Transact SQL

Antonio Espín Herranz

Contenidos

- Sintaxis y convenciones de T-SQL.
- Tipos de datos.
- Operadores.
- Variables.
- Funciones.
- Funciones definidas por el usuario.
- Más instrucciones de T-SQL.
- Definición mediante DDL.
- Vistas.
- Procedimientos avanzados.
- Triggers.
- Gestión de Excepciones
- Transacciones.
- Cursores.
- Apéndice.

Sintaxis y convenciones de T-SQL

- **Comentarios**:
  /* Varias líneas */ o -- En una línea
  - Son aconsejables en nuestros scripts de código para aclaraciones.

- **Valores Constantes:**
  - Los números se escriben tal cual.
  - Fechas y cadenas entre comillas simples.

Sintaxis y convenciones de T-SQL

- Referencia a objetos.
  - Tenemos 4 posibles partes de un objeto.
  - El nombre del Servidor que contiene el objeto.
  - El nombre del propietario del objeto.
  - El identificador del objeto.

- Por ejemplo, tenemos un servidor que se llama Server1 con una base de datos que se llama BdMaestra y que contiene un objeto llamado clientes que es propiedad del usuario dbo.

- La forma de referenciarla sería:
  - Server1.BdMaestra.dbo.Clientes

Ejemplo en un Select

- select distinct 'De AdventureWorks' as concepto, AdventureWorks.Person.Address.City from **AdventureWorks**.Person.Address

  UNION select 'De AdventureWorksDW' as concepto, **AdventureWorksDW**.dbo.DimEmployee.FirstName from AdventureWorksDW.dbo.DimEmployee

- Son dos BD distintas.

Sintaxis y convenciones de T-SQL

- Las palabras reservadas del lenguaje no se pueden utilizar como nombre de funciones, variables, etc.

- Cualquier identificador debe de cumplir las siguientes reglas:
  - El primer carácter debe ser una letra: a-z, A-Z, _, @ (sólo para variables), # (tablas temporales, procedimientos.
  - Siguientes caracteres: letras, números o los símbolos $, @, #, _.
  - Los identificadores no pueden tener espacios en blanco, si se define alguno lo tenemos que utilizar de esta manera:
    - Select * from “Nuevos Clientes”
    - Select * from [Nuevos Clientes]

Tipos de datos

- **ENTEROS** (clasificados de mayor capacidad a menor capacidad)

BIGINT	8 BYTE	
INT	4 BYTE	
SMALLINT	2 BYTE	-32.768 A 32.767
TINYINT	1 BYTE	VALORES DE 0 A 255
BIT	Datos enteros con 0 o 1 o NULL (Valor desconocido).	

- El que más se suele usar es el INT, pero depende del tamaño de los números a almacenar.

- Cuando estamos diseñando tablas por optimizar es bueno elegir el tipo de datos que mas se ajuste al rango de valores que vamos a representar. Consultas con tipos de datos mas pequeños se ejecutarán más rápido.

Tipos de datos

- **DECIMAL Y NUMERIC**
  - Para trabajar con números en coma flotante.
  - Tiene dos nombres distintos.
  - Se define mediante dos parámetros:
    - Precisión: Número total de dígitos.
    - Escala: Es el número de decimales.
    - Ejemplo: decimal(5,3) puede almacenar números del estilo: 12,345
    - La máxima escala son 38.

Tipos de datos

- **MONEY Y SMALLMONEY**
  - Tipos para almacenar valores de moneda.
- **NUMÉRICOS EN COMA FLOTANTE.**
  - Estos se pueden redondear si no pueden representar.
  - FLOAT: precisión de 1 a 53.
  - REAL: precisión de 1 a 24.
- **DATETIME Y SMALLDATETIME**
  - Dependiendo del número de años que vayamos a cubrir
  - Datetime: de 1753 a 9999.
  - Smalldatetime: 1900 a 2079.

Tipos de datos

De texto:

- **CHAR(n)**: Número **fijo** de caracteres no Unicode. Si decimos char(30), siempre se ocuparán los 30. Máximo 8000.

- **VARCHAR(n)**: Número **variable** de caracteres no Unicode. Hasta 8000. Utilizándolo varchar(max) serán hasta 2^31 caracteres.

- **NCHAR(n):** Número **fijo** de caracteres Unicode. Cada carácter ocupa 2 bytes, como máximo 4000.

- **NVARCHAR(n):** Número **variable** de caracteres Unicode. Indicamos en N el número de caracteres. De longitud variable. Hasta 4000. Utilizándolo nvarchar(max) serán hasta 2^31 caracteres.

Tipos de datos

- **Datos Binarios**:
  - Binary(n): pueden ser hasta 8000 bytes. De tamaño fijo.
  - varBinary(n): De tamaño variable. Hasta 8000 bytes, utilizándola varBinary(max) → 2^31 bytes.

Tipos de datos

- **ESPECIALES**:
  - **Cursor**: No se puede utilizar para definir una columna de SQL Server. Puede ser la salida de un procedimiento, devuelve un puntero a un conjunto de registros.
  - **Sql_Variant**: Equivalente a variant de los lenguajes de programación, puede almacenar cualquier tipo de dato menos varchar(max), nvarchar(max), timestamp y sql_variant. El procesamiento es lento, pero ofrece mucha flexibilidad.

Tipos de datos

- **ESPECIALES**:
  - **Table**: Almacenamiento temporal de un conjunto de resultados durante una función o procedimiento almacenado.
    - Ejemplo de uso:

```
Declare @T1 TABLE
(PK int primary key, col2 varchar(3))
Insert into @T1 values (2, 'xxxx')
Insert into @T1 values(3, 'yyyy')
Select * from T1
```

Tipos de datos

- **ESPECIALES**:
  - **Timestamp**: Columna binaria de 8 bytes que contiene un único valor generado por SQL Server, sirve para saber si se han producido cambios por otro usuario en los campos fecha de esa fila.
  - **Uniqueidentifier**: Identificador único generado por SQL Server, pero a nivel global.
  - **Xml**: Se puede almacenar el contenido de un fichero XML dentro de una columna de la tabla.

Operadores

- **Operadores Matemáticos:**
  + suma.
  - Diferencia.
  * multiplicación.
  / división.
  % módulo.
  = asignación
  + Número positivo.
  - Número negativo.

- **Sobre bits:**
  & and sobre bits.
  | or sobre bits.
  ^ xor.
  ~ Complemento a 1.

- **De texto:**
  + Concatenar texto.

Operadores

- Relacionales:

  = Comparación de Igualdad.

  > Mayor que.

  < Menor que.

  >= Mayor o igual que.

  <= Menor o igual que.

  <> Distinto.

  != Distinto.

  !> No mayor que.

  !< No menor que.

Operadores

- Otros:

ALL	True si el conjunto completo de comparaciones es True.
AND	True si dos expresiones son true.
ANY	True si cualquier miembro del conjunto de comparaciones es True.
BETWEEN	True si el intervalo está dentro del intervalo.
EXISTS	True si la subconsulta contiene cualquiera de las filas.
IN	True si el operando está en una lista de valores.
LIKE	True si coincide con un patrón.
NOT	Invierte.
OR	True si cualquiera de las dos expresiones es true.
SOME	True si alguna de las comparaciones de un conjunto es true.

Precedencia de Operadores

Positivo, negativo, y complemento a 1. + - ~
* / %
+ (adición o concatenación), -, &
= (Comparación), >, <, >=, <=, <>, !=, !>, !<
^, |
NOT
AND
ALL, ANY, BETWEEN, IN, LIKE, OR, SOME
= (Asignación)

- Ojo, siempre ante cualquier duda, utilizar paréntesis.

Comodines

- En el operador **LIKE** podemos utilizar comodines para ajustar la búsqueda a un determinado patrón.

%	Cualquier cadena.
_	Cualquier carácter.
[a-d]	Un carácter dentro del intervalo.
[aef]	Carácter individual.
[^a-d]	Un carácter que NO esté en el intervalo.
[^aef]	Cualquier carácter individual que NO esté en el intervalo.

Variables

- SQL Server dispone de una lista de variables globales.
- Están precedidas de dos **@@**.
- Para consultar su valor los podemos hacer mediante un select.
- Ejemplo:

```
Select @@CONNECTIONS
```

Variables del Sistema

- Algunas variables interesantes:

  @@CONNECTIONS
  - Número de conexiones establecidas desde la última vez que se inició.

  @@CURSOR_ROWS
  - Número de filas del último cursor abierto.

  @@ERROR
  - Número del último error de T-SQL.

  @@IDENTITY
  - Último valor de identidad insertado.

  @@LANGID
  - ID del idioma que se está utilizando actualmente.

  @@LANGUAGE
  - El idioma que se está utilizando.

Variables del Sistema

- Mas variables:

  @@ROWCOUNT
  - Número de filas afectadas por la última instrucción de SQL.

  @@SERVERNAME
  - Nombre del servidor local.

  @@TOTAL_ERRORS
  - Número total de errores.

  @@TOTAL_READ
  - Número total de errores de lectura.

  @@TOTAL_WRITE
  - Número total de errores de escritura.

  @@TRANCOUNT
  - Número total de transacciones abiertas en la conexión actual.

  @@VERSION
  - La versión instalada. Y también información sobre el Servipack que tenemos instalado.

Variables locales

- T-SQL nos permite definirnos nuestras propias variables.
- Las variables definidas por el usuario van precedidas de una **@ (solo una).**
- Tenemos que darlas un nombre y un tipo.
- Se definen con declare. **Declare @nombre_var tipo**
  Declare @nombre varchar(40)
- Se las puede asignar valor mediante select o set.
  Set @nombre_var = expresion
  Select @nombre_var = expresion
- Y la podemos recuperar con un select o se pueden utilizar en criterios y expresiones.
  Select * from clientes where nombreCliente = @nombre

Funciones

- Al igual que con las variables para probarlas podemos hacer algo así.

- Select getdate() → devuelve la fecha / hora del sistema.

- Las podemos utilizar dentro campos calculados, criterios, expresiones y se pueden combinar con nuestras variables para cálculos intermedios dentro de funciones y procedimientos definidos por el usuario.

- Ejemplo:

```
declare @miFecha datetime
set @miFecha = getdate()
print @miFecha
```

Funciones del Sistema

- **Matemáticas** → ROUND, ABS, POWER
- **Texto** → LEFT, RIGHT, SUBSTRING, LEN
- **Fecha y hora** → GETDATE, DATEDIFF, DATEADD, EOMONTH
- **Conversión** → CAST, CONVERT, TRY_CONVERT
- **Agregación** → SUM, AVG, COUNT
- **Metadatos** → OBJECT_ID, DB_ID, SCHEMA_NAME

Funciones de Cadena

- **Str(expresión numérica):** Transforma a cadena de texto un expresión numérica o campo que sea numérico.

- **Substring(expresión texto, inicio, longitud)**: Nos devuelve una subcadena dentro de un campo o expresión de texto. Indicamos el inicio en que posición del texto queremos empezar y la longitud, el número de caracteres.

  - **Ejemplo** : Select substring('abcdefg', 3, 2) → Devuelve: 'cd'

Funciones de Cadena

- **CharIndex(expresión1, expresión2, [posición inicial])**: Devuelve la posición inicial de Expresión1 en Expresión2 si no la encuentra devuelve un 0.
  - **Ejemplo**: Select charIndex(‘abcgegeffff’, ‘bc’) → Devuelve 2

- **Replace(ExpresiónTexto, expresion2, expresión3)**: ExpresiónTexto es una expresión o un campo de tipo texto. Expresión2 la expresión a buscar.
  - **Ejemplo:** Select replace(‘abcdefg’, ‘cd’,’xx’) → Devuelve: ‘abxxefg’

Funciones de Cadena

- **Ltrim(cadena), Rtrim(cadena)**
  - Recortar los blancos por la izquierda y por la derecha.
  - No quita blancos dentro de la cadena.

- **Left(cadena, número), Right(cadena, numero)**
  - Cortan el número de caracteres de una cadena por la izquierda y la derecha respectivamente.
  - Sirven para extraer prefijos y sufijos.

- **Lower(cadena), Upper(cadena)**
  - Devuelve la cadena en minúsculas y mayúsculas respectivamente.

- **Len(cadena)**
  - Devuelve el número de caracteres de la cadena.

Funciones de Fecha y Hora

- **Datediff(parte de fecha, fecha inicial, fecha final)**
  - *Parte de Fecha: Ver tabla de constantes.*
  - Restar dos fechas y devuelve la diferencia en partes de fecha, es decir, años, meses, horas, etc.

- **Dateadd(parte de fecha, número, fecha)**
  - Suma tantas partes de fecha a una fecha dada, indicamos: meses, años, días, el número de estos y la fecha a la que la queremos sumar las unidades indicadas en número.
  - Se puede usar para restar si aplicamos un número negativo.

Constantes de datepart

- Ojo, no son cadenas, van sin comillas simples.
- yyyy o yy — Año.
- qq o q — Trimestre.
- mm o m — Mes.
- wk o ww — Semana.
- dw o w — Día de la semana.
- dy o y — Día del año.
- dd o d — Día.
- hh — Hora.
- mi o n — Minuto.
- ss o s — Segundo.
- ms — Milisegundo.

Funciones de Fecha y Hora

- **Getdate()**
  - Devuelve la fecha y la hora del sistema.

- **Day(fecha), Month(fecha), Year(fecha)**
  - Devuelve el día, el mes y el año. Se le pasan una fecha por argumento.

- **Datepart(parte de fecha, fecha)**
  - Devuelve la parte de fecha indicada, por ejemplo:

  select datepart(d, getdate()) → Devuelve el día actual.

Funciones de Fecha y Hora

- **Datename(parte de fecha, fecha)**
  - Devuelve una cadena de caracteres que representa el Datepart especificado de la fecha especificada.
  - Ejemplo:

    select datename(m, getdate())

    También soporta: day, year, etc.

Funciones de Fecha y Hora

- select **eomonth**('12/05/2026', 1);
  - Devuelve el último día del mes siguiente al de la fecha → 30/06/2026
  - Valen número negativos
  - Primer día del mes usando eomonth:
    - SELECT DATEADD(DAY, 1, EOMONTH(GETDATE(), -1));

Funciones Matemáticas

- **Abs(número)**
  - Devuelve el valor absoluto de un número.
- **Ceiling(número_real)**
  - Devuelve el entero mas pequeño mayor de la expresión.
- **Floor(número_real)**
  - Devuelve el entero mas grande menor de la expresión.
- **Round(número_real, num_decimales)**
  - Redondea el número a los decimales indicados.

Funciones Matemáticas

- **Power(número, exponente)**
  - Devuelve número elevado al exponente.

- **Sign(expresión)**
  - Devuelve el signo de la expresión. Devuelve 1 para los positivos, 0 para los negativos.

- **Rand([semilla])**
  - Devuelve un número aleatorio entre 0 y 1. Se le puede pasar una semilla.

- **Sqtr(número)**
  - Devuelve la raíz cuadrada.

- El resto de las funciones, son las trigonométricas, tener en cuenta que trabaja en radianes.

Ejemplo

- Generar varios números aleatorios:

```
DECLARE @counter smallint
SET @counter = 1
WHILE @counter < 5
  BEGIN
    SELECT RAND() Random_Number
    SET @counter = @counter + 1
  END
GO
```

Funciones del Sistema y Metadatos

TESTEAR VALORES: estas funciones devuelven 1 o 0.

- **IsNull(valor, valor sustitución):**
  - Cuando valor es null, devuelve verdadero y lo sustituye por el valor de sustitución.
- **IsNumeric(valor)**
  - Devuelve 1 si valor es un número y 0 en caso contrario.
- **IsDate(valor)**
  - Devuelve 1 si valor es una fecha.
  - Print isdate('12/4/2007')

Funciones del Sistema y Metadatos

- **Convert(tipo de datos, expresión):**
  - Convierte una expresión, un campo a otro tipo de datos. En el primer parámetro indicamos el nuevo tipo, y el segundo el nombre del campo que queremos convertir o el resultado de una expresión.

- Ejemplo:

  convert(varchar(10), precio):
  - convierte el precio que sería float lo pasamos a texto.

Funciones del Sistema y Metadatos

- **CAST(expresión AS Tipo_Datos):**
  - Donde expresión es un campo o una expresión, y tipo de datos es un tipo de SQL Server.

- Ejemplo:
  - select cast(FechaPedido as Varchar(20)) from Pedidos.

Try_convert

- SELECT TRY_CONVERT(DATETIME, '30/2024/01') AS FechaInvalida;
  - Devuelve null

- SELECT TRY_CONVERT(DATETIME, '30/01/2024', **103**) AS FechaEuropea;

- 103 → anexo convert

Ejemplos

- SELECT * FROM Tabla WHERE TRY_CONVERT(INT, CampoTexto) IS NOT NULL;

Comparativa

Función	Si la conversión falla
`CONVERT`	Lanza error
`CAST`	Lanza error
`TRY_CONVERT`	Devuelve NULL
`TRY_CAST`	Devuelve NULL

Existe un objeto

```
IF OBJECT_ID('dbo.MiTabla', 'U') IS NOT NULL
   PRINT 'La tabla existe’;

IF OBJECT_ID('dbo.MiVista', 'V') IS NOT NULL
   PRINT 'La vista existe’;

IF OBJECT_ID('dbo.MiFuncion', 'FN') IS NOT NULL
   PRINT 'La función escalar existe’;

IF OBJECT_ID('dbo.MiFuncion', 'IF') IS NOT NULL
   PRINT 'La función inline existe’;

IF OBJECT_ID('dbo.MiFuncion', 'TF') IS NOT NULL
   PRINT 'La función multisentencia existe’;

IF OBJECT_ID('dbo.MiProcedimiento', 'P') IS NOT NULL
   PRINT 'El procedimiento existe’;

IF OBJECT_ID('dbo.NombreObjeto') IS NOT NULL
BEGIN
   PRINT 'Existe';
END
```

Funciones definidas por el Usuario

- Devuelve un valor.
- Pueden recibir parámetros, siempre el nombre de los parámetros va precedido de @.
- Se suelen realizar cálculos y se devuelve un valor al exterior.
- Tenemos que utilizar la instrucción **return** para devolver un valor.

- Utilizar **create**, **alter**, **drop** o **create or alter**

Tipos de función

- **Escalares (FN)**: devuelven un único dato
  - Se usan en SELECT, WHERE, ORDER BY, etc.
  - Devuelven un único dato.
  - Pueden contener lógica compleja.

```
CREATE FUNCTION dbo.Suma (@a INT, @b INT)
RETURNS INT
AS
BEGIN
   RETURN @a + @b;
END;
```

- **Valores de tabla**: devuelven una tabla
  - **Inline Table-Valued Function (IF)**
    - No tiene BEGIN/END.
    - Es la más rápida.
    - Se comporta como una vista parametrizada.

```
CREATE FUNCTION dbo.ClientesPorCiudad (@Ciudad VARCHAR(50))
        RETURNS TABLE
        AS
        RETURN (
           SELECT * FROM Clientes WHERE Ciudad = @Ciudad
        );
```

Tipos de función II

- **Multi-Statement Table-Valued Function (TF)**
  - Tiene BEGIN/END.
  - Permite lógica compleja.
  - Más lenta que la inline.

```
CREATE FUNCTION dbo.ListaFechas (@Inicio DATE, @Fin DATE)
RETURNS @Tabla TABLE (Fecha DATE)
AS
BEGIN
   DECLARE @F DATE = @Inicio;

   WHILE @F <= @Fin
   BEGIN
      INSERT INTO @Tabla VALUES (@F);
      SET @F = DATEADD(DAY, 1, @F);
   END

   RETURN;
END;
```

Funciones definidas por el Usuario

CREATE FUNCTION [OWNER].[FUNCTION NAME]
(PARAMETER LIST)
RETURNS (return_type_spec) **AS**
BEGIN
(FUNCTION BODY)
END

- En la lista de parámetros tenemos que indicar el nombre del parámetro (al igual que las variables siempre precedidas de una @), el tipo y se le puede dar un valor por defecto.

Ejemplo de Función

```
Create Function doblar
(@input int=0) -- Recibe un parámetro con valor por defecto 0.
Returns int -- El tipo que devuelve la función.
AS
BEGIN
    Return 2 * @input  -- Las funciones siempre devuelven un valor.
END
```

- Esta función la podemos utilizar en una consulta o desde código T-SQL, la podemos asignar a una variable.

Otro Ejemplo de Función

```
CREATE FUNCTION dbo.Actualizar_trim_entrada_mora (@FEC_EMISION DATETIME,
                                                  @MOROSIDADNEW DATETIME)
RETURNS INT AS
BEGIN
    DECLARE @RESUL INTEGER
    DECLARE @DIF_FECHAS FLOAT

    SET @DIF_FECHAS = DATEDIFF(DAY, @FEC_EMISION, @MOROSIDADNEW)/30.0;

    IF (@DIF_FECHAS > 36)
        SET @RESUL = 12;

    ELSE IF (@DIF_FECHAS <= 0)
        SET @RESUL = 1;

    ELSE IF ( ( (@DIF_FECHAS / 3.0) - ROUND(@DIF_FECHAS / 3.0, 0) ) > 0)
        SET @RESUL = ROUND (@DIF_FECHAS / 3.0, 0) + 1;

    ELSE
        SET @RESUL = ROUND (@DIF_FECHAS / 3.0, 0);

    RETURN @RESUL;
END
```

Resumen

Tipo	Código	Devuelve	Uso típico
Scalar Function	`FN`	Un valor	Cálculos, validaciones
Inline Table-Valued Function	`IF`	Tabla	Vistas parametrizadas, muy rápida
Multi-Statement Table-Valued Function	`TF`	Tabla	Lógica compleja, más lenta
Funciones del sistema	—	Varios	Texto, fechas, matemáticas, metadatos

- FN → select dbo.funcionEscalar();
- IF → select * from dbo.funcionInline();
- TF → select * from dbo.multiSentencia();
- Sistema → select funcion_sistema();

Importante

- SQL Server NO tiene CREATE OR REPLACE FUNCTION.
- Las funciones no pueden modificar datos (no permiten INSERT/UPDATE/DELETE salvo en tablas variables internas).
- Las funciones escalares pueden afectar al rendimiento si se usan en SELECT masivos.
- Las inline TVF son las más rápidas porque el optimizador las trata como una vista.

Ejecución del código T-SQL

- Mediante la herramienta SQL Server Management Studio podemos crear Scripts SQL y luego ejecutarlos.
- Lo podemos utilizar para realizar pruebas o para grabar scripts y luego ejecutarlos.
- Botón nueva consulta.
- Es un editor de texto que reconoce el código T-SQL.

Estructuras de control T-SQL

- **Print**: Imprimir un mensaje.
  - Print @Nombre, imprime el contenido de la variable.
  - Print ‘Hola’
  - Print ‘hola ‘ + @Nombre;
  - Si hay null:
    - SELECT CONCAT('Nombre: ', @nombre, ', Edad: ', @edad);

- **Condicional (con 1 sola instrucción):**

IF Condición
    instrucción_A
ELSE
    instrucción_B

Estructuras de control T-SQL

```
-- También se pueden anidar:
IF (<expresion>)
	BEGIN

	END

ELSE IF (<expresion>)
	BEGIN

	END

ELSE
	BEGIN

	END
```

Estructuras de control T-SQL

- **Condicional (con varias instrucciones)**

```
IF Condición
    BEGIN
      instrucciones
    END
ELSE
    instrucción_B
```

- **Case:**

```
CASE <expresion>
        WHEN <valor_expresion> THEN <valor_devuelto>
        WHEN <valor_expresion> THEN <valor_devuelto>
        ELSE <valor_devuelto> -- Valor por defecto
      END
```

- Sentencia **EXISTS**:
  - Para detectar si hay filas, en una subconsulta o en una sentencia IF.

Ejemplo

- Case se puede utilizar dentro de una sentencia SQL:

```
SELECT
    CASE
        WHEN Edad >= 18 THEN 'Adulto'
        ELSE 'Menor'
    END AS Categoria
FROM Personas;
```

Ejemplos

```
-- La sentencia CASE se puede utilizar para evaluar y
    asignar valor a una variable.
DECLARE @Web varchar(100),@diminutivo varchar(3)
SET @diminutivo = 'DJK'
SET @Web = (CASE
WHEN @diminutivo = 'DJK' THEN 'www.devjoker.com’
WHEN @diminutivo = 'ALM' THEN 'www.aleamedia.com'
ELSE 'www.devjoker.com'
END)
PRINT @Web
```

Ejemplos

```
USE AdventureWorks;
GO
SELECT ProductNumber, Category =
CASE ProductLine
      WHEN 'R' THEN 'Road'
      WHEN 'M' THEN 'Mountain'
      WHEN 'T' THEN 'Touring'
      WHEN 'S' THEN 'Other sale items'
      ELSE 'Not for sale'
END, Name
FROM Production.Product
ORDER BY ProductNumber;
GO
```

Ejemplos

```
USE AdventureWorks;
GO
SELECT ProductNumber, Name,
'Price Range' = CASE
      WHEN ListPrice = 0 THEN 'Mfg item - not for resale'
      WHEN ListPrice < 50 THEN 'Under $50'
      WHEN ListPrice >= 50 and ListPrice < 250 THEN 'Under $250'
      WHEN ListPrice >= 250 and ListPrice < 1000 THEN 'Under $1000'
      ELSE 'Over $1000'
END
FROM Production.Product
ORDER BY ProductNumber ;
GO
```

Ejemplos

```
-- La instrucción admite subconsultas:
    DECLARE @coPais int, @descripcion varchar(255)

    set @coPais = 5
    set @descripcion = 'España‘

IF EXISTS(SELECT * FROM PAISES WHERE CO_PAIS = @coPais)
    BEGIN
        UPDATE PAISES SET DESCRIPCION = @descripcion WHERE CO_PAIS = @coPais
    END
ELSE
    BEGIN
        INSERT INTO PAISES (CO_PAIS, DESCRIPCION) VALUES (@coPais, @descripcion)
    END
```

Ejemplos

```
-- También se admiten subconsultas:
DECLARE @Web varchar(100), @diminutivo varchar(3)
SET @diminutivo = 'DJK‘
SET @Web = (CASE
      WHEN @diminutivo = 'DJK' THEN
            (SELECT  web FROM WEBS WHERE id=1)

      WHEN @diminutivo = 'ALM' THEN
            (SELECT  web  FROM WEBS WHERE id=2)

      ELSE 'www.devjoker.com‘
END)

PRINT @Web
```

Sentencias de control T-SQL

- **Bucle:**

  WHILE Condición

  begin

  Instrucciones

  end
- **Break**
  - Rompe el bucle, no continua la ejecución.
  - Lo podemos utilizar cuando no queremos que el bucle se siga ejecutando.
  - Produce la salida del bucle WHILE más interno. Se ejecutan las instrucciones que aparecen después de la palabra clave END, que marca el final del bucle.
- **Continue**
  - Fuerza otra iteración. Las instrucciones que estén por debajo de continue no se ejecutarán.

Ejemplos

```
declare @i int
set @i = 1

while @i < 10
begin
    if (@i = 5)
        break
    print @i
    set @i = @i + 1
end
```

```
declare @i int
set @i = 0

while @i < 10
begin
    set @i = @i + 1
    if (@i = 5)
        continue
    print @i
end
```

¿Qué hace este código?

Ejemplo

```
USE AdventureWorks;
GO
WHILE (SELECT AVG(ListPrice) FROM Production.Product) < $300
      BEGIN
      UPDATE Production.Product SET ListPrice = ListPrice * 2
      SELECT MAX(ListPrice) FROM Production.Product
      IF (SELECT MAX(ListPrice) FROM Production.Product) > $500
          BREAK
      ELSE
           CONTINUE
END
```

- Dentro de los bucles podemos tener consultas, consultas con cursores o directamente código.

Definición mediante DDL

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

Definición mediante DDL

- Todos estos comandos los vamos a utilizar también para crear, modificar y eliminar objetos de la BD, como pueden ser: tablas, vistas, procedimientos almacenados, funciones.

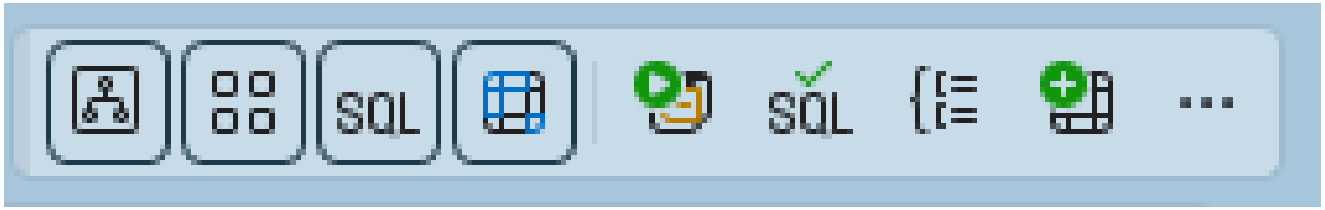
- Ejemplo:

```
CREATE PROCEDURE Test2 AS
CREATE TABLE #t(x INT PRIMARY KEY)
INSERT INTO #t VALUES (2)
SELECT Test2Col = x FROM #t;
GO
CREATE PROCEDURE Test1 AS
CREATE TABLE #t(x INT PRIMARY KEY)
INSERT INTO #t VALUES (1)
SELECT Test1Col = x FROM #t;
EXEC Test2;
GO
CREATE TABLE #t(x INT PRIMARY KEY)
 INSERT INTO #t VALUES (99);
GO
EXEC Test1;
GO
```

Vistas

- Mediante la vistas podemos consultar datos de las tablas originales.
- Incluso podemos realizar consultas sobre las vistas ya grabadas.
- SQL Server dispone de un editor que nos ayuda con el código SQL para componer vistas.
- Cuando agregamos una nueva vista nos aparece el editor de vistas.

Vistas



- El editor se divide en 4 paneles: tablas, campos, sql, resultados. Los podemos visualizar o no.
- Ejecutar la vista.
- Chequear la sintaxis de SQL.
- Agrupar resultados, para meter funciones de agregado.
- Abrir la pantalla de agregar tablas.
- Añadir una tabla derivada.

Vistas: Utilización de Alias

- Al crear vistas o consultas de selección nos puede interesar cambiar el nombre de las columnas.

- SELECT AddressID, City **AS ciudad** FROM Person.Address ORDER BY **ciudad**

- También podremos utilizar el alias en el nombre de la tabla.

Combinar varias tablas en una vista

- Para combinar varias tablas en una vista vamos a utilizar JOIN.

Select top 100 percent nombre_empleado, producto, precio from empleados e

Join ventas v on e.idemp = v.idemp

- Se pueden combinar mas tablas, siempre combinándolas mediante las claves.

Modificar datos mediante una vista

- Podemos utilizar las vistas para añadir, modificar o eliminar registros de la tabla asociada a la vista.

- Primero, crear una vista (por ejemplo):
  - Select * from clientes
  - La grabamos como vista_clientes.

- Segundo, utilizamos la vista para insertar registros:
  - Insert into vista_clientes values (‘jose’,’fernandez’,’91234433’)

Procedimientos Almacenados

- No devuelven nada.
- Pueden recibir parámetros, siempre el nombre de los parámetros va precedido de @.
- Dentro del procedimiento se suelen realizar sentencias SQL, por ejemplo dar de alta un nuevo registro.
- O ejecutar una consulta.

Procedimientos Almacenados

Sintaxis de un procedimiento:

CREATE PROCEDURE Nombre_esquema.Nombre
@NombreParam1 TipoParam1 numeric**,**
@NombreParam2 TipoParam2, … **AS**
Instrucciones
GO

Ejemplo de Procedimiento

```
CREATE PROCEDURE [Production].[Show_Products]
-- Se indica el nombre del esquema y el nombre del procedimiento.
AS
BEGIN
    -- Cuando SET NOCOUNT es ON, no se devuelve el número de filas
    afectado por una instrucción Transact-SQL. Cuando SET NOCOUNT es OFF,
    sí se devuelve ese número.
    SET NOCOUNT ON;

   -- Instrucciones del procedimiento:
    SELECT Name,Color,ListPrice, SellStartDate
    FROM Production.Product
    WHERE SellStartDate > '1/1/2003'
    ORDER BY SellStartDate, Name
END
GO
```

Ejecutar el procedimiento

- Dentro de otra ventana.
- Introducimos estas dos sentencias:

```
-- USE selecciona una Base de datos:
USE AdventureWorks

-- EXEC nombre_esquema.nombre_procedure
EXEC Production.Show_Product
```

- Es igual de válido utilizar EXECUTE en vez EXEC.

Modificar un procedimiento

- Una vez que está grabado un objeto en la BD no podemos repetirlo.
- Tenemos que modificarlo, podemos hacer uso de la instrucción ALTER PROCEDURE.

- Podemos tener parámetros de entrada, se define después del nombre del procedimiento.
- @fecha [datetime]

- Si queremos darle un valor por defecto:
- @fecha [datetime] = ‘1/1/2003’

Ejemplo

```
-- Modificamos el procedimiento añadiendo un parámetro de entrada
    con un valor por defecto:

ALTER PROCEDURE [Production].[Show_Products]
@fecha [Datetime] = '1/1/2003'
AS
BEGIN
    SET NOCOUNT ON;

    SELECT Name,Color,ListPrice, SellStartDate
    FROM Production.Product
    WHERE SellStartDate > @fecha
    ORDER BY SellStartDate, Name
END
GO
```

Ejecutar el procedimiento

- use AdventureWorks
- EXEC Production.Show_Products '7/1/2003‘

- Le pasamos el valor de los parámetro en la línea de ejecución.

- Si obviamos el parámetro, tomará el valor por defecto definido dentro de procedimiento.

Varios parámetros

```
-- Separados por comas y con la misma sintaxis:

ALTER PROCEDURE [Production].[Show_Products]
@fecha [Datetime] = '1/1/2003',
@fecha2 [DateTime] = '1/2/2003'
AS
BEGIN
	-- SET NOCOUNT ON added to prevent extra result sets from
	-- interfering with SELECT statements.
	SET NOCOUNT ON;

   -- Insert statements for procedure here
	SELECT Name,Color,ListPrice, SellStartDate
	FROM Production.Product
	WHERE SellStartDate > @fecha AND SellStartDate < @fecha2
	ORDER BY SellStartDate, Name
END
GO

-- La ejecución:
use AdventureWorks
EXEC Production.Show_Products '7/1/2003', '10/10/2003'
```

Parámetros de Salida

- A parte de parámetros de entrada podemos tener parámetros de salida para devolver valores.
- Se definen igual que los de entrada, pero al final de la definición se indica la palabra reservada: **OUTPUT**.
- Para asignarle un valor dentro del procedimiento se utiliza **SET**.
- Y a la hora de ejecutarlo tenemos que pasar una variable, no podemos pasar un valor en la posición del parámetro de salida.

Ejemplo

```
CREATE PROCEDURE Production.Precio_con_iva
@precio [float],
@iva [float],
@precio_final [float] output
AS
BEGIN
   set @precio_final = @precio * (1 + @iva)
END
GO
```

Ejecutar con parámetros de salida

```
use AdventureWorks
Declare @precioFinal float

exec Production.Precio_con_iva 1000.0, 16.0,
    @precioFinal OUTPUT
print 'El precio final es: ' + convert(varchar(10),
    @precioFinal)
```

Mas sobre procedimientos

- Podemos ver:

```
Create Procedure nombre_esquema.nombre_proc
With Execute As Caller
As
Begin
    -- Instrucciones.
End
Go
```

- With Execute As Caller: para que SQL Server personifique al llamador al ejecutar el procedimiento.

Mas sobre procedimientos

- **Eliminar** un procedimiento:
  drop procedure Production.Show_Products_2

- Proteger los procedimientos:

```
ALTER PROCEDURE Production.Precio_con_iva
@precio [float],
@iva [float] = 16.0,
@precio_final [float] output
WITH ENCRYPTION
AS
BEGIN
    set @precio_final = @precio + (@precio * @iva / 100.0)
END
GO
```

Triggers

- Son tipos especiales de procedimientos almacenados que se activan de forma controlada antes de por llamadas directas, están asociados a tablas.
- Son una gran herramienta para controlar las reglas de negocio mas complejas que una simple integridad referencial.
- Los desencadenadores y la sentencias que desencadenan su ejecución trabajan unidas como una transacción.

Triggers

- El grueso de sentencias de un desencadenador deben ser INSERT, UPDATE, DELETE.

- Los desencadenadores siempre forman parte después de que la operación fue registrada en el fichero de Log.

Triggers

CREATE TRIGGER [*schema_name* .]*trigger_name*

ON { *table* | *view* }

{ FOR | AFTER | INSTEAD OF }

{ [INSERT] [,] [UPDATE] [,] [DELETE] }

AS { *sql_statement* [;] [...*n*]}

Triggers

- **Ejemplo de TRIGGER**: Inserta un nuevo pedido sobre la tabla, y se incrementan las ventas del representante y se actualizan las existencias**.**

```
CREATE  TRIGGER  NuevoPedido
ON Pedidos
FOR INSERT
AS
        UPDATE RepVentas
        SET VENTAS =VENTAS + INSERTED.IMPORTE
        FROM REPVENTAS INNER JOIN INSERTED
        ON REPVENTAS.NUM_EMPL = INSERTED.REP

        UPDATE PRODUCTOS
        SET EXISTENCIAS = EXISTENCIAS - INSERTED.CANT
        FROM PRODUCTOS INNER JOIN INSERTED
        ON PRODUCTOS.ID_FAB = INSERTED.FAB
        AND PRODUCTOS.ID_PRODUCTO = INSERTED.PRODUCTO
 GO
```

Triggers / INSERT

- **Insert**: Se activan cada vez que alguien intenta insertar un nuevo registro.

- Para la inserción de registros se genera una tabla en la caché con la información que se intenta añadir (INSERTED) y podemos hacer comparaciones con esta tabla.

Triggres / DELETE

- **Delete**: Salta cuando una alguien intenta hacer una eliminación de una tabla.

- Para la eliminación de registros se genera una tabla llamada (DELETED) y se pueden hacer comparaciones.

Triggers / UPDATE

- **Update**: Saltan cuando alguien intenta hacer una actualización en una tabla.

- En el caso de una actualización, la nueva información irá en INSERTED y DELETED la información que será reemplazada.

Triggers / INSTEAD OF

- Este tipo tienen que ver con las vistas cuando utilizamos una vista que no dispone de todas las columnas para modificar la tabla asociada e intentamos realizar una inserción mediante la vista.

- Para evitar problemas se puede utilizar un desencadenador instead of.

- En este desencadenador, se indicará instead of e insert y dentro del código se insertarán todos los campos, para ello podemos utilizar la tabla inserted y los valores que faltan los damos un valor.

- Después haríamos una inserción directamente en la tabla.

Gestión de Excepciones

- **Una excepción es un error que ocurre durante la ejecución de una instrucción T-SQL:**
  - División por cero
  - Violación de clave primaria
  - Conversión inválida
  - Error de sintaxis
  - Timeout
  - Violación de restricción CHECK
  - Error de conexión
  - Etc.
- **Cuando ocurre una excepción, SQL Server:**
  - Interrumpe la ejecución del bloque actual
  - Devuelve un error al cliente
  - Si hay un bloque TRY/CATCH, salta al CATCH

Esquema

- BEGIN TRY
- -- Código que puede fallar
- END TRY
- BEGIN CATCH
- -- Código que se ejecuta si hay error
- END CATCH

Ejemplo

```
BEGIN TRY
    SELECT 1 / 0;  -- Error
END TRY
BEGIN CATCH
    PRINT 'Ha ocurrido un error';
END CATCH;
```

Gestión de Excepciones

- TRY / CATCH / THROW

```
BEGIN TRY
    INSERT INTO Tabla VALUES (1/0); -- error
END TRY
BEGIN CATCH
    PRINT ERROR_MESSAGE();
END CATCH;

THROW 50001, 'Error personalizado', 1;
```

Funciones para el catch

- ERROR_NUMBER() -- Número de error
- ERROR_MESSAGE() -- Mensaje completo
- ERROR_SEVERITY() -- Severidad
- ERROR_STATE() -- Estado
- ERROR_LINE() -- Línea donde ocurrió
- ERROR_PROCEDURE() -- Procedimiento donde ocurrió

Ejemplo

```
BEGIN TRY
   SELECT CAST('ABC' AS INT);
END TRY
BEGIN CATCH
   SELECT
      ERROR_NUMBER() AS Numero,
      ERROR_MESSAGE() AS Mensaje,
      ERROR_LINE() AS Linea;
END CATCH;
```

Ejemplo

```
BEGIN TRY
   SELECT 1/0;
END TRY
BEGIN CATCH
   PRINT 'Error detectado';
   THROW;  -- vuelve a lanzar el error original
END CATCH;
```

Transacciones

- Las transacciones son unidades de trabajo que no se pueden dividir.

- Cuando realizamos una operación de actualización sobre un registro, se ejecuta como una única transacción.
  - Update Empleados set sueldo = 1500 where idEmpleado = 55

¿Qué son las transacciones?

- Si a la vez que realizamos una actualización en una tabla tenemos que actualizar otro campo en otra tabla.
  - Update Empleados set sueldo = 1500 where idEmpleado = 55
  - Update Empleados set irpf=15 where idEmpleado = 55
  - En este caso si se ejecutan por separado, se puede producir un error en una de ellas y la otra se ejecutaría normalmente.

  - Las transacciones nos va a permitir agrupar varias instrucciones como si fueran solo una.

Ejemplo

```
Declare @e_irpf int, @e_sueldo int
Begin transaction -- Inicio de transacción.
    Update Empleados set sueldo = 1500 where idEmpleado = 55
    Set @e_sueldo = @@ERROR

    Update Empleados set irpf=15  where idEmpleado = 55
    Set @e_irpf = @@ERROR

If @e_irpf = 0 and @e_sueldo = 0
   Commit transaction -- Hacer efectiva la transacción.
Else
   Rollback transaction -- Restaurar
```

Las propiedades ACID

- Las transacciones se identifican con las propiedades:
  - Atomicidad
  - Coherencia
  - Aislamiento
  - Durabilidad

- **Atomicidad**: La transacción se ejecuta como una sola instrucción.

Las propiedades ACID

- **Coherencia**: Cuando una transacción se hace permanente o se cancela no va a dejar la BD con inconsistencias.

- **Aislamiento**: Si se están realizando varias transacciones a la vez son aisladas. Si un usuario está leyendo y otro escribiendo en la misma tabla, es usuario que lee encuentra los datos como si todavía no se hubiera hecho ninguna modificación.

- **Durabilidad**: Al confirmar una transacción los efectos son permanentes.

Utilizar transacciones

- Disponemos de las instrucciones:
  - Begin Transaction
  - Commit Transaction
  - Rollback Transaction
  - Save Transaction

- Y también de estas dos variables globales del sistema:
  - @@error
  - @@trancount

Begin Transaction

- Indica a SQL Server que inicie una nueva transacción.
- Permite también begin tran.
- Se puede proporcionar un nombre a la transacción, para utilizarlo a la hora de cancelarla o hacerla efectiva.

- Las transacciones se puede anidar.

```
Begin transaction
    Instrucciones
    Begin transaction
        Instrucciones
    Commit Transaction -- Hace referencia a la última.
Commit Transaction
```

Commit Transaction

- Esta instrucción confirma la transacción y la hace permanente.
- También se permite Commit tran.
- Una vez que se ejecuta el commit no se puede volver al estado anterior.

Rollback Transaction

- Cancela una transacción que todavía no se ha hecho permanente.
- También permite Rollback tran.

- Si el rollback se ejecuta dentro un desencadenador no se ejecutarán las instrucciones SQL posteriores.

- Dentro de un procedimiento almacenado, si se ejecutarán.

- Rollback Work revierte todas las transacciones anidadas y establece @@trancount a cero.

Save Transaction

- La instrucción Save Transaction le permite confirmar una transacción parcialmente al tiempo que se permite revertir el resto de la transacción.
- También permite Save Tran.
- Cuando se ejecuta Save transaction hay que darla un nombre que constituye un punto de referencia.

Ejemplo

Begin transaction

Update Production.Product Set SafetyStockLevel = 1000 where ProductID = 321

Save transaction Ssaved

Update Production.Product Set ReordenPoint = 10 Where ProductID = 321

RollBack Transaction Ssaved

Commit Transaction

- Rollback elimina los efectos de la actualización en la columna ReorderPoint, pero deja la actualización de la columna SafetyStockLevel lista para su confirmación posterior.

- Después Commit confirma la parte de la transacciónque no se ha revertido.

@@TRANCOUNT

- Indica el número de transacciones anidadas que están pendientes actualmente.
- Si no existe ninguna transacción pendiente la variable será 0.
- Con esta variable podemos determinar si un trigger se está ejecutando dentro de una transacción.

@@ERROR

- Contiene el número de errores que se han producido al ejecutar una instrucción de T-SQL.
- Si no se ha producido ningún error, el valor será 0.
- Normalmente el valor de esta variable se guarda en otra variable local para posteriormente poderla comprobar.

Ejemplo

```
Declare @ss_err int, @rp_err int
Begin transaction
    Update Production.Product Set SafetyStockLevel = 1000 where ProductID = 321
    Set @ss_err = @ERROR
    Save transaction Ssaved -- Establecer un de almacenamiento.
    Update Production.Product Set ReordenPoint = 10 Where ProductID = 321
    Set @rp_err = @ERROR

    If @rp_err <> 0
        RollBack Transaction Ssaved

    If @ss_err = 0 and @rp_err = 0
        Begin
          Commit Transaction
          Print 'Cambios realizados con éxito'
        End
    Else
        Rollback Transaction
```

Instrucciones Avanzadas

- OpenQuery
  - Nos permite utilizar una consulta de tipo Select en un servidor vinculado para devolver una tabla virtual.
  - La sintaxis:
    - Openquery(linked_server, 'query')
    - Ejemplo:
      - Select CustomerID, FirstName, LastName FROM **openquery**(server2, 'select firstname, lastname, contactID from AdventureWorks.Person.Contact') as Contact inner join Sales.individual on contact.ContactID = Sales.individual.ContactID

Instrucciones Avanzadas

- OpenRowSet
  - También permite conectar con otro servidor y obtener datos, pero en este caso se le mandan todos los parámetros necesarios para conectar con el servidor.
  - Es útil cuando no se ha vinculado el otro servidor.
  - OpenRowSet('provider_name', 'datasource';'user_id';'password','query')
  - Ejemplo:
    - Select * from openrowset('SQLOLEDB', 'Server1' ; 'sa', 'password', 'select * from clientes') …

Cursores

- Cuando trabajamos con la instrucción select los resultados los devuelve como un bloque de filas, y no podemos acceder a una fila en concreto.

- Los cursores nos permiten acceder fila a fila, pero es mas lento que acceder mediante el SQL.

- Un cursor es un conjunto de filas junto con un puntero que identifica la fila actual.

Declare Cursor

- Declare Cursor_name [insensitive][scroll] cursor for select_statement [for {read_only | update [of column_name [, …n]]}]
- Insensitive: se establece una tabla temporal solamente para el cursor. Cambios que se realicen no se verán en el cursor.
- Scroll: Deberían estar disponibles todas las opciones de la instrucción Fetch.
- Select_statement: La instrucción select.
- Read_only: Impide actualizaciones a través del cursor.
- Update: Para poder actualizar datos, si se utiliza con la opción of column_name, sólo se podrán actualizar esas columnas.

Declare Cursor

- Otra forma mas completa de abrir el cursor:

Declare Cursor_name cursor
[Local | Global]
[Forward_only | scroll]
[static | keyset | Dynamic | Fast_Forward]
[Read_only | scroll_locks | optimistic]
[type warning]
for select_statement
[for update [of column_name [, …n]]}]

Declare Cursor

- Global / Local: Queremos que el cursor solo se limite al bloque de código o que esté disponible para cualquier instrucción en la conexión actual.
- Forward_only Solo compatible con la instrucción next de Fecth. Solo para avanzar hacia delante.
- Scroll: Deben estar disponibles todas las opciones de Fetch.
- Static no permite cambios.
- Keyset: Actualizable.
- Dynamic: Actualizable y muestra las nuevas filas.
- Read_only: Solo de lectura.
- Scroll_Locks: Sql Server bloquea las filas cuando se leen en el cursor.
- Optimistics: Emplea bloqueo optimista cuando se intenta cambiar una fila a través del cursor.
- Type_warning: Envía advertencias cuando las opciones del cursor no se pueden realizar.
- For update: El cursor permite la actualización.

Open y @@CURSOR_ROWS

- Open se utilizar para llenar el cursor con los datos obtenidos de la instrucción select indicada en la declaración del cursor.

- Open {{[global] nombre_cursor} | cursor_variable_name }

- La variable del Sistema @@Cursor_rows nos da el número de registros del último cursor abierto.

Ejemplo

```
use AdventureWorks;

declare micursor CURSOR
local scroll static
for select * from Person.Address
open micursor
print @@Cursor_rows
```

FETCH y @@FETCH_STATUS

- Esta instrucción se emplea para obtener datos de un cursor a variables para poder trabajar con los datos.

FETCH

[[next | prior | first | last

| absolute {n | @n_variable }

| relative { n | @n_variable }

]

FROM

{{[Global] cursor_name} | @cursor_variable_name }

[Into @variable_name [, … n]]

FETCH y @@FETCH_STATUS

- Next: Devuelve la fila siguiente.
- Prior: La fila anterior.
- First: Primera fila.
- Last: La última fila.
- Absolute: Devuelve el registro especificado. Absolute 5.
- Relative: Devuelve el registro indicado a partir del actual.
- Into: Permite especificar variables para contener los datos devueltos.

- La variable @@FETCH_STATUS devuelve el estado del cursor. Lo utilizaremos en un bucle para poder avanzar registro a registro.

Ejemplo

```
Declare @firstname nvarchar(50), @lastname nvarchar(50)
Declare contact_cursor CURSOR
Local scroll static
For
    Select firstname, lastname from clients
Open contact_cursor
Fetch next from contact_cursor into @firstname, @lastname
Print @firstname + ‘ ‘ + @lastname
While @@fetch_status = 0
    Begin
        Fetch next from contact_cursor into @firstname, @lastname
        Print @firstname + ‘ ‘ + @lastname
    End
```

Close

- La inversa de open.
- Close {{[global] nombre_cursor} | cursor_variable_name }
- Cuando se termine de trabajar con un cursor se cierra y se liberan todas las filas.

Deallocate

- Es la inversa de Declare Cursor.
- Deallocate {{[global] nombre_cursor} | cursor_variable_name }
- Después de cerrar el cursor se utiliza para destruir las estructuras de datos del cursor.