ARCHIVADOR MOD. TR	180,00	3	1	NULL

• Hallar todos los proveedores de las ciudades de Santander, Madrid y Barcelona.

```
SELECT codigpro, nombrpro, direcpro, cpostpro, localpro FROM Proveedores
WHERE localpro IN ('Santander', 'Madrid', 'Barcelona')
```

codigpro	nombrpro	direcpro	cpostpro	localpro
P001	Bau Pi, Pablo	Alta 3, 2º	39390	Santander
P003	Gras León, Luz	Pez 14, 5º dcha.	28119	Madrid

SELECT: Búsquedas cualificadas: condiciones de comparación

• Encontrar todos los proveedores cuyo primer apellido comience por una letra comprendida entre la A y la J.

```
SELECT codigpro, nombrpro, direcpro, cpostpro, localpro FROM Proveedores
WHERE nombrpro LIKE '[A-J]%'
```

codigpro	nombrpro	direcpro	cpostpro	localpro
P001	Bau Pi, Pablo	Alta 3, 2º	39390	Santander
P003	Gras León, Luz	Pez 14, 5º dcha.	28119	Madrid
P004	Gil Laso, Luis	Uría 18, 2º	85223	Oviedo

• Hallar todos los proveedores de los que no se tenga información sobre su correo electrónico.

```
SELECT codigpro, nombrpro, direcpro, cpostpro, localpro, telefpro
FROM Proveedores
WHERE emailpro IS NULL
```

codigpro	nombrpro	direcpro	cpostpro	localpro	telefpro
P003	Gras León, Luz	Pez 14, 5º dcha.	28119	Madrid	(34) 916 677 829
P004	Gil Laso, Luis	Uría 18, 2º	85223	Oviedo	(34) 952 345 6632

SELECT: Búsquedas cualificadas: condiciones de combinación simple

Las búsquedas cualificadas son las que afectan a datos de más de una tabla.

Una **Combinación Simple** es aquella en la que la condición de la cláusula **FROM** (o WHERE) contiene una comparación de igualdad entre campos pertenecientes a dos tablas distintas.

• Listar todos los proveedores a los que se ha efectuado algún pedido entre el 20/1/2006 y el 15/9/2006.

```
SELECT DISTINCT Proveedores.codigpro, nombrpro, direcpro, localpro
    FROM Proveedores INNER JOIN Pedidos
        ON Proveedores.codigpro = Pedidos.codigpro
    WHERE fechaped BETWEEN '20/01/2010' AND '15/09/2010'
```

SELECT: Búsquedas cualificadas: condiciones de combinación simple

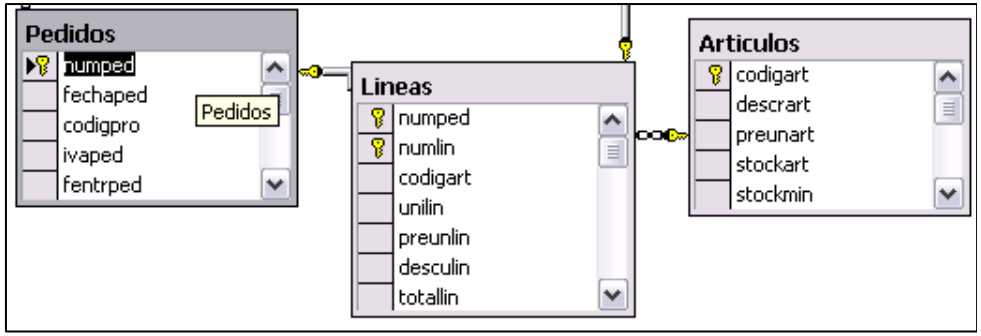
```
SELECT DISTINCT Proveedores.codigpro, nombrpro, direcpro, localpro
  FROM Proveedores INNER JOIN Pedidos
       ON Proveedores.codigpro = Pedidos.codigpro
  WHERE fechaped BETWEEN '20/01/2010' AND '15/09/2010'
```

codigpro	cifpro	nombrpro	direcpro	cpostpro	localpro	telefpro	faxpro	emailpro	procepro	numped	fechaped	codigpro	ivaped	fentrped
P001	A39184215	Bau Pi, Pablo	Alta 3, 2º	39390	Santander	(34) 94...	(34) 9...	mailto:ba...	UE	1	2010-05-22	P001	18.0	2010-06-16
P002	A48162311	Zar Luna, Ana	Ercilla ...	48002	Bilbao	(34) 94...	(34) 9...	mailto:za...	UE	1	2010-05-22	P001	18.0	2010-06-16
P001	A39184215	Bau Pi, Pablo	Alta 3, 2º	39390	Santander	(34) 94...	(34) 9...	mailto:ba...	UE	2	2010-06-10	P002	18.0	2010-07-16
P002	A48162311	Zar Luna, Ana	Ercilla ...	48002	Bilbao	(34) 94...	(34) 9...	mailto:za...	UE	2	2010-06-10	P002	18.0	2010-07-16
P001	A39184215	Bau Pi, Pablo	Alta 3, 2º	39390	Santander	(34) 94...	(34) 9...	mailto:ba...	UE	3	2010-10-15	P003	18.0	2010-12-15
P002	A48162311	Zar Luna, Ana	Ercilla ...	48002	Bilbao	(34) 94...	(34) 9...	mailto:za...	UE	3	2010-10-15	P003	18.0	2010-12-15
P003	B28373212	Gras León, Luz	Pez 14,...	28119	Madrid	(34) 91...	(34) 9...	NULL	UE	3	2010-10-15	P003	18.0	2010-12-15
P004	B85392314	Gil Laso, Luis	Uría 18...	85223	Oviedo	(34) 95...	(34) 9...	NULL	UE	3	2010-10-15	P003	18.0	2010-12-15
P001	A39184215	Bau Pi, Pablo	Alta 3, 2º	39390	Santander	(34) 94...	(34) 9...	mailto:ba...	UE	4	2010-08-13	P001	18.0	2010-09-10
P002	A48162311	Zar Luna, Ana	Ercilla ...	48002	Bilbao	(34) 94...	(34) 9...	mailto:za...	UE	4	2010-08-13	P001	18.0	2010-09-10
P003	B28373212	Gras León, Luz	Pez 14,...	28119	Madrid	(34) 91...	(34) 9...	NULL	UE	4	2010-08-13	P001	18.0	2010-09-10
P004	B85392314	Gil Laso, Luis	Uría 18...	85223	Oviedo	(34) 95...	(34) 9...	NULL	UE	4	2010-08-13	P001	18.0	2010-09-10

codigpro	nombrpro	direcpro	localpro
P001	Bau Pi, Pablo	Alta 3, 2º	Santander
P002	Zar Luna, Ana	Ercilla 22, 1º	Bilbao

SELECT: Búsquedas cualificadas: condiciones de combinación múltiple

Una **Combinación Múltiple** es aquella combinación que relaciona varios campos de más de dos tablas.



- Encontrar todos los artículos que han sido pedidos entre el 15/5/2010 y el 30/5/2010.

```
SELECT DISTINCT Articulos.codigart, descrart
    FROM Pedidos INNER JOIN
            (Lineas  INNER JOIN Articulos ON Lineas.codigart = Articulos.codigart)
                    ON Pedidos.numped = Lineas.numped
    WHERE fechaped BETWEEN '15/05/2010' AND '30/05/2010'
```

codigart	descrart
0001	MESA OFICINA 90x1,80
0003	ARMARIO DIPLOMATIC

SELECT: Búsquedas cualificadas: condiciones de combinación Autocombinación

Una **Autocombinación** es una combinación de una tabla con ella misma.

- Localizar todos los pedidos que tienen varias líneas del mismo artículo.

```
SELECT x.numped, x.numlin, x.codigart
FROM Lineas x, Lineas y
WHERE x.numped  = y.numped   AND   x.numlin <> y.numlin
          AND x.codigart = y.codigart
```

SELECT: Búsquedas cualificadas: condiciones de combinación Autocombinación

- Localizar todos los pedidos que tienen varias líneas del mismo artículo.

```
SELECT x.numped, x.numlin, x.codigart
FROM Lineas x, Lineas y
WHERE x.numped = y.numped AND x.numlin <> y.numlin
      AND x.codigart = y.codigart
```

Información de proceso

Acceso X

Acceso Y

Pedido num.	Linea num.	Código art.	Unidades	Precio compra	Descuento	Pedido num.	Linea num.	Código art.	Unidades	Precio compra	Descuento
1	1	0001	1,00	220,00 €		1	1	0001	1,00	220,00 €	
2	3	0002	5,00			2	3	0002	3,00	120,00 €	2,00
2	1	0002	3,00			2	3	0002	5,00	120,00 €	
2	3	0002	5,00			2	1	0002	3,00	120,00 €	2,00
2	1	0002	3,00			2	1	0002	2,00	300,00 €	3,00
2	3	0002	5,00			2	2	0003	2,00	300,00 €	3,00
2	1	0002	3,00	120,00 €	2,00	2	3	0002	5,00	120,00 €	
2	2	0003	2,00	300,00 €	3,00	2	3	0002	5,00	120,00 €	
4	2	0004	10,00	180,00 €		4	1	0002	4,00	120,00 €	
4	1	0002	4,00	120,00 €		4	2	0004	10,00	180,00 €	
2	3	0002	5,00	120,00 €		2	3	0002	5,00	120,00 €	
3	1	0002	1,00	110,00 €		3	1	0002	1,00	110,00 €	
4	1	0002	4,00	120,00 €		4	1	0002	4,00	120,00 €	
4	2	0004	10,00	180,00 €		4	1	0002	4,00	120,00 €	
4	1	0002	4,00	120,00 €		4	2	0004	10,00	180,00 €	
4	2	0004	10,00	180,00 €		4	2	0004	10,00	180,00 €	

Pedido num.	Linea num.	Código art.
2	3	0002
2	1	0002

SELECT: Búsquedas cualificadas: condiciones de combinación
Combinación externa

Una **Combinación Externa** es aquella que da preferencia a una tabla con respecto a otra. Así, las filas de la tabla dominante serán seleccionadas aunque la condición de enlace no se haya verificado.

• Listar todos los proveedores indicando, en su caso, los que han recibido algún pedido.

```
SELECT Proveedores.codigpro, nombrpro, Pedidos.numped
   FROM Proveedores LEFT JOIN Pedidos
      ON  Proveedores.codigpro = Pedidos.codigpro
```

Preferencia a la tabla de la izquierda

	codigpro	nombrpro	numped
1	P001	Bau Pi, Pablo	1
2	P001	Bau Pi, Pablo	4
3	P002	Zar Luna, Ana	2
4	P003	Gras León, Luz	3
5	P004	Gil Laso, Luis	NULL

```
SELECT Proveedores.codigpro, nombrpro, Pedidos.numped
   FROM Pedidos RIGHT JOIN Proveedores
      ON  Proveedores.codigpro = Pedidos.codigpro
```

Preferencia a la tabla de la derecha

SELECT: Búsquedas cualificadas: condiciones de combinación
Combinación externa

En el **FROM** de una **Consulta** pueden establecerse predicados. Estos se aplican antes de ejecutar el join, mientras que cuando se establecen en el **WHERE** se ejecutan al resultado del join.
El resultado puede ser distinto si la condición de join es **LEFT** o **RIGHT**.

• Listar todos los proveedores de Santander indicando, en su caso, los que han recibido algún pedido.

```
SELECT Proveedores.codigpro, nombrpro, Pedidos.numped
   FROM Pedidos LEFT JOIN Proveedores
      ON  Proveedores.codigpro = Pedidos.codigpro
       AND Localpro='SANTANDER'
```

	codigpro	nombrpro	numped
1	P001	Bau Pi, Pablo	1
2	P001	Bau Pi, Pablo	4
3	NULL	NULL	2
4	NULL	NULL	3

```
SELECT Proveedores.codigpro, nombrpro, Pedidos.numped
   FROM Pedidos LEFT JOIN Proveedores
      ON  Proveedores.codigpro = Pedidos.codigpro
   WHERE Localpro='SANTANDER'
```

	codigpro	nombrpro	numped
1	P001	Bau Pi, Pablo	1
2	P001	Bau Pi, Pablo	4

SELECT: Búsquedas cualificadas: condiciones de subsentencias

La instrucción **SELECT** permite contrastar una expresión o un campo con el resultado de otra instrucción (subsentencia) **SELECT**. A este contraste se le llama condición de subsentencia y las dos instrucciones se llaman instrucciones imbricadas. A su vez, la subsentencia puede incluir en su condición a otra subsentencia y así sucesivamente. Las condiciones de subsentencia pueden adoptar una de las formas siguientes:

exp operador_de_comparación {ALL | [ANY | SOME] } (instrucción SELECT)

exp [NOT] IN (instrucción SELECT)

[NOT] EXISTS (instrucción SELECT)

- Encontrar los artículos cuyo stock es mayor que toda cantidad pedida del mismo artículo.

```
SELECT articulos.codigart, descrart, stockart FROM Articulos
    WHERE stockart > ALL (SELECT unilin   FROM Lineas
                                  WHERE Articulos.codigart = Lineas.codigart)
```

	codigart	descrart	stockart
1	0001	MESA OFICINA 90x1,80	100
2	0002	SILLA ERGONOMICA MOD.MX	25

SELECT: Búsquedas cualificadas: condiciones de subsentencias

• Listar los artículos que no han sido pedidos entre el 24 de Septiembre y el 21 de Noviembre de 2006.

```
SELECT DISTINCT Articulos.codigart, descrart   FROM Articulos
   WHERE Articulos.codigart NOT IN (SELECT Lineas.codigart FROM Lineas, Pedidos
                                    WHERE Pedidos.numped = Lineas.numped
                                    AND Pedidos.fechaped
                                    BETWEEN '24/09/2010' AND '21/11/2010')
```

	codigart	descrart
1	0001	MESA OFICINA 90x1,80
2	0003	ARMARIO DIPLOMATIC
3	0004	ARCHIVADOR MOD.TR

• Encontrar los proveedores de Madrid a los que se les ha realizado algún pedido entre el 24/09/2006 y el 21/11/2006 .

```
SELECT DISTINCT Proveedores.codigpro, nombrpro FROM Proveedores
   WHERE  EXISTS  ( SELECT *    FROM Pedidos
                                 WHERE Proveedores.codigpro = Pedidos.codigpro
AND fechaped BETWEEN '24/09/2010' AND '21/11/2010') AND localpro = 'Madrid'
```

	codigpro	nombrpro
1	P003	Gras León, Luz

FUNCIONES DE GRUPO

Con las filas de la información de proceso correspondiente a una instrucción **SELECT** se pueden establecer grupos.

En cada uno de estos grupos, mediante las funciones de grupo, se pueden efectuar ciertos cálculos.

COUNT(*)	Nº de filas que componen el grupo.
COUNT(campo)	Nº de filas con valor asignado al campo (nulos no cuentan).
SUM(exp)	Suma de valores obtenidos con la expresión en cada fila.
AVG(exp)	Media.
MAX(exp)	Máximo.
MIN(exp)	Mínimo.
STDEV(exp)	Desviación típica.
VAR(exp)	Varianza.

• Encontrar cuántos artículos hay registrados, el máximo y el mínimo precio unitario, el precio unitario medio y la valoración del almacén.

```
SELECT  COUNT(codigart) AS Cantidad,  MAX(preunart) AS Max,
        MIN(preunart) AS Min,  AVG(preunart) AS Precio_medio,
        SUM(preunart*stockart) AS Valoración
    FROM Articulos
```

Cantidad	Max	Min	Precio_medio	Valoración
4	300,00	120,00	206,25	26640,00

FUNCIONES DE FECHA

Los datos de tipo fecha son almacenados como una unidad de información. Cuando se necesita trabajar con componentes de una fecha, es preciso utilizar funciones de fecha.

DAY(fecha**)**	Devuelve el día de mes de fecha.
MONTH (fecha**)**	Devuelve el mes de fecha.
YEAR (fecha**)**	Devuelve el año de fecha.
DATEPART(dw,fecha**)**	Devuelve el día de la semana correspondiente a fecha (el 1 → domingo, el 2 → lunes, ...).

• Listar día, mes y año de cada pedido, así como el día de la semana al que corresponden sus fechas.

```
SELECT numped, fechaped,
DAY(fechaped) as dia_mes, MONTH(fechaped) as mes,
YEAR(fechaped) as año, DATEPART(dw,fechaped) as dia_sem
FROM Pedidos
```

numped	fechaped	dia_mes	mes	año	dia_sem
1	2010-05-22	22	5	2010	6
2	2010-06-10	10	6	2010	4
3	2010-10-15	15	10	2010	5
4	2010-08-13	13	8	2010	5

SELECT: Agrupamiento de datos

La cláusula **GROUP** permite formar grupos con las filas de datos que tengan valores iguales para determinados campos.

La respuesta tiene tantas filas como grupos haya establecido la instrucción.

- Obtener el importe de cada pedido sin aplicar el IVA.

```
SELECT numped, SUM((preunlin*unilin)*(1-desculin/100)) as Importe
    FROM Lineas
    GROUP BY numped
```

numped	Importe
1	810
2	1534,8
3	110
4	2280

SELECT: Agrupamiento de datos. Condición de agrupamiento

• Listar el importe, sin aplicar el IVA, de los pedidos que tienen más de una línea.

```
SELECT numped, SUM((preunlin*unilin)*(1-desculin/100)) as Importe
    FROM Lineas
    GROUP BY numped
    HAVING COUNT(*) > 1
```

numped	Importe
1	810
2	1534,8
4	2280

SELECT: Ordenación de resultado

[**ORDER BY** *colum1* { [ASC] | DESC } [, *colum2* { [ASC] | DESC }, ...]]

- *colum1, colum2, ...:* son nombres de elementos (campos, expresiones o funciones) de la lista de selección o la posición que ocupan en ella.
- ASC: quiere decir ordenación ascendente (opción por defecto) y DESC descendente.

• Listar los valores de los campos que componen el índice de unicidad de la tabla *Lineas* (*numped*, *numlin*), por orden decreciente de nº de pedido y de nº de línea.

```
SELECT numped, numlin
    FROM Lineas
    ORDER BY 1 DESC, 2 DESC
```

	numped	numlin
1	4	2
2	4	1
3	3	1
4	2	3
5	2	2
6	2	1
7	1	2
8	1	1

SELECT: Guardar el resultado en tabla

INTO *tabla_adicional*

- *tabla_adicional*: es el nombre de la tabla a generar, que ha de ser único. Esta tabla hereda el esquema de la lista de selección (nombres campos, tipos de datos, ...).

• Crear una tabla temporal, llamada **t1,** para guardar el importe, sin aplicar el IVA, de los pedidos que tienen más de una línea.

```
SELECT numped, SUM((preunlin*unilin)*(1-desculin/100)) as Importe
    INTO #t1
    FROM Lineas
    GROUP BY numped
    HAVING COUNT(*) > 1
```

SELECT: Unión de sentencias

Permite combinar los resultados de dos o más consultas. Para ello se requiere el operador **UNION [ALL]**.

• Listar todos los proveedores de las tablas Proveedores y Proveedores_Anulados ordenados por su código.

```
SELECT codigpro, nombrpro, localpro
      FROM Proveedores
  UNION
SELECT codigp, nombrp, localp
      FROM Proveedores_Anulados
  ORDER BY 1
```

SELECT: Intersección y excepción de conjuntos de datos

EXCEPT Devuelve los valores distintos de la primera consulta que no son devueltos por la segunda consulta.
INTERSECT Devuelve los distintos valores que son devueltos por ambas consultas.

- Listar los clientes exceptuando aquellos que son también Proveedores.

```
SELECT codigcli, nombrcli, localcli
        FROM clientes
EXCEPT
SELECT codigpro, nombrpro, localpro
        FROM Proveedores
```

UPDATE y DELETE con SELECT

- Actualizar los precios en un 2% de los artículos del grupo Bebidas.

```
UPDATE articulo
SET preunart = preunart * 1.02
WHERE tipo_id IN
   (SELECT tipo_id
   FROM TipoArticulo
   WHERE tipo_nombre = 'Bebidas')
```

- Borrar los pedidos de los proveedores no comunitarios

```
DELETE FROM pedidos
FROM pedidos INNER JOIN
    proveedores ON proveedores.codigpro = pedidos.codigpro
WHERE   procepro = 'No UE'
```

SELECT y UPDATE con CASE

• Listar los artículos mostrando su precio categorizado por alto, bajo y no establecido.

```
SELECT    'precio' =
      CASE
        WHEN preunart IS NULL THEN 'No establecido'
        WHEN preunart < 200 THEN 'Bajo'
      ELSE 'Alto'
      END,
   descrart
FROM articulos
```

	precio	descrart
1	Alto	MESA OFICINA 90x1,80
2	Bajo	SILLA ERGONOMICA MOD.MX
3	Alto	ARMARIO DIPLOMATIC
4	Bajo	ARCHIVADOR MOD.TR

• Actualizar los artículos con precio < 10 con 5% y los > =10 con un 7%

```
UPDATE    articulos SET preunart=
      CASE
        WHEN preunart <10 THEN preunart*1.05
      ELSE preunart*1.07
      END
```

Tabla de contenidos

VISTAS

Para crear una vista en la base de datos. Su sintaxis es:

CREATE VIEW [< *nombreBD* > .] [< *propietario* > .] *nombre* [(*campo* [,...n])]
[WITH <*view_attribute*> [,...n]]
AS
Instrucción_Select
[WITH CHECK OPTION]

- ***nombreBD****:* es el nombre de la base de datos en la que se crea.
- ***propietario***: cuenta de usuario que crea la vista
- ***nombre****:* es el nombre de la vista que se va a crear.
- ***campo****: e*s el nombre que se va a utilizar para una columna en una vista.
- ***instrucción_Select:*** consulta a través de la cuál se define la vista
- ***view_attribute:*** toma uno de los siguientes valores

[ENCRYPTION]: evita que la vista se publique como parte de la réplica de SQL Server

[SCHEMABINDING]: enlaza la vista al esquema de las tablas subyacentes. Cuando se especifica, las tablas base no se pueden modificar de una forma que afecte a la definición de la vista.

[VIEW_METADATA]: Especifica que la instancia de SQL Server devolverá a las API de DB-Library, ODBC y OLE DB la información de metadatos sobre la vista en vez de las tablas base.

[WITH CHECK OPTION] : Exige que todas las instrucciones de modificación de datos ejecutadas contra la vista se adhieran a los criterios establecidos en la instrucción Select.

VISTAS: ejemplo

```
CREATE VIEW dbo.EncabezadoPedido
AS
SELECT     dbo.Pedidos.numped, dbo.Pedidos.fechaped, dbo.Pedidos.codigpro,
dbo.Pedidos.ivaped, dbo.Pedidos.fentrped, dbo.Proveedores.nombrpro,
dbo.Proveedores.direcpro, dbo.Proveedores.cpostpro, dbo.Proveedores.localpro,
dbo.Proveedores.telefpro, dbo.Proveedores.faxpro,
dbo.Proveedores.procepro, dbo.Proveedores.emailpro, dbo.Proveedores.cifpro
FROM       dbo.Proveedores INNER JOIN
                dbo.Pedidos ON dbo.Proveedores.codigpro = dbo.Pedidos.codigpro
```

VISTAS ACTUALIZABLES Y MATERIALIZADAS

Vista actualizable si:

• El *SELECT* no tiene ninguna expresión de valor agregado ni especificación de DISTINCT

• Cualquier atributo que no aparezca en la cláusula SELECT puede definirse como nulo

• La consulta no tiene cláusulas GROUP BY ni HAVING

• Cualquier modificación, UPDATE, INSERT y DELETE, debe hacer referencia a las columnas de una única tabla base

• Las columnas que se van a modificar no se ven afectadas por las cláusulas GROUP BY, HAVING o DISTINCT.

Vista materializada (Indexed views SCHEMABINDING)

• La vista es computada y almacenada.

• Se crea al definir un índice cluster sobre ella.

• Son adecuadas para consultas frecuentes sobre datos agregados sobre muchas filas.

• No se aconsejan para datos que cambien frecuentemente.

VISTAS MATERIALIZABLE: EJEMPLO

```
CREATE VIEW UdsPedidas WITH SCHEMABINDING as
SELECT  A.descrart, sum(unilin) as unidadesPedidas, count_big(*) as nro_orden
FROM dbo.articulos as A inner join dbo.lineas as L on A.codigart=L.codigart
GROUP BY  A.descrart;

CREATE UNIQUE CLUSTERED INDEX cix_MiVistaMaterializada
   ON dbo.UdsPedidas(descrart);
```

Tabla de contenidos

PROCEDIMIENTOS ALMACENADOS

Para crear un procedimiento en la base de datos. Su sintaxis es:

```
CREATE PROC [ EDURE ] nombre
    [ { @parametros tipo_dato }
        [ = valor_por_defecto ] [ OUTPUT ]
    ] [ ,...n ]
[ WITH
    { RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]

[ FOR REPLICATION ]

AS instrucciones_sql [ ...n ]
```

- ***nombre***: es el nombre del procedimiento que se va a crear.
- ***parámetros***: parámetros de entrada y salida del procedimiento
- ***tipo_dato***: tipo de dato asociado al parámetro.
- ***valor_por_defecto***: valor asignado por defecto al parámetro
- ***instrucciones_sql:*** instrucciones SQL

PROCEDIMIENTOS ALMACENADOS: Ejemplo sin parámetros

```
CREATE PROCEDURE dameProveedores    AS

SELECT codigpro, nombrpro, direcpro, cpostpro, localpro, telefpro
FROM Proveedores;
Go


exec dameProveedores;
```

PROCEDIMIENTOS ALMACENADOS: Ejemplo con parámetros

```
CREATE PROCEDURE upd_precio_articulo   @ipc numeric(3,2)    AS

BEGIN TRANSACTION

  update articulos set preunart = preunart + (preunart*@ipc/100)
       where preunart is not null

  if @@ERROR <> 0
      begin
            ROLLBACK TRANSACTION
            RAISERROR ( 'No se han modificado los precios',16,1)
            RETURN
      end

COMMIT TRANSACTION
Go


exec upd_precio_articulo 3.2
```

PROCEDIMIENTOS ALMACENADOS: Ejemplo con parámetros. Otra gestión de errores

```
CREATE PROCEDURE upd_precio_articulo   @ipc numeric(3,2)    AS

BEGIN TRY
      BEGIN TRANSACTION

        update articulos set preunart = preunart + (preunart*@ipc/100)
              where preunart is not null

      COMMIT TRANSACTION
END TRY

BEGIN CATCH
  ROLLBACK TRANSACTION
  RAISERROR ( 'No se han modificado los precios',16,1)
END CATCH

RETURN

exec upd_precio_articulo 3.2
```

Gestión de errores

Procedimiento para mostrar la excepción.

```
CREATE PROCEDURE usp_showerrorinfo
AS
SELECT ERROR_NUMBER() AS [Numero de Error],
  ERROR_STATE() AS [Estado del Error],
  ERROR_SEVERITY() AS [Severidad del Error],
  ERROR_LINE() AS [Linea],
  ISNULL(ERROR_PROCEDURE(), 'No esta en un proc') AS [Procedimiento],
  ERROR_MESSAGE() AS [Mensaje]
GO

BEGIN TRY
   SELECT 1/0
END TRY
BEGIN CATCH
   EXEC usp_showerrorinfo
END CATCH
```

FUNCIONES ALMACENADAS: Valor escalar

Para crear una función escalar en la base de datos. Se pueden usar en el SELECT. Su sintaxis es:

```
CREATE FUNCTION [ propietario. ] nombre
   ( [ { @parametro [AS] tipo_dato [ = valor_por_defecto ] } [ ,...n ] ] )

RETURNS tipo_dato_valor_retorno
AS

BEGIN
   --cuerpo de la función, TSQL
   RETURN valor_retorno
END
```

- ***propietario***: cuenta de usuario que crea la función
- ***nombre****:* es el nombre de la función que se va a crear.
- ***parámetro****:* parámetros de entrada de la función
- ***tipo_dato****:* tipo de dato asociado a cada parámetro
- ***valor_por_defecto***: valor asignado por defecto al parámetro
- ***tipo_dato_valor_retorno****:* tipo de dato asociado al valor de retorno.
- ***valor_retorno***: valor de retorno de la función

FUNCIONES ALMACENADAS: Valor escalar. Ejemplo

```
CREATE FUNCTION Calcular_Pedido   (@codigo int)
RETURNS decimal (10,2)
AS
BEGIN

  DECLARE @precio money
  DECLARE @iva float

  SELECT @precio= sum(totallin) from lineas WHERE numped=@codigo

  SELECT @iva=ivaped from pedidos WHERE numped=@codigo

  SET @precio= (@precio* (@iva/100))+@precio

  RETURN @precio

END
GO
-- Ejecución
SELECT dbo.Calcular_Pedido (1)
```

FUNCIONES ALMACENADAS: Tabla

Para crear una función que devuelve una tabla en la base de datos. Se pueden usar en el SELECT Su sintaxis es:

```
CREATE FUNCTION [ propietario. ] nombre
   ( [ { @parametro [AS] tipo_dato [ = valor_por_defecto ] } [ ,...n ] ] )

RETURNS @variable_retorno TABLE < definicion_tabla >
AS

BEGIN
   --cuerpo de la función, TSQL
   RETURN
END
```

- ***propietario***: cuenta de usuario que crea la función
- ***nombre****:* es el nombre de la función que se va a crear.
- ***parámetro****:* parámetros de entrada de la función
- ***tipo_dato****:* tipo de dato asociado a cada parámetro
- ***valor_por_defecto***: valor asignado por defecto al parámetro
- ***tipo_dato_valor_retorno****:* tipo de dato asociado al valor de retorno.
- ***variable_retorno***: variable de retorno de la función
- ***definicion_tabla:*** definición de la tabla que devuelve la función

FUNCIONES ALMACENADAS: Tabla. Ejemplo

```
CREATE FUNCTION PedidosPorProveedor   (@codigo char(4))
RETURNS TABLE
AS

RETURN (SELECT count(numped) numero, nombrpro
     FROM pedidos p, proveedores pr
     WHERE pr.codigpro=@codigo and
               p.codigpro = pr.codigpro
     GROUP BY nombrpro)

GO

-- Ejecución
SELECT * from dbo.PedidosPorProveedor('0010')
```

FUNCIONES ALMACENADAS: Tabla. Ejemplo

```
CREATE FUNCTION PendientePago   (@fecha datetime)
RETURNS @Pagos TABLE (numped int primary key,
                      cantidad float NOT NULL,
                      fechaped datetime)
AS  BEGIN

   INSERT @Pagos
   SELECT numped,  dbo.calcular_pedido(numped), fechaped
   FROM pedidos
   WHERE fecpago is null or fecpago>=@fecha
   RETURN

END
GO

-- Ejecución
SELECT * from dbo.PendientePago(getdate())
```

FUNCIONES ALMACENADAS: uso con SELECT

• Obtener el precio y la descripción de los artículos haciendo uso de la función dame_precio_articulos(@codigo_art).

```
SELECT codigart as Codigo, descrart as Descripcion,
    dbo.dame_precio_articulo(codigart) as Precio
FROM ARTICULOS
```

Codigo	Descripcion	Precio
0001	MESA OFICINA 90x1,80	225.0000
0002	SILLA ERGONOMICA MOD.MX	120.0000
0003	ARMARIO DIPLOMATIC	300.0000
0004	ARCHIVADOR MOD.TR	180.0000

• Obtener los artículos haciendo uso de la función dame_precio_articulos cuyo precio sea superior a 190 €

```
SELECT codigart as Codigo, descrart as Descripcion, preunart as Precio
  FROM ARTICULOS
  WHERE dbo.dame_precio_articulo(codigart) > 190
```

Codigo	Descripcion	Precio
0001	MESA OFICINA 90x1,80	225.0000
0003	ARMARIO DIPLOMATIC	300.0000

Tabla de contenidos

BD activas. Disparadores

- Los **triggers DML** son procesos predefinidos que entran en acción en respuesta a eventos específicos de manipulación de datos (insert, update, delete).

- Generalmente se utilizan para:
  - recoger restricciones complejas
  - automatizar procesos
  - anotar acciones (log)

- En SQL Server 2008:
  - se pueden definir varios triggers para el mismo evento (definir orden de ejecución sp_settriggerorder)
  - sólo puede haber un trigger INSTEAD OF por tipo de operación
  - crean tablas inserted y deleted
  - !Ojo en su programación¡
    - pueden afectar a una o varias filas
    - se ejecutan transaccionalmente, si hay error se debe gestionar el ROLLBACK

DISPARADORES DML

Para crear un desencadenador en una tabla de la base de datos. Su sintaxis es:

```
CREATE TRIGGER nombre
ON { tabla | vista }
{
   { { FOR  | AFTER | INSTEAD OF } { [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
      [ NOT FOR REPLICATION ]
      AS
      [ { IF UPDATE ( campo )
         [ { AND | OR } UPDATE (campo ) ]
            [ ...n ]
      } ]
      instrucciones_sql [ ...n ]
   }
}
```

- ***nombre**:* es el nombre del desencadenador que se va a crear.
- ***tabla/vista**:* es el nombre de una tabla/vista sobre la que se crea.
- ***campo**:* campo de la tabla o vista afectada por el desencadenador .
- ***instrucciones_sql:*** reglas de negocio que se requieren especificar por medio de SQL

DISPARADOR DML: ejemplo

No pedir un artículo descatalogado

```
CREATE  TRIGGER tr_lineas ON dbo.Lineas AFTER  INSERT, UPDATE
AS  BEGIN

DECLARE @errmsg  char(255)

      IF ( SELECT count(*) FROM inserted i, articulos a
        WHERE i.codigart=a.codigart and a.fecbaja is not null) >0

       BEGIN
            set @errmsg = 'No puede seleccionar un artículo descatalogado'
            RAISERROR (  @errmsg,16,1)
            ROLLBACK TRANSACTION
            RETURN
       END

END
```

DISPARADOR DML: ejemplo

Notificar si el stock alcanza el mínimo establecido

```
CREATE TRIGGER tr_articulos ON articulos
FOR  UPDATE
AS BEGIN

if update(stockart) BEGIN
   DECLARE @v_codigart char(6)
   DECLARE @v_texto varchar(200)

   DECLARE @cursor_insert CURSOR
   SET @cursor_insert = CURSOR FOR
   SELECT codigart FROM inserted
      WHERE stockart<=stockmin
   OPEN  @cursor_insert
   FETCH NEXT FROM @cursor_insert INTO @v_codigart

   WHILE @@FETCH_STATUS=0
        BEGIN
              set @v_texto='Stock minimo alcanzado en articulo ' + @v_codigart
              insert into eventos (fecha, motivo) values (getdate(), @v_texto)
         FETCH NEXT FROM @cursor_insert INTO @v_codigart
        END
END
CLOSE @cursor_insert
DEALLOCATE @cursor_insert
END
```

	id	fecha	motivo
1	1	2007-03-27 12:20:48.903	Stock minimo alcanzado en articulo 0004
2	2	2007-03-27 12:20:48.903	Stock minimo alcanzado en articulo 0003

DISPARADOR DML: ejemplo

Notificar si el stock alcanza el mínimo establecido

```
CREATE TRIGGER tr_articulos ON articulos
FOR  UPDATE
AS BEGIN

if UPDATE(stockart) BEGIN

   INSERT INTO eventos (fecha, motivo)
   SELECT getdate(), 'Stock minimo alcanzado ' + codigart FROM inserted
      WHERE stockart<=stockmin

 END
END
```

	id	fecha	motivo
1	1	2007-03-27 12:20:48.903	Stock minimo alcanzado en articulo 0004
2	2	2007-03-27 12:20:48.903	Stock minimo alcanzado en articulo 0003

¡¡¡¡ **Mejor evitar el uso de cursores** !!!
Código más eficiente → trabaja el motor de almacenamiento y el optimizador para el conjunto de filas y no para cada una

DISPARADOR DML: ejemplo

Realizar borrados lógicos

```
CREATE TRIGGER trd_articulos on articulos
INSTEAD OF DELETE
AS
IF @@ROWCOUNT = 0
  RETURN

UPDATE articulos SET fecbaja = getdate()
  FROM articulos JOIN deleted d ON d.codigart = articulos.codigart
```

DISPARADORES, FUNCIONES Y PROCEDIMIENTOS

APLICACIONES:

- ❑ **Implementación de reglas de integridad complejas**
- ❑ **Gestión de Log (auditoría, control de cambios)**
- ❑ **Flujos de trabajo o proceso (actualizaciones, datos derivados, tablas resumen, etc.)**

VENTAJAS

Semántica del problema en un solo sitio → integridad
Facilita la construcción de aplicaciones
→ Procedimientos y funciones almacenados
- lógica de negocio compartida por aplicaciones
- reutilización de código
- seguridad de acceso a los usuarios
- reducción del tráfico de red
- mantenimiento más sencillo

INCONVENIENTES

Deben escribirse con cuidado → ejecución infinita de disparos
Control complejo de condiciones
Mayor carga computacional del servidor