Nociones básicas de SQL Server

Instalación

El proceso de instalación es el típico de Windows. Sobre la pantalla principal de instalación se puede pulsar sobre ***Comprobación de configuración del sistema*** para verificar que nuestro equipo cumple los requisitos necesarios.

Pulsar después sobre Instalación. Comienza un proceso guiado en el que elegimos las siguientes opciones:

- Instalación nueva e independiente
- Marcar TODAS las opciones de instalación
- Instancia con nombre: SQLEXPRESS
- Cuentas de servicio: se indica que ejecute como nombre de cuenta a NT_AUTHORITY\SYSTEM. También se indica que SQLBrowser se inicie como servicio de forma automática
- En la siguiente pantalla pulsar sobre el botón Agregar usuario actual. Indicar que se desea una autenticación de windows

SQL Server Configuration Manager

Inicio-programas-Microsoft SQLServer 2008-Herramientas-Administración de configuración de SQL Server

Permite configurar, arrancar, parar, … los distintos servicios de SQL Server:

- **Agente SQL Server**: es un servicio encargado de automatizar distintas tareas (como backup, …). Está detenido por defecto y, según la documentación, es conveniente que arranque en el inicio de windows pero no está disponible en la versión SQLEXPRESS.
- **SQLServerBrowser:** Servicio que permite el acceso desde otras máquinas al servidor de bases de datos. Por defecto está inhabilitado, así que hay que seguir los siguentes pasos:
  - Boton derecho: Propiedades - Pestaña Servicio - Modo de inicio: automático
  - Botón derecho: Iniciar

SQL Server Management Studio

Permite la administración del servidor. Se instala con el fichero SQLManagementStudio_x86_ESN.exe.

El proceso de instalación es idéntico al de SQLServer (ojo, que puede confundir porque parece que está reinstalando de nuevo el servidor). Da un problema de compatibilidad con W7 (ignorarlo y continuar con la instalación) y elegir las siguientes opciones:

- Instalación nueva e independiente (SI, SI, una nueva e independiente)
- Aceptar las distintas opciones por defecto, hasta que aparece en una pantalla REALIZAR UNA NUEVA INSTALACIÓN DE SQL SERVER 2008. (Parece que va a instalar un nuevo servidor, pero estará instalando el administrador)
- Debe aparecer la casilla Herramientas de administración básica. Marcarla y comenzará el proceso de instalación de SQLServer Management Studio

Para iniciarlo:

Programas - SQLServer2008 - SQLManagementStudio

- Tipo de servidor: Motor de Base de Datos
- Nombre: PC-XX\SQLEXPRESS
- Autenticación: De Windows

Entrará con las credenciales del usuario de windows que realizó la instalación. Conviene crear otro superadministrador con autenticación de SQL Server

---

Asistentes sentencias SQL

Hay asistentes para casi todo (Ver – Explorador de plantillas). Elegir una sentencia y arrastrarla sobre la pantalla de consulta.

Gestión de servidores remotos con Management Studio

1) Habilitar TCP/IP

- SQL Server Configuration Manager
- Configuración de red de SQL Server
- Protocolos de SQLExpress
- Habilitar TCP/IP

2) Permitir la autenticación de Windows y de SQLServer (usuarios de la BD)

- Seleccionar el servidor en Management Studio
- Boton derecho – propiedades - seguridad
- Modo de autenticación: de Windows y SQL Server

Reiniciar el servicio desde SQL Server Configuration Manager o con el botón derecho sobre el servidor:

- Control de servicios – Reiniciar

También se puede reiniciar desde Herramientas Administrativas – Servicios

3) Crear en SQL Server un usuario con permisos de superadministrador

- Servidor – Seguridad – Inicio de sesión – Nuevo inicio de sesión
  - Usuario: chema con password chema

- Otorgarle todas las funciones de servidor, todas las bases de datos asignadas y el estado concedido y habilitado si se desea hacer un administrador remoto

4) Conexión remota con Management Studio (Activar el servicio de SQL Server Browser, ver el apartado de SQLServer Configuration Manager)

- Tipo de servidor: Motor de Base de Datos
- Nombre: 192.168.0.X\SQLEXPRESS
- Autenticación: de SQL Server
- Inicio: user
- Contraseña: password

5) Si hay problemas de conexión (por ejemplo si no se instaló SQLBrowser) hay que fijar un puerto de escucha para SQLServer.

- SQL Server Configuration Manager
- Protocolos de SQL Express – TCP/IP - Propiedades
- Direcciones IP – IP All
- Borrar el valor de los puertos dinámicos TCP y poner un valor fijo en el puerto TCP (por ejemplo 2301). El puerto estándar es el 1433
- Conectar igual que el punto 4) excepto en el apartado Nombre, que se especificará como **192.168.0.16\SQLEXPRESS, 2301**

ODBC

Se crea con un asistente, en el que se indica que el tipo es SQLServer, el servidor será algo así como 192.168.0.16\SQLEXPRESS, la autenticación tipo SQLServer, usuario chema <chema>. Indicar una BD por defecto y probar el origen de datos

---

Permisos sobre elementos de una BD

Ejercicio: se desea que el usuario probando<probando> pueda ver y modificar los campos codigo y nombre de la tabla gente)

Crear el usuario con la siguiente secuencia: Se elige la BD – Seguridad – Usuarios – Nuevo. Dar un nombre y asignar a un inicio de sesión creado anteriormente

Elementos que pueden protegerse – Botón buscar – Elegir el tipo y el nombre del elemento que pretendemos proteger (por ejemplo Tablas) Dar permisos sobre acciones (Permitir/denegar sobre eliminar, modificar, seleccionar, …) Dentro de cada apartado se puede pulsar sobre 'Permisos de columna' para permitir/denegar acciones sobre determinados campos. Aceptar y el usuario correspondiente podrá realizar las acciones permitidas

---

Diagramas de las BD

Elegir la BD – Carpeta 'Diagramas de base de datos' --> nuevo

En la pantalla del diagrama se pueden crear tablas directamente, así como las relaciones (FK)

Analizador de SQL

Cuando se introduce SQL se puede pulsar cualquier botón del menú superior para realizar diversas funciones:

- Ejecutar el comando SQL
- Depurar la consulta.
- Detener la depuración
- Verificar la sintaxis SQL
- Mostrar el plan de ejecución
- Otros

Tipos de datos

El conjunto de tipos de datos existente lo podemos ver en el explorador de objetos, en cada una de las bases de datos existentes:

tinyint, smallint, int, bigint

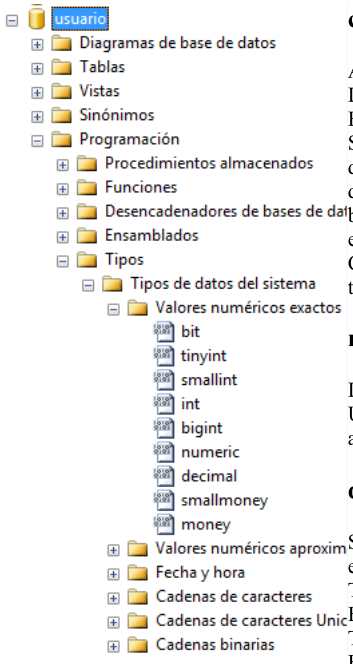
TINYINT 1 byte 0 a 255
SMALLINT 2 bytes -32.768 a 32.768
INT 4 bytes -2.147.483.648 a 2.147.483.648
BIGINT 8 bytes -9.223.372.036.854.775.808 a 9.223.372.036.854.775.808

binary y varbinary

Tipo de dato binario que tiene una longitud fija (binary [(n)]) o variable (varbinary [(n)]), donde n puede estar comprendido entre 1 y 8000.
El tamaño que ocupe al almacenar el dato será de n+4 bytes en binary y la longitud del dato+4 bytes en varbinary. Si no se especifican en la definición se entenderá n=1.

bit

Almacenará como valores posibles: 1, 0 o NULL.
Si en la tabla existen 8 columnas o menos de tipo de dato bit ocuparán todas ellas 1 byte, si hay entre 9 y 16 2 bytes y así sucesivamente.



char, varchar, text

Almacenan números, letras (mayúsculas o minúsculas) y caracteres especiales (@, &, !.....).
Longitud máxima 8000 caracteres, unos 8 KBytes.
El tipo de datos char es un tipo de datos de longitud fija cuando se especifica la cláusula NOT NULL.
Si en una columna char NOT NULL se inserta un valor más corto que la longitud de la columna, el valor se rellena a la derecha con blancos hasta completar el tamaño de la columna. Por ejemplo, si una columna se define como char(10) y el dato que se va a almacenar es "música", SQL Server almacena este dato como "música____" donde "_" indica un espacio en blanco. Esto sólo afectará al tamaño del campo almacenado, no por ejemplo al tamaño del campo (LEN()), o selección con ese valor en el WHERE.
Con varchar la longitud será variable pero inferior a 8Kbytes. Cuando el tamaño supere los 8Kbytes, utilizaremos text, el tamaño máximo de este campo será de hasta 2 GB.

nchar, ntext, nvarchar

Los tipos de datos anteriores registraban caracteres ASCII, con estos otros tipos de datos se utilizarán caracteres UNICODE UCS-2, que fundamentalmente se utiliza para representación de otros lenguajes. Ej.: la ñ es un carácter unicode, palabras acentuadas. El nombre proviene de national char, national text…

datetime, smalldatetime

Se trata de tipos de datos para representar fecha y hora. En datetime se almacenarán datos entre el 1 de enero de 1753 hasta el 31 de diciembre de 9999 con una precisión de 3,33 milisegundos o 0,00333 segundos. Ej.: 01/01/98 23:59:59.997
Tamaño: 4 bytes.
En smalldatetime se podrá almacenar desde el 1 de enero de 1900 hasta el 6 de junio de 2079, con una precisión de minutos.
Tamaño: 2 bytes.
Formatos válidos para especificar fechas:

Apr[il] [15][,] 1996
Apr[il] 15[,] [19]96
Apr[il] 1996 [15]
[15] Apr[il][,] 1996
15 Apr[il][,][19]96
15 [19]96 apr[il]
[15] 1996 apr[il]
1996 APR[IL] [15]
1996 [15] APR[IL]
Función útil: SET DATEFORMAT { format | @format_var } Definimos el formato de la fecha: mdy, dmy, ymd, ydm, myd y dym
Ejemplo:

```
-- Mes - dia - año.
SET DATEFORMAT mdy;
GO
DECLARE @datevar DATETIME;
SET @datevar = '12/31/1998';
SELECT @datevar AS DateVar;
GO
-- Ponerlo a año dia mes
SET DATEFORMAT ydm;
GO
DECLARE @datevar DATETIME;
SET @datevar = '1998/31/12';
SELECT @datevar AS DateVar;
GO
```

Al instalar SQL Server, se instala también la base de datos master, y en tablas del sistema de esta BD aparece la tabla syslanguages. Aquí podemos ver qué formato de fecha se está utilizando.

```
SELECT * FROM master.dbo.syslanguages
```

decimal y numeric

decimal[(p[, s])] y numeric[(p[, s])] donde p es la precisión y s la escala.

- P: Especifica el número máximo total de dígitos decimales que se pueden almacenar, a la izquierda y a la derecha del separador decimal.
  - La precisión debe ser un valor comprendido entre 1 y la precisión máxima.
  - La mayor precisión que se puede especificar es 38.
- S: Especifica el número máximo de dígitos decimales que se pueden almacenar a la derecha del separador decimal.
  - La escala debe ser un valor comprendido entre 0 y p.
  - La escala predeterminada es 0; por tanto, 0 <= s <= p.
  - Los tamaños máximos de almacenamiento varían según la precisión.

Nota.- decimal(5,5) y decimal(5,0) se consideran tipos de datos diferentes. Ambos son datos equivalentes.

Float y real

Se encarga de almacenar un número en punto flotante: - 1.79E + 308 y 1.79E +308.
4 bytes/8 bytes
La equivalencia entre los dos tipos de datos es : real=float(24)

image

Podremos almacenar imágenes en formatos BMP, TIFF, GIF o JPEG. Realmente son datos binarios de longitud variable desde 0 hasta 2^31-1 (2.147.483.647) bytes, son similares por lo tanto a binary y varbinary. Normalmente en una base de datos no ser insertan imágenes en ella, simplemente se utilizan rutas relativas del disco duro para posteriormente cargarlas en las diferentes aplicaciones. Se dice que crean mucha fragmentación en la BD.

money, smallmoney

money: valores de moneda comprendidos entre -2^63 (-922.337.203.685.477,5808) y 2^63 - 1 (+922.337.203.685.477,5807), con una precisión de una diezmilésima de la unidad monetaria.
Tamaño de almacenamiento 8 bytes.
smallmoney: valores de moneda comprendidos entre -214.748,3648 y +214.748,3647, con una precisión de una diezmilésima de la unidad monetaria.
Tamaño de almacenamiento 4 bytes.

sql_variant

Se trata de un tipo de datos que almacena valores de varios tipos de datos aceptados en SQL Server, excepto text, ntext, image, timestamp y sql_variant. Por lo tanto una columna del tipo sql_variant puede contener filas de tipos de datos diferentes.

timestamp

Es un tipo de datos que expone automáticamente números binarios generados, cuya exclusividad está garantizada en la base de datos. timestamp se suele utilizar como mecanismo para marcar la versión de las filas de la tabla. El tamaño de almacenamiento es de 8 bytes.

uniqueidentifier

El tipo de datos uniqueidentifier almacena valores binarios de 16 bytes que operan como identificadores exclusivos globales (GUID). Un GUID es un número binario exclusivo; ningún otro equipo del mundo generará un duplicado de ese GUID. El principal uso de un GUID se da cuando se asigna un identificador que debe ser exclusivo en una red que tiene muchos equipos en distintos emplazamientos.
Lo genera a partir del identificador de su tarjeta de red (MAC) más un número exclusivo del reloj de la CPU.

Ejemplo. Inserción de un nuevo valor sobre el campo uniqueidentifier:

```
USE AdventureWorks;
GO
IF OBJECT_ID ('dbo.T1', 'U') IS NOT NULL
DROP TABLE dbo.T1;
GO
CREATE TABLE dbo.T1(column_1 int IDENTITY,column_2 uniqueidentifier);
GO
INSERT INTO dbo.T1 (column_2) VALUES (NEWID());
INSERT INTO T1 DEFAULT VALUES;
GO
SELECT column_1, column_2 FROM dbo.T1;
GO
```

XML

Este tipo de datos se ha introducido en SQL Server 2008 dada su alta integración con este tipo de ficheros y las elevadas posibilidades de comunicación actualmente existentes con ficheros XML. Permite almacenar información en este formato de hasta 2 GB.

Los tipos de datos XML tienen como responsabilidad el garantizar que los documentos XML están bien formados, es decir, que son sintácticamente correctos.

Tipos de datos personalizados (definido por el usuario).

Este tipo de datos se agregará con la función sp_addtype o bien más fácilmente con el administrador corporativo.

Los tipos empid, id y tid únicamente varían char o varchar en su longitud Estos tipos de datos no aportan realmente demasiada funcionalidad, los tipos de datos generados a partir de clases serían realmente útiles. SQL Server 2008 sí permite la creación de este tipo de datos, y almacenar así objetos en un campo BD, pero se realiza mediante la definición de clases en C# o Visual Basic .net.

Ejemplo. Ejemplo de uso de un tipo de datos creado en C#

```
CREATE ASSEMBLY TipoPunto FROM 'c:\tipos\Punto.dll'
GO
CREATE TABLE Puntos( idPunto INT NOT NULL IDENTITY(1,1), ElPunto TipoPunto)
SELECT *, ElPunto.m_x, ElPunto.m_y FROM Puntos
GO
UPDATE * Puntos SET ElPunto.SetXY(20,30) WHERE ElPunto.m_x=0
DECLARE @Pnt TipoPunto
SET @Pnt=(SELECT Pnt FROM Puntos WHERE ID=2)
SELECT @Pnt.ToString() AS Pnt
GO
```

Tipo table

TABLE ({ column_definition | table_constraint } [,...n])
Un tipo especial de datos que puede utilizarse para almacenar un conjunto de resultados y procesarlo más adelante. Su uso principal es el almacenamiento temporal de un conjunto de filas, que se van a devolver como el conjunto de resultados de una función valorada en tabla.

Ejemplo.

```
USE pubs
DECLARE @TablaLibros table (Titulo varchar(50) PRIMARY KEY, Precio money, Fecha datetime)
DECLARE @vartitulo VARCHAR(50)
DECLARE @varprecio MONEY
DECLARE cursortitulos cursor FOR SELECT title, price From titles
OPEN cursortitulos
Fetch next from cursortitulos Into @vartitulo, @varprecio
While @@fetch_status=0
Begin
   INSERT @TablaLibros VALUES (@vartitulo, @varprecio, getdate())
   Fetch next from cursortitulos Into @vartitulo, @varprecio
End
```

```
Deallocate cursortitulos
Select * from @tablalibros
```

Restricciones - Check

Son los dominios de los campos de una tabla. Para asignar restricciones se despliega la tabla correspondiente y se pulsa sobre Restricciones - Nueva Restricción. En la pantalla que aparece, indicar la expresión válida para uno o varios campos. Acepta cualquier combinación de operadores lógicos OR / AND

- IN (valor1, valor2,…..)- da TRUE si el operando es igual a uno de la lista de expresiones.
- BETWEEN valor1 and valor2- TRUE si el operando está dentro de un intervalo.
- ALL - TRUE si el conjunto completo de comparaciones es TRUE
- ANY=SOME - TRUE si algún valor del conjunto de comparaciones es TRUE.
- EXISTS - TRUE si una subconsulta contiene a cualquiera de las filas.
- LIKE - da TRUE si el operando es igual a uno de la lista de expresiones.
- NOT - invierte el valor de cualquier otro operador booleano

Por ejemplo:

- ***Sueldo between 800 and 2000***: Sólo permite sueldos entre 800 y 2000
- ***Dia in ('L','M','X','J','V','S','D')***: Admite uno de los caracteres señalados

---

Funciones

La primera vez se pone CREATE, las sucesivas ALTER (menú contextual sobre la función - Modificar)

funcion1(): devuelve el valor 33

```
USE [primera]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE FUNCTION [dbo].[funcion1]()RETURNS int AS
BEGIN
   DECLARE @resultado int;
   SELECT @resultado = 33;
   RETURN @resultado;
END
```

uso: select dbo.funcion1();

funcion2(): se le pasa un numero de factura y devuelve el numero de líneas que tiene

```
USE [primera]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER FUNCTION [dbo].[funcion2](@a int) RETURNS int AS
BEGIN
   DECLARE @resultado int;
   SELECT @resultado = COUNT(*) FROM lineas_factura WHERE num_fact=@a;
   RETURN @resultado;
END
```

funcion3(): recibe 2 números, los multiplica y utiliza IF. (Ver el uso de BEGIN..END para delimitar un bloque de varias instrucciones)

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER FUNCTION funcion3(@a int,@b int) RETURNS varchar(25) AS
BEGIN
   DECLARE @respuesta varchar(25);
   DECLARE @c int;
   SELECT @c=@a*@b;
   if(@c>50) BEGIN
      SELECT @respuesta='Es mayor de 50';
      SELECT @c=@c+1;
   END
   else BEGIN
      SELECT @respuesta='Numero muy pequeño';
      SELECT @c=@c-1;
   END
   RETURN @respuesta+' '+CAST(@c AS char);
END
GO
```

funcion4(): devuelve la media de los productos que valen más de 3 euros utilizando el desplazamiento entre registros mediante un CURSOR

```
USE [primera]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER FUNCTION [dbo].[funcion4]() RETURNS double precision AS
BEGIN
   DECLARE @cod integer;
   DECLARE @nom varchar(25);
   DECLARE @pre double precision;
   DECLARE @acumulador double precision;
   DECLARE @contador int;
   select @acumulador=0;
   select @contador=0;
   DECLARE todo cursor FOR select cod_art,descripcion,precio from articulos;
   open todo;
   fetch next from todo into @cod,@nom,@pre;
   while @@fetch_status=0 begin
      if @pre>3 BEGIN
         SELECT @acumulador=@acumulador+@pre;
         SELECT @contador=@contador+1;
      END
      fetch next from todo into @cod,@nom,@pre;
   end
   return @acumulador/@contador;
END
```

Procedimientos almacenados:

Conjunto de instrucciones que se ejecutan como si fuera un programa. Pueden devolver un valor como las funciones o un resultado como una tabla

Expandir la BD - Programacion - Procedimientos almacenados - Boton derecho (Nuevo)

Como ejemplo, creamos un procedimiento que devuelve las fechas de las facturas de un cliente determinado del que pasamos su código:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE FacturasDe @n int AS
BEGIN
   SET NOCOUNT ON;
   SELECT fecha FROM facturas WHERE cod_cli=@n;
END
GO
```

Para ejecutarlo: **EXECUTE FacturasDe 1;**
que devolverá todas las fechas en las que el cliente 1 ha comprado

---

Disparadores - Desencadenadores:

Se elige la tabla sobre la que se desea realizar el disparador. Desplegar la ***icono 'Desencadenadores' - Nuevo desencadenador***
Los registros borrados (orden DELETE u orden UPDATE) se almacenan en una tabla virtual llamada DELETED
Los registros nuevos (orden INSERT u orden UPDATE) se almacenan en una tabla virtual llamada INSERTED

Con los registros de las tablas INSERTED o DELETED se puede "jugar" para realizar las acciones pertinentes del disparador.

Ejemplo: Se dispone de una base de datos llamada **almacen** con 2 tablas:

- **articulos**
  - id int. Es la PK de la tabla
  - nombre varchar. Es la descripcion del articulo
  - stock int. Se desea que el stock esté permanentemente actualizado (contenga el numero de unidades de cada articulo en el almacen). Este es un campo calculado y no debería almacenarse, pero el cliente se ha empeñado en tenerlo "a mano" en todo momento.
- **movimientos**
  - num int. Es la PK de la tabla
  - art int. Es FK respecto de la tabla articulos
  - fecha date. Fecha en la que se hace la entrada/salida de artículos en el almacen
  - cantidad int. Marca cuantos artículos entran o salen del almacen
  - tipo char. Puede tener 2 valores:
    - 'I' indica que el articulo entra en el almacen
    - 'O' indica que el articulo sale del almacen

A) Creamos un **trigger para las altas llamado movimientos_inserta**. Como se realizan de una en una, la tabla INSERTED sólo tendrá un registro

```
USE [almacen]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
```

```
GO
ALTER TRIGGER [dbo].[movimientos_inserta] ON  [dbo].[movimientos] AFTER INSERT AS
BEGIN
  DECLARE @ar int;
  DECLARE @ti char(1);
  DECLARE @ca int;
  -- No retorna el mensaje de cantidad de registros afectados
  SET NOCOUNT ON
  -- Miro qué articulo se ha insertado
  SELECT @ar=art,@ti=tipo,@ca=cantidad FROM inserted;
  -- Actualiza el stock de articulos dependiendo de la operación
  if (@ti='I') UPDATE articulos SET stock=stock+@ca WHERE id=@ar;
  else UPDATE articulos SET stock=stock-@ca WHERE id=@ar;
END
```

B) Creamos un **trigger para las bajas llamado movimientos_borra**. Las bajas se pueden hacer "de golpe" (por ejemplo DELETE FROM movimientos) y tenemos que controlar que la baja del movimiento puede ser de entrada o de salida del almacen

```
USE [almacen]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER TRIGGER [dbo].[movimientos_borra] ON [dbo].[movimientos] AFTER DELETE AS
BEGIN
    -- Variables donde depositar los campos de la tabla deleted
    DECLARE @n int;
    DECLARE @a int;
    DECLARE @c int;
    DECLARE @f date;
    DECLARE @t char(1);
    -- deleted es la tabla donde estan los registros borrados
    DECLARE borrado CURSOR LOCAL FOR select num,art,fecha,cantidad,tipo from deleted;
    SET NOCOUNT ON
    -- Recorremos los registros borrados uno a uno para actualizar el stock
    open borrado;
    fetch next from borrado into @n,@a,@f,@c,@t;
    while @@fetch_status=0 begin
        if @t='I' UPDATE articulos set stock=stock-@c WHERE id=@a;
        if @t='O' UPDATE articulos set stock=stock+@c WHERE id=@a;
        fetch next from borrado into @n,@a,@f,@c,@t;
    end;
END
```

C) Por último creamos un trigger para las modificaciones llamado movimientos_modifica. Trataremos los registros modificados en 2 partes: primero tratamos los registros borrados (los valores antiguos) y después los insertados (los nuevos valores)

```
USE [almacen]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER TRIGGER [dbo].[movimientos_modifica]  ON  [dbo].[movimientos] AFTER UPDATE AS
BEGIN
    -- Variables donde depositar los campos de las tablas INSERTED y DELETED
    DECLARE @n int;
    DECLARE @a int;
    DECLARE @f date;
    DECLARE @c int;
    DECLARE @t char(1);
    -- deleted e inserted los metemos en cursores
    DECLARE borrado CURSOR LOCAL FOR select num,art,fecha,cantidad,tipo from deleted;
    DECLARE insertado CURSOR LOCAL FOR select num,art,fecha,cantidad,tipo from inserted;
    SET NOCOUNT ON
    -- Recorremos los registros borrados uno a uno para actualizar el stock
    open borrado;
    fetch next from borrado into @n,@a,@f,@c,@t;
    while @@fetch_status=0 begin
        if @t='I' UPDATE articulos set stock=stock-@c WHERE id=@a;
        if @t='O' UPDATE articulos set stock=stock+@c WHERE id=@a;
        fetch next from borrado into @n,@a,@f,@c,@t;
    end;
    -- Ahora recorremos los registros insertados uno a uno para actualizar el stock
    open insertado;
    fetch next from insertado into @n,@a,@f,@c,@t;
    while @@fetch_status=0 begin
        if @t='I' UPDATE articulos set stock=stock+@c WHERE id=@a;
        if @t='O' UPDATE articulos set stock=stock-@c WHERE id=@a;
        fetch next from insertado into @n,@a,@f,@c,@t;
    end;
END
```

Depuración de procedimientos, funciones y triggers

Crear una función sencilla, como f1(int,int)

```
USE [usuario]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER FUNCTION [dbo].[f1](@a int,@b int) returns int as
BEGIN
DECLARE @c int;
SELECT @c=0;
SELECT @c=@a*@b;
return @c;
END
```

Crear una nueva consulta que incluya la función

```
use usuario;
select dbo.f1(4,5);
```

Depurar

Posicionarse sobre la primera instrucción de la consulta en iniciar la depuración (Depurar-Iniciar depuración, ***Alt+F5*** o pulsar sobre el icono que tiene una flecha verde). Comienza el proceso de depuración. Para pasar a la siguiente instrucción pulsar sobre ***F11*** y se verá en la pantalla los valores de las variables declaradas

---

Bases de datos distribuidas

Para acceder a tablas que no estén en la base de datos que se esté utilizando se distinguen 2 casos:

a) Acceso a tablas de otra base de datos situada en el mismo servidor

Basta con calificar la tabla con la sintaxis: base_datos.dbo.tabla
Ejemplo: deseamos acceder a los precios de los artículos de una tabla situada en la base de datos almacen desde la base de datos usuarios

```
use usuarios;
select a.precio from almacen.dbo.articulos a;
```

b) Acceso a tablas en una base de datos situada en otro servidor

b1) Vincular el servidor remoto en el servidor local

Servidor - Objetos de Servidor - Servidores vinculados - Nuevo servidor vinculado

- General
  - Servidor vinculado: 192.168.32.102\SQLEXPRESS
  - Tipo de servidor: SQL Server
- Seguridad
  - Agregar un inicio de sesión
    - local: usuario que se identifica en el servidor local
    - remoto y contraseña: parámetros del usuario en el sistema remoto

b2) Hacer consultas SQL con la tabla calificada

la sintaxis es [Servidor].baseDatos.dbo.tabla.
Ejemplo: SELECT a.precio FROM [192.168.32.115\SQLEXPRESS].almacen.dbo.articulos a;

Indices

Un índice es un mecanismo que proporciona **acceso rápido a filas de la tabla** o que permite **aplicar determinadas restricciones**. Se puede por tanto utilizar un índice para acelerar el acceso a datos de una tabla de base de datos. Los índices de tablas tienen un funcionamiento similar al de los índices de los libros, los cuales permiten acceder rápidamente a los datos.

Existen 2 tipos de índices:

- ***Clustered (agrupado):*** Sabiendo que los datos se almacenan o recuperan por grupos este índice ordena esos grupos.
  - Sólo puede existir uno por tabla.
  - Normalmente es más rápido que el resto de índices.
  - El orden físico de las filas coincide con el orden que proporciona las claves de este tipo de índices.
  - Consideraciones:
    - Los índices agrupados no son adecuados para los siguientes atributos:
      - Columnas sometidas a cambios frecuentes. Esto provoca que se mueva toda la fila, ya que Database Engine (Motor de base de datos) debe mantener los valores de los datos de la fila ordenados físicamente.
      - Claves amplias. Las claves amplias se componen de varias columnas o varias columnas de gran tamaño. Los valores clave del índice agrupado se utilizan en todos los índices no agrupados como claves de búsqueda. Los índices no agrupados definidos en la misma tabla serán bastante más grandes, ya que sus entradas contienen la clave de agrupación y las columnas de clave definidas para dicho índice no agrupado.
- ***Nonclustered index (no agrupado).*** Semejante a los índices que vienen al final de los libros que también permiten buscar información.
  - Consideraciones:
    - Las bases de datos o tablas que exigen **pocos requisitos para la actualización**, pero suelen contener un **gran volumen de datos**, se pueden beneficiar de **muchos índices**
      no agrupados para mejorar el optimizador de consultas.
    - Los índices **deben ser estrechos**, es decir, con la menor cantidad de columnas posible. Si se utiliza un gran número de índices en una tabla, el rendimiento de las instrucciones

INSERT, UPDATE y DELETE se verá afectado, ya que todos los índices deben ajustarse adecuadamente a medida que cambian los datos de la tabla.

- Debido a que las claves primarias identifican unívocamente los registros en las tablas, Están muy estrechamente relacionados con los índices, puesto que la dificultad de los índices es que sean modificados y las claves primarias no suelen ser cambiadas. Esto es porque cada vez que se modifica un campo con índice éste se debe modificar. Es por esto por lo que los **índices se crean automáticamente cuando las restricciones PRIMARY KEY y UNIQUE** se definen en las columnas de tabla.

Los índices pueden ser:

- **Único**. Un índice único garantiza que la clave de índice, valores del campo sobre el que se crea el índice, no contenga valores duplicados y, por tanto, cada fila de la tabla o vista es
en cierta forma única. Tanto los índices agrupados como los no agrupados pueden ser únicos.
- **Índice con columnas incluidas** Índice no agrupado que se extiende para incluir columnas sin clave además de las columnas de clave.
- **Texto** Tipo especial de índice funcional basado en símbolos (token) que crea y mantiene el servicio del Motor de texto completo de Microsoft para SQL Server (MSFTESQL). Proporciona
la compatibilidad adecuada para búsquedas de texto complejas en datos de cadenas de caracteres. Es decir, los índices habitualmente se crean con los primeros caracteres de una columna, en este caso de texto, esto evita tener un respuesta rápida si realizamos búsquedas con el operador LIKE donde el texto buscado está en el medio o al final del campo, en esos casos concretos debemos usar este tipo de índices.
- **XML** Representación dividida y permanente de los objetos XML binarios grandes (BLOB) de la columna de tipo de datos xml.

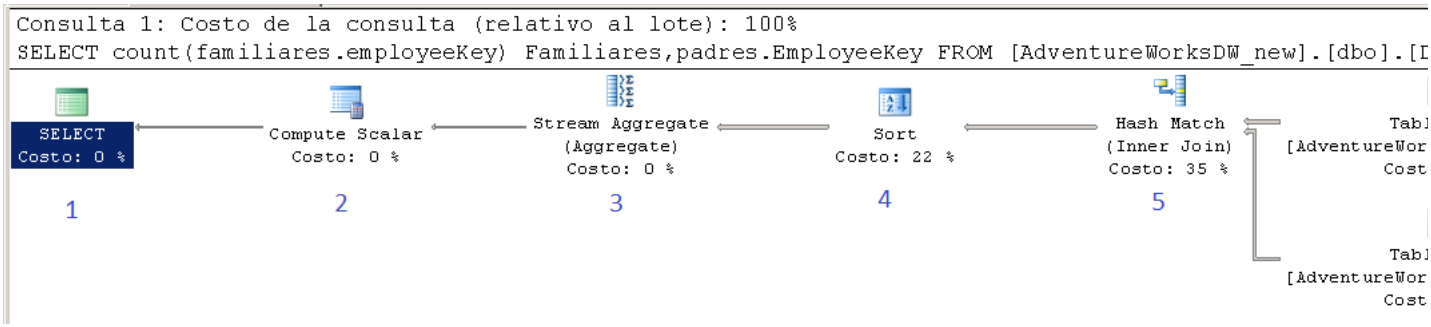
Ejemplo

para ver una comprobación de la ejecución de una query sobre una tabla con índices y otra sin ellos. La Query representa el número de familiares que tiene un empleado en la
empresa:

```
SELECT count(familiares.clave_empleado) ,padres.clave_empleado
FROM familiares, padres
WHERE familiares.clave_empleado=padres.clave_empleado
GROUP BY padres.clave_empleado
```

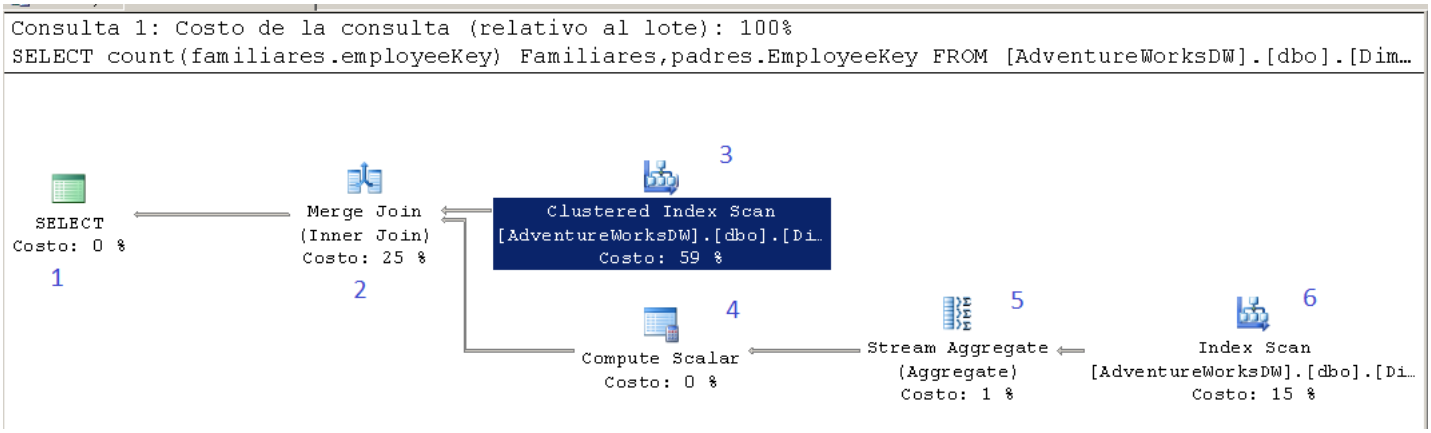
La tabla tiene tres índices IX_DimEmployee_ParentEmployeeKey(no único, no agrupado), IX_DimEmployee_SalesTerritoryKey (no único, no agrupado), y PK_DimEmployee_EmployeeKey (agrupado). Uno de ellos depende de la clave primaria. El plan de ejecución nos informará de cuales serán los tiempos estimados de ejecución.

A) Sin indices:



Nodo	1	2	3	4	5	6	7	Total
Tiempo CPU	0	0,0002005	0,0002005	0,0038758	0,0241302	0,0004826	0,0004826	0,0293722
Filas	47	47	47	295	295	296	296	1323

B) Con indices



Nodo	1	2	3	4	5	6	Total
Tiempo CPU		0,0063271	0,0004826	0,0002016	0,0002016	0,0004826	0,0076955
Filas	47	47	296	48	48	296	782

Resumiendo: el tiempo de CPU es un **74% menor** y el número de filas tratadas un **41% menor**. Por lo tanto, es evidente la mejoría que nos aporta la el uso de índices, los resultados son infinitamente mejor. Estos costes están calculados para tablas de 296 registros, con grandes tablas el tiempo de CPU ganado es increíble.

Mantenimiento de las bases de datos (tarea del DBA o administrador de la base de datos)

Backup de una BD: Elegir la BD - Boton derecho (Tareas) - Copia de Seguridad.

En la pantalla indicar el nombre del fichero donde hará el backup.

De forma análoga se realiza el Restore

En el mismo menú se pueden exportar e importar datos con un asistente (ojo a la indicación del fichero origen/destino de la información, leer bien los mensajes del asistente). Probar a exportar datos de una BD a excel. Puede servir para copiar bases de datos de un servidor a otro de una manera muy fácil (como una copia de respaldo).

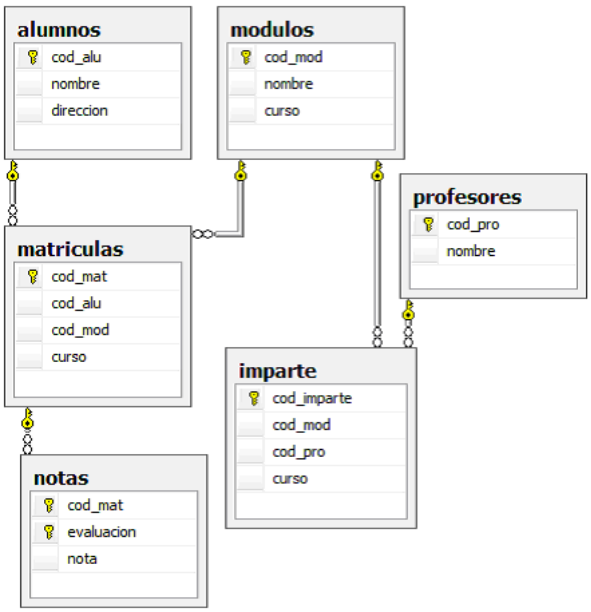
Acceso mediante comandos: SQLCMD

Salir a la linea de comandos y teclear la orden

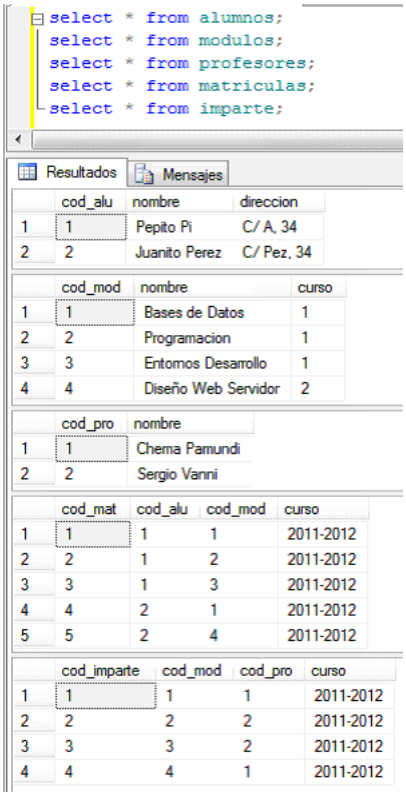
SQLCMD –U chema –S 192.168.0.16\sqlexpress
(con el usuario y el host apropiado para poder acceder). Pide la contraseña del usuario chema (chema) y se accede al intérprete de ódenes. Teclear algo como
1> USE PRIMERA
2> GO
(pone en uso la base de datos PRIMERA)
1> SELECT * FROM ARTICULOS;
2> GO
(saca el listado de todos los artículos)
1> QUIT

Práctica de procedimientos en SQLServer

Sea la base de datos instituto:



Tenemos datos de alumnos, modulos, profesores y matriculas:



```
select * from alumnos;
select * from modulos;
select * from profesores;
select * from matriculas;
select * from imparte;
```

Resultados | Mensajes

	cod_alu	nombre	direccion
1	1	Pepito Pi	C/ A, 34
2	2	Juanito Perez	C/ Pez, 34

	cod_mod	nombre	curso
1	1	Bases de Datos	1
2	2	Programacion	1
3	3	Entornos Desarrollo	1
4	4	Diseño Web Servidor	2

	cod_pro	nombre
1	1	Chema Pamundi
2	2	Sergio Vanni

	cod_mat	cod_alu	cod_mod	curso
1	1	1	1	2011-2012
2	2	1	2	2011-2012
3	3	1	3	2011-2012
4	4	2	1	2011-2012
5	5	2	4	2011-2012

	cod_imparte	cod_mod	cod_pro	curso
1	1	1	1	2011-2012
2	2	2	2	2011-2012
3	3	3	2	2011-2012
4	4	4	1	2011-2012

a) Realizar un procedimiento llamado Pon_Nota al que se le pase el nombre de un alumno, el nombre de un módulo, la evaluación y la nota y de de alta esa nota

Ejemplo: exec Pon_Nota 'Juanito Perez','Bases de Datos',1,8;

```
USE [instituto]
GO
/****** Object:  StoredProcedure [dbo].[Pon_Nota]    Script Date: 05/16/2012 10:57:01 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[Pon_Nota]
   -- Parametros del procedimiento
   @alu varchar(50),
   @mod varchar(50),
   @eva integer,
   @not integer
AS
BEGIN
   -- Variables auxiliares del procedimento:
   DECLARE @a integer;
   DECLARE @b integer;
   DECLARE @c integer;
   DECLARE @mal integer;
   -- SET NOCOUNT ON quita la linea de resultados cuando se hace un select
   SET NOCOUNT ON;
   -- Localizamos el codigo del alumno
   SELECT @a=cod_alu from alumnos where nombre=@alu;
   -- Localizamos el codigo del modulo
   SELECT @b=cod_mod from modulos where nombre=@mod;
   -- Localizamos la matricula de ese alumno en ese modulo
   SELECT @c=cod_mat from matriculas where cod_alu=@a and cod_mod=@b;
   -- Ponemos la nota correspondiente en la tabla notas (si podemos)
   begin try
      INSERT into notas (cod_mat,evaluacion,nota) values (@c,@eva,@not);
   end try
   -- Voy a comprobar que no ha habido error (no existe el alumno, el modulo o ya tenia nota)
   begin catch
      select 'Nombre de alumno/modulo incorrecto o nota ya existente';
   end catch
   -- Si todo va bien, llega hasta esta linea
   select 'Nota dada de alta correctamente';
END
```

b) Realizar la funcion Saca_Nota a la que se le pasa el nombre de un alumno, el nombre de un módulo y una evaluación y devuelve la nota correspondiente

Ejemplo: select dbo.Saca_Nota('Juanito Perez','Bases de Datos',1);

```
USE [instituto]
GO
/****** Object:  UserDefinedFunction [dbo].[Saca_Nota]    Script Date: 05/16/2012 11:33:29 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER FUNCTION [dbo].[Saca_Nota](@alu varchar(50),@mod varchar(50),@eva int) RETURNS int AS
BEGIN
   DECLARE @a int;
   DECLARE @b int;
   DECLARE @c int;
   DECLARE @resultado int;
   -- Localizo al alumno
   SELECT @a=cod_alu from alumnos where nombre=@alu;
   -- Localizo el modulo
   SELECT @b=cod_mod from modulos where nombre=@mod;
   -- Localizo la matricula
   SELECT @c=cod_mat from matriculas where cod_alu=@a and cod_mod=@b;
   -- Localizo la nota
   SELECT @resultado=nota from notas where cod_mat=@c and evaluacion=@eva;
   return @resultado;
END
```

c) Realizar un procedimiento llamado Boletín al que se le pasa el nombre de un alumno e imprime su boletín de notas

Ejemplo : exec Boletin 'Juanito Perez';

```
USE [instituto]
GO
/****** Object:  StoredProcedure [dbo].[Boletin]    Script Date: 05/16/2012 16:04:40 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[Boletin]
   @alu varchar(50)
AS
BEGIN
   SET NOCOUNT ON;
   -- Variables locales para utilizar en el Procedimiento
   DECLARE @tabloide table (modulo varchar(50),EV1 int,EV2 int,EV3 int);
   DECLARE @a int;
   DECLARE @n varchar(50);
   DECLARE @n1 int;
   DECLARE @n2 int;
   DECLARE @n3 int;
   -- Sacamos el codigo del alumno
   SELECT @a=cod_alu FROM alumnos WHERE nombre=@alu;
   -- Sacamos los modulos de los que esta matriculado y lo ponemos en un cursor
   DECLARE c CURSOR FOR SELECT modulos.nombre
                  FROM modulos,matriculas
                  WHERE modulos.cod_mod=matriculas.cod_mod
                  AND matriculas.cod_alu=@a;
   -- Ahora hacemos un recorrido por el CURSOR y buscamos las notas
   OPEN c;
   FETCH NEXT FROM c into @n;
   WHILE @@FETCH_STATUS=0 BEGIN
      -- Sacamos las notas de las 3 evaluaciones del modulo correspondiente)
      SELECT @n1=dbo.Saca_Nota(@alu,@n,1);
      SELECT @n2=dbo.Saca_Nota(@alu,@n,2);
      SELECT @n3=dbo.Saca_Nota(@alu,@n,3);
      INSERT INTO @tabloide VALUES (@n,@n1,@n2,@n3);
      -- Buscamos otro modulo del que esté matriculado
      FETCH NEXT FROM c into @n;
   END;
   -- Imprimimos la tabla temporal
   SELECT * FROM @tabloide;
END
```