E.T.S.
INGENIERÍA
INFORMÁTICA

# Gestión de la Información

Grado en Ingeniería del Software

UNIVERSIDAD
DE MÁLAGA

LENGUAJES Y
CIENCIAS DE LA
COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

# Tema 2

## Aplicaciones mediante SQL

José Luis Pastrana Brincones
pastrana@lcc.uma.es

# SQL Server

- Microsoft SQL Server es un Sistema gestor de Bases de Datos Relacionales desarrollado por Microsoft.
- Sus lenguajes de consulta primarios son: T-SQL y ANSI SQL.
- La primera versión salió en 1989.
- Pretende competir con Oracle, IBM, etc.
- Versión más estable actual: SQL Server 2014 (12.0) Abril de 2014
- Última Versión SQL Server 2017 (14/09/2017)

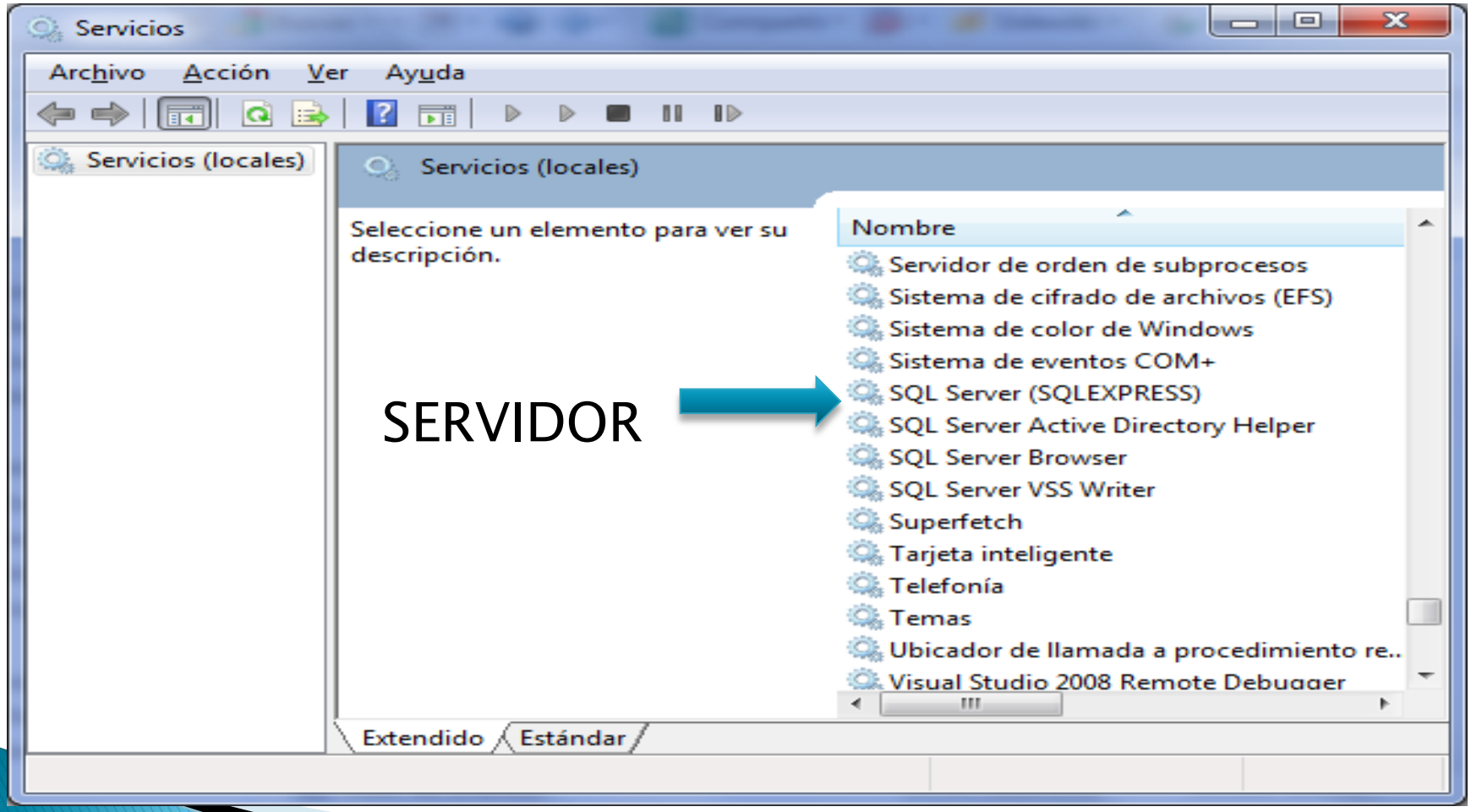
# SQL Server

- Dispone de un amplio abanico de opciones de conexión y una amplia variedad de tecnologías para importar y exportar datos (Tabular Data Stream, XML, Integration Services, bulk copy, SQL Native Connectivity, OLE DB, ODBC).
- Dispone de consultas distribuidas con un coordinador de transacciones distribuidas.
- El motor trabaja bien con diferentes datos (XML, vectoriales, palabras dentro de textos y binarios grandes “blob data”).
- Tiene un amplio conjunto de componentes y herramientas para trabajar con datos y hacer minería de datos.
- Incluye un amplio conjunto de herramientas para informes.

# SQL Server

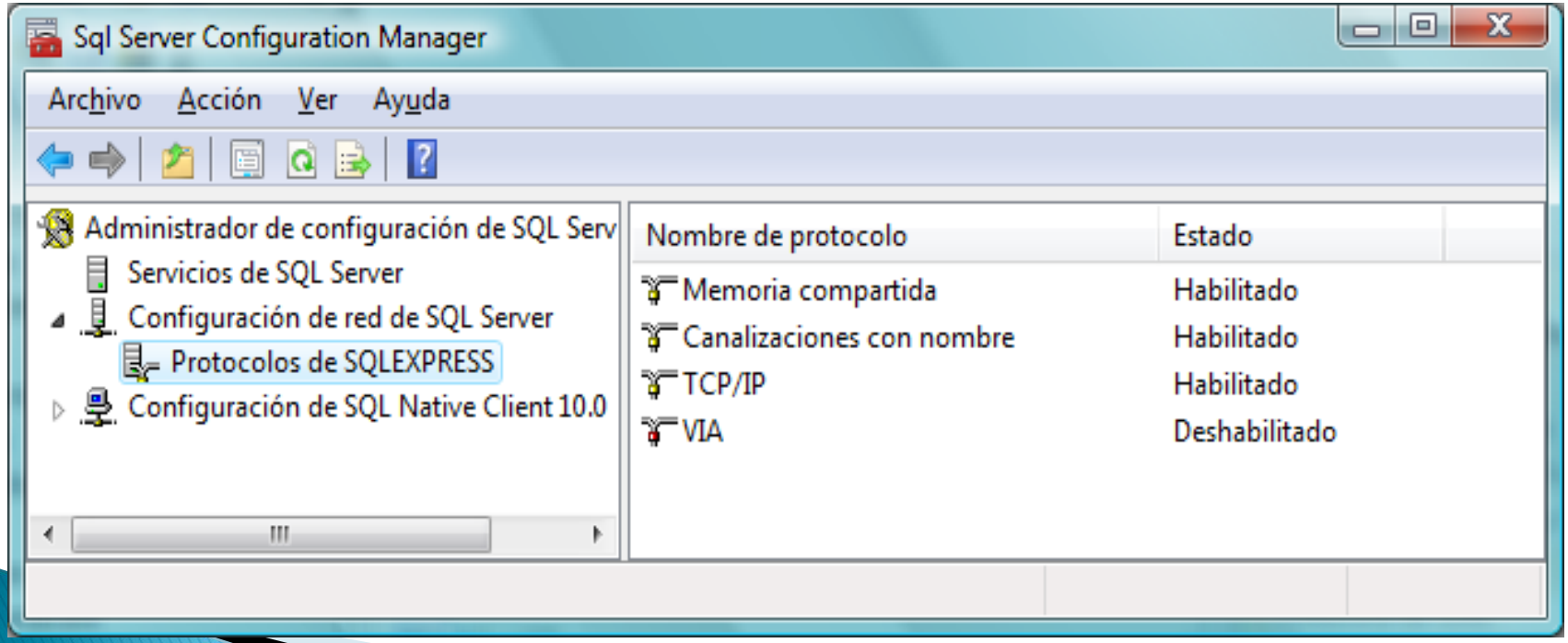
- Ofrece un impresionante nivel de detalle de diagnostico con “Performance Studio”, “SQL Trace/Profiler”, y el manejo de vistas y funciones de la base de datos.
- Incluye diferentes opciones para una alta disponibilidad con diferentes grados de latencia, rendimiento, número de nueves, distancia física y sincronización.
- Puede ser manejado de forma declarativa mediante Policy-Based Management.
- SQL Server’s Management Studio es una interfaz de usuario Madura y útil tanto para el desarrollador como para el administrador de bases de datos.
- Está disponible en diferentes ediciones  tanto para 32-bit como para  64-bit, siendo la versión Express, totalmente gratuita (4GB de almacenamiento y 5 usuarios concurrentes).

# SQL Server


Servicios
Archivo Acción Ver Ayuda
Servicios (locales)
Servicios (locales)
Seleccione un elemento para ver su descripción.
SERVIDOR
Nombre
Servidor de orden de subprocesos
Sistema de cifrado de archivos (EFS)
Sistema de color de Windows
Sistema de eventos COM+
SQL Server (SQLEXPRESS)
SQL Server Active Directory Helper
SQL Server Browser
SQL Server VSS Writer
Superfetch
Tarjeta inteligente
Telefonía
Temas
Ubicador de llamada a procedimiento re..
Visual Studio 2008 Remote Debugger
Extendido
Estándar

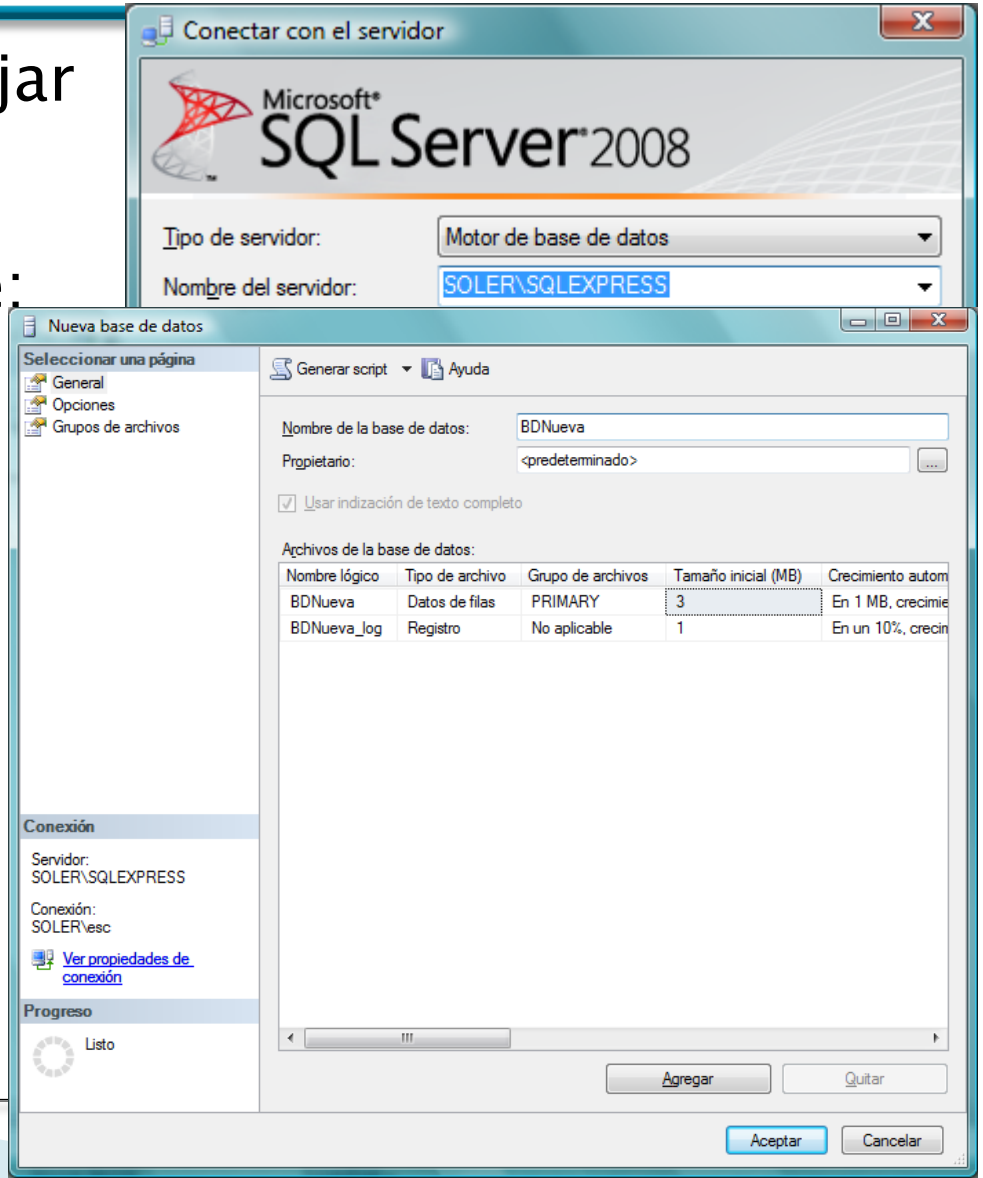
# SQL Server Configuration Manager

- Herramienta para gestionar los servicios asociados a SQL Server, configurar los protocolos de red usados por SQL Server, y gestionar la configuración de la conectividad desde los clientes de SQL Server.



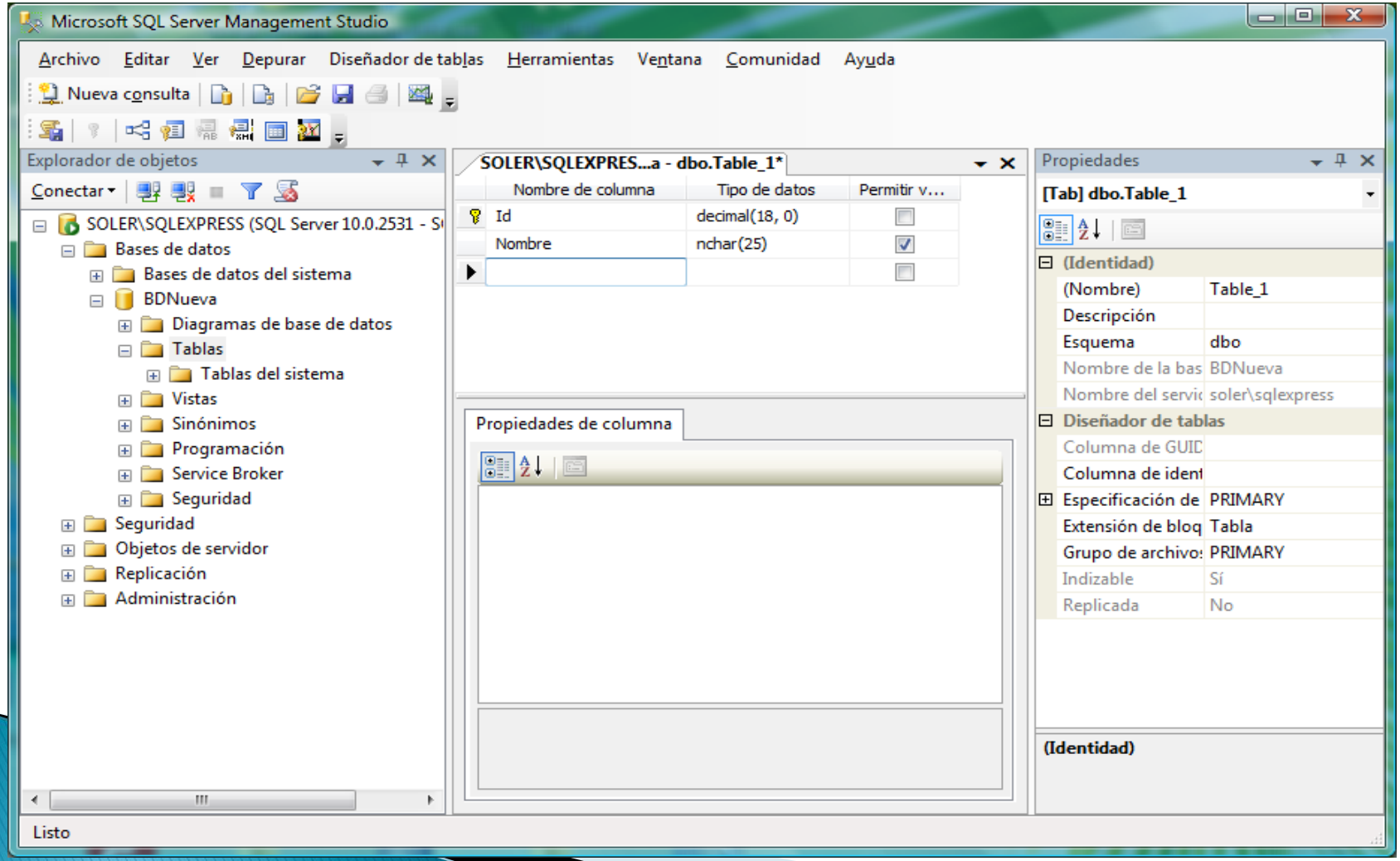
# SQL Server Management Studio

- Herramienta para Manejar una Base de Datos
- 2 Formas de conectarse:
  - Usuario de Windows
  - Usuario de SQL Server
- Creación de una Base de Datos



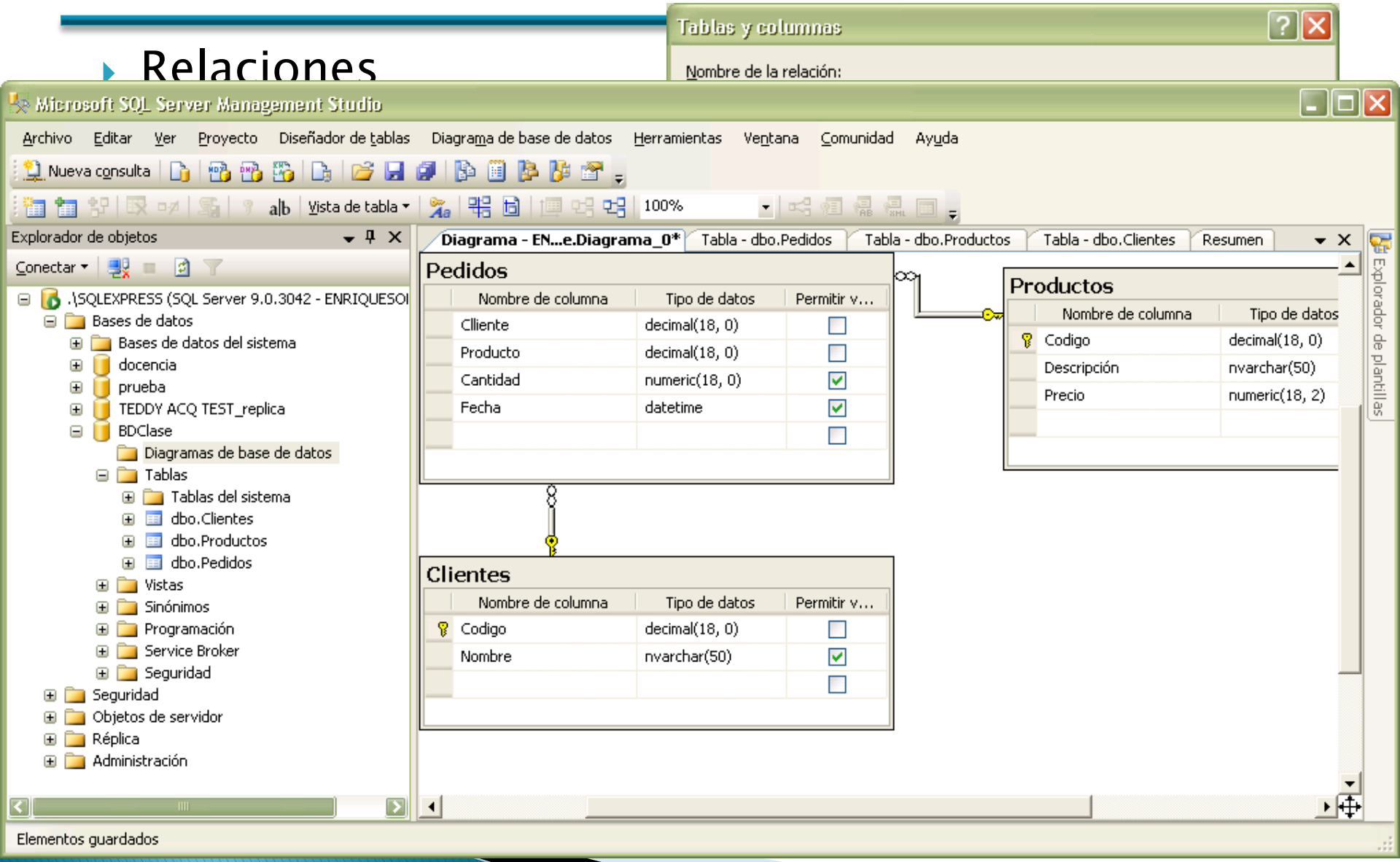
# SQL Server Management Studio

- Creación de Tablas



# SQL Server Management Studio

- Relaciones



# SQLCMD

- La utilidad **sqlcmd** permite introducir sentencias Transact-SQL statements, procedimientos del sistema y ficheros de script a través del prompt de comandos ofrecido al ejecutar dicha utilidad desde una ventana de comandos de MS-DOS (Cmd.exe).

```
sqlcmd -S host\sqlexpress
```

# SQLCMD

- Ejemplo. Crear una BD y una tabla usando SQLCMD

```
C:\Users\pastrana>sqlcmd -S GOTTEN\SQLEXPRESS
1> create database BD1
2> GO
1> USE BD1
2> go
1> create table estudiante( ID int, nombre varchar(40), nota
  int )
2> go
```

# Ejemplos SQLCMD

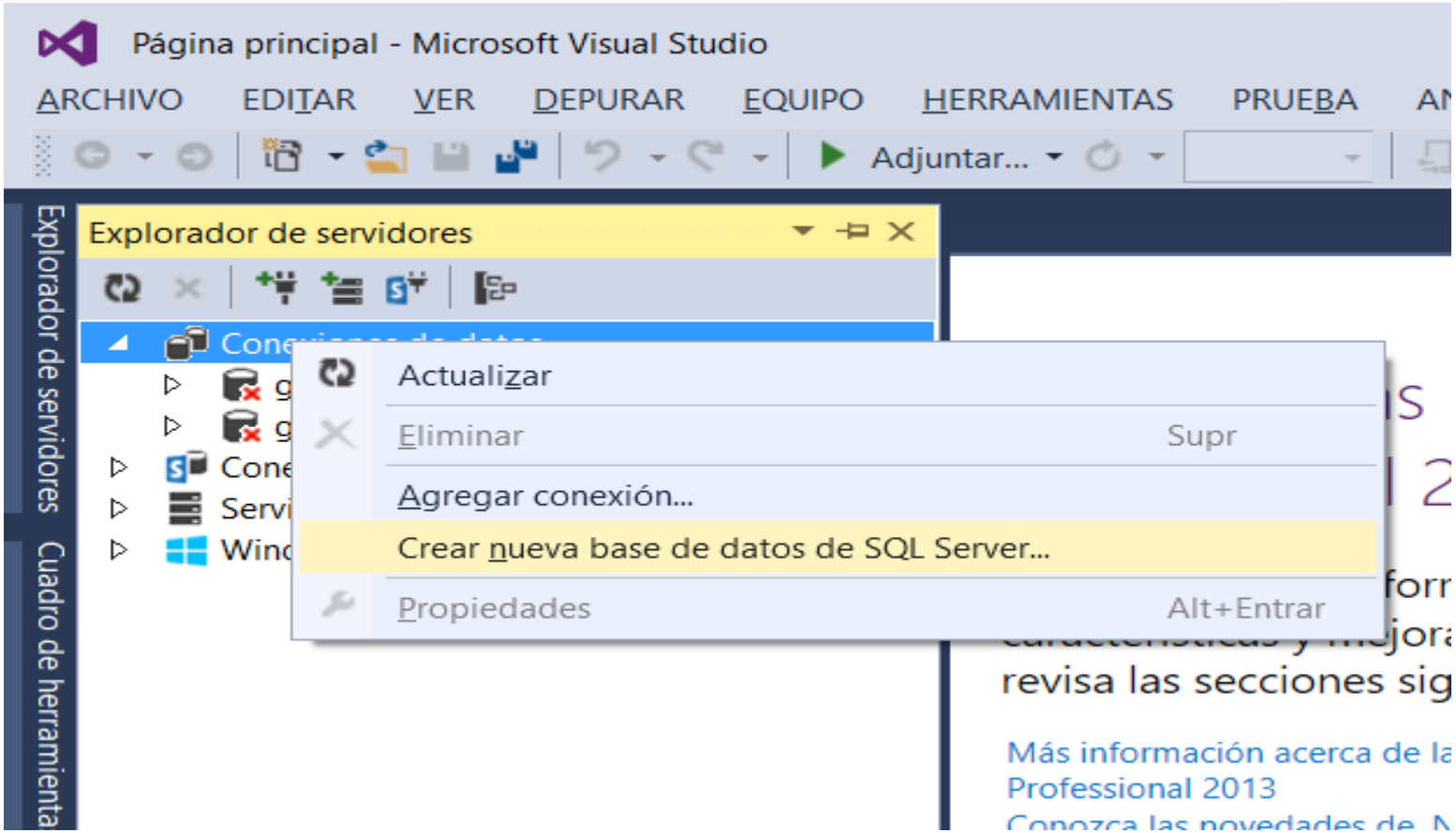
```
insert into estudiante values(1,"e1",null)
insert into Pruebas (ID_ALUMNO ,nombre_prueba, nota_prueba)
  values(1,"p1",7)
insert into Pruebas (ID_ALUMNO ,nombre_prueba, nota_prueba)
  values(1,"p2",9)
select * from Estudiante where nombre LIKE "%e%"
select * from Pruebas where nota_prueba >5
select * from Pruebas where nota_prueba between 7 AND 9
select Estudiante.nombre, Pruebas.nombre_prueba, Pruebas.nota_prueba FROM
  Estudiante INNER JOIN Pruebas ON Estudiante.ID = Pruebas.ID_ALUMNO
select Estudiante.nombre, Pruebas.nombre_prueba, Pruebas.nota_prueba FROM
  Estudiante INNER JOIN Pruebas ON Estudiante.ID = Pruebas.ID_ALUMNO WHERE
  nota_prueba >5
update Pruebas set nota_prueba = nota_prueba + 2
update Pruebas set nota_prueba = 10 where nota_prueba > 10
```

# Visual Studio 2013

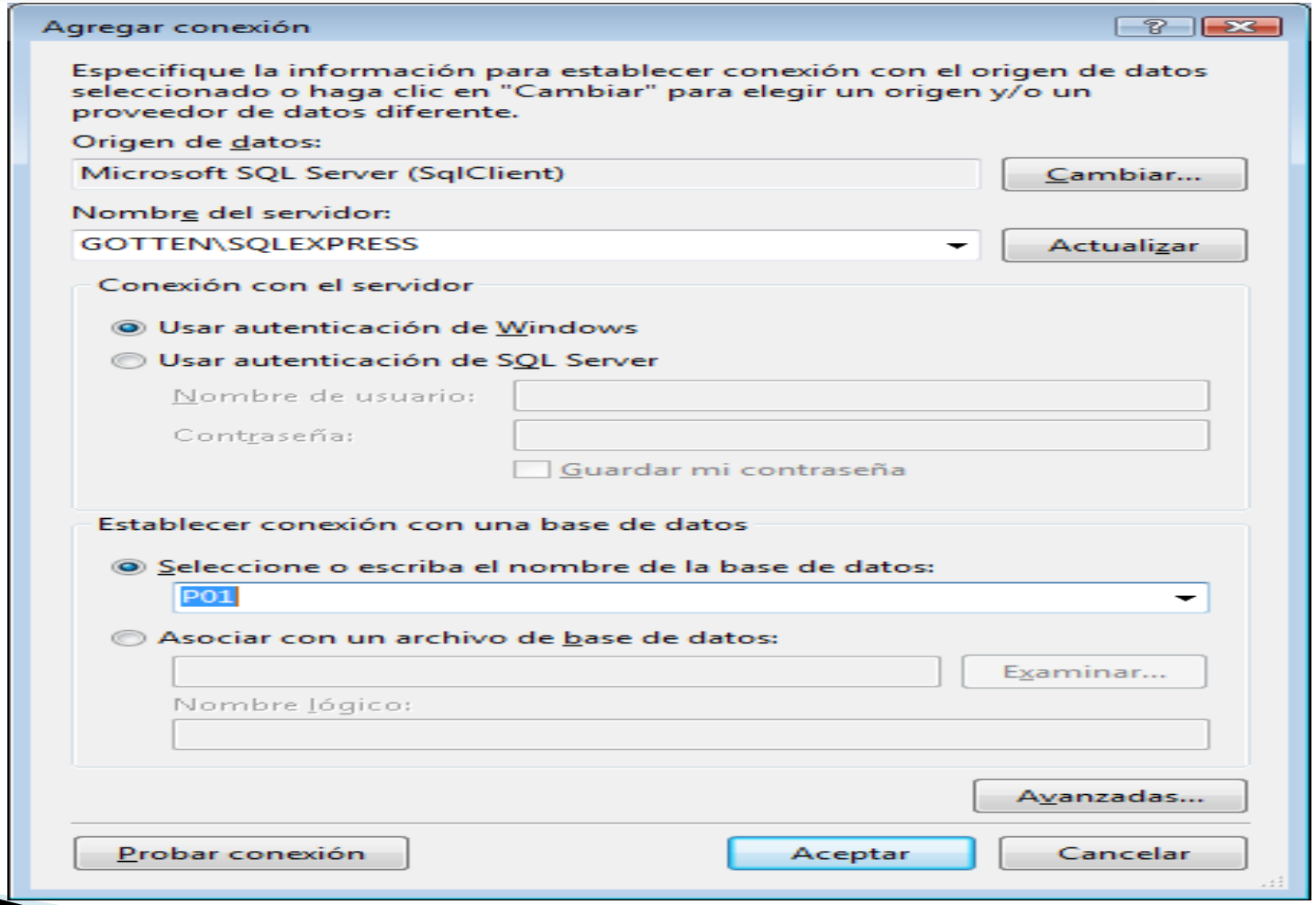
- Para crear una base de datos nueva en SQL-Server usando Microsoft Visual Studio 2013, debemos ir al explorador de servidores y con el botón derecho seleccionar “Crear nueva base de datos de SQL Server”.

- Para trabajar con una Base de Datos ya creada en Visual Studio 2013 habrá de agregar una nueva conexión de base de datos SQLSERVER.

# Visual Studio 2013



# Visual Studio 2013



# T-SQL

- T-SQL es el lenguaje usado por SQL Server.
- Las consultas realizadas con T-SQL se dividen en 3 categorías:

1. **Data Definition Language (DDL)** : Las sentencias de definición de datos se usan para crear y manejar los objetos en la base de datos: bases de datos, tablas, índices, vistas, procedimientos almacenados, etc. Por ejemplo: `CREATE` , `ALTER` , and `DROP` .
2. **Data Control Language (DCL)** : Controlan la seguridad y los permisos de los usuarios. Por ejemplo: `GRANT` , `REVOKE` , and `DENY` .
3. **Data Manipulation Language (DML)**: las sentencias de manipulación de datos se usan para trabajar con los datos. Por ejemplo: `SELECT` , `INSERT` , `UPDATE` , and `DELETE` .

# T-SQL

## Data Definition Language (DDL)

- *CREATE TABLE*

```
CREATE TABLE
[ database_name.[ owner ] . | owner. ]
  table_name
(    { < column_definition >
  | column_name AS computed_column_expression
  | < table_constraint > ::= [ CONSTRAINT
     constraint_name ] }
  | [ { PRIMARY KEY | UNIQUE }
)
```

# T-SQL

**Restricciones(Constraint)**

- `Not Null` :Asegura que esa columna no puede tomar valores nulos.
- `Primary Key`: Define esa columna/s como clave primaria. Debe tomar valores únicos y no admite valores nulos.
- `Check`: Valida una tupla en base la valor de una columna para identificar valores aceptables.
- `Unique`: Requiere que cada valor de dicha columna sea único. Acepta el valor nulo sólo en una tupla.
- `Foreign Key`: Fuerza la integridad referencial comprobando el valor de una columna frente a la clave primaria de la tabla relacionada.

# T-SQL

- CREATE VIEW

```
CREATE VIEW view_name [ ( column [ ,... n ] ) ]
[ WITH [ENCRYPTION | SCHEMABINDING |
  VIEW_METADATA] ]
AS
select_statement
[ WITH CHECK OPTION ]
```

# T-SQL

- ALTER TABLE
- Modifica una definición de tabla al alterar, agregar o quitar columnas y restricciones, reasignar particiones, o deshabilitar o habilitar restricciones y desencadenadores.

```
ALTER TABLE [ database_name . [ schema_name ] . | schema_name . ] table_name
{ ALTER COLUMN column_name
   ADD
    { <column_definition> | <computed_column_definition> | <table_constraint>
        | <column_set_definition>
    }
    | DROP
    { [CONSTRAINT] constraint_name [ WITH (<drop_clustered_constraint_option>)]
      | COLUMN column_name
    }
 | (<table_option>)
}
```

# T-SQL

- DROP TABLE.
- Quita una o varias definiciones de tabla y todos los datos, índices, desencadenadores, restricciones y especificaciones de permisos de esas tablas.
- Las vistas o procedimientos almacenados que hagan referencia a la tabla quitada se deben quitar explícitamente con DROP VIEW o DROP PROCEDURE

```
DROP TABLE [ database_name . [ schema_name ] .
  | schema_name . ] table_name
```

# T-SQL

## Data Control Language (DCL)

- *GRANT.*
- Concede permisos sobre un elemento protegible a una entidad de seguridad.
- El concepto general es GRANT <algún permiso> ON <algún objeto> TO <algún usuario, inicio de sesión o grupo>.

```
GRANT { ALL [ PRIVILEGES ] }
    | permission [ ( column [ ,...n ] ) ]
    [ ON [ class :: ] securable ] TO principal
    [ WITH GRANT OPTION ] [ AS principal ]
```

# T-SQL

- *DENY.*
- Deniega un permiso a una entidad de seguridad.
- Evita que la entidad de seguridad herede permisos por su pertenencia a grupos o roles .

```
DENY { ALL [ PRIVILEGES ] }
    | permission [ ( column [ ,...n ] ) ]
    [ ON [ class :: ] securable ] TO principal
    [ CASCADE] [ AS principal ]
```

# T-SQL

- *REVOKE.*
- Quita un permiso concedido o denegado previamente.

```
REVOKE [ GRANT OPTION FOR ]
    {
        [ ALL [ PRIVILEGES ] ]
        | permission [ ( column [ ,...n ] ) ]
    }
    [ ON [ class :: ] securable ]
    { TO | FROM } principal
    [ CASCADE] [ AS principal ]
```

# T-SQL

## Data Manipulation Language (DML)

- INSERT
- Agrega una o varias filas nuevas a una tabla o una vista.

```
INSERT
{
   [ INTO ] { <object> }
   [ ( column_list ) ]
   { VALUES ( { DEFAULT | NULL | expression }) }
}
```

# T-SQL

- DELETE
- Quita filas de una tabla o vista.

```
DELETE
    [ FROM <table_source> ]
    [ WHERE { <search_condition> } ]
```

# T-SQL

- SELECT
- Recupera filas de la base de datos. La sintaxis completa de la instrucción SELECT es compleja, aunque las cláusulas principales se pueden resumir del modo siguiente:

```
SELECT select_list [ INTO new_table ]
[ FROM table_source ] [ WHERE search_condition ]
[ GROUP BY group_by_expression ]
[ HAVING search_condition ]
[ ORDER BY order_expression [ ASC | DESC ] ]
```

- Los operadores UNION, EXCEPT e INTERSECT se pueden utilizar entre consultas para combinar o comparar resultados en un conjunto de resultados

# T-SQL

- UPDATE
- Cambia los datos existentes en una o varias columnas de una tabla o vista.

```
UPDATE
    SET
     { column_name = { expression
                     | DEFAULT
                     | NULL }
     }
     [ FROM { <table_source> }
     [ WHERE { <search_condition> } ]
```

# T-SQL. Procedimientos Almacenados

Los procedimientos almacenados de Microsoft SQL Server son similares a los procedimientos de otros lenguajes de programación en el sentido de que pueden:

- Aceptar parámetros de entrada y devolver varios valores en forma de parámetros de salida a quien realiza la llamada.
- Contener instrucciones de programación que realicen operaciones en la base de datos, incluidas las llamadas a otros procedimientos.
- Devolver un valor de estado para indicar si la operación se ha realizado correctamente o se han producido errores (y el motivo).
- Puede utilizar la instrucción EXECUTE de Transact-SQL para ejecutar un procedimiento almacenado. Los procedimientos almacenados difieren de las funciones en que no devuelven valores en lugar de sus nombres ni pueden utilizarse directamente en una expresión.

# T-SQL. Procedimientos Almacenados

- Permiten una programación modular.
- Puede crear el procedimiento una vez y llamarlo desde el programa tantas veces como desee. Así, puede mejorar el mantenimiento de la aplicación y permitir que las aplicaciones tengan acceso a la base de datos de manera uniforme.
- Constituyen código con nombre que permite el enlace diferido.
- Esto proporciona un nivel de direccionamiento indirecto que facilita la evolución del código.
- Pueden reducir el tráfico de red.
- Una operación que necesite centenares de líneas de código Transact-SQL puede realizarse mediante una sola instrucción que ejecute el código en un procedimiento, en vez de enviar cientos de líneas de código por la red.

# T-SQL. Procedimientos Almacenados

**Reglas para diseñar procedimientos almacenados.**

- La propia definición de `CREATE PROCEDURE` puede incluir cualquier número y tipo de instrucciones SQL, excepto : `CREATE AGGREGATE, CREATE RULE, CREATE DEFAULT, CREATE SCHEMA, CREATE o ALTER FUNCTION, CREATE o ALTER TRIGGER, CREATE o ALTER PROCEDURE, CREATE o ALTER VIEW, SET PARSEONLY, SET SHOWPLAN_ALL, SET SHOWPLAN_TEXT, SET SHOWPLAN_XML, USE database_name`.
- Puede crear otros objetos de base de datos dentro de un procedimiento almacenado.

# T-SQL. Procedimientos Almacenados

- Puede hacer referencia a un objeto creado en el mismo procedimiento almacenado, siempre que se haya creado antes de hacer referencia a él.
- Puede hacer referencia a tablas temporales dentro de un procedimiento almacenado.
- Si crea una tabla temporal local dentro de un procedimiento almacenado, ésta existirá únicamente para los fines del procedimiento y desaparecerá cuando éste finalice.
- Si ejecuta un procedimiento almacenado que llama a otro procedimiento almacenado, este último puede tener acceso a todos los objetos creados por el primero, incluidas las tablas temporales.

# T-SQL. Procedimientos Almacenados

- Si ejecuta un procedimiento almacenado remoto que realiza cambios en una instancia remota de Microsoft SQL Server, los cambios no se pueden revertir. Los procedimientos almacenados remotos no intervienen en las transacciones.
- El número máximo de parámetros en un procedimiento almacenado es de 2100.
- El número máximo de variables locales en un procedimiento almacenado está limitado únicamente por la memoria disponible.
- En función de la memoria disponible, el tamaño máximo de un procedimiento almacenado es de 128 megabytes (MB).

# T-SQL. Procedimientos Almacenados

- Sintaxis

```
CREATE {PROC|PROCEDURE} [schema_name.] procedure_name [;number]
    [ { @parameter [ type_schema_name. ] data_type }
        [ VARYING ] [ = default ] [ OUT | OUTPUT ] [READONLY]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }

<procedure_option> ::= [ ENCRYPTION ] [ RECOMPILE ]
                       [ EXECUTE AS Clause ]
```

# T-SQL. Procedimientos Almacenados

## Argumentos

- `schema_name`: El nombre del esquema al que pertenece el procedimiento. Si no se especifica el nombre del esquema cuando se crea el procedimiento, se asigna automáticamente el esquema predeterminado del usuario que crea este procedimiento.
- `procedure_name`: El nombre del procedimiento. Los nombres de los procedimientos deben cumplir las reglas de los identificadores y deben ser exclusivos en el esquema. Evite el uso del prefijo sp_ cuando asigne nombre a los procedimientos. SQL Server usa este prefijo para designar los procedimientos del sistema. El nombre completo de un procedimiento no puede superar los 128 caracteres.
- `;number`: Entero opcional que se usa para agrupar procedimientos con el mismo nombre. Estos procedimientos agrupados se pueden quitar juntos mediante una instrucción DROP PROCEDURE.

# T-SQL. Procedimientos Almacenados

- `@parameter`: Parámetro declarado en el procedimiento. Consiste en el nombre de parámetro precedido de una arroba (@).
- Se debe ajustar a las reglas de los identificadores.
- Los parámetros son locales respecto al procedimiento.
- Se pueden declarar uno o varios; el valor máximo es 2.100.
- El usuario debe proporcionar el valor de cada parámetro declarado cuando se llame al procedimiento, a menos que se haya definido un valor predeterminado para el parámetro o se haya establecido en el mismo valor que el de otro parámetro.
- Los parámetros solo pueden ocupar el lugar de expresiones constantes; no se pueden usar en lugar de nombres de tabla, nombres de columna o nombres de otros objetos.
- No se pueden declarar los parámetros si se especifica `FOR REPLICATION`.

# T-SQL. Procedimientos Almacenados

- `[ type_schema_name. ] data_type`: El tipo de datos del parámetro y el esquema al que pertenece el tipo de datos.
- `VARYING`: Especifica el conjunto de resultados admitido como parámetro de salida.
- `default`: Valor predeterminado de un parámetro.
- `OUT | OUTPUT`: Indica que se trata de un parámetro de salida. Los parámetros text, ntext e image no se pueden usar como parámetros OUTPUT
- `READONLY`: Indica que el parámetro no se puede actualizar ni modificar en el cuerpo del procedimiento.
- `RECOMPILE`: Indica que el Motor de base de datos no almacena en caché ningún plan de consulta para este procedimiento, forzándolo a ser compilado cada vez que se ejecute.

# T-SQL. Procedimientos Almacenados

- `ENCRYPTION`: Indica que SQL Server convertirá el texto original de la instrucción `CREATE PROCEDURE` en un formato “confuso” (ofuscado).
- `EXECUTE AS`: Especifica el contexto de seguridad en el que se ejecuta el procedimiento.
- `FOR REPLICATION`: Especifica que el procedimiento se crea para replicación. Por consiguiente, no se puede ejecutar en el suscriptor. Se usa un procedimiento creado con la opción FOR REPLICATION como filtro de procedimiento y solo se ejecuta durante la replicación.

# T-SQL. Procedimientos Almacenados

**Recomendaciones**

- Use la instrucción SET NOCOUNT ON como la primera instrucción del cuerpo del procedimiento. De esta forma, se desactivan los mensajes que devuelve SQL Server al cliente y el rendimiento general mejora si se elimina esta sobrecarga de red innecesaria.
- Use nombres de esquemas cuando cree o haga referencia a los objetos de base de datos del procedimiento. El tiempo de procesamiento será menor para que Motor de base de datos resuelva los nombres de los objetos si no tiene que buscar en varios esquemas.
- Evite las funciones de ajuste en las columnas especificadas en las cláusulas WHERE y JOIN. De esta forma, las columnas no son deterministas y se evita que el procesador de consultas use índices.

# T-SQL. Procedimientos Almacenados

- Evite usar funciones escalares en instrucciones SELECT que devuelvan muchas filas de datos. Dado que la función escalar se debe aplicar a todas las filas, el comportamiento resultante es similar al procesamiento basado en filas y degrada el rendimiento.
- Evite el uso de SELECT *. En su lugar, especifique los nombres de columna necesarios.
- Evite el procesamiento o la devolución de demasiados datos. Envíe únicamente los datos fundamentales a la aplicación cliente.
- Use transacciones explícitas mediante el uso de BEGIN/END TRANSACTION y mantenga las transacciones lo más cortas posible. Las transacciones más largas significan bloqueos de registro más largos y mayores posibilidades de interbloqueos.
- Use la característica TRY…CATCH de Transact-SQL para el control de errores dentro de un procedimiento

# T-SQL. Procedimientos Almacenados

- Use la palabra clave DEFAULT en todas las columnas de la tabla a las que haga referencia en las instrucciones CREATE TABLE o ALTER TABLE. De esta forma, se evita pasar el valor NULL a columnas que no admiten valores NULL.
- Use NULL o NOT NULL para todas las columnas de una tabla temporal.
- Use instrucciones de modificación que conviertan valores NULL e incluya lógica que elimine filas con valores NULL de las consultas.
- Use el operador UNION ALL en vez de los operadores UNION u OR, a menos que exista una necesidad específica de valores distintos. El operador UNION ALL necesita menos sobrecarga de procesamiento porque no se filtran los duplicados del conjunto de resultados.

# T-SQL. Procedimientos Almacenados

- **Ejemplos**

```
CREATE PROCEDURE HumanResources.uspGetEmployees
    @LastName nvarchar(50),
    @FirstName nvarchar(50)
AS
    SET NOCOUNT ON;
    SELECT FirstName, LastName,Department
    FROM HumanResources.vEmployeeDepartmentHistory
    WHERE FirstName = @FirstName AND LastName = @LastName;
```

# T-SQL. Procedimientos Almacenados

```
CREATE PROCEDURE Production.uspGetList @Product varchar(40)
    , @MaxPrice money , @ComparePrice money OUTPUT
    , @ListPrice money OUT
AS
    SET NOCOUNT ON;
    SELECT p.[Name] AS Product, p.ListPrice AS 'ListPrice'
    FROM Production.Product AS p
    JOIN Production.ProductSubcategory AS s
      ON p.ProductSubcategoryID = s.ProductSubcategoryID
    WHERE s.[Name] LIKE @Product AND p.ListPrice < @MaxPrice;
```

# T-SQL. Procedimientos Almacenados

```
-- Populate the output variable @ListPprice.
SET @ListPrice = (SELECT MAX(p.ListPrice)
        FROM Production.Product AS p
        JOIN  Production.ProductSubcategory AS s
          ON p.ProductSubcategoryID = s.ProductSubcategoryID
        WHERE s.[Name] LIKE @Product AND
              p.ListPrice < @MaxPrice);
-- Populate the output variable @compareprice.
SET @ComparePrice = @MaxPrice;
```

# *T-SQL.* Funciones definidas por el usuario

- Al igual que las funciones en los lenguajes de programación, las funciones definidas por el usuario de Microsoft SQL Server son rutinas que aceptan parámetros, realizan una acción, como un cálculo complejo, y devuelven el resultado de esa acción como un valor.
- El valor devuelto puede ser un valor escalar único o un conjunto de resultados.
- Permiten una programación modular.
- Puede crear la función una vez, almacenarla en la base de datos y llamarla desde el programa tantas veces como desee.
- Permiten una ejecución más rápida.

# *T-SQL.* Funciones definidas por el usuario

- Al igual que los procedimientos almacenados, las funciones definidas por el usuario Transact-SQL reducen el costo de compilación del código Transact-SQL almacenando los planes en la caché y reutilizándolos para ejecuciones repetidas.
- Pueden reducir el tráfico de red.
- Una operación que filtra datos basándose en restricciones complejas que no se puede expresar en una sola expresión escalar se puede expresar como una función. La función se puede invocar en la cláusula WHERE para reducir el número de filas que se envían al cliente.

# *T-SQL.* Funciones definidas por el usuario

- Todas las funciones definidas por el usuario tienen dos partes: un encabezado y un cuerpo.
- El encabezado define:
  - Nombre de función con nombre de propietario o esquema opcional
  - Nombre del parámetro de entrada y tipo de datos
  - Opciones aplicables al parámetro de entrada
  - Tipo de datos de parámetro devueltos y nombre opcional
  - Opciones aplicables al parámetro devuelto
- El cuerpo define la acción o la lógica que la función va a realizar. Contiene:
  - Una o más instrucciones Transact-SQL que ejecutan la lógica de la función

# *T-SQL.* Funciones definidas por el usuario

Funciones escalares.

- Devuelven un único valor de datos del tipo definido en la cláusula RETURNS.
- En una función escalar insertada no hay cuerpo de la función; el valor escalar es el resultado de una sola instrucción.
- Para una función escalar de varias instrucciones, el cuerpo de la función, definido en un bloque BEGIN...END, contiene una serie de instrucciones de Transact-SQL que devuelven el único valor.
- El tipo devuelto puede ser de cualquier tipo de datos excepto text, ntext, image, cursor y timestamp.

# *T-SQL.* Funciones definidas por el usuario

Funciones escalares. Ejemplo

```
CREATE FUNCTION dbo.ufnGetInventoryStock(@ProductID int)
RETURNS int
AS
BEGIN
    DECLARE @ret int;
    SELECT @ret = SUM(p.Quantity)
    FROM Production.ProductInventory p
    WHERE p.ProductID = @ProductID AND p.LocationID = '6';
     IF (@ret IS NULL)
        SET @ret = 0;
    RETURN @ret;
END;
```

# *T-SQL.* Funciones definidas por el usuario

Funciones con valores de tabla.

- Las funciones con valores de tabla definidas por el usuario devuelven un tipo de datos table.

- Las funciones con valores de tabla insertados no tienen cuerpo; la tabla es el conjunto de resultados de una sola instrucción SELECT.

# *T-SQL.* Funciones definidas por el usuario

Funciones con valores de tabla. Ejemplo

```
CREATE FUNCTION Sales.ufn_SalesByStore (@storeid int)
RETURNS TABLE
AS
RETURN
( SELECT P.ProductID, P.Name, SUM(SD.LineTotal) AS 'Total‘
FROM Production.Product AS P
JOIN Sales.SalesOrderDetail AS SD ON SD.ProductID =P.ProductID
JOIN Sales.SalesOrderHeader AS SH ON SH.SalesOrderID=SD.SalesOrderID
JOIN Sales.Customer AS C ON SH.CustomerID = C.CustomerID
WHERE C.StoreID = @storeid  GROUP BY P.ProductID, P.Name );
```

# *T-SQL.* Funciones definidas por el usuario

Funciones integradas.

- SQL Server proporciona funciones integradas para ayudarle a realizar diversas operaciones como:
- Tener acceso a información de las tablas del sistema de SQL Server sin tener acceso a las tablas del sistema directamente.
- Realizar tareas habituales como SUM, GETDATE o IDENTITY.
- Las funciones integradas devuelven tipos de datos escalares o table. Por ejemplo, @@ERROR devuelve 0 si la última instrucción Transact-SQL se ejecutó correctamente. Si la instrucción generó un error, @@ERROR devuelve el número de error.

# T-SQL. Lenguaje de control de flujo

Variables

SQL Server proporciona las siguientes instrucciones para declarar y establecer variables locales.

- Declare @local_variable
- SET @local_variable
- SELECT @local_variable

# T-SQL. Lenguaje de control de flujo

Declare @local_variable

- Se declaran en el cuerpo de un procedimiento con la instrucción DECLARE, y se les asignan valores con una instrucción SET o SELECT.
- Todas las variables se inicializan como NULL, salvo que se proporcione un valor en la declaración.
- Ejemplos:

```
DECLARE @find varchar(30) = 'Man%';
DECLARE @Group nvarchar(50), @Sales money;
SET @Group = N'North America';
SET @Sales = 2000000;
DECLARE @MyTableVar table( EmpID int NOT NULL,
    OldVacationHours int, NewVacationHours int,
    ModifiedDate datetime);
```

# T-SQL. Lenguaje de control de flujo

Set@local_variable

- Establece en el valor especificado la variable local indicada, creada previamente con la instrucción DECLARE @local_variable.
- Ejemplos:

```
/* Example one */
DECLARE @NewBalance int ;
SET @NewBalance = 10;
SET @NewBalance = @NewBalance * 10;
/* Example Two */
DECLARE @NewBalance int = 10;
SET @NewBalance *= 10;
```

# T-SQL. Lenguaje de control de flujo

Select@local_variable

- La variable local especificada que se crea con DECLARE @local_variable debe establecerse en la expresión especificada.
- SELECT @local_variable se suele utilizar para devolver un único valor a la variable. Si la instrucción SELECT devuelve más de un valor, se asigna a la variable el último valor devuelto. Si no devuelve filas, la variable mantiene su valor actual. Si expression es una subconsulta escalar que no devuelve ningún valor, la variable se establece en NULL.
- Ejemplo:

```
DECLARE @var1 nvarchar(30);
SELECT @var1 = Name FROM Sales.Store
                    WHERE CustomerID = 1000 ;
```

# T-SQL. Lenguaje de control de flujo

BEGIN...END

- Encierra un conjunto de instrucciones T-SQL.
- Los bloques BEGIN...END pueden anidarse.
- Todas las instrucciones T-SQL son válidas en un bloque BEGIN...END

Sintaxis

```
BEGIN
      {
         sql_statement | statement_block
      }
END
```

# T-SQL. Lenguaje de control de flujo

RETURN

- Sale incondicionalmente de una consulta o procedimiento.
- Las instrucciones que siguen a RETURN no se ejecutan.
- Todos los procedimientos almacenados del sistema devuelven el valor 0 si son correctos y un valor distinto de cero si se ha producido un error.

Sintaxis

```
RETURN [ integer_expression ]
```

# T-SQL. Lenguaje de control de flujo

## RETURN. Ejemplo

```
CREATE PROCEDURE findjobs @nm sysname = NULL
AS
IF @nm IS NULL
    BEGIN
        PRINT 'You must give a user name'
        RETURN
    END
ELSE
    BEGIN
        SELECT o.name, o.id, o.uid FROM sysobjects o INNER JOIN
    master..syslogins l ON o.uid = l.sid WHERE l.name = @nm
END;
```

# T-SQL. Lenguaje de control de flujo

WAITFOR

- Bloquea la ejecución de un procedimiento almacenado o una transacción hasta alcanzar la hora o el intervalo de tiempo especificado, o hasta que una instrucción especificada modifique o devuelva al menos una fila.

Sintaxis

```
WAITFOR
{
    DELAY 'time_to_pass'
  | TIME 'time_to_execute'
  | [ ( receive_statement ) | ( get_conversation_group_statement ) ]
    [ , TIMEOUT timeout ]
}
```

# T-SQL. Lenguaje de control de flujo

**Argumentos**

`DELAY`: Es el período de tiempo especificado (hasta un máximo de 24 horas) que debe transcurrir antes de la ejecución.

`'time_to_pass'`: Es el período de tiempo que hay que esperar.

`TIME`: Es la hora especificada a la que se ejecuta.

`'time_to_execute'`: Es la hora a la que termina la instrucción WAITFOR.

`receive_statement`: Es una instrucción RECEIVE válida.

`TIMEOUT timeout`: Especifica el período de tiempo, en milisegundos, que se espera a que llegue un mensaje en la cola.

# T-SQL. Lenguaje de control de flujo

WAITFOR. Ejemplo:

```
USE msdb;
EXECUTE sp_add_job @job_name = 'TestJob';
BEGIN
    WAITFOR TIME '22:20';
    EXECUTE sp_update_job @job_name = 'TestJob',
        @new_name = 'UpdatedJob';
END;
GO
```

# T-SQL. Lenguaje de control de flujo

TRY...CATCH

- Implementa un mecanismo de control de errores para T-SQL.
- Se puede incluir un grupo de instrucciones T-SQL en un bloque TRY.
- Si se produce un error en el bloque TRY, el control se transfiere a otro grupo de instrucciones que está incluido en un bloque CATCH.

Sintaxis

```
BEGIN TRY
      { sql_statement | statement_block }
END TRY
BEGIN CATCH
      [ { sql_statement | statement_block } ]
END CATCH
```

# T-SQL. Lenguaje de control de flujo

## TRY...CATCH

- Una construcción TRY...CATCH detecta todos los errores de ejecución que no cierran la conexión de la base de datos.
- Un bloque TRY debe ir seguido inmediatamente por un bloque CATCH asociado. Si se incluye cualquier otra instrucción entre las instrucciones END TRY y BEGIN CATCH se genera un error de sintaxis.
- TRY…CATCH no puede abarcar varios bloques de instrucciones
- Si no hay errores en el código incluido en un bloque TRY, cuando la última instrucción de este bloque ha terminado de ejecutarse, el control se transfiere a la instrucción inmediatamente posterior a la instrucción END CATCH asociada.

# T-SQL. Lenguaje de control de flujo

- Cuando finaliza el código del bloque CATCH, el control se transfiere a la instrucción inmediatamente posterior a la instrucción END CATCH.
- Los errores capturados por un bloque CATCH no se devuelven a la aplicación que realiza la llamada.
- Si es necesario devolver cualquier parte de la información sobre el error a la aplicación, debe hacerlo el código del bloque CATCH a través de mecanismos como los conjuntos de resultados SELECT o las instrucciones RAISERROR y PRINT.
- Si el procedimiento almacenado no contiene su propia construcción TRY…CATCH, el error devuelve el control al bloque CATCH asociado al bloque TRY que contiene la instrucción EXECUTE.

# T-SQL. Lenguaje de control de flujo

- La construcción TRY…CATCH no se puede utilizar en una función definida por el usuario.
- ERROR_NUMBER() devuelve el número del error.
- ERROR_SEVERITY() devuelve la gravedad.
- ERROR_STATE() devuelve el número de estado del error.
- ERROR_PROCEDURE() devuelve el nombre del procedimiento almacenado o desencadenador donde se produjo el error.
- ERROR_LINE() devuelve el número de línea de la rutina que provocó el error.
- ERROR_MESSAGE() devuelve el texto completo del mensaje de error. Este texto incluye los valores suministrados para los parámetros reemplazables, como longitudes, nombres de objetos u horas.

# T-SQL. Lenguaje de control de flujo

IF...ELSE

- Impone condiciones en la ejecución de una instrucción T-SQL.

Sintaxis

```
IF Boolean_expression
    { sql_statement | statement_block }
[ ELSE
    { sql_statement | statement_block } ]
```

- Una construcción IF...ELSE puede utilizarse en procedimientos almacenados y en consultas ad hoc.
- Las pruebas IF pueden estar anidadas después de otra área IF o a continuación de un área ELSE.

# T-SQL. Lenguaje de control de flujo

IF...ELSE. Ejemplo

```
DECLARE @compareprice money, @cost money
EXECUTE Production.uspGetList '%Bikes%', 700,
        @compareprice OUT,  @cost OUTPUT
IF @cost <= @compareprice
BEGIN
    PRINT 'These products can be purchased for less than
    $'+RTRIM(CAST(@compareprice AS varchar(20)))+'.'
END
ELSE
    PRINT 'The prices for all products in this category exceed
    $'+ RTRIM(CAST(@compareprice AS varchar(20)))+'.
```

# T-SQL. Lenguaje de control de flujo

WHILE

- Las instrucciones se ejecutan repetidamente siempre que la condición especificada sea verdadera. Se puede controlar la ejecución de instrucciones del con BREAK y CONTINUE.

Sintaxis

```
WHILE Boolean_expression
  { sql_statement | statement_block | BREAK | CONTINUE }
```

- `BREAK`: Produce la salida del bucle WHILE. Se ejecutan las instrucciones que aparecen después de la palabra clave END, que marca el final del bucle.
- `CONTINUE`: Hace que se reinicie el bucle WHILE y omite las instrucciones que haya después de la palabra clave `CONTINUE`.

# T-SQL. Lenguaje de control de flujo

WHILE. Ejemplo.

```
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
```