

**Lecciones aprendidas 2.3**

HOJA DE CONTROL

|  |  |  |  |
| --- | --- | --- | --- |
| **Organismo** | LECCIONES APRENDIDA SQL SERVER | | |
| **Proyecto** | LECCIONES APRENDIDA SQL SERVER | | |
| **Entregable** | LECCIONES APRENDIDA SQL SERVER | | |
| **Autor** | RAÚL MARCELO ARMAS BENAVIDES | | |
| **Versión** | 2.0 | **Fecha Versión** | 21/09/2019 |
| **Aprobado por** |  | **Fecha Aprobación** | 21/09/2019 |
|  |  | **Nº Total de Páginas** | 26 |

REGISTRO DE CAMBIOS

|  |  |  |  |
| --- | --- | --- | --- |
| **Versión** | **Causa del Cambio** | **Responsable del Cambio** | **Fecha del Cambio** |
| 1.0 | L.A | RMAB | 09/09/2019 |
| 1.2 | Proyecto web la UIGV con ws | RMAB | 25/09/2019 |
| 1.3 | Se actualizó el manual, se consolidará toda la información | RMAB | 05/09/2020 |
| 1.4 | Actualización de fechas | RMAB | 21/09/2020 |
| 1.5 | Se agregó 2 ejercicios transact sql | RMAB | 28/11/2020 |
| 1.6 | Se agregaron ejercicios de transact sql | RMAB | 11/12/2020 |
| 1.7 | Collation | RMAB | 15/03/2021 |
| 1.8 | Actualización | RMAB | 30/05/2021 |
| 1.9 | Actualización | RMAB | 30/08/2021 |
| 2.1 | Se refrescó truncate, índices | RMAB | 23/09/2021 |
| 2.3 | Cursors | RMAB | 26/06/2022 |

Contenido

[1 Introducción a la base de datos 5](#_Toc107107203)

[1.1 Instalación de sql server 5](#_Toc107107204)

[1.1.1 Intercalaciones de SQL Server ( Collation) 6](#_Toc107107205)

[1.1.2 DML ( Data manipulation language) 23](#_Toc107107206)

[1.1.3 Tipo de datos 26](#_Toc107107207)

[1.2 Date 27](#_Toc107107208)

[1.3 Esquemas 27](#_Toc107107209)

[1.4 UNIQUE 29](#_Toc107107210)

[1.5 IS NULL 30](#_Toc107107211)

[1.6 Relaciones 30](#_Toc107107212)

[1.7 Stored procedures 31](#_Toc107107213)

[1.7.1 Common sp 31](#_Toc107107214)

[1.8 Indexs 31](#_Toc107107215)

[1.8.1 Índices Agrupados 31](#_Toc107107216)

[1.8.2 Creando índices Agrupados Personalizados 35](#_Toc107107217)

[1.8.3 Índices no Agrupados 38](#_Toc107107218)

[1.8.4 Creando un Índice No Agrupado 38](#_Toc107107219)

[1.8.5 Claves externas no indexadas 41](#_Toc107107220)

[1.8.6 Fragmentación de índices 42](#_Toc107107221)

[1.9 Cursor 43](#_Toc107107222)

[1.9.2 Opinion about cursors ( vs set based alternatives) 47](#_Toc107107223)

[1.9.3 The fetch type first cannot be used with forward only cursors. 47](#_Toc107107224)

[2 TRIGGERS 47](#_Toc107107225)

[3 TRANSACT SQL SERVER 49](#_Toc107107226)

[3.1 Tablas temporales ( temporary tables) 50](#_Toc107107227)

[3.1.1 Tablas temporales en memoria 50](#_Toc107107228)

[3.1.2 Tablas temporales físicas 50](#_Toc107107229)

[3.1.3 Funciones 50](#_Toc107107230)

[3.2 EXEC 52](#_Toc107107231)

[3.3 Ejercicios 52](#_Toc107107232)

[3.3.1 BD EDUTEC 52](#_Toc107107233)

[3.3.2 BD EDUCA 57](#_Toc107107234)

[4 Business Inteligence 60](#_Toc107107235)

[4.1 SQL SERVER DATA TOOLS 60](#_Toc107107236)

[ SQL Server Analysis Services 61](#_Toc107107237)

[4.1.1 SQL Server Integration Services SSIS 62](#_Toc107107238)

[4.2 Adworks 68](#_Toc107107239)

[4.2.1 Desde un ODBC Source 68](#_Toc107107240)

[4.2.2 Desde un archivo plano ( territorios.txt) 69](#_Toc107107241)

[5 TAREAS 74](#_Toc107107242)

[5.1 Tarea 1 74](#_Toc107107243)

[6 Administration ( ADMINISTRACIÓN ) 81](#_Toc107107244)

[6.1 Extended Event 81](#_Toc107107245)

[6.1.1 Master certification 81](#_Toc107107246)

[6.2 Log Shipping 81](#_Toc107107247)

[6.3 Jobs 81](#_Toc107107248)

[6.3.1 Backup 81](#_Toc107107249)

[6.4 Job envío de correo 82](#_Toc107107250)

[7 Optimización de consultas 85](#_Toc107107251)

[8 Apéndice 86](#_Toc107107252)

[8.1 Entorno SSIS 86](#_Toc107107253)

[8.1.1 Ejecutar 86](#_Toc107107254)

# Introducción a la base de datos

La gestión de base de datos implica tanto la definición de estructuras para el almacenamiento de información como la provisión de mecanismos para la gestión de la información. Además, los sistemas de base de datos deben mantener la seguridad de la información registrada, pese a caídas del sistema o intentos de infiltrados.

Los aspectos importantes de la gestión de la base de datos

* Redundancia e inconsistencia de los datos
* Dificultad para tener acceso a los datos
* Aislamiento de datos
* Anomalías del acceso concurrente
* Problemas de seguridad
* Problemas de integridad

El SQL es un lenguaje más utilizado para el acceso a base de datos relacionales.

Microsoft SQLServer es un Sistema para la Gestión de Base de Datos basado en el Modelo Relacional. Sus lenguajes para consultas son Transact-SQL y Ansi SQL. Presenta un entorno gráfico de administración de los Objetos del motor de base de datos, que permite el uso de comando DDL y DML gráficamente.

* Lenguaje de Definición de Datos (DDL), comandos para crear y definir nuevas bases de datos, tablas, campos e índices.
* Lenguaje de Manipulación de Datos (DML), comandos para insertar, modificar y eliminar registros, así como ordenar, filtrar y extraer información de la base de datos.

La E/S de una instancia de SQL Server se divide en E/S lógica y física.

Se produce una lectura lógica cada vez que el motor de la base de datos solicita una página

de la memoria caché del búfer. Si la página no está actualmente en la memoria caché del búfer,

se realiza una lectura física luego se realizó para leer la página en la memoria caché del búfer.

Si la página está actualmente en el caché, no se genera lectura física;

la memoria caché del búfer simplemente usa la página que ya está en la memoria.

## Instalación de sql server

La configuración básica no viene con Jobs.

Se tiene que habilitar sa , luego reiniciar el servicio de sql server. Separar ( detach) adjuntar base de datos ( attach)

### Intercalaciones de SQL Server ( Collation)

Criterio:

Permite diferenciar entre mayúsculas y minúsculas ( cs case sensitive- ci case insensitive)

Permite diferenciar ente una misma letra acentuada o sin acentúar ( as accent sensitive – ac accent insensitive)

Create database nuevadb collate **modern\_spanish\_ci\_as**

**Laboratorio: Aplicación de Sentencias SQL**

**Objetivos:**

* Aprender la terminología del lenguaje estructurado SQL.
* Aprender a utilizar la sentencia SELECT, para el manejo de la información de una Base de Datos.
* Aplicar las sentencias FROM y WHERE
* Aplicar las cláusulas: GROUP BY y HAVING

**Instrucciones:**

**Ejercicio 1: Ejecutar una instrucción SQL que permita extraer todos los campos y registros de la tabla Categorías**

1.- Ejecutar una instrucción SQL que permita, extraer los campos y registros de la tabla categorías, para cumplir lo solicitado, en la ventana: Instrucción SQL, escribimos la siguiente sentencia:

Select \* from categorías

**Ejercicio 2: Ejecutar una instrucción SQL que presente los campos Nombrecategoría y Descripción de la tabla Categorías**

SELECT NombreCategoría, Descripción FROM Categorías

1.- Limpie la ventana: Instrucción SQL

2.- Escriba la sentencia en la ventana Instrucción SQL indicada

3.- Haga clic en Ejecutar, siga el procedimiento antes descrito y obtendrá el siguiente resultado:

**CAMPOS CALCULADOS**

**Ejercicio 3: Sentencia SQL que presente los campos nombre producto, preciounidad y preciounidad multiplicado por 10**

Como puede apreciar, además de las columnas cuyos valores provienen directamente de la Base de Datos, una consulta SQL, puede incluir campos calculados cuyos valores se obtiene a partir del cálculo de los valores almacenados. Para solicitar un campo calculado, se especifica una expresión SQL en la lista de selección. Las expresiones SQL pueden contener sumas, restas, multiplicaciones y divisiones.

SELECT NombreProducto, PrecioUnidad, PrecioUnidad \* 10 FROM Productos

**Selección de la fila (cláusula WHERE)**

Las consultas SQL que recupera todas las filas de una tabla son útiles para la inspección y elaboración de informes sobre la Base de Datos. Generalmente se deseará seleccionar solamente parte de las filas de una tabla y solo se incluirán esas filas en los resultados. La cláusula WHERE se emplea para especificar las filas que se desean recuperar.

La cláusula WHERE consta de la palabra clave WHERE seguida de una condición de búsqueda que especifica las filas a recuperar. Por cada fila, la condición de búsqueda puede producir uno de los tres resultados:

* Si la condición de búsqueda es TRUE, la fila se incluye en los resultados de la consulta.
* Si la condición de búsqueda es FALSE, la fila se excluye de los resultados de la consulta.
* Si la condición de búsqueda tiene un valor NULL (DESCONOCIDO), la fila se excluye de los resultados de la consulta.

**Ejercicio 4: Sentencia SQL que presente los campos nombreproducto, nombrearticulo, precio unitario pertenecientes a la categoría bebidas.**

SELECT NombreProducto, IdCategoría FROM Productos WHERE IdCategoría = 1

**Ejercicio 5: Escribir una sentencia SQL que presente los campos CodigoProducto, Nombreproducto, CantidadEnExistencias, del producto cuyas UnidadesEnExistencia o Stock sea igual o menor que 10 unidades.**

SELECT IdProducto, NombreProducto, UnidadesEnExistencia FROM Productos WHERE UnidadesEnExistencia <= 10

**Ejercicio 6: Escribir una sentencia SQL que presente los campos CodigoProducto, preciounitario pertenecientes a la categoría carnes pero con un precio unitario menor o igual a 50.**

SELECT IdProducto, NombreProducto, PrecioUnidad, IdCategoría FROM Productos WHERE IdCategoría = 2 and PrecioUnidad <= 50

**FILAS DUPLICADAS (DISTINCT)**

Si una consulta incluye la clave primaria de una tabla en su lista de selección, un valor diferente para cada fila). Si no se incluye la clave primaria en los resultados, puede producirse filas duplicadas

Se puede omitir filas duplicadas en los resultados de la consulta insertando la palabra clave DISTINCT en la sentencia SELECT justo antes de la selección. Si se omite la palabra clave DISTINCT, SQL no omite las filas duplicadas. También se puede especificar la palabra clave ALL para indicar explícitamente que las filas duplicadas sean retenidas, pero es innecesario ya que éste es el comportamiento por omisión.

**CONDICION DE BUSQUEDA**

SQL ofrece un rico conjunto de condiciones de búsqueda que permite especificar muchos tipos diferentes de consultas eficaz y naturalmente. A continuación describiré cinco condiciones básicas de búsqueda (llamadas predicadas en es estándar ANSI/ISO)

* Test de comparación. Compara el valor de una expresión con el valor de la otra.
* Test de rango. Examina si el valor de una expresión cae dentro de un rango especificado de valores.
* Test de pertenencia a conjunto. Comprueba si el valor de una expresión se corresponde con uno de un conjunto de valores.
* Test de correspondencia patrón. Comprueba si el valor de una columna que contiene datos de cadena de caracteres se corresponde a un patrón especificado.
* Test de valor nullo. Comprueba si una columna tiene el valor NULL (desconocido)

**Test de comparación ( = , <> , < , <= , > , >= )**

La condición de búsqueda mas común utilizada en una consulta SQL es el test de comparación. En un test de comparación, SQL calcula y compara los valores de dos expresiones SQL por cada fila de datos

**Test de rango ( BETWEEN)**

SQL proporciona una forma diferente de condición de búsqueda con un test de rango (BETWEEN). El test de rango comprueba si un valor de dato se encuentra entre dos valores especificados. Implica el uso de tres expresiones SQL. La primera expresión define el valor a comprobar, la segunda y tercera expresión definen los extremos superior e inferior del rango a comprobar. Los tipos de datos de las tres expresiones deben ser comparables.

**Ejercicio 7: Escribir una sentencia SQL que presente los campos Código Producto, nombre del producto y precio unitario de los productos cuyos precios sean mayores que 40 y menores que 10.**

SELECT IdProducto, NombreProducto, PrecioUnidad FROM Productos WHERE PrecioUnidad BETWEEN 40 AND 10

**Test de pertenencia a conjunto (IN)**

Otra condición habitual es el test de pertenencia a conjunto (IN), examina si un valor de dato coincide con uno de una lista de valores objetivo

**Ejercicio 8: Escribir una sentencia SQL que presente los campos código de producto, nombre de producto código de la categoría pertenecientes a la categoría bebidas y carnes.**

SELECT IdProducto, NombreProducto, IdCategoría, PrecioUnidad FROM Productos WHERE IdCategoría IN (1, 2)

**Test de correspondencia con patrón (LIKE)**

Se puede utilizar un test de comparación simple para recuperar las filas en donde el contenido de una columna de texto se corresponde con un cierto texto particular

El texto de correspondencia con patrón (LIKE) comprueba si el valor de dato de una columna se ajusta a un patrón especificado. El patrón es una cadena que puede incluir uno o más caracteres comodines.

**Ejercicio 9: Escribir una sentencia SQL que presente los campos codigo de producto, nombre de producto de los registros cuyo nombre contenga la letra n en la tercera posición.**

Nota:

El comodín ? reemplaza a un carácter.

El comodín % reemplaza a cero o más caracteres.

SELECT IdProducto, NombreProducto FROM Productos WHERE NombreProducto LIKE ‘\_n%’

Pruebe y observe los cambios al modificar el contenido de la condición LIKE

SELECT IdProducto, NombreProducto FROM Productos WHERE NombreProducto LIKE **‘\_­\_n%’**

SELECT IdProducto, NombreProducto FROM Productos WHERE NombreProducto LIKE **‘%n%’**

**Test de valor (IS NULL)**

Los valores NULL crean una lógica trivaluada para las condiciones de búsqueda en SQL. Para una fila determinada, el resultado de una condición de búsqueda puede ser TRUE o FALSE o puede ser NULL debido a que una de la columnas utilizadas en la evaluación de búsqueda contenga un valor NULL.

**ORDENACION DE LOS RESULTADOS DE UNA CONSULTA (CLAUSULA ORDER BY)**

Al igual que las filas de una tabla en la Base de Datos las filas de los resultados de una consulta no están dispuestas en ningún orden particular. Se puede pedir a SQL que ordene los resultados de una consulta incluyendo la cláusula ORDER BY en la sentencia SELECT. La cláusula ORDER BY consta de las palabras ORDER BY seguidas de una lista de especificaciones de la ordenación separadas por comas.

**Ejercicio 10: Escribir una sentencia SQL que presente los campos código de producto, nombre de producto y precio unitario ordenados por el precio unitario en forma descendente (DESC).**

SELECT IdProducto, NombreProducto, PrecioUnidad FROM Productos ORDER BY PrecioUnidad DESC

**CONSULTA SUMARIAS**

SQL proporciona funciones de columna que permiten proporcionar información totalizada. Una función de columnas SQL acepta una columna entera de datos como argumentos y produce un único dato que totaliza la columna.

SQL ofrece seis funciones de columnas diferentes que se muestran a continuación:

|  |  |
| --- | --- |
| **Funciones** | **Descripción** |
| SUM() | Calcula el total de una columna |
| AVG() | Calcula el valor promedio de una columna |
| MIN() | Encuentra el valor más pequeño en una columna |
| MAX() | Encuentra el valor mayor de una columna |
| COUNT() | Cuenta el número de valores en una columna |
| COUNT(\*) | Cuenta las filas de resultado de una consulta |

**Ejercicio 11: Escribir una sentencia SQL que presente la suma de todas las unidades en existencias de la tabla Productos.**

SELECT SUM (UnidadesEnExistencia) FROM Productos

**Ejercicio 12: Escribir una sentencia SQL que presente cuantas Bebidas existen en el stock (Unidades En Existencia) de la tabla Productos.**

SELECT SUM (UnidadesEnExistencia) FROM Productos WHERE IdCategoría = 1

**Ejercicio 13: Escribir una sentencia SQL que presente el stock valorizado de la tabla Productos.**

SELECT SUM (PrecioUnidad \* UnidadesEnExistencia) FROM Productos

**Ejercicio 14: Escribir una sentencia SQL que presente el precio promedio de los monitores.**

SELECT AVG (PrecioUnidad) FROM Productos WHERE IdCategoría = 1

**Ejercicio 15: Escribir una sentencia SQL que presente el precio mínimo de las bebidas.**

SELECT MIN (PrecioUnidad) FROM Productos WHERE IdCategoría = 1

**Ejercicio 16: Escribir una sentencia SQL que presente el precio maximo de las bebidas.**

SELECT MAX (PrecioUnidad) FROM Productos WHERE IdCategoría = 1

**Ejercicio 17: Escribir una sentencia SQL que presente cuantos tipos de categoría existen.**

SELECT COUNT (IdCategoría) FROM Productos

**Ejercicio 18: Escribir una sentencia SQL que presente cuantos tipos de carnes existen.**

SELECT COUNT (IdCategoría) FROM Productos WHERE IdCategoría = 3

**CONSULTA AGRUPADAS (CLAUSULA GROUP BY)**

Una consulta que incluye la cláusula GROUP BY se denomina consulta agrupada, ya que agrupa los datos de las tablas fuente y procede una única fila sumaria por cada grupo de filas. Las columnas indicadas en la cláusula GROUP BY se denominan columnas de agrupación de la consulta, ya que ellas son las que determinan como se dividen las filas en grupo.

**Ejercicio 19: Escribir una sentencia SQL que presente cuantas unidades existen por cada categoría.**

SELECT IdCategoría, SUM (UnidadesEnExistencia) FROM Productos GROUP BY IdCategoría

**CONDICIONES DE BUSQUEDA DE GRUPOS (CLAUSULA HAVING)**

Al igual que la cláusula WHERE puede ser utilizada para seleccionar y rechazar filas individuales que participan en una consulta, la cláusula HAVING puede ser utilizada para seleccionar y rechazar grupos de filas. El formato de la cláusula HAVING es análogo al de la cláusula WHERE, consistiendo en la palabra HAVING seguida de una condición de búsqueda.

**Ejercicio 20: Escribir una sentencia SQL que presente cuantas unidades existen por cada categoría pero sean mayores a 30.**

SELECT IdCategoría, SUM (UnidadesEnExistencia) FROM Productos GROUP BY IdCategoría HAVING SUM(UnidadesEnExistencia) > 30

SELECT IdCategoría, SUM (UnidadesEnExistencia) FROM Productos GROUP BY IdCategoría HAVING SUM(UnidadesEnExistencia) > 300

SELECT IdCategoría, SUM (UnidadesEnExistencia) FROM Productos GROUP BY IdCategoría HAVING SUM(UnidadesEnExistencia) > 400

SELECT IdCategoría, SUM (UnidadesEnExistencia) FROM Productos GROUP BY IdCategoría HAVING SUM(UnidadesEnExistencia) > 530

SELECT IdCategoría, SUM (UnidadesEnExistencia) FROM Productos GROUP BY IdCategoría HAVING SUM(UnidadesEnExistencia) BETWEEN 600 AND 200

**MAS EJEMPLOS DE CONSULTA**

**Catálogo**

SELECT DISTINCTROW Categorías.NombreCategoría, Categorías.Descripción, Categorías.Imagen, Productos.IdProducto, Productos.NombreProducto, Productos.CantidadPorUnidad, Productos.PrecioUnidad

FROM Categorías INNER JOIN Productos ON Categorías.IdCategoría = Productos.IdCategoría

WHERE (((Productos.Suspendido)=No));

**Clientes y proveedores por ciudad**

SELECT Ciudad, NombreCompañía, NombreContacto, "Clientes" AS [Relación]

FROM Clientes

UNION SELECT Ciudad, NombreCompañía, NombreContacto, "Proveedores"

FROM Proveedores

ORDER BY Ciudad, NombreCompañía;

**Consulta de Pedidos**

SELECT DISTINCTROW Pedidos.IdPedido, Pedidos.IdCliente, Pedidos.IdEmpleado, Pedidos.FechaPedido, Pedidos.FechaEntrega, Pedidos.FechaEnvío, Pedidos.FormaEnvío, Pedidos.Cargo, Pedidos.Destinatario, Pedidos.DirecciónDestinatario, Pedidos.CiudadDestinatario, Pedidos.RegiónDestinatario, Pedidos.CódPostalDestinatario, Pedidos.PaísDestinatario, Clientes.NombreCompañía, Clientes.Dirección, Clientes.Ciudad, Clientes.Región, Clientes.CódPostal, Clientes.País

FROM Clientes INNER JOIN Pedidos ON Clientes.IdCliente = Pedidos.IdCliente;

**Detalle de pedidos con descuentos**

SELECT DISTINCTROW [Detalles de pedidos].IdPedido, [Detalles de pedidos].IdProducto, Productos.NombreProducto, [Detalles de pedidos].PrecioUnidad, [Detalles de pedidos].Cantidad, [Detalles de pedidos].Descuento, CCur([Detalles de pedidos].[PrecioUnidad]\*[Cantidad]\*(1-[Descuento])/100)\*100 AS PrecioConDescuento

FROM Productos INNER JOIN [Detalles de pedidos] ON Productos.IdProducto = [Detalles de pedidos].IdProducto

ORDER BY [Detalles de pedidos].IdPedido;

**Facturas**

SELECT DISTINCTROW Pedidos.Destinatario, Pedidos.DirecciónDestinatario, Pedidos.CiudadDestinatario, Pedidos.RegiónDestinatario, Pedidos.CódPostalDestinatario, Pedidos.PaísDestinatario, Pedidos.IdCliente, Clientes.NombreCompañía, Clientes.Dirección, Clientes.Ciudad, Clientes.Región, Clientes.CódPostal, Clientes.País, [Nombre] & " " & [Apellidos] AS Vendedor, Pedidos.IdPedido, Pedidos.FechaPedido, Pedidos.FechaEntrega, Pedidos.FechaEnvío, [Compañías de envíos].NombreCompañía, [Detalles de pedidos].IdProducto, Productos.NombreProducto, [Detalles de pedidos].PrecioUnidad, [Detalles de pedidos].Cantidad, [Detalles de pedidos].Descuento, CCur([Detalles de pedidos].[PrecioUnidad]\*[Cantidad]\*(1-[Descuento])/100)\*100 AS PrecioConDescuento, Pedidos.Cargo

FROM Productos INNER JOIN ((Empleados INNER JOIN ([Compañías de envíos] INNER JOIN (Clientes INNER JOIN Pedidos ON Clientes.IdCliente = Pedidos.IdCliente) ON [Compañías de envíos].IdCompañíaEnvíos = Pedidos.FormaEnvío) ON Empleados.IdEmpleado = Pedidos.IdEmpleado) INNER JOIN [Detalles de pedidos] ON Pedidos.IdPedido = [Detalles de pedidos].IdPedido) ON Productos.IdProducto = [Detalles de pedidos].IdProducto;

**Filtro facturas**

SELECT DISTINCTROW Facturas.\*

FROM Facturas

WHERE (((Facturas.IdPedido)=[Forms]![Pedidos]![IdPedido]));

**Lista alfabética de productos**

SELECT DISTINCTROW Productos.\*, Categorías.NombreCategoría

FROM Categorías INNER JOIN Productos ON Categorías.IdCategoría = Productos.IdCategoría

WHERE (((Productos.Suspendido)=No));

**Lista de productos actual**

SELECT [Lista de productos].IdProducto, [Lista de productos].NombreProducto

FROM Productos AS [Lista de productos]

WHERE ((([Lista de productos].Suspendido)=No))

ORDER BY [Lista de productos].NombreProducto;

**Los diez productos más caros**

SELECT DISTINCTROW TOP 10 Productos.NombreProducto AS DiezProductosMasCaros, Productos.PrecioUnidad

FROM Productos

ORDER BY Productos.PrecioUnidad DESC;

**Pedidos trimestrales**

SELECT DISTINCTROW Clientes.IdCliente, Clientes.NombreCompañía, Clientes.Ciudad, Clientes.País

FROM Clientes RIGHT JOIN Pedidos ON Clientes.IdCliente = Pedidos.IdCliente

WHERE (((Pedidos.FechaPedido) Between #1/1/1995# And #12/31/1995#));

**Pedidos trimestrales por producto**

TRANSFORM Sum(CCur([Detalles de pedidos].[PrecioUnidad]\*[Cantidad]\*(1-[Descuento])/100)\*100) AS CantidadProducto

SELECT Productos.NombreProducto, Pedidos.IdCliente, Year([FechaPedido]) AS AñoEnvío

FROM Productos INNER JOIN (Pedidos INNER JOIN [Detalles de pedidos] ON Pedidos.IdPedido = [Detalles de pedidos].IdPedido) ON Productos.IdProducto = [Detalles de pedidos].IdProducto

WHERE (((Pedidos.FechaPedido) Between #1/1/1995# And #12/31/1995#))

GROUP BY Productos.NombreProducto, Pedidos.IdCliente, Year([FechaPedido])

PIVOT "Tri " & DatePart("q",[FechaPedido],1,0) In ("Tri 1","Tri 2","Tri 3","Tri 4");

**Productos por categoría**

SELECT DISTINCTROW Categorías.NombreCategoría, Productos.NombreProducto, Productos.CantidadPorUnidad, Productos.UnidadesEnExistencia, Productos.Suspendido

FROM Categorías INNER JOIN Productos ON Categorías.IdCategoría = Productos.IdCategoría

WHERE (((Productos.Suspendido)<>Yes))

ORDER BY Categorías.NombreCategoría, Productos.NombreProducto;

**Productos sobre el precio medio**

SELECT DISTINCTROW Productos.NombreProducto, Productos.PrecioUnidad

FROM Productos

WHERE (((Productos.PrecioUnidad)>(SELECT AVG([PrecioUnidad]) From Productos)))

ORDER BY Productos.PrecioUnidad DESC;

**Subtotales por pedido**

SELECT DISTINCTROW [Detalles de pedidos].IdPedido, Sum(CCur([PrecioUnidad]\*[Cantidad]\*(1-[Descuento])/100)\*100) AS Subtotal

FROM [Detalles de pedidos]

GROUP BY [Detalles de pedidos].IdPedido;

**Totales de ventas por cantidad**

SELECT DISTINCTROW [Subtotales por pedido].Subtotal AS ImporteVenta, Pedidos.IdPedido, Clientes.NombreCompañía, Pedidos.FechaEnvío

FROM Clientes INNER JOIN (Pedidos INNER JOIN [Subtotales por pedido] ON Pedidos.IdPedido = [Subtotales por pedido].IdPedido) ON Clientes.IdCliente = Pedidos.IdCliente

WHERE ((([Subtotales por pedido].Subtotal)>2500) AND ((Pedidos.FechaEnvío) Between #1/1/1995# And #12/31/1995#));

**Ventas de empleado por país**

PARAMETERS [Fecha de inicio] DateTime, [Fecha de fin] DateTime;

SELECT DISTINCTROW Empleados.País, Empleados.Apellidos, Empleados.Nombre, Pedidos.FechaEnvío, Pedidos.IdPedido, [Subtotales por pedido].Subtotal AS ImporteVenta

FROM Empleados INNER JOIN (Pedidos INNER JOIN [Subtotales por pedido] ON Pedidos.IdPedido = [Subtotales por pedido].IdPedido) ON Empleados.IdEmpleado = Pedidos.IdEmpleado

WHERE (((Pedidos.FechaEnvío) Between [Fecha de inicio] And [Fecha de fin]));

**Ventas de productos por 1995**

SELECT DISTINCTROW Categorías.NombreCategoría, Productos.NombreProducto, Sum(CCur([Detalles de pedidos].[PrecioUnidad]\*[Cantidad]\*(1-[Descuento])/100)\*100) AS VentasPorProducto

FROM (Categorías INNER JOIN Productos ON Categorías.IdCategoría = Productos.IdCategoría) INNER JOIN (Pedidos INNER JOIN [Detalles de pedidos] ON Pedidos.IdPedido = [Detalles de pedidos].IdPedido) ON Productos.IdProducto = [Detalles de pedidos].IdProducto

WHERE (((Pedidos.FechaEnvío) Between #1/1/1995# And #12/31/1995#))

GROUP BY Categorías.NombreCategoría, Productos.NombreProducto;

**Ventas por año**

PARAMETERS [[Forms]![Diálogo de Ventas por año]![FechaInicio]] DateTime, [[Forms]![Diálogo de Ventas por año]![FechaFin]] DateTime;

SELECT DISTINCTROW Pedidos.FechaEnvío, Pedidos.IdPedido, [Subtotales por pedido].Subtotal, Format([FechaEnvío],"yyyy") AS Año

FROM Pedidos INNER JOIN [Subtotales por pedido] ON Pedidos.IdPedido = [Subtotales por pedido].IdPedido

WHERE (((Pedidos.FechaEnvío) Is Not Null And (Pedidos.FechaEnvío) Between [Forms]![Diálogo de Ventas por año]![FechaInicio] And [Forms]![Diálogo de Ventas por año]![FechaFin]));

**Venta por categoría**

SELECT DISTINCTROW Categorías.IdCategoría, Categorías.NombreCategoría, Productos.NombreProducto, Sum([Detalle de pedidos con descuento].PrecioConDescuento) AS VentasPorProducto

FROM Categorías INNER JOIN (Productos INNER JOIN (Pedidos INNER JOIN [Detalle de pedidos con descuento] ON Pedidos.IdPedido = [Detalle de pedidos con descuento].IdPedido) ON Productos.IdProducto = [Detalle de pedidos con descuento].IdProducto) ON Categorías.IdCategoría = Productos.IdCategoría

WHERE (((Pedidos.FechaPedido) Between #1/1/1995# And #12/31/1995#))

GROUP BY Categorías.IdCategoría, Categorías.NombreCategoría, Productos.NombreProducto

ORDER BY Productos.NombreProducto;

**Ventas por categoría para 1995**

SELECT DISTINCTROW [Ventas de productos para 1995].NombreCategoría, Sum([Ventas de productos para 1995].VentasPorProducto) AS VentasPorCategoría

FROM [Ventas de productos para 1995]

GROUP BY [Ventas de productos para 1995].NombreCategoría;

### DML ( Data manipulation language)

#### DELETE

-para ejecutar solo una linea en especifico:

DELETE FROM productos

where idem=2

-Si ejecutamos solo esto eliminamos todos los datos de la tabla:

DELETE FROM productos

\*si al borrar todo, reinsertamos los values el idem empezara un numero más de la lista anterior borrada,

es decir si habia 11 values, ahora empezara desde el 12

#### TRUNCATE

-elimina todos los values de la tabla, pero reinicia el conteo del idem

TRUNCATE TABLE productos

#### UPDATE

-actualiza filas de datos específicos

UPDATE productos SET nombre='CABLE HDMI' WHERE idem=5

RELACIONES ENTRE TABLAS

Create table Productos

(

Cod varchar(4) primary key not null,

nombre varchar(50) not null,

)

GO

Create table Facturar

(

Fecha date not null,

factura int primary key not null,

Doc\_Identidad int not null,

Nom\_Cliente varchar(50) not null,

)

GO

Create table Detalle

(

Cod varchar(4) not null,

factura int not null,

Cantidad int not null,

Precio money not null,

Importe money not null,

CONSTRAINT FK\_Detalle\_Productos //nombre de restriccion de foreign key(llave foránea)

FOREIGN KEY (Cod) REFERENCES Productos(Cod) //usamos a Cod ya q se repite para relacionar

ON UPDATE CASCADE //para q cada registro se actualice en cascada

ON DELETE CASCADE, ////para q cada registro se elimine en cascada

CONSTRAINT FK\_Detalle\_Facturar //nombre de restriccion para saber q tablas se relacionan

FOREIGN KEY (factura) REFERENCES Facturar(factura) //usamos a factura ya q se repite para relacionar

ON UPDATE CASCADE

ON DELETE CASCADE

)

\*recordar que las llaves q unimos tienen q ser del mismo tipo de variable

-para modificar la database, por ejemplo el maxsize:

ALTER DATABASE Ventas

MODIFY FILE

(name=Ventas,

MAXSIZE=100MB)

-además para modificar la foreign key, en este caso volver a relacioner con references:

Use Ventas

ALTER TABLE Detalle

CONSTRAINT FK\_Detalle\_Productos

FOREIGN KEY (Cod) REFERENCES Productos(Cod)

ON UPDATE CASCADE

ON DELETE CASCADE,

CONSTRAINT FK\_Detalle\_Facturar

FOREIGN KEY (nFactura) REFERENCES Facturar(nFactura)

ON UPDATE CASCADE

ON DELETE CASCADE

### Tipo de datos

#### To modify the data type of a column

1. In **Object Explorer**, connect to an instance of Database Engine.
2. On the Standard bar, click **New Query**.
3. Copy and paste the following example into the query window and click **Execute**.

SQLCopy

CREATE TABLE dbo.doc\_exy (column\_a INT ) ;

GO

INSERT INTO dbo.doc\_exy (column\_a) VALUES (10) ;

GO

ALTER TABLE dbo.doc\_exy ALTER COLUMN column\_a DECIMAL (5, 2) ;

GO

ALTER TABLE table\_name  
 column\_name datatype;

ALTER TABLE table\_name

DROP COLUMN column\_name;

#### To add a foreign key

Alter table tienda

add iddistrito int;

ALTER TABLE tienda

#### add FOREIGN KEY (iddistrito) REFERENCES dotdistritos(iddistrito);

#### Boolean

En SQL Server 2017 ya no existe el tipo de datos BOOLEAN. ( You could use the BIT datatype to represent boolean data. A BIT field's value is either 1,0 or null.)

## Date

SQL-Server has no TO\_DATE function. You have to use convert[[1]](#footnote-1)

-- Specify a datetime string and its exact format

SELECT TO\_DATE('2012-06-05', 'YYYY-MM-DD') FROM dual;

-- Specify a datetime string and style 102 (ANSI format), raises an error if conversion fails

SELECT CONVERT(DATETIME, '2012-06-05', 102);

-- TRY\_CONVERT available since SQL Server 2012 (returns NULL if conversion fails)

SELECT TRY\_CONVERT(DATETIME, '2012-06-05', 102);

TO\_DATE() isn't a valid function in SQL Server (T-SQL)

The alternatives depend on what version of SQL Server you are using

[CAST() or CONVERT()](https://docs.microsoft.com/en-us/sql/t-sql/functions/cast-and-convert-transact-sql) may be used in any version. Starting with SQL Server 2012 onward one can use [TRY\_CAST()](https://docs.microsoft.com/en-us/sql/t-sql/functions/try-cast-transact-sql) or [TRY\_CONVERT()](https://docs.microsoft.com/en-us/sql/t-sql/functions/try-convert-transact-sql). The advantage of using the latter pair is they do not error if the strings cannot be converted when they return null instead.

For date/time strings generally using CONVERT() or TRY\_CONVERT() is used as you can pass ["style" numbers](https://docs.microsoft.com/en-us/sql/t-sql/functions/cast-and-convert-transact-sql) where YYYY-MM-DD is "style" 102 e.g.

SELECT TRY\_CONVERT(DATE,'2017-01-21',102)

However it is possible to use CAST/TRY\_CAST like this example:

SET DATEFORMAT mdy;

SELECT TRY\_CAST('12/31/2016' AS datetime)

#### VARCHAR

##### CONCATENAR

SELECT Region\_Name + ' ' + Store\_Name FROM Geography  
WHERE Store\_Name = 'Boston';

## Esquemas

Utilizaremos usersample para todo.

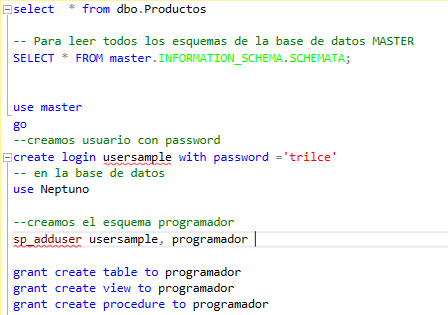
|  |  |  |
| --- | --- | --- |
| **Usuario** | **Password** |  |
| **usersample** | **trilce** |  |
|  |  |  |

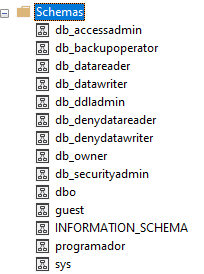
Se utiliza la base de datos Neptuno que ya viene por defecto. Se logea con el usuario sa.

Por defecto todas las tablas que se crean con el usuario sa , utilizan el esquema dbo.

En este caso creamos otros esquemas.

Cuando creamos un usuario ( con el store procedure sp\_user) , SQLSERVER asigna automáticamente un esquema con ese mismo nombre.

****

****

Luego, cuando se realiza otra conexión con el usuario **usersample** , que se ha creado solo tiene acceso a la tabla Neptuno . Además se le dan accesos para crear tablas, vistas y procedimientos almacenados con “grant”.

--creamos usuario con password

create login usersample with password ='trilce'

use sifco

select \* from material

sp\_user usersample, supervisor

grant create table to supervisor

grant create view to supervisor

grant create procedure to supervisor

Para darle permiso para crear cualquier base de datos

GRANT CREATE ANY DATABASE to usersample

c:\users\marcelo\documents\visual studio 2015\settings\sql server management studio

## UNIQUE

Si se quiere añadir una restricción única como una restricción de tabla, se debe agregar como un elemento de definición de tabla, como se muestra en la siguiente sintaxis:

[ CONSTRAINT <nombre de la restricción> ] UNIQUE ( <nombre de columna> [{, <nombre de columna>}… ] )

## IS NULL

The SQL Server [ISNULL()](https://www.w3schools.com/sql/func_sqlserver_isnull.asp) function lets you return an alternative value when an expression is NULL:

SELECT ProductName, UnitPrice \* (InStock + ISNULL(OnOrder, 0))  
FROM Products;

## Relaciones

I think you're misunderstanding what ON DELETE NO ACTION means. It does *not* mean to suppress the foreign-key constraint.

When you delete a record that is referred to by a foreign key, InnoDB has the ability to take an automatic action to rectify the situation:

* it can CASCADE, meaning, delete the referring record. (This would make sense for something like user\_ress.user\_id. If you hard-delete a user, you probably want to hard-delete all of the user's resses as well.)
* it can SET NULL, meaning, clear out the referring key. (This might make sense for something like file.last\_modified\_by. If you hard-delete a user, you might want the file's last-modified-by to become simply "unknown".)

If you specify NO ACTION, you're telling InnoDB that you don't want it to take either of these actions. So InnoDB can't fix the situation for you; all it can do is reject the DELETE and return an error.

As a result, ON DELETE NO ACTION is actually the same as ON DELETE RESTRICT (the default).

(Note: in some DBMSes, and in standard SQL, ON DELETE NO ACTION is a bit different from ON DELETE RESTRICT: in those, ON DELETE NO ACTION means "accept the DELETE within the current transaction, but reject the whole transaction if I try to commit it before rectifying the problem". But InnoDB doesn't support deferred checks, so it treats ON DELETE NO ACTION exactly the same as ON DELETE RESTRICT, and always rejects the DELETE *immediately*.)

## Stored procedures

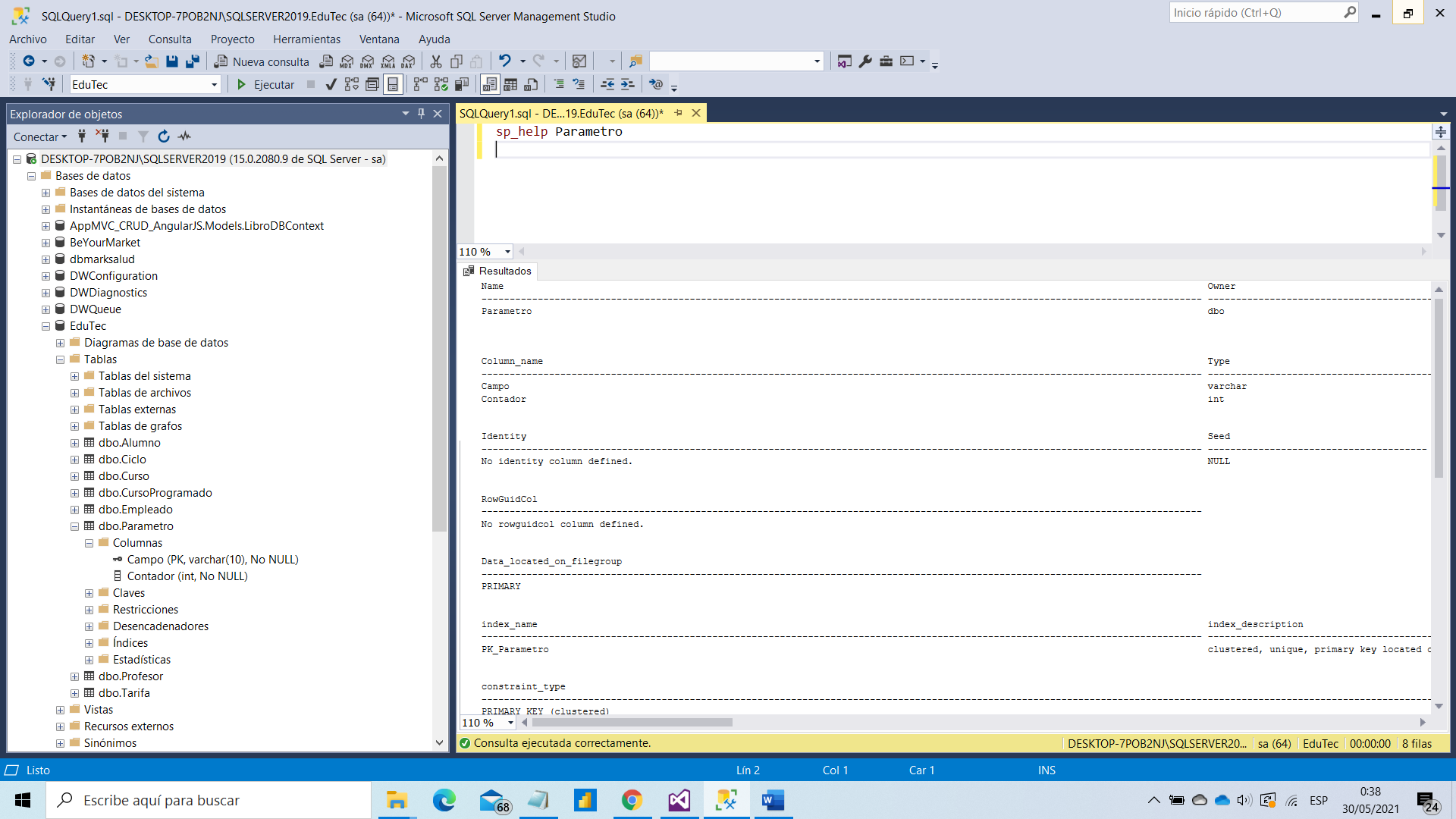
### Common sp

msdb

#### SP\_\_NOTIFICATION

Sets up a notification for an alert.

#### SP\_HELP



## Indexs

Hay dos tipos de Índices en SQL Server:

1. Índices Agrupados
2. Índices No Agrupados

### Índices Agrupados

Un índice agrupado define el orden en el cual los datos son físicamente almacenados en una tabla. Los datos de las tablas pueden ser ordenados sólo en una forma, por lo tanto, sólo puede haber un índice agrupado por tabla. En SQL Server, la restricción de llave primaria crea automáticamente un índice agrupado en esa columna en particular.

Demos un vistazo. Primero, cree una tabla “student” dentro de “schooldb” ejecutando el siguiente script:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | CREATE DATABASE schooldb    CREATE TABLE student  (      id INT PRIMARY KEY,      name VARCHAR(50) NOT NULL,      gender VARCHAR(50) NOT NULL,      DOB datetime NOT NULL,      total\_score INT NOT NULL,      city VARCHAR(50) NOT NULL  ) |

Note aquí que, en la tabla “student”, hemos establecido la restricción de llave primaria en la columna “id”. Esto crea automáticamente un índice agrupado en la columna “id”. Esto crea automáticamente un índice agrupado en la columna “id”. Para ver todos los índices en una tabla particular, ejecute el procedimiento agregado “sp\_helpindex”. Este procedimiento almacenado acepta el nombre de la tabla como parámetro y recupera todos los índices de la tabla. La siguiente consulta recupera los índices creados en la tabla “student”.

USE schooldb

EXECUTE sp\_helpindex student

La consulta de arriba retornará este resultado:

|  |  |  |
| --- | --- | --- |
| index\_name | index\_description | index\_keys |
| PK\_\_student\_\_3213E83F7F60ED59 | clustered, unique, primary key located on PRIMARY | id |

En la salida usted puede ver el único índice. Este es el índice que fue automáticamente creado debido a la restricción de llave primaria en la columna “id”.

Otra manera de ver índices de tablas es yendo a “Object Explorer->Databases->Database\_Name->Tables->Table\_Name -> Indexes”. Vea la siguiente captura de pantalla para referencia.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Este índice agrupado almacena el registro en la tabla “student” en orden ascendente de “id”. Por lo tanto, si el registro insertado tiene el id de 5, el registro será insertado en la 5ª fila de la tabla en lugar de la primera fila. De forma similar, si el cuarto registro tiene un id de 3, será insertado en la tercera fila en lugar de la cuarta fila. Esto se debe a que el índice agrupado tiene que mantener el orden físico de los registros almacenados de acuerdo a la columna indexada, por ejemplo, id. Para ver esta acción de orden, ejecute el siguiente script:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | USE schooldb    INSERT INTO student    VALUES  (6, 'Kate', 'Female', '03-JAN-1985', 500, 'Liverpool'),  (2, 'Jon', 'Male', '02-FEB-1974', 545, 'Manchester'),  (9, 'Wise', 'Male', '11-NOV-1987', 499, 'Manchester'),  (3, 'Sara', 'Female', '07-MAR-1988', 600, 'Leeds'),  (1, 'Jolly', 'Female', '12-JUN-1989', 500, 'London'),  (4, 'Laura', 'Female', '22-DEC-1981', 400, 'Liverpool'),  (7, 'Joseph', 'Male', '09-APR-1982', 643, 'London'),  (5, 'Alan', 'Male', '29-JUL-1993', 500, 'London'),  (8, 'Mice', 'Male', '16-AUG-1974', 543, 'Liverpool'),  (10, 'Elis', 'Female', '28-OCT-1990', 400, 'Leeds'); |

El script de arriba inserta diez registros en la tabla student. Note aquí que los registros son insertados en el orden al azar de los valores en la columna “id”. Pero, debido al índice agrupado por defecto en la columna id, los registros son almacenados físicamente en orden ascendiente de los valores en la columna “id”. Ejecute la siguiente sentencia SELECT para recuperar los registros desde la tabla student.

|  |  |
| --- | --- |
| 1  2  3 | USE schooldb    SELECT \* FROM student |

Los registros serán recuperados en el siguiente orden:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id | name | gender | DOB | total\_score | city |
| 1 | Jolly | Female | 1989-06-12 00:00:00.000 | 500 | London |
| 2 | Jon | Male | 1974-02-02 00:00:00.000 | 545 | Manchester |
| 3 | Sara | Female | 1988-03-07 00:00:00.000 | 600 | Leeds |
| 4 | Laura | Female | 1981-12-22 00:00:00.000 | 400 | Liverpool |
| 5 | Alan | Male | 1993-07-29 00:00:00.000 | 500 | London |
| 6 | Kate | Female | 1985-01-03 00:00:00.000 | 500 | Liverpool |
| 7 | Joseph | Male | 1982-04-09 00:00:00.000 | 643 | London |
| 8 | Mice | Male | 1974-08-16 00:00:00.000 | 543 | Liverpool |
| 9 | Wise | Male | 1987-11-11 00:00:00.000 | 499 | Manchester |
| 10 | Elis | Female | 1990-10-28 00:00:00.000 | 400 | Leeds |

### Creando índices Agrupados Personalizados

Usted puede crear su propio índice personalizado, así como el índice agrupado por defecto. Para crear un nuevo índice agrupado en una tabla, usted debe primero eliminar el índice previo

Para eliminar un índice vaya a “Object Explorer->Databases->Database\_Name->Tables->Table\_Name -> Indexes”. Haga clic derecho en el índice que desea eliminar y seleccione DELETE. Vea la siguiente captura de pantalla.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

Ahora, para crear un nuevo índice agrupado, ejecute el siguiente script:

|  |  |
| --- | --- |
| 1  2  3  4 | use schooldb    CREATE CLUSTERED INDEX IX\_tblStudent\_Gender\_Score  ON student(gender ASC, total\_score DESC) |

El proceso de crear un índice agrupado es similar a un índice normal con una excepción. Con un índice agrupado, usted tiene que usar la palabra reservada “CLUSTERED” antes de “INDEX”.

El script de arriba crea un índice llamado “IX\_tblStudent\_Gender\_Score” en la tabla student. Este índice es creado en las columnas “gender” y “total\_score”. Un índice que es creado en más que una columna es llamado “índice compuesto”.

El índice de arriba primero ordena todos los registros en orden ascendente de gender. Si gender es lo mismo para dos o más registros, los registros son ordenados en orden descendiente de los valores en su columna “total\_score”. Usted puede crear un índice agrupado en una sola columna también. Ahora, si usted selecciona todos los registros desde la tabla student, ellos serán recuperados en el siguiente orden:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id | name | gender | DOB | total\_score | city |
| 3 | Sara | Female | 1988-03-07 00:00:00.000 | 600 | Leeds |
| 1 | Jolly | Female | 1989-06-12 00:00:00.000 | 500 | London |
| 6 | Kate | Female | 1985-01-03 00:00:00.000 | 500 | Liverpool |
| 4 | Laura | Female | 1981-12-22 00:00:00.000 | 400 | Liverpool |
| 10 | Elis | Female | 1990-10-28 00:00:00.000 | 400 | Leeds |
| 7 | Joseph | Male | 1982-04-09 00:00:00.000 | 643 | London |
| 2 | Jon | Male | 1974-02-02 00:00:00.000 | 545 | Manchester |
| 8 | Mice | Male | 1974-08-16 00:00:00.000 | 543 | Liverpool |
| 5 | Alan | Male | 1993-07-29 00:00:00.000 | 500 | London |
| 9 | Wise | Male | 1987-11-11 00:00:00.000 | 499 | Manchester |

### Índices no Agrupados

Un índice no agrupado no ordena los datos físicos dentro de la tabla. De hecho, un índice no agrupado es agrupado en un solo lugar y los datos de la tabla son almacenados en otro lugar. Esto es similar a un libro de texto donde el contenido del libro está localizado en un lugar y el índice está localizado en otro. Esto permite tener más de un índice no agrupado por tabla.

Es importante mencionar que dentro de la tabla los datos serán ordenados por un índice agrupado. De todos modos, por dentro los datos del índice no agrupado son almacenados en un orden específico. El índice contiene valores de columna en los cuales el índice es creado y la dirección del registro a la que el valor de la columna pertenece.

Cuando una consulta es lanzada contra una columna en la cual el índice es creado, la base de datos primero irá al índice y buscará la dirección de la fila correspondiente en la tabla. Luego, irá a esa dirección de fila y obtendrá otros valores de columna. Es debido a este paso adicional que los índices no agrupados son más lentos que los índices agrupados.

### Creando un Índice No Agrupado

La sintaxis para crear un Índice no agrupado es similar a la del índice agrupado. De todos modos, en el caso de los índices no agrupados, la palabra reservada “NONCLUSTERED” es usada en lugar de “CLUSTERED”. Revise el siguiente script.

|  |  |
| --- | --- |
| 1  2  3  4 | use schooldb    CREATE NONCLUSTERED INDEX IX\_tblStudent\_Name  ON student(name ASC) |

El siguiente script crea un índice no agrupado en la columna “name” de la tabla student. El índice ordena por name en orden ascendente. Como dijimos anteriormente, los datos de la tabla e índice serán almacenados en lugares diferentes. Los registros de la tabla serán ordenados por un índice agrupado si hay uno. El índice será ordenado de acuerdo a su definición y será almacenado separadamente desde la tabla.

**Datos de la Tabla Student:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id | name | gender | DOB | total\_score | City |
| 1 | Jolly | Female | 1989-06-12 00:00:00.000 | 500 | London |
| 2 | Jon | Male | 1974-02-02 00:00:00.000 | 545 | Manchester |
| 3 | Sara | Female | 1988-03-07 00:00:00.000 | 600 | Leeds |
| 4 | Laura | Female | 1981-12-22 00:00:00.000 | 400 | Liverpool |
| 5 | Alan | Male | 1993-07-29 00:00:00.000 | 500 | London |
| 6 | Kate | Female | 1985-01-03 00:00:00.000 | 500 | Liverpool |
| 7 | Joseph | Male | 1982-04-09 00:00:00.000 | 643 | London |
| 8 | Mice | Male | 1974-08-16 00:00:00.000 | 543 | Liverpool |
| 9 | Wise | Male | 1987-11-11 00:00:00.000 | 499 | Manchester |
| 10 | Elis | Female | 1990-10-28 00:00:00.000 | 400 | Leeds |

**Datos del ÍndiceIX\_tblStudent\_Name**

|  |  |
| --- | --- |
| **name** | **Row ress** |
| Alan | Row ress |
| Elis | Row ress |
| Jolly | Row ress |
| Jon | Row ress |
| Joseph | Row ress |
| Kate | Row ress |
| Laura | Row ress |
| Mice | Row ress |
| Sara | Row ress |
| Wise | Row ress |

Note, aquí en el índice cada fila tiene una columna que almacena la dirección de la fila a la cual el nombre pertenece. Así que, si una consulta es lanzada para recuperar gender y DOB de un estudiante llamado “Jon”, la base de datos primero buscará el nombre “Jon” dentro del índice. Luego leerá la dirección de la fila de “Jon” e irá directamente a esa fila en la tabla “student” para recuperar gender y DOB de Jon.

#### Conclusión

De la discusión encontramos las siguientes diferencias entre índices agrupados y no agrupados.

1. Puede haber sólo un índice agrupado por tabla. De todos modos, usted puede crear múltiples índices no agrupados en una sola tabla.
2. Los índices agrupados sólo ordenan tablas. Por lo tanto, no consumen almacenaje extra. Los índices no agrupados son almacenados en un lugar separado de la tabla real. Reclamando más espacio de almacenamiento.
3. Los índices agrupados son más rápidos que los índices no agrupados, ya que no involucran ningún paso extra de búsqueda.

### Claves externas no indexadas

Cuando se crea una FOREIGN KEY, **SQL Server no crea automáticamente un índice para esta clave foránea**. Una restricción FOREIGN KEY es candidata para un índice porque:

* Los cambios en las restricciones PRIMARY KEY se verifican con restricciones FOREIGN KEY en tablas relacionadas.
* Las columnas de clave externa a menudo se usan en criterios de combinación cuando los datos de tablas relacionadas se combinan en consultas haciendo **coincidir las columnas en la restricción de CLAVE EXTRANJERA de una tabla con las columnas de clave principal o única en la otra tabla.** Un índice permite a Microsoft® SQL Server ™ 2000 encontrar rápidamente datos relacionados en la tabla de claves foráneas. Sin embargo, crear este índice no es un requisito. Los datos de dos tablas relacionadas se pueden combinar incluso si no se definen restricciones de PRIMARY KEY o FOREIGN KEY entre las tablas, **pero una relación de clave externa entre dos tablas indica que las dos tablas se han optimizado para combinarse en una consulta que utiliza las claves como sus criterios.**

/\*Identificar claves Externas no Indexadas\*/

WITH v\_NonIndexedFKColumns AS (

SELECT

Object\_Name(a.parent\_object\_id) AS Table\_Name

,b.NAME AS Column\_Name

FROM

sys.foreign\_key\_columns a

,sys.all\_columns b

,sys.objects c

WHERE

a.parent\_column\_id = b.column\_id

AND a.parent\_object\_id = b.object\_id

AND b.object\_id = c.object\_id

AND c.is\_ms\_shipped = 0

EXCEPT

SELECT

Object\_name(a.Object\_id)

,b.NAME

FROM

sys.index\_columns a

,sys.all\_columns b

,sys.objects c

WHERE

a.object\_id = b.object\_id

AND a.key\_ordinal = 1

AND a.column\_id = b.column\_id

AND a.object\_id = c.object\_id

AND c.is\_ms\_shipped = 0

)

SELECT

'CREATE NONCLUSTERED INDEX [IX\_'+v.Table\_Name + '\_' + v.Column\_Name + '] ON [dbo].['+v.Table\_Name+'] ('+ v.Column\_Name + ') WITH ( FILLFACTOR=80)' AS Indice,

'DROP INDEX [IX\_'+v.Table\_Name + '\_' + v.Column\_Name + '] ON [dbo].['+v.Table\_Name+']' AS borrarIndice,

v.Table\_Name AS NonIndexedCol\_Table\_Name

,v.Column\_Name AS NonIndexedCol\_Column\_Name

,fk.NAME AS Constraint\_Name

,SCHEMA\_NAME(fk.schema\_id) AS Ref\_Schema\_Name

,object\_name(fkc.referenced\_object\_id) AS Ref\_Table\_Name

,c2.NAME AS Ref\_Column\_Name

FROM

v\_NonIndexedFKColumns v

,sys.all\_columns c

,sys.all\_columns c2

,sys.foreign\_key\_columns fkc

,sys.foreign\_keys fk

WHERE

v.Table\_Name = Object\_Name(fkc.parent\_object\_id)

AND v.Column\_Name = c.NAME

AND fkc.parent\_column\_id = c.column\_id

AND fkc.parent\_object\_id = c.object\_id

AND fkc.referenced\_column\_id = c2.column\_id

AND fkc.referenced\_object\_id = c2.object\_id

AND fk.object\_id = fkc.constraint\_object\_id

ORDER BY 1,2

--CREATE NONCLUSTERED INDEX [IX\_tblEmpleado\_IdDepartamento] ON tblEmpleado(IdDepartamento) WITH(FILLFACTOR=80)

--CREATE NONCLUSTERED INDEX [IX\_tblEmpleado\_IdDepartamento] ON [dbo].[tblEmpleado] (IdDepartamento)

--DROP INDEX [IX\_tblEmpleado\_IdDepartamento] ON tblEmpleado

### Fragmentación de índices

Si está fragmentado menos de un 10% no es necesario hacer nada.

Si está fragmentado entre un 10% y un 30% es mejor reoganizar el índice

Si está fragmentado más de un 30% es mejor reconstruir el índice\*/

select CURRENT\_TIMESTAMP as Fecha,

DB\_NAME(db\_id()) as DatabaseName,

@@servername Servidor,

b.name as IndexName,

obj.name as ObjectName,

a.avg\_fragmentation\_in\_percent as '%Frag',

a.page\_count as NumeroPaginas ,

a.fragment\_count as PromPagFrag,

a.index\_type\_desc as TipoIndice,

a.avg\_fragment\_size\_in\_pages,

a.partition\_number

FROM sys.dm\_db\_index\_physical\_stats(db\_id(), NULL, NULL, NULL, 'limited') as a

INNER JOIN sys.indexes as b

ON a.object\_id = b.object\_id AND a.index\_id = b.index\_id

INNER JOIN sys.objects as Obj

ON a.object\_id = Obj.object\_id

--ALTER INDEX PK\_Grupos\_COD\_GRUP ON TBLGRUPOS REBUILD

--ALTER INDEX PK\_Clientes\_COD\_ID ON tblClientes REORGANIZE

## Cursor

En SQL Server un cursor puede definirse como un elemento que representará a un conjunto de datos determinado por una consulta T-SQL, el cursor permitirá recorrer fila a fila, leer y eventualmente modificar dicho conjunto de resultados.

SQL Server ofrecerá una amplia variedad de opciones y de funciones para crear y operar sobre cursores.

La creación y utilización de un cursor estará compuesta, como es de esperarse, por una serie de instrucciones T-SQL, las cuales podrán separarse en grupos bien diferenciados, los cuales son: Declaración, Apertura, Acceso a datos, Cierre y Desalojo, a continuación, detallaremos cada grupo de instrucciones.

FORWARD\_ONLY Specifies that the cursor can only be scrolled from the first to the last row. FETCH NEXT is the only supported fetch option.

DECLARE cursor\_name CURSOR [ LOCAL | GLOBAL ]

[ FORWARD\_ONLY | SCROLL ]

#### The 5-Step Process of Using a Cursor

The process of using a SQL cursor can be generally described as follows:

Declare Cursor

Open Cursor

Fetch rows

Close Cursor

Deallocate Cursor

DECLARE ProdInfo CURSOR FOR SELECT NameFROM Production.Product

DECLARE ProdInfo CURSOR READ\_ONLY FOR SELECT Name FROM Production.Product

Un detalle a comentar es que en la sentencia INTO (como puede verse en el ejemplo anterior) el mapeo entre columnas del cursor y variables se realizará implícitamente, asignándose la primera columna a la primera variable, la segunda columna a la segunda variable y así sucesivamente. Esto implica que deberán crearse tantas variables como columnas se definan en la declaración del cursor y las mismas deberán ubicarse en el mismo orden que se encuentran definidas las columnas en la sentencia SELECT de la declaración.

Como cada sentencia FETCH leerá un registro, una pregunta interesante que podríamos hacernos es, ¿de qué manera podremos saber si existe un próximo o previo registro, o si hemos llegado al límite (ya sea superior o inferior)?. La respuesta se encontrará en una variable de SQL Server llamada @@FETCH\_STATUS que tomará el valor 0 si la lectura del registro ha sido correcta.

**Un cursor no es un objeto persistente que se almacena en tablas del SQL** como por ejemplo las tablas, los **stored procedures** o los índices entre otros. Los cursores son objetos que se crean en memoria (por eso es importante utilizar la sentencia deallocate para desalojarlo de la memoria que utiliza SQL Server).

use bdgeneric

select \* From alumnos;

--creating a cursor

declare simpleCursorAlumnos cursor

local SCROLL

for select DNI from alumnos;

declare @DNI nchar(8);

declare @DNI5 nchar(8);

--create the cursor set

open simpleCursorAlumnos;

--fetch simpleCursorAlumnos

--retrieve a row (the first)

--fetch last from simpleCursorAlumnos into @DNI;

--displaying the result

print rtrim(@DNI);

fetch absolute 5 from simpleCursorAlumnos into @DNI5;

--displaying the result

print rtrim(@DNI5);

--release the cursor set

close simpleCursorAlumnos

deallocate simpleCursorAlumnos

### Opinion about cursors ( vs set based alternatives)

The "overhead" with cursors is merely part of the API. Cursors are how parts of the RDBMS work under the hood. Often CREATE TABLE and INSERT have SELECT statements, and the implementation is the obvious internal cursor implementation.

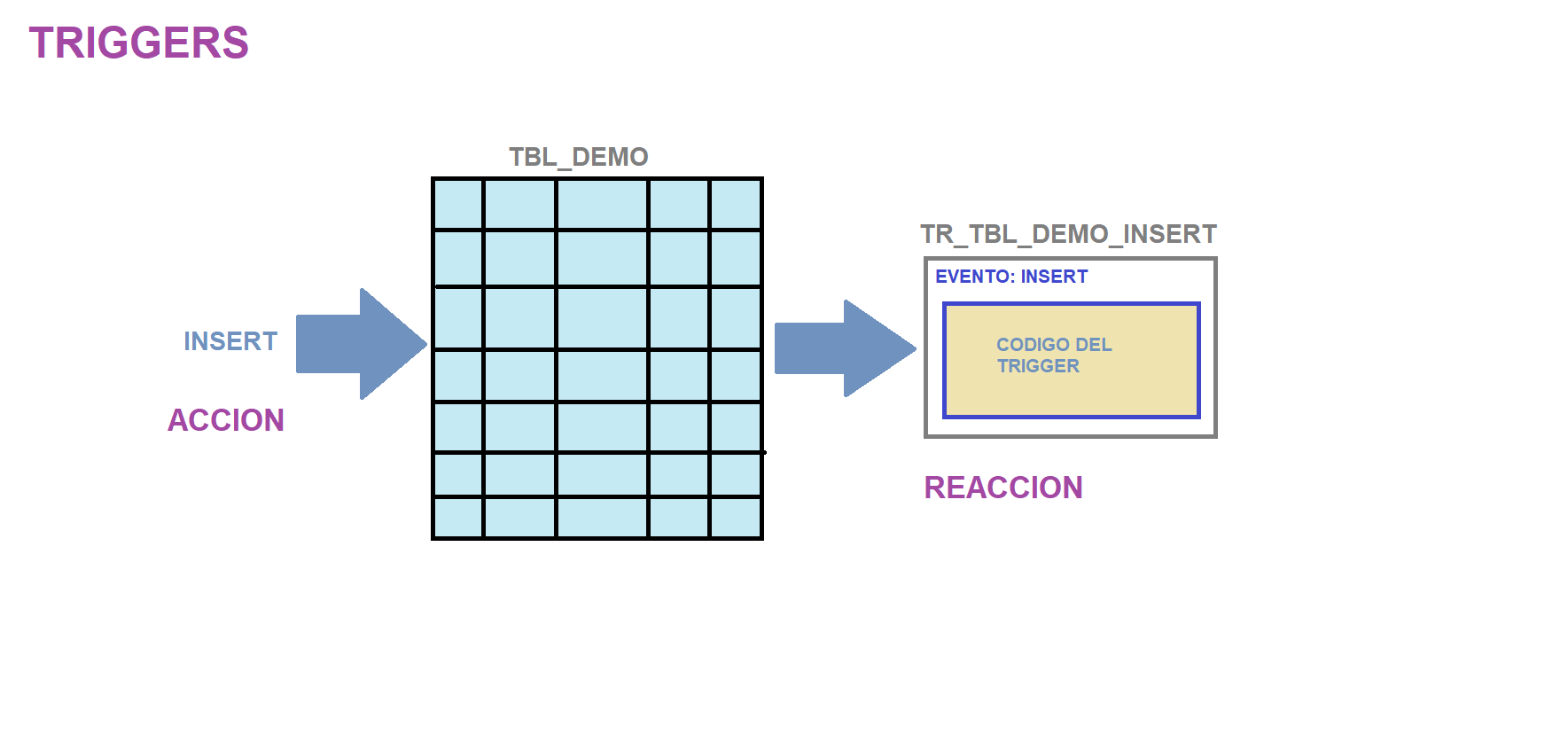
Using higher-level "set-based operators" bundles the cursor results into a single result set, meaning less API back-and-forth.

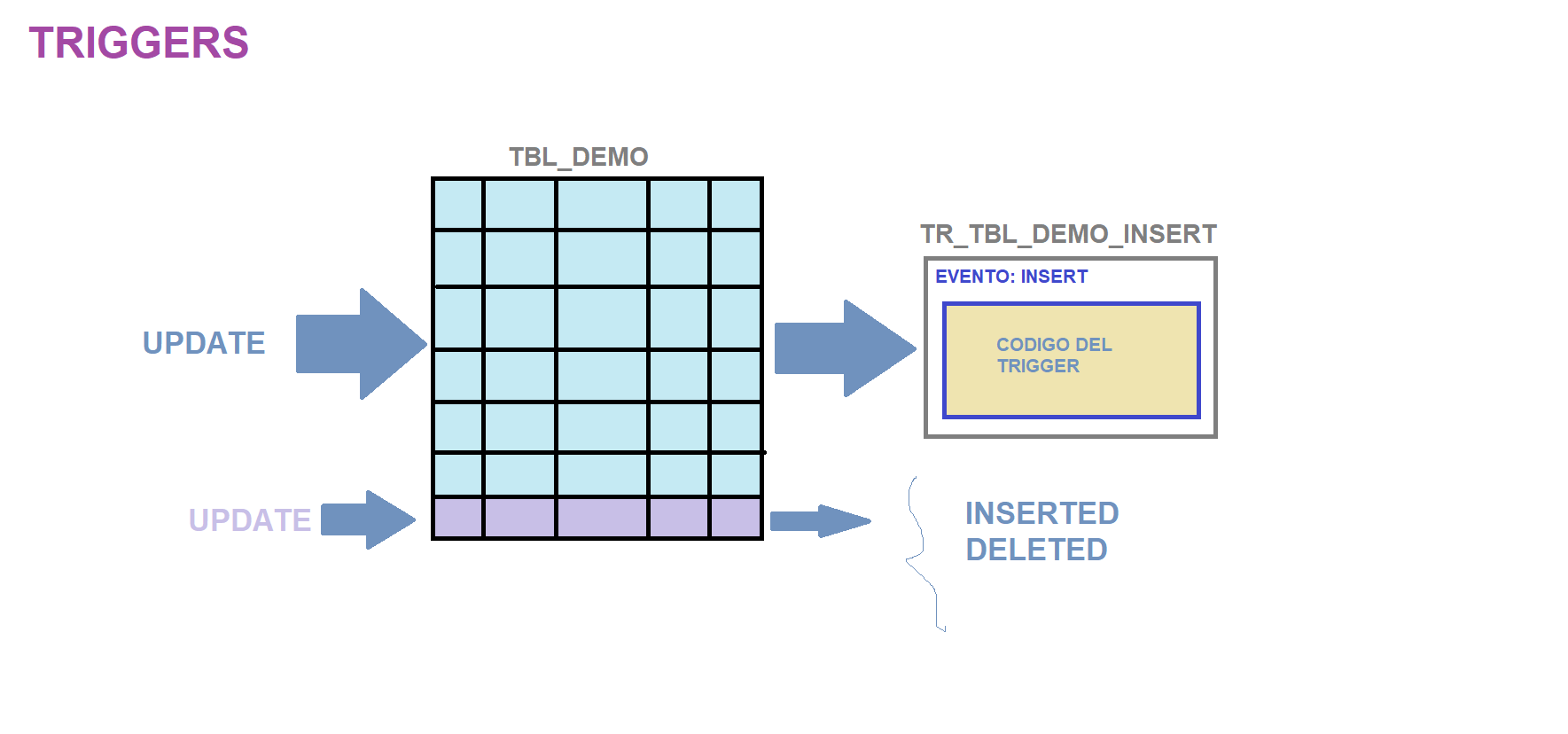
Cursors predate modern languages that provide first-class collections. Old C, COBOL, Fortran, etc., had to process rows one at a time because there was no notion of "collection" that could be used widely. Java, C#, Python, etc., have first-class list structures to contain result sets.

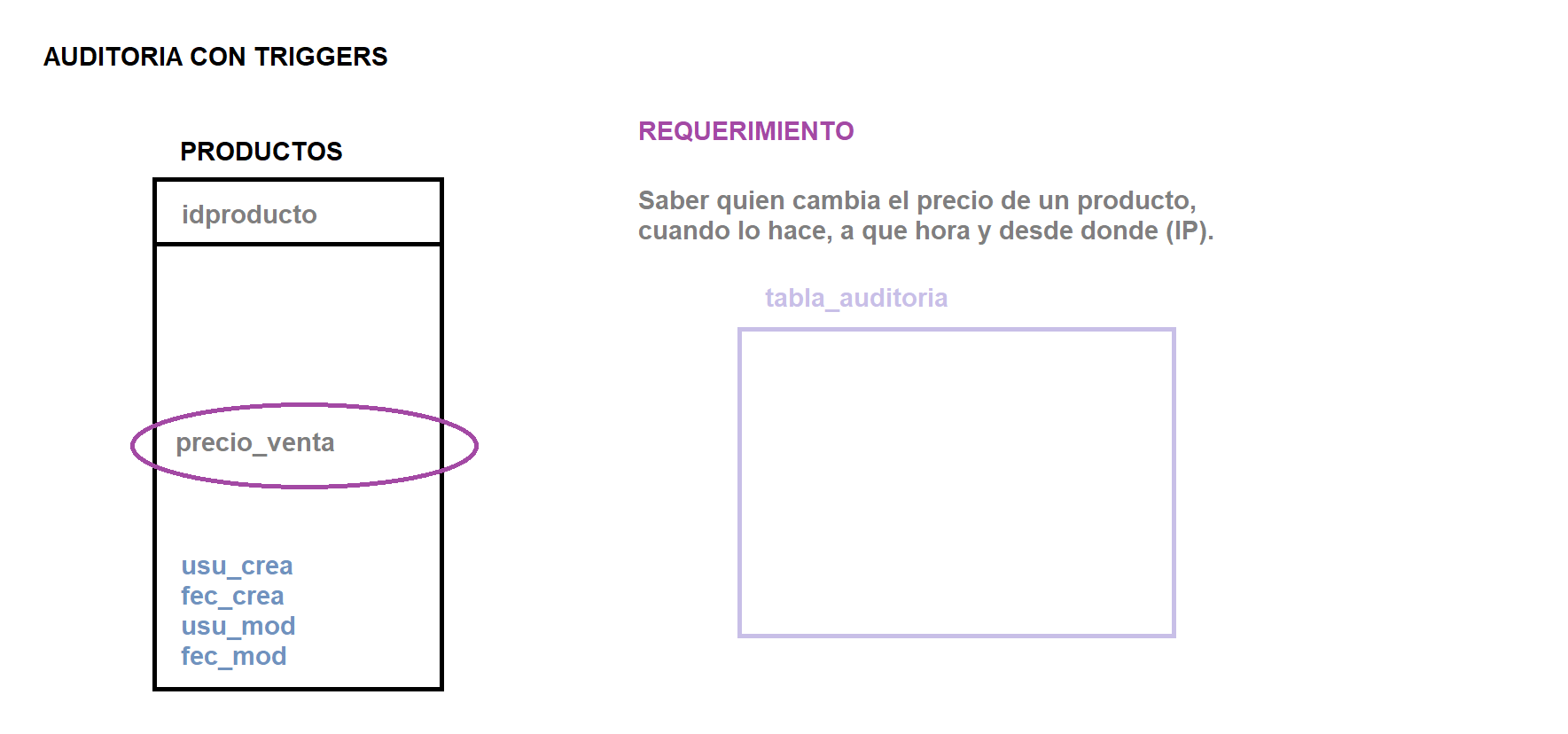
### The fetch type first cannot be used with forward only cursors.

# TRIGGERS

Su propósito es exigir la integridad referencial. Este es su principal propósito en una base de datos y son útiles en actualizaciones y eliminaciones en cascada. Si se viola una restricción el desencadenador nunca se ejecutará.







# TRANSACT SQL SERVER

Concepto de transacción: Es una secuencia de operaciones realizadas como una sola unidad de trabajo.

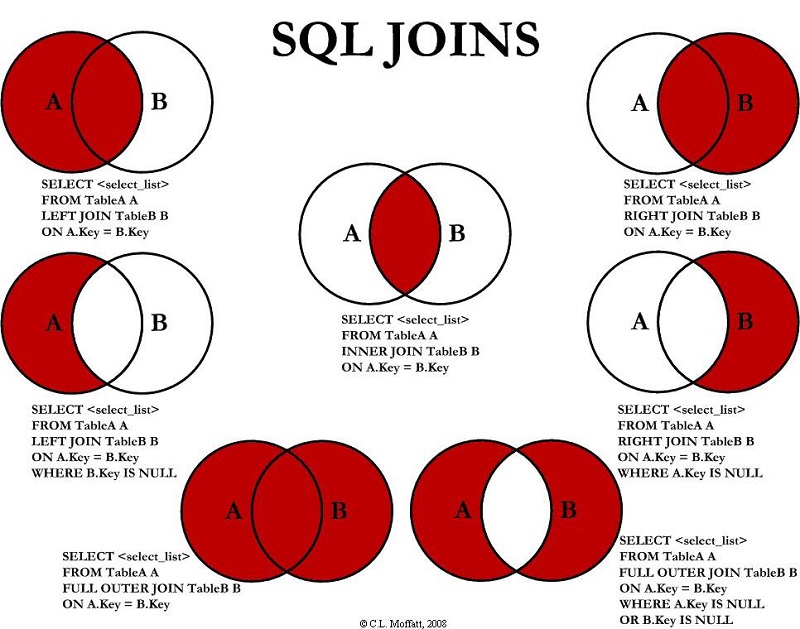
Las propiedades de las transacciones se las conoce como ACID: Atomicidad, Coherencia, Aislamiento, Durabilidad.

**Atomicidad**: Una transacción debe ser una unidad atómica de trabajo, o se hace todo o no se hace nada.

**Coherencia**: Debe dejar los datos en un estado coherente luego de realizada la transacción.

**Aislamiento**: Las modificaciones realizadas por transacciones son tratadas en forma independiente, como si fueran un solo y único usuario de la base de datos.

**Durabilidad**: Una vez concluida la transacción sus efectos son permanentes y no hay forma de deshacerlos.



## Tablas temporales ( temporary tables)

Es frecuente preguntarse si la creación / modificación de tablas temporales generaba E / S de disco físico. Un consultor había intentado justificar su elección de usar tablas temporales, en lugar de variables de tabla, declarando que las tablas temporales son estructuras puramente en memoria, que no incurren en E / S de disco físico y, por lo tanto, funcionan tan bien como las variables de tabla. Bueno, el consultor estaba equivocado, pero tenía razón al mismo tiempo.

Las tablas temporales tienen un alcance más flexible que las variables de tabla, pero no están menos definidas.

### Tablas temporales en memoria

Solo existen durante la query

### Tablas temporales físicas

### Funciones

Puede utilizar una función definida por el usuario o una vista en lugar de un procedimiento.

Un procedimiento puede devolver varios conjuntos de resultados, cada uno con su propio esquema. No es adecuado para usar en una instrucción SELECT.

--Desarrollar una función que consulte la estación del año según fecha

create function **dbo.FuncionEstaciones**(@fecha date)

returns varchar(20)

as

begin

declare @fechastring varchar(10);

declare @mes varchar(20);

declare @estacion varchar(20);

set @fechastring = convert(varchar, @fecha, 1);

set @mes = SUBSTRING(@fechastring, 1, 2);

--aqui se debe definir qué mes

set @estacion = case

WHEN @mes ='01' THEN 'Verano'

WHEN @mes ='02' THEN 'Verano'

WHEN @mes ='03' THEN 'Verano'

WHEN @mes ='04' THEN 'Invierno'

WHEN @mes ='05' THEN 'Otoño'

WHEN @mes ='06' THEN 'Otoño'

WHEN @mes ='07' THEN 'Otoño'

WHEN @mes ='08' THEN 'Primavera'

WHEN @mes ='09' THEN 'Primavera'

WHEN @mes ='10' THEN 'Primavera'

WHEN @mes ='11'THEN 'Primavera'

WHEN @mes ='12' THEN 'Primavera'

ELSE 'Otro'

end;

return @estacion;

end;

go

--consutlamos que estacion es hoy

select dbo.FuncionEstaciones(getdate())

go

--Desarrollar una función que calcule el promedio de 4 notas, pero que solo considere las 3 mejores

--son 4 notas

create function **dbo.PromedioTresMayores**(@n1 decimal, @n2 decimal , @n3 decimal ,@n4 decimal)

returns decimal

as

begin

DECLARE @promedio as decimal ;

DECLARE @TotalNotas AS TABLE

(NotaID INT NOT NULL PRIMARY KEY,

Nota decimal NOT NULL);

INSERT INTO @TotalNotas values(1,@n1);

INSERT INTO @TotalNotas values(2,@n2);

INSERT INTO @TotalNotas values(3,@n3);

INSERT INTO @TotalNotas values(4,@n4);

--set setea el promedio como el promedio (avg) de las 3 primeras notas

SELECT @promedio = AVG(t2.Nota) FROM (select top 3 t1.Nota from @TotalNotas t1 order by t1.Nota desc) t2;

return @promedio;

end;

go

--consultamos el promedio

select dbo.PromedioTresMayores(05,11,05,08)

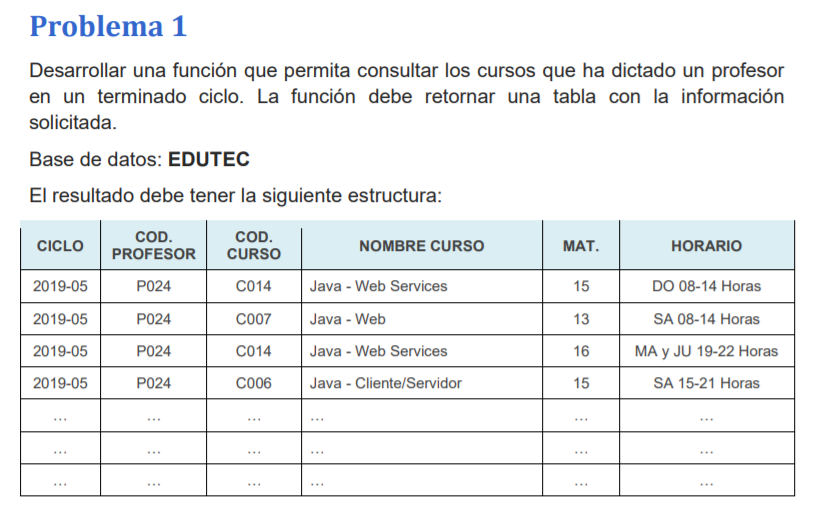
go

## EXEC

SELECT is not possible with EXEC, You will have to get the EXEC output to a temp table or table variable and fetch the output from there

## Ejercicios

### BD EDUTEC



--retorna una tabla

create OR ALTER function dbo.CursosDictadosDocente(@codigo char(4) ,@ciclo char(7))

returns @Dictados TABLE

( --columnas

Ciclo char(7)NOT NULL,

CodProfesor char(4) NOT NULL ,

CodCurso char(4) NOT NULL ,

NombreCurso varchar(50),

Matriculados numeric(3,0),

Horario varchar(30)

)

as

begin

insert into @Dictados

select cp.IdCiclo as Ciclo,cp.IdProfesor as CodProfesor,c.IdCurso as CodCurso,c.NomCurso as NombreCurso,cp.Matriculados,cp.Horario

from CursoProgramado cp, Curso c

where cp.idCurso = c.IdCurso

and cp.IdProfesor = @codigo

and cp.IdCiclo = @ciclo

return

end;

go

--consultamos los cursos dictados por el docente

select \* FROM dbo.CursosDictadosDocente('P002','2005-01')

go



--Desarrollar un procedimiento que permita registrar un nuevo curso

CREATE PROCEDURE [dbo].[usp\_registra\_Curso]

@IdCurso CHAR(4),

@IdTarifa CHAR(1),

@NomCurso VARCHAR(50)

AS

INSERT INTO Curso(idCurso,IdTarifa,NomCurso) VALUES

(@IdCurso,@IdTarifa,@NomCurso)

return 0

IF EXISTS ( SELECT name FROM sysobjects

WHERE type = 'U' AND name = 'aud\_curso\_modif' )

DROP TABLE aud\_curso\_modif

--se crea tabla de auditoría de la tabla CURSO

create table dbo.aud\_curso\_modif(

id int identity(1,1) primary key,

cur\_id char(4),--int,

observaciones varchar(100),

valor\_antes nvarchar(100) NULL,

valor\_despues nvarchar(100) NULL,

usuario\_bd varchar(100),

fecha date

);

go

--DISPARADOR AL MOMENTO DE INSERTAR UN NUEVO CURSO

drop trigger dbo.tr\_curso\_insert;

go

create trigger dbo.tr\_curso\_insert

on dbo.curso

for insert

as

begin

insert into dbo.aud\_curso\_modif(cur\_id, observaciones, valor\_antes,valor\_despues,

usuario\_bd,fecha)

SELECT I.IdCurso,'SE REGISTRÓ EL NUEVO CURSO EN CEPSUNI :'+ I.NomCurso,NULL, NULL, CURRENT\_USER, GETDATE()

FROM INSERTED I;

end;

go

drop trigger dbo.tr\_curso\_modifNombre;

go

create OR ALTER trigger dbo.tr\_curso\_modifNombre

on dbo.curso

for update

as

begin

if update(NomCurso)

begin

insert into dbo.aud\_curso\_modif(cur\_id, observaciones, valor\_antes,valor\_despues, usuario\_bd,fecha)

SELECT I.IdCurso,'SE MODIFICÓ EL NOMBRE DEL CURSO:'+ I.NomCurso,D.NomCurso, I.NomCurso, CURRENT\_USER, GETDATE()

FROM INSERTED I JOIN DELETED D ON I.IdCurso = D.IdCurso;

end;

end;

go

exec usp\_registra\_Curso 'C022','C','Matemática Discreta';

exec usp\_registra\_Curso 'C023','B','Lenguaje C';

exec usp\_registra\_Curso 'C024','C','Calidad';

exec usp\_registra\_Curso 'C025','C','Cálculo 2';

exec usp\_registra\_Curso 'C026','C','Derecho Penal';

exec usp\_registra\_Curso 'C027','C','INVENTOR';

--SE PROBARÁ MODIFICANDO LA TARIFA A UN CURSO

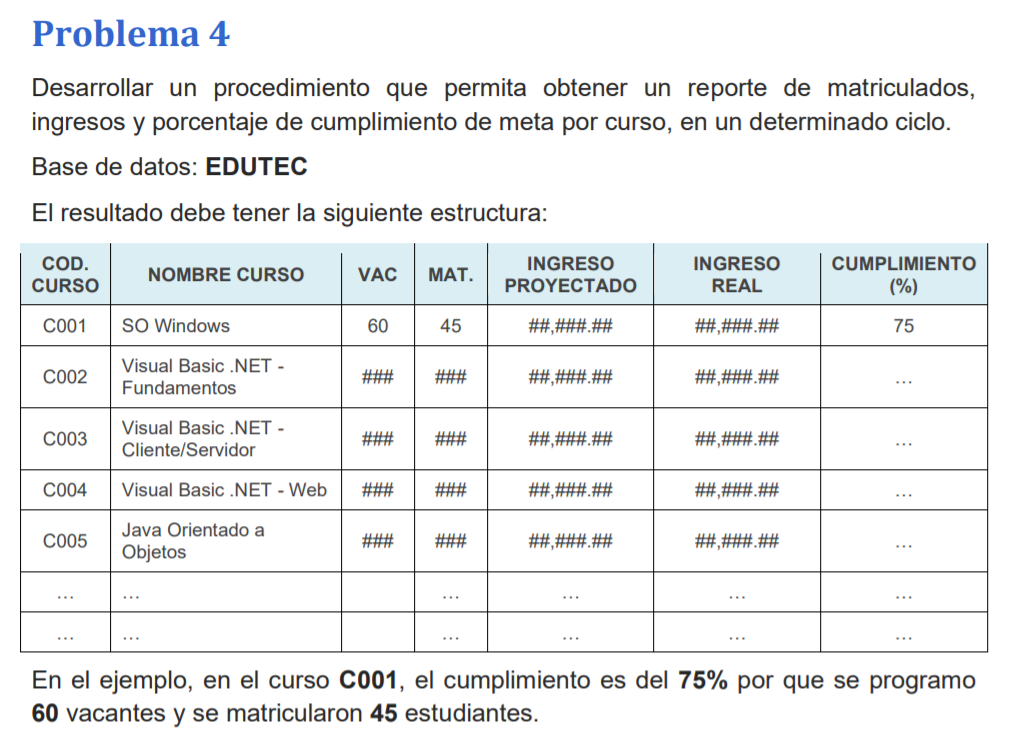
update Curso set NomCurso = 'AUTOCAD 2014' where IdCurso ='C022';

update Curso set NomCurso = 'Materiales' where IdCurso ='C024';

SELECT \* FROM Curso

--tabla de auditoría

select \* From aud\_curso\_modif



--Desarrollar un procedimiento que permita obtener un reporte de matriculados,

--ingresos y porcentaje de cumplimiento de meta por curso, en un determinado ciclo.

CREATE OR ALTER PROCEDURE [dbo].[usp\_reporte\_CumplimientoCurso]

@IdCiclo CHAR(7)

AS

BEGIN

select c.IdCurso,c.NomCurso,(cp.Vacantes +cp.Matriculados) Vacantes,cp.Matriculados,

--((cp.Vacantes +cp.Matriculados)\* t.Horas\*t.PagoHora) Ingreso\_proyectado,

(cp.Vacantes +cp.Matriculados)\*cp.PreCursoProg Ingreso\_proyectado ,(cp.Matriculados\* cp.PreCursoProg) Ingreso\_real,

(cp.Matriculados\*100/(cp.Vacantes +cp.Matriculados)) as cumplimiento

from CursoProgramado cp, Curso c, Tarifa t

where cp.idCurso = c.IdCurso

and t.IdTarifa = c.IdTarifa

and cp.IdCiclo = @IdCiclo

END

return 0

create table #Result

(idCurso char(4),

NomCurso varchar(85) NOT NULL,

Vacantes numeric,

Matriculados numeric,

Ingreso\_proyectado numeric,

Ingreso\_real numeric,

cumplimiento numeric

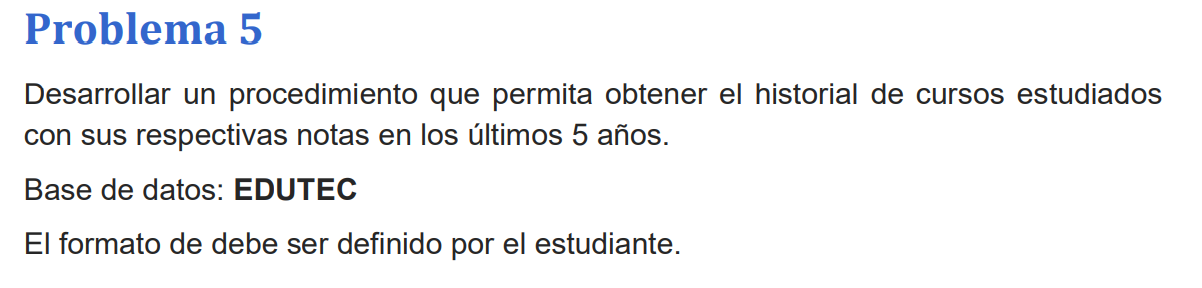
);

Insert into #Result

EXEC [dbo].usp\_reporte\_CumplimientoCurso '2011-04'

select \* From #Result

drop table #Result



--Desarrollar un procedimiento que permita obtener el historial de cursos estudiados

--con sus respectivas notas en los últimos 5 años.

CREATE OR ALTER PROCEDURE [dbo].[usp\_reporte\_HistorialAlumno]

@IdCodigo CHAR(5)

AS

BEGIN

select cp.IdCiclo,a.IdAlumno, a.NomAlumno + ' ' + a.ApeAlumno as Estudiante,m.FecMatricula,c.IdCurso , c.NomCurso, cp.Horario ,

m.ExaParcial,m.ExaFinal,m.Promedio,ISNULL(m.ExaSub, 0) as ExaSusti, p.NomProfesor + ' ' + p.ApeProfesor as Docente

From Matricula m , Alumno a , CursoProgramado cp , Curso c ,Profesor p

where a.idAlumno =@IdCodigo

and cp.idCurso = c.IdCurso

and a.IdAlumno = m.IdAlumno

and m.IdCursoProg = cp.IdCursoProg

and cp.IdProfesor = p.IdProfesor

and m.FecMatricula> DATE(YEAR,-5,GETDATE())

order by FecMatricula desc

END

return 0

create table #ResultHistorial

(CICLO char(7),

COD\_ALUMNO char(5) NOT NULL,

ALUMNO varchar(85) NOT NULL,

FEC\_MATRICULA varchar(85) NOT NULL,

COD\_CURSO char(4) NOT NULL,

CURSO varchar(50) NOT NULL,

HORARIO varchar(30),

NOTAPARCIAL numeric,

NOTAFINAL numeric,

PROMEDIO numeric,

SUSTITUTORIO numeric,

PROFESOR varchar(60) NOT NULL,

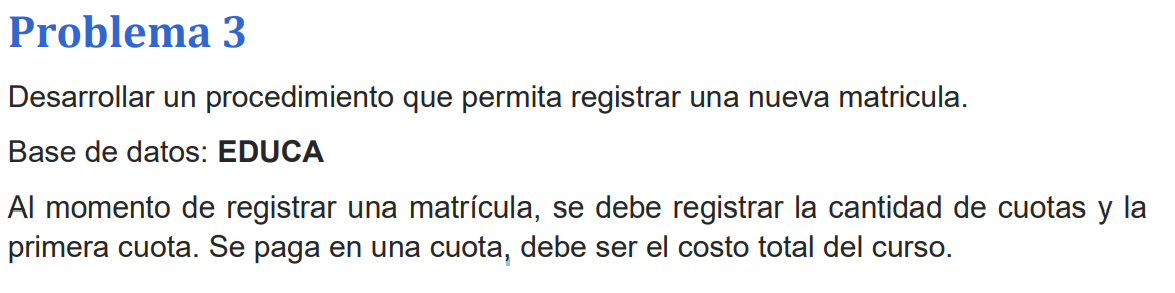
);

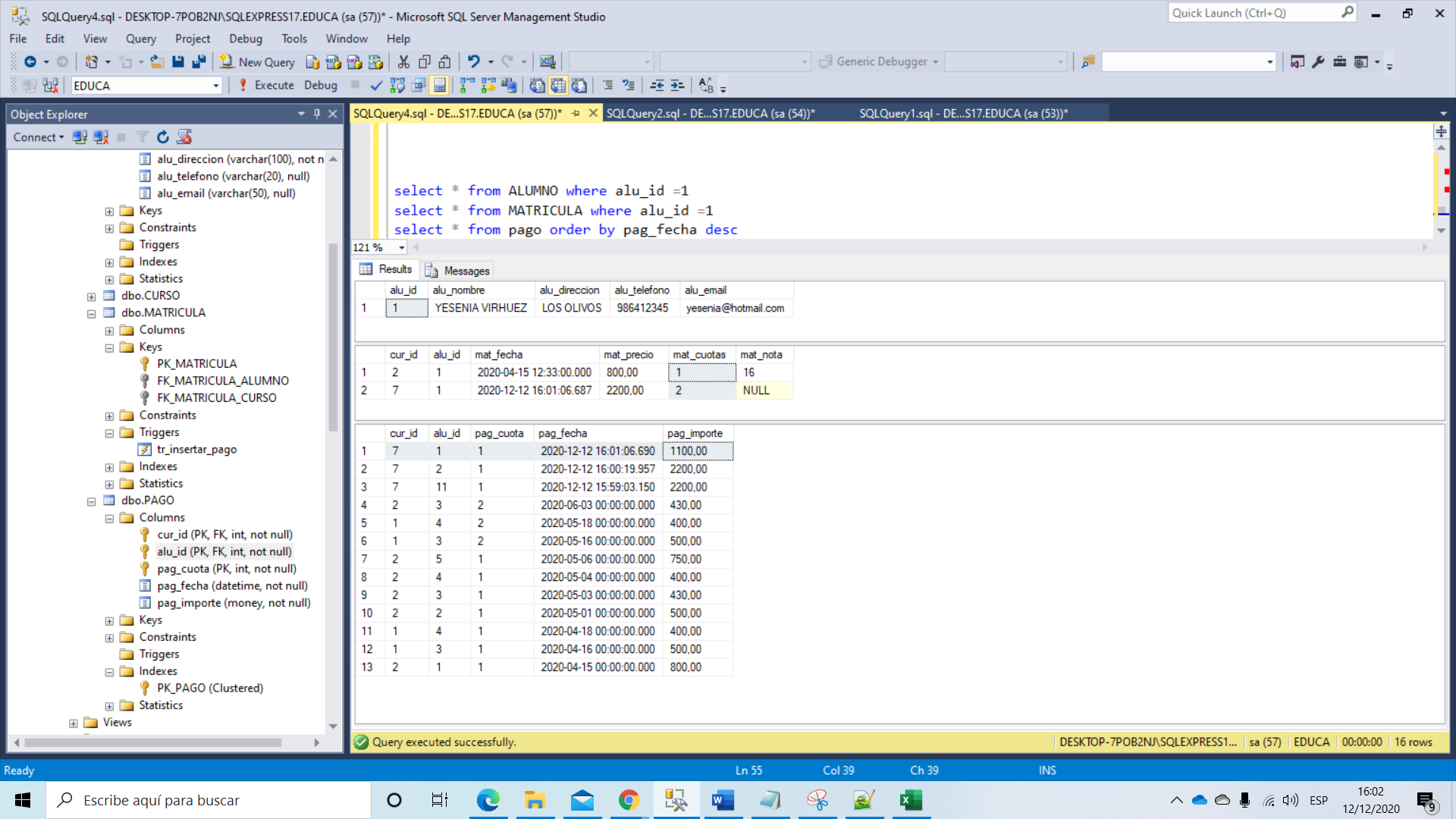
Insert into #ResultHistorial

EXEC [dbo].usp\_reporte\_HistorialAlumno 'A0025'

select \* From #ResultHistorial

drop table #ResultHistorial





### BD EDUCA

--Desarrollar un procedimiento que permita registrar una nuevo matricula

CREATE or ALTER PROCEDURE [dbo].[usp\_registra\_Matricula]

@IdCurso int,

@IdAlu int,

@num\_cuotas int

AS

DECLARE @precio as money;

--seteo el precio de la tabla CURSO

SELECT @precio = cur\_precio from CURSO where cur\_id =@IdCurso ;

INSERT INTO Matricula(cur\_id,alu\_id,mat\_precio,mat\_fecha,mat\_cuotas) VALUES

(@IdCurso,@IdAlu,@precio,GETDATE(),@num\_cuotas);

return 0

create or alter trigger dbo.tr\_insertar\_pago

on dbo.matricula

for insert

as

begin

--se dispara cuando se realiza un inserta automaticamente

insert into dbo.PAGO(cur\_id, alu\_id, pag\_cuota, pag\_fecha, pag\_importe)

SELECT I.CUR\_ID,I.alu\_id,1,GETDATE(),

CASE I.mat\_cuotas When 1 Then I.mat\_precio ELSE (I.mat\_precio/I.mat\_cuotas) END

FROM ALUMNO A , INSERTED I

WHERE A.alu\_id = i.alu\_id;

end;

go

exec [usp\_registra\_Matricula] 7,2,1 -- lo hace el usuario

exec [usp\_registra\_Matricula] 7,1,2

select \* from curso

select \* from ALUMNO where alu\_id =1

select \* from MATRICULA where alu\_id =1

select \* from pago order by pag\_fecha desc

select \* from MATRICULA order by 3 desc

|  |  |  |  |
| --- | --- | --- | --- |
| PROC 1 | REGISTRAS MATRICULA |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  | inserta matricula **curso**, **alumno** ,**fecha** y el número de **cuotas** |
| **TRIGGER** | inserta pago inicial | IF( CUOTA = 1 ) | insert pago pag\_importe = cur\_precio |
| Else | insert pago pag\_importe = cur-precio/nro\_cuotas |

use intranet

go

create table alumno

(

idalumno int identity,

nombre nvarchar(20),

apellido nvarchar(20),

dni varchar(10),

estado char(4),

usuario varchar(10),

clave VARBINARY(8000),

primary key (idalumno)

);

insert into alumno values('RAUL','ARMAS','73262442','S','rarmas',ENCRYPTBYPASSPHRASE('password', 'trilce'));

--

select \* from alumno

SELECT

CONVERT(VARCHAR(MAX), DECRYPTBYPASSPHRASE('password', clave))

FROM alumno

Para querys .

select \* from usuarios where usuario='RMB' and CONVERT(VARCHAR(MAX), DECRYPTBYPASSPHRASE('clave', clave)) = 'trilce'

create procedure uspLoginUsuario

@username varchar(250),

@clave varchar(250)

as

begin

if exists(select \* from alumno where usuario=@username and CONVERT(VARCHAR(MAX), DECRYPTBYPASSPHRASE('password', clave))=@clave)

begin

select\*from alumno where usuario=@username

end

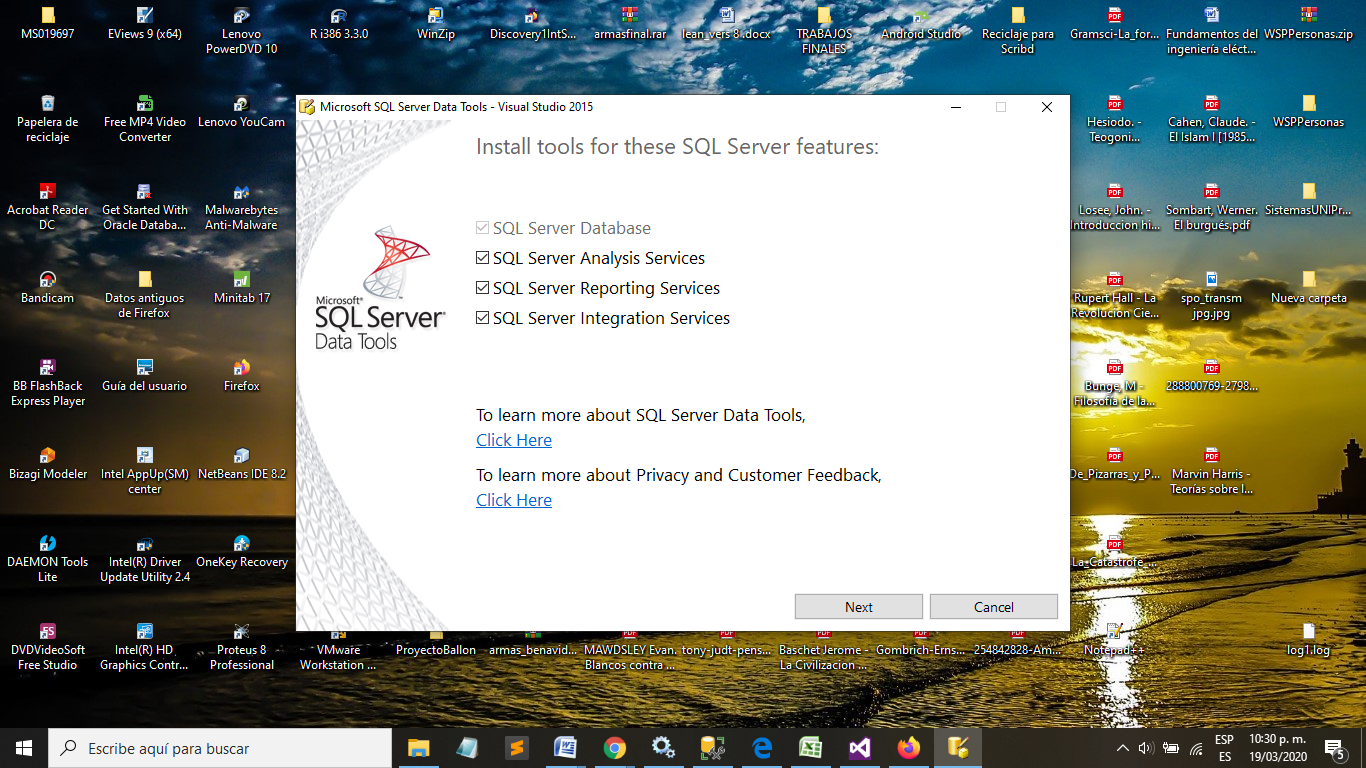
end

exec uspLoginUsuario 'rarmas','trilce'

# Business Inteligence

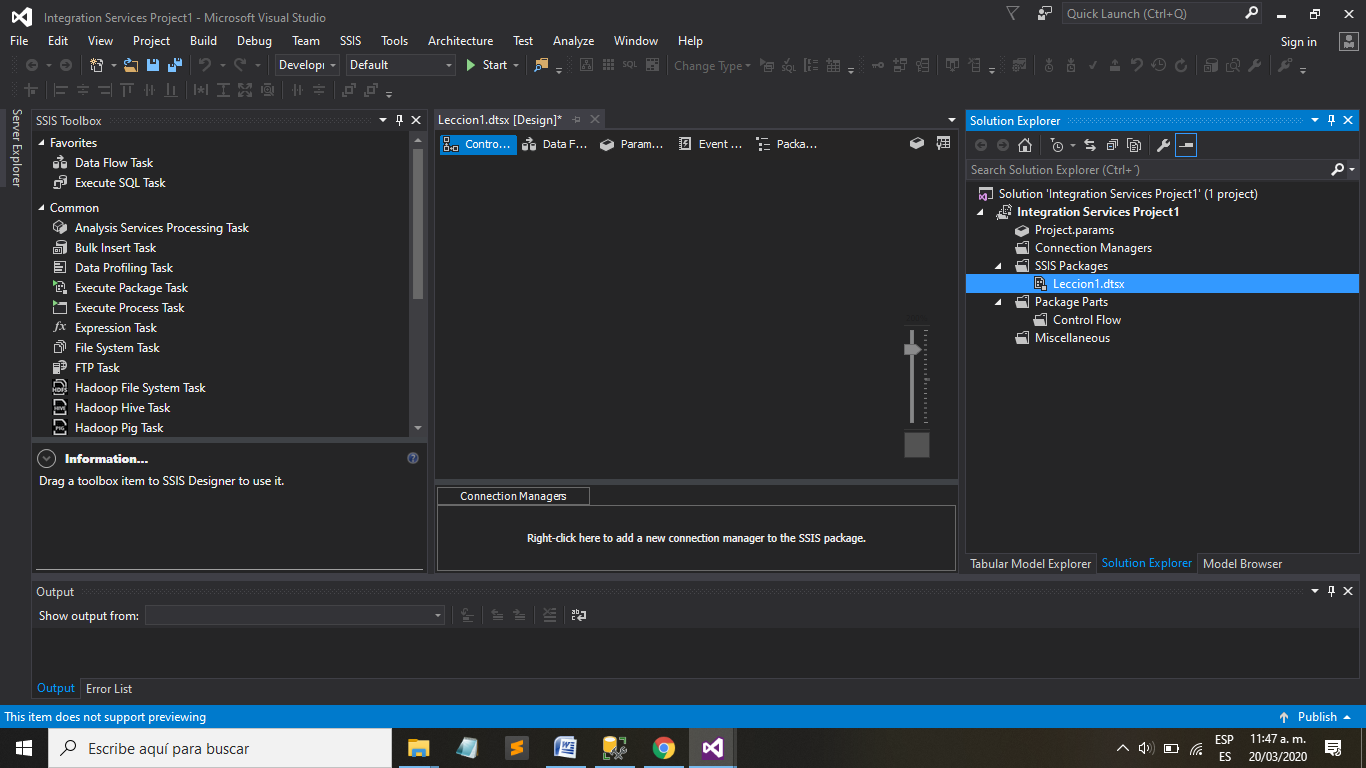
## SQL SERVER DATA TOOLS

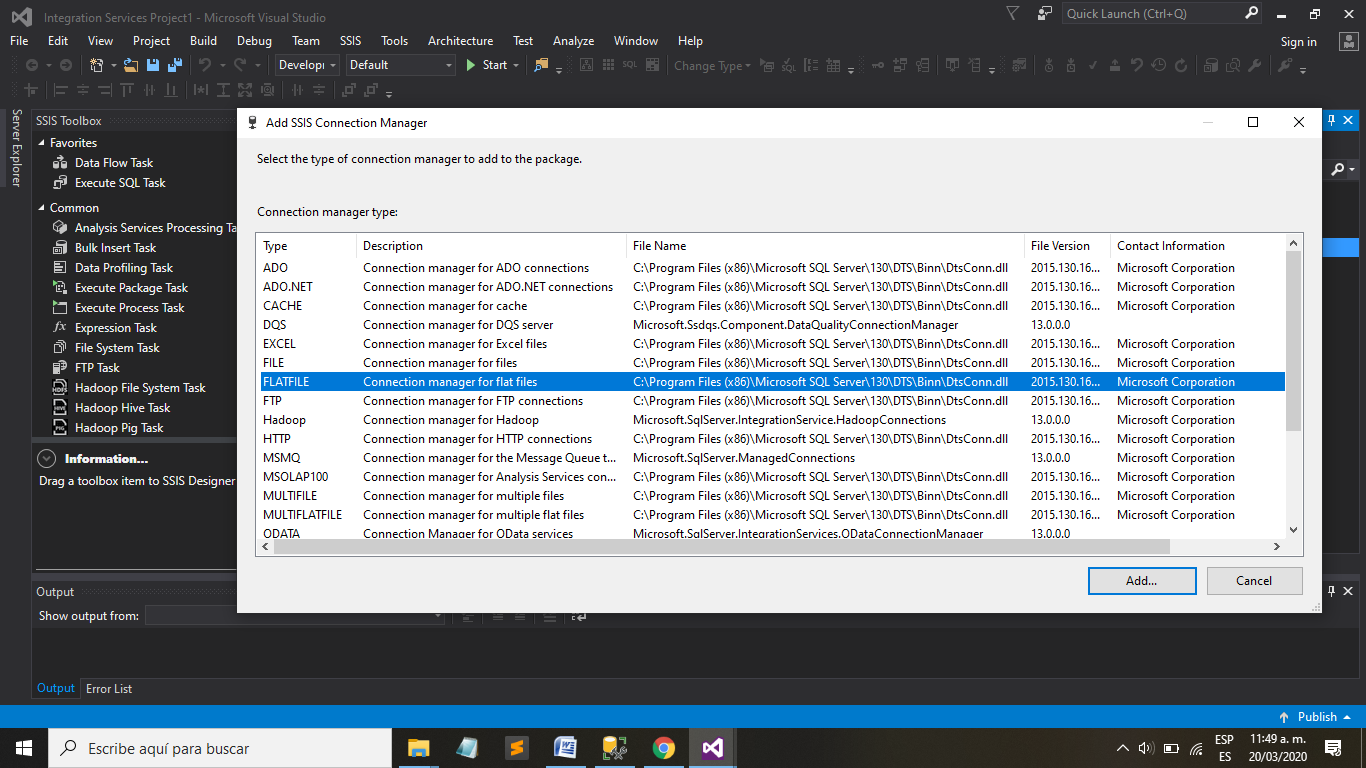


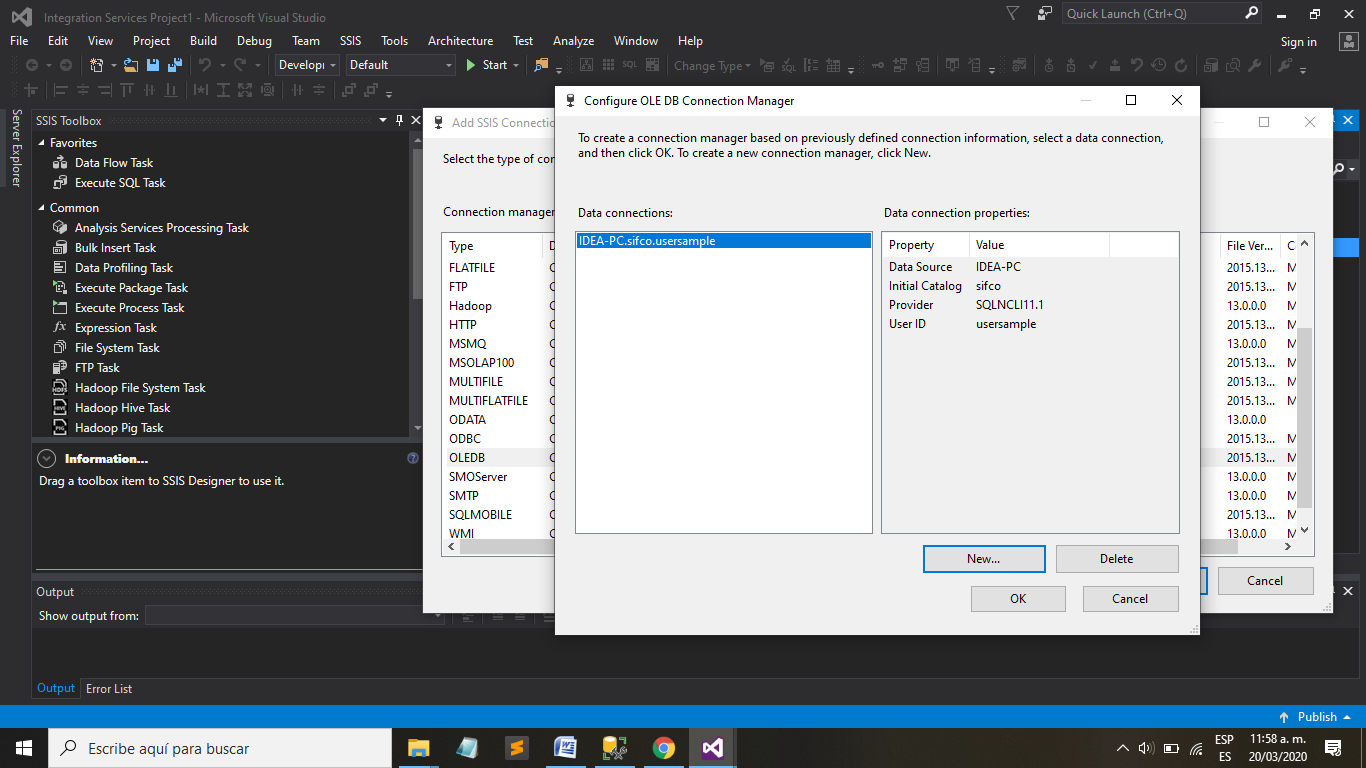


* **SQL Server Integration Services : Herramienta ETL que permite la extracción, transformación y carga de datos.**
* SQL Server Analysis Services : Herramienta OLAP que permite la creación de cubos de información y Datamining.
* SQL Server Reporting Services : Herrmienta Enterprise Reporting que satisface los 3 niveles del ciclo de vida de reportes empresariales : Creación , Administración y Envío.

### SQL Server Integration Services SSIS

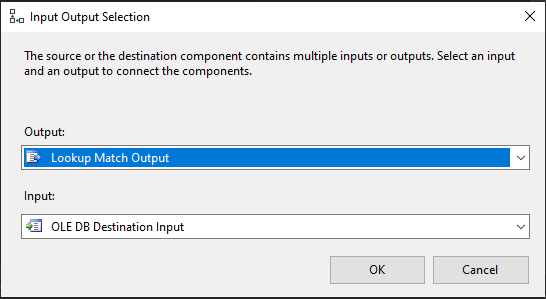


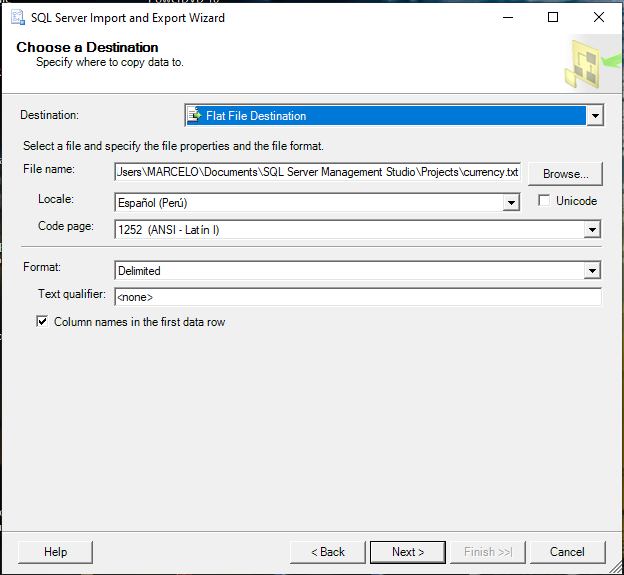


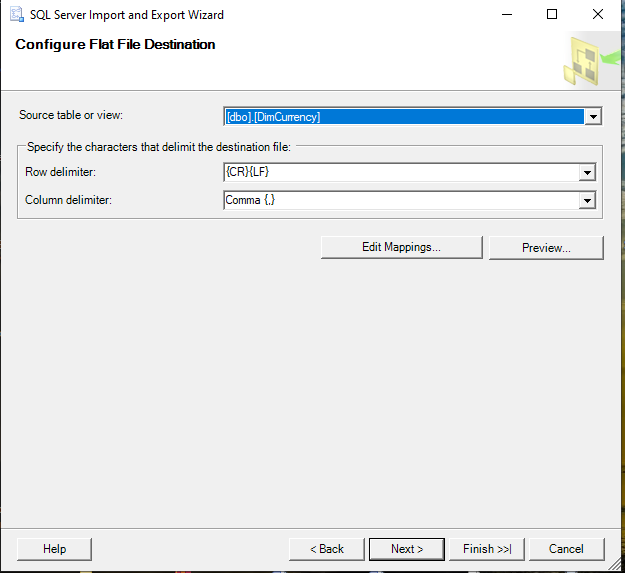


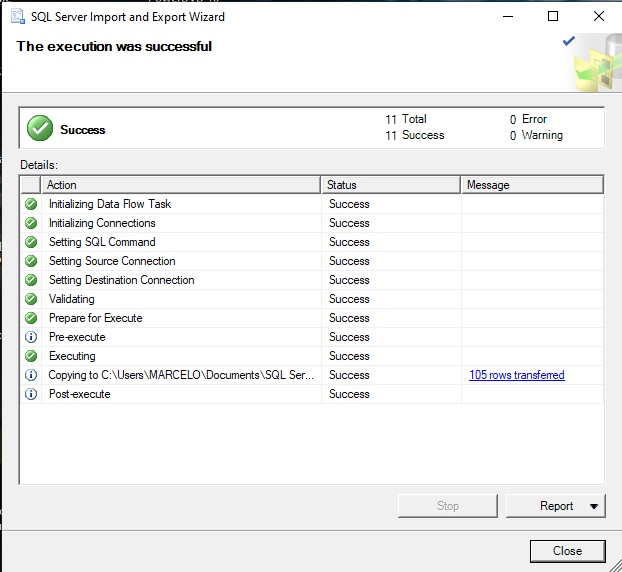


select **Lookup Match Output**





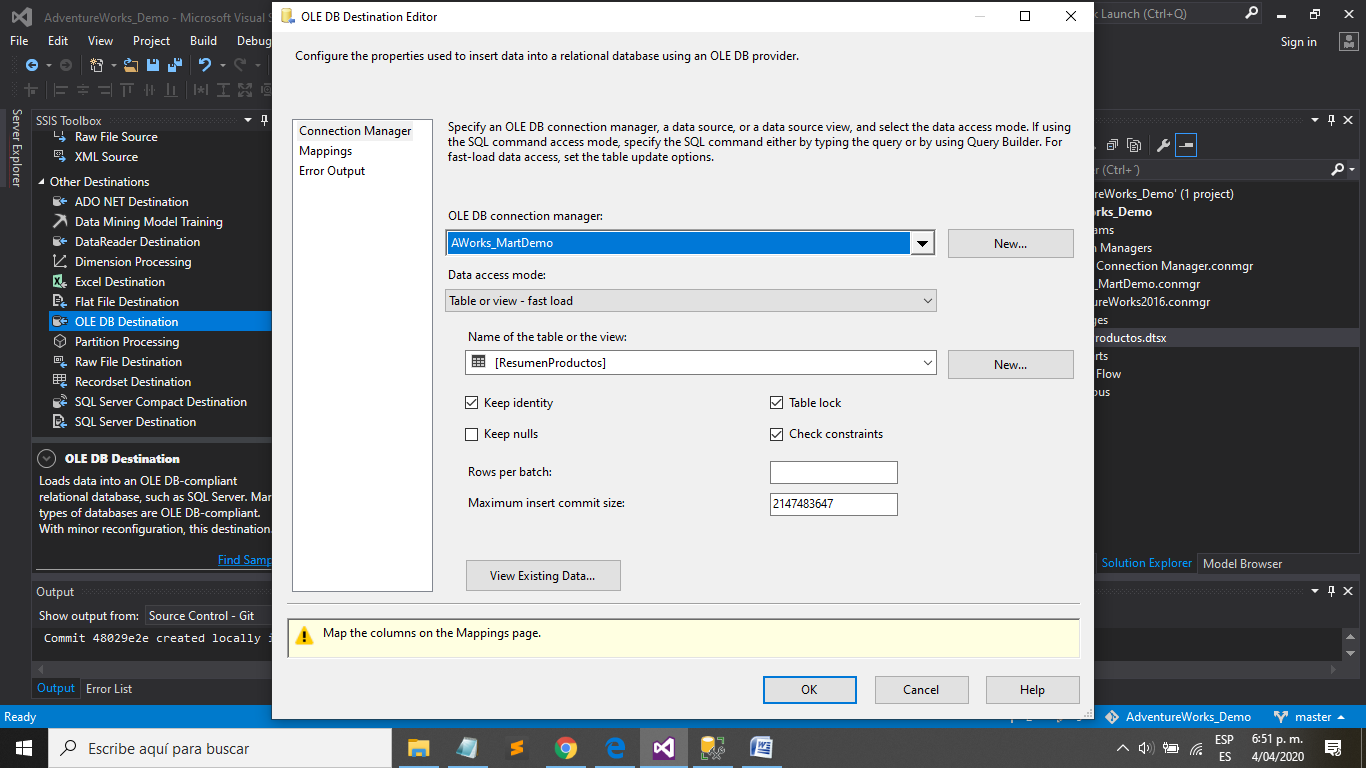




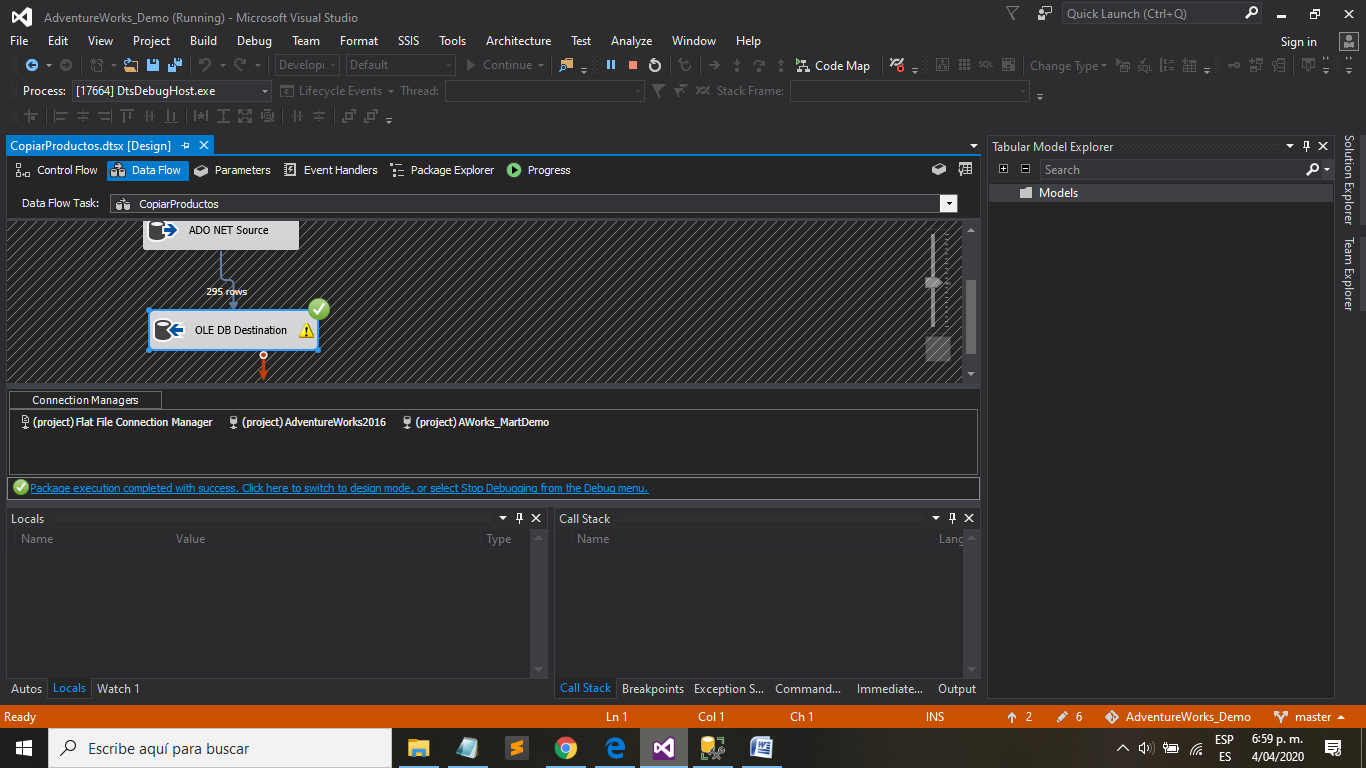
## Adworks

### Desde un ODBC Source

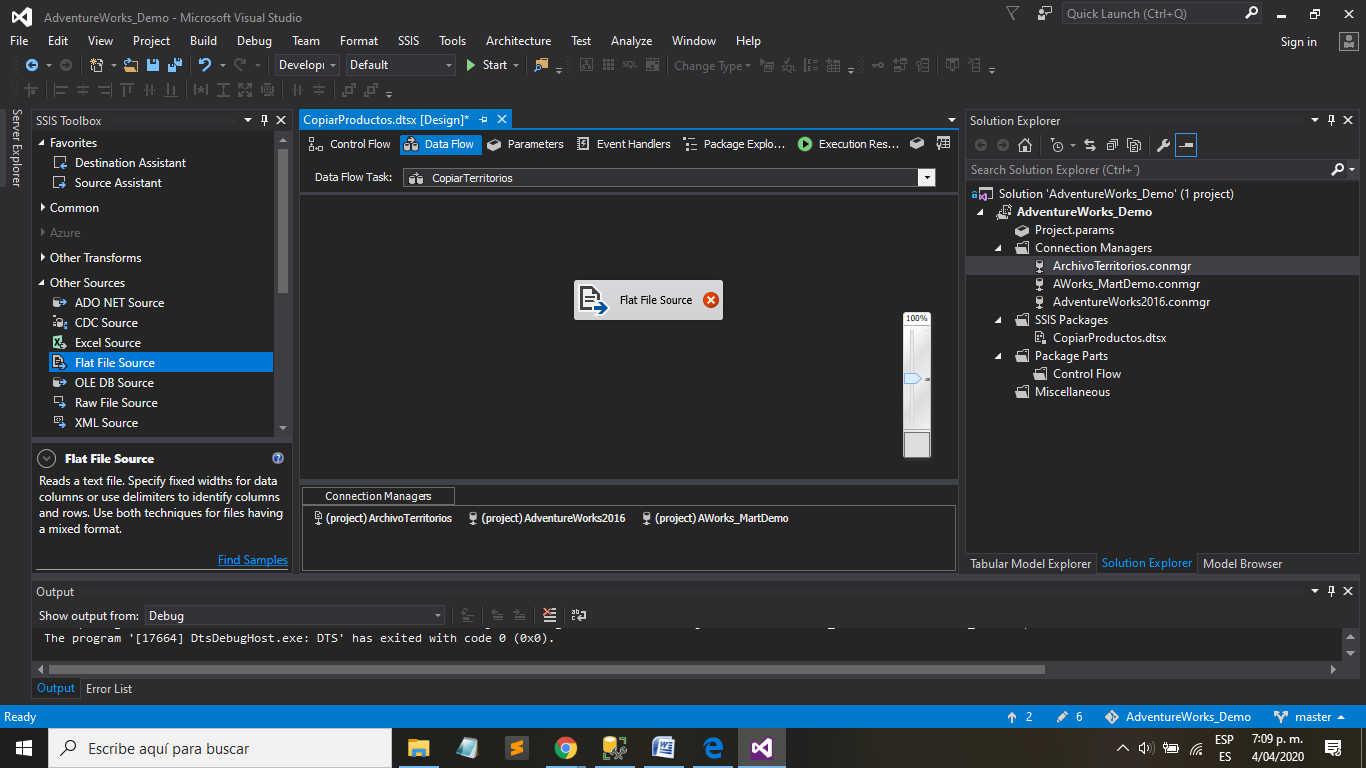
Se configura ODBC Data Source y OLEDB Destination. ( se crea automáticamente tablas) con la opción de “fast load”.



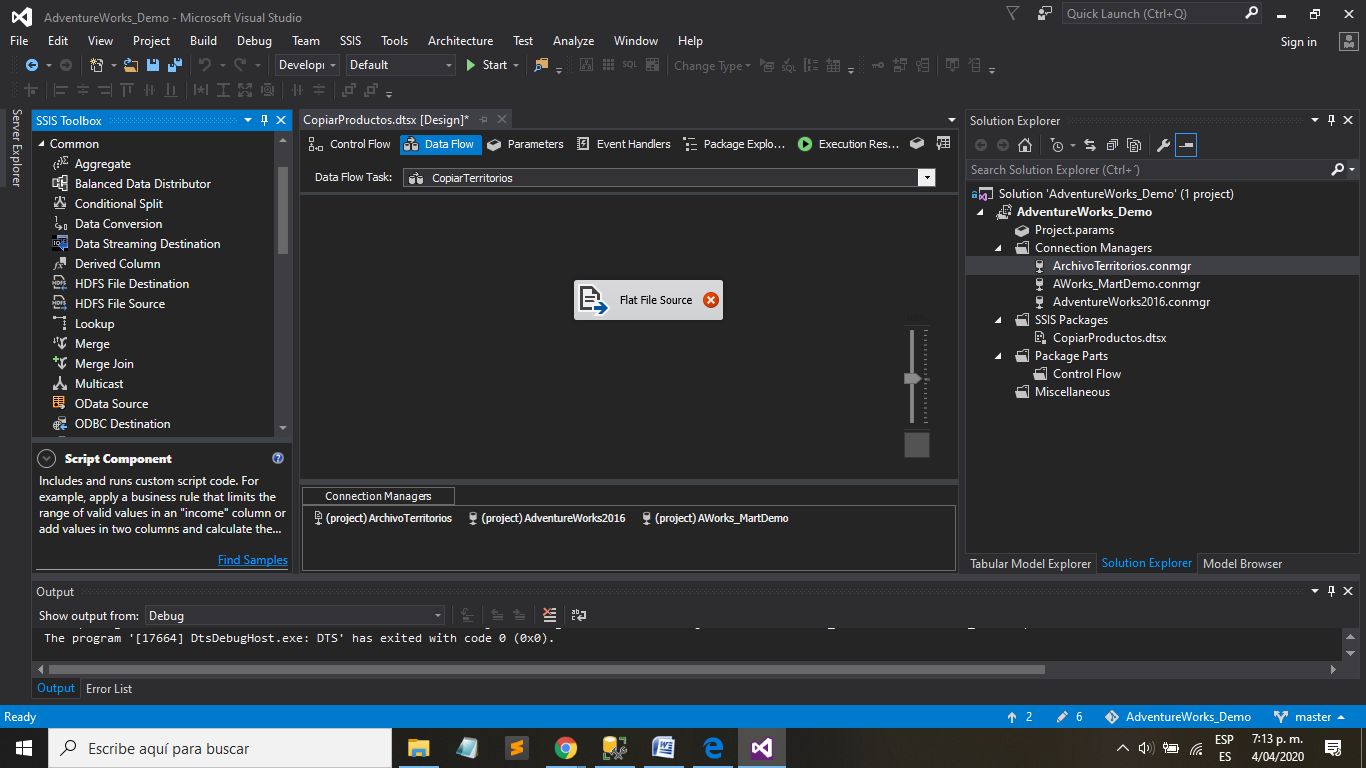
Resultado de la ejecución



### Desde un archivo plano ( territorios.txt)

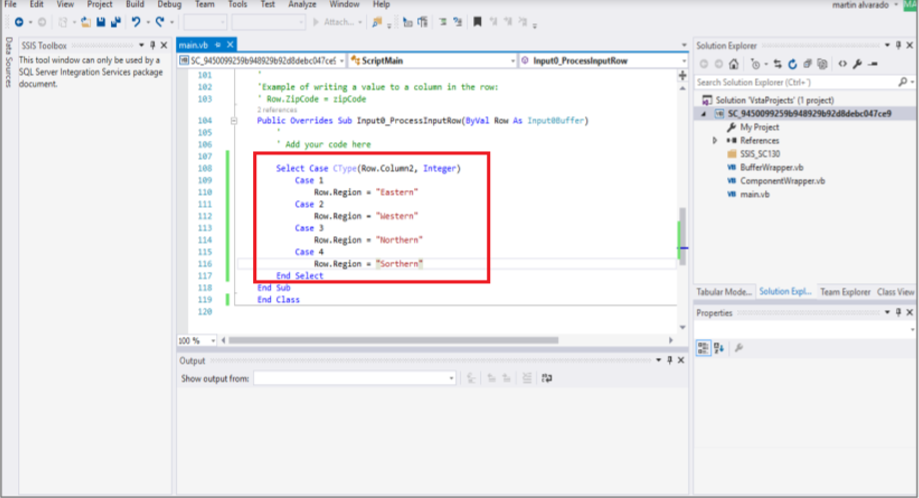


En el SSIS ToolBox están distintas opciones , para la transformación de datos está ScriptComponent.

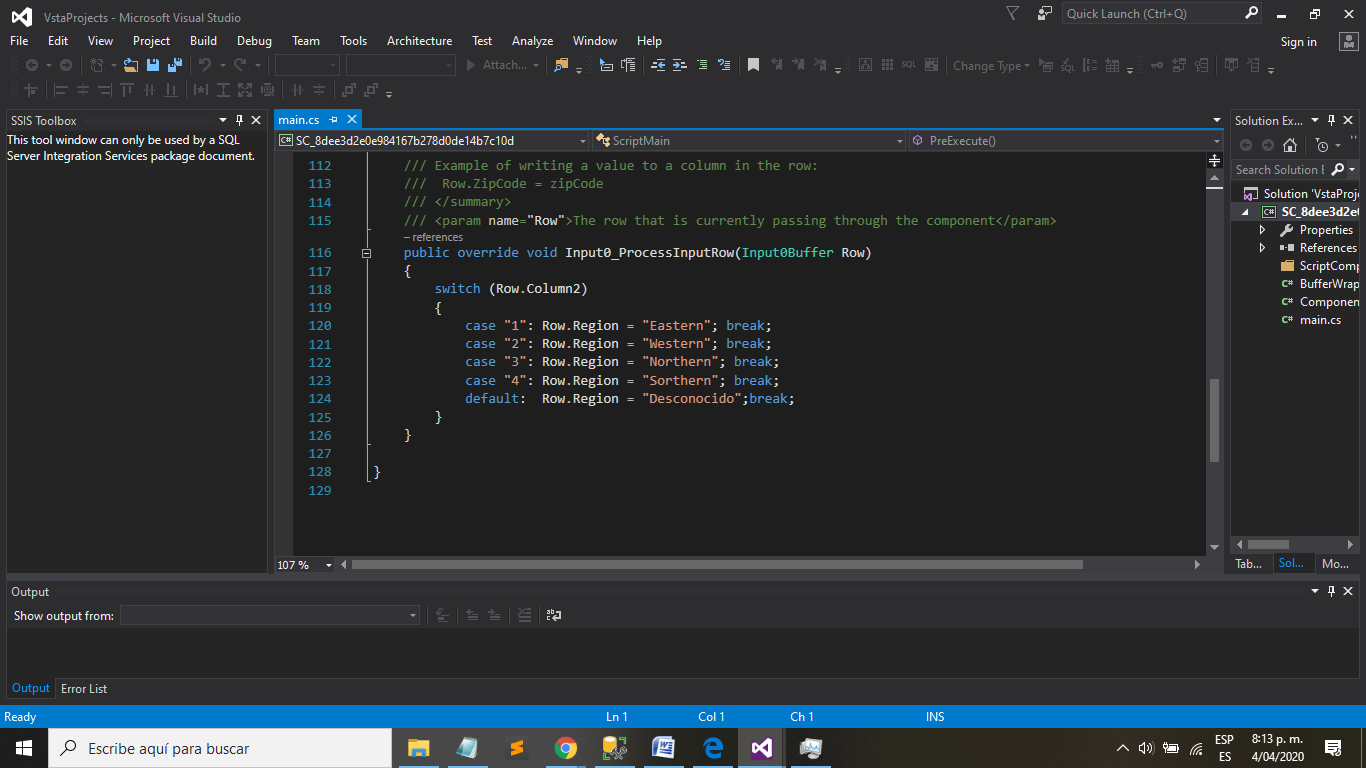


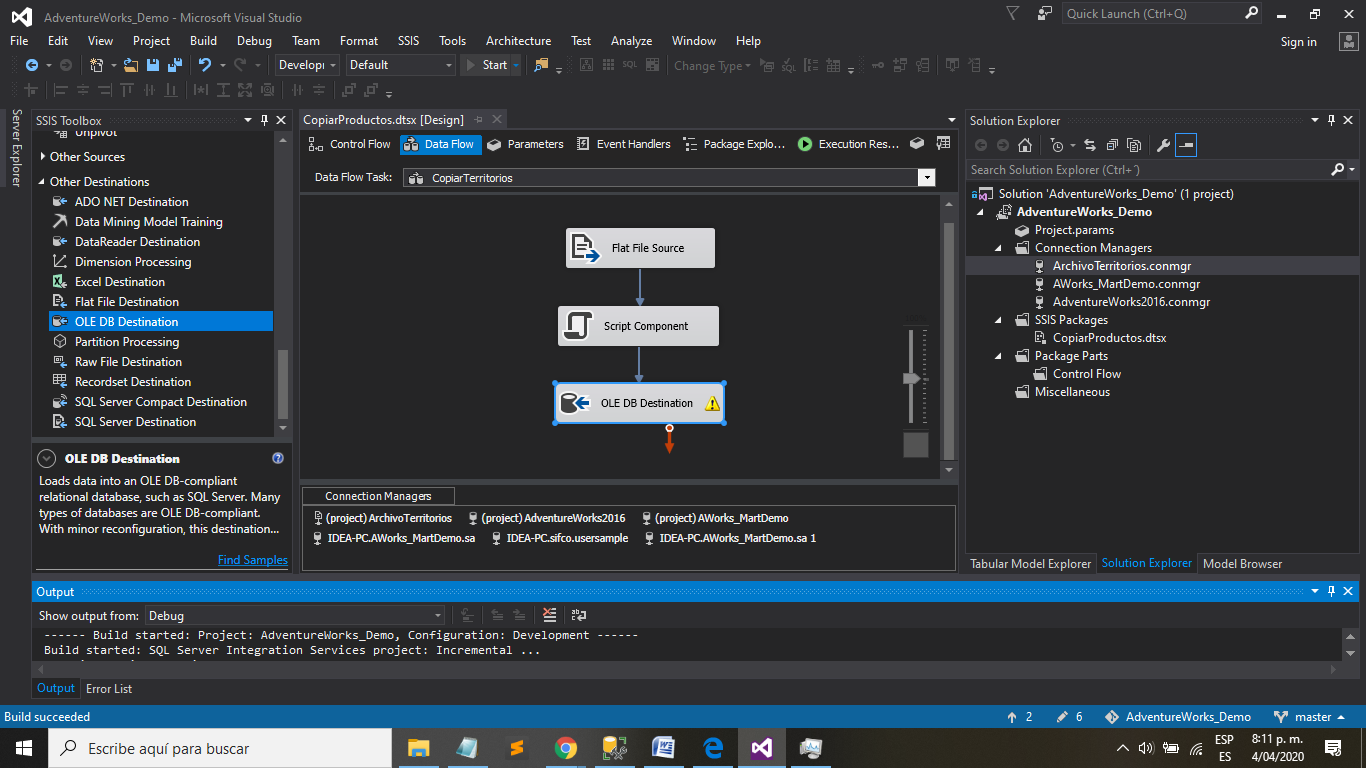
El método Input\_ProcessInputRow , ejemplo :

**En Visual Basic .NET**

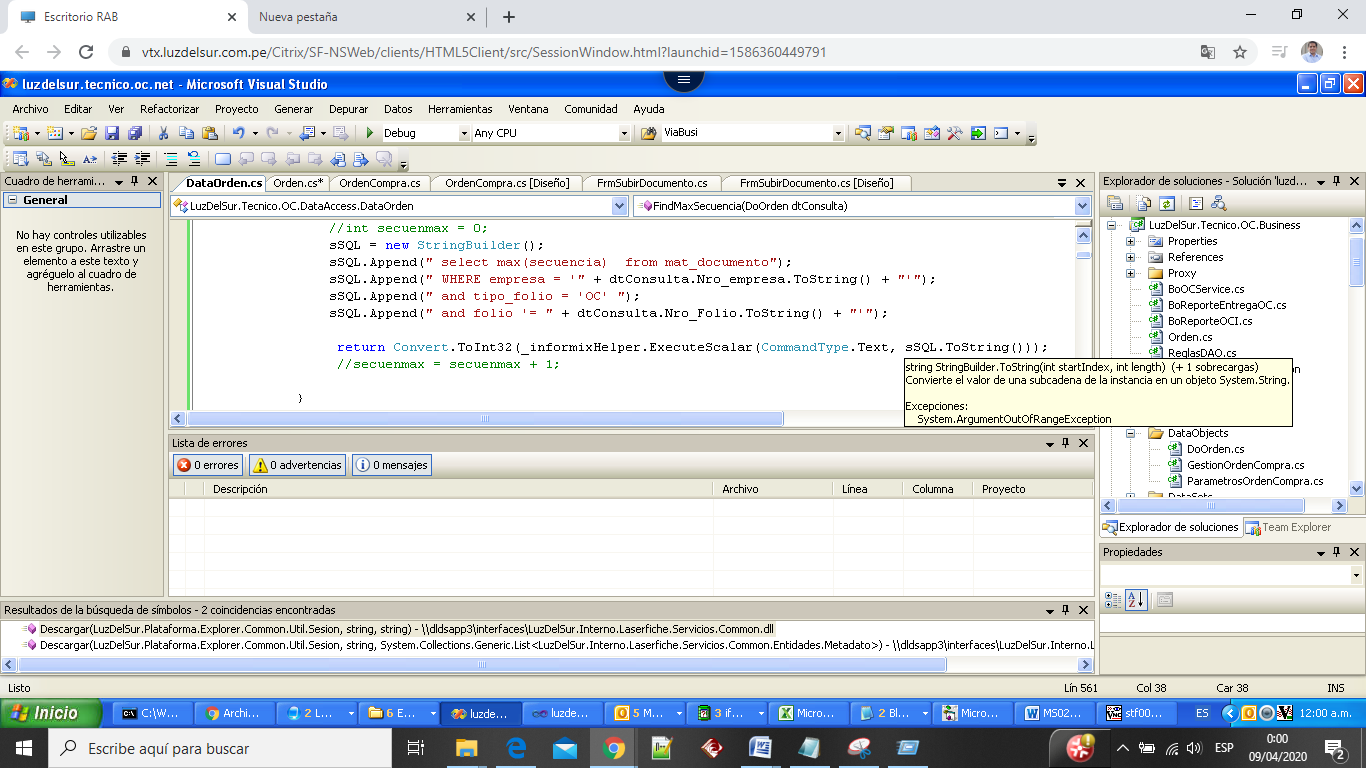


En **C#**

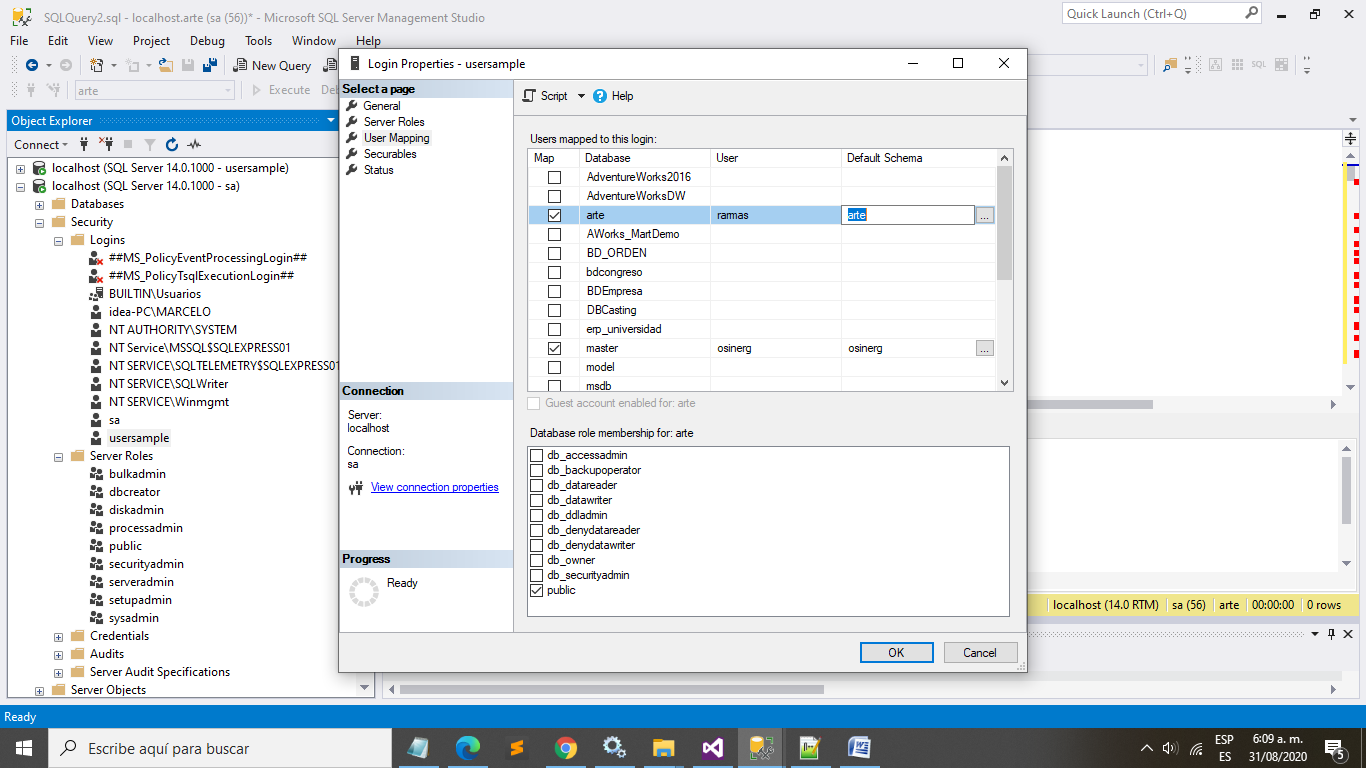




nvl



Se tiene que ver en User mapping , allí está los usuarios asociados a un login y su schema por tabla

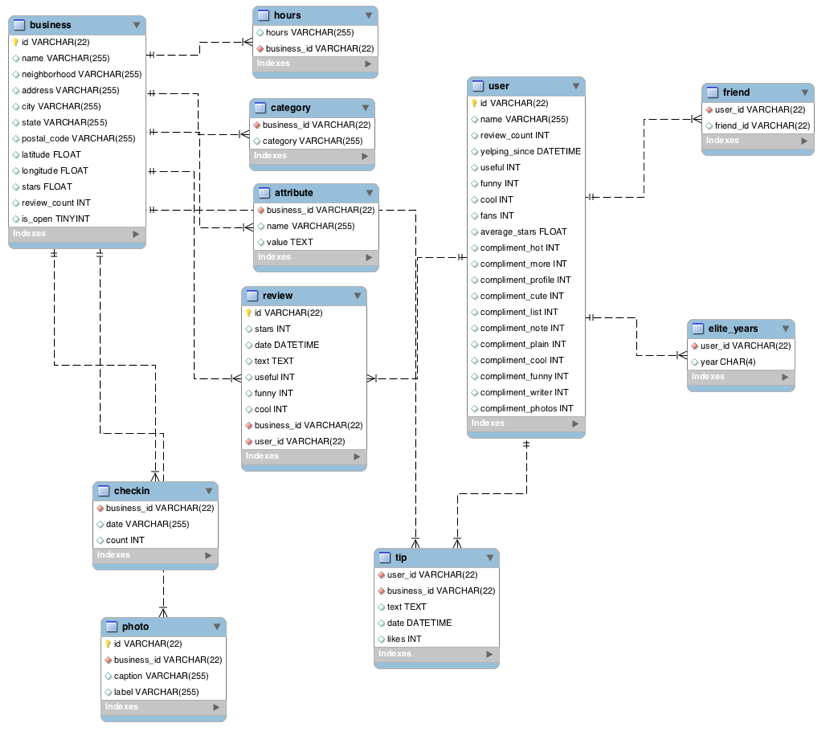


# TAREAS

## Tarea 1

Lo realicé con Dayana

**Database : Yelp**



Data Scientist Role Play: Profiling and Analyzing the Yelp Dataset Coursera Worksheet

This is a 2-part assignment. In the first part, you are asked a series of questions that will help you profile and understand the data just like a data scientist would. For this first part of the assignment, you will be assessed both on the correctness of your findings, as well as the code you used to arrive at your answer. You will be graded on how easy your code is to read, so remember to use proper formatting and comments where necessary.

In the second part of the assignment, you are asked to come up with your own inferences and analysis of the data for a particular research question you want to answer. You will be required to prepare the dataset for the analysis you choose to do. As with the first part, you will be graded, in part, on how easy your code is to read, so use proper formatting and comments to illustrate and communicate your intent as required.

For both parts of this assignment, use this "worksheet." It provides all the questions you are being asked, and your job will be to transfer your answers and SQL coding where indicated into this worksheet so that your peers can review your work. You should be able to use any Text Editor (Windows Notepad, Apple TextEdit, Notepad ++, Sublime Text, etc.) to copy and paste your answers. If you are going to use Word or some other page layout application, just be careful to make sure your answers and code are lined appropriately.

In this case, you may want to save as a PDF to ensure your formatting remains intact for you reviewer.

**Part 1: Yelp Dataset Profiling and Understanding**

**1. Profile the data by finding the total number of records for each of the tables below:**

i. Attribute table = 10000

ii. Business table = 10000

iii. Category table = 10000

iv. Checkin table = 10000

v. elite\_years table = 10000

vi. friend table = 10000

vii. hours table = 10000

viii. photo table = 10000

ix. review table = 10000

x. tip table = 10000

xi. user table = 10000

**2. Find the total distinct records by either the foreign key or primary key for each table. If two foreign keys are listed in the table, please specify which foreign key.**

i. Business =

ii. Hours =

iii. Category =

iv. Attribute =

v. Review =

vi. Checkin =

vii. Photo =

viii. Tip =

ix. User =

x. Friend =

xi. Elite\_years =

Note: Primary Keys are denoted in the ER-Diagram with a yellow key icon.

**3. Are there any columns with null values in the Users table? Indicate "yes," or "no."**

Answer: Zero rows

SQL code used to arrive at answer:

select \* from user

where name =null

or review\_count = null

or yelping\_since =null

or useful = null

or funny = null

or cool = null

or fans = null

or average\_stars = null

or compliment\_hot = null

or compliment\_more =null

or compliment\_profile = null

or compliment\_cute = null

or compliment\_list= null

or compliment\_note = null

or compliment\_plain= null

or compliment\_cool= null

or compliment\_funny = null

or compliment\_writer  = null

or compliment\_photos = null

**4. For each table and column listed below, display the smallest (minimum), largest (maximum), and average (mean) value for the following fields:**

i. Table: Review, Column: Stars

min:1 max: 5 avg ( average): 3.7082

ii. Table: Business, Column: Stars

min: 1.0 max: 5.0 avg: 3.6549

iii. Table: Tip, Column: Likes

min: 0 max: 2 avg: 0.0144

iv. Table: Checkin, Column: Count

min: 1 max: 53 avg: 1.9414

v. Table: User, Column: Review\_count

min: 0 max: 2000 avg: 24.2995

**5. List the cities with the most reviews in descending order:**

SQL code used to arrive at answer:

Copy and Paste the Result Below:

6. Find the distribution of star ratings to the business in the following cities:

i. Avon

SQL code used to arrive at answer:

Copy and Paste the Resulting Table Below (2 columns â€“ star rating and count):

ii. Beachwood

SQL code used to arrive at answer:

Copy and Paste the Resulting Table Below (2 columns â€“ star rating and count):

7. Find the top 3 users based on their total number of reviews:

SQL code used to arrive at answer:

Copy and Paste the Result Below:

8. Does posing more reviews correlate with more fans?

Please explain your findings and interpretation of the results:

**9. Are there more reviews with the word "love" or with the word "hate" in them?**

Answer: Yes

SQL code used to arrive at answer:

select 'LOVE' as palabra, count(\*) as total from ( select \* from review

where text LIKE '%love%')

union

select 'HATE' as palabra , count(\*) as total  from ( select \* from review

where text LIKE '%hate%')

+---------+-------+

| palabra | total |

+---------+-------+

| HATE | 232 |

| LOVE | 1780 |

+---------+-------+

**10. Find the top 10 users with the most fans**:

SQL code used to arrive at answer:

select name,fans from user

order by fans desc

limit 10

Copy and Paste the Result Below:

+-----------+------+

| name | fans |

+-----------+------+

| Amy | 503 |

| Mimi | 497 |

| Harald | 311 |

| Gerald | 253 |

| Christine | 173 |

| Lisa | 159 |

| Cat | 133 |

| William | 126 |

| Fran | 124 |

| Lissa | 120 |

+-----------+------+

Part 2: Inferences and Analysis

1. Pick one city and category of your choice and group the businesses in that city or category by their overall star rating. Compare the businesses with 2-3 stars to the businesses with 4-5 stars and answer the following questions. Include your code.

i. Do the two groups you chose to analyze have a different distribution of hours?

ii. Do the two groups you chose to analyze have a different number of reviews?

iii. Are you able to infer anything from the location data provided between these two groups? Explain.

SQL code used for analysis:

2. Group business based on the ones that are open and the ones that are closed. What differences can you find between the ones that are still open and the ones that are closed? List at least two differences and the SQL code you used to arrive at your answer.

i. Difference 1:

ii. Difference 2:

SQL code used for analysis:

3. For this last part of your analysis, you are going to choose the type of analysis you want to conduct on the Yelp dataset and are going to prepare the data for analysis.

Ideas for analysis include: Parsing out keywords and business attributes for sentiment analysis, clustering businesses to find commonalities or anomalies between them, predicting the overall star rating for a business, predicting the number of fans a user will have, and so on. These are just a few examples to get you started, so feel free to be creative and come up with your own problem you want to solve. Provide answers, in-line, to all of the following:

i. Indicate the type of analysis you chose to do:

ii. Write 1-2 brief paragraphs on the type of data you will need for your analysis and why you chose that data:

iii. Output of your finished dataset:

iv. Provide the SQL code you used to create your final dataset:

# Administration ( ADMINISTRACIÓN )

## Extended Event

* How to Navigate to Extended evento in SQL Server using SSMS?
* How to créate new Extended Event sesión using sesión Wizard in SSMS ?
* How to create Event Package

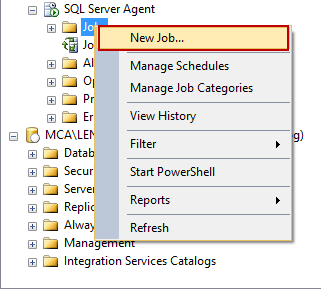
### Master certification

## Log Shipping

## Jobs

### Backup

Haga clic en el botón derecho en **Trabajos (Jobs)** y seleccione **Nuevo trabajo (New Job)** en el menú contextual, en el panel del explorador de objetos, en SSMS, debajo del nodo Agente SQL Server.



Haga clic en el botón derecho en **Trabajos (Jobs)** y seleccione **Nuevo trabajo (New Job)** en el menú contextual, en el panel del explorador de objetos, en SSMS, debajo del nodo Agente SQL Server.

Con el objeto de realizar una copia de seguridad de todas las bases de datos, por medio del uso del Agente SQL Server, para una instancia de SQL Server hay dos enfoques; en ambos se requerirá de un esfuerzo adicional. Una forma es crear un paquete en SSIS utilizando la opción **Tarea de base de datos** de respaldo de la barra de herramientas SSIS y así crear un trabajo del Agente SQL Server, luego programarlo

El enfoque es crear una secuencia de comandos en SQL para poder hacer una copia de seguridad de todas las bases de datos en el cuadro de diálogo del Paso de trabajo del Agente SQL Server.

Aquí existe un resumen de enfoques sobre cómo poder programar una copia de seguridad de la base de datos comparada de lado a lado:

|  |  |  |
| --- | --- | --- |
|  | **Maintenance Plans** | **SQL Server Agent** |
| Run jobs without SQL scripting required | X |  |
| Backup all databases without SQL coding | X |  |
| Execute a task with a single job |  | X |

## Job envío de correo

create or alter procedure sp\_notificacion2   
as  
DECLARE @MAIL\_BODY VARCHAR(8000)  
-- no tomar discapacitados ni personas con tbc   
--considerar madres gestantes de 0 a 42 semanas    
--considerar madres lactantes  de 0 a 1 año  
-- hacer consulta al 31/07/2004   
  
IF EXISTS ( SELECT name FROM sysobjects  
   WHERE type = 'U' AND name = 'servicio' )  
   DROP TABLE #Result  
  
create table  #Result   
( niños1 int,  
niños2 int,  
gesta int,  
lacta int,  
niños3 int,  
anciano int  
 );  
  
  
--QUERY GENERAL  
insert into #Result  
select   
(select count(\*) From Beneficiario b   
where b.cod\_tip\_ti= 1   
and  DATEDIFF(day, fec\_nac\_ben\_da,'31-07-2004')<365) as niños1,  
(select count(\*) as cant\_2 From Beneficiario b   
where b.cod\_tip\_ti= 1   
and  DATEDIFF(day, fec\_nac\_ben\_da,'31-07-2004') >=365  
and  DATEDIFF(day, fec\_nac\_ben\_da,'31-07-2004')<365\*7) as niños2  
,  
(  
select count(\*) From Beneficiario b   
where b.cod\_tip\_ti= 2  
 and  DATEDIFF(week, fec\_emb\_ben\_da,'31-07-2004') <= 42) as  gesta--gestante  
,  
(select count(\*)  From Beneficiario b   
where b.cod\_tip\_ti= 3  
 and  DATEDIFF(day, fec\_lac\_ben\_da,'31-07-2004') <= 365 ) as lacta --lactante  
,  
(select count(\*) as cant\_3 From Beneficiario b   
where b.cod\_tip\_ti= 1   
and  DATEDIFF(day, fec\_nac\_ben\_da,'31-07-2004')>=(7\*365)  
and  DATEDIFF(day, fec\_nac\_ben\_da,'31-07-2004')<=(13\*365)  
) as niños3  
,(select count(\*) as cant\_6 From Beneficiario b   
where b.cod\_tip\_ti= 4 ) as anciano --anciano  
  
  
/\* HEADER \*/  
SET @MAIL\_BODY = '<table border="1" align="center" cellpadding="2" cellspacing="0" style="color:black;font-family:consolas;text-align:center;">' +  
    '<tr>  
    <th>Niños</th>  
    <th>Niños</th>  
    <th>Gestantes</th>  
    <th>Lacantes</th>  
    <th>Niños3</th>  
    <th>Ancianos</th>  
<th><p>Total</p></th>  
    </tr>'  
   
/\* ROWS \*/  
SELECT  
    @MAIL\_BODY = @MAIL\_BODY +  
        '<tr>' +  
        '<td>' + CAST(niños1 AS VARCHAR(11)) + '</td>' +  
        '<td>' + CAST(niños2 AS VARCHAR(11)) + '</td>' +  
        '<td>' + CAST(gesta AS VARCHAR(11)) + '</td>' +  
        '<td>' + CAST(lacta AS VARCHAR(11)) + '</td>' +  
        '<td>' + CAST(niños3 AS VARCHAR(11)) + '</td>' +  
   '<td>' + CAST(anciano AS VARCHAR(11)) + '</td>' +  
'<td>' + CAST(niños1 + niños2 + gesta + lacta +niños3 + anciano  AS VARCHAR(11)) + '</td>' +  
        '</tr>'  
FROM  
      #Result  
   
SELECT @MAIL\_BODY = @MAIL\_BODY + '</table>'  
   
EXECUTE msdb.dbo.sp\_send\_dbmail @profile\_name =  'perAdministrador'   
    ,@recipients = '[raul\_armasb@hotmail.com](mailto:raul_armasb@hotmail.com)'  
    ,@subject = 'Reporte de Beneficiarios'  
,@body = @MAIL\_BODY  
    ,@body\_format = 'HTML'     
  
  
exec sp\_notificacion2

Tabla

Descripción generada automáticamente

# Optimización de consultas

CONTAINS puede buscar:

Una palabra o frase.

El prefijo de una palabra o frase.

Una palabra cerca de otra palabra.

Una palabra generada inflexivamente a partir de otra (por ejemplo, la unidad de Word es el vástago de inflexión de las unidades, de conducción, de conducción y de conducción).

Una palabra que es sinónimo de otra palabra que usa un diccionario de sinónimos (por ejemplo, la palabra "metal" puede tener sinónimos como "aluminio" y "acero").

Nota: CONTAINS es más rápido y funcionará si se actualiza el índice de texto completo. Si quieres una coincidencia exacta de expresiones regulares, entonces puedes usar like.

# PREGUNTAS

Buenas gente, pregunta de novato, estando una BD en un server web la cuál va a ser consultada por diferentes usuarios cuando usen la aplicación web, que es lo q pasa con la BD durante las transacciones para que varios usuarios puedan consultar a la vez, en algún momento entre uso de un usuario y otro, ésta se bloquea?

Robert Garcia

Todo eso lo administra el motor. En modo lectura no hay bloqueos, se usan en escritura.

# MALAS PRÁCTICAS

Graphical user interface, text

Description automatically generated

Respuesta:

Antes que todo, hola.

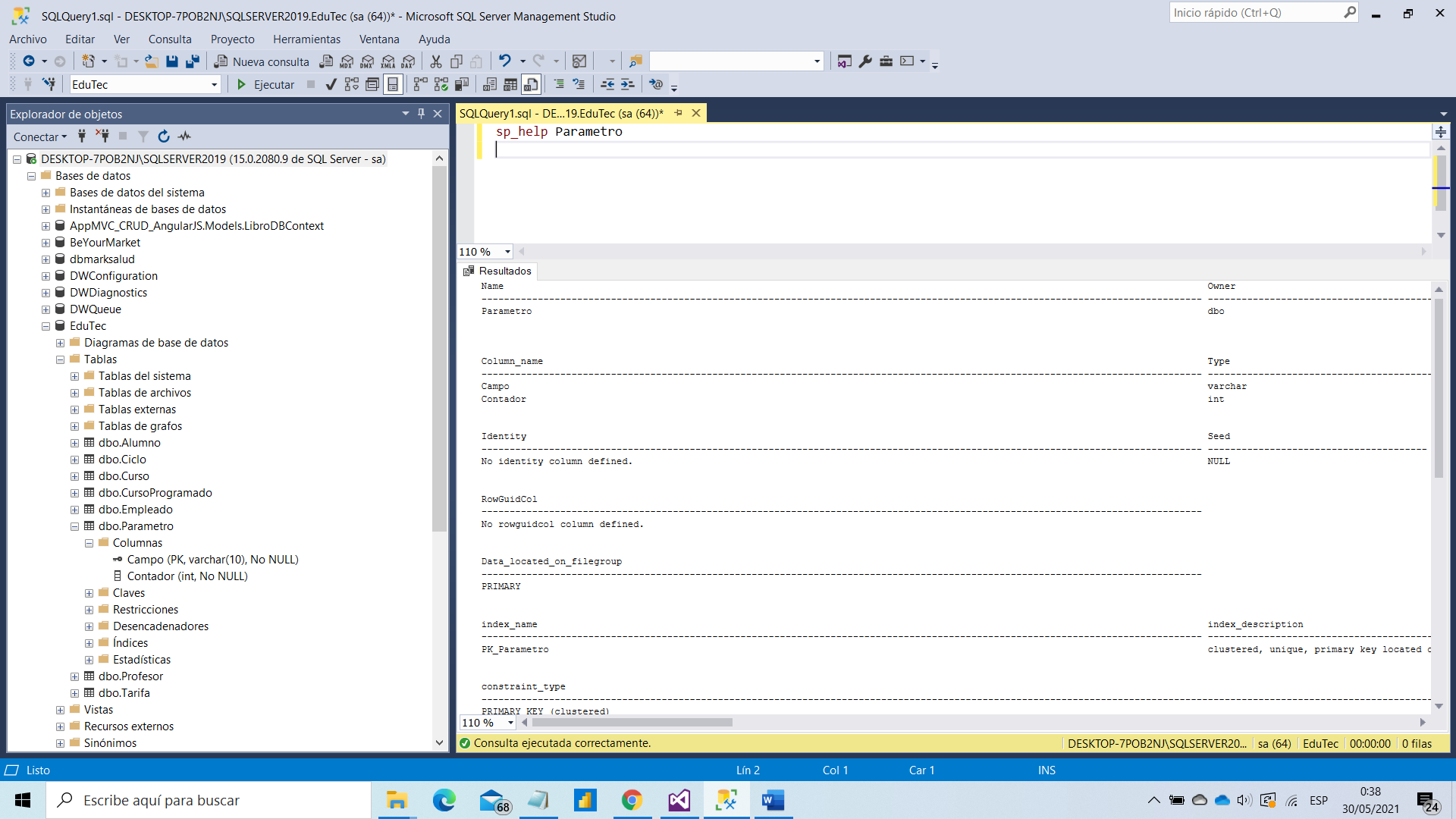
Por qué necesitas agregar eso en el resultado y no lo colocan en el front (o reporte o lo que sea que muestre tus datos)? Lo ideal es transportar el dato “limpio”. No es buena práctica agregar esos ya que convertirías un DECIMAL a un VARCHAR pero si aún deseas hacerlo es FORMAT(<columna>, ‘C’)

# Apéndice

## Entorno SSIS

### Ejecutar

Al momento de ejecutar se puede mostrar los resultados en cuadrículas, textos o en algún archivo plano.



1. <https://stackoverflow.com/questions/45074493/use-to-date-in-sql-server-2012>

   Para ver tipos de fechas en SQL SERVER : https://docs.microsoft.com/en-us/sql/t-sql/functions/cast-and-convert-transact-sql?view=sql-server-ver15 [↑](#footnote-ref-1)