



EBook Gratis

APRENDIZAJE

Microsoft SQL Server

Free unaffiliated eBook created from
Stack Overflow contributors.

#sql-server

Tabla de contenido

Acerca de.....	1
Capítulo 1: Empezando con Microsoft SQL Server.....	2
Observaciones.....	2
Versiones.....	2
Examples.....	2
INSERTAR / SELECCIONAR / ACTUALIZAR / BORRAR: los conceptos básicos del lenguaje de manipu.....	2
Se une.....	4
Alias de tabla.....	5
Uniones.....	5
Variables de tabla.....	6
IMPRESIÓN.....	7
SELECCIONE todas las filas y columnas de una tabla.....	7
Seleccionar filas que coincidan con una condición.....	8
ACTUALIZAR fila específica.....	8
ACTUALIZAR todas las filas.....	8
Comentarios en código.....	9
Recuperar información básica del servidor.....	10
Uso de transacciones para cambiar datos de forma segura.....	10
BORRAR todas las filas.....	11
TABLA DE TRUNCATOS.....	12
Crear nueva tabla e insertar registros de la tabla antigua.....	12
Obtención de recuento de filas de la tabla.....	13
Capítulo 2: Administrar la base de datos SQL de Azure.....	14
Examples.....	14
Encuentre información de nivel de servicio para la base de datos SQL de Azure.....	14
Cambiar el nivel de servicio de la base de datos SQL de Azure.....	14
Replicación de la base de datos SQL de Azure.....	14
Crear base de datos SQL de Azure en el grupo de Elastic.....	15
Capítulo 3: Agente de servicios.....	16
Examples.....	16

1. Fundamentos.....	16
2. Habilitar Service Broker en la base de datos.....	16
3. Crear la construcción del agente de servicios básicos en la base de datos (comunicación.....	16
4. Cómo enviar comunicación básica a través de Service Broker.....	17
5. Cómo recibir la conversación de TargetQueue automáticamente.....	17
Capítulo 4: AGRUPAR POR.....	20
Examples.....	20
Agrupación simple.....	20
GRUPO POR VARIAS COLECCIONES.....	21
Agrupar por con múltiples tablas, múltiples columnas.....	21
TENIENDO.....	23
GROUP BY con ROLLUP y CUBE.....	24
Capítulo 5: Almacenamiento de JSON en tablas SQL.....	26
Examples.....	26
JSON almacenado como columna de texto.....	26
Asegúrese de que JSON esté formateado correctamente utilizando ISJSON.....	26
Exponer valores de texto JSON como columnas calculadas.....	26
Agregando índice en la ruta JSON.....	27
JSON almacenado en tablas en memoria.....	27
Capítulo 6: Analizando una consulta.....	28
Examples.....	28
Escanear vs buscar.....	28
Capítulo 7: aplicación cruzada.....	29
Examples.....	29
Unir filas de tablas con filas generadas dinámicamente desde una celda.....	29
Unir filas de la tabla con la matriz JSON almacenada en la celda.....	29
Filtrar filas por valores de matriz.....	29
Capítulo 8: Base de datos del sistema - TempDb.....	31
Examples.....	31
Identificar el uso de TempDb.....	31
Detalles de la base de datos TempDB.....	31
Capítulo 9: Cifrado.....	32

Parámetros.....	32
Observaciones.....	32
Examples.....	32
Cifrado por certificado.....	32
Cifrado de la base de datos.....	33
Cifrado por clave simétrica.....	33
Cifrado por frase de contraseña.....	33
Capítulo 10: Columnas calculadas.....	34
Examples.....	34
Una columna se calcula a partir de una expresión.....	34
Ejemplo simple que usamos normalmente en tablas de registro.....	34
Capítulo 11: COLUMNSTORE CLUSTERADO.....	36
Examples.....	36
Tabla con índice CLUSTERED COLUMNSTORE.....	36
Agregar índice de almacén de columnas agrupado en la tabla existente.....	36
Reconstruir el índice CLUSTERED COLUMNSTORE.....	36
Capítulo 12: Con la opción de corbatas.....	38
Examples.....	38
Datos de prueba.....	38
Capítulo 13: Conjunto de resultados de límite.....	40
Introducción.....	40
Parámetros.....	40
Observaciones.....	40
Examples.....	40
Limitar con TOP.....	40
Limitar con el POR CIENTO.....	40
Limitar con FETCH.....	40
Capítulo 14: Consejos de consulta.....	42
Examples.....	42
ÚNETE a sugerencias.....	42
GRUPO POR CONSEJOS.....	42
Sugerencia de filas FAST.....	43

Consejos UNION.....	43
Opción MAXDOP.....	43
INDEX Consejos.....	44
Capítulo 15: Consulta de resultados por página.....	45
Examples.....	45
Numero de fila().....	45
Capítulo 16: Consultas con datos JSON.....	46
Examples.....	46
Usando valores de JSON en la consulta.....	46
Usando valores JSON en reportes.....	46
Filtra el texto JSON incorrecto de los resultados de la consulta.....	46
Actualizar el valor en la columna JSON.....	46
Agregar nuevo valor a la matriz JSON.....	47
Mesa JOIN con colección JSON interior.....	47
Encontrar filas que contienen valor en la matriz JSON.....	47
Capítulo 17: Convertir tipos de datos.....	49
Examples.....	49
PRUEBA PARSE.....	49
PRUEBA CONVERTIR.....	49
TRY CAST.....	50
Emitir.....	50
Convertir.....	51
Capítulo 18: Copia de seguridad y restauración de base de datos.....	52
Sintaxis.....	52
Parámetros.....	52
Examples.....	52
Copia de seguridad básica en disco sin opciones.....	52
Restauración básica desde disco sin opciones.....	52
RESTAURAR la base de datos con REEMPLAZAR.....	52
Capítulo 19: CREAR VISTA.....	54
Examples.....	54
CREAR VISTA.....	54

CREAR VISTA con cifrado.....	54
CREE VISTA CON INNER JOIN.....	54
CREAR VISTA indexada.....	55
VISTAS agrupadas.....	56
VISTAS de UNION-ed.....	56
Capítulo 20: Cursores.....	57
Sintaxis.....	57
Observaciones.....	57
Examples.....	57
Cursor de avance solo básico.....	57
Sintaxis del cursor rudimentario.....	58
Capítulo 21: Datos espaciales.....	60
Introducción.....	60
Examples.....	60
PUNTO.....	60
Capítulo 22: DBCC.....	62
Examples.....	62
Comandos de mantenimiento DBCC.....	62
Declaraciones de validación DBCC.....	63
Declaraciones informativas DBCC.....	63
Comandos DBCC Trace.....	63
Declaración DBCC.....	64
Capítulo 23: DBMAIL.....	65
Sintaxis.....	65
Examples.....	65
Enviar email simple.....	65
Enviar los resultados de una consulta.....	65
Enviar correo electrónico HTML.....	65
Capítulo 24: Declaración de caso.....	67
Observaciones.....	67
Examples.....	67
Sentencia CASE simple.....	67

Búsqueda de sentencia CASE.....	67
Capítulo 25: Delimitando caracteres especiales y palabras reservadas.....	68
Observaciones.....	68
Examples.....	68
Método básico.....	68
Capítulo 26: Desencadenar.....	69
Introducción.....	69
Examples.....	69
Tipos y clasificaciones de gatillo.....	69
Activadores de DML.....	69
Capítulo 27: Eliminar palabra clave.....	71
Introducción.....	71
Observaciones.....	71
Examples.....	71
Caer mesas.....	71
Eliminar bases de datos.....	72
Soltar tablas temporales.....	73
Capítulo 28: En memoria OLTP (Hekaton).....	74
Examples.....	74
Crear tabla optimizada de memoria.....	74
Mostrar los archivos .dll y las tablas creadas para las tablas de memoria optimizada.....	75
Tipos de tablas y tablas temporales optimizadas para la memoria.....	75
Declarar variables de tabla optimizadas para la memoria.....	76
Crear una tabla temporal optimizada para la versión del sistema de memoria.....	77
Capítulo 29: Enmascaramiento dinámico de datos.....	78
Examples.....	78
Enmascarar la dirección de correo electrónico utilizando enmascaramiento dinámico.....	78
Añadir máscara parcial en columna.....	78
Mostrando valores aleatorios del rango usando random () mask.....	78
Añadiendo máscara por defecto en la columna.....	79
Controlando quién puede ver los datos desenmascarados.....	79
Capítulo 30: Esquemas.....	80

Examples.....	80
Creando un esquema.....	80
Alter Schema.....	80
Esquemas de caída.....	80
Propósito.....	80
Capítulo 31: Exportar datos en el archivo txt utilizando SQLCMD.....	81
Sintaxis.....	81
Examples.....	81
Mediante el uso de SQLCMD en el símbolo del sistema.....	81
Capítulo 32: Expresiones de mesa comunes.....	82
Sintaxis.....	82
Observaciones.....	82
Examples.....	82
Jerarquía de empleados.....	82
Configuración de la tabla.....	82
Expresión de tabla común.....	82
Salida:.....	83
Encuentra el enésimo salario más alto usando CTE.....	83
Eliminar filas duplicadas utilizando CTE.....	84
Generar una tabla de fechas utilizando CTE.....	85
CTE recursivo.....	85
CTE con múltiples declaraciones AS.....	86
Capítulo 33: fechas.....	88
Sintaxis.....	88
Observaciones.....	88
Examples.....	89
Formato de fecha y hora utilizando CONVERT.....	89
Formato de fecha y hora usando FORMATO.....	90
Obtener el DateTime actual.....	92
DATEADD para sumar y restar períodos de tiempo.....	93
Referencia de piezas de fecha.....	94

DATEDIFF para calcular las diferencias de período de tiempo.....	94
DATEPART & DATENAME.....	95
Consiguiendo el último día de un mes.....	96
Devuelve solo la fecha de un DateTime.....	96
Crear una función para calcular la edad de una persona en una fecha específica.....	96
PLATAFORMA TRANSVERSAL FECHA OBJETO.....	97
Formato de fecha extendido.....	97
Capítulo 34: Filestream.....	106
Introducción.....	106
Examples.....	106
Ejemplo.....	106
Capítulo 35: Función de cadena dividida en el servidor SQL.....	107
Examples.....	107
Dividir una cadena en Sql Server 2016.....	107
Cadena dividida en Sql Server 2008/2012/2014 usando XML.....	108
Variable de tabla T-SQL y XML.....	108
Capítulo 36: Funciones agregadas.....	110
Introducción.....	110
Sintaxis.....	110
Examples.....	110
SUMA().....	110
AVG ().....	110
MAX ().....	111
MIN ().....	111
CONTAR().....	112
COUNT (nombre_columna) con GROUP BY Column_Name.....	112
Capítulo 37: Funciones agregadas de cadenas en SQL Server.....	114
Examples.....	114
Usando STUFF para la agregación de cadenas.....	114
String_Agg for String Aggregation.....	114
Capítulo 38: Funciones de cadena.....	115
Observaciones.....	115

Examples.....	116
Izquierda.....	116
Derecha.....	116
Subcadena.....	117
ASCII.....	117
Índice de caracteres.....	118
Carbonizarse.....	118
Len.....	118
Concat.....	119
Inferior.....	120
Superior.....	120
LTrim.....	121
RTrim.....	121
Unicode.....	121
NChar.....	122
Marcha atrás.....	122
PatIndex.....	122
Espacio.....	123
Reproducir exactamente.....	123
Reemplazar.....	123
String_Split.....	124
Str.....	125
Nombre de lugar.....	125
Soundex.....	126
Diferencia.....	126
Formato.....	127
String_escape.....	129
Capítulo 39: Funciones de clasificación.....	131
Sintaxis.....	131
Parámetros.....	131
Observaciones.....	131
Examples.....	132
RANGO().....	132

DENSE_RANK ().....	132
Capítulo 40: Funciones de ventana.....	133
Examples.....	133
Media móvil centrada.....	133
Encuentre el elemento más reciente en una lista de eventos con marca de tiempo.....	133
Promedio móvil de los últimos 30 artículos.....	133
Capítulo 41: Funciones lógicas.....	135
Examples.....	135
ESCOGER.....	135
IIF.....	135
Capítulo 42: Generando un rango de fechas.....	137
Parámetros.....	137
Observaciones.....	137
Examples.....	137
Generando intervalo de fechas con CTE recursivo.....	137
Generando un intervalo de fechas con una tabla de conteo.....	137
Capítulo 43: Gobernador de recursos.....	139
Observaciones.....	139
Examples.....	139
Leyendo las estadísticas.....	139
Crear un grupo para consultas ad hoc.....	139
Capítulo 44: Grupo de archivos.....	141
Examples.....	141
Crear grupo de archivos en base de datos.....	141
Capítulo 45: Importación a granel.....	143
Examples.....	143
INSERTO A GRANEL con opciones.....	143
INSERTO A GRANEL.....	143
Leyendo todo el contenido del archivo usando OPENROWSET (BULK).....	143
Lea el archivo utilizando OPENROWSET (BULK) y el archivo de formato.....	143
Lee el archivo json usando OPENROWSET (BULK).....	144
Capítulo 46: Índice.....	145

Examples.....	145
Crear índice agrupado.....	145
Crear índice no agrupado.....	145
Mostrar información del índice.....	145
Índice a la vista.....	145
Índice de caída.....	146
Devuelve los índices de tamaño y fragmentación.....	146
Reorganizar y reconstruir índice.....	146
Reconstruir o reorganizar todos los índices en una tabla.....	146
Reconstruir toda la base de datos de índice.....	147
Índice de investigaciones.....	147
Capítulo 47: Indización de texto completo.....	148
Examples.....	148
A. Creación de un índice único, un catálogo de texto completo y un índice de texto complet.....	148
Creación de un índice de texto completo en varias columnas de la tabla.....	148
Creación de un índice de texto completo con una lista de propiedades de búsqueda sin relleno.....	148
Búsqueda de texto completo.....	149
Capítulo 48: Insertar.....	150
Examples.....	150
Agregar una fila a una tabla llamada Facturas.....	150
Capítulo 49: INSERTAR EN.....	151
Introducción.....	151
Examples.....	151
INSERTAR la tabla Hello World INTO.....	151
INSERTAR en columnas específicas.....	151
INSERTAR múltiples filas de datos.....	151
INSERTAR una sola fila de datos.....	152
Use SALIDA para obtener el nuevo ID.....	152
INSERTAR desde SELECCIONE los resultados de la consulta.....	153
Capítulo 50: Instalar SQL Server en Windows.....	154
Examples.....	154
Introducción.....	154

Capítulo 51: Instantáneas de la base de datos	155
Observaciones	155
Examples	155
Crear una instantánea de base de datos	155
Restaurar una instantánea de base de datos	156
BORRAR Instantánea	156
Capítulo 52: Instrucción SELECT	157
Introducción	157
Examples	157
SELECT básico de la tabla	157
Filtrar filas usando la cláusula WHERE	157
Ordenar resultados utilizando ORDENAR POR	157
Resultado de grupo utilizando GROUP BY	157
Filtrar grupos utilizando la cláusula HAVING	158
Devolviendo solo las primeras N filas	158
Paginación utilizando OFFSET FETCH	158
SELECT sin FROM (sin fuente de datos)	159
Capítulo 53: Integración de Common Language Runtime	160
Examples	160
Habilitar CLR en la base de datos	160
Adición de .dll que contiene módulos Sql CLR	160
Crear función CLR en SQL Server	161
Crear CLR tipo definido por el usuario en SQL Server	161
Crear procedimiento CLR en SQL Server	161
Capítulo 54: JSON en Sql Server	163
Sintaxis	163
Parámetros	163
Observaciones	163
Examples	163
Formato de resultados de consultas como JSON con FOR JSON	163
Analizar texto JSON	164
Únase a las entidades JSON principales y secundarias utilizando CROSS APPLY OPENJSON	164

Índice en las propiedades JSON mediante el uso de columnas calculadas.....	165
Formatee una fila de la tabla como un solo objeto JSON usando FOR JSON.....	166
Analizar texto JSON utilizando la función OPENJSON.....	167
Capítulo 55: JUNTARSE.....	168
Sintaxis.....	168
Examples.....	168
Usando COALESCE para construir una cadena delimitada por comas.....	168
Ejemplo básico de coalesce.....	168
Obtener el primero no nulo de una lista de valores de columna.....	169
Capítulo 56: La función COSAS.....	170
Parámetros.....	170
Examples.....	170
Reemplazo de personaje básico con STUFF ().....	170
Uso de FOR XML para concatenar valores de varias filas.....	170
Obtener nombres de columna separados por comas (no una lista).....	171
Cosas para comas separadas en servidor sql.....	171
Ejemplo básico de la función STUFF ().....	172
Capítulo 57: Lectura fantasma.....	173
Introducción.....	173
Observaciones.....	173
Examples.....	173
Nivel de aislamiento LEER SIN COMPROMISO.....	173
Capítulo 58: Llaves extranjeras.....	175
Examples.....	175
Relación / restricción de clave externa.....	175
Mantener la relación entre las filas padre / hijo.....	175
Agregar una relación de clave externa en la tabla existente.....	176
Añadir clave externa en la tabla existente.....	176
Obtener información sobre restricciones de clave externa.....	176
Capítulo 59: Llaves primarias.....	177
Observaciones.....	177
Examples.....	177

Crear tabla con columna de identidad como clave principal.....	177
Crear tabla con clave primaria GUID.....	177
Crear mesa con llave natural.....	177
Crear tabla w / clave compuesta.....	177
Añadir clave principal a la tabla existente.....	178
Eliminar clave principal.....	178
Capítulo 60: Manejo de transacciones.....	179
Parámetros.....	179
Examples.....	179
Esqueleto básico de transacciones con manejo de errores.....	179
Capítulo 61: Mientras bucle.....	180
Observaciones.....	180
Examples.....	180
Usando While loop.....	180
Mientras que el bucle con el uso mínimo de la función agregada.....	180
Capítulo 62: Migración.....	181
Examples.....	181
Cómo generar scripts de migración.....	181
Capítulo 63: Modificar texto JSON.....	183
Examples.....	183
Modificar valor en texto JSON en la ruta especificada.....	183
Agregar un valor escalar en una matriz JSON.....	183
Insertar nuevo objeto JSON en texto JSON.....	183
Insertar nueva matriz JSON generada con la consulta FOR JSON.....	184
Insertar un solo objeto JSON generado con la cláusula FOR JSON.....	184
Capítulo 64: Módulos compilados de forma nativa (Hekaton).....	186
Examples.....	186
Procedimiento almacenado nativamente compilado.....	186
Función escalar compilada de forma nativa.....	186
Función de valor de tabla en línea nativa.....	187
Capítulo 65: Mueve y copia datos alrededor de tablas.....	189
Examples.....	189

Copia datos de una tabla a otra.....	189
Copia datos en una tabla, creando esa tabla sobre la marcha.....	189
Mover datos a una tabla (asumiendo el método de claves únicas).....	189
Capítulo 66: Niveles de aislamiento de transacciones.....	191
Sintaxis.....	191
Observaciones.....	191
Examples.....	191
Leer no comprometido.....	191
Leer comprometido.....	191
¿Qué son las "lecturas sucias"?.....	192
Lectura repetible.....	192
Instantánea.....	193
Serializable.....	193
Capítulo 67: Niveles de aislamiento y bloqueo.....	195
Observaciones.....	195
Examples.....	195
Ejemplos de ajuste del nivel de aislamiento.....	195
Capítulo 68: Nombres de alias en el servidor SQL.....	197
Introducción.....	197
Examples.....	197
Usando AS.....	197
Usando =.....	197
Dar alias después del nombre de la tabla Derivada.....	197
Sin usar AS.....	198
Capítulo 69: NULLs.....	199
Introducción.....	199
Observaciones.....	199
Examples.....	199
Comparación nula.....	199
ANSI NULLS.....	200
ES NULO().....	201
Es nulo / no es nulo.....	201

COALESCE ()	202
NULL con NOT IN SubQuery	202
Capítulo 70: Opciones avanzadas	203
Examples	203
Habilitar y mostrar opciones avanzadas	203
Habilitar compresión de respaldo por defecto	203
Establecer el porcentaje de relleno predeterminado	203
Establecer intervalo de recuperación del sistema	203
Habilitar permiso cmd	203
Establecer el tamaño máximo de memoria del servidor	203
Establecer el número de tareas de punto de control	204
Capítulo 71: OPENJSON	205
Examples	205
Obtener clave: pares de valores de texto JSON	205
Transformar la matriz JSON en un conjunto de filas	205
Transformar los campos JSON anidados en un conjunto de filas	206
Extracción de subobjetos JSON internos	206
Trabajar con subreglas JSON anidadas	207
Capítulo 72: Operaciones básicas de DDL en MS SQL Server	209
Examples	209
Empezando	209
Crear base de datos	209
Crear mesa	209
Crear vista	210
Crear procedimiento	210
Capítulo 73: ORDEN POR	212
Observaciones	212
Examples	212
Cláusula ORDER BY simple	212
ORDENAR por campos múltiples	212
ORDENAR con lógica compleja	213
Pedidos personalizados	213

Capítulo 74: Ordenar / ordenar filas	215
Examples	215
Lo esencial	215
Orden por caso	217
Capítulo 75: Paginación	219
Introducción	219
Sintaxis	219
Examples	219
Paginación usando ROW_NUMBER con una expresión de tabla común	219
Paginación con OFFSET FETCH	220
Paginaton con consulta interna	220
Paginación en varias versiones de SQL Server	220
SQL Server 2012/2014	220
SQL Server 2005/2008 / R2	221
SQL Server 2000	221
SQL Server 2012/2014 utilizando ORDER BY OFFSET y FETCH NEXT	221
Capítulo 76: PARA JSON	222
Examples	222
PARA JSON PATH	222
FOR JSON PATH con alias de columna	222
Cláusula FOR JSON sin contenedor de matriz (objeto único en la salida)	222
INCLUYE_NULL_VALUES	223
Envolviendo resultados con objeto ROOT	223
Para json auto	223
Creando estructura JSON anidada personalizada	224
Capítulo 77: PARA XML PATH	225
Observaciones	225
Examples	225
Hola mundo XML	225
Especificando espacios de nombres	225
Especificando la estructura usando expresiones XPath	225
Usando FOR XML PATH para concatenar valores	227

Capítulo 78: Parámetros de tabla de valores	228
Observaciones.....	228
Examples.....	228
Uso de un parámetro con valores de tabla para insertar varias filas en una tabla.....	228
Capítulo 79: Parsename	229
Sintaxis.....	229
Parámetros.....	229
Examples.....	229
Nombre de pila.....	229
Capítulo 80: Particionamiento	230
Examples.....	230
Recuperar valores de límite de partición.....	230
Cambio de particiones.....	230
Recupere los valores de la tabla de partición, columna, esquema, función, total y mínimo-m.....	230
Capítulo 81: Permisos de base de datos	232
Observaciones.....	232
Examples.....	232
Cambio de permisos.....	232
CREAR USUARIO.....	232
Crear un rol.....	232
Cambio de membresía de rol.....	233
Capítulo 82: Permisos y seguridad	234
Examples.....	234
Asignar permisos de objeto a un usuario.....	234
Capítulo 83: Pivot dinámico de SQL	235
Introducción.....	235
Examples.....	235
Pivote de SQL dinámico básico.....	235
Capítulo 84: PIVOTE / UNPIVOT	237
Sintaxis.....	237
Observaciones.....	237

Examples.....	237
Pivote simple - columnas estáticas.....	237
Simple PIVOT & UNPIVOT (T-SQL).....	238
PIVOTE Dinámico.....	240
Capítulo 85: Privilegios o permisos.....	242
Examples.....	242
Reglas simples.....	242
Capítulo 86: Procedimientos almacenados.....	243
Introducción.....	243
Sintaxis.....	243
Examples.....	243
Creación y ejecución de un procedimiento almacenado básico.....	243
PROCEDIMIENTO ALMACENADO con parámetros OUT.....	245
Creación de un procedimiento almacenado con un único parámetro de salida.....	245
Ejecutando el procedimiento almacenado.....	245
Creación de un procedimiento almacenado con múltiples parámetros de salida.....	245
Ejecutando el procedimiento almacenado.....	246
Procedimiento almacenado con If ... Else e Insertar en operación.....	246
SQL dinámico en procedimiento almacenado.....	247
Bucle simple.....	248
Bucle simple.....	249
Capítulo 87: Puntos de vista.....	250
Observaciones.....	250
Examples.....	250
Crear una vista.....	250
Crear o reemplazar vista.....	250
Crear una vista con enlace de esquema.....	250
Capítulo 88: Recupera información sobre tu instancia.....	252
Examples.....	252
Recuperar servidores locales y remotos.....	252
Obtenga información sobre las sesiones actuales y ejecuciones de consulta.....	252

Recuperar Edición y Versión de Instancia.....	253
Recuperar el tiempo de actividad de la instancia en días.....	253
Información sobre la versión de SQL Server.....	253
Información general sobre bases de datos, tablas, procedimientos almacenados y cómo buscar.....	253
Capítulo 89: Recuperar información sobre la base de datos.....	255
Observaciones.....	255
Examples.....	255
Cuenta el número de tablas en una base de datos.....	255
Recuperar una lista de todos los procedimientos almacenados.....	255
Obtener la lista de todas las bases de datos en un servidor.....	256
Archivos de base de datos.....	257
Recuperar opciones de base de datos.....	257
Mostrar tamaño de todas las tablas en la base de datos actual.....	257
Determine una ruta de acceso de inicio de sesión de Windows.....	258
Recuperar tablas que contienen una columna conocida.....	258
Ver si se están utilizando características específicas de la empresa.....	258
Buscar y devolver todas las tablas y columnas que contengan un valor de columna específica.....	258
Consigue todos los esquemas, tablas, columnas e índices.....	260
Devuelva una lista de trabajos del Agente SQL, con información de programación.....	260
Recuperar información sobre operaciones de copia de seguridad y restauración.....	263
Encuentra cada mención de un campo en la base de datos.....	263
Capítulo 90: SCOPE_IDENTITY ().....	265
Sintaxis.....	265
Examples.....	265
Introducción con ejemplo simple.....	265
Capítulo 91: Secuencias.....	266
Examples.....	266
Crear secuencia.....	266
Usa la secuencia en la tabla.....	266
Insertar en la mesa con secuencia.....	266
Eliminar de e insertar nuevo.....	266
Capítulo 92: Seguridad a nivel de fila.....	268

Examples.....	268
Predicado del filtro RLS.....	268
Alterar la política de seguridad RLS.....	269
Previendo actualización usando predicado de bloque RLS.....	269
Capítulo 93: Si ... otra cosa.....	271
Examples.....	271
Declaración única de FI.....	271
Múltiples declaraciones de IF.....	271
Una sola declaración IF..ELSE.....	271
Múltiples IF ... ELSE con declaraciones ELSE finales.....	272
Múltiples declaraciones de IF ... ELSE.....	272
Capítulo 94: Sobre la cláusula.....	274
Parámetros.....	274
Observaciones.....	274
Examples.....	274
Usando las funciones de agregación con OVER.....	274
Suma acumulativa.....	275
Usando funciones de agregación para encontrar los registros más recientes.....	275
Dividir datos en cubos igualmente particionados usando NTILE.....	276
Capítulo 95: SQL dinámico.....	278
Examples.....	278
Ejecutar la instrucción SQL proporcionada como una cadena.....	278
SQL dinámico ejecutado como usuario diferente.....	278
Inyección de SQL con SQL dinámico.....	278
SQL dinámico con parámetros.....	279
Capítulo 96: SQL Server Evolution a través de diferentes versiones (2000 - 2016).....	280
Introducción.....	280
Examples.....	280
SQL Server versión 2000 - 2016.....	280
Capítulo 97: SQL Server Management Studio (SSMS).....	284
Introducción.....	284
Examples.....	284

Actualizar el caché de IntelliSense.....	284
Capítulo 98: SQLCMD.....	285
Observaciones.....	285
Examples.....	285
SQLCMD.exe llamado desde un archivo por lotes o línea de comandos.....	285
Capítulo 99: Subconsultas.....	286
Examples.....	286
Subconsultas.....	286
Capítulo 100: Tablas temporales.....	289
Observaciones.....	289
Examples.....	289
Crear tablas temporales.....	289
¿Cómo puedo consultar los datos temporales?.....	290
Devuelve el valor real especificado en el tiempo (FOR SYSTEM_TIME AS OF).....	290
PARA SYSTEM_TIME ENTRE Y.....	290
PARA SYSTEM_TIME FROM A.....	290
PARA SYSTEM_TIME CONTENIDO EN (,).....	291
PARA SYSTEM_TIME ALL.....	291
Creación de una tabla temporal con versión de sistema optimizada para la memoria y limpieza.....	291
Capítulo 101: Tarea o trabajo programado.....	294
Introducción.....	294
Examples.....	294
Crear un trabajo programado.....	294
Capítulo 102: Teclas de acceso directo de Microsoft SQL Server Management Studio.....	296
Examples.....	296
Ejemplos de atajos.....	296
Accesos directos del teclado de activación del menú.....	296
Atajos de teclado personalizados.....	296
Capítulo 103: Tienda de consultas.....	299
Examples.....	299
Habilitar el almacén de consultas en la base de datos.....	299
Obtenga estadísticas de ejecución para consultas / planes SQL.....	299

Eliminar datos del almacén de consultas.....	299
Forzar plan de consulta.....	300
Capítulo 104: Tipos de datos.....	301
Introducción.....	301
Examples.....	301
Números exactos.....	301
Números aproximados.....	303
Fecha y hora.....	303
Cadenas de caracteres.....	304
Cadenas de caracteres Unicode.....	304
Cuerdas binarias.....	304
Otros tipos de datos.....	304
Capítulo 105: Tipos de tablas definidas por el usuario.....	305
Introducción.....	305
Observaciones.....	305
Examples.....	305
creando un UDT con una sola columna int que también es una clave principal.....	305
Creando un UDT con múltiples columnas.....	305
Creando un UDT con una restricción única:.....	305
Creación de un UDT con una clave principal y una columna con un valor predeterminado:.....	306
Capítulo 106: TRATA DE ATRAPARLO.....	307
Observaciones.....	307
Examples.....	307
Transacción en un TRY / CATCH.....	307
Aumento de errores en el bloque try-catch.....	307
Generar mensajes de información en el bloque try catch.....	308
Excepción de relanzamiento generada por RAISERROR.....	308
Excepción de lanzamiento en bloques TRY / CATCH.....	309
Capítulo 107: Última identidad insertada.....	310
Examples.....	310
SCOPE_IDENTITY ().....	310
@@IDENTIDAD.....	310

IDENT_CURRENT ('nombre de tabla').....	311
@@ IDENTIDAD y MAX (ID).....	311
Capítulo 108: UNIÓN	312
Examples.....	312
Unión y unión todos.....	312
Capítulo 109: UNIR	315
Introducción.....	315
Sintaxis.....	315
Observaciones.....	316
Examples.....	316
MERGE para insertar / actualizar / eliminar.....	316
Combinar usando la fuente CTE.....	317
MERGE usando la tabla de fuente derivada.....	317
Ejemplo de combinación: sincronizar origen y tabla de destino.....	317
Fusionar utilizando EXCEPTO.....	319
Capítulo 110: Unirse	320
Introducción.....	320
Examples.....	320
Unir internamente.....	320
Cruzar.....	321
Unión externa.....	322
Uso de unirse en una actualización.....	324
Únete a una subconsulta.....	325
Auto unirse.....	326
Eliminar usando Join.....	326
Accidentalmente convirtiendo una unión externa en una unión interna.....	327
Capítulo 111: Uso de la tabla TEMP	329
Observaciones.....	329
Examples.....	329
Tabla de temperatura local.....	329
Tabla de temperatura global.....	329
Tablas de temperatura de caída.....	330

Capítulo 112: Utilidad bcp (programa de copia masiva)	331
Introducción.....	331
Examples.....	331
Ejemplo para importar datos sin un archivo de formato (usando formato nativo).....	331
Capítulo 113: Variables	332
Sintaxis.....	332
Examples.....	332
Declarar una variable de tabla.....	332
Actualizando una variable usando SET.....	333
Actualizando variables usando SELECT.....	333
Declara múltiples variables a la vez, con valores iniciales.....	334
Operadores de asignación de compuestos.....	334
Actualizando variables seleccionando desde una tabla.....	334
Creditos	336

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [microsoft-sql-server](#)

It is an unofficial and free Microsoft SQL Server ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Microsoft SQL Server.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con Microsoft SQL Server

Observaciones

Este es un conjunto de ejemplos que destacan el uso básico de SQL Server.

Versiones

Versión	Fecha de lanzamiento
SQL Server 2016	2016-06-01
SQL Server 2014	2014-03-18
SQL Server 2012	2011-10-11
SQL Server 2008 R2	2010-04-01
SQL Server 2008	2008-08-06
SQL Server 2005	2005-11-01
SQL Server 2000	2000-11-01

Examples

INSERTAR / SELECCIONAR / ACTUALIZAR / BORRAR: los conceptos básicos del lenguaje de manipulación de datos

La función de lenguaje de L a D M aip (DML para abreviar) incluye operaciones como `INSERT` , `UPDATE` y `DELETE` :

```
-- Create a table HelloWorld

CREATE TABLE HelloWorld (
    Id INT IDENTITY,
    Description VARCHAR(1000)
)

-- DML Operation INSERT, inserting a row into the table
INSERT INTO HelloWorld (Description) VALUES ('Hello World')

-- DML Operation SELECT, displaying the table
```

```

SELECT * FROM HelloWorld

-- Select a specific column from table
SELECT Description FROM HelloWorld

-- Display number of records in the table
SELECT Count(*) FROM HelloWorld

-- DML Operation UPDATE, updating a specific row in the table
UPDATE HelloWorld SET Description = 'Hello, World!' WHERE Id = 1

-- Selecting rows from the table (see how the Description has changed after the update?)
SELECT * FROM HelloWorld

-- DML Operation - DELETE, deleting a row from the table
DELETE FROM HelloWorld WHERE Id = 1

-- Selecting the table. See table content after DELETE operation
SELECT * FROM HelloWorld

```

En este script estamos **creando una tabla** para demostrar algunas consultas básicas.

Los siguientes ejemplos muestran cómo **consultar las tablas**:

```

USE Northwind;
GO
SELECT TOP 10 * FROM Customers
ORDER BY CompanyName

```

seleccionará los primeros 10 registros de la tabla de `Customer`, ordenados por la columna `CompanyName` de la base de datos `Northwind` (que es una de las bases de datos de muestra de Microsoft, se puede descargar desde [aquí](#)):

	CustomerID	CompanyName	ContactName	ContactTitle	Address	City
▶	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin
	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.
	ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.
	AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London
	BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå
	BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim
	BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg
	BOLID	Bóldo Comidas preparadas	Martín Sommer	Owner	C/ Araquil, 67	Madrid
	BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille
	BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen

Tenga en cuenta que `Use Northwind;` cambia la base de datos predeterminada para todas las consultas posteriores. Aún puede hacer referencia a la base de datos utilizando la sintaxis completa en forma de `[Base de datos]. [Esquema]. [Tabla]`:

```
SELECT TOP 10 * FROM Northwind.dbo.Customers
ORDER BY CompanyName

SELECT TOP 10 * FROM Pubs.dbo.Authors
ORDER BY City
```

Esto es útil si está consultando datos de diferentes bases de datos. Tenga en cuenta que `dbo`, especificado "en medio" se denomina esquema y debe especificarse mientras se usa la sintaxis completamente calificada. Puedes considerarlo como una carpeta dentro de tu base de datos. `dbo` es el esquema predeterminado. El esquema predeterminado puede omitirse. Todos los otros esquemas definidos por el usuario deben ser especificados.

Si la tabla de la base de datos contiene columnas que se denominan como palabras reservadas, por ejemplo, `Date`, debe incluir el nombre de la columna entre paréntesis, como este:

```
-- descending order
SELECT TOP 10 [Date] FROM dbo.MyLogTable
ORDER BY [Date] DESC
```

Lo mismo se aplica si el nombre de la columna contiene espacios en su nombre (lo cual no se recomienda). Una sintaxis alternativa es utilizar comillas dobles en lugar de corchetes, por ejemplo:

```
-- descending order
SELECT top 10 "Date" from dbo.MyLogTable
order by "Date" desc
```

Es equivalente pero no tan comúnmente usado. Observe la diferencia entre comillas dobles y comillas simples: las comillas simples se usan para cadenas, es decir,

```
-- descending order
SELECT top 10 "Date" from dbo.MyLogTable
where UserId='johndoe'
order by "Date" desc
```

Es una sintaxis válida. Tenga en cuenta que T-SQL tiene un prefijo `N` para los tipos de datos `NChar` y `NVarChar`, por ejemplo,

```
SELECT TOP 10 * FROM Northwind.dbo.Customers
WHERE CompanyName LIKE N'AL%'
ORDER BY CompanyName
```

devuelve todas las compañías que tienen un nombre de compañía que comienza con `AL` (`%` es un comodín, úselo como usaría el asterisco en una línea de comando de DOS, por ejemplo, `DIR AL*`). Para `LIKE`, hay un par de comodines disponibles, mira [aquí](#) para conocer más detalles.

Se une

Las combinaciones son útiles si desea consultar campos que no existen en una sola tabla, sino

en varias tablas. Por ejemplo: desea consultar todas las columnas de la tabla de `Region` en la base de datos de `Northwind` . Pero observa que también necesita la `RegionDescription` , que se almacena en una tabla diferente, `Region` . Sin embargo, hay una clave común, `RegionID` que puede usar para combinar esta información en una sola consulta de la siguiente manera (las `Top 5` solo devuelven las primeras 5 filas, omítala para obtener todas las filas):

```
SELECT TOP 5 Territories.*,
           Regions.RegionDescription
FROM Territories
INNER JOIN Region
    ON Territories.RegionID=Region.RegionID
ORDER BY TerritoryDescription
```

mostrará todas las columnas de `Territories` más la columna `Region RegionDescription` de `Region` . El resultado está ordenado por `TerritoryDescription` .

Alias de tabla

Cuando su consulta requiere una referencia a dos o más tablas, puede que le resulte útil usar un Alias de tabla. Los alias de tabla son referencias abreviadas de tablas que se pueden usar en lugar de un nombre completo de tabla y pueden reducir la escritura y la edición. La sintaxis para usar un alias es:

```
<TableName> [as] <alias>
```

Donde `as` es una palabra clave opcional. Por ejemplo, la consulta anterior se puede reescribir como:

```
SELECT TOP 5 t.*,
           r.RegionDescription
FROM Territories t
INNER JOIN Region r
    ON t.RegionID = r.RegionID
ORDER BY TerritoryDescription
```

Los alias deben ser únicos para todas las tablas en una consulta, incluso si usa la misma tabla dos veces. Por ejemplo, si su tabla de Empleado incluyó un campo `SupervisorId`, puede usar esta consulta para devolver el nombre de un empleado y su supervisor:

```
SELECT e.*,
       s.Name as SupervisorName -- Rename the field for output
FROM Employee e
INNER JOIN Employee s
    ON e.SupervisorId = s.EmployeeId
WHERE e.EmployeeId = 111
```

Uniones

Como hemos visto antes, una unión agrega columnas de diferentes orígenes de tabla. ¿Pero qué

pasa si quieres combinar filas de diferentes fuentes? En este caso puedes usar un UNION. Supongamos que está planeando una fiesta y desea invitar no solo a los empleados, sino también a los clientes. Entonces podrías ejecutar esta consulta para hacerlo:

```
SELECT FirstName+' '+LastName as ContactName, Address, City FROM Employees
UNION
SELECT ContactName, Address, City FROM Customers
```

Devolverá nombres, direcciones y ciudades de los empleados y clientes en una sola tabla. Tenga en cuenta que las filas duplicadas (si las hubiera) se eliminan automáticamente (si no desea esto, use `UNION ALL` lugar). El número de columna, los nombres de columna, el orden y el tipo de datos deben coincidir en todas las declaraciones de selección que forman parte de la unión: esta es la razón por la que `SELECT` combina el `FirstName` y el `LastName` del Empleado en el `FirstName` del `ContactName`.

Variables de tabla

Puede ser útil, si necesita lidiar con datos temporales (especialmente en un procedimiento almacenado), para usar las variables de la tabla: la diferencia entre una tabla "real" y una variable de la tabla es que solo existe en la memoria para el procesamiento temporal.

Ejemplo:

```
DECLARE @Region TABLE
(
    RegionID int,
    RegionDescription NChar(50)
)
```

crea una tabla en la memoria. En este caso, el prefijo `@` es obligatorio porque es una variable. Puede realizar todas las operaciones DML mencionadas anteriormente para insertar, eliminar y seleccionar filas, por ejemplo,

```
INSERT INTO @Region values(3, 'Northern')
INSERT INTO @Region values(4, 'Southern')
```

Pero normalmente, lo rellenarías en base a una tabla real como

```
INSERT INTO @Region
SELECT * FROM dbo.Region WHERE RegionID>2;
```

que leería los valores filtrados de la tabla real `dbo.Region` e insertarlo en la tabla de memoria `@Region`, donde se puede utilizar para un procesamiento posterior. Por ejemplo, podrías usarlo en una unión como

```
SELECT * FROM Territories t
JOIN @Region r on t.RegionID=r.RegionID
```


Lo que en este caso devolvería todos `Southern` territorios del `Northern` y `Southern` . Más información detallada se puede encontrar [aquí](#) . Las tablas temporales se discuten [aquí](#) , si está interesado en leer más sobre ese tema.

NOTA: Microsoft solo recomienda el uso de variables de tabla si el número de filas de datos en la variable de tabla es inferior a 100. Si trabajará con grandes cantidades de datos, use una **tabla temporal** o tabla temporal, en su lugar.

IMPRESIÓN

Muestra un mensaje a la consola de salida. Al usar SQL Server Management Studio, esto se mostrará en la pestaña de mensajes, en lugar de la pestaña de resultados:

```
PRINT 'Hello World!';
```

SELECCIONE todas las filas y columnas de una tabla

Sintaxis:

```
SELECT *
FROM table_name
```

El uso del operador de asterisco `*` sirve como acceso directo para seleccionar todas las columnas de la tabla. Todas las filas también se seleccionarán porque esta instrucción `SELECT` no tiene una cláusula `WHERE` , para especificar ningún criterio de filtrado.

Esto también funcionaría de la misma manera si añadiera un alias a la tabla, por ejemplo `e` en este caso:

```
SELECT *
FROM Employees AS e
```

O si desea seleccionar todo de una tabla específica, puede usar el alias `.*`:

```
SELECT e.*, d.DepartmentName
FROM Employees AS e
INNER JOIN Department AS d
ON e.DepartmentID = d.DepartmentID
```

También se puede acceder a los objetos de la base de datos utilizando nombres completos:

```
SELECT * FROM [server_name].[database_name].[schema_name].[table_name]
```

Esto no se recomienda necesariamente, ya que cambiar los nombres del servidor y / o la base de datos causaría que las consultas que usan nombres completamente calificados ya no se ejecuten debido a nombres de objetos no válidos.

Tenga en cuenta que los campos antes de `table_name` se pueden omitir en muchos casos si las

consultas se ejecutan en un único servidor, base de datos y esquema, respectivamente. Sin embargo, es común que una base de datos tenga múltiples esquemas y, en estos casos, el nombre del esquema no debe omitirse cuando sea posible.

Advertencia: el uso de `SELECT *` en el código de producción o en los procedimientos almacenados puede llevar a problemas más adelante (a medida que se agregan nuevas columnas a la tabla, o si las columnas se reorganizan en la tabla), especialmente si su código hace suposiciones simples sobre el orden de las columnas, o el número de columnas devueltas. Por lo tanto, siempre es más seguro especificar explícitamente los nombres de columna en las instrucciones `SELECT` para el código de producción.

```
SELECT col1, col2, col3
FROM table_name
```

Seleccionar filas que coincidan con una condición

En general, la sintaxis es:

```
SELECT <column names>
FROM <table name>
WHERE <condition>
```

Por ejemplo:

```
SELECT FirstName, Age
FROM Users
WHERE LastName = 'Smith'
```

Las condiciones pueden ser complejas:

```
SELECT FirstName, Age
FROM Users
WHERE LastName = 'Smith' AND (City = 'New York' OR City = 'Los Angeles')
```

ACTUALIZAR fila específica

```
UPDATE HelloWorlds
SET HelloWorld = 'HELLO WORLD!!!'
WHERE Id = 5
```

El código anterior actualiza el valor del campo "HelloWorld" con "HELLO WORLD !!!" para el registro donde "Id = 5" en la tabla HelloWorlds.

Nota: En una declaración de actualización, se recomienda usar una cláusula "donde" para evitar actualizar la tabla completa a menos que y hasta que su requerimiento sea diferente.

ACTUALIZAR todas las filas

Una forma simple de actualización es incrementar todos los valores en un campo dado de la tabla. Para hacerlo, necesitamos definir el campo y el valor de incremento

El siguiente es un ejemplo que incrementa el campo `score` en 1 (en todas las filas):

```
UPDATE Scores
SET score = score + 1
```

Esto puede ser peligroso ya que puede corromper sus datos si accidentalmente realiza una ACTUALIZACIÓN para una **fila específica** con una ACTUALIZACIÓN para **todas las filas** en la tabla.

Comentarios en código

Transact-SQL admite dos formas de escribir comentarios. Los comentarios son ignorados por el motor de la base de datos y están destinados a que las personas los lean.

Los comentarios están precedidos por `--` y se ignoran hasta que se encuentra una nueva línea:

```
-- This is a comment
SELECT *
FROM MyTable -- This is another comment
WHERE Id = 1;
```

Los comentarios de Slash Star comienzan con `/*` y terminan con `*/`. Todo el texto entre esos delimitadores se considera un bloque de comentarios.

```
/* This is
a multi-line
comment block. */
SELECT Id = 1, [Message] = 'First row'
UNION ALL
SELECT 2, 'Second row'
/* This is a one liner */
SELECT 'More';
```

Los comentarios de Slash Star tienen la ventaja de mantener el comentario utilizable si la declaración SQL pierde nuevos caracteres de línea. Esto puede suceder cuando se captura SQL durante la resolución de problemas.

Los comentarios de barra diagonal pueden anidarse y un inicio `/*` dentro de un comentario de barra diagonal debe finalizar con una `*/` para que sea válido. El siguiente código dará como resultado un error.

```
/*
SELECT *
FROM CommentTable
WHERE Comment = '/*'
*/
```

La barra diagonal, aunque esté dentro de la cita, se considera como el inicio de un comentario.

Por lo tanto, debe ser terminado con otra barra de la estrella de cierre. La forma correcta sería

```
/*
SELECT *
FROM CommentTable
WHERE Comment = '/*'
*/ */
```

Recuperar información básica del servidor

```
SELECT @@VERSION
```

Devuelve la versión de MS SQL Server que se ejecuta en la instancia.

```
SELECT @@SERVERNAME
```

Devuelve el nombre de la instancia de MS SQL Server.

```
SELECT @@SERVICENAME
```

Devuelve el nombre del servicio de Windows con el que se ejecuta MS SQL Server.

```
SELECT serverproperty('ComputerNamePhysicalNetBIOS');
```

Devuelve el nombre físico de la máquina donde se ejecuta SQL Server. Útil para identificar el nodo en un clúster de conmutación por error.

```
SELECT * FROM fn_virtualservernodes();
```

En un clúster de conmutación por error, se devuelven todos los nodos donde se puede ejecutar SQL Server. No devuelve nada si no es un cluster.

Uso de transacciones para cambiar datos de forma segura

Siempre que cambie los datos, en un comando del Lenguaje de manipulación de datos (DML), puede ajustar sus cambios en una transacción. DML incluye `UPDATE` , `TRUNCATE` , `INSERT` y `DELETE` . Una de las maneras en que puede asegurarse de que está cambiando los datos correctos sería utilizar una transacción.

Los cambios en DML llevarán un bloqueo en las filas afectadas. Cuando comienza una transacción, debe finalizar la transacción o todos los objetos que se modifican en el DML permanecerán bloqueados por quien haya iniciado la transacción. Puede finalizar su transacción con `ROLLBACK` o `COMMIT` . `ROLLBACK` devuelve todo dentro de la transacción a su estado original. `COMMIT` coloca los datos en un estado final en el que no puede deshacer sus cambios sin otra declaración DML.

Ejemplo:

```

--Create a test table

USE [your database]
GO
CREATE TABLE test_transaction (column_1 varchar(10))
GO

INSERT INTO
    dbo.test_transaction
        ( column_1 )
VALUES
    ( 'a' )

BEGIN TRANSACTION --This is the beginning of your transaction

UPDATE dbo.test_transaction
SET column_1 = 'B'
OUTPUT INSERTED.*
WHERE column_1 = 'A'

ROLLBACK TRANSACTION --Rollback will undo your changes
--Alternatively, use COMMIT to save your results

SELECT * FROM dbo.test_transaction --View the table after your changes have been run

DROP TABLE dbo.test_transaction

```

Notas:

- Este es un **ejemplo simplificado** que no incluye el manejo de errores. Pero cualquier operación de base de datos puede fallar y por lo tanto lanzar una excepción. [Aquí hay un ejemplo de](#) cómo podría ser un manejo de errores tan requerido. **Nunca se** debe utilizar transacciones **sin un controlador de errores**, de lo contrario podría salir de la transacción en un estado desconocido.
- Según el [nivel de aislamiento](#) , las transacciones ponen bloqueos en los datos que se consultan o modifican. Debe asegurarse de que las transacciones no se estén ejecutando durante mucho tiempo, ya que bloquearán los registros en una base de datos y pueden provocar [interbloqueos](#) con otras transacciones paralelas. Mantenga las operaciones encapsuladas en las transacciones lo más cortas posible y minimice el impacto con la cantidad de datos que está bloqueando.

BORRAR todas las filas

```

DELETE
FROM HelloWorlds

```

Esto borrará todos los datos de la tabla. La tabla no contendrá filas después de ejecutar este código. A diferencia de `DROP TABLE` , esto preserva la tabla y su estructura, y puede continuar insertando nuevas filas en esa tabla.

Otra forma de eliminar todas las filas de la tabla es truncarla de la siguiente manera:

```
TRUNCATE TABLE HelloWorlds
```

Diferencia con la operación DELETE son varias:

1. La operación truncada no se almacena en el archivo de registro de transacciones
2. Si existe el campo `IDENTITY` , este será restablecido
3. TRUNCATE se puede aplicar en una tabla completa y no en parte de ella (en lugar de con el comando `DELETE` , puede asociar una cláusula `WHERE`)

Restricciones de TRUNCATE

1. No se puede TRUNCAR una tabla si hay una referencia `FOREIGN KEY`
2. Si la mesa está participada en una `INDEXED VIEW`
3. Si la tabla se publica utilizando `TRANSACTIONAL REPLICATION` o `MERGE REPLICATION`
4. No disparará ningún GATILLO definido en la tabla

[sic]

TABLA DE TRUNCATOS

```
TRUNCATE TABLE Helloworlds
```

Este código borrará todos los datos de la tabla Helloworlds. Truncar tabla es casi similar a `Delete from Table` código de `Delete from Table` . La diferencia es que no se pueden usar las cláusulas `where` con `Truncate`. Truncar la tabla se considera mejor que eliminar porque usa menos espacios de registro de transacciones.

Tenga en cuenta que si existe una columna de identidad, se restablece al valor inicial inicial (por ejemplo, la ID auto-incrementada se reiniciará desde 1). Esto puede llevar a una inconsistencia si las columnas de identidad se usan como una clave externa en otra tabla.

Crear nueva tabla e insertar registros de la tabla antigua

```
SELECT * INTO NewTable FROM OldTable
```

Creo una nueva tabla con la estructura de la tabla antigua e inserta todas las filas en la nueva tabla.

Algunas restricciones

1. No puede especificar una variable de tabla o un parámetro con valores de tabla como la nueva tabla.
2. No puede utilizar `SELECT ... INTO` para crear una tabla particionada, incluso cuando la tabla fuente está particionada. `SELECT ... INTO` no usa el esquema de partición de la tabla fuente; en su lugar, la nueva tabla se crea en el grupo de archivos predeterminado. Para insertar filas en una tabla particionada, primero debe crear la tabla particionada y luego usar la instrucción `INSERT INTO ... SELECT FROM`.

3. Los índices, las restricciones y los activadores definidos en la tabla de origen no se transfieren a la nueva tabla, ni pueden especificarse en la instrucción `SELECT ... INTO`. Si estos objetos son necesarios, puede crearlos después de ejecutar la instrucción `SELECT ... INTO`.
4. La especificación de una cláusula `ORDER BY` no garantiza que las filas se inserten en el orden especificado. Cuando se incluye una columna dispersa en la lista de selección, la propiedad de la columna dispersa no se transfiere a la columna en la nueva tabla. Si se requiere esta propiedad en la nueva tabla, modifique la definición de la columna después de ejecutar la instrucción `SELECT ... INTO` para incluir esta propiedad.
5. Cuando se incluye una columna calculada en la lista de selección, la columna correspondiente en la nueva tabla no es una columna calculada. Los valores en la nueva columna son los valores que se calcularon en el momento en que se ejecutó `SELECT ... INTO`.

[sic]

Obtención de recuento de filas de la tabla

El siguiente ejemplo se puede usar para encontrar el número total de filas para una tabla específica en una base de datos si `table_name` se reemplaza por la tabla que desea consultar:

```
SELECT COUNT(*) AS [TotalRowCount] FROM table_name;
```

También es posible obtener el recuento de filas para todas las tablas uniéndose de nuevo a la partición de la tabla basada en el HEAP de las tablas (`index_id = 0`) o el índice agrupado (`index_id = 1`) usando la siguiente secuencia de comandos:

```
SELECT [Tables].name AS [TableName],
       SUM( [Partitions].[rows] ) AS [TotalRowCount]
FROM sys.tables AS [Tables]
JOIN sys.partitions AS [Partitions]
      ON [Tables].[object_id] = [Partitions].[object_id]
      AND [Partitions].index_id IN ( 0, 1 )
--WHERE [Tables].name = N'table name' /* uncomment to look for a specific table */
GROUP BY [Tables].name;
```

Esto es posible ya que cada tabla es esencialmente una tabla de partición única, a menos que se le agreguen particiones adicionales. Este script también tiene la ventaja de no interferir con las operaciones de lectura / escritura en las filas de las tablas '.

Lea Empezando con Microsoft SQL Server en línea: <https://riptutorial.com/es/sql-server/topic/236/empezando-con-microsoft-sql-server>

Capítulo 2: Administrar la base de datos SQL de Azure

Examples

Encuentre información de nivel de servicio para la base de datos SQL de Azure

Azure SQL Database tiene diferentes ediciones y niveles de rendimiento.

Puede encontrar la versión, edición (básica, estándar o premium) y objetivo de servicio (S0, S1, P4, P11, etc.) de la base de datos SQL que se ejecuta como servicio en Azure usando las siguientes declaraciones:

```
select @@version
SELECT DATABASEPROPERTYEX('Wwi', 'EDITION')
SELECT DATABASEPROPERTYEX('Wwi', 'ServiceObjective')
```

Cambiar el nivel de servicio de la base de datos SQL de Azure

Puede ampliar o reducir la base de datos de Azure SQL mediante la instrucción ALTER DATABASE:

```
ALTER DATABASE WWI
MODIFY (SERVICE_OBJECTIVE = 'P6')
-- or
ALTER DATABASE CURRENT
MODIFY (SERVICE_OBJECTIVE = 'P2')
```

Si intenta cambiar el nivel de servicio mientras se está cambiando el nivel de servicio de la base de datos actual, obtendrá el siguiente error:

Msg 40802, Nivel 16, Estado 1, Línea 1 Una asignación de objetivo de servicio en el servidor '.....' y la base de datos '.....' ya está en progreso. Espere hasta que el estado de asignación del objetivo de servicio para la base de datos se marque como 'Completado'.

Vuelva a ejecutar la instrucción ALTER DATABASE cuando finalice el período de transición.

Replicación de la base de datos SQL de Azure

Puede crear una réplica secundaria de la base de datos con el mismo nombre en otro Azure SQL Server, haciendo que la base de datos local sea la primaria, y comienza a replicar de forma asíncrona los datos de la primaria a la nueva secundaria.


```
ALTER DATABASE <<mydb>>  
ADD SECONDARY ON SERVER <<secondaryserver>>  
WITH ( ALLOW_CONNECTIONS = ALL )
```

El servidor de destino puede estar en otro centro de datos (utilizable para la replicación geográfica). Si ya existe una base de datos con el mismo nombre en el servidor de destino, el comando fallará. El comando se ejecuta en la base de datos maestra en el servidor que aloja la base de datos local que se convertirá en la principal. Cuando ALLOW_CONNECTIONS se establece en ALL (se establece en NO de forma predeterminada), la réplica secundaria será una base de datos de solo lectura que permitirá la conexión de todos los inicios de sesión con los permisos adecuados.

La réplica de la base de datos secundaria puede promocionarse a primaria mediante el siguiente comando:

```
ALTER DATABASE mydb FAILOVER
```

Puede eliminar la base de datos secundaria en el servidor secundario:

```
ALTER DATABASE <<mydb>>  
REMOVE SECONDARY ON SERVER <<testsecondaryserver>>
```

Crear base de datos SQL de Azure en el grupo de Elastic

Puedes poner tu base de datos SQL de Azure en el grupo elástico de SQL:

```
CREATE DATABASE wwi  
( SERVICE_OBJECTIVE = ELASTIC_POOL ( name = mypool1 ) )
```

Puede crear una copia de una base de datos existente y colocarla en un grupo elástico:

```
CREATE DATABASE wwi  
AS COPY OF myserver.WideWorldImporters  
( SERVICE_OBJECTIVE = ELASTIC_POOL ( name = mypool1 ) )
```

Lea **Administrar la base de datos SQL de Azure en línea**: <https://riptutorial.com/es/sql-server/topic/7113/administrar-la-base-de-datos-sql-de-azure>

Capítulo 3: Agente de servicios

Examples

1. Fundamentos

Service Broker es una tecnología basada en la comunicación asíncrona entre dos (o más) entidades. Service Broker consta de: tipos de mensajes, contratos, colas, servicios, rutas y al menos puntos finales de instancia

Más: <https://msdn.microsoft.com/en-us/library/bb522893.aspx>

2. Habilitar Service Broker en la base de datos

```
ALTER DATABASE [MyDatabase] SET ENABLE_BROKER WITH ROLLBACK IMMEDIATE;
```

3. Crear la construcción del agente de servicios básicos en la base de datos (comunicación de base de datos única)

```
USE [MyDatabase]

CREATE MESSAGE TYPE [//initiator] VALIDATION = WELL_FORMED_XML;
GO

CREATE CONTRACT [//call/contract]
(
    [//initiator] SENT BY INITIATOR
)
GO

CREATE QUEUE InitiatorQueue;
GO

CREATE QUEUE TargetQueue;
GO

CREATE SERVICE InitiatorService
    ON QUEUE InitiatorQueue
(
    [//call/contract]
)

CREATE SERVICE TargetService
    ON QUEUE TargetQueue
(
    [//call/contract]
)

GRANT SEND ON SERVICE::[InitiatorService] TO PUBLIC
GO
```

```
GRANT SEND ON SERVICE::[TargetService] TO PUBLIC
GO
```

No necesitamos ruta para una comunicación de base de datos.

4. Cómo enviar comunicación básica a través de Service Broker.

Para esta demostración usaremos la construcción del agente de servicios creada en otra parte de esta documentación. La parte mencionada se llama **3. Cree una construcción básica de Service Broker en la base de datos (comunicación de base de datos única)** .

```
USE [MyDatabase]

DECLARE @ch uniqueidentifier = NEWID()
DECLARE @msg XML

BEGIN DIALOG CONVERSATION @ch
  FROM SERVICE [InitiatorService]
  TO SERVICE 'TargetService'
  ON CONTRACT [//call/contract]
  WITH ENCRYPTION = OFF; -- more possible options

  SET @msg = (
    SELECT 'HelloThere' "elementNum1"
    FOR XML PATH(''), ROOT('ExampleRoot'), ELEMENTS XSINIL, TYPE
  );

SEND ON CONVERSATION @ch MESSAGE TYPE [//initiator] (@msg);
END CONVERSATION @ch;
```

Después de esta conversación será tu msg en TargetQueue

5. Cómo recibir la conversación de TargetQueue automáticamente

Para esta demostración usaremos la construcción del agente de servicios creada en otra parte de esta documentación. La parte mencionada se llama **3. Crear la construcción básica de Service Broker en la base de datos (comunicación de base de datos única)** .

Primero necesitamos crear un procedimiento que sea capaz de leer y procesar datos de la cola.

```
USE [MyDatabase]
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[p_RecieveMessageFromTargetQueue]

AS
BEGIN
```

```

declare
@message_body xml,
@message_type_name nvarchar(256),
@conversation_handle uniqueidentifier,
@messagetypername nvarchar(256);

WHILE 1=1
BEGIN

BEGIN TRANSACTION
    WAITFOR(
    RECEIVE TOP(1)
    @message_body = CAST(message_body as xml),
    @message_type_name = message_type_name,
    @conversation_handle = conversation_handle,
    @messagetypername = message_type_name
    FROM DwhInsertSmsQueuee
    ), TIMEOUT 1000;

    IF (@@ROWCOUNT = 0)
        BEGIN
            ROLLBACK TRANSACTION
            BREAK
        END

    IF (@messagetypername = '//initiator')
        BEGIN

            IF OBJECT_ID('MyDatabase..MyExampleTableHelloThere') IS NOT NULL
                DROP TABLE dbo.MyExampleTableHelloThere

            SELECT @message_body.value('/ExampleRoot/"elementNum1"[1]', 'VARCHAR(50)')
AS MyExampleMessage
            INTO dbo.MyExampleTableHelloThere

        END

    IF (@messagetypername = 'http://schemas.microsoft.com/SQL/ServiceBroker/EndDialog')
        BEGIN
            END CONVERSATION @conversation_handle;
        END

    COMMIT TRANSACTION
END

END

```

Segundo paso: permita que su TargetQueue ejecute automáticamente su procedimiento:

```

USE [MyDatabase]

ALTER QUEUE [dbo].[TargetQueue] WITH STATUS = ON , RETENTION = OFF ,
ACTIVATION
( STATUS = ON , --activation status
PROCEDURE_NAME = dbo.p_RecieveMessageFromTargetQueue , --procedure name

```

```
MAX_QUEUE_READERS = 1 , --number of readers  
EXECUTE AS SELF )
```

Lea Agente de servicios en línea: <https://riptutorial.com/es/sql-server/topic/7651/agente-de-servicios>

Capítulo 4: AGRUPAR POR

Examples

Agrupación simple

Tabla de Pedidos

Identificación del cliente	Identificación de producto	Cantidad	Precio
1	2	5	100
1	3	2	200
1	4	1	500
2	1	4	50
3	5	6	700

Al agrupar por una columna específica, solo se devuelven los valores únicos de esta columna.

```
SELECT customerId
FROM orders
GROUP BY customerId;
```

Valor de retorno:

Identificación del cliente
1
2
3

Las funciones agregadas como `count()` aplican a cada grupo y no a la tabla completa:

```
SELECT customerId,
       COUNT(productId) as numberOfProducts,
       sum(price) as totalPrice
FROM orders
GROUP BY customerId;
```

Valor de retorno:

Identificación del cliente	número de productos	precio total
1	3	800
2	1	50
3	1	700

GRUPO POR VARIAS COLECCIONES

Uno podría querer agrupar por más de una columna

```
declare @temp table(age int, name varchar(15))

insert into @temp
select 18, 'matt' union all
select 21, 'matt' union all
select 21, 'matt' union all
select 18, 'luke' union all
select 18, 'luke' union all
select 21, 'luke' union all
select 18, 'luke' union all
select 21, 'luke'

SELECT Age, Name, count(1) count
FROM @temp
GROUP BY Age, Name
```

Agrupará por edad y nombre y producirá:

Años	Nombre	contar
18	luke	3
21	luke	2
18	mate	1
21	mate	2

Agrupar por con múltiples tablas, múltiples columnas.

Agrupar por se utiliza a menudo con la instrucción de unión. Supongamos que tenemos dos tablas. La primera es la tabla de alumnos:

Carné de identidad	Nombre completo	Años
1	Matt Jones	20
2	Frank blue	21

Carné de identidad	Nombre completo	Años
3	Anthony Angel	18

La segunda mesa es la tabla de materias que cada estudiante puede tomar:

Subject_Id	Tema
1	Matemáticas
2	EDUCACIÓN FÍSICA
3	Física

Y debido a que un estudiante puede asistir a muchas asignaturas y a una sola materia pueden asistir muchos (de ahí la relación N: N), necesitamos tener una tercera tabla de "límites".

Llamemos a la tabla Students_subjects:

Subject_Id	Identificación del Estudiante
1	1
2	2
2	1
3	2
1	3
1	1

Ahora digamos que queremos saber la cantidad de asignaturas a las que asiste cada estudiante. En este caso, la instrucción `GROUP BY` independiente no es suficiente, ya que la información no está disponible a través de una sola tabla. Por lo tanto, necesitamos usar `GROUP BY` con la instrucción

`JOIN` :

```
Select Students.FullName, COUNT(Subject Id) as SubjectNumber FROM Students_Subjects
LEFT JOIN Students
ON Students_Subjects.Student_id = Students.Id
GROUP BY Students.FullName
```

El resultado de la consulta dada es el siguiente:

Nombre completo	SubjectNumber
Matt Jones	3
Frank blue	2

Nombre completo	SubjectNumber
Anthony Angel	1

Para un ejemplo aún más complejo de uso de GROUP BY, digamos que el estudiante podría asignar el mismo tema a su nombre más de una vez (como se muestra en la tabla Students_Subjects). En este escenario, podríamos contar el número de veces que cada materia fue asignada a un estudiante agrupando por más de una columna:

```
SELECT Students.FullName, Subjects.Subject,
COUNT(Students_subjcts.Subject_id) AS NumberOfOrders
FROM ((Students_Subjects
INNER JOIN Students
ON Students_Subjcts.Student_id=Students.Id)
INNER JOIN Subjects
ON Students_Subjects.Subject_id=Subjects.Subject_id)
GROUP BY Fullname, Subject
```

Esta consulta da el siguiente resultado:

Nombre completo	Tema	SubjectNumber
Matt Jones	Matemáticas	2
Matt Jones	EDUCACIÓN FÍSICA	1
Frank blue	EDUCACIÓN FÍSICA	1
Frank blue	Física	1
Anthony Angel	Matemáticas	1

TENIENDO

Debido a que la cláusula WHERE se evalúa antes de GROUP BY, no puede usar WHERE para reducir los resultados de la agrupación (generalmente una función agregada, como COUNT(*)). Para satisfacer esta necesidad, se puede utilizar la cláusula HAVING.

Por ejemplo, utilizando los siguientes datos:

```
DECLARE @orders TABLE(OrderID INT, Name NVARCHAR(100))

INSERT INTO @orders VALUES
( 1, 'Matt' ),
( 2, 'John' ),
( 3, 'Matt' ),
( 4, 'Luke' ),
( 5, 'John' ),
( 6, 'Luke' ),
( 7, 'John' ),
( 8, 'John' ),
( 9, 'Luke' ),
```

```
( 10, 'John' ),  
( 11, 'Luke' )
```

Si deseamos obtener el número de pedidos que ha realizado cada persona, utilizaríamos

```
SELECT Name, COUNT(*) AS 'Orders'  
FROM @orders  
GROUP BY Name
```

y obten

Nombre	Pedidos
Mate	2
Juan	5
Lucas	4

Sin embargo, si queremos limitar esto a las personas que han realizado más de dos órdenes, podemos agregar una cláusula `HAVING`.

```
SELECT Name, COUNT(*) AS 'Orders'  
FROM @orders  
GROUP BY Name  
HAVING COUNT(*) > 2
```

rendirá

Nombre	Pedidos
Juan	5
Lucas	4

Tenga en cuenta que, al igual que `GROUP BY`, las columnas que se colocan en `HAVING` deben coincidir exactamente con sus contrapartes en la instrucción `SELECT`. Si en el ejemplo anterior hubiéramos dicho en su lugar

```
SELECT Name, COUNT(DISTINCT OrderID)
```

Nuestra cláusula `HAVING` tendría que decir

```
HAVING COUNT(DISTINCT OrderID) > 2
```

GROUP BY con ROLLUP y CUBE

El operador `ROLLUP` es útil para generar informes que contienen subtotales y totales.

- CUBE genera un conjunto de resultados que muestra agregados para todas las combinaciones de valores en las columnas seleccionadas.
- ROLLUP genera un conjunto de resultados que muestra agregados para una jerarquía de valores en las columnas seleccionadas.

Ít	Color	Cantidad
Mesa	Azul	124
Mesa	rojo	223
Silla	Azul	101
Silla	rojo	210

```
SELECT CASE WHEN (GROUPING(Item) = 1) THEN 'ALL'
         ELSE ISNULL(Item, 'UNKNOWN')
        END AS Item,
       CASE WHEN (GROUPING(Color) = 1) THEN 'ALL'
         ELSE ISNULL(Color, 'UNKNOWN')
        END AS Color,
       SUM(Quantity) AS QtySum
FROM Inventory
GROUP BY Item, Color WITH ROLLUP
```

Item	Color	QtySum
Chair	Blue	101.00
Chair	Red	210.00
Chair	ALL	311.00
Table	Blue	124.00
Table	Red	223.00
Table	ALL	347.00
ALL	ALL	658.00

(7 fila (s) afectadas)

Si la palabra clave ROLLUP en la consulta se cambia a CUBE, el conjunto de resultados de CUBE es el mismo, excepto que estas dos filas adicionales se devuelven al final:

ALL	Blue	225.00
ALL	Red	433.00

[https://technet.microsoft.com/en-us/library/ms189305\(v=sql.90\).aspx](https://technet.microsoft.com/en-us/library/ms189305(v=sql.90).aspx)

Lea AGRUPAR POR en línea: <https://riptutorial.com/es/sql-server/topic/3231/agrupar-por>

Capítulo 5: Almacenamiento de JSON en tablas SQL

Examples

JSON almacenado como columna de texto

JSON es un formato de texto, por lo que se almacena en columnas NVARCHAR estándar. La colección NoSQL es equivalente a dos tablas de valores de clave de columna:

```
CREATE TABLE ProductCollection (  
    Id int identity primary key,  
    Data nvarchar(max)  
)
```

Use `nvarchar(max)` ya que no está seguro de cuál sería el tamaño de sus documentos JSON. `nvarchar(4000)` y `varchar(8000)` tienen un mejor rendimiento pero con un límite de tamaño de 8 KB.

Asegúrese de que JSON esté formateado correctamente utilizando ISJSON

Dado que JSON se almacena en la columna de texto, es posible que desee asegurarse de que está formateado correctamente. Puede agregar la restricción CHECK en la columna JSON que verifica si JSON tiene el formato correcto de texto:

```
CREATE TABLE ProductCollection (  
    Id int identity primary key,  
    Data nvarchar(max)  
        CONSTRAINT [Data should be formatted as JSON]  
        CHECK (ISJSON(Data) > 0)  
)
```

Si ya tiene una tabla, puede agregar una restricción de verificación mediante la instrucción ALTER TABLE:

```
ALTER TABLE ProductCollection  
    ADD CONSTRAINT [Data should be formatted as JSON]  
        CHECK (ISJSON(Data) > 0)
```

Exponer valores de texto JSON como columnas calculadas

Puede exponer los valores de la columna JSON como columnas calculadas:

```
CREATE TABLE ProductCollection (  
    Id int identity primary key,  
    Data nvarchar(max),  
    Price AS JSON_VALUE(Data, '$.Price'),  
    Color JSON_VALUE(Data, '$.Color') PERSISTED
```

```
)
```

Si agrega la columna calculada PERSISTADA, el valor del texto JSON se materializará en esta columna. De esta manera, sus consultas pueden leer más rápidamente el valor del texto JSON porque no se necesita un análisis. Cada vez que JSON en esta fila cambia, el valor se volverá a calcular.

Agregando índice en la ruta JSON

Las consultas que filtran u ordenan los datos por algún valor en la columna JSON usualmente usan el escaneo completo de la tabla.

```
SELECT * FROM ProductCollection
WHERE JSON_VALUE(Data, '$.Color') = 'Black'
```

Para optimizar este tipo de consultas, puede agregar una columna calculada no persistente que exponga la expresión JSON utilizada en el filtro u ordenación (en este ejemplo, JSON_VALUE (Datos, '\$.Color')), y crear un índice en esta columna:

```
ALTER TABLE ProductCollection
ADD vColor as JSON_VALUE(Data, '$.Color')

CREATE INDEX idx_JsonColor
ON ProductCollection(vColor)
```

Las consultas utilizarán el índice en lugar de la exploración de tabla simple.

JSON almacenado en tablas en memoria

Si puede usar tablas optimizadas en memoria, puede almacenar JSON como texto:

```
CREATE TABLE ProductCollection (
  Id int identity primary key nonclustered,
  Data nvarchar(max)
) WITH (MEMORY_OPTIMIZED=ON)
```

Ventajas de JSON en memoria:

- Los datos JSON están siempre en la memoria, por lo que no hay acceso al disco
- No hay bloqueos ni pestillos mientras se trabaja con JSON

Lea Almacenamiento de JSON en tablas SQL en línea: <https://riptutorial.com/es/sql-server/topic/5029/almacenamiento-de-json-en-tablas-sql>

Capítulo 6: Analizando una consulta

Examples

Escanear vs buscar

Al ver un plan de ejecución, puede ver que SQL Server decidió realizar una Búsqueda o un Análisis.

Una búsqueda ocurre cuando SQL Server sabe a dónde debe ir y solo toma elementos específicos. Esto ocurre normalmente cuando se ponen buenos filtros en una consulta, como

```
where name = 'Foo' .
```

Una exploración es cuando SQL Server no sabe exactamente dónde están todos los datos que necesita, o decidió que la exploración sería más eficiente que una búsqueda si se seleccionan suficientes datos.

Las búsquedas suelen ser más rápidas, ya que solo capturan una subsección de los datos, mientras que las exploraciones seleccionan la mayoría de los datos.

Lea **Analizando una consulta en línea**: <https://riptutorial.com/es/sql-server/topic/7713/analizando-una-consulta>

Capítulo 7: aplicación cruzada

Examples

Unir filas de tablas con filas generadas dinámicamente desde una celda

CROSS APPLY le permite "unir" filas de una tabla con filas generadas dinámicamente devueltas por alguna función de valor de tabla.

Imagine que tiene una tabla de empresa con una columna que contiene una matriz de productos (columna *ProductList*) y una función que analiza estos valores y devuelve un conjunto de productos. Puede seleccionar todas las filas de una tabla de la Compañía, aplicar esta función en una columna *ProductList* y "unir" los resultados generados con la fila de la Compañía principal:

```
SELECT *
FROM Companies c
     CROSS APPLY dbo.GetProductList( c.ProductList ) p
```

Para cada fila, el valor de la celda *ProductList* se proporcionará a la función, y la función devolverá esos productos como un conjunto de filas que se pueden unir con la fila principal.

Unir filas de la tabla con la matriz JSON almacenada en la celda

CROSS APPLY le permite "unir" filas de una tabla con una colección de objetos JSON almacenados en una columna.

Imagine que tiene una tabla de empresa con una columna que contiene una matriz de productos (columna *ProductList*) formateada como matriz JSON. La función de valor de tabla OPENJSON puede analizar estos valores y devolver el conjunto de productos. Puede seleccionar todas las filas de una tabla de la Compañía, analizar los productos JSON con OPENJSON y "unir" los resultados generados con la fila de la Compañía principal:

```
SELECT *
FROM Companies c
     CROSS APPLY OPENJSON( c.ProductList )
                WITH ( Id int, Title nvarchar(30), Price money)
```

Para cada fila, el valor de la celda *ProductList* se proporcionará a la función OPENJSON que transformará los objetos JSON en filas con el esquema definido en la cláusula WITH.

Filtrar filas por valores de matriz

Si almacena una lista de etiquetas en una fila como valores separados por coma, la función *STRING_SPLIT* le permite transformar la lista de etiquetas en una tabla de valores. **CROSS APPLY** le permite "unir" valores analizados por la función *STRING_SPLIT* con una fila principal.

Imagine que tiene una tabla de productos con una columna que contiene una matriz de etiquetas

separadas por comas (p. Ej., Promoción, ventas, nuevas). `STRING_SPLIT` y `CROSS APPLY` te permiten unir filas de productos con sus etiquetas para que puedas filtrar productos por etiquetas:

```
SELECT *
FROM Products p
     CROSS APPLY STRING_SPLIT( p.Tags, ',' ) tags
WHERE tags.value = 'promo'
```

Para cada fila, se proporcionará el valor de la celda *Etiquetas* a la función `STRING_SPLIT` que devolverá los valores de las etiquetas. Entonces puedes filtrar filas por estos valores.

Nota: la función `STRING_SPLIT` no está disponible antes de **SQL Server 2016**

Lea aplicación cruzada en línea: <https://riptutorial.com/es/sql-server/topic/5462/aplicacion-cruzada>

Capítulo 8: Base de datos del sistema - TempDb

Examples

Identificar el uso de TempDb

La siguiente consulta proporcionará información sobre el uso de TempDb. Analizando los conteos puedes identificar qué cosa está afectando a TempDb

```
SELECT
  SUM (user_object_reserved_page_count)*8 as usr_obj_kb,
  SUM (internal_object_reserved_page_count)*8 as internal_obj_kb,
  SUM (version_store_reserved_page_count)*8 as version_store_kb,
  SUM (unallocated_extent_page_count)*8 as freespace_kb,
  SUM (mixed_extent_page_count)*8 as mixedextent_kb
FROM sys.dm_db_file_space_usage
```

Attribute	Meaning
Higher number of user objects	More usage of Temp tables , cursors or temp variables
Higher number of internal objects	Query plan is using a lot of database. Ex: sorting, Group by etc.
Higher number of version stores	Long running transaction or high transaction throughput

Detalles de la base de datos TempDB

La siguiente consulta se puede utilizar para obtener los detalles de la base de datos TempDB:

```
USE [MASTER]
SELECT * FROM sys.databases WHERE database_id = 2
```

O

```
USE [MASTER]
SELECT * FROM sys.master_files WHERE database_id = 2
```

Con la ayuda de debajo del DMV, puede verificar cuánto espacio TempDb está usando su sesión. Esta consulta es bastante útil al depurar problemas de TempDb

```
SELECT * FROM sys.dm_db_session_space_usage WHERE session_id = @@SPID
```

Lea Base de datos del sistema - TempDb en línea: <https://riptutorial.com/es/sql-server/topic/4427/base-de-datos-del-sistema---tempdb>

Capítulo 9: Cifrado

Parámetros

Parámetros opcionales	Detalles
WITH PRIVATE KEY	Para CREAR CERTIFICADO, se puede especificar una clave privada: (FILE='D:\Temp\CertTest\private.pvk', DECRYPTION BY PASSWORD = 'password');

Observaciones

La creación de un certificado DER funcionará bien. Sin embargo, cuando se usa un certificado Base64, el servidor SQL se quejará con el mensaje críptico:

```
Msg 15468, Level 16, State 6, Line 1  
An error occurred during the generation of the certificate.
```

Importe su certificado Base64 al almacén de certificados de su sistema operativo para poder reexportarlo a formato binario DER.

Otra cosa importante que hacer es tener una jerarquía de cifrado para que una proteja a la otra, hasta el nivel del sistema operativo. Vea el artículo sobre 'Cifrado de base de datos / TDE'

Para obtener más información sobre la creación de certificados, visite:

<https://msdn.microsoft.com/en-us/library/ms187798.aspx>

Para obtener más información sobre el cifrado de la base de datos / TDE, visite:

<https://msdn.microsoft.com/en-us/library/bb934049.aspx>

Para obtener más información sobre el cifrado de datos, visite: <https://msdn.microsoft.com/en-us/library/ms188061.aspx>

Examples

Cifrado por certificado

```
CREATE CERTIFICATE My_New_Cert  
FROM FILE = 'D:\Temp\CertTest\certificateDER.cer'  
GO
```

Crear el certificado

```
SELECT EncryptByCert (Cert_ID('My_New_Cert'),  
'This text will get encrypted') encryption_test
```

Generalmente, usted cifraría con una clave simétrica, esa clave se cifraría con la clave asimétrica (clave pública) de su certificado.

Además, tenga en cuenta que el cifrado está limitado a ciertas longitudes dependiendo de la longitud de la clave y, de lo contrario, devuelve NULL. Microsoft escribe: "Los límites son: una clave RSA de 512 bits puede cifrar hasta 53 bytes, una clave de 1024 bits puede cifrar hasta 117 bytes y una clave de 2048 bits puede cifrar hasta 245 bytes".

EncryptByAsymKey tiene los mismos límites. Para UNICODE, esto se dividiría por 2 (16 bits por carácter), por lo que 58 caracteres para una clave de 1024 bits.

Cifrado de la base de datos

```
USE TDE  
CREATE DATABASE ENCRYPTION KEY  
WITH ALGORITHM = AES_256  
ENCRYPTION BY SERVER CERTIFICATE My_New_Cert  
GO  
  
ALTER DATABASE TDE  
SET ENCRYPTION ON  
GO
```

Esto utiliza 'cifrado de datos transparente' (TDE)

Cifrado por clave simétrica

```
-- Create the key and protect it with the cert  
CREATE SYMMETRIC KEY My_Sym_Key  
WITH ALGORITHM = AES_256  
ENCRYPTION BY CERTIFICATE My_New_Cert;  
GO  
  
-- open the key  
OPEN SYMMETRIC KEY My_Sym_Key  
DECRYPTION BY CERTIFICATE My_New_Cert;  
  
-- Encrypt  
SELECT EncryptByKey(Key_GUID('SSN_Key_01'), 'This text will get encrypted');
```

Cifrado por frase de contraseña

```
SELECT EncryptByPassphrase('MyPassPhrase', 'This text will get encrypted')
```

Esto también se cifrará, pero luego mediante una contraseña en lugar de una clave asimétrica (certificado) o mediante una clave simétrica explícita.

Lea Cifrado en línea: <https://riptutorial.com/es/sql-server/topic/7096/cifrado>

Capítulo 10: Columnas calculadas

Examples

Una columna se calcula a partir de una expresión

Una columna calculada se calcula a partir de una expresión que puede usar otras columnas en la misma tabla. La expresión puede ser un nombre de columna, una constante, una función y cualquier combinación de estos no conectados por uno o más operadores.

Crear tabla con una columna calculada.

```
Create table NetProfit
(
    SalaryToEmployee          int,
    BonusDistributed          int,
    BusinessRunningCost       int,
    BusinessMaintenanceCost   int,
    BusinessEarnings          int,
    BusinessNetIncome         AS BusinessEarnings - (SalaryToEmployee +
                                                BonusDistributed +
                                                BusinessRunningCost +
                                                BusinessMaintenanceCost )
)
```

El valor se calcula y almacena en la columna calculada automáticamente al insertar otros valores.

```
Insert Into NetProfit
(SalaryToEmployee,
 BonusDistributed,
 BusinessRunningCost,
 BusinessMaintenanceCost,
 BusinessEarnings)
Values
(1000000,
 10000,
 1000000,
 50000,
 2500000)
```

Ejemplo simple que usamos normalmente en tablas de registro

```
CREATE TABLE [dbo].[ProcessLog] (
    [LogId] [int] IDENTITY(1,1) NOT NULL,
    [LogType] [varchar](20) NULL,
    [StartTime] [datetime] NULL,
    [EndTime] [datetime] NULL,
    [RunMinutes] AS
    (datediff(minute, coalesce([StartTime], getdate()), coalesce([EndTime], getdate())))
)
```

Esto da una diferencia de ejecución en minutos para el tiempo de ejecución que será muy útil.

Lea Columnas calculadas en línea: <https://riptutorial.com/es/sql-server/topic/5561/columnas-calculadas>

Capítulo 11: COLUMNSTORE CLUSTRADO

Examples

Tabla con índice CLUSTERED COLUMNSTORE

Si desea tener una tabla organizada en formato de almacén de columnas en lugar de una fila, agregue INDEX cci CLUSTERED COLUMNSTORE en la definición de tabla:

```
DROP TABLE IF EXISTS Product
GO
CREATE TABLE Product (
    ProductID int,
    Name nvarchar(50) NOT NULL,
    Color nvarchar(15),
    Size nvarchar(5) NULL,
    Price money NOT NULL,
    Quantity int,
    INDEX cci CLUSTERED COLUMNSTORE
)
```

Las tablas de COLUMNSTORE son mejores para las tablas en las que espera exploraciones e informes completos, mientras que las tablas del almacén de filas son mejores para las tablas en las que leerá o actualizará conjuntos más pequeños de filas.

Agregar índice de almacén de columnas agrupado en la tabla existente

CREATE CLUSTERED COLUMNSTORE INDEX le permite organizar una tabla en formato de columna:

```
DROP TABLE IF EXISTS Product
GO
CREATE TABLE Product (
    Name nvarchar(50) NOT NULL,
    Color nvarchar(15),
    Size nvarchar(5) NULL,
    Price money NOT NULL,
    Quantity int
)
GO
CREATE CLUSTERED COLUMNSTORE INDEX cci ON Product
```

Reconstruir el índice CLUSTERED COLUMNSTORE

El índice del almacén de columnas agrupadas se puede reconstruir si tiene muchas filas eliminadas:

```
ALTER INDEX cci ON Products
REBUILD PARTITION = ALL
```

La reconstrucción de CLUSTERED COLUMNSTORE "recargará" los datos de la tabla actual en uno nuevo y aplicará la compresión nuevamente, eliminará las filas eliminadas, etc.

Puedes reconstruir una o más particiones.

Lea COLUMNSTORE CLUSTRADO en línea: <https://riptutorial.com/es/sql-server/topic/5774/columnstore-clustrado>

Capítulo 12: Con la opción de corbatas

Examples

Datos de prueba

```
CREATE TABLE #TEST
(
  Id INT,
  Name VARCHAR(10)
)

Insert Into #Test
select 1, 'A'
Union All
Select 1, 'B'
union all
Select 1, 'C'
union all
Select 2, 'D'
```

A continuación se muestra la salida de la tabla anterior. Como puede ver, la columna Id se repite tres veces ...

Id	Name
1	A
1	B
1	C
2	D

Ahora permite comprobar la salida usando orden simple por ...

```
Select Top (1) Id,Name From
#test
Order By Id ;
```

Salida: (La salida de la consulta anterior no se garantiza que sea la misma cada vez)

Id	Name
1	B

Permite ejecutar la misma consulta con la opción de enlaces.

```
Select Top (1) With Ties Id,Name
From
#test
Order By Id
```

Salida:

Id	Name
1	A
1	B
1	C

Como puede ver, el servidor SQL genera todas las filas **que están vinculadas con** orden por columna. Veamos un ejemplo más para entender esto mejor ..

```
Select Top (1) With Ties Id,Name
From
#test
Order By Id ,Name
```

Salida:

Id	Name
1	A

En resumen, cuando usamos con la opción de vínculos, SQL Server genera todas las filas vinculadas independientemente del límite que impongamos

Lea Con la opción de corbatas en línea: <https://riptutorial.com/es/sql-server/topic/2546/con-la-opcion-de-corbatas>

Capítulo 13: Conjunto de resultados de límite

Introducción

A medida que crecen las tablas de la base de datos, a menudo es útil limitar los resultados de las consultas a un número o porcentaje fijo. Esto se puede lograr usando la palabra clave `TOP` SQL Server o la cláusula `OFFSET FETCH`.

Parámetros

Parámetro	Detalles
<code>TOP</code>	Limitar palabra clave. Utilizar con un número.
<code>PERCENT</code>	Porcentaje de palabras clave. Viene después de <code>TOP</code> y número limitante.

Observaciones

Si se utiliza la cláusula `ORDER BY`, la limitación se aplica al conjunto de resultados ordenados.

Examples

Limitar con TOP

Este ejemplo limita el resultado `SELECT` a 100 filas.

```
SELECT TOP 100 *  
FROM table_name;
```

También es posible usar una variable para especificar el número de filas:

```
DECLARE @CountDesiredRows int = 100;  
SELECT TOP (@CountDesiredRows) *  
FROM table_name;
```

Limitar con el POR CIENTO

Este ejemplo limita el resultado de `SELECT` al 15 por ciento del recuento total de filas.

```
SELECT TOP 15 PERCENT *  
FROM table_name
```

Limitar con FETCH

SQL Server 2012

`FETCH` es generalmente más útil para la paginación, pero se puede usar como una alternativa a `TOP` :

```
SELECT *
FROM table_name
ORDER BY 1
OFFSET 0 ROWS
FETCH NEXT 50 ROWS ONLY
```

Lea Conjunto de resultados de límite en línea: <https://riptutorial.com/es/sql-server/topic/1555/conjunto-de-resultados-de-limite>

Capítulo 14: Consejos de consulta

Examples

ÚNETE a sugerencias

Cuando se unen dos tablas, el optimizador de consultas (QO) de SQL Server puede elegir diferentes tipos de combinaciones que se usarán en la consulta:

- HASH unirse
- Loop unirse
- MERGE unirse

QO explorará los planes y elegirá al operador óptimo para unir tablas. Sin embargo, si está seguro de saber cuál sería el operador de combinación óptimo, puede especificar qué tipo de combinación debe usarse. La combinación interna de LOOP obligará a QO a elegir la combinación de bucle anidado al unir dos tablas:

```
select top 100 *
from Sales.Orders o
    inner loop join Sales.OrderLines ol
    on o.OrderID = ol.OrderID
```

la unión interna de combinación forzará el operador de combinación de MERGE:

```
select top 100 *
from Sales.Orders o
    inner merge join Sales.OrderLines ol
    on o.OrderID = ol.OrderID
```

La combinación de hash interno forzará al operador de unión HASH:

```
select top 100 *
from Sales.Orders o
    inner hash join Sales.OrderLines ol
    on o.OrderID = ol.OrderID
```

GRUPO POR CONSEJOS

Cuando utiliza la cláusula GROUP BY, el optimizador de consultas de SQL Server (QO) puede elegir diferentes tipos de operadores de agrupación:

- Agregado de HASH que crea un hash-map para agrupar entradas
- Agregado de flujo que funciona bien con entradas pre-ordenadas

Puede requerir explícitamente que QO elija uno u otro operador agregado si sabe cuál sería el óptimo. Con OPTION (ORDER GROUP), QO siempre elegirá Stream aggregate y agregará el

operador Sort al frente de Stream agregate si la entrada no está ordenada:

```
select OrderID, AVG(Quantity)
from Sales.OrderLines
group by OrderID
OPTION (ORDER GROUP)
```

Con OPTION (HASH GROUP), QO siempre elegirá el agregado de hash:

```
select OrderID, AVG(Quantity)
from Sales.OrderLines
group by OrderID
OPTION (HASH GROUP)
```

Sugerencia de filas FAST

Especifica que la consulta está optimizada para la recuperación rápida de las primeras `number_rows`. Este es un entero no negativo. Después de que se devuelven los primeros `number_rows`, la consulta continúa la ejecución y produce su conjunto de resultados completo.

```
select OrderID, AVG(Quantity)
from Sales.OrderLines
group by OrderID
OPTION (FAST 20)
```

Consejos UNION

Cuando utiliza el operador UNION en dos resultados de consulta, el Optimizador de consultas (QO) puede usar los siguientes operadores para crear una unión de dos conjuntos de resultados:

- Fusionar (Unión)
- Concat (Unión)
- Hash Match (Unión)

Puede especificar explícitamente qué operador se debe usar usando la sugerencia OPTION ():

```
select OrderID, OrderDate, ExpectedDeliveryDate, Comments
from Sales.Orders
where OrderDate > DATEADD(day, -1, getdate())
UNION
select PurchaseOrderID as OrderID, OrderDate, ExpectedDeliveryDate, Comments
from Purchasing.PurchaseOrders
where OrderDate > DATEADD(day, -1, getdate())
OPTION (HASH UNION)
-- or OPTION (CONCAT UNION)
-- or OPTION (MERGE UNION)
```

Opción MAXDOP

Especifica el grado máximo de paralelismo para la consulta que especifica esta opción.

```
SELECT OrderID,  
       AVG(Quantity)  
FROM Sales.OrderLines  
GROUP BY OrderID  
OPTION (MAXDOP 2);
```

Esta opción anula la opción de configuración MAXDOP de sp_configure y Resource Governor. Si MAXDOP se establece en cero, el servidor elige el grado máximo de paralelismo.

INDEX Consejos

Las sugerencias de índice se usan para forzar a una consulta a usar un índice específico, en lugar de permitir que el Optimizador de consultas de SQL Server elija el que considere el mejor índice. En algunos casos, puede obtener beneficios al especificar el índice que debe usar una consulta. Por lo general, el Optimizador de consultas de SQL Server elige el mejor índice adecuado para la consulta, pero debido a estadísticas faltantes / obsoletas o necesidades específicas, puede forzarlo.

```
SELECT *  
FROM mytable WITH (INDEX (ix_date))  
WHERE field1 > 0  
      AND CreationDate > '20170101'
```

Lea Consejos de consulta en línea: <https://riptutorial.com/es/sql-server/topic/6881/consejos-de-consulta>

Capítulo 15: Consulta de resultados por página.

Examples

Numero de fila()

```
SELECT Row_Number() OVER(ORDER BY UserName) As RowID, UserFirstName, UserLastName  
FROM Users
```

Desde donde generará un conjunto de resultados con un campo RowID que puede usar para la página entre.

```
SELECT *  
FROM  
    ( SELECT Row_Number() OVER(ORDER BY UserName) As RowID, UserFirstName, UserLastName  
      FROM Users  
    ) As RowResults  
WHERE RowID Between 5 AND 10
```

Lea Consulta de resultados por página. en línea: <https://riptutorial.com/es/sql-server/topic/5803/consulta-de-resultados-por-pagina->

Capítulo 16: Consultas con datos JSON

Examples

Usando valores de JSON en la consulta

La función `JSON_VALUE` le permite tomar datos del texto JSON en la ruta especificada como segundo argumento y usar este valor en cualquier parte de la consulta de selección:

```
select ProductID, Name, Color, Size, Price, JSON_VALUE(Data, '$.Type') as Type
from Product
where JSON_VALUE(Data, '$.Type') = 'part'
```

Usando valores JSON en reportes

Una vez que los valores JSON se extraen del texto JSON, puede usarlos en cualquier parte de la consulta. Puede crear algún tipo de informe sobre datos JSON con agregaciones de agrupación, etc.:

```
select JSON_VALUE(Data, '$.Type') as type,
       AVG( cast(JSON_VALUE(Data, '$.ManufacturingCost') as float) ) as cost
from Product
group by JSON_VALUE(Data, '$.Type')
having JSON_VALUE(Data, '$.Type') is not null
```

Filtra el texto JSON incorrecto de los resultados de la consulta

Si algún texto JSON puede no tener el formato correcto, puede eliminar esas entradas de la consulta mediante la función `ISJSON`.

```
select ProductID, Name, Color, Size, Price, JSON_VALUE(Data, '$.Type') as Type
from Product
where JSON_VALUE(Data, '$.Type') = 'part'
and ISJSON(Data) > 0
```

Actualizar el valor en la columna JSON

La función `JSON_MODIFY` se puede usar para actualizar el valor en alguna ruta. Puede usar esta función para modificar el valor original de la celda JSON en la declaración `UPDATE`:

```
update Product
set Data = JSON_MODIFY(Data, '$.Price', 24.99)
where ProductID = 17;
```

La función `JSON_MODIFY` actualizará o creará la clave de precio (si no existe). Si el nuevo valor es `NULL`, la clave se eliminará. La función `JSON_MODIFY` tratará el nuevo valor como una cadena (escapa de los caracteres especiales, la envuelve con comillas dobles para crear la

cadena JSON adecuada). Si su nuevo valor es el fragmento JSON, debe envolverlo con la función `JSON_QUERY`:

```
update Product
set Data = JSON_MODIFY(Data, '$.tags', JSON_QUERY(['promo","new"]))
where ProductID = 17;
```

La función `JSON_QUERY` sin el segundo parámetro se comporta como una "conversión a JSON". Dado que el resultado de `JSON_QUERY` es un fragmento JSON válido (objeto o matriz), `JSON_MODIFY` no escapará a este valor cuando modifique la entrada JSON.

Agregar nuevo valor a la matriz JSON

La función `JSON_MODIFY` se puede usar para agregar un nuevo valor a alguna matriz dentro de JSON:

```
update Product
set Data = JSON_MODIFY(Data, 'append $.tags', "sales")
where ProductID = 17;
```

Se agregará un nuevo valor al final de la matriz, o se creará una nueva matriz con valor ["ventas"]. La función `JSON_MODIFY` tratará el nuevo valor como una cadena (escapa de los caracteres especiales, la envuelve con comillas dobles para crear la cadena JSON adecuada). Si su nuevo valor es el fragmento JSON, debe envolverlo con la función `JSON_QUERY`:

```
update Product
set Data = JSON_MODIFY(Data, 'append $.tags', JSON_QUERY({'type':"new"}))
where ProductID = 17;
```

La función `JSON_QUERY` sin el segundo parámetro se comporta como una "conversión a JSON". Dado que el resultado de `JSON_QUERY` es un fragmento JSON válido (objeto o matriz), `JSON_MODIFY` no escapará a este valor cuando modifique la entrada JSON.

Mesa JOIN con colección JSON interior

Si tiene una "tabla secundaria" formateada como colección JSON y almacenada en fila como columna JSON, puede descomprimir esta colección, transformarla en tabla y unirla a la fila principal. En lugar del operador estándar `JOIN`, debe usar `CROSS APPLY`. En este ejemplo, las partes del producto se formatean como una colección de objetos JSON y se almacenan en la columna de datos:

```
select ProductID, Name, Size, Price, Quantity, PartName, Code
from Product
  CROSS APPLY OPENJSON(Data, '$.Parts') WITH (PartName varchar(20), Code varchar(5))
```

El resultado de la consulta es equivalente a la unión entre las tablas de `Producto` y `Parte`.

Encontrar filas que contienen valor en la matriz JSON

En este ejemplo, la matriz de etiquetas puede contener varias palabras clave como ["promo", "ventas"], por lo que podemos abrir esta matriz y filtrar los valores:

```
select ProductID, Name, Color, Size, Price, Quantity
from Product
    CROSS APPLY OPENJSON(Data, '$.Tags')
where value = 'sales'
```

OPENJSON abrirá la colección interna de etiquetas y la devolverá como tabla. Luego podemos filtrar los resultados por algún valor en la tabla.

Lea Consultas con datos JSON en línea: <https://riptutorial.com/es/sql-server/topic/5028/consultas-con-datos-json>

Capítulo 17: Convertir tipos de datos

Examples

PRUEBA PARSE

SQL Server 2012

Convierte el tipo de datos de cadena al tipo de datos de destino (Fecha o Numérico).

Por ejemplo, los datos de origen son de tipo cadena y debemos convertirlos en tipo de fecha. Si el intento de conversión falla, devuelve un valor NULL.

Sintaxis: TRY_PARSE (string_value AS data_type [USING culture])

String_value: este argumento es el valor de origen que es el tipo NVARCHAR (4000).

Tipo de datos: este argumento es el tipo de datos de destino, ya sea fecha o numérico.

Cultura: es un argumento opcional que ayuda a convertir el valor en formato de Cultura.

Supongamos que desea mostrar la fecha en francés, entonces necesita pasar el tipo de cultura como 'Fr-FR'. Si no pasa ningún nombre de cultura válido, entonces PARSE generará un error.

```
DECLARE @fakeDate AS varchar(10);
DECLARE @realDate AS VARCHAR(10);
SET @fakeDate = 'iamnotadate';
SET @realDate = '13/09/2015';

SELECT TRY_PARSE(@fakeDate AS DATE); --NULL as the parsing fails

SELECT TRY_PARSE(@realDate AS DATE); -- NULL due to type mismatch

SELECT TRY_PARSE(@realDate AS DATE USING 'Fr-FR'); -- 2015-09-13
```

PRUEBA CONVERTIR

SQL Server 2012

Convierte el valor al tipo de datos especificado y, si la conversión falla, devuelve NULL. Por ejemplo, el valor de origen en formato de cadena y necesitamos el formato de fecha / entero. Entonces esto nos ayudará a lograr lo mismo.

Sintaxis: TRY_CONVERT (tipo de datos [(longitud)], expresión [, estilo])

TRY_CONVERT () devuelve una conversión de valor al tipo de datos especificado si la conversión tiene éxito; De lo contrario, devuelve nulo.

Tipo de datos: el tipo de datos al que se va a convertir. Aquí la longitud es un parámetro opcional que ayuda a obtener resultados en una longitud específica.

Expresión - El valor a convertir

Estilo: es un parámetro opcional que determina el formato. Supongamos que desea un formato de fecha como "18 de mayo de 2013", entonces necesita un estilo de pase como 111.

```
DECLARE @sampletext AS VARCHAR(10);
SET @sampletext = '123456';
DECLARE @realDate AS VARCHAR(10);
SET @realDate = '13/09/2015';
SELECT TRY_CONVERT(INT, @sampletext); -- 123456
SELECT TRY_CONVERT(DATETIME, @sampletext); -- NULL
SELECT TRY_CONVERT(DATETIME, @realDate, 111); -- Sep, 13 2015
```

TRY CAST

SQL Server 2012

Convierte el valor al tipo de datos especificado y, si la conversión falla, devuelve NULL. Por ejemplo, el valor de origen en formato de cadena y lo necesitamos en formato doble / entero. Entonces esto nos ayudará a lograrlo.

Sintaxis: TRY_CAST (expresión AS tipo de datos [(longitud)])

TRY_CAST () devuelve un valor de conversión al tipo de datos especificado si la conversión tiene éxito; De lo contrario, devuelve nulo.

Expresión: el valor de origen que se va a emitir.

Tipo de datos: el tipo de datos de destino que emitirá el valor de origen.

Longitud: es un parámetro opcional que especifica la longitud del tipo de datos de destino.

```
DECLARE @sampletext AS VARCHAR(10);
SET @sampletext = '123456';

SELECT TRY_CAST(@sampletext AS INT); -- 123456
SELECT TRY_CAST(@sampletext AS DATE); -- NULL
```

Emitir

La función Cast () se utiliza para convertir una variable de tipo de datos o datos de un tipo de datos a otro tipo de datos.

Sintaxis

CAST ([Expresión] AS Datatype)

El tipo de datos al que está emitiendo una expresión es el tipo de destino. El tipo de datos de la expresión de la que está emitiendo es el tipo de origen.

```
DECLARE @A varchar(2)
DECLARE @B varchar(2)

set @A='25a'
```

```

set @B='15'

Select CAST(@A as int) + CAST(@B as int) as Result
--'25a' is casted to 25 (string to int)
--'15' is casted to 15 (string to int)

--Result
--40

DECLARE @C varchar(2) = 'a'

select CAST(@C as int) as Result
--Result
--Conversion failed when converting the varchar value 'a' to data type int.

```

Lanza el error si falla

Convertir

Cuando convierte expresiones de un tipo a otro, en muchos casos habrá una necesidad dentro de un procedimiento almacenado u otra rutina para convertir datos de un tipo de fecha y hora a un tipo de varchar. La función Convertir se usa para tales cosas. La función CONVERTIR () se puede usar para mostrar datos de fecha / hora en varios formatos. Sintaxis

CONVERTIR (tipo de datos (longitud), expresión, estilo)

Estilo: valores de estilo para la conversión datetime o smalldatetime en datos de caracteres. Agregue 100 a un valor de estilo para obtener un año de cuatro lugares que incluya el siglo (aaaa).

```

select convert (varchar(20),GETDATE(),108)

13:27:16

```

Lea **Convertir tipos de datos en línea**: <https://riptutorial.com/es/sql-server/topic/5034/convertir-tipos-de-datos>

Capítulo 18: Copia de seguridad y restauración de base de datos

Sintaxis

- BACKUP DATABASE la *base de datos* TO *backup_device* [, ... n] WITH *with_options* [, ... o]
- RESTORE DATABASE la *base de datos* DE *backup_device* [, ... n] WITH *with_options* [, ... o]

Parámetros

Parámetro	Detalles
<i>base de datos</i>	El nombre de la base de datos para respaldar o restaurar.
<i>dispositivo de respaldo</i>	El dispositivo para respaldar o restaurar la base de datos, como {DISK o TAPE}. Puede ser separado por comas (,)
<i>con_opciones</i>	Varias opciones que se pueden utilizar mientras se realiza la operación. Al igual que formatear el disco donde se coloca la copia de seguridad o restaurar la base de datos con la opción de reemplazo.

Examples

Copia de seguridad básica en disco sin opciones

El siguiente comando realiza una copia de la base de datos 'Usuarios' a 'D:\db_backup' archivo. Es mejor no dar una extensión.

```
BACKUP DATABASE Users TO DISK = 'D:\DB_Backup'
```

Restauración básica desde disco sin opciones

El siguiente comando restaura la base de datos 'Usuarios' del archivo 'D:\DB_Backup'.

```
RESTORE DATABASE Users FROM DISK = 'D:\DB_Backup'
```

RESTAURAR la base de datos con REEMPLAZAR

Cuando intenta restaurar la base de datos desde otro servidor, puede obtener el siguiente error:

Error 3154: el conjunto de copia de seguridad contiene una copia de seguridad de una base de datos que no es la base de datos existente.

En ese caso, debe usar la opción WITH REPLACE para reemplazar la base de datos con la base de datos de la copia de seguridad:

```
RESTORE DATABASE WWIDW
FROM DISK = 'C:\Backup\WideWorldImportersDW-Full.bak'
WITH REPLACE
```

Incluso en este caso, podría obtener los errores que indican que los archivos no se pueden encontrar en alguna ruta:

Msg 3156, Nivel 16, Estado 3, Línea 1 El archivo 'WWI_Primary' no se puede restaurar a 'D: \ Data \ WideWorldImportersDW.mdf'. Utilice WITH MOVE para identificar una ubicación válida para el archivo.

Este error ocurre probablemente porque sus archivos no se colocaron en la misma ruta de la carpeta que existe en el nuevo servidor. En ese caso, debe mover los archivos de base de datos individuales a una nueva ubicación:

```
RESTORE DATABASE WWIDW
FROM DISK = 'C:\Backup\WideWorldImportersDW-Full.bak'
WITH REPLACE,
MOVE 'WWI_Primary' to 'C:\Data\WideWorldImportersDW.mdf',
MOVE 'WWI_UserData' to 'C:\Data\WideWorldImportersDW_UserData.ndf',
MOVE 'WWI_Log' to 'C:\Data\WideWorldImportersDW.ldf',
MOVE 'WWIDW_InMemory_Data_1' to 'C:\Data\WideWorldImportersDW_InMemory_Data_1'
```

Con esta declaración puede reemplazar la base de datos con todos los archivos de base de datos movidos a una nueva ubicación.

Lea Copia de seguridad y restauración de base de datos en línea: <https://riptutorial.com/es/sql-server/topic/5826/copia-de-seguridad-y-restauracion-de-base-de-datos>

Capítulo 19: CREAR VISTA

Examples

CREAR VISTA

```
CREATE VIEW view_EmployeeInfo
AS
    SELECT EmployeeID,
           FirstName,
           LastName,
           HireDate
    FROM Employee
GO
```

Las filas de las vistas se pueden seleccionar como tablas:

```
SELECT FirstName
FROM view_EmployeeInfo
```

También puede crear una vista con una columna calculada. Podemos modificar la vista anterior de la siguiente manera agregando una columna calculada:

```
CREATE VIEW view_EmployeeReport
AS
    SELECT EmployeeID,
           FirstName,
           LastName,
           Coalesce(FirstName, '') + ' ' + Coalesce(LastName, '') as FullName,
           HireDate
    FROM Employee
GO
```

Esta vista agrega una columna adicional que aparecerá cuando `SELECT` filas de ella. Los valores en esta columna adicional dependerán de los campos `FirstName` y `LastName` en la tabla `Employee` y se actualizarán automáticamente cuando esos campos se actualicen.

CREAR VISTA con cifrado

```
CREATE VIEW view_EmployeeInfo
WITH ENCRYPTION
AS
    SELECT EmployeeID, FirstName, LastName, HireDate
    FROM Employee
GO
```

CREE VISTA CON INNER JOIN

```
CREATE VIEW view_PersonEmployee
```



```

AS
SELECT P.LastName,
       P.FirstName,
       E.JobTitle
FROM Employee AS E
INNER JOIN Person AS P
       ON P.BusinessEntityID = E.BusinessEntityID
GO

```

Las vistas pueden usar uniones para seleccionar datos de numerosas fuentes, como tablas, funciones de tabla o incluso otras vistas. Este ejemplo utiliza las columnas `FirstName` y `LastName` de la tabla `Person` y la columna `JobTitle` de la tabla `Employee`.

Esta vista ahora se puede utilizar para ver todas las filas correspondientes para los administradores en la base de datos:

```

SELECT *
FROM view_PersonEmployee
WHERE JobTitle LIKE '%Manager%'

```

CREAR VISTA indexada

Para crear una vista con un índice, la vista debe crearse con las palabras clave `WITH SCHEMABINDING` :

```

CREATE VIEW view_EmployeeInfo
WITH SCHEMABINDING
AS
SELECT EmployeeID,
       FirstName,
       LastName,
       HireDate
FROM [dbo].Employee
GO

```

Ahora se puede crear cualquier índice agrupado o no agrupado:

```

CREATE UNIQUE CLUSTERED INDEX IX_view_EmployeeInfo
ON view_EmployeeInfo
(
    EmployeeID ASC
)

```

Hay algunas limitaciones para las vistas indexadas:

- La definición de vista puede hacer referencia a una o más tablas en la misma base de datos.
- Una vez que se crea el índice agrupado único, se pueden crear índices no agrupados adicionales en la vista.
- Puede actualizar los datos en las tablas subyacentes, incluidas las inserciones,

actualizaciones, eliminaciones e incluso trunca.

- No puede modificar las tablas y columnas subyacentes. La vista se crea con la opción WITH SCHEMABINDING.
- No puede contener COUNT, MIN, MAX, TOP, uniones externas o algunas otras palabras clave o elementos.

Para obtener más información sobre la creación de vistas indexadas, puede leer este [artículo de MSDN](#)

VISTAS agrupadas

Una vista agrupada se basa en una consulta con una cláusula GROUP BY. Dado que cada uno de los grupos puede tener más de una fila en la base a partir de la cual se creó, estos son necesariamente VIEWS de solo lectura. Tales VIEWS usualmente tienen una o más funciones agregadas y se usan para propósitos de informes. También son útiles para trabajar alrededor de las debilidades en SQL. Considere una VISTA que muestre la venta más grande en cada estado. La consulta es sencilla:

<https://www.simple-talk.com/sql/t-sql-programming/sql-view-beyond-the-basics/>

```
CREATE VIEW BigSales (state_code, sales_amt_total)
AS SELECT state_code, MAX(sales_amt)
   FROM Sales
   GROUP BY state_code;
```

VISTAS de UNION-ed

Las VISTAS basadas en una operación UNION o UNION ALL son de solo lectura porque no hay una única forma de asignar un cambio en una sola fila en una de las tablas base. El operador UNION eliminará filas duplicadas de los resultados. Los operadores UNION y UNION ALL ocultan de qué tabla provienen las filas. Tales VISTAS deben usar a, porque las columnas en UNION [ALL] no tienen nombres propios. En teoría, una UNIÓN de dos tablas desunidas, ninguna de las cuales tiene filas duplicadas en sí misma debería ser actualizable.

<https://www.simple-talk.com/sql/t-sql-programming/sql-view-beyond-the-basics/>

```
CREATE VIEW DepTally2 (emp_nbr, dependent_cnt)
AS (SELECT emp_nbr, COUNT(*)
   FROM Dependents
   GROUP BY emp_nbr)
UNION
(SELECT emp_nbr, 0
   FROM Personnel AS P2
   WHERE NOT EXISTS
     (SELECT *
      FROM Dependents AS D2
      WHERE D2.emp_nbr = P2.emp_nbr));
```

Lea **CREAR VISTA** en línea: <https://riptutorial.com/es/sql-server/topic/3815/crear-vista>

Capítulo 20: Cursores

Sintaxis

- DECLARAR cursor_name CURSOR [LOCAL | GLOBAL]
 - [FORWARD_ONLY | SCROLL]
[ESTÁTICA | KEYSSET | Dinamica | AVANCE RÁPIDO]
[READ_ONLY | SCROLL_LOCKS | OPTIMISTA]
[TYPE_WARNING]
 - PARA select_statement
 - [PARA ACTUALIZACIÓN [OF column_name [, ... n]]]

Observaciones

Normalmente, desearía evitar el uso de cursores, ya que pueden tener un impacto negativo en el rendimiento. Sin embargo, en algunos casos especiales, es posible que deba recorrer su registro de datos por registro y realizar alguna acción.

Examples

Cursor de avance solo básico

Normalmente, desearía evitar el uso de cursores, ya que pueden tener un impacto negativo en el rendimiento. Sin embargo, en algunos casos especiales, es posible que deba recorrer su registro de datos por registro y realizar alguna acción.

```
DECLARE @orderId AS INT

-- here we are creating our cursor, as a local cursor and only allowing
-- forward operations
DECLARE rowCursor CURSOR LOCAL FAST_FORWARD FOR
    -- this is the query that we want to loop through record by record
    SELECT [OrderId]
    FROM [dbo].[Orders]

-- first we need to open the cursor
OPEN rowCursor

-- now we will initialize the cursor by pulling the first row of data, in this example the
[OrderId] column,
-- and storing the value into a variable called @orderId
FETCH NEXT FROM rowCursor INTO @orderId

-- start our loop and keep going until we have no more records to loop through
WHILE @@FETCH_STATUS = 0
BEGIN

    PRINT @orderId
```

```

    -- this is important, as it tells SQL Server to get the next record and store the
    [OrderId] column value into the @orderId variable
    FETCH NEXT FROM rowCursor INTO @orderId

END

-- this will release any memory used by the cursor
CLOSE rowCursor
DEALLOCATE rowCursor

```

Sintaxis del cursor rudimentario

Una simple sintaxis de cursor, que funciona en algunas filas de prueba de ejemplo:

```

/* Prepare test data */
DECLARE @test_table TABLE
(
    Id INT,
    Val VARCHAR(100)
);
INSERT INTO @test_table(Id, Val)
VALUES
    (1, 'Foo'),
    (2, 'Bar'),
    (3, 'Baz');
/* Test data prepared */

/* Iterator variable @myId, for example sake */
DECLARE @myId INT;

/* Cursor to iterate rows and assign values to variables */
DECLARE myCursor CURSOR FOR
    SELECT Id
    FROM @test_table;

/* Start iterating rows */
OPEN myCursor;
FETCH NEXT FROM myCursor INTO @myId;

/* @@FETCH_STATUS global variable will be 1 / true until there are no more rows to fetch */
WHILE @@FETCH_STATUS = 0
BEGIN

    /* Write operations to perform in a loop here. Simple SELECT used for example */
    SELECT Id, Val
    FROM @test_table
    WHERE Id = @myId;

    /* Set variable(s) to the next value returned from iterator; this is needed otherwise the
    cursor will loop infinitely. */
    FETCH NEXT FROM myCursor INTO @myId;
END
/* After all is done, clean up */
CLOSE myCursor;
DEALLOCATE myCursor;

```

Resultados del SSMS. Tenga en cuenta que todas estas son consultas separadas, de ninguna manera están unificadas. Observe cómo el motor de consultas procesa cada iteración una por

una en lugar de como un conjunto.

Carné de identidad	Val
1	Foo
(1 fila (s) afectadas)	
Carné de identidad	Val
2	Bar
(1 fila (s) afectadas)	
Carné de identidad	Val
3	Baz
(1 fila (s) afectadas)	

Lea Cursores en línea: <https://riptutorial.com/es/sql-server/topic/870/cursores>

Capítulo 21: Datos espaciales

Introducción

Hay 2 tipos de datos espaciales

Sistema de coordenadas **Geometría** X / Y para una superficie plana

Geografía Sistema de coordenadas de latitud / longitud para una superficie curva (la tierra). Existen múltiples proyecciones de superficies curvas, por lo que cada espacio geográfico debe permitir a SQL Server saber qué proyección usar. El ID de referencia espacial (SRID) habitual es 4326, que mide distancias en kilómetros. Este es el SRID predeterminado utilizado en la mayoría de los mapas web

Examples

PUNTO

Creas un solo punto. Este será un punto de geometría o geografía según la clase utilizada.

Parámetro	Detalle
Lat o X	Es una expresión flotante que representa la coordenada x del punto que se genera.
Largo o Y	Es una expresión flotante que representa la coordenada y del punto que se genera.
Cuerda	Texto conocido (WKB) de una forma de geometría / geografía
Binario	Binario bien conocido (WKB) de una forma de geometría / geografía
SRID	Es una expresión int que representa el ID de referencia espacial (SRID) de la instancia de geometría / geografía que desea devolver

```
--Explicit constructor
DECLARE @gm1 GEOMETRY = GEOMETRY::Point(10,5,0)

DECLARE @gg1 GEOGRAPHY = GEOGRAPHY::Point(51.511601,-0.096600,4326)

--Implicit constructor (using WKT - Well Known Text)
DECLARE @gm1 GEOMETRY = GEOMETRY::STGeomFromText('POINT(5 10)', 0)

DECLARE @gg1 GEOGRAPHY= GEOGRAPHY::STGeomFromText('POINT(-0.096600 51.511601)', 4326)

--Implicit constructor (using WKB - Well Known Binary)
DECLARE @gm1 GEOMETRY = GEOMETRY::STGeomFromWKB(0x010100000000000000000000000144000000000000002440,
0)
```

```
DECLARE @gg1 GEOGRAPHY= GEOGRAPHY::STGeomFromWKB (0x010100000005F29CB10C7BAB8BFEACC3D247CC14940,
4326)
```

Lea Datos espaciales en línea: <https://riptutorial.com/es/sql-server/topic/6816/datos-espaciales>

Capítulo 22: DBCC

Examples

Comandos de mantenimiento DBCC

Los comandos DBCC permiten al usuario mantener espacio en la base de datos, limpiar cachés, reducir bases de datos y tablas.

Algunos ejemplos son:

```
DBCC DROPCLEANBUFFERS
```

Elimina todos los búferes limpios del grupo de búferes y los objetos de almacén de columnas del grupo de objetos de almacén de columnas.

```
DBCC FREEPROCCACHE  
-- or  
DBCC FREEPROCCACHE (0x060006001ECA270EC0215D050000000000000000000000000000);
```

Elimina todas las consultas SQL en el caché del plan. Cada nuevo plan se volverá a compilar: puede especificar el identificador del plan, el identificador de la consulta para limpiar los planes para el plan de consulta específico o la declaración SQL.

```
DBCC FREESYSTEMCACHE ('ALL', myresourcepool);  
-- or  
DBCC FREESYSTEMCACHE;
```

Limpia todas las entradas en caché creadas por el sistema. Puede limpiar las entradas o = en todo o en un conjunto de recursos especificado (**myresourcepool** en el ejemplo anterior)

```
DBCC FLUSHAUTHCACHE
```

Vacía el caché de autenticación de la base de datos que contiene información sobre los inicios de sesión y las reglas del cortafuegos.

```
DBCC SHRINKDATABASE (MyDB [, 10]);
```

Reduce la base de datos MyDB al 10%. El segundo parámetro es opcional. Puedes usar el ID de la base de datos en lugar del nombre.

```
DBCC SHRINKFILE (DataFile1, 7);
```

Reduce el archivo de datos llamado DataFile1 en la base de datos actual. El tamaño objetivo es de 7 MB (este parámetro es opcional).


```
DBCC CLEANMIRROR (AdventureWorks2012, 'Production.Document', 0)
```

Recupera un espacio de la tabla especificada

Declaraciones de validación DBCC

Los comandos DBCC permiten al usuario validar el estado de la base de datos.

```
ALTER TABLE Table1 WITH NOCHECK ADD CONSTRAINT chkTab1 CHECK (Col1 > 100);  
GO  
DBCC CHECKCONSTRAINTS (Table1);  
--OR  
DBCC CHECKCONSTRAINTS ('Table1.chkTable1');
```

La restricción de verificación se agrega con las opciones nocheck, por lo que no se verificará en los datos existentes. DBCC activará la verificación de restricciones.

Los siguientes comandos DBCC comprueban la integridad de la base de datos, tabla o catálogo:

```
DBCC CHECKTABLE tablename | tableid  
DBCC CHECKDB databasename | dbid  
DBCC CHECKFILEGROUP filegroup_name | filegroup_id | 0  
DBCC CHECKCATALOG databasename1 | database_id1 | 0
```

Declaraciones informativas DBCC

Los comandos DBCC pueden mostrar información sobre objetos de base de datos.

```
DBCC PROCCACHE
```

Muestra información en formato de tabla sobre el caché de procedimientos.

```
DBCC OUTPUTBUFFER ( session_id [ , request_id ])
```

Devuelve el búfer de salida actual en formato hexadecimal y ASCII para el session_id especificado (y request_id opcional).

```
DBCC INPUTBUFFER ( session_id [ , request_id ])
```

Muestra la última declaración enviada desde un cliente a una instancia de Microsoft SQL Server.

```
DBCC SHOW_STATISTICS ( table_or_indexed_view_name , column_statistic_or_index_name)
```

Comandos DBCC Trace

Los indicadores de seguimiento en SQL Server se utilizan para modificar el comportamiento del servidor SQL, activar / desactivar algunas características. Los comandos DBCC pueden controlar los indicadores de seguimiento:

El siguiente ejemplo activa el indicador de traza 3205 globalmente y 3206 para la sesión actual:

```
DBCC TRACEON (3205, -1);  
DBCC TRACEON (3206);
```

El siguiente ejemplo desactiva el indicador de traza 3205 globalmente y 3206 para la sesión actual:

```
DBCC TRACEON (3205, -1);  
DBCC TRACEON (3206);
```

El siguiente ejemplo muestra el estado de los indicadores de seguimiento 2528 y 3205:

```
DBCC TRACESTATUS (2528, 3205);
```

Declaración DBCC

Las sentencias DBCC actúan como comandos de la consola de base de datos para SQL Server. Para obtener la información de sintaxis para el comando DBCC especificado, use la instrucción DBCC HELP (...).

El siguiente ejemplo devuelve todas las declaraciones de DBCC para las que la Ayuda está disponible:

```
DBCC HELP ('?');
```

El siguiente ejemplo devuelve opciones para la declaración DBCC CHECKDB:

```
DBCC HELP ('CHECKDB');
```

Lea DBCC en línea: <https://riptutorial.com/es/sql-server/topic/7316/dbcc>

Capítulo 23: DBMAIL

Sintaxis

- `sp_send_dbmail` [[@profile_name =] 'profile_name'] [, [@recipients =] 'recipients [; ... n] '] [, [@copy_recipients =] 'copy_recipient [; ... n] '] [, [@blind_copy_recipients =] 'blind_copy_recipient [; ... n] '] [, [@from_address =] 'from_address '] [, [@reply_to =] 'reply_to '] [, [@subject =] 'subject '] [, [@body =] 'body '] [, [@body_format =] 'body_format'] [, [@importance =] 'importancia'] [, [@sensitivity =] 'sensibilidad'] [, [@file_attachments =] 'archivo adjunto [; ... n] '] [, [@query =] 'query '] [, [@execute_query_database =] 'execute_query_database '] [, [@attach_query_result_as_file =] 'attach_query_result_as_archivo_ @query_result_header =] query_result_header] [, [@quical_result_wult.width =] query_result_width] [, [@query_result_separator =] 'persona_ascarilla_comultada_gamino_particulo_Personal_Parque_gaminos_Parque_reparador_Pe de la mano:] query_no_truncate] [, [@query_result_no_padding =] @query_result_no_padding] [, [@mailitem_id =] mailitem_id] [OUTPUT]

Examples

Enviar email simple

Este código envía un correo electrónico de solo texto a `destinatario@someaddress.com`

```
EXEC msdb.dbo.sp_send_dbmail
  @profile_name = 'The Profile Name',
  @recipients = 'recipient@someaddress.com',
  @body = 'This is a simple email sent from SQL Server.',
  @subject = 'Simple email'
```

Enviar los resultados de una consulta

Esto adjunta los resultados de la consulta `SELECT * FROM Users` y los envía a `recipient@someaddress.com`

```
EXEC msdb.dbo.sp_send_dbmail
  @profile_name = 'The Profile Name',
  @recipients = 'recipient@someaddress.com',
  @query = 'SELECT * FROM Users',
  @subject = 'List of users',
  @attach_query_result_as_file = 1;
```

Enviar correo electrónico HTML

El contenido HTML se debe pasar a `sp_send_dbmail`

SQL Server 2012

```
DECLARE @html VARCHAR(MAX);
SET @html = CONCAT
(
    '<html><body>',
    '<h1>Some Header Text</h1>',
    '<p>Some paragraph text</p>',
    '</body></html>'
)
```

SQL Server 2012

```
DECLARE @html VARCHAR(MAX);
SET @html =
    '<html><body>' +
    '<h1>Some Header Text</h1>' +
    '<p>Some paragraph text</p>' +
    '</body></html>';
```

Luego usa la variable @html con el @body argument . La cadena HTML también se puede pasar directamente a @body , aunque puede hacer que el código sea más difícil de leer.

```
EXEC msdb.dbo.sp_send_dbmail
    @recipients='recipient@someaddress.com',
    @subject = 'Some HTML content',
    @body = @html,
    @body_format = 'HTML';
```

Lea DBMAIL en línea: <https://riptutorial.com/es/sql-server/topic/4908/dbmail>

Capítulo 24: Declaración de caso

Observaciones

El ejemplo anterior es solo para mostrar la sintaxis del uso de declaraciones de casos en SQL Server con el ejemplo del día de la semana. Aunque se puede lograr el mismo resultado utilizando "SELECT DATENAME (WEEKDAY, GETDATE ())" también.

Examples

Sentencia CASE simple

En una declaración de caso simple, se comprueba un valor o variable contra múltiples respuestas posibles. El siguiente código es un ejemplo de una declaración de caso simple:

```
SELECT CASE DATEPART(WEEKDAY, GETDATE())
  WHEN 1 THEN 'Sunday'
  WHEN 2 THEN 'Monday'
  WHEN 3 THEN 'Tuesday'
  WHEN 4 THEN 'Wednesday'
  WHEN 5 THEN 'Thursday'
  WHEN 6 THEN 'Friday'
  WHEN 7 THEN 'Saturday'
END
```

Búsqueda de sentencia CASE

En una declaración de caso de búsqueda, cada opción puede probar uno o más valores de forma independiente. El siguiente código es un ejemplo de una declaración de caso buscado:

```
DECLARE @FirstName varchar(30) = 'John'
DECLARE @LastName varchar(30) = 'Smith'

SELECT CASE
  WHEN LEFT(@FirstName, 1) IN ('a','e','i','o','u')
    THEN 'First name starts with a vowel'
  WHEN LEFT(@LastName, 1) IN ('a','e','i','o','u')
    THEN 'Last name starts with a vowel'
  ELSE
    'Neither name starts with a vowel'
END
```

Lea Declaración de caso en línea: <https://riptutorial.com/es/sql-server/topic/7238/declaracion-de-caso>

Capítulo 25: Delimitando caracteres especiales y palabras reservadas.

Observaciones

En términos generales, es mejor no usar [palabras reservadas de T-SQL](#) como nombres de tablas, nombres de columnas, nombres de objetos de programación, alias, etc. Por lo tanto, el método para escapar de estas palabras clave solo debe aplicarse si está heredando un diseño de base de datos que no se puede cambiar. .

Para palabras reservadas, el uso de los corchetes no es obligatorio. Al usar una herramienta como SQL Server Management Studio, las palabras reservadas se resaltarán para llamar la atención sobre el hecho de que están reservadas.

Examples

Método básico

El método básico para escapar de las palabras reservadas para SQL Server es el uso de los corchetes ([y]). Por ejemplo, *Descripción* y *Nombre* son palabras reservadas; sin embargo, si hay un objeto que usa ambos como nombres, la sintaxis utilizada es:

```
SELECT [Description]
FROM   dbo.TableName
WHERE  [Name] = 'foo'
```

El único carácter especial para SQL Server es la comilla simple ' y se escapa al duplicar su uso. Por ejemplo, para encontrar el nombre *O'Shea* en la misma tabla, se usará la siguiente sintaxis:

```
SELECT [Description]
FROM   dbo.TableName
WHERE  [Name] = 'O'Shea'
```

Lea [Delimitando caracteres especiales y palabras reservadas.](#) en línea:

<https://riptutorial.com/es/sql-server/topic/7156/delimitando-caracteres-especiales-y-palabras-reservadas->

Capítulo 26: Desencadenar

Introducción

Un disparador es un tipo especial de procedimiento almacenado, que se ejecuta automáticamente después de que ocurre un evento. Hay dos tipos de desencadenadores: desencadenadores de lenguaje de definición de datos y desencadenadores de lenguaje de manipulación de datos.

Normalmente está ligado a una mesa y se dispara automáticamente. No puedes llamar explícitamente a ningún disparador.

Examples

Tipos y clasificaciones de gatillo

En SQL Server, hay dos categorías de desencadenantes: Desencadenadores DDL y Desencadenadores DML.

Los disparadores DDL se activan en respuesta a los eventos del lenguaje de definición de datos (DDL). Estos eventos corresponden principalmente a las instrucciones Transact-SQL que comienzan con las palabras clave `CREATE` , `ALTER` y `DROP` .

Los disparadores de DML se activan en respuesta a los eventos del lenguaje de manipulación de datos (DML). Estos eventos corresponden a las instrucciones Transact-SQL que comienzan con las palabras clave `INSERT` , `UPDATE` y `DELETE` .

Los disparadores de DML se clasifican en dos tipos principales:

1. Después de los disparadores (para los disparadores)

- DESPUÉS DE INSERTAR el gatillo.
- DESPUÉS DE ACTUALIZACIÓN Trigger.
- DESPUÉS DE BORRAR el disparador.

2. En lugar de desencadenantes

- INSTEAD OF INSERT Trigger.
- EN LUGAR DE ACTUALIZAR el disparador.
- INSTEAD DE DELETE Trigger.

Activadores de DML

Los activadores de DML se activan como respuesta a las declaraciones de dml (`insert` , `update` o `delete`).

Se puede crear un activador de dml para tratar uno o más eventos de dml para una sola tabla o vista. Esto significa que un solo activador de dml puede manejar la inserción, actualización y

eliminación de registros de una tabla o vista específica, pero solo puede manejar los datos que se cambian en esa tabla o vista individual.

Los activadores DML brindan acceso a las tablas `inserted` y `deleted` que contienen información sobre los datos que fueron afectados por la inserción, actualización o eliminación de la declaración que activó el activador.

Tenga en cuenta que los activadores DML se basan en sentencias, no en filas. Esto significa que si la declaración efectuó más de una fila, las tablas insertadas o eliminadas contendrán más de una fila.

Ejemplos:

```
CREATE TRIGGER tblSomething_InsertOrUpdate ON tblSomething
FOR INSERT
AS

    INSERT INTO tblAudit (TableName, RecordId, Action)
    SELECT 'tblSomething', Id, 'Inserted'
    FROM Inserted

GO

CREATE TRIGGER tblSomething_InsertOrUpdate ON tblSomething
FOR UPDATE
AS

    INSERT INTO tblAudit (TableName, RecordId, Action)
    SELECT 'tblSomething', Id, 'Updated'
    FROM Inserted

GO

CREATE TRIGGER tblSomething_InsertOrUpdate ON tblSomething
FOR DELETE
AS

    INSERT INTO tblAudit (TableName, RecordId, Action)
    SELECT 'tblSomething', Id, 'Deleted'
    FROM Deleted

GO
```

Todos los ejemplos anteriores agregarán registros a `tblAudit` cada vez que se agregue, borre o actualice un registro en `tblSomething`.

Lea **Desencadenar en línea**: <https://riptutorial.com/es/sql-server/topic/5032/desencadenar>

Capítulo 27: Eliminar palabra clave

Introducción

La palabra clave Drop se puede usar con varios objetos SQL, este tema proporciona ejemplos rápidos de diferentes usos con objetos de base de datos.

Observaciones

Enlaces a MSDN.

- [DROP TABLE \(Transact-SQL\)](#)
- [PROCEDIMIENTO DE GOTA \(Transact-SQL\)](#)
- [BASE DE DATOS DROP \(Transact-SQL\)](#)

Examples

Caer mesas

El comando **DROP TABLE** elimina las definiciones de tabla y todos los datos, índices, activadores, restricciones y permisos relacionados.

Antes de eliminar una tabla, debe verificar si hay algún objeto (vistas, procedimientos almacenados, otras tablas) que haga referencia a la tabla.

No puede eliminar una tabla a la que hace referencia otra tabla mediante FOREIGN KEY. Primero debes soltar la LLAVE EXTERNA que hace referencia a ella.

Puede eliminar una tabla a la que hace referencia una vista o un procedimiento almacenado, pero después de eliminar la tabla, la vista o el procedimiento almacenado ya no se pueden utilizar.

La sintaxis

```
DROP TABLE [ IF EXISTS ] [ database_name . [ schema_name ] . | schema_name . ]  
table_name [ ,...n ] [ ; ]
```

- `IF EXISTS` - Soltar la tabla solo si existe
- `database_name` : especifique el nombre de la base de datos donde se encuentra la tabla
- `schema_name` : especifique el nombre del esquema en el que se encuentra la tabla
- `table_name` : especifique el nombre de la tabla que se va a quitar

Ejemplos

Elimine la tabla con el nombre **TABLE_1** de la base de datos actual y el esquema predeterminado

dbo

```
DROP TABLE Table_1;
```

Elimine la tabla con **TABLE_1** de la base de datos **HR** y el esquema predeterminado **dbo**

```
DROP TABLE HR.Table_1;
```

Elimine la tabla con **TABLE_1** de la base de datos **HR** y el esquema **externo**

```
DROP TABLE HR.external.TABLE_1;
```

Eliminar bases de datos

El comando **DROP DATABASE** elimina un catálogo de base de datos, independientemente de su estado (fuera de línea, solo lectura, sospechoso, etc.) de la instancia actual de SQL Server.

No se puede descartar una base de datos si hay instantáneas de la base de datos asociadas a ella, ya que las instantáneas de la base de datos se deben descartar primero.

Una caída de la base de datos elimina todos los archivos de disco físico (a menos que esté fuera de línea) utilizados por la base de datos a menos que use el Procedimiento almacenado 'sp_detach_db'.

Una caída de la instantánea de la base de datos elimina la instantánea de la instancia de SQL Server y elimina los archivos físicos que también utiliza.

Una base de datos eliminada solo se puede volver a crear restaurando una copia de seguridad (tampoco desde una instantánea de la base de datos).

La sintaxis

```
DROP DATABASE [ IF EXISTS ] { database_name | database_snapshot_name } [ ,...n ] [;]
```

- **IF EXISTS** - Soltar la tabla solo si existe
- **database_name** : especifica el nombre de la base de datos que se eliminará
- **database_snapshot_name** : especifica la instantánea de la base de datos que se eliminará
-

Ejemplos

Eliminar una sola base de datos;

```
DROP DATABASE Database1;
```

Eliminar múltiples bases de datos

```
DROP DATABASE Database1, Database2;
```

Eliminar una instantánea

```
DROP DATABASE Database1_snapshot17;
```

Eliminar si existe la base de datos

```
DROP DATABASE IF EXISTS Database1;
```

Soltar tablas temporales

En el servidor SQL tenemos 2 tipos de tablas temporales:

1. `##GlobalTempTable` es un tipo de tabla temporal que se esconde entre las sesiones de todos los usuarios.
2. `#LocalTempTable` temporal `#LocalTempTable` : es un tipo de tabla temporal que solo existe en el ámbito actual (solo en el proceso real: puede obtener una identificación de su proceso actual mediante `SELECT @@SPID`)

El proceso de eliminación de tablas temporales es el mismo que para la tabla normal:

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

ANTES de SQL Server 2016:

```
IF(OBJECT_ID('tempdb..#TempTable') is not null)  
DROP TABLE #TempTable;
```

SQL Server 2016:

```
DROP TABLE IF EXISTS #TempTable
```

Lea Eliminar palabra clave en línea: <https://riptutorial.com/es/sql-server/topic/9532/eliminar-palabra-clave>

Capítulo 28: En memoria OLTP (Hekaton)

Examples

Crear tabla optimizada de memoria

```
-- Create demo database
CREATE DATABASE SQL2016_Demo
ON PRIMARY
(
    NAME = N'SQL2016_Demo',
    FILENAME = N'C:\Dump\SQL2016_Demo.mdf',
    SIZE = 5120KB,
    FILEGROWTH = 1024KB
)
LOG ON
(
    NAME = N'SQL2016_Demo_log',
    FILENAME = N'C:\Dump\SQL2016_Demo_log.ldf',
    SIZE = 1024KB,
    FILEGROWTH = 10%
)
GO

use SQL2016_Demo
go

-- Add Filegroup by MEMORY_OPTIMIZED_DATA type
ALTER DATABASE SQL2016_Demo
    ADD FILEGROUP MemFG CONTAINS MEMORY_OPTIMIZED_DATA
GO

--Add a file to defined filegroup
ALTER DATABASE SQL2016_Demo ADD FILE
(
    NAME = MemFG_File1,
    FILENAME = N'C:\Dump\MemFG_File1' -- your file path, check directory exist before
executing this code
)
TO FILEGROUP MemFG
GO

--Object Explorer -- check database created
GO

-- create memory optimized table 1
CREATE TABLE dbo.MemOptTable1
(
    Column1 INT NOT NULL,
    Column2 NVARCHAR(4000) NULL,
    SpidFilter SMALLINT NOT NULL DEFAULT (@@spid),

    INDEX ix_SpidFiler NONCLUSTERED (SpidFilter),
    INDEX ix_SpidFilter HASH (SpidFilter) WITH (BUCKET_COUNT = 64),

    CONSTRAINT CHK_soSessionC_SpidFilter
```

```

        CHECK ( SpidFilter = @@spid ),
    )
    WITH
        (MEMORY_OPTIMIZED = ON,
         DURABILITY = SCHEMA_AND_DATA); --or DURABILITY = SCHEMA_ONLY
go

-- create memory optimized table 2
CREATE TABLE MemOptTable2
(
    ID INT NOT NULL PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET_COUNT = 10000),
    FullName NVARCHAR(200) NOT NULL,
    DateAdded DATETIME NOT NULL
) WITH (MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_AND_DATA)
GO

```

Mostrar los archivos .dll y las tablas creadas para las tablas de memoria optimizada

```

SELECT
    OBJECT_ID('MemOptTable1') AS MemOptTable1_ObjectID,
    OBJECT_ID('MemOptTable2') AS MemOptTable2_ObjectID
GO

SELECT
    name,description
FROM sys.dm_os_loaded_modules
WHERE name LIKE '%XTP%'
GO

```

Mostrar todas las tablas de memoria optimizada:

```

SELECT
    name,type_desc,durability_desc,Is_memory_Optimized
FROM sys.tables
WHERE Is_memory_Optimized = 1
GO

```

Tipos de tablas y tablas temporales optimizadas para la memoria

Por ejemplo, este es el tipo de tabla tradicional basado en tempdb:

```

CREATE TYPE dbo.testTableType AS TABLE
(
    col1 INT NOT NULL,
    col2 CHAR(10)
);

```

Para optimizar en memoria este tipo de tabla, simplemente agregue la opción `memory_optimized=on`, y agregue un índice si no hay ninguno en el tipo original:

```

CREATE TYPE dbo.testTableType AS TABLE
(
    col1 INT NOT NULL,

```

```
col2 CHAR(10)
)WITH (MEMORY_OPTIMIZED=ON);
```

La tabla temporal global es así:

```
CREATE TABLE ##tempGlobalTabel
(
    Col1 INT NOT NULL ,
    Col2 NVARCHAR(4000)
);
```

Tabla temporal global optimizada para memoria:

```
CREATE TABLE dbo.tempGlobalTabel
(
    Col1 INT NOT NULL INDEX ix NONCLUSTERED,
    Col2 NVARCHAR(4000)
)
WITH
(MEMORY_OPTIMIZED = ON,
DURABILITY = SCHEMA_ONLY);
```

Para optimizar las tablas temporales de memoria (## temp):

1. Cree una nueva `SCHEMA_ONLY` memoria optimizada `SCHEMA_ONLY` con el mismo esquema que la tabla `##temp` global `##temp`
 - Asegúrese de que la nueva tabla tenga al menos un índice
2. Cambie todas las referencias a `##temp` en sus instrucciones Transact-SQL a la nueva temperatura de tabla optimizada para memoria
3. Reemplace las declaraciones `DROP TABLE ##temp` en su código con `DELETE FROM temp`, para limpiar el contenido
4. Elimine las declaraciones `CREATE TABLE ##temp` de su código; ahora son redundantes

[más información](#)

Declarar variables de tabla optimizadas para la memoria

Para un rendimiento más rápido, puede optimizar la memoria de su variable de tabla. Aquí está el T-SQL para una variable de tabla tradicional:

```
DECLARE @tvp TABLE
(
    col1 INT NOT NULL ,
    Col2 CHAR(10)
);
```

Para definir las variables optimizadas para la memoria, primero debe crear un tipo de tabla optimizada para la memoria y luego declarar una variable a partir de ella:

```
CREATE TYPE dbo.memTypeTable
AS TABLE
```

```
(
    Col1 INT NOT NULL INDEX ix1,
    Col2 CHAR(10)
)
WITH
    (MEMORY_OPTIMIZED = ON);
```

Entonces podemos usar el tipo de tabla así:

```
DECLARE @tvp memTypeTable
insert INTO @tvp
values (1, '1'), (2, '2'), (3, '3'), (4, '4'), (5, '5'), (6, '6')

SELECT * FROM @tvp
```

Resultado:

Col1	Col2
1	1
2	2
3	3
4	4
5	5
6	6

Crear una tabla temporal optimizada para la versión del sistema de memoria

```
CREATE TABLE [dbo].[MemOptimizedTemporalTable]
(
    [BusinessDocNo] [bigint] NOT NULL,
    [ProductCode] [int] NOT NULL,
    [UnitID] [tinyint] NOT NULL,
    [PriceID] [tinyint] NOT NULL,
    [SysStartTime] [datetime2](7) GENERATED ALWAYS AS ROW START NOT NULL,
    [SysEndTime] [datetime2](7) GENERATED ALWAYS AS ROW END NOT NULL,
    PERIOD FOR SYSTEM_TIME ([SysStartTime], [SysEndTime]),

    CONSTRAINT [PK_MemOptimizedTemporalTable] PRIMARY KEY NONCLUSTERED
    (
        [BusinessDocNo] ASC,
        [ProductCode] ASC
    )
)
WITH (
    MEMORY_OPTIMIZED = ON , DURABILITY = SCHEMA_AND_DATA, -- Memory Optimized Option ON
    SYSTEM_VERSIONING = ON (HISTORY_TABLE = [dbo].[MemOptimizedTemporalTable_History] ,
    DATA_CONSISTENCY_CHECK = ON )
)
```

[más información](#)

Lea **En memoria OLTP (Hekaton)** en línea: <https://riptutorial.com/es/sql-server/topic/5295/en-memoria-oltp--hekaton->

Capítulo 29: Enmascaramiento dinámico de datos

Examples

Enmascarar la dirección de correo electrónico utilizando enmascaramiento dinámico

Si tiene una columna de correo electrónico, puede enmascararla con la máscara de correo electrónico ():

```
ALTER TABLE Company
ALTER COLUMN Email ADD MASKED WITH (FUNCTION = 'email()')
```

Cuando el usuario intenta seleccionar correos electrónicos de la tabla de la Compañía, obtendrá algo como los siguientes valores:

mXXX@XXXX.com

zXXX@XXXX.com

rXXX@XXXX.com

Añadir máscara parcial en columna

Puede agregar una máscara parcial en la columna que mostrará algunos caracteres desde el principio y el final de la cadena y mostrará una máscara en lugar de los caracteres en el centro:

```
ALTER TABLE Company
ALTER COLUMN Phone ADD MASKED WITH (FUNCTION = 'partial(5,"XXXXXXX",2)')
```

En los parámetros de la función parcial, puede especificar cuántos valores desde el principio se mostrarán, cuántos valores desde el final se mostrarán y cuál será el patrón que se muestra en el centro.

Cuando el usuario intenta seleccionar correos electrónicos de la tabla de la Compañía, obtendrá algo como los siguientes valores:

(381) XXXXXXXX39

(360) XXXXXXXX01

(415) XXXXXXXX05

Mostrando valores aleatorios del rango usando random () mask

La máscara aleatoria mostrará un número de fondo del rango especificado en lugar del valor real:

```
ALTER TABLE Product
ALTER COLUMN Price ADD MASKED WITH (FUNCTION = 'random(100,200)')
```

Tenga en cuenta que en algunos casos el valor mostrado puede coincidir con el valor real en la columna (si el número seleccionado al azar coincide con el valor en la celda).

Añadiendo máscara por defecto en la columna

Si agrega una máscara predeterminada en la columna, en lugar del valor real en la instrucción SELECT se mostrará la máscara:

```
ALTER TABLE Company
ALTER COLUMN Postcode ADD MASKED WITH (FUNCTION = 'default()')
```

Controlando quién puede ver los datos desenmascarados.

Puede otorgar a los usuarios sin privilegios el derecho de ver los valores sin máscara utilizando la siguiente declaración:

```
GRANT UNMASK TO MyUser
```

Si algún usuario ya tiene permiso para desenmascarar, puede revocar este permiso:

```
REVOKE UNMASK TO MyUser
```

Lea [Enmascaramiento dinámico de datos en línea](https://riptutorial.com/es/sql-server/topic/7052/enmascaramiento-dinamico-de-datos): <https://riptutorial.com/es/sql-server/topic/7052/enmascaramiento-dinamico-de-datos>

Capítulo 30: Esquemas

Examples

Creando un esquema

```
CREATE SCHEMA dvr AUTHORIZATION Owner
    CREATE TABLE sat_Sales (source int, cost int, partid int)
    GRANT SELECT ON SCHEMA :: dvr TO User1
    DENY SELECT ON SCHEMA :: dvr to User 2
GO
```

Alter Schema

```
ALTER SCHEMA dvr
    TRANSFER dbo.tbl_Staging;
GO
```

Esto transferiría la tabla tbl_Staging del esquema dbo al esquema dvr

Esquemas de caída

```
DROP SCHEMA dvr
```

Propósito

El esquema se refiere a las tablas de una base de datos específica y cómo se relacionan entre sí. Proporciona un plano organizativo de cómo se construye la base de datos. Los beneficios adicionales de la implementación de esquemas de base de datos es que los esquemas se pueden usar como un método que restringe / otorga acceso a tablas específicas dentro de una base de datos.

Lea Esquemas en línea: <https://riptutorial.com/es/sql-server/topic/5806/esquemas>

Capítulo 31: Exportar datos en el archivo txt utilizando SQLCMD

Sintaxis

- sqlcmd -S SHERAZM-E7450 \ SQL2008R2 -d Baseline_DB_Aug_2016 -o c: \ employee.txt -Q "seleccionar * del empleado"

Examples

Mediante el uso de SQLCMD en el símbolo del sistema

La estructura de comando es

```
sqlcmd -S yourservername \ instancename -d database_name -o outputfilename_withpath -Q "su consulta de selección"
```

Los interruptores son los siguientes

-S para servername y nombre de instancia

-d para la base de datos fuente

-o para el archivo de salida de destino (creará un archivo de salida)

-Q por consulta para recuperar datos

Lea [Exportar datos en el archivo txt utilizando SQLCMD en línea](https://riptutorial.com/es/sql-server/topic/7076/exportar-datos-en-el-archivo-txt-utilizando-sqlcmd): <https://riptutorial.com/es/sql-server/topic/7076/exportar-datos-en-el-archivo-txt-utilizando-sqlcmd>

Capítulo 32: Expresiones de mesa comunes

Sintaxis

- `CON cte_name [(column_name_1, column_name_2, ...)] AS (cte_expression)`

Observaciones

Es necesario separar un CTE de la declaración anterior con un carácter de punto y coma (;).

es decir `WITH CommonTableName (...) SELECT ... FROM CommonTableName ...`

El alcance de un CTE es un solo lote, y solo en sentido descendente de su definición. Un lote puede contener varios CTE y un CTE puede hacer referencia a otro CTE definido anteriormente en el lote, pero un CTE no puede hacer referencia a otro CTE que se define más adelante en el lote.

Examples

Jerarquía de empleados

Configuración de la tabla

```
CREATE TABLE dbo.Employees
(
    EmployeeID INT NOT NULL PRIMARY KEY,
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    ManagerID INT NULL
)

GO

INSERT INTO Employees VALUES (101, 'Ken', 'Sánchez', NULL)
INSERT INTO Employees VALUES (102, 'Keith', 'Hall', 101)
INSERT INTO Employees VALUES (103, 'Fred', 'Bloggs', 101)
INSERT INTO Employees VALUES (104, 'Joseph', 'Walker', 102)
INSERT INTO Employees VALUES (105, 'Žydrė', 'Klybė', 101)
INSERT INTO Employees VALUES (106, 'Sam', 'Jackson', 105)
INSERT INTO Employees VALUES (107, 'Peter', 'Miller', 103)
INSERT INTO Employees VALUES (108, 'Chloe', 'Samuels', 105)
INSERT INTO Employees VALUES (109, 'George', 'Weasley', 105)
INSERT INTO Employees VALUES (110, 'Michael', 'Kensington', 106)
```

Expresión de tabla común

```

;WITH cteReports (EmpID, FirstName, LastName, SupervisorID, EmpLevel) AS
(
    SELECT EmployeeID, FirstName, LastName, ManagerID, 1
    FROM Employees
    WHERE ManagerID IS NULL

    UNION ALL

    SELECT e.EmployeeID, e.FirstName, e.LastName, e.ManagerID, r.EmpLevel + 1
    FROM Employees          AS e
    INNER JOIN cteReports AS r ON e.ManagerID = r.EmpID
)

SELECT
    FirstName + ' ' + LastName AS FullName,
    EmpLevel,
    (SELECT FirstName + ' ' + LastName FROM Employees WHERE EmployeeID =
cteReports.SupervisorID) AS ManagerName
FROM cteReports
ORDER BY EmpLevel, SupervisorID

```

Salida:

Nombre completo	EmpLevel	Nombre del gerente
Ken Sánchez	1	<i>nulo</i>
Keith Hall	2	Ken Sánchez
Fred Bloggs	2	Ken Sánchez
Žydre Klybe	2	Ken Sánchez
Joseph walker	3	Keith Hall
Peter Miller	3	Fred Bloggs
Sam Jackson	3	Žydre Klybe
Chloe Samuels	3	Žydre Klybe
George Weasley	3	Žydre Klybe
Michael Kensington	4	Sam Jackson

Encuentra el enésimo salario más alto usando CTE

Tabla de empleados:

```

| ID | FirstName | LastName | Gender | Salary |
+----+-----+-----+-----+-----+

```

1	Jahangir	Alam	Male	70000
2	Arifur	Rahman	Male	60000
3	Oli	Ahammed	Male	45000
4	Sima	Sultana	Female	70000
5	Sudeepta	Roy	Male	80000

CTE (expresión de tabla común):

```
WITH RESULT AS
(
  SELECT SALARY,
         DENSE_RANK() OVER (ORDER BY SALARY DESC) AS DENSERANK
  FROM EMPLOYEES
)
SELECT TOP 1 SALARY
FROM RESULT
WHERE DENSERANK = 1
```

Para encontrar el segundo salario más alto, simplemente reemplace N por 2. Del mismo modo, para encontrar el tercer salario más alto, simplemente reemplace N por 3.

Eliminar filas duplicadas utilizando CTE

Tabla de empleados:

ID	FirstName	LastName	Gender	Salary
1	Mark	Hastings	Male	60000
1	Mark	Hastings	Male	60000
2	Mary	Lambeth	Female	30000
2	Mary	Lambeth	Female	30000
3	Ben	Hoskins	Male	70000
3	Ben	Hoskins	Male	70000
3	Ben	Hoskins	Male	70000

CTE (expresión de tabla común):

```
WITH EmployeesCTE AS
(
  SELECT *, ROW_NUMBER() OVER (PARTITION BY ID ORDER BY ID) AS RowNumber
  FROM Employees
)
DELETE FROM EmployeesCTE WHERE RowNumber > 1
```

Resultado de la ejecución:

ID	FirstName	LastName	Gender	Salary
1	Mark	Hastings	Male	60000
2	Mary	Lambeth	Female	30000
3	Ben	Hoskins	Male	70000

Generar una tabla de fechas utilizando CTE

```
DECLARE @startdate CHAR(8), @numberDays TINYINT

SET @startdate = '20160101'
SET @numberDays = 10;

WITH CTE_DatesTable
AS
(
    SELECT CAST(@startdate as date) AS [date]
    UNION ALL
    SELECT DATEADD(dd, 1, [date])
    FROM CTE_DatesTable
    WHERE DATEADD(dd, 1, [date]) <= DateAdd(DAY, @numberDays-1, @startdate)
)

SELECT [date] FROM CTE_DatesTable

OPTION (MAXRECURSION 0)
```

Este ejemplo devuelve una tabla de fechas de una sola columna, comenzando con la fecha especificada en la variable @startdate y devolviendo el siguiente @numberDays valor de fechas.

CTE recursivo

Este ejemplo muestra cómo obtener cada año desde este año hasta 2011 (2012 - 1).

```
WITH yearsAgo
(
    myYear
)
AS
(
    -- Base Case: This is where the recursion starts
    SELECT DATEPART(year, GETDATE()) AS myYear

    UNION ALL -- This MUST be UNION ALL (cannot be UNION)

    -- Recursive Section: This is what we're doing with the recursive call
    SELECT yearsAgo.myYear - 1
    FROM yearsAgo
    WHERE yearsAgo.myYear >= 2012
)

SELECT myYear FROM yearsAgo; -- A single SELECT, INSERT, UPDATE, or DELETE
```

mi año

2016

2015

2014

2013

mi año

2012

2011

Puede controlar la recursión (piense el desbordamiento de pila en el código) con `MAXRECURSION` como una opción de consulta que limitará el número de llamadas recursivas.

```
WITH yearsAgo
(
    myYear
)
AS
(
    -- Base Case
    SELECT DATEPART(year , GETDATE()) AS myYear
    UNION ALL
    -- Recursive Section
    SELECT yearsAgo.myYear - 1
    FROM yearsAgo
    WHERE yearsAgo.myYear >= 2002
)
SELECT * FROM yearsAgo
OPTION (MAXRECURSION 10);
```

Mensaje 530, nivel 16, estado 1, línea 2 La declaración terminó. La recursión máxima 10 se ha agotado antes de completar la declaración.

CTE con múltiples declaraciones AS

```
;WITH cte_query_1
AS
(
    SELECT *
    FROM database.table1
),
cte_query_2
AS
(
    SELECT *
    FROM database.table2
)
SELECT *
FROM cte_query_1
WHERE cte_query_one.fk IN
(
    SELECT PK
    FROM cte_query_2
)
```

Con las expresiones de tabla comunes, es posible crear múltiples consultas utilizando instrucciones `AS` separadas por comas. Una consulta puede hacer referencia a cualquiera o todas esas consultas de muchas maneras diferentes, incluso uniéndolas.

Lea Expresiones de mesa comunes en línea: <https://riptutorial.com/es/sql-server/topic/1343/expresiones-de-mesa-comunes>

Capítulo 33: fechas

Sintaxis

- EOMONTH (*fecha_inicial* [, *month_to_add*])

Observaciones

`DateTime` <https://msdn.microsoft.com/en-us/library/ms187819.aspx> , los `DateTime` s solo son precisos a 3ms.

Redondeo de los valores de `datetime` de Fractional Second Precision se redondean a incrementos de .000, .003 o .007 segundos, como se muestra en la siguiente tabla.

Valor especificado por el usuario	Valor almacenado del sistema
01/01/98 23: 59: 59.999	1998-01-02 00: 00: 00.000
-----	-----
01/01/98 23: 59: 59.995	1998-01-01 23: 59: 59.997
01/01/98 23: 59: 59.996	
01/01/98 23: 59: 59.997	
01/01/98 23: 59: 59.998	
-----	-----
01/01/98 23: 59: 59.992	1998-01-01 23: 59: 59.993
01/01/98 23: 59: 59.993	
01/01/98 23: 59: 59.994	
-----	-----
01/01/98 23: 59: 59.990	1998-01-01 23: 59: 59.990
01/01/98 23: 59: 59.991	
-----	-----

Si se requiere más precisión, se debe usar `time` , `datetime2` o `datetimeoffset` .

Examples

Formato de fecha y hora utilizando CONVERT

Puede usar la función CONVERTIR para convertir un tipo de datos de fecha y hora en una cadena con formato.

```
SELECT GETDATE() AS [Result] -- 2016-07-21 07:56:10.927
```

También puede usar algunos códigos incorporados para convertirlos en un formato específico. Aquí están las opciones integradas en SQL Server:

```
DECLARE @convert_code INT = 100 -- See Table Below  
SELECT CONVERT(VARCHAR(30), GETDATE(), @convert_code) AS [Result]
```

@convert_code	Resultado
100	"21 de julio de 2016 7:56 a.m."
101	"21/07/2016"
102	"2016.07.21"
103	"21/07/2016"
104	"21.07.2016"
105	"21-07-2016"
106	"21 jul 2016"
107	"21 de julio de 2016"
108	"07:57:05"
109	"21 de julio de 2016 7: 57: 45: 707AM"
110	"07-21-2016"
111	"2016/07/21"
112	"20160721"
113	"21 jul 2016 07: 57: 59: 553"
114	"07: 57: 59: 553"
120	"2016-07-21 07:57:59"

@convert_code	Resultado
121	"2016-07-21 07: 57: 59.553"
126	"2016-07-21T07: 58: 34.340"
127	"2016-07-21T07: 58: 34.340"
130	"16? 1437 7: 58: 34: 340AM"
131	"16/10/1437 7: 58: 34: 340AM"

```

SELECT GETDATE() AS [Result] -- 2016-07-21 07:56:10.927
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),100) AS [Result] -- Jul 21 2016 7:56AM
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),101) AS [Result] -- 07/21/2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),102) AS [Result] -- 2016.07.21
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),103) AS [Result] -- 21/07/2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),104) AS [Result] -- 21.07.2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),105) AS [Result] -- 21-07-2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),106) AS [Result] -- 21 Jul 2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),107) AS [Result] -- Jul 21, 2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),108) AS [Result] -- 07:57:05
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),109) AS [Result] -- Jul 21 2016 7:57:45:707AM
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),110) AS [Result] -- 07-21-2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),111) AS [Result] -- 2016/07/21
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),112) AS [Result] -- 20160721
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),113) AS [Result] -- 21 Jul 2016 07:57:59:553
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),114) AS [Result] -- 07:57:59:553
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),120) AS [Result] -- 2016-07-21 07:57:59
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),121) AS [Result] -- 2016-07-21 07:57:59.553
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),126) AS [Result] -- 2016-07-21T07:58:34.340
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),127) AS [Result] -- 2016-07-21T07:58:34.340
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),130) AS [Result] -- 16 ???? 1437 7:58:34:340AM
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),131) AS [Result] -- 16/10/1437 7:58:34:340AM

```

Formato de fecha y hora usando FORMATO

SQL Server 2012

Puede utilizar la nueva función: [FORMAT \(\)](#) .

Usando esto, puede transformar sus campos `DATETIME` a su propio formato `VARCHAR` personalizado.

Ejemplo

```

DECLARE @Date DATETIME = '2016-09-05 00:01:02.333'

SELECT FORMAT(@Date, N'dddd, MMMM dd, yyyy hh:mm:ss tt')

```

Lunes, 05 de septiembre de 2016 12:01:02 a.m.

Argumentos

Dado que el `DATETIME` se formatea es el `2016-09-05 00:01:02.333` , el siguiente cuadro muestra cuál

sería su salida para el argumento proporcionado.

Argumento	Salida
aaaa	2016
yy	dieciséis
MMMM	septiembre
MM	09
METRO	9
dddd	lunes
ddd	Lun
dd	05
re	5
S.S	00
H	0
S.S	12
h	12
mm	01
metro	1
ss	02
s	2
tt	A.M
t	UNA
fff	333
ff	33
F	3

También puede proporcionar un único argumento a la función `FORMAT()` para generar una salida con formato previo:

```
DECLARE @Date DATETIME = '2016-09-05 00:01:02.333'
```

```
SELECT FORMAT(@Date, N'U')
```

Lunes, 05 de septiembre de 2016 4:01:02 AM

Argumento único	Salida
re	Lunes 05 de septiembre de 2016
re	5/5/2016
F	Lunes, 05 de septiembre de 2016 12:01:02 a.m.
F	Lunes 05 de septiembre de 2016 a las 12:01 a.m.
sol	5/5/2016 12:01:02 AM
sol	5/5/2016 12:01 AM
METRO	Septiembre 05
O	2016-09-05T00: 01: 02.3330000
R	Lun, 05 Sep 2016 00:01:02 GMT
s	2016-09-05T00: 01: 02
T	12:01:02 a.m.
t	12:01 a.m.
U	Lunes, 05 de septiembre de 2016 4:01:02 AM
tu	2016-09-05 00: 01: 02Z
Y	Septiembre 2016

Nota: la lista anterior utiliza la cultura en-US . Se puede especificar una cultura diferente para el FORMAT () través del tercer parámetro:

```
DECLARE @Date DATETIME = '2016-09-05 00:01:02.333'
```

```
SELECT FORMAT(@Date, N'U', 'zh-cn')
```

2016 9 5 4:01:02

Obtener el DateTime actual

Las funciones integradas [GETDATE](#) y [GETUTCDATE](#) devuelven la fecha y la hora actuales sin un

desplazamiento de zona horaria.

El valor de retorno de ambas funciones se basa en el sistema operativo del equipo en el que se ejecuta la instancia de SQL Server.

El valor de retorno de GETDATE representa la hora actual en la misma zona horaria que el sistema operativo. El valor de retorno de GETUTCDATE representa la hora UTC actual.

Cualquiera de las funciones puede incluirse en la cláusula `SELECT` de una consulta o como parte de una expresión booleana en la cláusula `WHERE` .

Ejemplos:

```
-- example query that selects the current time in both the server time zone and UTC
SELECT GETDATE() as SystemDateTime, GETUTCDATE() as UTCDateTime

-- example query records with EventDate in the past.
SELECT * FROM MyEvents WHERE EventDate < GETDATE()
```

Hay algunas otras funciones integradas que devuelven diferentes variaciones de la fecha y hora actual:

```
SELECT
    GETDATE(),           --2016-07-21 14:27:37.447
    GETUTCDATE(),       --2016-07-21 18:27:37.447
    CURRENT_TIMESTAMP,  --2016-07-21 14:27:37.447
    SYSDATETIME(),      --2016-07-21 14:27:37.4485768
    SYSDATETIMEOFFSET(), --2016-07-21 14:27:37.4485768 -04:00
    SYSUTCDATETIME()    --2016-07-21 18:27:37.4485768
```

DATEADD para sumar y restar períodos de tiempo

Sintaxis general:

```
DATEADD (datepart , number , datetime_expr)
```

Para agregar una medida de tiempo, el `number` debe ser positivo. Para restar una medida de tiempo, el `number` debe ser negativo.

Ejemplos

```
DECLARE @now DATETIME2 = GETDATE();
SELECT @now;           --2016-07-21 14:39:46.4170000
SELECT DATEADD(YEAR, 1, @now)  --2017-07-21 14:39:46.4170000
SELECT DATEADD(QUARTER, 1, @now) --2016-10-21 14:39:46.4170000
SELECT DATEADD(WEEK, 1, @now)  --2016-07-28 14:39:46.4170000
SELECT DATEADD(DAY, 1, @now)   --2016-07-22 14:39:46.4170000
SELECT DATEADD(HOUR, 1, @now)  --2016-07-21 15:39:46.4170000
SELECT DATEADD(MINUTE, 1, @now) --2016-07-21 14:40:46.4170000
SELECT DATEADD(SECOND, 1, @now) --2016-07-21 14:39:47.4170000
SELECT DATEADD(MILLISECOND, 1, @now) --2016-07-21 14:39:46.4180000
```

NOTA: `DATEADD` también acepta abreviaturas en el parámetro `datepart` . El uso de estas abreviaturas generalmente se desaconseja ya que pueden ser confusos (`m` vs `mi` , `ww` vs `w` , etc.).

Referencia de piezas de fecha

Estos son los `datepart` valores disponibles para las funciones de fecha y hora:

fecha parcial	Abreviaturas
año	yy, yyyy
trimestre	qq q
mes	mm m
día de año	dy, y
día	dd, d
semana	wk ww
día laborable	dw, w
hora	S.S
minuto	mi, n
segundo	ss, s
milisegundo	Sra
microsegundo	mcs
nanosegundo	ns

NOTA : El uso de abreviaturas generalmente se desaconseja ya que pueden ser confusos (`m` vs `mi` , `ww` vs `w` , etc.). La versión larga de la representación de la fecha `datepart` promueve la claridad y la legibilidad, y debe usarse siempre que sea posible (`month` , `minute` , `week` , `weekday` , etc.).

DATEDIFF para calcular las diferencias de período de tiempo

Sintaxis general:

```
DATEDIFF (datepart, datetime_expr1, datetime_expr2)
```

Devolverá un número positivo si `datetime_expr` está en el pasado en relación con `datetime_expr2` , y un número negativo de lo contrario.

Ejemplos


```

DECLARE @now DATETIME2 = GETDATE();
DECLARE @oneYearAgo DATETIME2 = DATEADD(YEAR, -1, @now);
SELECT @now --2016-07-21 14:49:50.9800000
SELECT @oneYearAgo --2015-07-21 14:49:50.9800000
SELECT DATEDIFF(YEAR, @oneYearAgo, @now) --1
SELECT DATEDIFF(QUARTER, @oneYearAgo, @now) --4
SELECT DATEDIFF(WEEK, @oneYearAgo, @now) --52
SELECT DATEDIFF(DAY, @oneYearAgo, @now) --366
SELECT DATEDIFF(HOUR, @oneYearAgo, @now) --8784
SELECT DATEDIFF(MINUTE, @oneYearAgo, @now) --527040
SELECT DATEDIFF(SECOND, @oneYearAgo, @now) --31622400

```

NOTA: `DATEDIFF` también acepta abreviaturas en el parámetro `datepart`. El uso de estas abreviaturas generalmente se desaconseja ya que pueden ser confusos (`m` vs `mi`, `ww` vs `w`, etc.).

`DATEDIFF` también se puede usar para determinar el desplazamiento entre UTC y la hora local del servidor SQL. La siguiente declaración se puede usar para calcular el desplazamiento entre UTC y la hora local (incluida la zona horaria).

```
select DATEDIFF(hh, getutcdate(), getdate()) as 'CentralTimeOffset'
```

DATEPART & DATENAME

`DATEPART` devuelve la `datepart` de fecha especificada de la expresión `datetime` especificada como un valor numérico.

`DATENAME` devuelve una cadena de caracteres que representa la `datepart` de la fecha especificada de la fecha especificada. En la práctica, `DATENAME` es útil para obtener el nombre del mes o el día de la semana.

También hay algunas funciones de la taquigrafía para conseguir el año, mes o día de una expresión de fecha y hora, que se comportan como `DATEPART` con sus respectivos `datepart` unidades.

Sintaxis:

```

DATEPART ( datepart , datetime_expr )
DATENAME ( datepart , datetime_expr )
DAY ( datetime_expr )
MONTH ( datetime_expr )
YEAR ( datetime_expr )

```

Ejemplos:

```

DECLARE @now DATETIME2 = GETDATE();
SELECT @now --2016-07-21 15:05:33.8370000
SELECT DATEPART(YEAR, @now) --2016
SELECT DATEPART(QUARTER, @now) --3
SELECT DATEPART(WEEK, @now) --30
SELECT DATEPART(HOUR, @now) --15
SELECT DATEPART(MINUTE, @now) --5
SELECT DATEPART(SECOND, @now) --33

```

```

-- Differences between DATEPART and DATENAME:
SELECT DATEPART(MONTH, @now)      --7
SELECT DATENAME(MONTH, @now)     --July
SELECT DATEPART(WEEKDAY, @now)   --5
SELECT DATENAME(WEEKDAY, @now)   --Thursday
--shorthand functions
SELECT DAY(@now)                 --21
SELECT MONTH(@now)              --7
SELECT YEAR(@now)                --2016

```

NOTA: `DATEPART` y `DATENAME` también aceptan abreviaturas en el parámetro `datepart`. El uso de estas abreviaturas generalmente se desaconseja ya que pueden ser confusos (`m` vs `mi`, `ww` vs `w`, etc.).

Consiguiendo el último día de un mes.

Usando las funciones `DATEADD` y `DATEDIFF`, es posible devolver la última fecha de un mes.

```

SELECT DATEADD(d, -1, DATEADD(m, DATEDIFF(m, 0, '2016-09-23') + 1, 0))
-- 2016-09-30 00:00:00.000

```

SQL Server 2012

La función `EOMONTH` proporciona una forma más concisa de devolver la última fecha de un mes y tiene un parámetro opcional para compensar el mes.

```

SELECT EOMONTH('2016-07-21')      --2016-07-31
SELECT EOMONTH('2016-07-21', 4)  --2016-11-30
SELECT EOMONTH('2016-07-21', -5) --2016-02-29

```

Devuelve solo la fecha de un DateTime

Hay muchas formas de devolver una fecha desde un objeto `DateTime`

1. `SELECT CONVERT(Date, GETDATE())`
2. `SELECT DATEADD(dd, 0, DATEDIFF(dd, 0, GETDATE()))` devuelve 2016-07-21 00: 00: 00.000
3. `SELECT CAST(GETDATE() AS DATE)`
4. `SELECT CONVERT(CHAR(10), GETDATE(), 111)`
5. `SELECT FORMAT(GETDATE(), 'yyyy-MM-dd')`

Tenga en cuenta que las opciones 4 y 5 devuelven una cadena, no una fecha.

Crear una función para calcular la edad de una persona en una fecha específica

Esta función tomará 2 parámetros de fecha y hora, el `DOB` y una fecha para verificar la edad en

```

CREATE FUNCTION [dbo].[Calc_Age]
(
    @DOB datetime , @calcDate datetime
)

```

```

    RETURNS int
    AS
    BEGIN
declare @age int

IF (@calcDate < @DOB )
RETURN -1

-- If a DOB is supplied after the comparison date, then return -1
SELECT @age = YEAR(@calcDate) - YEAR(@DOB) +
    CASE WHEN DATEADD (year, YEAR(@calcDate) - YEAR(@DOB)
, @DOB) > @calcDate THEN -1 ELSE 0 END

RETURN @age

END

```

por ejemplo, para comprobar la edad actual de alguien nacido el 1/1/2000

```
SELECT dbo.Calc_Age ('2000-01-01', Getdate ())
```

PLATAFORMA TRANSVERSAL FECHA OBJETO

SQL Server 2012

En Transact SQL, puede definir un objeto como `Date` (o `Date` y `DateTime`) utilizando la función `[DATEFROMPARTS] [1]` (o `[DATETIMEFROMPARTS] [1]`) de la siguiente manera:

```

DECLARE @myDate DATE=DATEFROMPARTS (1988,11,28)
DECLARE @someMoment DATETIME=DATEFROMPARTS (1988,11,28,10,30,50,123)

```

Los parámetros que proporciona son Año, Mes, Día para la función `DATEFROMPARTS` y, para la función `DATETIMEFROMPARTS` , deberá proporcionar el año, mes, día, hora, minutos, segundos y milisegundos.

Estos métodos son útiles y vale la pena usarlos porque el uso de la cadena simple para crear una fecha (o fecha y hora) puede fallar según la región del equipo host, la ubicación o la configuración del formato de fecha.

Formato de fecha extendido

Formato de fecha	Declaración SQL	Salida de muestra
YY-MM-DD	SELECCIONAR DERECHO (CONVERTIR (VARCHAR (10), SYSDATETIME (), 20), 8) AS [YY-MM-DD] SELECCIONAR REEMPLAZAR (CONVERTIR (VARCHAR (8), SYSDATETIME (), 11), '/', '-')	11-06-08

Formato de fecha	Declaración SQL	Salida de muestra
	COMO [YY-MM-DD]	
YYYY-MM-DD	SELECCIONAR CONVERTIR (VARCHAR (10), SYSDATETIME (), 120) COMO [YYYY-MM-DD] SELECCIONAR REEMPLAZAR (CONVERTIR (VARCHAR (10), SYSDATETIME (), 111), '/', '-') COMO [YYYY-MM-DD]	2011-06-08
YYYY-MD	SELECCIONAR CAST (AÑO (SYSDATETIME ()) COMO VARCHAR (4)) + '-' + CAST (MES (SYSDATETIME ()) COMO VARCHAR (2)) + '-' + CAST (DÍA (SYSDATETIME ()) COMO VARCHAR (2)) COMO [YYYY-MD]	2011-6-8
YY-MD	SELECCIONAR A LA DERECHA (CAST (SYSDATETIME ()) COMO VARCHAR (4)), 2) + '-' + CAST (MES (SYSDATETIME ()) COMO VARCHAR (2)) + '-' + CAST (DÍA (SYSDATETIME ()) COMO VARCHAR (2)) COMO [YY-MD]	11-6-8
MD-YYYYY	SELECCIONAR CAST (MES (SYSDATETIME ()) COMO VARCHAR (2)) + '-' + CAST (DÍA (SYSDATETIME ()) COMO VARCHAR (2)) + '-' + CAST (AÑO (SYSDATETIME ()) COMO VARCHAR (4)) AS [MD-YYYYY]	6-8-2011
MD-YY	SELECCIONAR CAST (MES (SYSDATETIME ()) COMO VARCHAR (2)) + '-' + CAST (DÍA (SYSDATETIME ()) COMO VARCHAR (2)) + '-' + DERECHA (CAST (AÑO (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [MD-YY]	6-8-11
DM-YYYYY	SELECCIONAR CAST (DÍA (SYSDATETIME ()) COMO VARCHAR (2)) + '-' + CAST (MES (SYSDATETIME ()) COMO VARCHAR (2)) + '-' + CAST (AÑO (SYSDATETIME ()) COMO VARCHAR (4)) AS [DM-YYYYY]	8-6-2011
DM-YY	SELECCIONAR CAST (DÍA (SYSDATETIME ()) COMO VARCHAR (2)) + '-' + CAST (MES (SYSDATETIME ()) COMO VARCHAR (2)) + '-' + DERECHA (CAST (AÑO (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [DM-YY]	8-6-11
YY-MM	SELECCIONAR DERECHO (CONVERTIR	11-06

Formato de fecha	Declaración SQL	Salida de muestra
	(VARCHAR (7), SYSDATETIME (), 20), 5) AS [YY-MM] SELECCIÓN DE SUBSTRING (CONVERTIR (VARCHAR (10), SYSDATETIME (), 120), 3, 5) AS [YY-MM]	
YYYY-MM	SELECCIONAR CONVERTIR (VARCHAR (7), SYSDATETIME (), 120) COMO [YYYY-MM]	2011-06
YY-M	SELECCIONAR DERECHO (CAST (AÑO (SYSDATETIME ())) COMO VARCHAR (4)), 2) + '-' + CAST (MONTH (SYSDATETIME ())) COMO VARCHAR (2)) AS [YY-M]	11-6
YYYY-M	SELECCIONAR CAST (AÑO (SYSDATETIME ())) COMO VARCHAR (4)) + '-' + CAST (MES (SYSDATETIME ())) COMO VARCHAR (2)) AS [YYYY-M]	2011-6
MM-YY	SELECCIONAR DERECHO (CONVERTIR (VARCHAR (8), SYSDATETIME (), 5), 5) AS [MM-YY] SELECCIÓN DE SUBSTRING (CONVERTIR (VARCHAR (8), SYSDATETIME (), 5), 4, 5) AS [MM-YY]	06-11
MM-YYYYY	SELECCIONAR DERECHO (CONVERTIR (VARCHAR (10), SYSDATETIME (), 105), 7) AS [MM-YYYYY]	06-2011
M-YY	SELECCIONAR CAST (MES (SYSDATETIME ())) COMO VARCHAR (2)) + '-' + DERECHO (CAST (AÑO (SYSDATETIME ())) COMO VARCHAR (4)), 2) AS [M-YY]	6-11
M-YYYY	SELECCIONAR CAST (MES (SYSDATETIME ())) COMO VARCHAR (2)) + '-' + CAST (AÑO (SYSDATETIME ())) COMO VARCHAR (4)) AS [M-YYYY]	6-2011
MM-DD	SELECCIONAR CONVERTIR (VARCHAR (5), SYSDATETIME (), 10) AS [MM-DD]	06-08
DD-MM	SELECCIONAR CONVERTIR (VARCHAR (5), SYSDATETIME (), 5) COMO [DD-MM]	08-06

Formato de fecha	Declaración SQL	Salida de muestra
Maryland	SELECCIONAR CAST (MES (SYSDATETIME ())) COMO VARCHAR (2)) + '-' + CAST (DÍA (SYSDATETIME ())) COMO VARCHAR (2)) AS [MD]	6-8
DM	SELECCIONAR CAST (DÍA (SYSDATETIME ())) COMO VARCHAR (2)) + '-' + CAST (MES (SYSDATETIME ())) COMO VARCHAR (2)) AS [DM]	8-6
M / D / YYYY	SELECCIONAR CAST (MES (SYSDATETIME ())) COMO VARCHAR (2)) + '/' + CAST (DÍA (SYSDATETIME ())) COMO VARCHAR (2)) + '/' + CAST (AÑO (SYSDATETIME ())) COMO VARCHAR (4) AS [M / D / YYYY]	8/8/2011
M / D / YY	SELECCIONAR CAST (MES (SYSDATETIME ())) COMO VARCHAR (2)) + '/' + CAST (DÍA (SYSDATETIME ())) COMO VARCHAR (2)) + '/' + DERECHA (CAST (AÑO (SYSDATETIME ())) AS VARCHAR (4)), 2) AS [M / D / YY]	8/8/11
D / M / YYYY	SELECCIONAR CAST (DÍA (SYSDATETIME ())) COMO VARCHAR (2)) + '/' + CAST (MES (SYSDATETIME ())) COMO VARCHAR (2)) + '/' + CAST (AÑO (SYSDATETIME ())) COMO VARCHAR (4) AS [D / M / YYYY]	8/6/2011
D / M / YY	SELECCIONAR CAST (DÍA (SYSDATETIME ())) COMO VARCHAR (2)) + '/' + CAST (MES (SYSDATETIME ())) COMO VARCHAR (2)) + '/' + DERECHA (CAST (AÑO (SYSDATETIME ())) AS VARCHAR (4)), 2) AS [D / M / YY]	8/6/11
YYYY / M / D	SELECCIONAR CAST (AÑO (SYSDATETIME ())) COMO VARCHAR (4)) + '/' + CAST (MES (SYSDATETIME ())) COMO VARCHAR (2)) + '/' + CAST (DÍA (SYSDATETIME ())) COMO VARCHAR (2) AS [YYYY / M / D]	2011/6/8
YY / M / D	SELECCIONAR A LA DERECHA (CAST (SYSDATETIME ())) COMO VARCHAR (4)), 2) + '/' + CAST (MES (SYSDATETIME ())) COMO VARCHAR (2)) + '/' + CAST (DÍA (SYSDATETIME ())) COMO VARCHAR (2)) COMO [YY / M / D]	11/6/8

Formato de fecha	Declaración SQL	Salida de muestra
MM / YY	SELECCIONAR DERECHO (CONVERTIR (VARCHAR (8), SYSDATETIME (), 3), 5) AS [MM / YY]	06/11
MM / YYYY	SELECCIONAR DERECHO (CONVERTIR (VARCHAR (10), SYSDATETIME (), 103), 7) AS [MM / YYYY]	06/2011
M / YY	SELECCIONAR CAST (MES (SYSDATETIME ()) COMO VARCHAR (2)) + '/' + DERECHO (CAST (AÑO (SYSDATETIME ()) COMO VARCHAR (4)), 2) AS [M / YY]	6/11
M / YYYY	SELECCIONAR CAST (MES (SYSDATETIME ()) COMO VARCHAR (2)) + '/' + CAST (AÑO (SYSDATETIME ()) COMO VARCHAR (4)) AS [M / YYYY]	6/2011
AA / MM	SELECCIONE CONVERTIR (VARCHAR (5), SYSDATETIME (), 11) AS [YY / MM]	11/06
YYYY / MM	SELECCIONAR CONVERTIR (VARCHAR (7), SYSDATETIME (), 111) AS [YYYY / MM]	2011/06
YY / M	SELECCIONAR DERECHO (CAST (AÑO (SYSDATETIME ()) COMO VARCHAR (4)), 2) + '/' + CAST (MONTH (SYSDATETIME ()) COMO VARCHAR (2)) AS [YY / M]	11/6
YYYY / M	SELECCIONAR CAST (AÑO (SYSDATETIME ()) COMO VARCHAR (4)) + '/' + CAST (MES (SYSDATETIME ()) COMO VARCHAR (2)) AS [YYYY / M]	2011/6
MM / DD	SELECCIONAR CONVERTIR (VARCHAR (5), SYSDATETIME (), 1) AS [MM / DD]	06/08
DD / MM	SELECCIONAR CONVERTIR (VARCHAR (5), SYSDATETIME (), 3) COMO [DD / MM]	08/06
MARYLAND	SELECCIONAR CAST (MES (SYSDATETIME ()) COMO VARCHAR (2)) + '/' + CAST (DÍA (SYSDATETIME ()) COMO VARCHAR (2)) AS [M / D]	6/8
D / M	SELECCIONAR CAST (DÍA (SYSDATETIME ()))	8/6

Formato de fecha	Declaración SQL	Salida de muestra
	COMO VARCHAR (2)) + '/' + CAST (MES (SYSDATETIME ()) COMO VARCHAR (2)) AS [D / M]	
MM.DD.YYYY	SELECCIONAR REEMPLAZAR (CONVERTIR (VARCHAR (10), SYSDATETIME (), 101), '/', '.') COMO [MM.DD.YYYY]	06.08.2011
MM.DD.YY	SELECCIONAR REEMPLAZAR (CONVERTIR (VARCHAR (8), SYSDATETIME (), 1), '/', '.') COMO [MM.DD.YY]	06.08.11
MDYYYY	SELECCIONAR CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + REPUESTO (DÍA (SYSDATETIME ()) COMO VARCHAR (2)) + '.' + CAST (AÑO (SYSDATETIME ()) AS VARCHAR (4)) AS [MDYYYY]	6.8.2011
MDYY	SELECCIONAR CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + REPUESTO (DÍA (SYSDATETIME ()) COMO VARCHAR (2)) + '.' + DERECHA (CAST (AÑO (SYSDATETIME ()) COMO VARCHAR (4)), 2) COMO [MDYY]	6.8.11
DD.MM.YYYY	SELECCIONAR CONVERTIR (VARCHAR (10), SYSDATETIME (), 104) COMO [DD.MM.YYYY]	08.06.2011
DD.MM.YY	SELECCIONAR CONVERTIR (VARCHAR (10), SYSDATETIME (), 4) COMO [DD.MM.YY]	08.06.11
DMYYYY	SELECCIONAR CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (AÑO (SYSDATETIME ()) COMO VARCHAR (4)) COMO [DMYYYY]	8.6.2011
DMYY	SELECCIONAR CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + DERECHA (CAST (AÑO (SYSDATETIME ()) COMO VARCHAR (4)), 2) COMO [DMYY]	8.6.11
YYYY.MD	SELECCIONAR CAST (AÑO (SYSDATETIME ()) COMO VARCHAR (4)) + '.' + CAST (MONTH	2011.6.8

Formato de fecha	Declaración SQL	Salida de muestra
	(SYSDATETIME ()) AS VARCHAR (2)) + '.' + REPUESTO (DÍA (SYSDATETIME ()) AS VARCHAR (2)) AS [YYYY.MD]	
YY.MD	SELECCIONAR A LA DERECHA (CAST (AÑO (SYSDATETIME ()) COMO VARCHAR (4)), 2) + '.' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + REPUESTO (DÍA (SYSDATETIME ()) AS VARCHAR (2)) AS [YY.MD]	11.6.8
MM.YYYY	SELECCIONAR DERECHO (CONVERTIR (VARCHAR (10), SYSDATETIME (), 104), 7) AS [MM.YYYY]	06.2011
MM.YY	SELECCIONAR DERECHO (CONVERTIR (VARCHAR (8), SYSDATETIME (), 4), 5) AS [MM.YY]	06.11
M.YYYY	SELECCIONAR CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (AÑO (SYSDATETIME ()) AS VARCHAR (4)) AS [M.YYYY]	6.2011
M.YY	SELECCIONAR CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + DERECHA (REPUESTO (AÑO (TIEMPO DE SISTEMAS ()) COMO VARCHAR (4)), 2) COMO [M.YY]	6.11
YYYY.MM	SELECCIONAR CONVERTIR (VARCHAR (7), SYSDATETIME (), 102) AS [YYYY.MM]	2011.06
YY.MM	SELECCIONAR CONVERTIR (VARCHAR (5), SYSDATETIME (), 2) AS [YY.MM]	11.06
YYYY.M	SELECCIONAR CAST (AÑO (SYSDATETIME ()) COMO VARCHAR (4)) + '.' + REPUESTO (MES (SYSDATETIME ()) COMO VARCHAR (2)) COMO [YYYY.M]	2011.6
YY.M	SELECCIONAR A LA DERECHA (CAST (AÑO (SYSDATETIME ()) COMO VARCHAR (4)), 2) + '.' + REPUESTO (MES (SYSDATETIME ()) AS VARCHAR (2)) AS [YY.M]	11.6

Formato de fecha	Declaración SQL	Salida de muestra
MM.DD	SELECCIONAR DERECHO (CONVERTIR (VARCHAR (8), SYSDATETIME (), 2), 5) AS [MM.DD]	06.08
DD.MM	SELECCIONAR CONVERTIR (VARCHAR (5), SYSDATETIME (), 4) AS [DD.MM]	08.06
MMDDYYYYYY	SELECCIONAR REEMPLAZAR (CONVERTIR (VARCHAR (10), SYSDATETIME (), 101), '/', '') COMO [MMDDYYYYYY]	06082011
MMDDYY	SELECCIONAR REEMPLAZAR (CONVERTIR (VARCHAR (8), SYSDATETIME (), 1), '/', '') COMO [MMDDYY]	060811
DDMMYYYYYYYYYYYYYY	SELECCIONAR REEMPLAZAR (CONVERTIR (VARCHAR (10), SYSDATETIME (), 103), '/', '') COMO [DDMMYYYYYY]	08062011
DDMMYY	SELECCIONAR REEMPLAZAR (CONVERTIR (VARCHAR (8), SYSDATETIME (), 3), '/', '') COMO [DDMMYY]	080611
MMYYYY	SELECCIONAR DERECHO (REEMPLAZAR (CONVERTIR (VARCHAR (10), SYSDATETIME (), 103), '/', ''), 6) COMO [MMYYYY]	062011
MMYY	SELECCIONE EL DERECHO (REEMPLAZAR (CONVERTIR (VARCHAR (8), SYSDATETIME (), 3), '/', ''), 4) AS [MMYY]	0611
YYYYMM	SELECCIONAR CONVERTIR (VARCHAR (6), SYSDATETIME (), 112) AS [YYYYMM]	201106
YYMM	SELECCIONAR CONVERTIR (VARCHAR (4), SYSDATETIME (), 12) AS [YYMM]	1106
Mes DD, YYYY	SELECCIONE EL NOMBRE DE DATO (MES, SYSDATETIME ()) + " + DERECHO ('0' + DATENAME (DÍA, SYSDATETIME ()), 2) + ',' + DATENAME (AÑO, SYSDATETIME ()) AS [Month DD, YYYY]	08 de junio de 2011
Lun YYYY	SELECCIONAR IZQUIERDA (DATENAME (MES, SYSDATETIME ()), 3) + " + DATENAME (AÑO, SYSDATETIME ()) COMO [Mon YYYY]	Junio 2011

Formato de fecha	Declaración SQL	Salida de muestra
Mes YYYY	SELECCIONE EL NOMBRE DE DATOS (MES, TIEMPO DE SISTEMA ()) + " + DATOS (AÑO, TIEMPO DE SISTEMAS ()) AS [Month YYYY]	Junio 2011
Mes DD	SELECCIONE A LA DERECHA ('0' + DATENAME (DÍA, SYSDATETIME ()), 2) + " + DATENAME (MES, SYSDATETIME ()) AS [DD Month]	08 de junio
Mes DD	SELECCIONE EL NOMBRE DE DATOS (MES, TIEMPO DE SISTEMA ()) + " + DERECHA ('0' + DATOS (DÍA, SISTEMAS DE TIEMPO ()), 2) AS [Month DD]	Junio 08
DD mes YY	SELECCIONAR CAST (DÍA (SYSDATETIME ()) COMO VARCHAR (2)) + " + DATENAME (MM, SYSDATETIME ()) + " + DERECHA (CAST (AÑO (SYSDATETIME ()) COMO VARCHAR (4)), 2) AS [DD mes YY]	08 de junio 11
DD mes YYYY	SELECCIONAR A LA DERECHA ('0' + DATENAME (DÍA, SYSDATETIME ()), 2) + " + DATENAME (MES, SYSDATETIME ()) + " + DATENAME (AÑO, SYSDATETIME ()) AS [DD mes YYYY]	08 de junio de 2011
Lun-yy	SELECCIONAR REEMPLAZAR (DERECHA (CONVERTIR (VARCHAR (9), SYSDATETIME ()), 6), 6), ", '-') AS [Mon-YY]	Junio-08
LUNA YYYY	SELECCIONAR REEMPLAZAR (DERECHA (CONVERTIR (VARCHAR (11), SYSDATETIME ()), 106), 8), ", '-') AS [Mon-YYYY]	Junio de 2011
DD-Mon-YY	SELECCIONAR REEMPLAZAR (CONVERTIR (VARCHAR (9), SYSDATETIME ()), 6), ", '-') COMO [DD-Mon-YY]	08-jun-11
DD-Mon-YYYY	SELECCIONAR REEMPLAZAR (CONVERTIR (VARCHAR (11), SYSDATETIME ()), 106), ", '-') COMO [DD-Mon-YYYY]	08-jun-2011

Lea fechas en línea: <https://riptutorial.com/es/sql-server/topic/1471/fechas>

Capítulo 34: Filestream

Introducción

FILESTREAM integra el motor de base de datos de SQL Server con un sistema de archivos NTFS almacenando datos binarios (BLOB) de objetos grandes binarios varbinary (max) como archivos en el sistema de archivos. Las instrucciones Transact-SQL pueden insertar, actualizar, consultar, buscar y realizar copias de seguridad de los datos de FILESTREAM. Las interfaces del sistema de archivos Win32 proporcionan acceso de transmisión a los datos.

Examples

Ejemplo

Fuente: MSDN [https://technet.microsoft.com/en-us/library/bb933993\(v=sql.105\).aspx](https://technet.microsoft.com/en-us/library/bb933993(v=sql.105).aspx)

Lea Filestream en línea: <https://riptutorial.com/es/sql-server/topic/9509/filestream>

Capítulo 35: Función de cadena dividida en el servidor SQL

Examples

Dividir una cadena en Sql Server 2016

En **SQL Server 2016** finalmente han introducido la función de cadena dividida: [STRING_SPLIT](#)

Parámetros: Acepta dos parámetros.

Cuerda :

Es una expresión de cualquier tipo de carácter (es decir, nvarchar, varchar, nchar o char).

separador

Es una expresión de un solo carácter de cualquier tipo de carácter (por ejemplo, nvarchar (1), varchar (1), nchar (1) o char (1)) que se utiliza como separador para cadenas concatenadas.

Nota: Siempre debe verificar si la expresión es una cadena no vacía.

Ejemplo:

```
Select Value
From STRING_SPLIT('a|b|c','|')
```

En el ejemplo anterior

```
String      : 'a|b|c'
separator   : '|'
```

Resultado:

```
+-----+
|Value|
+-----+
|a    |
+-----+
|b    |
+-----+
|c    |
+-----+
```

Si es una cadena vacía:

```
SELECT value
FROM STRING_SPLIT('','|')
```

Resultado:

```
+-----+
|Value|
+-----+
1 |    |
+-----+
```

Puede evitar la situación anterior agregando una cláusula `WHERE`

```
SELECT value
FROM STRING_SPLIT('','|')
WHERE LTRIM(RTRIM(value))<>''
```

Cadena dividida en Sql Server 2008/2012/2014 usando XML

Como no hay `STRING_SPLIT` función `STRING_SPLIT` , necesitamos usar el truco XML para dividir la cadena en filas:

Ejemplo:

```
SELECT split.a.value('.', 'VARCHAR(100)') AS Value
FROM (SELECT Cast ('<M>' + Replace('A|B|C', '|', '</M><M>')+ '</M>' AS XML) AS Data) AS A
CROSS apply data.nodes ('/M') AS Split(a);
```

Resultado:

```
+-----+
|Value|
+-----+
|A    |
+-----+
|B    |
+-----+
|C    |
+-----+
```

Variable de tabla T-SQL y XML

```
Declare @userList Table(UserKey VARCHAR(60))
Insert into @userList values ('bill'),('jcom'),('others')
--Declared a table variable and insert 3 records

Declare @text XML
Select @text = (
    select UserKey from @userList for XML Path('user'), root('group')
)
--Set the XML value from Table
```

```
Select @text
```

```
--View the variable value
```

```
XML:
```

```
\<group>\<user>\<UserKey>bill\</UserKey>\</user>\<user>\<UserKey>jcom\</UserKey>\</user>\<user>\<UserKey>
```

Lea Función de cadena dividida en el servidor SQL en línea: <https://riptutorial.com/es/sql-server/topic/3713/funcion-de-cadena-dividida-en-el-servidor-sql>

Capítulo 36: Funciones agregadas

Introducción

Las funciones agregadas en SQL Server ejecutan cálculos en conjuntos de valores, devolviendo un solo valor.

Sintaxis

- AVG ([ALL | DISTINCT] *expresión*)
- COUNT ([ALL | DISTINCT] *expresión*)
- MAX (*expresión* [ALL | DISTINCT])
- MIN ([ALL | DISTINCT] *expresión*)
- SUM ([ALL | DISTINCT] *expresión*)

Examples

SUMA()

Devuelve la suma de valores numéricos en una columna dada.

Tenemos la tabla como se muestra en la figura que se utilizará para realizar diferentes funciones agregadas. El nombre de la tabla es *Marksheet* .

SubjectCode	SubjectName	MarksObtained
101	Physics	87
102	Chemistry	75
103	Maths	85
104	English	89
105	Computer	95

```
Select SUM(MarksObtained) From Marksheet
```

La función de `sum` no considera filas con valor NULL en el campo usado como parámetro

En el ejemplo anterior si tenemos otra fila como esta:

```
106    Italian    NULL
```

Esta fila no será considerada en el cálculo de la suma.

AVG ()

Devuelve el promedio de los valores numéricos en una columna dada.

Tenemos la tabla como se muestra en la figura que se utilizará para realizar diferentes funciones agregadas. El nombre de la tabla es *Marksheet*.

SubjectCode	SubjectName	MarksObtained
101	Physics	87
102	Chemistry	75
103	Maths	85
104	English	89
105	Computer	95

```
Select AVG(MarksObtained) From Marksheet
```

La función de `average` no considera filas con valor NULL en el campo usado como parámetro

En el ejemplo anterior si tenemos otra fila como esta:

```
106    Italian    NULL
```

Esta fila no será considerada en el cálculo promedio.

MAX ()

Devuelve el valor más grande en una columna dada.

Tenemos la tabla como se muestra en la figura que se utilizará para realizar diferentes funciones agregadas. El nombre de la tabla es *Marksheet*.

SubjectCode	SubjectName	MarksObtained
101	Physics	87
102	Chemistry	75
103	Maths	85
104	English	89
105	Computer	95

```
Select MAX(MarksObtained) From Marksheet
```

MIN ()

Devuelve el valor más pequeño en una columna dada.

Tenemos la tabla como se muestra en la figura que se utilizará para realizar diferentes funciones agregadas. El nombre de la tabla es *Marksheet*.

SubjectCode	SubjectName	MarksObtained
101	Physics	87
102	Chemistry	75
103	Maths	85
104	English	89
105	Computer	95

```
Select MIN(MarksObtained) From Marksheet
```

CONTAR()

Devuelve el número total de valores en una columna dada.

Tenemos la tabla como se muestra en la figura que se utilizará para realizar diferentes funciones agregadas. El nombre de la tabla es *Marksheet*.

SubjectCode	SubjectName	MarksObtained
101	Physics	87
102	Chemistry	75
103	Maths	85
104	English	89
105	Computer	95

```
Select COUNT(MarksObtained) From Marksheet
```

La función de `count` no considera filas con valor NULL en el campo usado como parámetro. Por lo general, el parámetro de conteo es `*` (todos los campos), por lo que solo si todos los campos de la fila son NULL, esta fila no se considerará

En el ejemplo anterior si tenemos otra fila como esta:

106	Italian	NULL
-----	---------	------

Esta fila no será considerada en el cálculo del conteo.

NOTA

La función `COUNT(*)` devuelve el número de filas en una tabla. Este valor también se puede obtener utilizando una expresión constante no nula que no contiene referencias de columna, como `COUNT(1)`.

Ejemplo

```
Select COUNT(1) From Marksheet
```

COUNT (nombre_columna) con GROUP BY Column_Name

La mayoría de las veces nos gusta obtener el número total de ocurrencia de un valor de columna

en una tabla, por ejemplo:

NOMBRE DE LA TABLA: INFORMES

Reportar nombre	InformePrecio
Prueba	10.00 \$
Prueba	10.00 \$
Prueba	10.00 \$
Prueba 2	11.00 \$
Prueba	10.00 \$
Prueba 3	14.00 \$
Prueba 3	14.00 \$
Prueba 4	100.00 \$

```
SELECT
    ReportName AS REPORT NAME,
    COUNT(ReportName) AS COUNT
FROM
    REPORTS
GROUP BY
    ReportName
```

REPORTAR NOMBRE	CONTAR
Prueba	4
Prueba 2	1
Prueba 3	2
Prueba 4	1

Lea Funciones agregadas en línea: <https://riptutorial.com/es/sql-server/topic/5802/funciones-agregadas>

Capítulo 37: Funciones agregadas de cadenas en SQL Server

Examples

Usando STUFF para la agregación de cadenas

Tenemos una mesa de estudiantes con SubjectId. Aquí el requisito es concatenar en base a subjectId.

Todas las versiones de SQL Server

```
create table #yourstudent (subjectid int, studentname varchar(10))

insert into #yourstudent (subjectid, studentname) values
( 1      , 'Mary'    )
, ( 1      , 'John'    )
, ( 1      , 'Sam'     )
, ( 2      , 'Alaina'  )
, ( 2      , 'Edward'  )

select subjectid, stuff(( select concat( ',', studentname) from #yourstudent y where
y.subjectid = u.subjectid for xml path('')),1,1, '')
from #yourstudent u
group by subjectid
```

String_Agg for String Aggregation

En el caso de SQL Server 2017 o vnext, podemos usar STRING_AGG incorporado para esta agregación. Para la misma mesa de estudiantes,

```
create table #yourstudent (subjectid int, studentname varchar(10))

insert into #yourstudent (subjectid, studentname) values
( 1      , 'Mary'    )
, ( 1      , 'John'    )
, ( 1      , 'Sam'     )
, ( 2      , 'Alaina'  )
, ( 2      , 'Edward'  )

select subjectid, string_agg(studentname, ',') from #yourstudent
group by subjectid
```

Lea Funciones agregadas de cadenas en SQL Server en línea: <https://riptutorial.com/es/sql-server/topic/9892/funciones-agregadas-de-cadenas-en-sql-server>

Capítulo 38: Funciones de cadena

Observaciones

Lista de funciones de cadena (ordenadas alfabéticamente):

- [Ascii](#)
- [Carbonizarse](#)
- [Charindex](#)
- [Concat](#)
- [Diferencia](#)
- [Formato](#)
- [Izquierda](#)
- [Len](#)
- [Inferior](#)
- [Ltrim](#)
- [Nchar](#)
- [Patindex](#)
- [Nombre de lugar](#)
- [Reemplazar](#)
- [Reproducir exactamente](#)
- [Marcha atrás](#)
- [Derecha](#)
- [Rtrim](#)
- [Soundex](#)
- [Espacio](#)
- [Str](#)
- [String_escape](#)

- [String_split](#)
- [Cosas](#)
- [Subcadena](#)
- [Unicode](#)
- [Superior](#)

Examples

Izquierda

Devuelve una cadena secundaria que comienza con el carácter más a la izquierda de una cadena y hasta la longitud máxima especificada.

Parámetros:

1. expresión del personaje. La expresión de caracteres puede ser de cualquier tipo de datos que se pueda convertir implícitamente a `varchar` o `nvarchar`, excepto `text` o `ntext`
2. longitud máxima. Un número entero entre 0 y el valor máximo de `bigint` (9.223.372.036.854.775.807).
Si el parámetro de longitud máxima es negativo, se generará un error.

```
SELECT LEFT('This is my string', 4) -- result: 'This'
```

Si la longitud máxima es mayor que el número de caracteres en la cadena, se devuelve la cadena principal.

```
SELECT LEFT('This is my string', 50) -- result: 'This is my string'
```

Derecha

Devuelve una cadena secundaria que es la parte más derecha de la cadena, con la longitud máxima especificada.

Parámetros:

1. expresión del personaje. La expresión de caracteres puede ser de cualquier tipo de datos que se pueda convertir implícitamente a `varchar` o `nvarchar`, excepto `text` o `ntext`
2. longitud máxima. Un número entero entre 0 y el valor máximo de `bigint` (9.223.372.036.854.775.807). Si el parámetro de longitud máxima es negativo, se generará un error.

```
SELECT RIGHT('This is my string', 6) -- returns 'string'
```

Si la longitud máxima es mayor que el número de caracteres en la cadena, se devuelve la cadena

principal.

```
SELECT RIGHT('This is my string', 50) -- returns 'This is my string'
```

Subcadena

Devuelve una subcadena que comienza con el carácter que está en el índice de inicio especificado y la longitud máxima especificada.

Parámetros:

1. Expresión del personaje La expresión de caracteres puede ser de cualquier tipo de datos que se pueda convertir implícitamente a `varchar` o `nvarchar`, excepto para `text` o `ntext`.
2. Índice de comienzo. Un número (`int` o `bigint`) que especifica el índice de inicio de la subcadena solicitada. (**Nota: las** cadenas en el servidor sql son el índice base 1, lo que significa que el primer carácter de la cadena es el índice 1). Este número puede ser menor que 1. En este caso, si la suma del índice de inicio y la longitud máxima es mayor que 0, la cadena de retorno sería una cadena que comienza desde el primer carácter de la expresión de caracteres y con la longitud de (índice de inicio + max longitud - 1). Si es menos de 0, se devolverá una cadena vacía.
3. Longitud máxima. Un número entero entre 0 y el valor máximo de `bigint` (9.223.372.036.854.775.807). Si el parámetro de longitud máxima es negativo, se generará un error.

```
SELECT SUBSTRING('This is my string', 6, 5) -- returns 'is my'
```

Si la longitud máxima + el índice de inicio es mayor que el número de caracteres en la cadena, se devuelve la cadena principal.

```
SELECT SUBSTRING('Hello World',1,100) -- returns 'Hello World'
```

Si el índice de inicio es más grande que el número de caracteres en la cadena, se devuelve una cadena vacía.

```
SELECT SUBSTRING('Hello World',15,10) -- returns ''
```

ASCII

Devuelve un valor `int` que representa el código ASCII del carácter más a la izquierda de una cadena.

```
SELECT ASCII('t') -- Returns 116
SELECT ASCII('T') -- Returns 84
SELECT ASCII('This') -- Returns 84
```

Si la cadena es Unicode y el carácter más a la izquierda no es ASCII pero se puede representar en la intercalación actual, se puede devolver un valor mayor a 127:

```
SELECT ASCII(N'i') -- returns 239 when `SERVERPROPERTY('COLLATION') =
'SQL_Latin1_General_CP1_CI_AS`
```

Si la cadena es Unicode y el carácter más a la izquierda no se puede representar en la intercalación actual, se devuelve el valor int de 63: (que representa un signo de interrogación en ASCII):

```
SELECT ASCII(N'ï') -- returns 63
SELECT ASCII(nchar(2039)) -- returns 63
```

Índice de caracteres

Devuelve el índice de inicio de la primera aparición de una expresión de cadena dentro de otra expresión de cadena.

Lista de parámetros:

1. Cadena para encontrar (hasta 8000 caracteres)
2. Cadena para buscar (cualquier tipo de datos de carácter válido y longitud, incluido el binario)
3. (Opcional) índice para comenzar. Un número de tipo int o big int. Si se omite o menos de 1, la búsqueda comienza al principio de la cadena.

Si la cadena a buscar es `varchar(max)`, `nvarchar(max)` o `varbinary(max)`, la función `CHARINDEX` devolverá un valor `bigint`. De lo contrario, devolverá un `int`.

```
SELECT CHARINDEX('is', 'this is my string') -- returns 3
SELECT CHARINDEX('is', 'this is my string', 4) -- returns 6
SELECT CHARINDEX(' is', 'this is my string') -- returns 5
```

Carbonizarse

Devuelve un char representado por un código ASCII int.

```
SELECT CHAR(116) -- Returns 't'
SELECT CHAR(84) -- Returns 'T'
```

Esto se puede usar para introducir el nuevo avance de línea / línea `CHAR(10)`, retornos de carro `CHAR(13)`, etc. Consulte [AsciiTable.com](https://www.asciitable.com) para referencia.

Si el valor del argumento no está entre 0 y 255, la función `CHAR` devuelve `NULL`.

El tipo de datos de retorno de la función `CHAR` es `char(1)`.

Len

Devuelve el número de caracteres de una cadena.

Nota: la función `LEN` ignora los espacios finales:

```
SELECT LEN('My string'), -- returns 9
```



```
LEN('My string '), -- returns 9
LEN(' My string') -- returns 12
```

Si se desea la longitud que incluye espacios finales, existen varias técnicas para lograrlo, aunque cada una tiene sus inconvenientes. Una técnica es agregar un solo carácter a la cadena y luego usar el `LEN` menos uno:

```
DECLARE @str varchar(100) = 'My string '
SELECT LEN(@str + 'x') - 1 -- returns 12
```

El inconveniente de esto es que si el tipo de la cadena o la variable de la cadena es de la longitud máxima, el apéndice del carácter adicional se descarta y la longitud resultante todavía no contará los espacios finales. Para solucionar esto, la siguiente versión modificada resuelve el problema y proporciona los resultados correctos en todos los casos a costa de una pequeña cantidad de tiempo de ejecución adicional, y debido a esto (resultados correctos, incluso con pares sustitutos, y una velocidad de ejecución razonable) Parece ser la mejor técnica para usar:

```
SELECT LEN(CONVERT(NVARCHAR(MAX), @str) + 'x') - 1
```

Otra técnica es usar la función `DATALENGTH`.

```
DECLARE @str varchar(100) = 'My string '
SELECT DATALENGTH(@str) -- returns 12
```

Es importante tener en cuenta que `DATALENGTH` devuelve la longitud en bytes de la cadena en la memoria. Esto será diferente para `varchar` vs `nvarchar`.

```
DECLARE @str nvarchar(100) = 'My string '
SELECT DATALENGTH(@str) -- returns 24
```

Puede ajustar esto dividiendo la longitud de datos de la cadena por la longitud de datos de un solo carácter (que debe ser del mismo tipo). El siguiente ejemplo hace esto, y también maneja el caso donde la cadena objetivo está vacía, evitando así una división por cero.

```
DECLARE @str nvarchar(100) = 'My string '
SELECT DATALENGTH(@str) / DATALENGTH(LEFT(LEFT(@str, 1) + 'x', 1)) -- returns 12
```

Incluso esto, sin embargo, tiene un problema en SQL Server 2012 y superior. Producirá resultados incorrectos cuando la cadena contenga pares sustitutos (algunos caracteres pueden ocupar más bytes que otros caracteres en la misma cadena).

Otra técnica es usar `REPLACE` para convertir espacios en un carácter no espacial, y tomar el `LEN` del resultado. Esto da resultados correctos en todos los casos, pero tiene una velocidad de ejecución muy pobre con cadenas largas.

Concat

SQL Server 2012

Devuelve una cadena que es el resultado de dos o más cadenas unidas. `CONCAT` acepta dos o más argumentos.

```
SELECT CONCAT('This', ' is', ' my', ' string') -- returns 'This is my string'
```

Nota: A diferencia de las cadenas de concatenación que usan el operador de concatenación de cadenas (+), al pasar un valor nulo a la función `concat` , se convertirá implícitamente en una cadena vacía:

```
SELECT CONCAT('This', NULL, ' is', ' my', ' string'), -- returns 'This is my string'
       'This' + NULL + ' is' + ' my' + ' string' -- returns NULL.
```

También los argumentos de un tipo no de cadena se convertirán implícitamente en una cadena:

```
SELECT CONCAT('This', ' is my ', 3, 'rd string') -- returns 'This is my 3rd string'
```

Las variables de tipo no de cadena también se convertirán a formato de cadena, sin necesidad de convertirlas manualmente o convertirlas en cadena:

```
DECLARE @Age INT=23;
SELECT CONCAT('Ram is ', @Age, ' years old'); -- returns 'Ram is 23 years old'
```

SQL Server 2012

Las versiones anteriores no admiten la función `CONCAT` y deben usar el operador de concatenación de cadenas (+) en su lugar. Los tipos que no son de cadena deben ser convertidos o convertidos en tipos de cadena para poder concatenarlos de esta manera.

```
SELECT 'This is the number ' + CAST(42 AS VARCHAR(5)) --returns 'This is the number 42'
```

Inferior

Devuelve una expresión de caracteres (`varchar` o `nvarchar`) después de convertir todos los caracteres en mayúsculas a minúsculas.

Parámetros:

1. Expresión del personaje Cualquier expresión de caracteres o datos binarios que puede convertirse implícitamente a `varchar` .

```
SELECT LOWER('This IS my STRING') -- Returns 'this is my string'
```

```
DECLARE @String nchar(17) = N'This IS my STRING';
SELECT LOWER(@String) -- Returns 'this is my string'
```

Superior

Devuelve una expresión de caracteres (`varchar` o `nvarchar`) después de convertir todos los

caracteres en minúsculas en mayúsculas.

Parámetros:

1. Expresión del personaje Cualquier expresión de caracteres o datos binarios que puede convertirse implícitamente a `varchar` .

```
SELECT UPPER('This IS my STRING') -- Returns 'THIS IS MY STRING'  
  
DECLARE @String nchar(17) = N'This IS my STRING';  
SELECT UPPER(@String) -- Returns 'THIS IS MY STRING'
```

LTrim

Devuelve una expresión de carácter (`varchar` o `nvarchar`) después de eliminar todos los espacios en blanco iniciales, es decir, espacios en blanco desde la izquierda hasta el primer carácter de espacio que no sea blanco.

Parámetros:

1. expresión del personaje. Cualquier expresión de caracteres o datos binarios que pueda convertirse implícitamente a `varcher` , excepto `text` , `ntext` e `image` .

```
SELECT LTRIM('   This is my string') -- Returns 'This is my string'
```

RTrim

Devuelve una expresión de caracteres (`varchar` o `nvarchar`) después de eliminar todos los espacios en blanco finales, es decir, espacios desde el extremo derecho de la cadena hasta el primer carácter de espacio no blanco a la izquierda.

Parámetros:

1. expresión del personaje. Cualquier expresión de caracteres o datos binarios que pueda convertirse implícitamente a `varcher` , excepto `text` , `ntext` e `image` .

```
SELECT RTRIM('This is my string   ') -- Returns 'This is my string'
```

Unicode

Devuelve el valor entero que representa el valor Unicode del primer carácter de la expresión de entrada.

Parámetros:

1. Expresión de caracteres Unicode. Cualquier expresión `nchar` o `nvarchar` válida.

```
SELECT UNICODE(N'€') -- Returns 400
```

```
DECLARE @Unicode nvarchar(11) = N'ε is a char'  
SELECT UNICODE(@Unicode) -- Returns 400
```

NChar

Devuelve los caracteres Unicode (`nchar(1)` o `nvarchar(2)`) correspondientes al argumento entero que recibe, según lo define el estándar de Unicode.

Parámetros:

1. expresión entera Cualquier expresión entera que sea un número positivo entre 0 y 65535, o si la intercalación de la base de datos admite el indicador de carácter suplementario (CS), el rango admitido está entre 0 y 1114111. Si la expresión entera no se encuentra dentro de este rango, `null` devuelto

```
SELECT NCHAR(257) -- Returns 'ā'  
SELECT NCHAR(400) -- Returns 'ε'
```

Marcha atrás

Devuelve un valor de cadena en orden inverso.

Parámetros:

1. expresión de la cadena. Cualquier cadena o datos binarios que pueden convertirse implícitamente a `varchar`.

```
Select REVERSE('Sql Server') -- Returns 'revreS lqS'
```

PatIndex

Devuelve la posición inicial de la primera aparición de un patrón especificado en la expresión especificada.

Parámetros:

1. modelo. Una expresión de caracteres contiene la secuencia a encontrar. Limitado a una longitud máxima de 8000 caracteres. Se pueden usar comodines (`%` , `_`) en el patrón. Si el patrón no comienza con un comodín, puede que solo coincida con lo que está al principio de la expresión. Si no termina con un comodín, puede que solo coincida con lo que está al final de la expresión.
2. expresión. Cualquier tipo de datos de cadena.

```
SELECT PATINDEX('%ter%', 'interesting') -- Returns 3.  
SELECT PATINDEX('%t_r%', 'interesting') -- Returns 3.  
SELECT PATINDEX('ter%', 'interesting') -- Returns 0, since 'ter' is not at the start.
```

```
SELECT PATINDEX('inter%', 'interesting') -- Returns 1.
SELECT PATINDEX('%ing', 'interesting') -- Returns 9.
```

Espacio

Devuelve una cadena (`varchar`) de espacios repetidos.

Parámetros:

1. expresión entera Cualquier expresión entera, hasta 8000. Si es negativa, se devuelve `null` . Si es 0, se devuelve una cadena vacía. (Para devolver una cadena más larga que 8000 espacios, use `Replicar`).

```
SELECT SPACE(-1) -- Returns NULL
SELECT SPACE(0) -- Returns an empty string
SELECT SPACE(3) -- Returns '   ' (a string containing 3 spaces)
```

Reproducir exactamente

Repite un valor de cadena un número especificado de veces.

Parámetros:

1. expresión de la cadena. La expresión de cadena puede ser una cadena de caracteres o datos binarios.
2. expresión entera Cualquier tipo entero, incluyendo `bigint` . Si es negativo, se devuelve `null` . Si es 0, se devuelve una cadena vacía.

```
SELECT REPLICATE('a', -1) -- Returns NULL
SELECT REPLICATE('a', 0) -- Returns ''
SELECT REPLICATE('a', 5) -- Returns 'aaaaa'
SELECT REPLICATE('Abc', 3) -- Returns 'AbcAbcAbc'
```

Nota: Si la expresión de cadena no es del tipo `varchar(max)` o `nvarchar(max)` , el valor de retorno no excederá los 8000 caracteres. La replicación se detendrá antes de agregar la cadena que hará que el valor de retorno exceda ese límite:

```
SELECT LEN(REPLICATE('a b c d e f g h i j k l', 350)) -- Returns 7981
SELECT LEN(REPLICATE(cast('a b c d e f g h i j k l' as varchar(max)), 350)) -- Returns 8050
```

Reemplazar

Devuelve una cadena (`varchar` o `nvarchar`) donde todas las apariciones de una cadena secundaria especificada se reemplazan con otra cadena secundaria.

Parámetros:

1. expresión de la cadena. Esta es la cadena que se buscaría. Puede ser un carácter o tipo de datos binarios.
2. modelo. Esta es la cadena secundaria que se reemplazaría. Puede ser un carácter o tipo de datos binarios. El argumento de patrón no puede ser una cadena vacía.
3. reemplazo. Esta es la cadena secundaria que reemplazaría la cadena secundaria del patrón. Puede ser un carácter o datos binarios.

```
SELECT REPLACE('This is my string', 'is', 'XX') -- Returns 'ThXX XX my string'.
```

Notas:

- Si la expresión de cadena no es del tipo `varchar(max)` o `nvarchar(max)`, la función de `replace` trunca el valor de retorno en 8,000 caracteres.
- El tipo de datos de retorno depende de los tipos de datos de entrada: devuelve `nvarchar` si uno de los valores de entrada es `nvarchar`, o `varchar` contrario.
- Devuelve `NULL` si alguno de los parámetros de entrada es `NULL`

String_Split

SQL Server 2016

Divide una expresión de cadena usando un separador de caracteres. Tenga en cuenta que `STRING_SPLIT()` es una función con valores de tabla y, por lo tanto, debe usarse dentro de la cláusula `FROM`.

Parámetros:

1. cuerda. Cualquier expresión de tipo de carácter (`char`, `nchar`, `varchar` o `nvarchar`)
2. separador Una expresión de un solo carácter de cualquier tipo (`char(1)`, `nchar(1)`, `varchar(1)` o `nvarchar(1)`).

Devuelve una tabla de una sola columna donde cada fila contiene un fragmento de la cadena. El nombre de las columnas es `value`, y el tipo de datos es `nvarchar` si alguno de los parámetros es `nchar` o `nvarchar`, de lo contrario `varchar`.

El siguiente ejemplo divide una cadena usando el espacio como separador:

```
SELECT value FROM STRING_SPLIT('Lorem ipsum dolor sit amet.', ' ');
```

Resultado:

```
value
-----
Lorem
ipsum
dolor
sit
amet.
```

Observaciones:

La función `STRING_SPLIT` está disponible solo en el nivel de compatibilidad **130** . Si el nivel de compatibilidad de su base de datos es inferior a 130, SQL Server no podrá encontrar y ejecutar la función `STRING_SPLIT` . Puede cambiar el nivel de compatibilidad de una base de datos usando el siguiente comando:

```
ALTER DATABASE [database_name] SET COMPATIBILITY_LEVEL = 130
```

SQL Server 2016

Las versiones anteriores del servidor SQL no tienen una función de cadena de división integrada. Hay muchas funciones definidas por el usuario que resuelven el problema de dividir una cadena. Puede leer el artículo de Aaron Bertrand [Dividir las cuerdas de la manera correcta, o la mejor manera de](#) obtener una comparación completa de algunas de ellas.

Str

Devuelve datos de caracteres (`varchar`) convertidos de datos numéricos.

Parámetros:

1. expresión flotante Un tipo de datos numérico aproximado con un punto decimal.
2. longitud. **Opcional**. La longitud total de la expresión de cadena que se devolvería, incluidos los dígitos, el punto decimal y los espacios iniciales (si es necesario). El valor predeterminado es 10.
3. decimal. **Opcional**. El número de dígitos a la derecha del punto decimal. Si es mayor que 16, el resultado se truncaría a dieciséis lugares a la derecha del punto decimal.

```
SELECT STR(1.2) -- Returns '          1'
SELECT STR(1.2, 3) -- Returns '   1'
SELECT STR(1.2, 3, 2) -- Returns '1.2'
SELECT STR(1.2, 5, 2) -- Returns ' 1.20'
SELECT STR(1.2, 5, 5) -- Returns '1.200'
SELECT STR(1, 5, 2) -- Returns ' 1.00'
SELECT STR(1) -- Returns '          1'
```

Nombre de lugar

Devuelve una cadena Unicode rodeada de delimitadores para convertirla en un identificador válido delimitado por SQL Server.

Parámetros:

1. cadena de caracteres. Una cadena de datos Unicode, hasta 128 caracteres (`sysname`). Si

una cadena de entrada tiene más de 128 caracteres, la función devuelve `null` .

2. carácter de la cita. **Opcional** Un solo carácter para utilizar como delimitador. Puede ser una comilla simple (`'` o ```), un corchete izquierdo o derecho (`{` , `[` , `(` , `<` `o` `>` , `)` , `]` , `}`) o una comilla doble (`"`). Cualquier otro valor devolverá nulo El valor por defecto es entre corchetes.

```
SELECT QUOTENAME('what''s my name?')          -- Returns [what's my name?]

SELECT QUOTENAME('what''s my name?', '[')      -- Returns [what's my name?]
SELECT QUOTENAME('what''s my name?', ']')      -- Returns [what's my name?]

SELECT QUOTENAME('what''s my name?', ''')      -- Returns 'what''s my name?'

SELECT QUOTENAME('what''s my name?', '"')      -- Returns "what's my name?"

SELECT QUOTENAME('what''s my name?', ')')      -- Returns (what's my name?)
SELECT QUOTENAME('what''s my name?', '(')      -- Returns (what's my name?)

SELECT QUOTENAME('what''s my name?', '<')      -- Returns <what's my name?>
SELECT QUOTENAME('what''s my name?', '>')      -- Returns <what's my name?>

SELECT QUOTENAME('what''s my name?', '{')      -- Returns {what's my name?}
SELECT QUOTENAME('what''s my name?', '}')      -- Returns {what's my name?}

SELECT QUOTENAME('what''s my name?', '`')      -- Returns `what's my name?`
```

Soundex

Devuelve un código de cuatro caracteres (`varchar`) para evaluar la similitud fonética de dos cadenas.

Parámetros:

1. expresión del personaje. Una expresión alfanumérica de datos de caracteres.

La función `soundex` crea un código de cuatro caracteres que se basa en cómo sonaría la expresión del carácter cuando se habla. la primera char es la versión en mayúsculas del primer carácter del parámetro, los restantes 3 caracteres son números que representan las letras en la expresión (excepto a, e, i, o, u, h, w e y que se ignoran) .

```
SELECT SOUNDEX ('Smith') -- Returns 'S530'

SELECT SOUNDEX ('Smythe') -- Returns 'S530'
```

Diferencia

Devuelve un valor entero (`int`) que indica la diferencia entre los valores `soundex` de dos expresiones de caracteres.

Parámetros:

1. expresión de caracteres 1.

2. expresión de caracteres 2.

Ambos parámetros son expresiones alfanuméricas de datos de caracteres.

El número entero devuelto es el número de caracteres en los valores soundex de los parámetros que son iguales, por lo que 4 significa que las expresiones son muy similares y 0 significa que son muy diferentes.

```
SELECT  SOUNDEX('Green'),    -- G650
        SOUNDEX('Greene'),   -- G650
        DIFFERENCE('Green','Greene') -- Returns 4

SELECT  SOUNDEX('Blotchet-Halls'), -- B432
        SOUNDEX('Greene'),        -- G650
        DIFFERENCE('Blotchet-Halls', 'Greene') -- Returns 0
```

Formato

SQL Server 2012

Devuelve un valor `NVARCHAR` formateado con el formato y la cultura especificados (si se especifica). Esto se utiliza principalmente para convertir los tipos de fecha y hora en cadenas.

Parámetros:

1. `value` Una expresión de un tipo de datos soportado para formatear. Los tipos válidos se enumeran a continuación.
2. `format` Un patrón de formato `NVARCHAR`. Consulte la documentación oficial de Microsoft para [las cadenas de formato estándar](#) y [personalizado](#).
3. `culture` **Opcional** Argumento `nvarchar` especificando una cultura. El valor predeterminado es la cultura de la sesión actual.

FECHA

Usando cadenas de formato estándar:

```
DECLARE @d DATETIME = '2016-07-31';

SELECT
    FORMAT ( @d, 'd', 'en-US' ) AS 'US English Result' -- Returns '7/31/2016'
,FORMAT ( @d, 'd', 'en-gb' ) AS 'Great Britain English Result' -- Returns '31/07/2016'
,FORMAT ( @d, 'd', 'de-de' ) AS 'German Result' -- Returns '31.07.2016'
,FORMAT ( @d, 'd', 'zh-cn' ) AS 'Simplified Chinese (PRC) Result' -- Returns '2016/7/31'
,FORMAT ( @d, 'D', 'en-US' ) AS 'US English Result' -- Returns 'Sunday, July 31, 2016'
,FORMAT ( @d, 'D', 'en-gb' ) AS 'Great Britain English Result' -- Returns '31 July 2016'
,FORMAT ( @d, 'D', 'de-de' ) AS 'German Result' -- Returns 'Sonntag, 31. Juli 2016'
```

Usando cadenas de formato personalizado:

```
SELECT FORMAT( @d, 'dd/MM/yyyy', 'en-US' ) AS 'DateTime Result' -- Returns '31/07/2016'
,FORMAT(123456789, '###-##-####') AS 'Custom Number Result' -- Returns '123-45-6789',
,FORMAT( @d, 'dddd, MMMM dd, yyyy hh:mm:ss tt', 'en-US') AS 'US' -- Returns 'Sunday, July
```

```
31, 2016 12:00:00 AM'
,FORMAT( @d,'dddd, MMMM dd, yyyy hh:mm:ss tt','hi-IN') AS 'Hindi' -- Returns रविवार, जुलाई 31,
2016 12:00:00 पूरवाहन
,FORMAT ( @d, 'dddd', 'en-US' ) AS 'US' -- Returns 'Sunday'
,FORMAT ( @d, 'dddd', 'hi-IN' ) AS 'Hindi' -- Returns 'रविवार'
```

FORMAT también se puede utilizar para formatear CURRENCY , PERCENTAGE y NUMBERS .

MONEDA

```
DECLARE @Price1 INT = 40
SELECT FORMAT(@Price1,'c','en-US') AS 'CURRENCY IN US Culture' -- Returns '$40.00'
,FORMAT(@Price1,'c','de-DE') AS 'CURRENCY IN GERMAN Culture' -- Returns '40,00 €'
```

Podemos especificar el número de dígitos después del decimal.

```
DECLARE @Price DECIMAL(5,3) = 40.356
SELECT FORMAT( @Price, 'C') AS 'Default', -- Returns '$40.36'
FORMAT( @Price, 'C0') AS 'With 0 Decimal', -- Returns '$40'
FORMAT( @Price, 'C1') AS 'With 1 Decimal', -- Returns '$40.4'
FORMAT( @Price, 'C2') AS 'With 2 Decimal', -- Returns '$40.36'
```

PORCENTAJE

```
DECLARE @Percentage float = 0.35674
SELECT FORMAT( @Percentage, 'P') AS '% Default', -- Returns '35.67 %'
FORMAT( @Percentage, 'P0') AS '% With 0 Decimal', -- Returns '36 %'
FORMAT( @Percentage, 'P1') AS '% with 1 Decimal' -- Returns '35.7 %'
```

NÚMERO

```
DECLARE @Number AS DECIMAL(10,2) = 454545.389
SELECT FORMAT( @Number, 'N','en-US') AS 'Number Format in US', -- Returns '454,545.39'
FORMAT( @Number, 'N','en-IN') AS 'Number Format in INDIA', -- Returns '4,54,545.39'
FORMAT( @Number, '#.0') AS 'With 1 Decimal', -- Returns '454545.4'
FORMAT( @Number, '#.00') AS 'With 2 Decimal', -- Returns '454545.39'
FORMAT( @Number, '#,##.00') AS 'With Comma and 2 Decimal', -- Returns '454,545.39'
FORMAT( @Number, '##.00') AS 'Without Comma and 2 Decimal', -- Returns '454545.39'
FORMAT( @Number, '000000000') AS 'Left-padded to nine digits' -- Returns '000454545'
```

Lista de tipos de valores válidos: ([fuente](#))

Category	Type	.Net type
Numeric	bigint	Int64
Numeric	int	Int32
Numeric	smallint	Int16
Numeric	tinyint	Byte
Numeric	decimal	SqlDecimal
Numeric	numeric	SqlDecimal
Numeric	float	Double
Numeric	real	Single
Numeric	smallmoney	Decimal
Numeric	money	Decimal

Date and Time	date	DateTime
Date and Time	time	TimeSpan
Date and Time	datetime	DateTime
Date and Time	smalldatetime	DateTime
Date and Time	datetime2	DateTime
Date and Time	datetimeoffset	DateTimeOffset

Notas importantes:

- `FORMAT` devuelve `NULL` para errores distintos de una cultura que no es válida. Por ejemplo, se devuelve `NULL` si el valor especificado en el formato no es válido.
- `FORMAT` basa en la presencia de .NET Framework Common Language Runtime (CLR).
- `FORMAT` basa en las reglas de formato de CLR que dictan que los dos puntos y los periodos deben escaparse. Por lo tanto, cuando la cadena de formato (segundo parámetro) contiene dos puntos o puntos, los dos puntos o puntos deben escaparse con una barra invertida cuando un valor de entrada (primer parámetro) es del tipo de datos de tiempo.

Vea también [Formato de fecha y hora usando el ejemplo de documentación FORMATO](#) .

String_escape

SQL Server 2016

Escapa caracteres especiales en textos y devuelve texto (`nvarchar(max)`) con caracteres escapados.

Parámetros:

1. `texto`. es una expresión `nvarchar` que representa la cadena que debe escaparse.
2. `tipo`. Reglas de escape que se aplicarán. Actualmente el único valor soportado es `'json'` .

```
SELECT STRING_ESCAPE('\ /
\\ " ', 'json') -- returns '\\t\\n\\\\t\"t'
```

Lista de personajes que se escaparán:

Special character	Encoded sequence

Quotation mark (")	\"
Reverse solidus (\)	\\
Solidus (/)	\/
Backspace	\b
Form feed	\f
New line	\n
Carriage return	\r
Horizontal tab	\t
Control character	Encoded sequence

CHAR(0)	\u0000
CHAR(1)	\u0001

...
CHAR(31)

...
\u001f

Lea Funciones de cadena en línea: <https://riptutorial.com/es/sql-server/topic/4113/funciones-de-cadena>

Capítulo 39: Funciones de clasificación

Sintaxis

- `DENSE_RANK () OVER ([<partition_by_clause>] <order_by_clause>)`
- `RANK () OVER ([partition_by_clause] order_by_clause)`

Parámetros

Argumentos	Detalles
<code><partition_by_clause></code>	Divide el conjunto de resultados producido por la cláusula <code>FROM</code> en particiones a las que se aplica la función <code>DENSE_RANK</code> . Para la sintaxis de <code>PARTITION BY</code> , vea Cláusula OVER (Transact-SQL) .
<code><order_by_clause></code>	Determina el orden en que se aplica la función <code>DENSE_RANK</code> a las filas en una partición.
<code>OVER ([partition_by_clause] order_by_clause)</code>	<code>partition_by_clause</code> divide el conjunto de resultados producido por la cláusula <code>FROM</code> en particiones a las que se aplica la función. Si no se especifica, la función trata todas las filas del conjunto de resultados de la consulta como un solo grupo. <code>order_by_clause</code> determina el orden de los datos antes de que se aplique la función. El <code>order_by_clause</code> es obligatorio. La <code><rows or range clause></code> de la cláusula <code>OVER</code> no se puede especificar para la función <code>RANK</code> . Para obtener más información, consulte la cláusula OVER (Transact-SQL) .

Observaciones

Si dos o más filas se unen para un rango en la misma partición, cada fila recibe el mismo rango. Por ejemplo, si los dos principales vendedores tienen el mismo valor de `SalesYTD`, ambos están clasificados uno. El vendedor con el siguiente mayor `SalesYTD` ocupa el puesto número dos. Este es uno más que el número de filas distintas que vienen antes de esta fila. Por lo tanto, los números devueltos por la función `DENSE_RANK` no tienen espacios vacíos y siempre tienen rangos consecutivos.

El orden de clasificación utilizado para toda la consulta determina el orden en que aparecen las filas en un resultado. Esto implica que una fila clasificada número uno no tiene que ser la primera fila en la partición.

`DENSE_RANK` no es determinista. Para obtener más información, consulte [Funciones deterministas y no deterministas](#).

Examples

RANGO()

UN RANGO () Devuelve el rango de cada fila en el conjunto de resultados de la columna particionada.

P.ej :

```
Select Studentid,Name,Subject,Marks,  
RANK() over(partition by name order by Marks desc)Rank  
From Exam  
order by name,subject
```

Studentid	Name	Subject	Marks	Rank
101	Ivan	Maths	70	2
101	Ivan	Science	80	1
101	Ivan	Social	60	3
102	Ryan	Maths	60	2
102	Ryan	Science	50	3
102	Ryan	Social	70	1
103	Tanvi	Maths	90	1
103	Tanvi	Science	90	1
103	Tanvi	Social	80	3

DENSE_RANK ()

Igual que el de RANK (). Vuelve el rango sin ningún tipo de brechas:

```
Select Studentid, Name,Subject,Marks,  
DENSE_RANK() over(partition by name order by Marks desc)Rank  
From Exam  
order by name
```

Studentid	Name	Subject	Marks	Rank
101	Ivan	Science	80	1
101	Ivan	Maths	70	2
101	Ivan	Social	60	3
102	Ryan	Social	70	1
102	Ryan	Maths	60	2
102	Ryan	Science	50	3
103	Tanvi	Maths	90	1
103	Tanvi	Science	90	1
103	Tanvi	Social	80	2

Lea Funciones de clasificación en línea: <https://riptutorial.com/es/sql-server/topic/5031/funciones-de-clasificacion>

Capítulo 40: Funciones de ventana

Examples

Media móvil centrada

Calcule un promedio móvil de 6 meses (126 días hábiles) centrado en un precio:

```
SELECT TradeDate, AVG(Px) OVER (ORDER BY TradeDate ROWS BETWEEN 63 PRECEDING AND 63 FOLLOWING)
AS PxMovingAverage
FROM HistoricalPrices
```

Tenga en cuenta que, debido a que tomará *hasta* 63 filas antes y después de cada fila devuelta, al comienzo y al final del rango de TradeDate no estará centrado: cuando alcance la mayor TradeDate, solo podrá encontrar 63 valores anteriores a incluir en la media.

Encuentre el elemento más reciente en una lista de eventos con marca de tiempo

En las tablas que registran eventos, a menudo hay un campo de fecha y hora que registra la hora en que ocurrió un evento. Encontrar el evento más reciente puede ser difícil porque siempre es posible que dos eventos se registraron con marcas de tiempo exactamente idénticas. Puede usar el número de fila () sobre (ordenar por ...) para asegurarse de que todos los registros estén clasificados de forma única, y seleccionar el primero (donde my_ranking = 1)

```
select *
from (
    select
        *,
        row_number() over (order by crdate desc) as my_ranking
    from sys.sysobjects
) g
where my_ranking=1
```

Esta misma técnica se puede usar para devolver una sola fila desde cualquier conjunto de datos con valores potencialmente duplicados.

Promedio móvil de los últimos 30 artículos

Promedio móvil de los últimos 30 artículos vendidos

```
SELECT
    value_column1,
    (
        SELECT
            AVG(value_column1) AS moving_average
        FROM Table1 T2
        WHERE ( SELECT
            COUNT(*)
```

```
        FROM Table1 T3
        WHERE date_column1 BETWEEN T2.date_column1 AND T1.date_column1
        ) BETWEEN 1 AND 30
    ) as MovingAvg
FROM Table1 T1
```

Lea Funciones de ventana en línea: <https://riptutorial.com/es/sql-server/topic/3209/funciones-de-ventana>

Capítulo 41: Funciones lógicas

Examples

ESCOGER

SQL Server 2012

Devuelve el elemento en el índice especificado de una lista de valores. Si el `index` excede los límites de los `values`, se devuelve `NULL`.

Parámetros:

1. `index` : entero, índice al ítem en `values`. Basado en 1
2. `values` : cualquier tipo, lista separada por comas.

```
SELECT CHOOSE (1, 'apples', 'pears', 'oranges', 'bananas') AS chosen_result

chosen_result
-----
apples
```

IIF

SQL Server 2012

Devuelve uno de los dos valores, dependiendo de si una expresión booleana determinada se evalúa como verdadera o falsa.

Parámetros:

1. `boolean_expression` evaluó para determinar qué valor devolver
2. `true_value` devuelve `true_value` si `boolean_expression` evalúa como verdadero
3. `false_value` devuelto si `boolean_expression` evalúa como falso

```
SELECT IIF (42 > 23, 'I knew that!', 'That is not true.') AS iif_result

iif_result
-----
I knew that!
```

SQL Server 2012

`IIF` puede ser reemplazado por una declaración `CASE`. El ejemplo anterior puede ser escrito como

```
SELECT CASE WHEN 42 > 23 THEN 'I knew that!' ELSE 'That is not true.' END AS iif_result

iif_result
-----
```

I knew that!

Lea Funciones logicas en línea: <https://riptutorial.com/es/sql-server/topic/10647/funciones-logicas>

Capítulo 42: Generando un rango de fechas

Parámetros

Parámetro	Detalles
@Partir de la fecha	El límite inferior inclusivo del intervalo de fechas generado.
@Hasta la fecha	El límite superior inclusivo del rango de fechas generado.

Observaciones

La mayoría de los expertos parecen recomendar la creación de una tabla de fechas en lugar de generar una secuencia sobre la marcha. Consulte

<http://dba.stackexchange.com/questions/86435/filling-in-date-holes-in-grouped-by-date-sql-data>

Examples

Generando intervalo de fechas con CTE recursivo

Usando un CTE recursivo, puede generar un rango de fechas inclusivo:

```
Declare @FromDate Date = '2014-04-21',
        @ToDate Date = '2014-05-02'

;With DateCte (Date) As
(
    Select @FromDate Union All
    Select DateAdd(Day, 1, Date)
    From DateCte
    Where Date < @ToDate
)
Select Date
From DateCte
Option (MaxRecursion 0)
```

La configuración predeterminada de `MaxRecursion` es 100. Generar más de 100 fechas usando este método requerirá el segmento `Option (MaxRecursion N)` de la consulta, donde `N` es la configuración deseada de `MaxRecursion`. Establecer esto en 0 eliminará la limitación de `MaxRecursion` completo.

Generando un intervalo de fechas con una tabla de conteo

Otra forma de generar un rango de fechas es mediante la utilización de una Tabla de Tally para crear las fechas entre el rango:

```

Declare    @FromDate    Date = '2014-04-21',
           @ToDate      Date = '2014-05-02'

;With
  E1(N) As (Select 1 From (Values (1), (1), (1), (1), (1), (1), (1), (1), (1), (1)) DT(N)),
  E2(N) As (Select 1 From E1 A Cross Join E1 B),
  E4(N) As (Select 1 From E2 A Cross Join E2 B),
  E6(N) As (Select 1 From E4 A Cross Join E2 B),
  Tally(N) As
  (
    Select    Row_Number() Over (Order By (Select Null))
    From      E6
  )
Select    DateAdd(Day, N - 1, @FromDate) Date
From      Tally
Where     N <= DateDiff(Day, @FromDate, @ToDate) + 1

```

Lea Generando un rango de fechas en línea: <https://riptutorial.com/es/sql-server/topic/3232/generando-un-rango-de-fechas>

Capítulo 43: Gobernador de recursos

Observaciones

El regulador de recursos en SQL Server es una característica que le permite administrar el uso de recursos por diferentes aplicaciones y usuarios. Se activa en tiempo real al establecer los límites de la CPU y la memoria. Ayudará a prevenir que un proceso pesado consuma todos los recursos del sistema mientras, por ejemplo, las tareas más pequeñas los esperan.

Solo disponible en Enterprise Editions

Examples

Leyendo las estadísticas

```
select *
from sys.dm_resource_governor_workload_groups

select *
from sys.dm_resource_governor_resource_pools
```

Crear un grupo para consultas ad hoc.

Primero crea un conjunto de recursos además del predeterminado

```
CREATE RESOURCE POOL [PoolAdhoc] WITH(min_cpu_percent=0,
    max_cpu_percent=50,
    min_memory_percent=0,
    max_memory_percent=50)
GO
```

Crea el grupo de trabajo para la piscina.

```
CREATE WORKLOAD GROUP [AdhocMedium] WITH(importance=Medium) USING [PoolAdhoc]
```

Cree la función que contiene la lógica del gobernador de recursos y adjúntela

```
create function [dbo].[ufn_ResourceGovernorClassifier]()
    returns sysname with schemabinding
as
begin
    return CASE
        WHEN APP_NAME() LIKE 'Microsoft Office%' THEN
            'AdhocMedium' -- Excel
        WHEN APP_NAME() LIKE 'Microsoft SQL Server Management Studio%' THEN
            'AdhocMedium' -- Adhoc SQL
        WHEN SUSER_NAME() LIKE 'DOMAIN\username' THEN 'AdhocMedium'
        -- Ssis
        ELSE 'default'
```

```
        END
end
GO

alter resource governor
with (classifier_function = dbo.ufn_ResourceGovernorClassifier)
GO

alter resource governor reconfigure
GO
```

Lea Gobernador de recursos en línea: <https://riptutorial.com/es/sql-server/topic/4146/gobernador-de-recursos>

Capítulo 44: Grupo de archivos

Examples

Crear grupo de archivos en base de datos

Podemos crearlo por dos vías. Primero del modo de diseñador de propiedades de la base de datos:

The screenshot shows the 'Database Properties - Test' dialog box in SQL Server Enterprise Manager. The 'Filegroups' tab is selected in the left-hand pane. The main area displays a table with the following data:

Name	Files
PRIMARY	1
newFilegroupName	0

Below this table, there are sections for 'FILESTREAM' and 'MEMORY OPTIMIZED DATA', both of which are currently empty. The 'Connection' section on the left shows the server name as 'DBITABRIZ\930919' and the connection status as 'Ready'.

Y por scripts sql:

```
USE master;
GO
-- Create the database with the default data
-- filegroup and a log file. Specify the
```

```

-- growth increment and the max size for the
-- primary data file.

CREATE DATABASE TestDB ON PRIMARY
(
    NAME = 'TestDB_Primary',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\TestDB_Prm.mdf',
    SIZE = 1 GB,
    MAXSIZE = 10 GB,
    FILEGROWTH = 1 GB
), FILEGROUP TestDB_FG1
(
    NAME = 'TestDB_FG1_1',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\TestDB_FG1_1.ndf',
    SIZE = 10 MB,
    MAXSIZE = 10 GB,
    FILEGROWTH = 1 GB
),
(
    NAME = 'TestDB_FG1_2',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\TestDB_FG1_2.ndf',
    SIZE = 10 MB,
    MAXSIZE = 10 GB,
    FILEGROWTH = 1 GB
) LOG ON
(
    NAME = 'TestDB_log',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\TestDB.ldf',
    SIZE = 10 MB,
    MAXSIZE = 10 GB,
    FILEGROWTH = 1 GB
);

go
ALTER DATABASE TestDB MODIFY FILEGROUP TestDB_FG1 DEFAULT;
go

-- Create a table in the user-defined filegroup.
USE TestDB;
Go

CREATE TABLE MyTable
(
    col1 INT PRIMARY KEY,
    col2 CHAR(8)
)
ON TestDB_FG1;
GO

```

Lea Grupo de archivos en línea: <https://riptutorial.com/es/sql-server/topic/5461/grupo-de-archivos>

Capítulo 45: Importación a granel

Examples

INSERTO A GRANEL con opciones

Puede personalizar las reglas de análisis utilizando diferentes opciones en la cláusula WITH:

```
BULK INSERT People
FROM 'f:\orders\people.csv'
WITH ( CODEPAGE = '65001',
      FIELDTERMINATOR = ',',
      ROWTERMINATOR = '\n'
    );
```

En este ejemplo, CODEPAGE especifica que un archivo fuente en el archivo UTF-8, y los TERMINATORS son coma y nueva línea.

INSERTO A GRANEL

El comando BULK INSERT se puede usar para importar archivos a SQL Server:

```
BULK INSERT People
FROM 'f:\orders\people.csv'
```

El comando BULK INSERT asignará columnas en archivos con columnas en la tabla de destino.

Leyendo todo el contenido del archivo usando OPENROWSET (BULK)

Puede leer el contenido del archivo usando la función OPENROWSET (BULK) y almacenar el contenido en alguna tabla:

```
INSERT INTO myTable(content)
SELECT BulkColumn
FROM OPENROWSET(BULK N'C:\Text1.txt', SINGLE_BLOB) AS Document;
```

La opción SINGLE_BLOB leerá todo el contenido de un archivo como una sola celda.

Lea el archivo utilizando OPENROWSET (BULK) y el archivo de formato

Yu puede definir el formato del archivo que se importará usando la opción FORMATFILE:

```
INSERT INTO mytable
SELECT a.*
FROM OPENROWSET(BULK 'c:\test\values.txt',
  FORMATFILE = 'c:\test\values.fmt') AS a;
```

El archivo de formato, format_file.fmt, describe las columnas en values.txt:

```
9.0
2
1  SQLCHAR  0  10  "\t"          1  ID          SQL_Latin1_General_Cp437_BIN
2  SQLCHAR  0  40  "\r\n"        2  Description  SQL_Latin1_General_Cp437_BIN
```

Lee el archivo json usando OPENROWSET (BULK)

Puede usar OPENROWSET para leer el contenido del archivo y pasarlo a alguna otra función que analice los resultados.

El siguiente ejemplo muestra cómo leer todo el contenido del archivo JSON utilizando OPENROWSET (BULK) y luego proporcionar BulkColumn a la función OPENJSON que analizará JSON y devolverá las columnas:

```
SELECT book.*
FROM OPENROWSET (BULK 'C:\JSON\Books\books.json', SINGLE_CLOB) as j
CROSS APPLY OPENJSON(BulkColumn)
WITH( id nvarchar(100), name nvarchar(100), price float,
      pages int, author nvarchar(100)) AS book
```

Lea Importación a granel en línea: <https://riptutorial.com/es/sql-server/topic/7330/importacion-a-granel>

Capítulo 46: Índice

Examples

Crear índice agrupado

Con un índice agrupado, las páginas de hojas contienen las filas de la tabla real. Por lo tanto, solo puede haber un índice agrupado.

```
CREATE TABLE Employees
(
    ID CHAR(900),
    FirstName NVARCHAR(3000),
    LastName NVARCHAR(3000),
    StartYear CHAR(900)
)
GO

CREATE CLUSTERED INDEX IX_Clustered
ON Employees(ID)
GO
```

Crear índice no agrupado

Los índices no agrupados tienen una estructura separada de las filas de datos. Un índice no agrupado contiene los valores de clave de índice no agrupado y cada entrada de valor clave tiene un puntero a la fila de datos que contiene el valor clave. Puede haber un máximo de 999 índices no agrupados en SQL Server 2008/2012.

Enlace de referencia: <https://msdn.microsoft.com/en-us/library/ms143432.aspx>

```
CREATE TABLE Employees
(
    ID CHAR(900),
    FirstName NVARCHAR(3000),
    LastName NVARCHAR(3000),
    StartYear CHAR(900)
)
GO

CREATE NONCLUSTERED INDEX IX_NonClustered
ON Employees(StartYear)
GO
```

Mostrar información del índice

```
SP_HELPINDEX tableName
```

Índice a la vista

```

CREATE VIEW View_Index02
WITH SCHEMABINDING
AS
SELECT c.CompanyName, o.OrderDate, o.OrderID, od.ProductID
FROM dbo.Customers C
INNER JOIN dbo.orders O ON c.CustomerID=o.CustomerID
INNER JOIN dbo.[Order Details] od ON o.OrderID=od.OrderID
GO

CREATE UNIQUE CLUSTERED INDEX IX1 ON
View_Index02 (OrderID, ProductID)

```

Índice de caída

```
DROP INDEX IX_NonClustered ON Employees
```

Devuelve los índices de tamaño y fragmentación.

```

sys.dm_db_index_physical_stats (
    { database_id | NULL | 0 | DEFAULT }
, { object_id | NULL | 0 | DEFAULT }
, { index_id | NULL | 0 | -1 | DEFAULT }
, { partition_number | NULL | 0 | DEFAULT }
, { mode | NULL | DEFAULT }
)

```

Sample :

```

SELECT * FROM sys.dm_db_index_physical_stats
(DB_ID(N'DBName'), OBJECT_ID(N'IX_NonClustered '), NULL, NULL , 'DETAILED');

```

Reorganizar y reconstruir índice

avg_fragmentation_in_percent value	Declaración correctiva
> 5% y <= 30%	REORGANIZAR
> 30%	RECONSTRUIR

```
ALTER INDEX IX_NonClustered ON tableName REORGANIZE;
```

```

ALTER INDEX ALL ON Production.Product
REBUILD WITH (FILLFACTOR = 80, SORT_IN_TEMPDB = ON,
STATISTICS_NORECOMPUTE = ON);

```

Reconstruir o reorganizar todos los índices en una tabla

La reconstrucción de los índices se realiza utilizando la siguiente declaración

```
ALTER INDEX All ON tableName REBUILD;
```

Esto elimina el índice y lo vuelve a crear, eliminando la administración, recupera el espacio en disco y vuelve a ordenar las páginas de índice.

También se puede reorganizar un índice usando

```
ALTER INDEX All ON tableName REORGANIZE;
```

que utilizará recursos mínimos del sistema y desfragmentará el nivel de hoja de los índices agrupados y no agrupados en tablas y vistas al reordenar físicamente las páginas de nivel de hoja para que coincidan con el orden lógico, de izquierda a derecha, de los nodos de hoja

Reconstruir toda la base de datos de índice

```
EXEC sp_MSForEachTable 'ALTER INDEX ALL ON ? REBUILD'
```

Índice de investigaciones

Podría usar "SP_HELPINDEX Table_Name", pero Kimberly Tripp tiene un procedimiento almacenado (que se puede encontrar [aquí](#)), que es un mejor ejemplo, ya que muestra más información sobre los índices, incluidas las columnas y la definición del filtro, por ejemplo:

Uso:

```
USE Adventureworks  
EXEC sp_SQLskills_SQL2012_helpindex 'dbo.Product'
```

Alternativamente, Tibor Karaszi tiene un procedimiento almacenado (que se encuentra [aquí](#)). El último también mostrará información sobre el uso del índice y, opcionalmente, proporcionará una lista de sugerencias de índice. Uso:

```
USE Adventureworks  
EXEC sp_indexinfo 'dbo.Product'
```

Lea Índice en línea: <https://riptutorial.com/es/sql-server/topic/4998/indice>

Capítulo 47: Indización de texto completo

Examples

A. Creación de un índice único, un catálogo de texto completo y un índice de texto completo

El siguiente ejemplo crea un índice único en la columna JobCandidateID de la tabla HumanResources.JobCandidate de la base de datos de ejemplo AdventureWorks2012. Luego, el ejemplo crea un catálogo de texto completo predeterminado, ft. Finalmente, el ejemplo crea un índice de texto completo en la columna Reanudar, utilizando el catálogo de ft y la lista de paradas del sistema.

```
USE AdventureWorks2012;
GO
CREATE UNIQUE INDEX ui_ukJobCand ON HumanResources.JobCandidate (JobCandidateID);
CREATE FULLTEXT CATALOG ft AS DEFAULT;
CREATE FULLTEXT INDEX ON HumanResources.JobCandidate (Resume)
    KEY INDEX ui_ukJobCand
    WITH STOPLIST = SYSTEM;
GO
```

<https://www.simple-talk.com/sql/learn-sql-server/understanding-full-text-indexing-in-sql-server/>

<https://msdn.microsoft.com/en-us/library/cc879306.aspx>

<https://msdn.microsoft.com/en-us/library/ms142571.aspx>

Creación de un índice de texto completo en varias columnas de la tabla

```
USE AdventureWorks2012;
GO
CREATE FULLTEXT CATALOG production_catalog;
GO
CREATE FULLTEXT INDEX ON Production.ProductReview
(
    ReviewerName
        Language 1033,
    EmailAddress
        Language 1033,
    Comments
        Language 1033
)
KEY INDEX PK_ProductReview_ProductReviewID
ON production_catalog;
GO
```

Creación de un índice de texto completo con una lista de propiedades de búsqueda sin rellenarla

```
USE AdventureWorks2012;
GO
CREATE FULLTEXT INDEX ON Production.Document
(
    Title
        Language 1033,
    DocumentSummary
        Language 1033,
    Document
        TYPE COLUMN FileExtension
        Language 1033
)
KEY INDEX PK_Document_DocumentID
    WITH STOPLIST = SYSTEM, SEARCH PROPERTY LIST = DocumentPropertyList, CHANGE_TRACKING
OFF, NO POPULATION;
GO
```

Y poblando luego con

```
ALTER FULLTEXT INDEX ON Production.Document SET CHANGE_TRACKING AUTO;
GO
```

Búsqueda de texto completo

```
SELECT product_id
FROM products
WHERE CONTAINS(product_description, "Snap Happy 100EZ" OR FORMSOF(THESAURUS,'Snap Happy') OR
'100EZ')
AND product_cost < 200 ;
```

```
SELECT candidate_name,SSN
FROM candidates
WHERE CONTAINS(candidate_resume,"SQL Server") AND candidate_division =DBA;
```

Para obtener más información detallada <https://msdn.microsoft.com/en-us/library/ms142571.aspx>

Lea Indización de texto completo en línea: <https://riptutorial.com/es/sql-server/topic/4557/indizacion-de-texto-completo>

Capítulo 48: Insertar

Examples

Agregar una fila a una tabla llamada Facturas

```
INSERT INTO Invoices [ /* column names may go here */ ]  
VALUES (123, '1234abc', '2016-08-05 20:18:25.770', 321, 5, '2016-08-04');
```

- Los nombres de columna son obligatorios si la tabla en la que está insertando contiene una columna con el atributo IDENTIDAD.

```
INSERT INTO Invoices ([ID], [Num], [DateTime], [Total], [Term], [DueDate])  
VALUES (123, '1234abc', '2016-08-05 20:18:25.770', 321, 5, '2016-08-25');
```

Lea Insertar en línea: <https://riptutorial.com/es/sql-server/topic/5323/insertar>

Capítulo 49: INSERTAR EN

Introducción

La instrucción INSERT INTO se utiliza para insertar nuevos registros en una tabla.

Examples

INSERTAR la tabla Hello World INTO

```
CREATE TABLE MyTableName
(
    Id INT,
    MyColumnName NVARCHAR(1000)
)
GO

INSERT INTO MyTableName (Id, MyColumnName)
VALUES (1, N'Hello World!')
GO
```

INSERTAR en columnas específicas

Para realizar una inserción en columnas específicas (a diferencia de todas ellas), debe especificar las columnas que desea actualizar.

```
INSERT INTO USERS (FIRST_NAME, LAST_NAME)
VALUES ('Stephen', 'Jiang');
```

Esto solo funcionará si las columnas que no enumeró son anulables, identidad, tipo de datos de marca de tiempo o columnas computadas; o columnas que tienen una restricción de valor por defecto. Por lo tanto, si alguno de ellos no es anulable, no tiene identidad, marca de tiempo, no computado, no tiene valores de valores predeterminados ... entonces, intentar este tipo de inserción activará un mensaje de error que le indicará que debe proporcionar un valor para el (los) campo (s) aplicable (s).

INSERTAR múltiples filas de datos

Para insertar varias filas de datos en SQL Server 2008 o posterior:

```
INSERT INTO USERS VALUES
(2, 'Michael', 'Blythe'),
(3, 'Linda', 'Mitchell'),
(4, 'Jillian', 'Carson'),
(5, 'Garrett', 'Vargas');
```

Para insertar varias filas de datos en versiones anteriores de SQL Server, use "UNION ALL" así:

```
INSERT INTO USERS (FIRST_NAME, LAST_NAME)
SELECT 'James', 'Bond' UNION ALL
SELECT 'Miss', 'Money Penny' UNION ALL
SELECT 'Raoul', 'Silva'
```

Tenga en cuenta que la palabra clave "INTO" es opcional en las consultas INSERT. Otra advertencia es que el servidor SQL solo admite 1000 filas en un INSERT para que tenga que dividirlos en lotes.

INSERTAR una sola fila de datos

Una sola fila de datos se puede insertar de dos maneras:

```
INSERT INTO USERS (Id, FirstName, LastName)
VALUES (1, 'Mike', 'Jones');
```

O

```
INSERT INTO USERS
VALUES (1, 'Mike', 'Jones');
```

Tenga en cuenta que la segunda instrucción de inserción solo permite los valores exactamente en el mismo orden que las columnas de la tabla, mientras que en la primera inserción, el orden de los valores se puede cambiar como:

```
INSERT INTO USERS (FirstName, LastName, Id)
VALUES ('Mike', 'Jones', 1);
```

Use SALIDA para obtener el nuevo ID

Al INSERTAR, puede usar `OUTPUT INSERTED.ColumnName` para obtener valores de la fila recién insertada, por ejemplo, el Id. Recién generado: útil si tiene una columna de `IDENTITY` o cualquier tipo de valor predeterminado o calculado.

Al llamar a este programa (por ejemplo, desde ADO.net), lo trataría como una consulta normal y leería los valores como si hubiera hecho una declaración `SELECT`.

```
-- CREATE TABLE OutputTest ([Id] INT NOT NULL PRIMARY KEY IDENTITY, [Name] NVARCHAR(50))

INSERT INTO OutputTest ([Name])
OUTPUT INSERTED.[Id]
VALUES ('Testing')
```

Si se requiere el ID de la fila recientemente agregada dentro del mismo conjunto de consulta o procedimiento almacenado.

```
-- CREATE a table variable having column with the same datatype of the ID

DECLARE @LastId TABLE ( id int);
```

```
INSERT INTO OutputTest ([Name])
OUTPUT INSERTED.[Id] INTO @LastId
VALUES ('Testing')

SELECT id FROM @LastId

-- We can set the value in a variable and use later in procedure

DECLARE @LatestId int = (SELECT id FROM @LastId)
```

INSERTAR desde SELECCIONE los resultados de la consulta

Para insertar datos recuperados de la consulta SQL (filas simples o múltiples)

```
INSERT INTO Table_name (FirstName, LastName, Position)
SELECT FirstName, LastName, 'student' FROM Another_table_name
```

Tenga en cuenta que 'alumno' en SELECT es una constante de cadena que se insertará en cada fila.

Si es necesario, puede seleccionar e insertar datos de / en la misma tabla

Lea **INSERTAR EN** en línea: <https://riptutorial.com/es/sql-server/topic/3814/insertar-en>

Capítulo 50: Instalar SQL Server en Windows

Examples

Introducción

Estas son las ediciones disponibles de SQL Server, como lo indica la [Matriz de Ediciones](#) :

- Expreso: Base de datos gratuita de nivel de entrada. Incluye funcionalidad core-RDBMS. Limitado a 10G de tamaño de disco. Ideal para desarrollo y pruebas.
- Edición estándar: edición estándar con licencia. Incluye funcionalidad central y capacidades de Business Intelligence.
- Edición Enterprise: Edición SQL Server con todas las funciones. Incluye capacidades avanzadas de seguridad y almacenamiento de datos.
- Edición para desarrolladores: incluye todas las características de la Edición para empresas y sin limitaciones, y se puede [descargar y utilizar](#) de forma gratuita solo con fines de desarrollo.

Después de descargar / adquirir SQL Server, la instalación se ejecuta con SQLSetup.exe, que está disponible como una GUI o un programa de línea de comandos.

La instalación a través de cualquiera de estos requerirá que especifique una clave de producto y ejecute alguna configuración inicial que incluya funciones habilitadas, servicios separados y la configuración de los parámetros iniciales para cada uno de ellos. Los servicios y características adicionales se pueden habilitar en cualquier momento ejecutando el programa SQLSetup.exe en la línea de comandos o en la versión de la GUI.

Lea [Instalar SQL Server en Windows en línea](https://riptutorial.com/es/sql-server/topic/5801/instalar-sql-server-en-windows): <https://riptutorial.com/es/sql-server/topic/5801/instalar-sql-server-en-windows>

Capítulo 51: Instantáneas de la base de datos

Observaciones

Una instantánea de la base de datos es una vista estática de solo lectura de una base de datos de SQL Server que es transaccionalmente consistente con la base de datos de origen desde el momento de la creación de la instantánea.

Una instantánea de la base de datos siempre reside en la misma instancia del servidor que su base de datos de origen. A medida que se actualiza la base de datos de origen, la instantánea de la base de datos se actualiza.

Una instantánea difiere de una copia de seguridad ya que el proceso de creación de instantáneas es instantáneo y la instantánea ocupa espacio solo a medida que se aplican los cambios en la base de datos de origen. Una copia de seguridad, por otro lado, almacena una copia completa de los datos en el momento de la creación de la copia de seguridad.

Además, una instantánea proporciona una copia instantánea de solo lectura de la base de datos, mientras que una copia de seguridad debe restaurarse en un servidor para que sea legible (y una vez restaurada también se puede escribir en)

Las instantáneas de la base de datos solo están disponibles en las ediciones Enterprise y Developer.

Examples

Crear una instantánea de base de datos

Una instantánea de la base de datos es una vista estática de solo lectura de una base de datos de SQL Server (la base de datos de origen). Es similar a la copia de seguridad, pero está disponible como cualquier otra base de datos para que el cliente pueda consultar la base de datos de instantáneas.

```
CREATE DATABASE MyDatabase_morning -- name of the snapshot
ON (
    NAME=MyDatabase_data, -- logical name of the data file of the source database
    FILENAME='C:\SnapShots\MySnapshot_Data.ss' -- snapshot file;
)
AS SNAPSHOT OF MyDatabase; -- name of source database
```

También puede crear una instantánea de la base de datos con varios archivos:

```
CREATE DATABASE MyMultiFileDBSnapshot ON
(NAME=MyMultiFileDb_ft, FILENAME='C:\SnapShots\MyMultiFileDb_ft.ss'),
(NAME=MyMultiFileDb_sys, FILENAME='C:\SnapShots\MyMultiFileDb_sys.ss'),
(NAME=MyMultiFileDb_data, FILENAME='C:\SnapShots\MyMultiFileDb_data.ss'),
(NAME=MyMultiFileDb_indx, FILENAME='C:\SnapShots\MyMultiFileDb_indx.ss')
```

```
AS SNAPSHOT OF MultiFileDb;
```

Restaurar una instantánea de base de datos

Si los datos en una base de datos de origen se dañan o algunos datos incorrectos se escriben en la base de datos, en algunos casos, revertir la base de datos a una instantánea de la base de datos anterior al daño podría ser una alternativa adecuada para restaurar la base de datos desde una copia de seguridad.

```
RESTORE DATABASE MYDATABASE FROM DATABASE_SNAPSHOT='MyDatabase_morning';
```

Advertencia: ¡ Esto *eliminará todos los cambios* realizados en la base de datos de origen desde que se tomó la instantánea!

BORRAR Instantánea

Puede eliminar las instantáneas existentes de la base de datos utilizando la declaración DELETE DATABASE:

```
DROP DATABASE Mydatabase_morning
```

En esta declaración debe hacer referencia al nombre de la instantánea de la base de datos.

Lea *Instantáneas de la base de datos en línea*: <https://riptutorial.com/es/sql-server/topic/677/instantaneas-de-la-base-de-datos>

Capítulo 52: Instrucción SELECT

Introducción

En SQL, las instrucciones `SELECT` devuelven conjuntos de resultados de colecciones de datos como tablas o vistas. `SELECT` se pueden usar con varias otras cláusulas como `WHERE`, `GROUP BY` u `ORDER BY` para refinar aún más los resultados deseados.

Examples

SELECT básico de la tabla

Seleccione todas las columnas de alguna tabla (tabla del sistema en este caso):

```
SELECT *
FROM sys.objects
```

O seleccione solo algunas columnas específicas:

```
SELECT object_id, name, type, create_date
FROM sys.objects
```

Filtrar filas usando la cláusula WHERE

La cláusula `WHERE` filtra solo aquellas filas que satisfacen alguna condición:

```
SELECT *
FROM sys.objects
WHERE type = 'IT'
```

Ordenar resultados utilizando ORDENAR POR

La cláusula `ORDER BY` ordena las filas en el conjunto de resultados devuelto por alguna columna o expresión:

```
SELECT *
FROM sys.objects
ORDER BY create_date
```

Resultado de grupo utilizando GROUP BY

La cláusula `GROUP BY` agrupa las filas por algún valor:

```
SELECT type, count(*) as c
FROM sys.objects
```

```
GROUP BY type
```

Puede aplicar alguna función en cada grupo (función agregada) para calcular la suma o el recuento de los registros en el grupo.

tipo	do
SQ	3
S	72
ESO	dieciséis
PK	1
U	5

Filtrar grupos utilizando la cláusula HAVING.

La cláusula HAVING elimina los grupos que no satisfacen la condición:

```
SELECT type, count(*) as c
FROM sys.objects
GROUP BY type
HAVING count(*) < 10
```

tipo	do
SQ	3
PK	1
U	5

Devolviendo solo las primeras N filas

La cláusula TOP devuelve solo las primeras N filas en el resultado:

```
SELECT TOP 10 *
FROM sys.objects
```

Paginación utilizando OFFSET FETCH

La cláusula OFFSET FETCH es una versión más avanzada de TOP. Le permite omitir filas N1 y tomar las siguientes filas N2:

```
SELECT *
FROM sys.objects
```



```
ORDER BY object_id  
OFFSET 50 ROWS FETCH NEXT 10 ROWS ONLY
```

Puede usar **OFFSET** sin recuperación para saltar las primeras 50 filas:

```
SELECT *  
FROM sys.objects  
ORDER BY object_id  
OFFSET 50 ROWS
```

SELECT sin FROM (sin fuente de datos)

La instrucción **SELECT** se puede ejecutar sin la cláusula **FROM**:

```
declare @var int = 17;  
  
SELECT @var as c1, @var + 2 as c2, 'third' as c3
```

En este caso, se devuelve una fila con valores / resultados de expresiones.

Lea **Instrucción SELECT** en línea: <https://riptutorial.com/es/sql-server/topic/4662/instruccion-select>

Capítulo 53: Integración de Common Language Runtime

Examples

Habilitar CLR en la base de datos

Los procedimientos CLR no están habilitados por defecto. Debe ejecutar las siguientes consultas para habilitar CLR:

```
sp_configure 'show advanced options', 1;
GO
RECONFIGURE;
GO
sp_configure 'clr enabled', 1;
GO
RECONFIGURE;
GO
```

Además, si algún módulo CLR necesita acceso externo, debe establecer la propiedad TRUSTWORTHY en ON en su base de datos:

```
ALTER DATABASE MyDbWithClr SET TRUSTWORTHY ON
```

Adición de .dll que contiene módulos Sql CLR

Los procedimientos, funciones, disparadores y tipos escritos en lenguajes .Net se almacenan en archivos .dll. Una vez que cree el archivo .dll que contiene procedimientos CLR, debe importarlo a SQL Server:

```
CREATE ASSEMBLY MyLibrary
FROM 'C:\lib\MyStoredProcedures.dll'
WITH PERMISSION_SET = EXTERNAL_ACCESS
```

PERMISSION_SET es seguro por defecto, lo que significa que el código en .dll no necesita permiso para acceder a recursos externos (por ejemplo, archivos, sitios web, otros servidores) y que no utilizará código nativo que pueda acceder a la memoria.

PERMISSION_SET = EXTERNAL_ACCESS se usa para marcar ensamblajes que contienen código que accederá a recursos externos.

Puede encontrar información sobre los archivos de ensamblaje de CLR actuales en la vista de sys.assemblies:

```
SELECT *
FROM sys.assemblies asms
```

```
WHERE is_user_defined = 1
```

Crear función CLR en SQL Server

Si creó la función .Net, la compiló en .dll y la importó en el servidor SQL como un ensamblaje, puede crear una función definida por el usuario que haga referencia a la función en ese ensamblaje:

```
CREATE FUNCTION dbo.TextCompress(@input nvarchar(max))  
RETURNS varbinary(max)  
AS EXTERNAL NAME MyLibrary.[Name.Space.ClassName].TextCompress
```

Debe especificar el nombre de la función y la firma con los parámetros de entrada y los valores de retorno que coincidan con la función .Net. En la cláusula AS EXTERNAL NAME, debe especificar el nombre del conjunto, el espacio de nombres / nombre de la clase donde se coloca esta función y el nombre del método en la clase que contiene el código que se expondrá como función.

Puede encontrar información sobre las funciones CLR usando la siguiente consulta:

```
SELECT * FROM dbo.sysobjects WHERE TYPE = 'FS'
```

Crear CLR tipo definido por el usuario en SQL Server

Si ha creado una clase .Net que represente algún tipo definido por el usuario, lo compiló en .dll y lo importó en el servidor SQL como un ensamblaje, puede crear una función definida por el usuario que haga referencia a esta clase:

```
CREATE TYPE dbo.Point  
EXTERNAL NAME MyLibrary.[Name.Space.Point]
```

Debe especificar el nombre del tipo que se utilizará en las consultas T-SQL. En la cláusula EXTERNAL NAME, debe especificar el nombre del conjunto, el espacio de nombres y el nombre de la clase.

Crear procedimiento CLR en SQL Server

Si ha creado el método .Net en alguna clase, lo ha compilado en .dll y lo ha importado en el servidor SQL como un conjunto, puede crear un procedimiento almacenado definido por el usuario que haga referencia al método en ese conjunto:

```
CREATE PROCEDURE dbo.DoSomething(@input nvarchar(max))  
AS EXTERNAL NAME MyLibrary.[Name.Space.ClassName].DoSomething
```

Debe especificar el nombre del procedimiento y la firma con parámetros de entrada que coincidan con el método .Net. En la cláusula AS EXTERNAL NAME, debe especificar el nombre del conjunto, el espacio de nombres / nombre de la clase donde se coloca este procedimiento y el nombre del método en la clase que contiene el código que se expondrá como procedimiento.

Lea Integración de Common Language Runtime en línea: <https://riptutorial.com/es/sql-server/topic/7116/integracion-de-common-language-runtime>

Capítulo 54: JSON en Sql Server

Sintaxis

- **JSON_VALUE** (expresión, ruta): extrae un valor escalar de una cadena JSON.
- **JSON_QUERY** (expresión [, ruta]) - Extrae un objeto o una matriz de una cadena JSON.
- **OPENJSON** (jsonExpression [, ruta]) - función de valor de tabla que analiza el texto JSON y devuelve objetos y propiedades en JSON como filas y columnas.
- **ISJSON** (expresión): **comprueba** si una cadena contiene JSON válido.
- **JSON_MODIFY** (expresión, ruta, newValue): actualiza el valor de una propiedad en una cadena JSON y devuelve la cadena JSON actualizada.

Parámetros

Parámetros	Detalles
expresión	Normalmente, el nombre de una variable o una columna que contiene texto JSON.
camino	Una expresión de ruta JSON que especifica la propiedad para actualizar. ruta tiene la siguiente sintaxis: [añadir] [lax estricto] \$. <ruta json>
jsonexpression	Es una expresión de caracteres Unicode que contiene el texto JSON.

Observaciones

La función OPENJSON solo está disponible bajo el nivel de compatibilidad 130. Si su nivel de compatibilidad de la base de datos es inferior a 130, SQL Server no podrá encontrar y ejecutar la función OPENJSON. Actualmente, todas las bases de datos de Azure SQL están configuradas en 120 de forma predeterminada. Puede cambiar el nivel de compatibilidad de una base de datos usando el siguiente comando:

```
ALTER DATABASE <Database-Name-Here> SET COMPATIBILITY_LEVEL = 130
```

Examples

Formato de resultados de consultas como JSON con FOR JSON

Datos de la tabla de entrada (tabla de personas)

Carné de identidad	Nombre	Años
1	Juan	23

Carné de identidad	Nombre	Años
2	Jane	31

Consulta

```
SELECT Id, Name, Age
FROM People
FOR JSON PATH
```

Resultado

```
[
  {"Id":1,"Name":"John","Age":23},
  {"Id":2,"Name":"Jane","Age":31}
]
```

Analizar texto JSON

Las funciones **JSON_VALUE** y **JSON_QUERY** analizan el texto JSON y devuelven valores **escalables** u objetos / matrices en la ruta en el texto JSON.

```
DECLARE @json NVARCHAR(100) = '{"id": 1, "user":{"name":"John"}, "skills":["C#","SQL"]}'

SELECT
  JSON_VALUE(@json, '$.id') AS Id,
  JSON_VALUE(@json, '$.user.name') AS Name,
  JSON_QUERY(@json, '$.user') AS UserObject,
  JSON_QUERY(@json, '$.skills') AS Skills,
  JSON_VALUE(@json, '$.skills[0]') AS Skill0
```

Resultado

Carné de identidad	Nombre	UserObject	Habilidades	Habilidad0
1	Juan	{"nombre": "Juan"}	["C #", "SQL"]	DO#

Únase a las entidades JSON principales y secundarias utilizando CROSS APPLY OPENJSON

Unir los objetos principales con sus entidades secundarias, por ejemplo, queremos una tabla relacional de cada persona y sus aficiones

```
DECLARE @json nvarchar(1000) =
N' [
  {
    "id":1,
    "user":{"name":"John"},
    "hobbies":[
      {"name": "Reading"},
      {"name": "Surfing"}
    ]
  }
]
```

```

    ]
  },
  {
    "id":2,
    "user":{"name":"Jane"},
    "hobbies":[
      {"name": "Programming"},
      {"name": "Running"}
    ]
  }
]'
```

Consulta

```

SELECT
  JSON_VALUE(person.value, '$.id') as Id,
  JSON_VALUE(person.value, '$.user.name') as PersonName,
  JSON_VALUE(hobbies.value, '$.name') as Hobby
FROM OPENJSON (@json) as person
  CROSS APPLY OPENJSON(person.value, '$.hobbies') as hobbies
```

Alternativamente, esta consulta puede escribirse usando la cláusula WITH.

```

SELECT
  Id, person.PersonName, Hobby
FROM OPENJSON (@json)
WITH(
  Id int '$.id',
  PersonName nvarchar(100) '$.user.name',
  Hobbies nvarchar(max) '$.hobbies' AS JSON
) as person
  CROSS APPLY OPENJSON(Hobbies)
WITH(
  Hobby nvarchar(100) '$.name'
)
```

Resultado

Carné de identidad	Nombre de la persona	Hobby
1	Juan	Leyendo
1	Juan	Surf
2	Jane	Programación
2	Jane	Corriendo

Índice en las propiedades JSON mediante el uso de columnas calculadas

Al almacenar documentos JSON en SQL Server, debemos poder filtrar y clasificar de manera eficiente los resultados de las consultas en las propiedades de los documentos JSON.

```
CREATE TABLE JsonTable
(
    id int identity primary key,
    jsonInfo nvarchar(max),
    CONSTRAINT [Content should be formatted as JSON]
    CHECK (ISJSON(jsonInfo)>0)
)
```

```
INSERT INTO JsonTable
VALUES (N'{"Name":"John","Age":23}'),
(N'{"Name":"Jane","Age":31}'),
(N'{"Name":"Bob","Age":37}'),
(N'{"Name":"Adam","Age":65}')
GO
```

Dada la tabla anterior Si queremos encontrar la fila con el nombre = 'Adam', ejecutaríamos la siguiente consulta.

```
SELECT *
FROM JsonTable Where
JSON_VALUE(jsonInfo, '$.Name') = 'Adam'
```

Sin embargo, esto requerirá que el servidor SQL realice una tabla completa que en una tabla grande no es eficiente.

Para acelerar esto, nos gustaría agregar un índice, sin embargo, no podemos hacer referencia directamente a las propiedades en el documento JSON. La solución es agregar una columna computada en la ruta JSON `$.Name` , luego agregar un índice en la columna computada.

```
ALTER TABLE JsonTable
ADD vName as JSON_VALUE(jsonInfo, '$.Name')

CREATE INDEX idx_name
ON JsonTable(vName)
```

Ahora, cuando ejecutamos la misma consulta, en lugar de una tabla completa, el servidor SQL usa un índice para buscar en el índice no agrupado y encontrar las filas que satisfacen las condiciones especificadas.

Nota: para que el servidor SQL use el índice, debe crear la columna calculada con la misma expresión que planea usar en sus consultas; en este ejemplo, `JSON_VALUE(jsonInfo, '$.Name')` , también puede usar el nombre de la columna computada `vName`

Formatee una fila de la tabla como un solo objeto JSON usando FOR JSON

La opción **WITHOUT_ARRAY_WRAPPER** en la cláusula *FOR JSON* eliminará los soportes de matriz de la salida JSON. Esto es útil si está devolviendo una sola fila en la consulta.

Nota: esta opción producirá una salida JSON no válida si se devuelve más de una fila.

Datos de la tabla de entrada (tabla de personas)

Carné de identidad	Nombre	Años
1	Juan	23
2	Jane	31

Consulta

```
SELECT Id, Name, Age
FROM People
WHERE Id = 1
FOR JSON PATH, WITHOUT_ARRAY_WRAPPER
```

Resultado

```
{"Id":1,"Name":"John","Age":23}
```

Analizar texto JSON utilizando la función OPENJSON.

La función **OPENJSON** analiza el texto JSON y devuelve varias salidas. Los valores que deben devolverse se especifican utilizando las rutas definidas en la cláusula **WITH**. Si no se especifica una ruta para alguna columna, el nombre de la columna se usa como una ruta. Esta función convierte valores devueltos a los tipos de SQL definidos en la cláusula **WITH**. La opción **AS JSON** debe especificarse en la definición de columna si se debe devolver algún objeto / matriz.

```
DECLARE @json NVARCHAR(100) = '{"id": 1, "user":{"name":"John"}, "skills":["C#","SQL"]}'

SELECT *
FROM OPENJSON (@json)
  WITH(Id int '$.id',
       Name nvarchar(100) '$.user.name',
       UserObject nvarchar(max) '$.user' AS JSON,
       Skills nvarchar(max) '$.skills' AS JSON,
       Skill0 nvarchar(20) '$.skills[0]')
```

Resultado

Carné de identidad	Nombre	UserObject	Habilidades	Habilidad0
1	Juan	{"nombre": "Juan"}	["C #", "SQL"]	DO#

Lea JSON en Sql Server en línea: <https://riptutorial.com/es/sql-server/topic/2568/json-en-sql-server>

Capítulo 55: JUNTARSE

Sintaxis

- COALESCE ([Column1], [Column2] [ColumnN])

Examples

Usando COALESCE para construir una cadena delimitada por comas

Podemos obtener una cadena delimitada por comas de varias filas usando la fusión como se muestra a continuación.

Como se usa la variable de tabla, necesitamos ejecutar una consulta completa una vez. Así que para que sea fácil de entender, he agregado el bloque BEGIN y END.

```
BEGIN

--Table variable declaration to store sample records
DECLARE @Table TABLE (FirstName varchar(256), LastName varchar(256))

--Inserting sample records into table variable @Table
INSERT INTO @Table (FirstName, LastName)
VALUES
('John','Smith'),
('Jane','Doe')

--Creating variable to store result
DECLARE @Names varchar(4000)

--Used COALESCE function, so it will concatenate comma separated FirstName into @Names
variable
SELECT @Names = COALESCE(@Names + ', ', '') + FirstName
FROM @Table

--Now selecting actual result
SELECT @Names
END
```

Ejemplo básico de coalesce

COALESCE() devuelve el primer valor `NON NULL` en una lista de argumentos. Supongamos que tuviéramos una tabla con números de teléfono y números de teléfono celular y quisiéramos devolver solo uno para cada usuario. Para obtener solo uno, podemos obtener el primer valor `NON NULL`.

```
DECLARE @Table TABLE (UserID int, PhoneNumber varchar(12), CellNumber varchar(12))
INSERT INTO @Table (UserID, PhoneNumber, CellNumber)
VALUES
(1, '555-869-1123', NULL),
```

```
(2, '555-123-7415', '555-846-7786'),  
(3, NULL, '555-456-8521')
```

```
SELECT  
    UserID,  
    COALESCE(PhoneNumber, CellNumber)  
FROM  
    @Table
```

Obtener el primero no nulo de una lista de valores de columna

```
SELECT COALESCE(NULL, NULL, 'TechOnTheNet.com', NULL, 'CheckYourMath.com');  
Result: 'TechOnTheNet.com'
```

```
SELECT COALESCE(NULL, 'TechOnTheNet.com', 'CheckYourMath.com');  
Result: 'TechOnTheNet.com'
```

```
SELECT COALESCE(NULL, NULL, 1, 2, 3, NULL, 4);  
Result: 1
```

Lea **JUNTARSE** en línea: <https://riptutorial.com/es/sql-server/topic/3234/juntarse>

Capítulo 56: La función COSAS

Parámetros

Parámetro	Detalles
expresión_caracteres	la cadena existente en sus datos
Posición de salida	la posición en <code>character_expression</code> para eliminar la <code>length</code> y luego insertar la <code>replacement_string</code>
longitud	el número de caracteres que se eliminarán de la expresión de <code>character_expression</code>
reemplazo de cadena	La secuencia de caracteres para insertar en <code>character_expression</code>

Examples

Reemplazo de personaje básico con STUFF ()

La función `STUFF()` inserta una cadena en otra cadena al eliminar primero un número específico de caracteres. El siguiente ejemplo, elimina "Svr" y lo reemplaza por "Servidor". Esto sucede al especificar la posición de `start_position` y la `length` del reemplazo.

```
SELECT STUFF('SQL Svr Documentation', 5, 3, 'Server')
```

La ejecución de este ejemplo dará como resultado la devolución de la `SQL Server Documentation` lugar de la `SQL Server Documentation` de `SQL Svr Documentation`.

Uso de FOR XML para concatenar valores de varias filas

Un uso común de la función `FOR XML` es concatenar los valores de varias filas.

Aquí hay un ejemplo usando la [tabla Clientes](#) :

```
SELECT
  STUFF( (SELECT ';' + Email
          FROM Customers
          where (Email is not null and Email <> ''))
        ORDER BY Email ASC
        FOR XML PATH('')),
  1, 1, ''
```

En el ejemplo anterior, `FOR XML PATH('')` se usa para concatenar direcciones de correo electrónico, usando ; Como el caracter delimitador. Además, el propósito de `STUFF` es eliminar el líder ; De la cadena concatenada. `STUFF` también está `STUFF` implícitamente la cadena concatenada

de XML a varchar.

Nota: el resultado del ejemplo anterior será XML codificado, lo que significa que reemplazará < caracteres con < etc. Si no desea esto, cambie FOR XML PATH('') a FOR XML PATH, TYPE).value('.[1]', 'varchar(MAX)') , por ejemplo:

```
SELECT
  STUFF( (SELECT ';' + Email
        FROM Customers
        where (Email is not null and Email <> ''))
        ORDER BY Email ASC
        FOR XML PATH, TYPE).value('.[1]', 'varchar(900)'),
  1, 1, '')
```

Esto se puede usar para lograr un resultado similar a `GROUP_CONCAT` en MySQL o `string_agg` en PostgreSQL 9.0+, aunque usamos subconsultas en lugar de agregados de GROUP BY. (Como alternativa, puede instalar un agregado definido por el usuario como [este](#) si está buscando una funcionalidad más cercana a la de `GROUP_CONCAT`).

Obtener nombres de columna separados por comas (no una lista)

```
/*
The result can be use for fast way to use columns on Insertion/Updates.
Works with tables and views.

Example: eTableColumns 'Customers'
ColumnNames
-----
Id, FName, LName, Email, PhoneNumber, PreferredContact

INSERT INTO Customers (Id, FName, LName, Email, PhoneNumber, PreferredContact)
VALUES (5, 'Ringo', 'Star', 'two@beatles.now', NULL, 'EMAIL')
*/
CREATE PROCEDURE eTableColumns (@Table VARCHAR(100))
AS
SELECT ColumnNames =
  STUFF( (SELECT ', ' + c.name
        FROM
          sys.columns c
        INNER JOIN
          sys.types t ON c.user_type_id = t.user_type_id
        WHERE
          c.object_id = OBJECT_ID( @Table)
          FOR XML PATH, TYPE).value('.[1]', 'varchar(2000)'),
  1, 1, '')
GO
```

Cosas para comas separadas en servidor sql

FOR XML PATH y STUFF para concatenar las múltiples filas en una sola fila:

```
select distinct t1.id,
  STUFF(
    (SELECT ', ' + convert(varchar(10), t2.date, 120)
```

```
FROM yourtable t2
where t1.id = t2.id
FOR XML PATH ('')
, 1, 1, '') AS date
from yourtable t1;
```

Ejemplo básico de la función STUFF ().

STUFF (Original_Expression, Start, Length, Replacement_expression)

La función STUFF () inserta Replacement_expression, en la posición de inicio especificada, junto con la eliminación de los caracteres especificados usando el parámetro Length.

```
Select FirstName, LastName,Email, STUFF(Email, 2, 3, '*****') as StuffedEmail From Employee
```

Ejecutar este ejemplo resultará en devolver la tabla dada

Nombre de pila	Apellido	Email	StuffedEmail
Jomes	Cazador	James@hotmail.com	J*****s@hotmail.com
Shyam	Rathod	Shyam@hotmail.com	S*****m@hotmail.com
RAM	brillante	Ram@hotmail.com	R ***** hotmail.com

Lea La función COSAS en línea: <https://riptutorial.com/es/sql-server/topic/703/la-funcion-cosas>

Capítulo 57: Lectura fantasma

Introducción

En los sistemas de bases de datos, el aislamiento determina cómo la integridad de la transacción es visible para otros usuarios y sistemas, por lo que define cómo / cuándo los cambios realizados por una operación se vuelven visibles por otros. La lectura fantasma puede ocurrir cuando obtiene datos que aún no están comprometidos con la base de datos.

Observaciones

Puede leer los distintos `ISOLATION LEVEL` en [MSDN](#)

Examples

Nivel de aislamiento LEER SIN COMPROMISO

Crear una tabla de muestra en una base de datos de muestra

```
CREATE TABLE [dbo].[Table_1](
  [Id] [int] IDENTITY(1,1) NOT NULL,
  [title] [varchar](50) NULL,
  CONSTRAINT [PK_Table_1] PRIMARY KEY CLUSTERED
(
  [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

Ahora abra un Editor de primera consulta (en la base de datos) inserte el código a continuación y ejecute (**no toque el --rollback**) en este caso, inserte una fila en la base de datos pero **no** confirme los cambios.

```
begin tran

INSERT INTO Table_1 values('Title 1')

SELECT * FROM [Test].[dbo].[Table_1]

--rollback
```

Ahora abra un Segundo Editor de consultas (en la base de datos), inserte el código a continuación y ejecute

```
begin tran

set transaction isolation level READ UNCOMMITTED
```

```
SELECT * FROM [Test].[dbo].[Table_1]
```

Puede observar que en el segundo editor puede ver la fila recién creada (pero no confirmada) de la primera transacción. En el primer editor, ejecute el rollback (seleccione la palabra rollback y ejecute).

```
-- Rollback the first transaction  
rollback
```

Ejecute la consulta en el segundo editor y verá que el registro desaparece (lectura fantasma), esto ocurre porque le dice a la segunda transacción que obtenga todas las filas, también las no confirmadas.

Esto ocurre cuando cambias el nivel de aislamiento con

```
set transaction isolation level READ UNCOMMITTED
```

Lea **Lectura fantasma en línea**: <https://riptutorial.com/es/sql-server/topic/8235/lectura-fantasma>

Capítulo 58: Llaves extranjeras

Examples

Relación / restricción de clave externa

Las claves externas le permiten definir la relación entre dos tablas. Una tabla (principal) debe tener una clave principal que identifique de forma única las filas de la tabla. Otra tabla (secundaria) puede tener el valor de la clave primaria de la matriz en una de las columnas. La restricción FOREIGN KEY REFERENCES garantiza que los valores en la tabla secundaria deben existir como un valor de clave principal en la tabla principal.

En este ejemplo, tenemos la tabla de la Compañía principal con la clave principal de CompanyId y la tabla de Empleado secundario que tiene el ID de la compañía donde trabaja este empleado.

```
create table Company (  
    CompanyId int primary key,  
    Name nvarchar(200)  
)  
create table Employee (  
    EmployeeId int,  
    Name nvarchar(200),  
    CompanyId int  
        foreign key references Company(companyId)  
)
```

las referencias de clave externa garantizan que los valores insertados en la columna Employee.CompanyId también deben existir en la columna Company.CompanyId. Además, nadie puede eliminar la compañía en la tabla de la compañía si hay al menos un empleado con una companyId coincidente en la tabla secundaria.

La relación FOREIGN KEY garantiza que las filas de dos tablas no puedan "desvincularse".

Mantener la relación entre las filas padre / hijo

Supongamos que tenemos una fila en la tabla Empresa con companyId 1. Podemos insertar una fila en la tabla de empleados que tiene companyId 1:

```
insert into Employee values (17, 'John', 1)
```

Sin embargo, no podemos insertar un empleado que no tenga un ID de empresa:

```
insert into Employee values (17, 'John', 111111)
```

Msg 547, nivel 16, estado 0, línea 12 La instrucción INSERT entró en conflicto con la restricción FOREIGN KEY "FK__Employee__Compan__1EE485AA". El conflicto se produjo en la base de datos "MyDb", tabla "dbo.Company", columna 'CompanyId'. La instrucción se ha terminado.

Además, no podemos eliminar la fila principal en la tabla de la compañía siempre que haya al menos una fila secundaria en la tabla de empleados que la referencia.

```
delete from company where CompanyId = 1
```

Msg 547, nivel 16, estado 0, línea 14 La instrucción DELETE entró en conflicto con la restricción de REFERENCIA "FK__Employee__Compan__1EE485AA". El conflicto se produjo en la base de datos "MyDb", tabla "dbo.Employee", columna 'CompanyId'. La instrucción se ha terminado.

La relación de clave externa garantiza que las filas de la empresa y los empleados no se "desvincularán".

Agregar una relación de clave externa en la tabla existente

La restricción **FOREIGN KEY** se puede agregar a las tablas existentes que todavía no están en relación. Imagine que tenemos tablas de Empresa y Empleado en las que la columna de Id. Compañía de la tabla de Empleado no tiene una relación de clave externa. La declaración ALTER TABLE le permite agregar **una** restricción de **clave externa** en una columna existente que hace referencia a otra tabla y clave principal en esa tabla:

```
alter table Employee
    add foreign key (CompanyId) references Company(CompanyId)
```

Añadir clave externa en la tabla existente

Las columnas **FOREIGN KEY** con restricción se pueden agregar a las tablas existentes que aún no están en relación. Imagine que tenemos tablas de Empresa y Empleado donde la tabla de Empleado no tiene la columna CompanyId. La declaración ALTER TABLE le permite agregar una nueva columna con **una** restricción de **clave externa** que hace referencia a otra tabla y clave principal en esa tabla:

```
alter table Employee
    add CompanyId int foreign key references Company(CompanyId)
```

Obtener información sobre restricciones de clave externa

La vista del sistema sys.foreignkeys devuelve información sobre todas las relaciones de clave externa en la base de datos:

```
select name,
    OBJECT_NAME(referenced_object_id) as [parent table],
    OBJECT_NAME(parent_object_id) as [child table],
    delete_referential_action_desc,
    update_referential_action_desc
from sys.foreign_keys
```

Lea Llaves extranjeras en línea: <https://riptutorial.com/es/sql-server/topic/5355/llaves-extranjeras>

Capítulo 59: Llaves primarias

Observaciones

Las claves primarias se utilizan para identificar de forma única un registro en una tabla. Una tabla solo puede tener una única clave principal (aunque la clave principal puede constar de varias columnas), y se requiere una clave principal para ciertos tipos de replicación.

Las claves primarias a menudo se usan como (pero no tienen que ser) el [índice agrupado](#) en una tabla.

Examples

Crear tabla con columna de identidad como clave principal

```
-- Identity primary key - unique arbitrary increment number
create table person (
  id int identity(1,1) primary key not null,
  firstName varchar(100) not null,
  lastName varchar(100) not null,
  dob DateTime not null,
  ssn varchar(9) not null
)
```

Crear tabla con clave primaria GUID

```
-- GUID primary key - arbitrary unique value for table
create table person (
  id uniqueIdentifier default (newId()) primary key,
  firstName varchar(100) not null,
  lastName varchar(100) not null,
  dob DateTime not null,
  ssn varchar(9) not null
)
```

Crear mesa con llave natural.

```
-- natural primary key - using an existing piece of data within the table that uniquely
identifies the record
create table person (
  firstName varchar(100) not null,
  lastName varchar(100) not null,
  dob DateTime not null,
  ssn varchar(9) primary key not null
)
```

Crear tabla w / clave compuesta

```
-- composite key - using two or more existing columns within a table to create a primary key
create table person (
  firstName varchar(100) not null,
  lastName varchar(100) not null,
  dob DateTime not null,
  ssn varchar(9) not null,
  primary key (firstName, lastName, dob)
)
```

Añadir clave principal a la tabla existente

```
ALTER TABLE person
  ADD CONSTRAINT pk_PersonSSN PRIMARY KEY (ssn)
```

Tenga en cuenta que si la columna de clave principal (en este caso, `ssn`) tiene más de una fila con la misma clave candidata, la declaración anterior fallará, ya que los valores de la clave principal deben ser únicos.

Eliminar clave principal

```
ALTER TABLE Person
  DROP CONSTRAINT pk_PersonSSN
```

Lea Llaves primarias en línea: <https://riptutorial.com/es/sql-server/topic/4543/llaves-primarias>

Capítulo 60: Manejo de transacciones

Parámetros

Parámetro	Detalles
nombre de transacción	para nombrar su transacción - útil con el parámetro [<i>con marca</i>] que permitirá un registro significativo - distingue entre mayúsculas y minúsculas (!)
con marca [descripción']	se puede agregar a [nombre de <i>transacción</i>] y almacenará una marca en el registro

Examples

Esqueleto básico de transacciones con manejo de errores.

```
BEGIN TRY -- start error handling
    BEGIN TRANSACTION; -- from here on transactions (modifications) are not final
        -- start your statement(s)
        select 42/0 as ANSWER -- simple SQL Query with an error
        -- end your statement(s)
    COMMIT TRANSACTION; -- finalize all transactions (modifications)
END TRY -- end error handling -- jump to end
BEGIN CATCH -- execute this IF an error occurred
    ROLLBACK TRANSACTION; -- undo any transactions (modifications)
-- put together some information as a query
    SELECT
        ERROR_NUMBER() AS ErrorNumber
        ,ERROR_SEVERITY() AS ErrorSeverity
        ,ERROR_STATE() AS ErrorState
        ,ERROR_PROCEDURE() AS ErrorProcedure
        ,ERROR_LINE() AS ErrorLine
        ,ERROR_MESSAGE() AS ErrorMessage;

END CATCH; -- final line of error handling
GO -- execute previous code
```

Lea Manejo de transacciones en línea: <https://riptutorial.com/es/sql-server/topic/5859/manejo-de-transacciones>

Capítulo 61: Mientras bucle

Observaciones

El uso de un bucle `WHILE` u otro proceso iterativo no suele ser la forma más eficiente de procesar datos en SQL Server.

Debe preferir utilizar una consulta basada en conjuntos en los datos para lograr los mismos resultados, cuando sea posible

Examples

Usando While loop

El bucle `WHILE` se puede utilizar como una alternativa a los `CURSORS` . El siguiente ejemplo imprimirá números del 0 al 99.

```
DECLARE @i int = 0;
WHILE(@i < 100)
BEGIN
    PRINT @i;
    SET @i = @i+1
END
```

Mientras que el bucle con el uso mínimo de la función agregada

```
DECLARE @ID AS INT;

SET @ID = (SELECT MIN(ID) from TABLE);

WHILE @ID IS NOT NULL
BEGIN
    PRINT @ID;
    SET @ID = (SELECT MIN(ID) FROM TABLE WHERE ID > @ID);
END
```

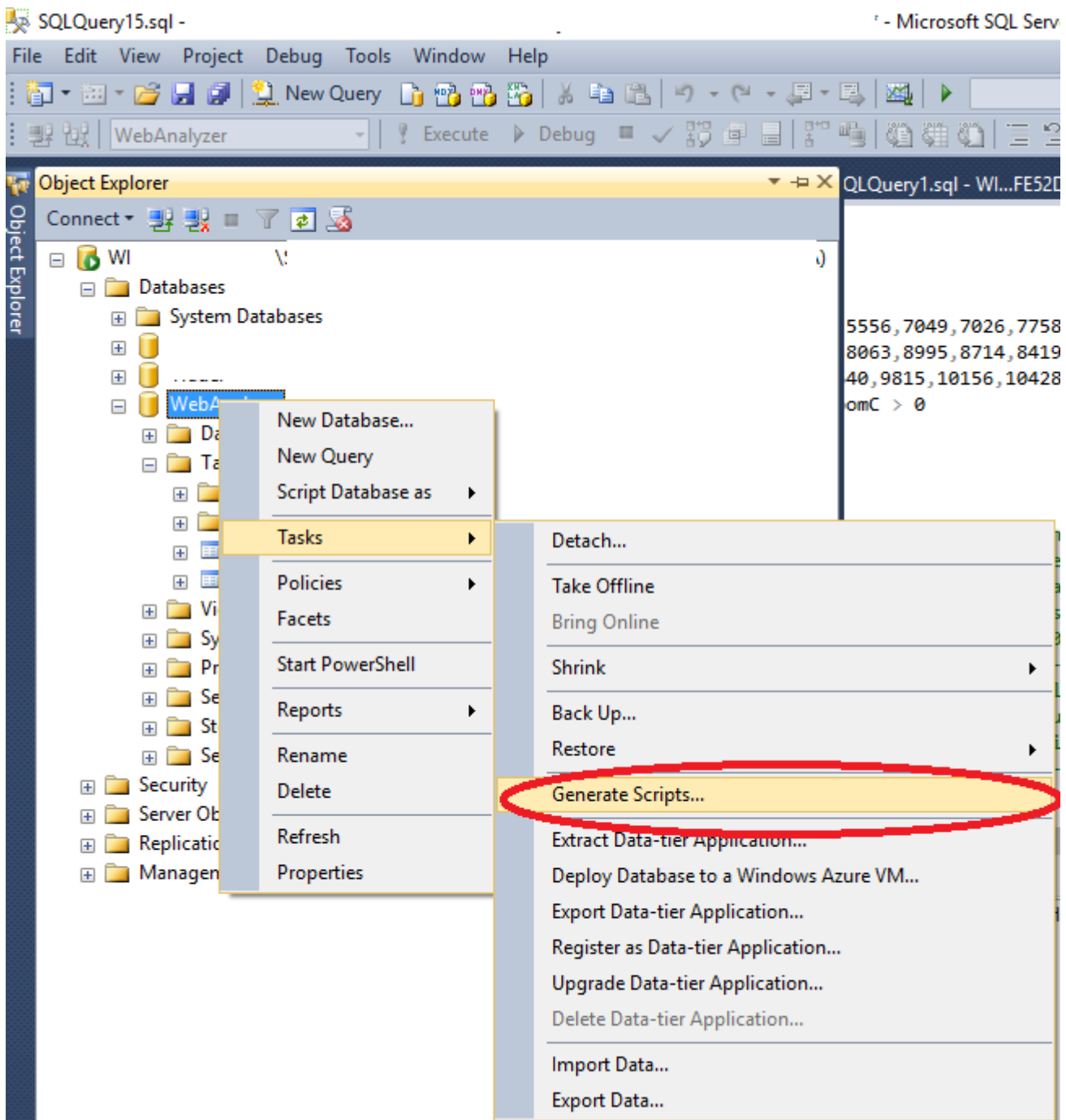
Lea Mientras bucle en línea: <https://riptutorial.com/es/sql-server/topic/4249/mientras-bucle>

Capítulo 62: Migración

Examples

Cómo generar scripts de migración.

1. Haga clic con el botón derecho del ratón en la base de datos que desea migrar y luego -> Tasks -> Generate Scripts...



2. El asistente abrirá, haga clic en `Next` luego seleccione los objetos que desea migrar y haga clic en `Next` nuevamente, luego haga clic en `Advanced` desplácese un poco hacia abajo y en `Types of data to script` elija `Schema and data` (a menos que solo desee estructuras)



Set Scripting Options

Introduction

Choose Objects

Set Scripting Options

Summary

Save or Publish Scripts

Help

Specify how scripts should be saved or published.

Output Type

- Save scripts to a specific location
- Publish to Web service

Save to file

- Files to generate:
- Single file
 - Single file per object

File name: C:\Users\NL

Overwrite

Save as:

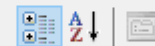
- Unicode text
- ANSI text

- Save to Clipboard
- Save to new query window

Advanced

Advanced Scripting Options

Options



Script Object-Level Permissions	False
Script Owner	False
Script Statistics	Do not script statistics
Script USE DATABASE	True
Types of data to script	Schema and data
Table/View Options	Data only
Script Change Tracking	Schema and data
Script Check Constraints	Schema only
Script Data Compression Options	False
Script Foreign Keys	True
Script Full-Text Indexes	False
Script Indexes	True
Script Primary Keys	True

Types of data to script

Generates script that contains schema only or schema and data

3. Haga clic un par de veces más en **Next** y **Finish** y debe tener su base de datos en un archivo `.sql`.

4. ejecute el archivo `.sql` en su nuevo servidor, y debería estar listo.

Lea Migración en línea: <https://riptutorial.com/es/sql-server/topic/4451/migracion>

Capítulo 63: Modificar texto JSON

Examples

Modificar valor en texto JSON en la ruta especificada

La función `JSON_MODIFY` utiliza el texto JSON como parámetro de entrada y modifica un valor en la ruta especificada utilizando el tercer argumento:

```
declare @json nvarchar(4000) = N'{"Id":1,"Name":"Toy Car","Price":34.99}'
set @json = JSON_MODIFY(@json, '$.Price', 39.99)
print @json -- Output: {"Id":1,"Name":"Toy Car","Price":39.99}
```

Como resultado, tendremos un nuevo texto JSON con "Precio": 39.99 y el otro valor no se cambiará. Si el objeto en la ruta especificada no existe, `JSON_MODIFY` insertará el par clave: valor.

Para eliminar el par clave: valor, ponga `NULL` como nuevo valor:

```
declare @json nvarchar(4000) = N'{"Id":1,"Name":"Toy Car","Price":34.99}'
set @json = JSON_MODIFY(@json, '$.Price', NULL)
print @json -- Output: {"Id":1,"Name":"Toy Car"}
```

`JSON_MODIFY` eliminará la clave por defecto si no tiene valor, por lo que puede usarla para eliminar una clave.

Agregar un valor escalar en una matriz JSON

`JSON_MODIFY` tiene el modo 'agregar' que agrega valor a la matriz.

```
declare @json nvarchar(4000) = N'{"Id":1,"Name":"Toy Car","Tags":["toy","game"]}'
set @json = JSON_MODIFY(@json, 'append $.Tags', 'sales')
print @json -- Output: {"Id":1,"Name":"Toy Car","Tags":["toy","game","sales"]}
```

Si la matriz en la ruta especificada no existe, `JSON_MODIFY` (anexar) creará una nueva matriz con un solo elemento:

```
declare @json nvarchar(4000) = N'{"Id":1,"Name":"Toy Car","Price":34.99}'
set @json = JSON_MODIFY(@json, 'append $.Tags', 'sales')
print @json -- Output {"Id":1,"Name":"Toy Car","Tags":["sales"]}
```

Insertar nuevo objeto JSON en texto JSON

La función `JSON_MODIFY` le permite insertar objetos JSON en el texto JSON:

```
declare @json nvarchar(4000) = N'{"Id":1,"Name":"Toy Car"}'
set @json = JSON_MODIFY(@json, '$.Price',
```

```

                                JSON_QUERY ('{"Min":34.99,"Recommended":45.49}'))
print @json
-- Output: {"Id":1,"Name":"Toy Car","Price":{"Min":34.99,"Recommended":45.49}}

```

Como el tercer parámetro es texto, debe envolverlo con la función `JSON_QUERY` para "convertir" el texto a JSON. Sin este "cast", `JSON_MODIFY` tratará el tercer parámetro como texto sin formato y caracteres de escape antes de insertarlo como valor de cadena. Sin `JSON_QUERY` los resultados serán:

```

{"Id":1,"Name":"Toy Car","Price":'{"Min":34.99,"Recommended":45.49}'}

```

`JSON_MODIFY` insertará este objeto si no existe, o lo eliminará si el valor del tercer parámetro es `NULL`.

Insertar nueva matriz JSON generada con la consulta FOR JSON

Puede generar el objeto JSON utilizando la consulta `SELECT` estándar con la cláusula `FOR JSON` e insertarlo en el texto JSON como tercer parámetro:

```

declare @json nvarchar(4000) = N'{"Id":17,"Name":"WWI"}'
set @json = JSON_MODIFY(@json, '$.tables',
                        (select name from sys.tables FOR JSON PATH) )
print @json

(1 row(s) affected)
{"Id":17,"Name":"WWI","tables":[{"name":"Colors"}, {"name":"Colors_Archive"}, {"name":"OrderLines"}, {"name":"..."}]}

```

`JSON_MODIFY` sabrá que la consulta de selección con la cláusula `FOR JSON` genera una matriz JSON válida y solo la insertará en el texto JSON.

Puede usar todas las opciones de `FOR JSON` en la consulta `SELECT`, **excepto `WITHOUT_ARRAY_WRAPPER`**, que generará un solo objeto en lugar de la matriz JSON. Vea otro ejemplo en este tema para ver cómo insertar un solo objeto JSON.

Insertar un solo objeto JSON generado con la cláusula FOR JSON

Puede generar el objeto JSON utilizando la consulta `SELECT` estándar con la cláusula `FOR JSON` y la opción `WITHOUT_ARRAY_WRAPPER`, e insertarlo en el texto JSON como tercer parámetro:

```

declare @json nvarchar(4000) = N'{"Id":17,"Name":"WWI"}'
set @json = JSON_MODIFY(@json, '$.table',
                        JSON_QUERY(
                            (select name, create_date, schema_id
                             from sys.tables
                             where name = 'Colors'
                             FOR JSON PATH, WITHOUT_ARRAY_WRAPPER)))
print @json

(1 row(s) affected)
{"Id":17,"Name":"WWI","table":{"name":"Colors","create_date":"2008-08-04 10:00:00","schema_id":1}}

```

```
{"Id":17,"Name":"WWI","table":{"name":"Colors","create_date":"2016-06-02T10:04:03.280","schema_id":13}}
```

La opción **FOR JSON with WITHOUT_ARRAY_WRAPPER** puede generar texto JSON no válido si la consulta **SELECT** devuelve más de un resultado (en este caso, debe usar **TOP 1** o filtrar por clave principal). Por lo tanto, **JSON_MODIFY** asumirá que el resultado devuelto es solo un texto sin formato y lo eliminará como cualquier otro texto si no lo ajusta con la función **JSON_QUERY**.

Debe ajustar la consulta **FOR JSON, WITHOUT_ARRAY_WRAPPER** con la función **JSON_QUERY** para convertir el resultado a JSON.

Lea **Modificar texto JSON en línea**: <https://riptutorial.com/es/sql-server/topic/6883/modificar-texto-json>

Capítulo 64: Módulos compilados de forma nativa (Hekaton)

Examples

Procedimiento almacenado nativamente compilado

En un procedimiento con compilación nativa, el código T-SQL se compila a dll y se ejecuta como código C nativo. Para crear un procedimiento almacenado nativo compilado, necesita:

- Utilice la sintaxis estándar de **CREAR PROCEDIMIENTO**
- Establecer la opción **NATIVE_COMPILATION** en la definición de procedimiento almacenado
- Utilice la opción **SCHEMABINDING** en la definición de procedimiento almacenado
- Defina la opción **EJECUTAR COMO PROPIETARIO** en la definición de procedimiento almacenado

En lugar del bloque **BEGIN END** estándar, necesita usar el bloque **BEGIN ATOMIC**:

```
BEGIN ATOMIC
    WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='us_english')
    -- T-Sql code goes here
END
```

Ejemplo:

```
CREATE PROCEDURE usp_LoadMemOptTable (@maxRows INT, @FullName NVARCHAR(200))
WITH
    NATIVE_COMPILATION,
    SCHEMABINDING,
    EXECUTE AS OWNER
AS
BEGIN ATOMIC
WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='us_english')
    DECLARE @i INT = 1
    WHILE @i <= @maxRows
    BEGIN
        INSERT INTO dbo.MemOptTable3 VALUES(@i, @FullName, GETDATE())
        SET @i = @i+1
    END
END
GO
```

Función escalar compilada de forma nativa

El código en la función compilada de forma nativa se transformará en código C y se compilará como dll. Para crear una función escalar compilada nativa necesitas:

- Utilice la sintaxis estándar de **CREATE FUNCTION**

- Establecer la opción `NATIVE_COMPILATION` en la definición de la función
- Utilice la opción `SCHEMABINDING` en la definición de la función

En lugar del bloque `BEGIN END` estándar, necesita usar el bloque `BEGIN ATOMIC`:

```
BEGIN ATOMIC
  WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='us_english')
  -- T-Sql code goes here
END
```

Ejemplo:

```
CREATE FUNCTION [dbo].[udfMultiply]( @v1 int, @v2 int )
RETURNS bigint
WITH NATIVE_COMPILATION, SCHEMABINDING
AS
BEGIN ATOMIC WITH (TRANSACTION ISOLATION LEVEL = SNAPSHOT, LANGUAGE = N'English')

  DECLARE @ReturnValue bigint;
  SET @ReturnValue = @v1 * @v2;

  RETURN (@ReturnValue);
END

-- usage sample:
SELECT dbo.udfMultiply(10, 12)
```

Función de valor de tabla en línea nativa

La función de valor de tabla compilada nativa devuelve tabla como resultado. El código en la función compilada de forma nativa se transformará en código C y se compilará como dll. Solo las funciones con valor de tabla en línea son compatibles con la versión 2016. Para crear una función de valor de tabla nativa, necesita:

- Utilice la sintaxis estándar de `CREATE FUNCTION`
- Establecer la opción `NATIVE_COMPILATION` en la definición de la función
- Utilice la opción `SCHEMABINDING` en la definición de la función

En lugar del bloque `BEGIN END` estándar, necesita usar el bloque `BEGIN ATOMIC`:

```
BEGIN ATOMIC
  WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='us_english')
  -- T-Sql code goes here
END
```

Ejemplo:

```
CREATE FUNCTION [dbo].[udft_NativeGetBusinessDoc]
(
  @RunDate VARCHAR(25)
)
RETURNS TABLE
WITH SCHEMABINDING,
```

```
NATIVE_COMPILATION
AS
RETURN
(
SELECT BusinessDocNo,
        ProductCode,
        UnitID,
        ReasonID,
        PriceID,
        RunDate,
        ReturnPercent,
        Qty,
        RewardAmount,
        ModifyDate,
        UserID
FROM dbo.[BusinessDocDetail_11]
WHERE RunDate >= @RunDate
);
```

Lea Módulos compilados de forma nativa (Hekaton) en línea: <https://riptutorial.com/es/sql-server/topic/6089/modulos-compilados-de-forma-nativa--hekaton->

Capítulo 65: Mueve y copia datos alrededor de tablas

Examples

Copia datos de una tabla a otra

Este código selecciona los datos de una tabla y los muestra en la herramienta de consulta (generalmente SSMS)

```
SELECT Column1, Column2, Column3 FROM MySourceTable;
```

Este código inserta esos datos en una tabla:

```
INSERT INTO MyTargetTable (Column1, Column2, Column3)
SELECT Column1, Column2, Column3 FROM MySourceTable;
```

Copia datos en una tabla, creando esa tabla sobre la marcha

Este código selecciona los datos de una tabla:

```
SELECT Column1, Column2, Column3 FROM MySourceTable;
```

Este código crea una nueva tabla llamada `MyNewTable` y coloca esos datos en ella.

```
SELECT Column1, Column2, Column3
INTO MyNewTable
FROM MySourceTable;
```

Mover datos a una tabla (asumiendo el método de claves únicas)

Para *mover* los datos, primero insértelos en el destino y luego elimine lo que haya insertado de la tabla de origen. Esta no es una operación SQL normal pero puede ser esclarecedor

¿Qué insertaste? Normalmente, en las bases de datos debe tener una o más columnas que pueda usar para identificar de forma única las filas, por lo que supondremos eso y las utilizaremos.

Esta declaración selecciona algunas filas

```
SELECT Key1, Key2, Column3, Column4 FROM MyTable;
```

Primero insertamos estos en nuestra tabla de objetivos:

```
INSERT INTO TargetTable (Key1, Key2, Column3, Column4)
SELECT Key1, Key2, Column3, Column4 FROM MyTable;
```

Ahora, *asumiendo que los registros en ambas tablas son únicos* en `Key1` , `Key2` , podemos usar eso para encontrar y eliminar datos de la tabla de origen

```
DELETE MyTable
WHERE EXISTS (
    SELECT * FROM TargetTable
    WHERE TargetTable.Key1 = SourceTable.Key1
    AND TargetTable.Key2 = SourceTable.Key2
);
```

Esto solo funcionará correctamente si `Key1` , `Key2` son únicas en ambas tablas

Por último, no queremos que el trabajo esté hecho a medias. Si envolvemos esto en una transacción, todos los datos se moverán o no sucederá nada. Esto garantiza que no insertemos los datos y luego no podamos eliminarlos de la fuente.

```
BEGIN TRAN;

INSERT INTO TargetTable (Key1, Key2, Column3, Column4)
SELECT Key1, Key2, Column3, Column4 FROM MyTable;

DELETE MyTable
WHERE EXISTS (
    SELECT * FROM TargetTable
    WHERE TargetTable.Key1 = SourceTable.Key1
    AND TargetTable.Key2 = SourceTable.Key2
);

COMMIT TRAN;
```

Lea **Mueve y copia datos alrededor de tablas en línea**: <https://riptutorial.com/es/sql-server/topic/1467/mueve-y-copia-datos-alrededor-de-tablas>

Capítulo 66: Niveles de aislamiento de transacciones.

Sintaxis

- CONFIGURAR EL NIVEL DE AISLAMIENTO DE TRANSACCIONES {LEER SIN COMPROMISO | LEER COMPROMETIDO | READATABLE LEER | SNAPSHOT | SERIALIZABLE} [;]

Observaciones

Referencia de MSDN: [AJUSTE EL NIVEL DE AISLAMIENTO DE TRANSACCIONES](#)

Examples

Leer no comprometido

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
```

Este es el nivel de aislamiento más permisivo, ya que no causa ningún bloqueo. Especifica que las declaraciones pueden leer todas las filas, incluidas las que se han escrito en transacciones pero que aún no se han confirmado (es decir, aún están en transacción). Este nivel de aislamiento puede estar sujeto a "lecturas sucias".

Leer comprometido

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
```

Este nivel de aislamiento es el segundo más permisivo. Previene lecturas sucias. El comportamiento de `READ COMMITTED` depende de la configuración de `READ_COMMITTED_SNAPSHOT` :

- Si se establece en DESACTIVADO (la configuración predeterminada), la transacción utiliza bloqueos compartidos para evitar que otras transacciones modifiquen las filas utilizadas por la transacción actual, así como para bloquear la lectura de las filas modificadas por otras transacciones.
- Si se establece en ACTIVADO, la `READCOMMITTEDLOCK` tabla `READCOMMITTEDLOCK` se puede usar para solicitar un bloqueo compartido en lugar del control de versiones de la fila para transacciones que se ejecutan en el modo `READ COMMITTED` .

Nota: `READ COMMITTED` es el comportamiento predeterminado de SQL Server.

¿Qué son las "lecturas sucias"?

Las lecturas sucias (o lecturas no confirmadas) son lecturas de filas que están siendo modificadas por una transacción abierta.

Este comportamiento se puede replicar utilizando 2 consultas separadas: una para abrir una transacción y escribir algunos datos en una tabla sin confirmar, la otra para seleccionar los datos que se escribirán (pero aún no se han confirmado) con este nivel de aislamiento.

Consulta 1 - Prepare una transacción pero no la termine:

```
CREATE TABLE dbo.demo (  
    col1 INT,  
    col2 VARCHAR(255)  
);  
GO  
--This row will get committed normally:  
BEGIN TRANSACTION;  
    INSERT INTO dbo.demo(col1, col2)  
    VALUES (99, 'Normal transaction');  
COMMIT TRANSACTION;  
--This row will be "stuck" in an open transaction, causing a dirty read  
BEGIN TRANSACTION;  
    INSERT INTO dbo.demo(col1, col2)  
    VALUES (42, 'Dirty read');  
--Do not COMMIT TRANSACTION or ROLLBACK TRANSACTION here
```

Consulta 2 : lea las filas que incluyen la transacción abierta:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;  
SELECT * FROM dbo.demo;
```

Devoluciones:

```
col1      col2  
-----  
99        Normal transaction  
42        Dirty read
```

PD: No te olvides de limpiar estos datos de demostración:

```
COMMIT TRANSACTION;  
DROP TABLE dbo.demo;  
GO
```

Lectura repetible

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
```

Este nivel de aislamiento de la transacción es ligeramente menos permisivo que `READ COMMITTED`, ya que los bloqueos compartidos se colocan en todos los datos leídos por cada declaración en la transacción y se mantienen **hasta que la transacción se completa**, en lugar de liberarse después de cada declaración.

Nota: use esta opción solo cuando sea necesario, ya que es más probable que cause una degradación del rendimiento de la base de datos, así como puntos muertos que `READ COMMITTED`.

Instantánea

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT
```

Especifica que los datos leídos por cualquier declaración en una transacción serán la versión consistente de la transacción de los datos que existían al inicio de la transacción, es decir, solo leerán los datos que se hayan confirmado antes de que comience la transacción.

`SNAPSHOT` transacciones `SNAPSHOT` no solicitan ni causan ningún bloqueo en los datos que se están leyendo, ya que solo está leyendo la versión (o instantánea) de los datos que existían en el momento en que comenzó la transacción.

Una transacción que se ejecuta en el nivel de aislamiento `SNAPSHOT` lee solo sus propios cambios de datos mientras se ejecuta. Por ejemplo, una transacción podría actualizar algunas filas y luego leer las filas actualizadas, pero ese cambio solo será visible para la transacción actual hasta que se confirme.

Nota: la opción de la base de datos `ALLOW_SNAPSHOT_ISOLATION` debe estar activada antes de poder utilizar el nivel de aislamiento `SNAPSHOT`.

Serializable

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
```

Este nivel de aislamiento es el más restrictivo. El **rango de solicitudes bloquea** el rango de valores clave que lee cada declaración en la transacción. Esto también significa que las declaraciones `INSERT` de otras transacciones se bloquearán si las filas que se insertarán están en el rango bloqueado por la transacción actual.

Esta opción tiene el mismo efecto que configurar `HOLDLOCK` en todas las tablas en todas las declaraciones `SELECT` en una transacción.

Nota: Este aislamiento de transacción tiene la concurrencia más baja y solo debe usarse cuando sea necesario.

Lea Niveles de aislamiento de transacciones. en línea: <https://riptutorial.com/es/sql-server/topic/5114/niveles-de-aislamiento-de-transacciones->

Capítulo 67: Niveles de aislamiento y bloqueo.

Observaciones

Encontré este enlace, es útil como referencia: "[Niveles de aislamiento](#)"

Examples

Ejemplos de ajuste del nivel de aislamiento.

Ejemplo de configuración del nivel de aislamiento:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;  
SELECT * FROM Products WHERE ProductId=1;  
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ; --return to the default one
```

1. **READ UNCOMMITTED** : significa que una consulta en la transacción actual no puede acceder a los datos modificados de otra transacción que aún no se ha confirmado, ¡sin lecturas sucias! PERO, las lecturas no repetibles y las lecturas fantasma son posibles, porque los datos todavía pueden ser modificados por otras transacciones.
2. **REPEATABLE READ** : significa que una consulta en la transacción actual no puede acceder a los datos modificados de otra transacción que aún no se ha confirmado, ¡sin lecturas sucias! Ninguna otra transacción puede modificar los datos que lee la transacción actual hasta que se complete, lo que elimina las lecturas NO REPETIBLES. PERO, si otra transacción inserta NEW ROWS y la consulta se ejecuta más de una vez, las filas fantasmas pueden aparecer comenzando la segunda lectura (si coincide con la declaración where de la consulta).
3. **SNAPSHOT** : solo puede devolver datos que existen al principio de la consulta. Asegura la consistencia de los datos. Previene lecturas sucias, lecturas no repetibles y lecturas fantasma. Para usar eso, se requiere configuración de DB:

```
ALTER DATABASE DBTestName SET ALLOW_SNAPSHOT_ISOLATION ON;GO;  
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
```

4. **READ COMMITTED** : aislamiento predeterminado del servidor SQL. Previene la lectura de los datos que son modificados por otra transacción hasta que se confirme. Utiliza el bloqueo compartido y el control de versiones de filas en las tablas, lo que evita las lecturas sucias. Depende de la configuración de la base de datos READ_COMMITTED_SNAPSHOT, si está habilitada, se utiliza el control de versiones de fila. para habilitar - usa esto:

```
ALTER DATABASE DBTestName SET ALLOW_SNAPSHOT_ISOLATION ON;GO;  
SET TRANSACTION ISOLATION LEVEL READ COMMITTED; --return to the default one
```

5. `SERIALIZABLE` : utiliza bloqueos físicos que se adquieren y mantienen hasta el final de la transacción, lo que evita lecturas sucias, lecturas fantasma, lecturas no repetibles. PERO, tiene un impacto en el rendimiento de la Base de Datos, porque las transacciones concurrentes se serializan y se ejecutan una por una.

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ;
```

Lea Niveles de aislamiento y bloqueo. en línea: <https://riptutorial.com/es/sql-server/topic/5331/niveles-de-aislamiento-y-bloqueo->

Capítulo 68: Nombres de alias en el servidor SQL

Introducción

Estas son algunas de las diferentes formas de proporcionar nombres de alias a las columnas en el Servidor SQL.

Examples

Usando AS

Este es un método ANSI SQL que funciona en todos los RDBMS. Enfoque muy utilizado.

```
CREATE TABLE AliasNameDemo (id INT,firstname VARCHAR(20),lastname VARCHAR(20))

INSERT INTO AliasNameDemo
VALUES      (1,'MyFirstName','MyLastName')

SELECT FirstName + ' ' + LastName As FullName
FROM      AliasNameDemo
```

Usando =

Este es mi enfoque preferido. Nada relacionado con el rendimiento solo una elección personal. Hace que el código se vea limpio. Puede ver los nombres de las columnas resultantes fácilmente en lugar de desplazarse por el código si tiene una expresión grande.

```
CREATE TABLE AliasNameDemo (id INT,firstname VARCHAR(20),lastname VARCHAR(20))

INSERT INTO AliasNameDemo
VALUES      (1,'MyFirstName','MyLastName')

SELECT FullName = FirstName + ' ' + LastName
FROM      AliasNameDemo
```

Dar alias después del nombre de la tabla Derivada

Este es un enfoque extraño que la mayoría de las personas ni siquiera saben que existe.

```
CREATE TABLE AliasNameDemo(id INT,firstname VARCHAR(20),lastname VARCHAR(20))

INSERT INTO AliasNameDemo
VALUES      (1,'MyFirstName','MyLastName')

SELECT *
FROM      (SELECT firstname + ' ' + lastname
```

```
FROM AliasNameDemo) a (fullname)
```

- **Manifestación**

Sin usar AS

Esta sintaxis será similar a usar la palabra clave `AS` . Simplemente no tenemos que usar la palabra clave `AS`

```
CREATE TABLE AliasNameDemo (id INT,firstname VARCHAR(20),lastname VARCHAR(20))

INSERT INTO AliasNameDemo
VALUES      (1,'MyFirstName','MyLastName')

SELECT FirstName +' '+ LastName FullName
FROM      AliasNameDemo
```

Lea Nombres de alias en el servidor SQL en línea: <https://riptutorial.com/es/sql-server/topic/10784/nombres-de-alias-en-el-servidor-sql>

Capítulo 69: NULLs

Introducción

En SQL Server, `NULL` representa datos que faltan o son desconocidos. Esto significa que `NULL` no es realmente un valor; Se describe mejor como un marcador de posición para un valor. Esta es también la razón por la que no puede comparar `NULL` con ningún valor, y ni siquiera con otro `NULL`.

Observaciones

SQL Server proporciona otros métodos para manejar nulos, como `IS NULL`, `IS NOT NULL`, `ISNULL()`, `COALESCE()` y otros.

Examples

Comparación nula

`NULL` es un caso especial cuando se trata de comparaciones.

Supongamos los siguientes datos.

```
id someVal
----
0 NULL
1 1
2 2
```

Con una consulta:

```
SELECT id
FROM table
WHERE someVal = 1
```

volvería id 1

```
SELECT id
FROM table
WHERE someVal <> 1
```

volvería id 2

```
SELECT id
FROM table
WHERE someVal IS NULL
```

devolvería id 0

```
SELECT id
FROM table
WHERE someVal IS NOT NULL
```

devolvería ambos identificadores 1 y 2 .

Si desea que los NULL se "cuenten" como valores en una comparación = , <> , primero se deben convertir a un tipo de datos contables:

```
SELECT id
FROM table
WHERE ISNULL(someVal, -1) <> 1
```

O

```
SELECT id
FROM table
WHERE someVal IS NULL OR someVal <> 1
```

devuelve 0 y 2

O puede cambiar la configuración de [ANSI Null](#) .

ANSI NULLS

Desde [MSDN](#)

En una versión futura de SQL Server, ANSI_NULLS siempre estará ENCENDIDO y cualquier aplicación que establezca explícitamente la opción en APAGADO generará un error. Evite usar esta función en nuevos trabajos de desarrollo y planee modificar las aplicaciones que actualmente usan esta función.

ANSI NULLS en off permite una comparación = / <> de valores nulos.

Teniendo en cuenta los siguientes datos:

```
id someVal
----
0 NULL
1 1
2 2
```

Y con ANSI NULLS en, esta consulta:

```
SELECT id
FROM table
WHERE someVal = NULL
```

no produciría resultados Sin embargo, la misma consulta, con ANSI NULLS desactivado:

```
set ansi_nulls off

SELECT id
FROM table
WHERE someVal = NULL
```

Volvería id 0 .

ES NULO()

La función `IsNull()` acepta dos parámetros y devuelve el segundo parámetro si el primero es `null` .

Parámetros:

1. comprobar expresión Cualquier expresión de cualquier tipo de datos.
2. Valor de reposición. Este es el valor que se devolvería si la expresión de verificación es nula. El valor de reemplazo debe ser de un tipo de datos que pueda convertirse implícitamente al tipo de datos de la expresión de verificación.

La función `IsNull()` devuelve el mismo tipo de datos que la expresión de verificación.

```
DECLARE @MyInt int -- All variables are null until they are set with values.

SELECT ISNULL(@MyInt, 3) -- Returns 3.
```

Ver también `COALESCE` , arriba.

Es nulo / no es nulo

Dado que nulo no es un valor, no puede usar operadores de comparación con nulos. Para verificar si una columna o variable tiene un valor nulo, debe usar `is null` :

```
DECLARE @Date date = '2016-08-03'
```

La siguiente declaración seleccionará el valor 6 , ya que todas las comparaciones con valores nulos se evalúan como falsas o desconocidas:

```
SELECT CASE WHEN @Date = NULL THEN 1
            WHEN @Date <> NULL THEN 2
            WHEN @Date > NULL THEN 3
            WHEN @Date < NULL THEN 4
            WHEN @Date IS NULL THEN 5
            WHEN @Date IS NOT NULL THEN 6
```

Al establecer el contenido de la variable `@Date` en `null` y volver a intentarlo, la siguiente declaración devolverá 5 :

```
SET @Date = NULL -- Note that the '=' here is an assignment operator!
```

```

SELECT CASE WHEN @Date = NULL THEN 1
           WHEN @Date <> NULL THEN 2
           WHEN @Date > NULL THEN 3
           WHEN @Date < NULL THEN 4
           WHEN @Date IS NULL THEN 5
           WHEN @Date IS NOT NULL THEN 6

```

COALESCE ()

COALESCE () Evalúa los argumentos en orden y devuelve el valor actual de la primera expresión que inicialmente no se evalúa como NULL .

```

DECLARE @MyInt int -- variable is null until it is set with value.
DECLARE @MyInt2 int -- variable is null until it is set with value.
DECLARE @MyInt3 int -- variable is null until it is set with value.

SET @MyInt3 = 3

SELECT COALESCE (@MyInt, @MyInt2 ,@MyInt3 ,5) -- Returns 3 : value of @MyInt3.

```

Aunque ISNULL () funciona de manera similar a COALESCE (), la función ISNULL () solo acepta dos parámetros: uno para verificar y otro para usar si el primer parámetro es NULL. Ver también ISNULL , abajo.

NULL con NOT IN SubQuery

Si bien el manejo no se realiza en subconsulta con nulo en la subconsulta, debemos eliminar NULLS para obtener los resultados esperados.

```

create table #outertable (i int)
create table #innertable (i int)

insert into #outertable (i) values (1), (2),(3),(4), (5)
insert into #innertable (i) values (2), (3), (null)

select * from #outertable where i in (select i from #innertable)
--2
--3
--So far so good

select * from #outertable where i not in (select i from #innertable)
--Expectation here is to get 1,4,5 but it is not. It will get empty results because of the
NULL it executes as {select * from #outertable where i not in (null)}

--To fix this
select * from #outertable where i not in (select i from #innertable where i is not null)
--you will get expected results
--1
--4
--5

```

Si bien el manejo no está en sub-consulta con nulo, tenga cuidado con el resultado esperado.

Lea NULLs en línea: <https://riptutorial.com/es/sql-server/topic/5044/nulls>

Capítulo 70: Opciones avanzadas

Examples

Habilitar y mostrar opciones avanzadas

```
Exec sp_configure 'show advanced options' ,1
RECONFIGURE
GO
-- Show all configure
sp_configure
```

Habilitar compresión de respaldo por defecto

```
Exec sp_configure 'backup compression default',1
GO
RECONFIGURE;
```

Establecer el porcentaje de relleno predeterminado

```
sp_configure 'fill factor', 100;
GO
RECONFIGURE;
```

El servidor debe reiniciarse antes de que el cambio surta efecto.

Establecer intervalo de recuperación del sistema

```
USE master;
GO
-- Set recovery every 3 min
EXEC sp_configure 'recovery interval', '3';
RECONFIGURE WITH OVERRIDE;
```

Habilitar permiso cmd

```
EXEC sp_configure 'xp_cmdshell', 1
GO
RECONFIGURE
```

Establecer el tamaño máximo de memoria del servidor

```
USE master
EXEC sp_configure 'max server memory (MB)', 64
RECONFIGURE WITH OVERRIDE
```

Establecer el número de tareas de punto de control

```
EXEC sp_configure "number of checkpoint tasks", 4
```

Lea Opciones avanzadas en línea: <https://riptutorial.com/es/sql-server/topic/5185/opciones-avanzadas>

Capítulo 71: OPENJSON

Examples

Obtener clave: pares de valores de texto JSON

La función OPENJSON analiza el texto JSON y devuelve todos los pares clave: valor en el primer nivel de JSON:

```
declare @json NVARCHAR(4000) = N'{"Name":"Joe","age":27,"skills":["C#","SQL"]}';
SELECT * FROM OPENJSON(@json);
```

llave	valor	tipo
Nombre	Joe	1
años	27	2
habilidades	["C #", "SQL"]	4

El tipo de columna describe el tipo de valor, es decir, nulo (0), cadena (1), número (2), booleano (3), matriz (4) y objeto (5).

Transformar la matriz JSON en un conjunto de filas

La función OPENJSON analiza la colección de objetos JSON y devuelve valores del texto JSON como un conjunto de filas.

```
declare @json nvarchar(4000) = N'[
  {"Number":"SO43659","Date":"2011-05-31T00:00:00","Customer":
"MSFT","Price":59.99,"Quantity":1},
  {"Number":"SO43661","Date":"2011-06-
01T00:00:00","Customer":"Nokia","Price":24.99,"Quantity":3}
]'
```

```
SELECT      *
FROM OPENJSON (@json)
  WITH (
    Number    varchar(200),
    Date      datetime,
    Customer  varchar(200),
    Quantity  int
  )
```

En la cláusula WITH se especifica el esquema de retorno de la función OPENJSON. Las claves en los objetos JSON se recuperan por nombres de columna. Si alguna clave en JSON no se especifica en la cláusula WITH (por ejemplo, Precio en este ejemplo) se ignorará. Los valores se convierten automáticamente en tipos especificados.

Número	Fecha	Cliente	Cantidad
SO43659	2011-05-31T00:00:00	MSFT	1
SO43661	2011-06-01T00:00:00	Nokia	3

Transformar los campos JSON anidados en un conjunto de filas

La función OPENJSON analiza la colección de objetos JSON y devuelve valores del texto JSON como un conjunto de filas. Si los valores en el objeto de entrada están anidados, se puede especificar un parámetro de mapeo adicional en cada columna en la cláusula WITH:

```
declare @json nvarchar(4000) = N'[
  {"data":{"num":"SO43659","date":"2011-05-31T00:00:00"},"info":{"customer":"MSFT","Price":59.99,"qty":1}},
  {"data":{"number":"SO43661","date":"2011-06-01T00:00:00"},"info":{"customer":"Nokia","Price":24.99,"qty":3}}
]'
```

```
SELECT      *
FROM OPENJSON(@json)
  WITH (
    Number    varchar(200) '$.data.num',
    Date      datetime '$.data.date',
    Customer  varchar(200) '$.info.customer',
    Quantity  int '$.info.qty',
  )
```

En la cláusula WITH se especifica el esquema de retorno de la función OPENJSON. Después de que el tipo se especifique, la ruta a los nodos JSON donde se debe encontrar el valor devuelto. Las claves en los objetos JSON son recuperadas por estas rutas. Los valores se convierten automáticamente en tipos especificados.

Número	Fecha	Cliente	Cantidad
SO43659	2011-05-31T00:00:00	MSFT	1
SO43661	2011-06-01T00:00:00	Nokia	3

Extracción de subobjetos JSON internos

OPENJSON puede extraer fragmentos de objetos JSON dentro del texto JSON. En la definición de columna que hace referencia al subobjeto JSON, establezca el tipo nvarchar (max) y la opción AS JSON:

```
declare @json nvarchar(4000) = N'[
  {"Number":"SO43659","Date":"2011-05-31T00:00:00","info":{"customer":"MSFT","Price":59.99,"qty":1}},
  {"Number":"SO43661","Date":"2011-06-01T00:00:00","info":{"customer":"Nokia","Price":24.99,"qty":3}}
]'
```



```

SELECT      *
FROM OPENJSON (@json)
  WITH (
    Number   varchar(200),
    Date     datetime,
    Info     nvarchar(max) '$.info' AS JSON
  )

```

La columna de información se asignará al objeto "Info". Los resultados serán:

Número	Fecha	Información
SO43659	2011-05-31T00:00:00	{"cliente": "MSFT", "Precio": 59.99, "cantidad": 1}
SO43661	2011-06-01T00:00:00	{"cliente": "Nokia", "Precio": 24.99, "cantidad": 3}

Trabajar con subarreglas JSON anidadas

JSON puede tener una estructura compleja con matrices internas. En este ejemplo, tenemos una matriz de órdenes con una matriz matriz anidada de artículos de pedido.

```

declare @json nvarchar(4000) = N'[
  {"Number":"SO43659","Date":"2011-05-31T00:00:00",
    "Items":[{"Price":11.99,"Quantity":1}, {"Price":12.99,"Quantity":5}]},
  {"Number":"SO43661","Date":"2011-06-01T00:00:00",
    "Items":[{"Price":21.99,"Quantity":3}, {"Price":22.99,"Quantity":2}, {"Price":23.99,"Quantity":2}]}
]'

```

Podemos analizar las propiedades de nivel de raíz utilizando OPENJSON que devolverá el arreglo de elementos según el fragmento JSON. Luego podemos aplicar OPENJSON nuevamente en la matriz de elementos y abrir la tabla JSON interna. La tabla de primer nivel y la tabla interna se "unirán" como en UNIR entre tablas estándar:

```

SELECT      *
FROM
  OPENJSON (@json)
  WITH (
    Number varchar(200), Date datetime,
    Items nvarchar(max) AS JSON )
  CROSS APPLY
    OPENJSON (Items)
  WITH (
    Price float, Quantity int)

```

Resultados:

Número	Fecha	Artículos	Precio	Cantidad
SO43659	2011-05-31 00:00: 00.000	[{"Precio": 11.99, "Cantidad": 1}, {"Precio": 12.99, "Cantidad": 5}]	11.99	1

Número	Fecha	Artículos	Precio	Cantidad
SO43659	2011-05-31 00: 00: 00.000	[{"Precio": 11.99, "Cantidad": 1}, {"Precio": 12.99, "Cantidad": 5}]	12.99	5
SO43661	2011-06-01 00: 00: 00.000	[{"Precio": 21.99, "Cantidad": 3}, {"Precio": 22.99, "Cantidad": 2}, {"Precio": 23.99, "Cantidad": 2}]	21.99	3
SO43661	2011-06-01 00: 00: 00.000	[{"Precio": 21.99, "Cantidad": 3}, {"Precio": 22.99, "Cantidad": 2}, {"Precio": 23.99, "Cantidad": 2}]	22.99	2
SO43661	2011-06-01 00: 00: 00.000	[{"Precio": 21.99, "Cantidad": 3}, {"Precio": 22.99, "Cantidad": 2}, {"Precio": 23.99, "Cantidad": 2}]	23.99	2

Lea OPENJSON en línea: <https://riptutorial.com/es/sql-server/topic/5030/openjson>

Capítulo 72: Operaciones básicas de DDL en MS SQL Server

Examples

Empezando

Esta sección describe algunos comandos básicos de **DDL** (= " **D** ata **D** efinition **L** anguage") para crear una base de datos, una tabla dentro de una base de datos, una vista y finalmente un procedimiento almacenado.

Crear base de datos

El siguiente comando SQL crea una nueva base de datos `Northwind` en el servidor actual, utilizando la ruta `C:\Program Files\Microsoft SQL Server\MSSQL11.INSTSQL2012\MSSQL\DATA\ :`

```
USE [master]
GO

CREATE DATABASE [Northwind]
  CONTAINMENT = NONE
  ON PRIMARY
  (
    NAME = N'Northwind',
    FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL11.INSTSQL2012\MSSQL\DATA\Northwind.mdf' , SIZE = 5120KB , MAXSIZE = UNLIMITED,
    FILEGROWTH = 1024KB
  )
  LOG ON
  (
    NAME = N'Northwind_log',
    FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL11.INSTSQL2012\MSSQL\DATA\Northwind_log.ldf' , SIZE = 1536KB , MAXSIZE = 2048GB ,
    FILEGROWTH = 10%
  )
GO

ALTER DATABASE [Northwind] SET COMPATIBILITY_LEVEL = 110
GO
```

Nota: una base de datos T-SQL consta de dos archivos, el archivo de base de datos `*.mdf` y su registro de transacciones `*.ldf` . Ambos deben especificarse cuando se crea una nueva base de datos.

Crear mesa

El siguiente comando SQL crea una nueva tabla `Categories` en la base de datos actual, usando el esquema `dbo` (puede cambiar el contexto de la base de datos con `Use <DatabaseName>`):

```
CREATE TABLE dbo.Categories (
    CategoryID int IDENTITY NOT NULL,
    CategoryName nvarchar(15) NOT NULL,
    Description ntext NULL,
    Picture image NULL,
    CONSTRAINT PK_Categories PRIMARY KEY CLUSTERED
    (
        CategoryID ASC
    )
    WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
        ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON PRIMARY
) ON PRIMARY TEXTIMAGE_ON PRIMARY
```

Crear vista

El siguiente comando SQL crea una nueva vista `Summary_of_Sales_by_Year` en la base de datos actual, usando el esquema `dbo` (puede cambiar el contexto de la base de datos con `Use <DatabaseName>`):

```
CREATE VIEW dbo.Summary_of_Sales_by_Year AS
    SELECT ord.ShippedDate, ord.OrderID, ordSub.Subtotal
    FROM Orders ord
    INNER JOIN [Order Subtotals] ordSub ON ord.OrderID = ordSub.OrderID
```

Esto unirá las tablas `Orders` y `[Order Subtotals]` para mostrar las columnas `ShippedDate`, `OrderID` y `Subtotal`. Debido a que la tabla `[Order Subtotals]` tiene un espacio en blanco en su nombre en la base de datos de Northwind, debe estar entre corchetes.

Crear procedimiento

El siguiente comando SQL crea un nuevo procedimiento almacenado `CustOrdersDetail` en la base de datos actual, usando el esquema `dbo` (puede cambiar el contexto de la base de datos con `Use <DatabaseName>`):

```
CREATE PROCEDURE dbo.MyCustOrdersDetail @OrderID int, @MinQuantity int=0
AS BEGIN
    SELECT ProductName,
        UnitPrice=ROUND(Od.UnitPrice, 2),
        Quantity,
        Discount=CONVERT(int, Discount * 100),
        ExtendedPrice=ROUND(CONVERT(money, Quantity * (1 - Discount) * Od.UnitPrice), 2)
    FROM Products P, [Order Details] Od
    WHERE Od.ProductID = P.ProductID and Od.OrderID = @OrderID
    and Od.Quantity>=@MinQuantity
END
```

Este procedimiento almacenado, una vez creado, se puede invocar de la siguiente manera:

```
exec dbo.MyCustOrdersDetail 10248
```

que devolverá todos los detalles del pedido con @ OrderId = 10248 (y la cantidad > = 0 como predeterminado). O puede especificar el parámetro opcional

```
exec dbo.MyCustOrdersDetail 10248, 10
```

que devolverá solo pedidos con una cantidad mínima de 10 (o más).

Lea Operaciones básicas de DDL en MS SQL Server en línea: <https://riptutorial.com/es/sql-server/topic/5463/operaciones-basicas-de-ddl-en-ms-sql-server>

Capítulo 73: ORDEN POR

Observaciones

El propósito de la cláusula ORDER BY es ordenar los datos devueltos por una consulta.

Es importante tener en cuenta que el **orden de las filas devueltas por una consulta no está definido a menos que haya una cláusula ORDER BY.**

Consulte la documentación de MSDN para obtener todos los detalles de la cláusula ORDER BY: <https://msdn.microsoft.com/en-us/library/ms188385.aspx>

Examples

Cláusula ORDER BY simple

Usando la [tabla de empleados](#) , a continuación se muestra un ejemplo para devolver las columnas Id, FName y LName en orden LName (ascendente):

```
SELECT Id, FName, LName FROM Employees
ORDER BY LName
```

Devoluciones:

Carné de identidad	FName	LName
2	Juan	Johnson
1	James	Herrero
4	Johnathon	Herrero
3	Miguel	Williams

Para ordenar en orden descendente, agregue la palabra clave DESC después del parámetro de campo, por ejemplo, la misma consulta en orden descendente de LName es:

```
SELECT Id, FName, LName FROM Employees
ORDER BY LName DESC
```

ORDENAR por campos múltiples

Se pueden especificar varios campos para la cláusula ORDER BY , en orden ASCending o DESCending.

Por ejemplo, al usar la tabla <http://stackoverflow.com/documentation/sql/280/example->

[databases/1207/item-sales-table#t=201607211314066434211](#) , podemos devolver una consulta que se ordena por SaleDate en orden ascendente y Cantidad en orden descendente.

```
SELECT ItemId, SaleDate, Quantity
FROM [Item Sales]
ORDER BY SaleDate ASC, Quantity DESC
```

Tenga en cuenta que la palabra clave `ASC` es opcional, y los resultados se clasifican en orden ascendente de un campo determinado de forma predeterminada.

ORDENAR con lógica compleja

Si queremos ordenar los datos de manera diferente por grupo, podemos agregar una sintaxis de `CASE` a `ORDER BY` . En este ejemplo, queremos ordenar a los empleados del Departamento 1 por apellido y a los empleados del Departamento 2 por salario.

Carné de identidad	FName	LName	Número de teléfono	ManagerId	DepartmentId	Salario	Fecha de contratación
1	James	Herrero	1234567890	NULO	1	1000	01-01-2002
2	Juan	Johnson	2468101214	1	1	400	23-03-2005
3	Miguel	Williams	1357911131	1	2	600	12-05-2009
4	Johnathon	Herrero	1212121212	2	1	500	24-07-2016
5	Sam	sajón	1372141312	2	2	400	25-03-2015

```
The following query will provide the required results:
SELECT Id, FName, LName, Salary FROM Employees
ORDER BY Case When DepartmentId = 1 then LName else Salary end
```

Pedidos personalizados

Si desea ordenar por una columna utilizando algo distinto al orden alfabético / numérico, puede usar `case` y `case` para especificar el orden que desea.

`order by Group devoluciones de order by Group :`

Grupo	Contar
No jubilado	6
Retirado	4
Total	10

```
order by case group when 'Total' then 1 when 'Retired' then 2 else 3 end
```

devuelve:

Grupo	Contar
Total	10
Retirado	4
No jubilado	6

Lea **ORDEN POR** en línea: <https://riptutorial.com/es/sql-server/topic/4149/orden-por>

Capítulo 74: Ordenar / ordenar filas

Examples

Lo esencial

Primero, configuremos la tabla de ejemplo.

```
-- Create a table as an example
CREATE TABLE SortOrder
(
    ID INT IDENTITY PRIMARY KEY,
    [Text] VARCHAR(256)
)
GO

-- Insert rows into the table
INSERT INTO SortOrder ([Text])
SELECT ('Lorem ipsum dolor sit amet, consectetur adipiscing elit')
UNION ALL SELECT ('Pellentesque eu dapibus libero')
UNION ALL SELECT ('Vestibulum et consequat est, ut hendrerit ligula')
UNION ALL SELECT ('Suspendisse sodales est congue lorem euismod, vel facilisis libero
pulvinar')
UNION ALL SELECT ('Suspendisse lacus est, aliquam at varius a, fermentum nec mi')
UNION ALL SELECT ('Praesent tincidunt tortor est, nec consequat dolor malesuada quis')
UNION ALL SELECT ('Quisque at tempus arcu')
GO
```

Recuerde que al recuperar datos, si no especifica una cláusula de ordenamiento de filas (ORDER BY), el servidor SQL no garantiza la clasificación (orden de las columnas) **en ningún momento** . Realmente, en cualquier momento. Y no tiene sentido discutir sobre eso, se ha mostrado literalmente miles de veces y en todo el Internet.

No ORDER BY == sin clasificación. Fin de la historia.

```
-- It may seem the rows are sorted by identifiers,
-- but there is really no way of knowing if it will always work.
-- And if you leave it like this in production, Murphy gives you a 100% that it wont.
SELECT * FROM SortOrder
GO
```

Hay dos direcciones de datos que se pueden ordenar por:

- ascendiendo (moviéndose hacia arriba), usando ASC
- descendiendo (moviéndose hacia abajo), usando DESC

```
-- Ascending - upwards
SELECT * FROM SortOrder ORDER BY ID ASC
GO

-- Ascending is default
SELECT * FROM SortOrder ORDER BY ID
```

```
GO

-- Descending - downwards
SELECT * FROM SortOrder ORDER BY ID DESC
GO
```

Cuando ordene por la columna textual ((n) char o (n) varchar), preste atención a que la orden respeta la intercalación. Para obtener más información sobre la intercalación, consulte el tema.

Ordenar y ordenar datos puede consumir recursos. Aquí es donde los índices creados correctamente son útiles. Para más información sobre los índices busque el tema.

Existe la posibilidad de pseudoaleatorizar el orden de las filas en su conjunto de resultados. Solo obliga al orden a aparecer no determinista.

```
SELECT * FROM SortOrder ORDER BY CHECKSUM(NEWID())
GO
```

El orden se puede recordar en un procedimiento almacenado, y esa es la forma en que debe hacerlo si es el último paso para manipular el conjunto de filas antes de mostrárselo al usuario final.

```
CREATE PROCEDURE GetSortOrder
AS
    SELECT *
    FROM SortOrder
    ORDER BY ID DESC
GO

EXEC GetSortOrder
GO
```

Existe un soporte limitado (y pirateado) para ordenar también en las vistas de SQL Server, pero se recomienda que NO lo use.

```
/* This may or may not work, and it depends on the way
   your SQL Server and updates are installed */
CREATE VIEW VwSortOrder1
AS
    SELECT TOP 100 PERCENT *
    FROM SortOrder
    ORDER BY ID DESC
GO

SELECT * FROM VwSortOrder1
GO

-- This will work, but hey... should you really use it?
CREATE VIEW VwSortOrder2
AS
    SELECT TOP 99999999 *
    FROM SortOrder
    ORDER BY ID DESC
GO
```

```
SELECT * FROM VwSortOrder2
GO
```

Para ordenar puede usar nombres de columna, alias o números de columna en su ORDEN POR.

```
SELECT *
FROM SortOrder
ORDER BY [Text]

-- New resultset column aliased as 'Msg', feel free to use it for ordering
SELECT ID, [Text] + ' (' + CAST(ID AS nvarchar(10)) + ')' AS Msg
FROM SortOrder
ORDER BY Msg

-- Can be handy if you know your tables, but really NOT GOOD for production
SELECT *
FROM SortOrder
ORDER BY 2
```

Aconsejo no utilizar los números en su código, excepto si desea olvidarlo al momento de ejecutarlo.

Orden por caso

Si desea ordenar sus datos numéricamente o alfabéticamente, simplemente puede usar el `order by [column]`. Si desea ordenar usando una jerarquía personalizada, use una declaración de caso.

```
Group
-----
Total
Young
MiddleAge
Old
Male
Female
```

Utilizando un `order by` básico `order by` :

```
Select * from MyTable
Order by Group
```

devuelve un orden alfabético, que no siempre es deseable:

```
Group
-----
Female
Male
MiddleAge
Old
Total
Young
```

Agregando una declaración de 'caso', asignando valores numéricos ascendentes en el orden en

que desea ordenar sus datos:

```
Select * from MyTable
Order by case Group
  when 'Total' then 10
  when 'Male' then 20
  when 'Female' then 30
  when 'Young' then 40
  when 'MiddleAge' then 50
  when 'Old' then 60
end
```

Devuelve los datos en el orden especificado:

```
Group
-----
Total
Male
Female
Young
MiddleAge
Old
```

Lea Ordenar / ordenar filas en línea: <https://riptutorial.com/es/sql-server/topic/5332/ordenar---ordenar-filas>

Capítulo 75: Paginación

Introducción

Desplazamiento de fila y paginación en varias versiones de SQL Server

Sintaxis

- SELECCIONAR * DE TableName ORDENAR POR ID OFFSET 10 ROWS FETCH SIGUIENTE 10 ROWS SOLAMENTE;

Examples

Paginación usando ROW_NUMBER con una expresión de tabla común

SQL Server 2008

La función `ROW_NUMBER` puede asignar un número creciente a cada fila en un conjunto de resultados. Combinado con una [expresión de tabla común](#) que usa un operador `BETWEEN`, es posible crear 'páginas' de conjuntos de resultados. Por ejemplo: la primera página contiene los resultados 1-10, la segunda página contiene los resultados 11-20, la tercera página contiene los resultados 21-30 y así sucesivamente.

```
WITH data
AS
(
    SELECT ROW_NUMBER() OVER (ORDER BY name) AS row_id,
           object_id,
           name,
           type,
           create_date
    FROM sys.objects
)
SELECT *
FROM data
WHERE row_id BETWEEN 41 AND 50
```

Nota: no es posible utilizar `ROW_NUMBER` en una cláusula `WHERE` como:

```
SELECT object_id,
       name,
       type,
       create_date
FROM sys.objects
WHERE ROW_NUMBER() OVER (ORDER BY name) BETWEEN 41 AND 50
```

Aunque esto sería más conveniente, el servidor SQL devolverá el siguiente error en este caso:

Msg 4108, nivel 15, estado 1, línea 6

Las funciones de ventana solo pueden aparecer en las cláusulas SELECT u ORDER BY.

Paginación con OFFSET FETCH

SQL Server 2012

La cláusula `OFFSET FETCH` implementa la paginación de una manera más concisa. Con esto, es posible omitir las filas N1 (especificadas en `OFFSET`) y devolver las siguientes filas N2 (especificadas en `FETCH`):

```
SELECT *
FROM sys.objects
ORDER BY object_id
OFFSET 40 ROWS FETCH NEXT 10 ROWS ONLY
```

Se requiere la cláusula `ORDER BY` para proporcionar resultados deterministas.

Paginaton con consulta interna.

En versiones anteriores de SQL Server, los desarrolladores tenían que usar la clasificación doble combinada con la palabra clave `TOP` para devolver filas en una página:

```
SELECT TOP 10 *
FROM
(
    SELECT
        TOP 50 object_id,
            name,
            type,
            create_date
        FROM sys.objects
        ORDER BY name ASC
    ) AS data
ORDER BY name DESC
```

La consulta interna devolverá las primeras 50 filas ordenadas por `name`. Luego, la consulta externa revertirá el orden de estas 50 filas y seleccionará las 10 filas superiores (estas serán las últimas 10 filas en el grupo antes de la reversión).

Paginación en varias versiones de SQL Server

SQL Server 2012/2014

```
DECLARE @RowsPerPage INT = 10, @PageNumber INT = 4
SELECT OrderId, ProductId
FROM OrderDetail
```

```
ORDER BY OrderId
OFFSET (@PageNumber - 1) * @RowsPerPage ROWS
FETCH NEXT @RowsPerPage ROWS ONLY
```

SQL Server 2005/2008 / R2

```
DECLARE @RowsPerPage INT = 10, @PageNumber INT = 4
SELECT OrderId, ProductId
FROM (
    SELECT OrderId, ProductId, ROW_NUMBER() OVER (ORDER BY OrderId) AS RowNum
    FROM OrderDetail) AS OD
WHERE OD.RowNum BETWEEN ((@PageNumber - 1) * @RowsPerPage) + 1
AND @RowsPerPage * @PageNumber
```

SQL Server 2000

```
DECLARE @RowsPerPage INT = 10, @PageNumber INT = 4
SELECT OrderId, ProductId
FROM (SELECT TOP (@RowsPerPage) OrderId, ProductId
      FROM (SELECT TOP (@PageNumber)*@RowsPerPage OrderId, ProductId
            FROM OrderDetail
            ORDER BY OrderId) AS OD
      ORDER BY OrderId DESC) AS OD2
ORDER BY OrderId ASC
```

SQL Server 2012/2014 utilizando ORDER BY OFFSET y FETCH NEXT

Para obtener las siguientes 10 filas simplemente ejecuta esta consulta:

```
SELECT * FROM TableName ORDER BY id OFFSET 10 ROWS FETCH NEXT 10 ROWS ONLY;
```

Puntos clave a considerar cuando se usa:

- ORDER BY es obligatorio para usar la cláusula OFFSET y FETCH .
- OFFSET cláusula OFFSET es obligatoria con FETCH . Nunca puedes usar, ORDER BY ... FETCH .
- TOP no se puede combinar con OFFSET y FETCH en la misma expresión de consulta.

Lea Paginación en línea: <https://riptutorial.com/es/sql-server/topic/6874/paginacion>

Capítulo 76: PARA JSON

Examples

PARA JSON PATH

Formatea los resultados de la consulta SELECT como texto JSON. La cláusula FOR JSON PATH se agrega después de la consulta:

```
SELECT top 3 object_id, name, type, principal_id FROM sys.objects
FOR JSON PATH
```

Los nombres de columna se utilizarán como claves en JSON, y los valores de celda se generarán como valores JSON. El resultado de la consulta sería una matriz de objetos JSON:

```
[
  {"object_id":3,"name":"sysrscols","type":"S "},
  {"object_id":5,"name":"sysrowsets","type":"S "},
  {"object_id":6,"name":"sysclones","type":"S "}
]
```

Los valores NULL en la columna principal_id se ignorarán (no se generarán).

FOR JSON PATH con alias de columna

FOR JSON PATH le permite controlar el formato del JSON de salida utilizando alias de columna:

```
SELECT top 3 object_id as id, name as [data.name], type as [data.type]
FROM sys.objects
FOR JSON PATH
```

El alias de columna se utilizará como nombre de clave. Los alias de columna separados por puntos (data.name y data.type) se generarán como objetos anidados. Si dos columnas tienen el mismo prefijo en notación de puntos, se agruparán en un solo objeto (datos en este ejemplo):

```
[
  {"id":3,"data":{"name":"sysrscols","type":"S "}},
  {"id":5,"data":{"name":"sysrowsets","type":"S "}},
  {"id":6,"data":{"name":"sysclones","type":"S "}}
]
```

Cláusula FOR JSON sin contenedor de matriz (objeto único en la salida)

La opción WITHOUT_ARRAY_WRAPPER le permite generar un solo objeto en lugar de la matriz. Utilice esta opción si sabe que devolverá una sola fila / objeto:

```
SELECT top 3 object_id, name, type, principal_id
```



```
FROM sys.objects
WHERE object_id = 3
FOR JSON PATH, WITHOUT_ARRAY_WRAPPER
```

Solo se devolverá un objeto en este caso:

```
{"object_id":3,"name":"sysrscols","type":"S "}
```

INCLUYE_NULL_VALUES

La cláusula FOR JSON ignora los valores NULL en las celdas. Si desea generar "clave": pares nulos para celdas que contienen valores NULL, agregue la opción INCLUDE_NULL_VALUES en la consulta:

```
SELECT top 3 object_id, name, type, principal_id
FROM sys.objects
FOR JSON PATH, INCLUDE_NULL_VALUES
```

Se generarán valores NULL en la columna principal_id:

```
[
  {"object_id":3,"name":"sysrscols","type":"S ","principal_id":null},
  {"object_id":5,"name":"sysrowsets","type":"S ","principal_id":null},
  {"object_id":6,"name":"sysclones","type":"S ","principal_id":null}
]
```

Envolviendo resultados con objeto ROOT

Envuelve la matriz JSON devuelta en un objeto raíz adicional con la clave especificada:

```
SELECT top 3 object_id, name, type FROM sys.objects
FOR JSON PATH, ROOT('data')
```

El resultado de la consulta sería una matriz de objetos JSON dentro del objeto contenedor:

```
{
  "data":[
    {"object_id":3,"name":"sysrscols","type":"S "},
    {"object_id":5,"name":"sysrowsets","type":"S "},
    {"object_id":6,"name":"sysclones","type":"S "}
  ]
}
```

Para json auto

Anida automáticamente los valores de la segunda tabla como una sub-matriz anidada de objetos JSON:

```
SELECT top 5 o.object_id, o.name, c.column_id, c.name
FROM sys.objects o
```

```
JOIN sys.columns c ON o.object_id = c.object_id
FOR JSON AUTO
```

El resultado de la consulta sería una matriz de objetos JSON:

```
[
  {
    "object_id":3,
    "name":"sysrscols",
    "c":[
      {"column_id":12,"name":"bitpos"},
      {"column_id":6,"name":"cid"}
    ]
  },
  {
    "object_id":5,
    "name":"sysrowsets",
    "c":[
      {"column_id":13,"name":"colguid"},
      {"column_id":3,"name":"hbcolid"},
      {"column_id":8,"name":"maxinrowlen"}
    ]
  }
]
```

Creando estructura JSON anidada personalizada

Si necesita alguna estructura JSON compleja que no se pueda crear utilizando FOR JSON PATH o FOR JSON AUTO, puede personalizar su salida JSON colocando las subconsultas FOR JSON como expresiones de columna:

```
SELECT top 5 o.object_id, o.name,
  (SELECT column_id, c.name
   FROM sys.columns c WHERE o.object_id = c.object_id
   FOR JSON PATH) as columns,
  (SELECT parameter_id, name
   FROM sys.parameters p WHERE o.object_id = p.object_id
   FOR JSON PATH) as parameters
FROM sys.objects o
FOR JSON PATH
```

Cada subconsulta producirá un resultado JSON que se incluirá en el contenido principal de JSON.

Lea **PARA JSON en línea**: <https://riptutorial.com/es/sql-server/topic/4661/para-json>

Capítulo 77: PARA XML PATH

Observaciones

También hay varios otros modos FOR XML :

- FOR XML RAW : crea un elemento <row> por fila.
- FOR XML AUTO - Intenta autogenerar heurísticamente una jerarquía.
- FOR XML EXPLICIT : proporciona más control sobre la forma del XML, pero es más engorroso que FOR XML PATH .

Examples

Hola mundo XML

```
SELECT 'Hello World' FOR XML PATH('example')
```

```
<example>Hello World</example>
```

Especificando espacios de nombres

SQL Server 2008

```
WITH XMLNAMESPACES (  
    DEFAULT 'http://www.w3.org/2000/svg',  
    'http://www.w3.org/1999/xlink' AS xlink  
)  
SELECT  
    'example.jpg' AS 'image/@xlink:href',  
    '50px' AS 'image/@width',  
    '50px' AS 'image/@height'  
FOR XML PATH('svg')
```

```
<svg xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://www.w3.org/2000/svg">  
    <image xlink:href="firefox.jpg" width="50px" height="50px"/>  
</svg>
```

Especificando la estructura usando expresiones XPath

```
SELECT  
    'XPath example' AS 'head/title',  
    'This example demonstrates ' AS 'body/p',  
    'https://www.w3.org/TR/xpath/' AS 'body/p/a/@href',  
    'XPath expressions' AS 'body/p/a'  
FOR XML PATH('html')
```

```
<html>
```

```

<head>
  <title>XPath example</title>
</head>
<body>
  <p>This example demonstrates <a href="https://www.w3.org/TR/xpath/">XPath
expressions</a></p>
</body>
</html>

```

En FOR XML PATH , las columnas sin nombre se convierten en nodos de texto. NULL o '' por lo tanto se convierten en nodos de texto vacíos. Nota: puede convertir una columna con nombre en una sin nombre usando AS *

```

DECLARE @tempTable TABLE (Ref INT, Des NVARCHAR(100), Qty INT)
INSERT INTO @tempTable VALUES (100001, 'Normal', 1), (100002, 'Foobar', 1), (100003, 'Hello
World', 2)

SELECT ROW_NUMBER() OVER (ORDER BY Ref) AS '@NUM',
       'REF' AS 'FLD/@NAME', REF AS 'FLD', '',
       'DES' AS 'FLD/@NAME', DES AS 'FLD', '',
       'QTY' AS 'FLD/@NAME', QTY AS 'FLD'
FROM @tempTable
FOR XML PATH('LIN'), ROOT('row')

```

```

<row>
  <LIN NUM="1">
    <FLD NAME="REF">100001</FLD>
    <FLD NAME="DES">Normal</FLD>
    <FLD NAME="QTY">1</FLD>
  </LIN>
  <LIN NUM="2">
    <FLD NAME="REF">100002</FLD>
    <FLD NAME="DES">Foobar</FLD>
    <FLD NAME="QTY">1</FLD>
  </LIN>
  <LIN NUM="3">
    <FLD NAME="REF">100003</FLD>
    <FLD NAME="DES">Hello World</FLD>
    <FLD NAME="QTY">2</FLD>
  </LIN>
</row>

```

El uso de nodos de texto (vacíos) ayuda a separar el nodo de salida anterior del siguiente, de modo que SQL Server sabe que debe comenzar un nuevo elemento para la siguiente columna. De lo contrario, se confunde cuando el atributo ya existe en lo que cree que es el elemento "actual".

Por ejemplo, sin las cadenas vacías entre el elemento y el atributo en la declaración SELECT , SQL Server da un error:

La columna centrada en los atributos 'FLD / @ NAME' no debe aparecer después de un hermano no centrado en los atributos en la jerarquía XML en FOR XML PATH.

También tenga en cuenta que este ejemplo también envolvió el XML en un elemento raíz llamado

row , especificado por `ROOT('row')`

Usando FOR XML PATH para concatenar valores

El `FOR XML PATH` se puede usar para concatenar valores en una cadena. El siguiente ejemplo concatena valores en una cadena `CSV` :

```
DECLARE @DataSource TABLE
(
    [rowID] TINYINT
    , [FirstName] NVARCHAR(32)
);

INSERT INTO @DataSource ([rowID], [FirstName])
VALUES (1, 'Alex')
    , (2, 'Peter')
    , (3, 'Alexsandyr')
    , (4, 'George');

SELECT STUFF
(
    (
        SELECT ',' + [FirstName]
        FROM @DataSource
        ORDER BY [rowID] DESC
        FOR XML PATH(''), TYPE
    ).value('.', 'NVARCHAR(MAX)')
    , 1
    , 1
    , ''
);
```

Algunas notas importantes:

- La cláusula `ORDER BY` se puede utilizar para ordenar los valores de una manera preferida
- si se utiliza un valor más largo como separador de concatenación, el parámetro de la función `STUFF` debe cambiarse;

```
SELECT STUFF
(
    (
        SELECT '---' + [FirstName]
        FROM @DataSource
        ORDER BY [rowID] DESC
        FOR XML PATH(''), TYPE
    ).value('.', 'NVARCHAR(MAX)')
    , 1
    , 3 -- the "3" could also be represented as: LEN('---') for clarity
    , ''
);
```

- a medida que se usan la opción `TYPE` y la función `.value` , la concatenación funciona con la cadena `NVARCHAR (MAX)`

Lea **PARA XML PATH** en línea: <https://riptutorial.com/es/sql-server/topic/727/para-xml-path>

Capítulo 78: Parámetros de tabla de valores

Observaciones

Los parámetros con valores de tabla (TVP para abreviar) son parámetros que se pasan a un procedimiento almacenado o una función que contiene datos estructurados en tablas. El uso de parámetros con valores de tabla requiere la creación de un [tipo de tabla definido por el usuario](#) para el parámetro que se está utilizando.

Los parámetros de valor de tabla son parámetros de solo lectura.

Examples

Uso de un parámetro con valores de tabla para insertar varias filas en una tabla

Primero, defina un [tipo de tabla definida utilizada](#) para usar:

```
CREATE TYPE names as TABLE
(
    FirstName varchar(10),
    LastName varchar(10)
)
GO
```

Crear el procedimiento almacenado:

```
CREATE PROCEDURE prInsertNames
(
    @Names dbo.Names READONLY -- Note: You must specify the READONLY
)
AS

INSERT INTO dbo.TblNames (FirstName, LastName)
SELECT FirstName, LastName
FROM @Names
GO
```

Ejecutando el procedimiento almacenado:

```
DECLARE @names dbo.Names
INSERT INTO @Names VALUES
('Zohar', 'Peled'),
('First', 'Last')

EXEC dbo.prInsertNames @Names
```

Lea [Parámetros de tabla de valores en línea](https://riptutorial.com/es/sql-server/topic/5285/parametros-de-tabla-de-valores): <https://riptutorial.com/es/sql-server/topic/5285/parametros-de-tabla-de-valores>

Capítulo 79: Parsename

Sintaxis

- PARSENAME ('object_name', object_piece)

Parámetros

'nombre del objeto'	object_piece
Es el nombre del objeto para el cual recuperar la parte del objeto especificado. nombre_objeto es nombre_sistema. Este parámetro es un nombre de objeto calificado opcionalmente. Si todas las partes del nombre del objeto están calificadas, este nombre puede tener cuatro partes: el nombre del servidor, el nombre de la base de datos, el nombre del propietario y el nombre del objeto.	Es la parte objeto a devolver. object_piece es de tipo int y puede tener estos valores: 1 = nombre de objeto 2 = nombre de esquema 3 = nombre de base de datos 4 = nombre de servidor

Examples

Nombre de pila

```
Declare @ObjectName nVarChar(1000)
Set @ObjectName = 'HeadOfficeSQL1.Northwind.dbo.Authors'

SELECT
  PARSENAME(@ObjectName, 4) as Server
, PARSENAME(@ObjectName, 3) as DB
, PARSENAME(@ObjectName, 2) as Owner
, PARSENAME(@ObjectName, 1) as Object
```

Devoluciones:

Servidor	DB
HeadofficeSQL1	Viento del norte

Propietario	Objeto
dbo	Autores

Lea Parsename en línea: <https://riptutorial.com/es/sql-server/topic/5775/parsename>

Capítulo 80: Particionamiento

Examples

Recuperar valores de límite de partición

```
SELECT      ps.name AS PartitionScheme
           , fg.name AS [FileGroup]
           , prv.*
           , LAG(prv.Value) OVER (PARTITION BY ps.name ORDER BY ps.name, boundary_id) AS
PreviousBoundaryValue

FROM        sys.partition_schemes ps
INNER JOIN  sys.destination_data_spaces dds
           ON dds.partition_scheme_id = ps.data_space_id
INNER JOIN  sys.filegroups fg
           ON dds.data_space_id = fg.data_space_id
INNER JOIN  sys.partition_functions f
           ON f.function_id = ps.function_id
INNER JOIN  sys.partition_range_values prv
           ON f.function_id = prv.function_id
           AND dds.destination_id = prv.boundary_id
```

Cambio de particiones

Según esta [página de Microsoft TechNet] [1],

La partición de datos le permite administrar y acceder a subconjuntos de sus datos de manera rápida y eficiente, mientras mantiene la integridad de toda la recopilación de datos.

Cuando llama a la siguiente consulta, los datos no se mueven físicamente; solo los metadatos sobre la ubicación de los datos cambian.

```
ALTER TABLE [SourceTable] SWITCH TO [TargetTable]
```

Las tablas deben tener las mismas columnas con los mismos tipos de datos y configuraciones NULL, deben estar en el mismo grupo de archivos y la nueva tabla de destino debe estar vacía. Consulte el enlace de la página anterior para obtener más información sobre el cambio de particiones.

[1]: [https://technet.microsoft.com/en-us/library/ms191160\(v=sql.105\).aspx](https://technet.microsoft.com/en-us/library/ms191160(v=sql.105).aspx) La propiedad `IDENTITY` columna puede diferir.

Recupere los valores de la tabla de partición, columna, esquema, función, total y mínimo-máximo utilizando una única consulta

```
SELECT DISTINCT
    object_name(i.object_id) AS [Object Name],
```



```

c.name AS [Partition Column],
s.name AS [Partition Scheme],
pf.name AS [Partition Function],
prv.tot AS [Partition Count],
prv.miVal AS [Min Boundry Value],
prv.maVal AS [Max Boundry Value]
FROM sys.objects o
INNER JOIN sys.indexes i ON i.object_id = o.object_id
INNER JOIN sys.columns c ON c.object_id = o.object_id
INNER JOIN sys.index_columns ic ON ic.object_id = o.object_id
    AND ic.column_id = c.column_id
    AND ic.partition_ordinal = 1
INNER JOIN sys.partition_schemes s ON i.data_space_id = s.data_space_id
INNER JOIN sys.partition_functions pf ON pf.function_id = s.function_id
OUTER APPLY(SELECT
    COUNT(*) tot, MIN(value) miVal, MAX(value) maVal
    FROM sys.partition_range_values prv
    WHERE prv.function_id = pf.function_id) prv
--WHERE object_name(i.object_id) = 'table_name'
ORDER BY OBJECT_NAME(i.object_id)

```

Solo elimine el comentario `where` cláusula y reemplace `table_name` con actual table name para ver los detalles del objeto deseado.

Lea Particionamiento en línea: <https://riptutorial.com/es/sql-server/topic/3212/particionamiento>

Capítulo 81: Permisos de base de datos

Observaciones

Sintaxis Básica:

```
{GRANT| REVOKE | DENY} {PERMISSION_NAME} [ON {SECURABLE}] TO {PRINCIPAL};
```

- {GRANT | Revocado | DENY} - Lo que estás tratando de lograr
 - Subvención: "Dar este permiso al principal declarado"
 - Revocar: "Quita este permiso del principal declarado"
 - Denegar: "Asegúrate de que el principal establecido nunca tenga este permiso (es decir," DENY SELECT "significa que, independientemente de cualquier otro permiso, SELECT fallará para este principal)
- PERMISSION_NAME: la operación a la que estás intentando afectar. Esto dependerá de lo asegurable. Por ejemplo, no tiene sentido GRANT SELECT en un procedimiento almacenado.
- SEGURO: el nombre de la cosa en la que está tratando de afectar los permisos. Esto es *opcional* . Diciendo GRANT SELECT TO [aUser]; es perfectamente aceptable; significa "para cualquier asegurable para el que el permiso SELECT tenga sentido, GRANT ese permiso".
- PRINCIPAL: para quién está tratando de afectar los permisos. A nivel de base de datos, puede ser un rol (aplicación o base de datos) o un usuario (asignado a un inicio de sesión o no), por ejemplo.

Examples

Cambio de permisos

```
GRANT SELECT ON [dbo].[someTable] TO [aUser];

REVOKE SELECT ON [dbo].[someTable] TO [aUser];
--REVOKE SELECT [dbo].[someTable] FROM [aUser]; is equivalent

DENY SELECT ON [dbo].[someTable] TO [aUser];
```

CREAR USUARIO

```
--implicitly map this user to a login of the same name as the user
CREATE USER [aUser];

--explicitly mapping what login the user should be associated with
CREATE USER [aUser] FOR LOGIN [aUser];
```

Crear un rol

```
CREATE ROLE [myRole];
```

Cambio de membresía de rol

```
-- SQL 2005+
exec sp_addrolemember @rolename = 'myRole', @membername = 'aUser';
exec sp_droprolemember @rolename = 'myRole', @membername = 'aUser';

-- SQL 2008+
ALTER ROLE [myRole] ADD MEMBER [aUser];
ALTER ROLE [myRole] DROP MEMBER [aUser];
```

Nota: los miembros de rol pueden ser cualquier principal de nivel de base de datos. Es decir, puede agregar un rol como miembro en otro rol. Además, agregar / quitar miembros de rol es idempotente. Es decir, intentar agregar / soltar resultará en su presencia / ausencia (respectivamente) en el rol, independientemente del estado actual de su rol como miembro.

Lea Permisos de base de datos en línea: <https://riptutorial.com/es/sql-server/topic/6788/permisos-de-base-de-datos>

Capítulo 82: Permisos y seguridad

Examples

Asignar permisos de objeto a un usuario

En Producción, es una buena práctica proteger sus datos y permitir que solo se realicen operaciones a través de procedimientos almacenados. Esto significa que su aplicación no puede ejecutar directamente operaciones CRUD en sus datos y potencialmente causar problemas. Asignar permisos es una tarea complicada, que requiere mucho tiempo y generalmente es onerosa. Por este motivo, a menudo es más fácil aprovechar parte del (considerable) poder contenido en el esquema de [información INFORMATION_SCHEMA](#) que se encuentra en cada base de datos de SQL Server.

En lugar de asignar individualmente permisos a un usuario en una sola pieza, simplemente ejecute el script a continuación, copie el resultado y luego ejecútelo en una ventana de consulta.

```
SELECT 'GRANT EXEC ON core.' + r.ROUTINE_NAME + ' TO ' + <MyDatabaseUsername>
FROM INFORMATION_SCHEMA.ROUTINES r
WHERE r.ROUTINE_CATALOG = '<MyDataBaseName>'
```

Lea Permisos y seguridad en línea: <https://riptutorial.com/es/sql-server/topic/7929/permisos-y-seguridad>

Capítulo 83: Pivot dinámico de SQL

Introducción

Este tema trata sobre cómo hacer un pivote dinámico en SQL Server.

Examples

Pivote de SQL dinámico básico

```
if object_id('tempdb.dbo.#temp') is not null drop table #temp
create table #temp
(
    dateValue datetime,
    category varchar(3),
    amount decimal(36,2)
)

insert into #temp values ('1/1/2012', 'ABC', 1000.00)
insert into #temp values ('2/1/2012', 'DEF', 500.00)
insert into #temp values ('2/1/2012', 'GHI', 800.00)
insert into #temp values ('2/10/2012', 'DEF', 700.00)
insert into #temp values ('3/1/2012', 'ABC', 1100.00)

DECLARE
    @cols AS NVARCHAR(MAX),
    @query AS NVARCHAR(MAX);

SET @cols = STUFF((SELECT distinct ',' + QUOTENAME(c.category)
FROM #temp c
FOR XML PATH(''), TYPE
).value('.', 'NVARCHAR(MAX)')
,1,1, '')

set @query = '
SELECT
    dateValue,
    ' + @cols + '
from
(
    select
        dateValue,
        amount,
        category
    from #temp
) x
pivot
(
    sum(amount)
    for category in (' + @cols + ')
) p '

exec sp_executeSql @query
```

Lea Pivot dinámico de SQL en línea: <https://riptutorial.com/es/sql-server/topic/10751/pivot-dinamico-de-sql>

Capítulo 84: PIVOTE / UNPIVOT

Sintaxis

- SELECCIONAR <non-pivoted column> ,
[primera columna pivotada] AS <column name> ,
[segunda columna pivotada] AS <column name> ,
...
[última columna pivotada] AS <column name>
DESDE
(<SELECT query that produces the data>)
AS <alias for the source query>
PIVOTE
(
<aggregation function> (<column being aggregated>)
PARA
[<column that contains the values that will become column headers>]
IN ([primera columna pivotada], [segunda columna pivotada],
... [última columna pivotada])
) AS <alias for the pivot table> <optional ORDER BY clause> ;

Observaciones

Usando los operadores PIVOT y UNPIVOT, transforma una tabla al cambiar las filas (valores de columna) de una tabla a columnas y viceversa. Como parte de esta transformación, las funciones de agregación se pueden aplicar a los valores de la tabla.

Examples

Pivote simple - columnas estáticas

Usando la [tabla de ventas de artículos](#) de la [base de datos de ejemplo](#) , calculemos y mostremos la cantidad total que vendimos de cada producto.

Esto se puede hacer fácilmente con un grupo, pero supongamos que "giramos" nuestra tabla de resultados de manera que para cada ID de producto tengamos una columna.

```
SELECT [100], [145]
FROM (SELECT ItemId , Quantity
      FROM #ItemSalesTable
      ) AS pivotIntermediate
PIVOT ( SUM(Quantity)
      FOR ItemId IN ([100], [145])
      ) AS pivotTable
```

Dado que nuestras columnas 'nuevas' son números (en la tabla de origen), necesitamos corchetes []

Esto nos dará una salida como

100	145
45	18

Simple PIVOT & UNPIVOT (T-SQL)

A continuación se muestra un ejemplo simple que muestra el precio promedio de cada artículo por día de la semana.

Primero, suponga que tenemos una tabla que mantiene registros diarios de los precios de todos los artículos.

```
CREATE TABLE tbl_stock(item NVARCHAR(10), weekday NVARCHAR(10), price INT);

INSERT INTO tbl_stock VALUES
('Item1', 'Mon', 110), ('Item2', 'Mon', 230), ('Item3', 'Mon', 150),
('Item1', 'Tue', 115), ('Item2', 'Tue', 231), ('Item3', 'Tue', 162),
('Item1', 'Wed', 110), ('Item2', 'Wed', 240), ('Item3', 'Wed', 162),
('Item1', 'Thu', 109), ('Item2', 'Thu', 228), ('Item3', 'Thu', 145),
('Item1', 'Fri', 120), ('Item2', 'Fri', 210), ('Item3', 'Fri', 125),
('Item1', 'Mon', 122), ('Item2', 'Mon', 225), ('Item3', 'Mon', 140),
('Item1', 'Tue', 110), ('Item2', 'Tue', 235), ('Item3', 'Tue', 154),
('Item1', 'Wed', 125), ('Item2', 'Wed', 220), ('Item3', 'Wed', 142);
```

La tabla debe verse como a continuación:

```
+=====+=====+=====+
|  item | weekday | price |
+=====+=====+=====+
| Item1 |   Mon   |  110 |
+-----+-----+-----+
| Item2 |   Mon   |  230 |
+-----+-----+-----+
| Item3 |   Mon   |  150 |
+-----+-----+-----+
| Item1 |   Tue   |  115 |
+-----+-----+-----+
| Item2 |   Tue   |  231 |
+-----+-----+-----+
| Item3 |   Tue   |  162 |
+-----+-----+-----+
|      |   . . .   |      |
+-----+-----+-----+
| Item2 |   Wed   |  220 |
+-----+-----+-----+
| Item3 |   Wed   |  142 |
+-----+-----+-----+
```

Para realizar la agregación, que consiste en encontrar el precio promedio por artículo para cada

día de la semana, usaremos el operador relacional `PIVOT` para rotar la columna `weekday` de la `weekday` de la expresión con valores de tabla en valores de fila agregados como se muestra a continuación:

```
SELECT * FROM tbl_stock
PIVOT (
    AVG(price) FOR weekday IN ([Mon], [Tue], [Wed], [Thu], [Fri])
) pvt;
```

Resultado:

```
+-----+-----+-----+-----+-----+
| item | Mon | Tue | Wed | Thu | Fri |
+-----+-----+-----+-----+-----+
| Item1 | 116 | 112 | 117 | 109 | 120 |
| Item2 | 227 | 233 | 230 | 228 | 210 |
| Item3 | 145 | 158 | 152 | 145 | 125 |
+-----+-----+-----+-----+-----+
```

Por último, para realizar la operación inversa de `PIVOT`, podemos usar el operador relacional `UNPIVOT` para rotar las columnas en filas como se muestra a continuación:

```
SELECT * FROM tbl_stock
PIVOT (
    AVG(price) FOR weekday IN ([Mon], [Tue], [Wed], [Thu], [Fri])
) pvt
UNPIVOT (
    price FOR weekday IN ([Mon], [Tue], [Wed], [Thu], [Fri])
) unpvt;
```

Resultado:

```
+=====+=====+=====+
| item | price | weekday |
+=====+=====+=====+
| Item1 | 116 | Mon |
+-----+-----+-----+
| Item1 | 112 | Tue |
+-----+-----+-----+
| Item1 | 117 | Wed |
+-----+-----+-----+
| Item1 | 109 | Thu |
+-----+-----+-----+
| Item1 | 120 | Fri |
+-----+-----+-----+
| Item2 | 227 | Mon |
+-----+-----+-----+
| Item2 | 233 | Tue |
+-----+-----+-----+
| Item2 | 230 | Wed |
+-----+-----+-----+
| Item2 | 228 | Thu |
+-----+-----+-----+
| Item2 | 210 | Fri |
+-----+-----+-----+
| Item3 | 145 | Mon |
```

```

+-----+-----+-----+
| Item3 |    158 |    Tue |
+-----+-----+-----+
| Item3 |    152 |    Wed |
+-----+-----+-----+
| Item3 |    145 |    Thu |
+-----+-----+-----+
| Item3 |    125 |    Fri |
+-----+-----+-----+

```

PIVOTE Dinámico

Un problema con la consulta `PIVOT` es que debe especificar todos los valores dentro de la selección `IN` si desea verlos como columnas. Una forma rápida de evitar este problema es crear una selección `IN` dinámica que haga que su `PIVOT` dinámico.

Para demostración usaremos una tabla `Books` en la base de datos de una `Bookstore`. Suponemos que la tabla está bastante des-normalizada y tiene las siguientes columnas

```

Table: Books
-----
BookId (Primary Key Column)
Name
Language
NumberOfPages
EditionNumber
YearOfPrint
YearBoughtIntoStore
ISBN
AuthorName
Price
NumberOfUnitsSold

```

El script de creación para la tabla será como:

```

CREATE TABLE [dbo].[BookList](
    [BookId] [int] NOT NULL,
    [Name] [nvarchar](100) NULL,
    [Language] [nvarchar](100) NULL,
    [NumberOfPages] [int] NULL,
    [EditionNumber] [nvarchar](10) NULL,
    [YearOfPrint] [int] NULL,
    [YearBoughtIntoStore] [int] NULL,
    [NumberOfBooks] [int] NULL,
    [ISBN] [nvarchar](30) NULL,
    [AuthorName] [nvarchar](200) NULL,
    [Price] [money] NULL,
    [NumberOfUnitsSold] [int] NULL,
    CONSTRAINT [PK_BookList] PRIMARY KEY CLUSTERED
(
    [BookId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS =
ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

```

Ahora, si necesitamos consultar en la base de datos y calcular el número de libros en inglés, ruso, alemán, hindi y latinos que compramos en la librería cada año y presentar nuestra producción en un formato de informe pequeño, podemos usar la consulta PIVOT de esta manera.

```
SELECT * FROM
(
    SELECT YearBoughtIntoStore AS [Year Bought],[Language], NumberOfBooks
    FROM BookList
) sourceData
PIVOT
(
    SUM(NumberOfBooks)
    FOR [Language] IN (English, Russian, German, Hindi, Latin)
) pivotrReport
```

El caso especial es cuando no tenemos una lista completa de los idiomas, por lo que usaremos SQL dinámico como se muestra a continuación

```
DECLARE @query VARCHAR(4000)
DECLARE @languages VARCHAR(2000)
SELECT @languages =
    STUFF((SELECT DISTINCT ', [' + LTRIM([Language]) FROM [dbo].[BookList]
    ORDER BY ', [' + LTRIM([Language]) FOR XML PATH(') ),1,2,') + ')')
SET @query=
'SELECT * FROM
    (SELECT YearBoughtIntoStore AS [Year Bought],[Language],NumberOfBooks
    FROM BookList) sourceData
PIVOT(SUM(NumberOfBooks)FOR [Language] IN (' + @languages +')) pivotrReport' EXECUTE(@query)
```

Lea PIVOTE / UNPIVOT en línea: <https://riptutorial.com/es/sql-server/topic/591/pivote---unpivot>

Capítulo 85: Privilegios o permisos

Examples

Reglas simples

Otorgando permiso para crear tablas

```
USE AdventureWorks;  
GRANT CREATE TABLE TO MelanieK;  
GO
```

Concesión del permiso SHOWPLAN a un rol de aplicación

```
USE AdventureWorks2012;  
GRANT SHOWPLAN TO AuditMonitor;  
GO
```

Otorgando CREATE VIEW con GRANT OPTION

```
USE AdventureWorks2012;  
GRANT CREATE VIEW TO CarmineEs WITH GRANT OPTION;  
GO
```

Otorgar todos los derechos a un usuario en una base de datos específica

```
use YourDatabase  
go  
exec sp_addrolemember 'db_owner', 'UserName'  
go
```

Lea Privilegios o permisos en línea: <https://riptutorial.com/es/sql-server/topic/5333/privilegios-o-permisos>

Capítulo 86: Procedimientos almacenados

Introducción

En SQL Server, un procedimiento es un programa almacenado al que puede pasar parámetros. No devuelve un valor como lo hace una función. Sin embargo, puede devolver un estado de éxito / falla al procedimiento que lo llamó.

Sintaxis

- **CREATE** {PROCEDIMIENTO | PROC} [schema_name.] Procedure_name
- [@parameter [type_schema_name.] datatype
- [VARYING] [= predeterminado] [OUT | SALIDA | SOLO LECTURA]
- , @parameter [type_schema_name.] datatype
- [VARYING] [= predeterminado] [OUT | SALIDA | SOLO LECTURA]]
- [CON {ENCRYPTION | RECOMPILE | EJECUTAR COMO Cláusula}]
- [PARA REPLICACIÓN]
- COMO
- EMPEZAR
- [declaración_sección]
- sección ejecutable
- FIN;

Examples

Creación y ejecución de un procedimiento almacenado básico.

Usando la tabla de `Authors` en la [base de datos de la biblioteca](#)

```
CREATE PROCEDURE GetName
(
    @input_id INT = NULL,          --Input parameter, id of the person, NULL default
    @name VARCHAR(128) = NULL    --Input parameter, name of the person, NULL default
)
AS
BEGIN
    SELECT Name + ' is from ' + Country
    FROM Authors
    WHERE Id = @input_id OR Name = @name
END
GO
```

Puede ejecutar un procedimiento con algunas sintaxis diferentes. Primero, puedes usar `EXECUTE` o `EXEC`

```
EXECUTE GetName @id = 1
EXEC Getname @name = 'Ernest Hemingway'
```

Además, puede omitir el comando EXEC. Además, no tiene que especificar qué parámetro está pasando, ya que pasa en todos los parámetros.

```
GetName NULL, 'Ernest Hemingway'
```

Cuando desee especificar los parámetros de entrada en un orden diferente al de cómo se declaran en el procedimiento, puede especificar el nombre del parámetro y asignar valores. Por ejemplo

```
CREATE PROCEDURE dbo.sProcTemp
(
    @Param1 INT,
    @Param2 INT
)
AS
BEGIN

    SELECT
        Param1 = @Param1,
        Param2 = @Param2

END
```

el orden normal para ejecutar este procedimiento es especificar el valor para @ Param1 primero y luego @ Param2 segundo. Así se verá algo como esto

```
EXEC dbo.sProcTemp @Param1 = 0,@Param2=1
```

Pero también es posible que puedas usar lo siguiente.

```
EXEC dbo.sProcTemp @Param2 = 0,@Param1=1
```

en esto, está especificando el valor para @ param2 primero y @ Param1 segundo. Lo que significa que no tiene que mantener el mismo orden que se declara en el procedimiento, pero puede tener cualquier orden que desee. pero tendrá que especificar a qué parámetro está configurando el valor

Acceda al procedimiento almacenado desde cualquier base de datos.

Y también puede crear un procedimiento con un prefijo `sp_` estos procedimientos, como todos los procedimientos almacenados del sistema, se pueden ejecutar sin especificar la base de datos debido al comportamiento predeterminado de SQL Server. Cuando ejecuta un procedimiento almacenado que comienza con "sp_", SQL Server busca primero el procedimiento en la base de datos maestra. Si el procedimiento no se encuentra en el maestro, busca en la base de datos activa. Si tiene un procedimiento almacenado al que desea acceder desde todas sus bases de datos, créelo en el maestro y use un nombre que incluya el prefijo "sp_".

```
Use Master

CREATE PROCEDURE sp_GetName
```

```
(
    @input_id INT = NULL,          --Input parameter, id of the person, NULL default
    @name VARCHAR(128) = NULL    --Input parameter, name of the person, NULL default
)
AS
BEGIN
    SELECT Name + ' is from ' + Country
    FROM Authors
    WHERE Id = @input_id OR Name = @name
END
GO
```

PROCEDIMIENTO ALMACENADO con parámetros OUT

Los procedimientos almacenados pueden devolver valores utilizando la palabra clave `OUTPUT` en su lista de parámetros.

Creación de un procedimiento almacenado con un único parámetro de salida.

```
CREATE PROCEDURE SprocWithOutParams
(
    @InParam VARCHAR(30),
    @OutParam VARCHAR(30) OUTPUT
)
AS
BEGIN
    SELECT @OutParam = @InParam + ' must come out'
    RETURN
END
GO
```

Ejecutando el procedimiento almacenado

```
DECLARE @OutParam VARCHAR(30)
EXECUTE SprocWithOutParams 'what goes in', @OutParam OUTPUT
PRINT @OutParam
```

Creación de un procedimiento almacenado con múltiples parámetros de salida.

```
CREATE PROCEDURE SprocWithOutParams2
(
    @InParam VARCHAR(30),
    @OutParam VARCHAR(30) OUTPUT,
    @OutParam2 VARCHAR(30) OUTPUT
```

```

)
AS
BEGIN
    SELECT @OutParam = @InParam + ' must come out'
    SELECT @OutParam2 = @InParam + ' must come out'
    RETURN
END
GO

```

Ejecutando el procedimiento almacenado

```

DECLARE @OutParam VARCHAR(30)
DECLARE @OutParam2 VARCHAR(30)
EXECUTE SprocWithOutParams2 'what goes in', @OutParam OUTPUT, @OutParam2 OUTPUT
PRINT @OutParam
PRINT @OutParam2

```

Procedimiento almacenado con If ... Else e Insertar en operación

Crear tabla de ejemplo `Employee` :

```

CREATE TABLE Employee
(
    Id INT,
    EmpName VARCHAR(25),
    EmpGender VARCHAR(6),
    EmpDeptId INT
)

```

Creará un procedimiento almacenado que verifica si los valores pasados en el procedimiento almacenado no son nulos o no están vacíos y realizan una operación de inserción en la tabla `Empleado`.

```

CREATE PROCEDURE spSetEmployeeDetails
(
    @ID int,
    @Name VARCHAR(25),
    @Gender VARCHAR(6),
    @DeptId INT
)
AS
BEGIN
    IF (
        (@ID IS NOT NULL AND LEN(@ID) !=0)
        AND (@Name IS NOT NULL AND LEN(@Name) !=0)
        AND (@Gender IS NOT NULL AND LEN(@Gender) !=0)
        AND (@DeptId IS NOT NULL AND LEN(@DeptId) !=0)
    )
    BEGIN
        INSERT INTO Employee
        (
            Id,
            EmpName,
            EmpGender,

```



```

        EmpDeptId
    )
VALUES
(
    @ID,
    @Name,
    @Gender,
    @DeptId
)
END
ELSE
    PRINT 'Incorrect Parameters'
END
GO

```

Ejecutar el procedimiento almacenado.

```

DECLARE @ID INT,
        @Name VARCHAR(25),
        @Gender VARCHAR(6),
        @DeptId INT

EXECUTE spSetEmployeeDetails
    @ID = 1,
    @Name = 'Subin Nepal',
    @Gender = 'Male',
    @DeptId = 182666

```

SQL dinámico en procedimiento almacenado

El SQL dinámico nos permite generar y ejecutar sentencias de SQL en tiempo de ejecución. SQL dinámico es necesario cuando nuestras sentencias de SQL contienen un identificador que puede cambiar en diferentes tiempos de compilación.

Ejemplo simple de SQL dinámico:

```

CREATE PROC sp_dynamicSQL
@table_name      NVARCHAR(20),
@col_name        NVARCHAR(20),
@col_value       NVARCHAR(20)
AS
BEGIN
DECLARE @Query NVARCHAR(max)
SET @Query = 'SELECT * FROM ' + @table_name
SET @Query = @Query + ' WHERE ' + @col_name + ' = ' + '''+@col_value+''''
EXEC (@Query)
END

```

En la consulta SQL anterior, podemos ver que podemos usar la consulta anterior definiendo valores en `@table_name`, `@col_name`, and `@col_value` en tiempo de ejecución. La consulta se genera en tiempo de ejecución y se ejecuta. Esta es una técnica en la que podemos crear scripts completos como una cadena en una variable y ejecutarla. Podemos crear consultas más complejas utilizando SQL dinámico y el concepto de concatenación. Este concepto es muy poderoso cuando se desea crear un script que se pueda usar bajo varias condiciones.

Ejecutando procedimiento almacenado

```
DECLARE @table_name NVARCHAR(20) = 'ITCompanyInNepal',
        @col_name NVARCHAR(20) = 'Headquarter',
        @col_value NVARCHAR(20) = 'USA'

EXEC sp_dynamicSQL @table_name,
                 @col_name,
                 @col_value
```

Tabla que he usado

ID	CompanyName	CompanyAddress	Headquarter	NumberOfEmployee
1	CompanyOne	Kathmandu	USA	300
2	CompanyTwo	Kathmandu	USA	260
3	CompanyThree	Kathmandu	Nepal	300
4	CompanyFour	Kathmandu	Nepal	180
6	CompanySix	Janakpur	USA	50
7	CompanySeven	Janakpur	Australia	100
8	CompanyEight	Birganj	Australia	150
9	CompanyNine	Biratnagar	Canada	200
10	CompanyTen	Pokhara	India	85

Salida

ID	CompanyName	CompanyAddress	Headquarter	NumberOfEmployee
1	CompanyOne	Kathmandu	USA	300
2	CompanyTwo	Kathmandu	USA	260
6	CompanySix	Janakpur	USA	50
1	CompanyA	Banglore	USA	400
2	CompanyB	Banglore	USA	450

Bucle simple

Primero obtengamos algunos datos en una tabla temporal llamada #systables y un número de fila incremental para que podamos consultar un registro a la vez

```
select
    o.name,
    row_number() over (order by o.name) as rn
into
    #systables
from
    sys.objects as o
where
    o.type = 'S'
```

A continuación declaramos algunas variables para controlar el bucle y almacenar el nombre de la tabla en este ejemplo.

```
declare
    @rn int = 1,
    @maxRn int = (
```

```

        select
            max(rn)
        from
            #systables as s
    )
declare @tablename sys name

```

Ahora podemos hacer un bucle usando un simple tiempo. Incrementamos @rn en la declaración de select , pero esto también podría haber sido una declaración separada para el set @rn = @rn + 1 ex set @rn = @rn + 1 , dependerá de sus requisitos. También usamos el valor de @rn antes de que se incremente para seleccionar un solo registro de #systables . Por último imprimimos el nombre de la tabla.

```

while @rn <= @maxRn
begin

    select
        @tablename = name,
        @rn = @rn + 1
    from
        #systables as s
    where
        s.rn = @rn

    print @tablename

end

```

Bucle simple

```

CREATE PROCEDURE SprocWithSimpleLoop
(
    @SayThis VARCHAR(30),
    @ThisManyTimes INT
)
AS
BEGIN
    WHILE @ThisManyTimes > 0
    BEGIN
        PRINT @SayThis;
        SET @ThisManyTimes = @ThisManyTimes - 1;
    END

    RETURN;
END
GO

```

Lea Procedimientos almacenados en línea: <https://riptutorial.com/es/sql-server/topic/3213/procedimientos-almacenados>

Capítulo 87: Puntos de vista

Observaciones

Las vistas son consultas almacenadas que se pueden consultar como tablas regulares. Las vistas no forman parte del modelo físico de la base de datos. Cualquier cambio que se aplique al origen de datos de una vista, como una tabla, también se reflejará en la vista.

Examples

Crear una vista

```
CREATE VIEW dbo.PersonsView
AS
SELECT
    name,
    address
FROM persons;
```

Crear o reemplazar vista

Esta consulta eliminará la vista, si ya existe, y creará una nueva.

```
IF OBJECT_ID('dbo.PersonsView', 'V') IS NOT NULL
    DROP VIEW dbo.PersonsView
GO

CREATE VIEW dbo.PersonsView
AS
SELECT
    name,
    address
FROM persons;
```

Crear una vista con enlace de esquema

Si se crea una vista **CON VISUALIZACIÓN DE HERRAMIENTAS**, las tablas subyacentes no se pueden eliminar ni modificar de tal manera que puedan romper la vista. Por ejemplo, una columna de tabla a la que se hace referencia en una vista no se puede eliminar.

```
CREATE VIEW dbo.PersonsView
WITH SCHEMABINDING
AS
SELECT
    name,
    address
FROM dbo.PERSONS -- database schema must be specified when WITH SCHEMABINDING is present
```

Las vistas *sin* enlace de esquema pueden romperse si sus tablas subyacentes cambian o se eliminan. La consulta de una vista rota da como resultado un mensaje de error. `sp_refreshview` se puede utilizar para garantizar que las vistas existentes sin enlace de esquema no se rompan.

Lea Puntos de vista en línea: <https://riptutorial.com/es/sql-server/topic/5327/puntos-de-vista>

Capítulo 88: Recupera información sobre tu instancia

Examples

Recuperar servidores locales y remotos

Para recuperar una lista de todos los servidores registrados en la instancia:

```
EXEC sp_helpserver;
```

Obtenga información sobre las sesiones actuales y ejecuciones de consulta

```
sp_who2
```

Este procedimiento se puede utilizar para buscar información sobre las sesiones actuales del servidor SQL. Como es un procedimiento, a menudo es útil almacenar los resultados en una tabla temporal o variable de tabla para que uno pueda ordenar, filtrar y transformar los resultados según sea necesario.

Lo siguiente se puede usar para una versión consultable de `sp_who2` :

```
-- Create a variable table to hold the results of sp_who2 for querying purposes

DECLARE @who2 TABLE (
    SPID INT NULL,
    Status VARCHAR(1000) NULL,
    Login SYSNAME NULL,
    HostName SYSNAME NULL,
    BlkBy SYSNAME NULL,
    DBName SYSNAME NULL,
    Command VARCHAR(8000) NULL,
    CPUtime INT NULL,
    DiskIO INT NULL,
    LastBatch VARCHAR(250) NULL,
    ProgramName VARCHAR(250) NULL,
    SPID2 INT NULL, -- a second SPID for some reason...?
    REQUESTID INT NULL
)

INSERT INTO @who2
EXEC sp_who2

SELECT *
FROM @who2 w
WHERE 1=1
```

Ejemplos:

```

-- Find specific user sessions:
SELECT *
FROM @who2 w
WHERE 1=1
      and login = 'userName'

-- Find longest CPUtime queries:
SELECT top 5 *
FROM @who2 w
WHERE 1=1
order by CPUtime desc

```

Recuperar Edición y Versión de Instancia

```

SELECT SERVERPROPERTY('ProductVersion') AS ProductVersion,
SERVERPROPERTY('ProductLevel') AS ProductLevel,
SERVERPROPERTY('Edition') AS Edition,
SERVERPROPERTY('EngineEdition') AS EngineEdition;

```

Recuperar el tiempo de actividad de la instancia en días

```

SELECT DATEDIFF(DAY, login_time, getdate()) UpDays
FROM master..sysprocesses
WHERE spid = 1

```

Información sobre la versión de SQL Server

Para descubrir la edición de SQL Server, el nivel de producto y el número de versión, así como el nombre de la máquina host y el tipo de servidor:

```

SELECT SERVERPROPERTY('MachineName') AS Host,
SERVERPROPERTY('InstanceName') AS Instance,
DB_NAME() AS DatabaseContext,
SERVERPROPERTY('Edition') AS Edition,
SERVERPROPERTY('ProductLevel') AS ProductLevel,
CASE SERVERPROPERTY('IsClustered')
  WHEN 1 THEN 'CLUSTERED'
  ELSE 'STANDALONE' END AS ServerType,
@@VERSION AS VersionNumber;

```

Información general sobre bases de datos, tablas, procedimientos almacenados y cómo buscarlos.

Consulta para buscar los últimos sp ejecutados en db

```

SELECT execquery.last_execution_time AS [Date Time], execsql.text AS [Script]
FROM sys.dm_exec_query_stats AS execquery
CROSS APPLY sys.dm_exec_sql_text(execquery.sql_handle) AS execsql
ORDER BY execquery.last_execution_time DESC

```

Consulta para buscar a través de procedimientos almacenados.

```
SELECT o.type_desc AS ROUTINE_TYPE,o.[name] AS ROUTINE_NAME,
m.definition AS ROUTINE_DEFINITION
FROM sys.sql_modules AS m INNER JOIN sys.objects AS o
ON m.object_id = o.object_id WHERE m.definition LIKE '%Keyword%'
order by ROUTINE_NAME
```

Consulta para encontrar la columna de todas las tablas de la base de datos

```
SELECT t.name AS table_name,
SCHEMA_NAME(schema_id) AS schema_name,
c.name AS column_name
FROM sys.tables AS t
INNER JOIN sys.columns c ON t.OBJECT_ID = c.OBJECT_ID
where c.name like 'Keyword%'
ORDER BY schema_name, table_name;
```

Consulta para verificar detalles de restauración

```
WITH LastRestores AS
(
SELECT
    DatabaseName = [d].[name] ,
    [d].[create_date] ,
    [d].[compatibility_level] ,
    [d].[collation_name] ,
    r.*,
    RowNum = ROW_NUMBER() OVER (PARTITION BY d.Name ORDER BY r.[restore_date] DESC)
FROM master.sys.databases d
LEFT OUTER JOIN msdb.dbo.[restorehistory] r ON r.[destination_database_name] = d.Name
)
SELECT *
FROM [LastRestores]
WHERE [RowNum] = 1
```

Consulta para encontrar el registro.

```
select top 100 * from databaselog
Order by Posttime desc
```

Consulta para comprobar los detalles del SPS.

```
SELECT name, create_date, modify_date
FROM sys.objects
WHERE type = 'P'
Order by modify_date desc
```

Lea Recupera información sobre tu instancia en línea: <https://riptutorial.com/es/sql-server/topic/2029/recupera-informacion-sobre-tu-instancia>

Capítulo 89: Recuperar información sobre la base de datos.

Observaciones

Al igual que con otros sistemas de bases de datos relacionales, SQL Server expone metadatos sobre sus bases de datos.

Esto se proporciona a través del esquema estándar de ISO `INFORMATION_SCHEMA`, o las vistas de catálogo de `sys` específicas de SQL Server.

Examples

Cuenta el número de tablas en una base de datos

Esta consulta devolverá el número de tablas en la base de datos especificada.

```
USE YourDatabaseName
SELECT COUNT(*) from INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE TABLE'
```

La siguiente es otra forma en que esto se puede hacer para todas las tablas de usuarios con SQL Server 2008+. La referencia está [aquí](#).

```
SELECT COUNT(*) FROM sys.tables
```

Recuperar una lista de todos los procedimientos almacenados

Las siguientes consultas devolverán una lista de todos los procedimientos almacenados en la base de datos, con información básica sobre cada procedimiento almacenado:

SQL Server 2005

```
SELECT *
FROM INFORMATION_SCHEMA.ROUTINES
WHERE ROUTINE_TYPE = 'PROCEDURE'
```

Las `ROUTINE_NAME`, `ROUTINE_SCHEMA` y `ROUTINE_DEFINITION` son generalmente las más útiles.

SQL Server 2005

```
SELECT *
FROM sys.objects
WHERE type = 'P'
```

SQL Server 2005

```
SELECT *
FROM sys.procedures
```

Tenga en cuenta que esta versión tiene una ventaja sobre la selección de `sys.objects`, ya que incluye las columnas adicionales `is_auto_executed`, `is_execution_replicated`, `is_repl_serializable` y `skips_repl_constraints`.

SQL Server 2005

```
SELECT *
FROM sysobjects
WHERE type = 'P'
```

Tenga en cuenta que la salida contiene muchas columnas que nunca se relacionarán con un procedimiento almacenado.

El siguiente conjunto de consultas devolverá todos los procedimientos almacenados en la base de datos que incluyen la cadena 'SearchTerm':

SQL Server 2005

```
SELECT o.name
FROM syscomments c
INNER JOIN sysobjects o
    ON c.id=o.id
WHERE o.xtype = 'P'
    AND c.TEXT LIKE '%SearchTerm%'
```

SQL Server 2005

```
SELECT p.name
FROM sys.sql_modules AS m
INNER JOIN sys.procedures AS p
    ON m.object_id = p.object_id
WHERE definition LIKE '%SearchTerm%'
```

Obtener la lista de todas las bases de datos en un servidor

Método 1: la siguiente consulta será aplicable para la versión de SQL Server 2000+ (Contiene 12 columnas)

```
SELECT * FROM dbo.sysdatabases
```

Método 2: a continuación, obtenga información sobre las bases de datos con más información (por ejemplo, estado, aislamiento, modelo de recuperación, etc.)

Nota: Esta es una vista de catálogo y estará disponible en las versiones SQL SERVER 2005+

```
SELECT * FROM sys.databases
```

Método 3: para ver solo los nombres de las bases de datos, puede usar `sp_MSForEachDB` no

documentado

```
EXEC sp_MSForEachDB 'SELECT ''?' AS DatabaseName'
```

Método 4: debajo de SP le ayudará a proporcionar el tamaño de la base de datos junto con el nombre de la base de datos, el propietario, el estado, etc. en el servidor

```
EXEC sp_helpdb
```

Método 5 De manera similar, a continuación el procedimiento almacenado proporcionará el nombre de la base de datos, el tamaño de la base de datos y los Comentarios

```
EXEC sp_databases
```

Archivos de base de datos

Muestra todos los archivos de datos de todas las bases de datos con información de tamaño y crecimiento.

```
SELECT d.name AS 'Database',
       d.database_id,
       SF.fileid,
       SF.name AS 'LogicalFileName',
       CASE SF.status & 0x100000
         WHEN 1048576 THEN 'Percentage'
         WHEN 0 THEN 'MB'
       END AS 'FileGrowthOption',
       Growth AS GrowthUnit,
       ROUND(((CAST(Size AS FLOAT)*8)/1024)/1024,2) [SizeGB], -- Convert 8k pages to GB
       Maxsize,
       filename AS PhysicalFileName

FROM   Master.SYS.SYSALTFILES SF
Join   Master.SYS.Databases d on sf.fileid = d.database_id

Order by d.name
```

Recuperar opciones de base de datos

La siguiente consulta devuelve las opciones de base de datos y los metadatos:

```
select * from sys.databases WHERE name = 'MyDatabaseName';
```

Mostrar tamaño de todas las tablas en la base de datos actual

```
SELECT
  s.name + '.' + t.NAME AS TableName,
  SUM(a.used_pages)*8 AS 'TableSizeKB' --a page in SQL Server is 8kb
FROM sys.tables t
JOIN sys.schemas s on t.schema_id = s.schema_id
LEFT JOIN sys.indexes i ON t.OBJECT_ID = i.object_id
```

```

LEFT JOIN sys.partitions p ON i.object_id = p.OBJECT_ID AND i.index_id = p.index_id
LEFT JOIN sys.allocation_units a ON p.partition_id = a.container_id
GROUP BY
    s.name, t.name
ORDER BY
    --Either sort by name:
    s.name + '.' + t.NAME
    --Or sort largest to smallest:
    --SUM(a.used_pages) desc

```

Determine una ruta de acceso de inicio de sesión de Windows

Esto mostrará el tipo de usuario y la ruta de los permisos (desde qué grupo de ventanas está obteniendo los permisos).

```
xp_logininfo 'DOMAIN\user'
```

Recuperar tablas que contienen una columna conocida

Esta consulta devolverá todas las `COLUMNS` y sus `TABLES` asociadas para un nombre de columna dado. Está diseñado para mostrar qué tablas (desconocidas) contienen una columna específica (conocida)

```

SELECT
    c.name AS ColName,
    t.name AS TableName
FROM
    sys.columns c
    JOIN sys.tables t ON c.object_id = t.object_id
WHERE
    c.name LIKE '%MyName%'

```

Ver si se están utilizando características específicas de la empresa

A veces es útil verificar que su trabajo en Developer Edition no haya introducido una dependencia en ninguna de las funciones restringidas a la edición Enterprise.

Puedes hacer esto usando la vista del sistema `sys.dm_db_persisted_sku_features` , así:

```
SELECT * FROM sys.dm_db_persisted_sku_features
```

Contra la propia base de datos.

Esto mostrará una lista de las características que se utilizan, en su caso.

Buscar y devolver todas las tablas y columnas que contengan un valor de columna especificado

Esta secuencia de comandos, desde [aquí](#) y [aquí](#) , devolverá todas las tablas y columnas donde exista un valor específico. Esto es poderoso para descubrir dónde se encuentra un determinado

valor en una base de datos. Puede ser gravoso, por lo que se sugiere que se ejecute primero en un entorno de respaldo / prueba.

```
DECLARE @SearchStr nvarchar(100)
SET @SearchStr = '## YOUR STRING HERE ##'

-- Copyright © 2002 Narayana Vyas Kondreddi. All rights reserved.
-- Purpose: To search all columns of all tables for a given search string
-- Written by: Narayana Vyas Kondreddi
-- Site: http://vyaskn.tripod.com
-- Updated and tested by Tim Gaunt
-- http://www.thesitedoctor.co.uk
--
http://blogs.thesitedoctor.co.uk/tim/2010/02/19/Search+Every+Table+And+Field+In+A+SQL+Server+Database+

-- Tested on: SQL Server 7.0, SQL Server 2000, SQL Server 2005 and SQL Server 2010
-- Date modified: 03rd March 2011 19:00 GMT
CREATE TABLE #Results (ColumnName nvarchar(370), ColumnValue nvarchar(3630))

SET NOCOUNT ON

DECLARE @TableName nvarchar(256), @ColumnName nvarchar(128), @SearchStr2 nvarchar(110)
SET @TableName = ''
SET @SearchStr2 = QUOTENAME('%' + @SearchStr + '%','''')

WHILE @TableName IS NOT NULL

BEGIN
    SET @ColumnName = ''
    SET @TableName =
    (
        SELECT MIN(QUOTENAME(TABLE_SCHEMA) + '.' + QUOTENAME(TABLE_NAME))
        FROM     INFORMATION_SCHEMA.TABLES
        WHERE          TABLE_TYPE = 'BASE TABLE'
                  AND  QUOTENAME(TABLE_SCHEMA) + '.' + QUOTENAME(TABLE_NAME) > @TableName
                  AND  OBJECTPROPERTY(
                        OBJECT_ID(
                            QUOTENAME(TABLE_SCHEMA) + '.' + QUOTENAME(TABLE_NAME)
                        ), 'IsMSShipped'
                    ) = 0
    )

    WHILE (@TableName IS NOT NULL) AND (@ColumnName IS NOT NULL)

    BEGIN
        SET @ColumnName =
        (
            SELECT MIN(QUOTENAME(COLUMN_NAME))
            FROM     INFORMATION_SCHEMA.COLUMNS
            WHERE          TABLE_SCHEMA    = PARSENAME(@TableName, 2)
                  AND  TABLE_NAME      = PARSENAME(@TableName, 1)
                  AND  DATA_TYPE IN ('char', 'varchar', 'nchar', 'nvarchar', 'int',
'decimal')
                  AND  QUOTENAME(COLUMN_NAME) > @ColumnName
        )

        IF @ColumnName IS NOT NULL

        BEGIN
```

```

        INSERT INTO #Results
        EXEC
        (
            'SELECT ''' + @TableName + '.' + @ColumnName + ''', LEFT(' + @ColumnName +
', 3630) FROM ' + @TableName + ' (NOLOCK) ' +
            ' WHERE ' + @ColumnName + ' LIKE ' + @SearchStr2
        )
    END
END
END

SELECT ColumnName, ColumnValue FROM #Results

DROP TABLE #Results
- See more at: http://thesitedoctor.co.uk/blog/search-every-table-and-field-in-a-sql-server-database-updated#sthash.bBEqfJVZ.dpuf

```

Consigue todos los esquemas, tablas, columnas e índices.

```

SELECT
    s.name AS [schema],
    t.object_id AS [table_object_id],
    t.name AS [table_name],
    c.column_id,
    c.name AS [column_name],
    i.name AS [index_name],
    i.type_desc AS [index_type]
FROM sys.schemas AS s
INNER JOIN sys.tables AS t
    ON s.schema_id = t.schema_id
INNER JOIN sys.columns AS c
    ON t.object_id = c.object_id
LEFT JOIN sys.index_columns AS ic
    ON c.object_id = ic.object_id and c.column_id = ic.column_id
LEFT JOIN sys.indexes AS i
    ON ic.object_id = i.object_id and ic.index_id = i.index_id
ORDER BY [schema], [table_name], c.column_id;

```

Devuelva una lista de trabajos del Agente SQL, con información de programación

```

USE msdb
Go

SELECT dbo.sysjobs.Name AS 'Job Name',
    'Job Enabled' = CASE dbo.sysjobs.Enabled
        WHEN 1 THEN 'Yes'
        WHEN 0 THEN 'No'
    END,
    'Frequency' = CASE dbo.sysschedules.freq_type
        WHEN 1 THEN 'Once'
        WHEN 4 THEN 'Daily'
        WHEN 8 THEN 'Weekly'
        WHEN 16 THEN 'Monthly'
        WHEN 32 THEN 'Monthly relative'
        WHEN 64 THEN 'When SQLServer Agent starts'
    END

```

```

END,
'Start Date' = CASE active_start_date
    WHEN 0 THEN null
    ELSE
        substring(convert (varchar(15),active_start_date),1,4) + '/' +
        substring(convert (varchar(15),active_start_date),5,2) + '/' +
        substring(convert (varchar(15),active_start_date),7,2)
END,
'Start Time' = CASE len(active_start_time)
    WHEN 1 THEN cast ('00:00:0' + right (active_start_time,2) as char(8))
    WHEN 2 THEN cast ('00:00:' + right (active_start_time,2) as char(8))
    WHEN 3 THEN cast ('00:0'
        + Left (right (active_start_time,3),1)
        + ':' + right (active_start_time,2) as char (8))
    WHEN 4 THEN cast ('00:'
        + Left (right (active_start_time,4),2)
        + ':' + right (active_start_time,2) as char (8))
    WHEN 5 THEN cast ('0'
        + Left (right (active_start_time,5),1)
        + ':' + Left (right (active_start_time,4),2)
        + ':' + right (active_start_time,2) as char (8))
    WHEN 6 THEN cast (Left (right (active_start_time,6),2)
        + ':' + Left (right (active_start_time,4),2)
        + ':' + right (active_start_time,2) as char (8))

END,

CASE len(run_duration)
    WHEN 1 THEN cast ('00:00:0'
        + cast (run_duration as char) as char (8))
    WHEN 2 THEN cast ('00:00:'
        + cast (run_duration as char) as char (8))
    WHEN 3 THEN cast ('00:0'
        + Left (right (run_duration,3),1)
        + ':' + right (run_duration,2) as char (8))
    WHEN 4 THEN cast ('00:'
        + Left (right (run_duration,4),2)
        + ':' + right (run_duration,2) as char (8))
    WHEN 5 THEN cast ('0'
        + Left (right (run_duration,5),1)
        + ':' + Left (right (run_duration,4),2)
        + ':' + right (run_duration,2) as char (8))
    WHEN 6 THEN cast (Left (right (run_duration,6),2)
        + ':' + Left (right (run_duration,4),2)
        + ':' + right (run_duration,2) as char (8))

END as 'Max Duration',
CASE (dbo.syssschedules.freq_subday_interval)
    WHEN 0 THEN 'Once'
    ELSE cast ('Every '
        + right (dbo.syssschedules.freq_subday_interval,2)
        + ' '
        + CASE (dbo.syssschedules.freq_subday_type)
            WHEN 1 THEN 'Once'
            WHEN 4 THEN 'Minutes'
            WHEN 8 THEN 'Hours'
        END as char(16))

END as 'Subday Frequency'
FROM dbo.sysjobs
LEFT OUTER JOIN dbo.sysjobschedules
ON dbo.sysjobs.job_id = dbo.sysjobschedules.job_id
INNER JOIN dbo.syssschedules ON dbo.sysjobschedules.schedule_id = dbo.syssschedules.schedule_id
LEFT OUTER JOIN (SELECT job_id, max(run_duration) AS run_duration

```

```

        FROM dbo.sysjobhistory
        GROUP BY job_id) Q1
ON dbo.sysjobs.job_id = Q1.job_id
WHERE Next_run_time = 0

UNION

SELECT dbo.sysjobs.Name AS 'Job Name',
       'Job Enabled' = CASE dbo.sysjobs.Enabled
                        WHEN 1 THEN 'Yes'
                        WHEN 0 THEN 'No'
        END,
       'Frequency' = CASE dbo.sysschedules.freq_type
                       WHEN 1 THEN 'Once'
                       WHEN 4 THEN 'Daily'
                       WHEN 8 THEN 'Weekly'
                       WHEN 16 THEN 'Monthly'
                       WHEN 32 THEN 'Monthly relative'
                       WHEN 64 THEN 'When SQLServer Agent starts'
        END,
       'Start Date' = CASE next_run_date
                       WHEN 0 THEN null
                       ELSE
                           substring(convert(varchar(15),next_run_date),1,4) + '/' +
                           substring(convert(varchar(15),next_run_date),5,2) + '/' +
                           substring(convert(varchar(15),next_run_date),7,2)
        END,
       'Start Time' = CASE len(next_run_time)
                       WHEN 1 THEN cast('00:00:0' + right(next_run_time,2) as char(8))
                       WHEN 2 THEN cast('00:00:' + right(next_run_time,2) as char(8))
                       WHEN 3 THEN cast('00:0'
                                       + Left(right(next_run_time,3),1)
                                       + ':' + right(next_run_time,2) as char(8))
                       WHEN 4 THEN cast('00:'
                                       + Left(right(next_run_time,4),2)
                                       + ':' + right(next_run_time,2) as char(8))
                       WHEN 5 THEN cast('0' + Left(right(next_run_time,5),1)
                                       + ':' + Left(right(next_run_time,4),2)
                                       + ':' + right(next_run_time,2) as char(8))
                       WHEN 6 THEN cast(Left(right(next_run_time,6),2)
                                       + ':' + Left(right(next_run_time,4),2)
                                       + ':' + right(next_run_time,2) as char(8))
        END,

       CASE len(run_duration)
           WHEN 1 THEN cast('00:00:0'
                           + cast(run_duration as char) as char(8))
           WHEN 2 THEN cast('00:00:'
                           + cast(run_duration as char) as char(8))
           WHEN 3 THEN cast('00:0'
                           + Left(right(run_duration,3),1)
                           + ':' + right(run_duration,2) as char(8))
           WHEN 4 THEN cast('00:'
                           + Left(right(run_duration,4),2)
                           + ':' + right(run_duration,2) as char(8))
           WHEN 5 THEN cast('0'
                           + Left(right(run_duration,5),1)
                           + ':' + Left(right(run_duration,4),2)
                           + ':' + right(run_duration,2) as char(8))
           WHEN 6 THEN cast(Left(right(run_duration,6),2)
                           + ':' + Left(right(run_duration,4),2)

```



```

        + ':' + right(run_duration,2) as char (8))
    END as 'Max Duration',
CASE(dbo.sysschedules.freq_subday_interval)
    WHEN 0 THEN 'Once'
    ELSE cast('Every '
        + right(dbo.sysschedules.freq_subday_interval,2)
        + ' '
        + CASE(dbo.sysschedules.freq_subday_type)
            WHEN 1 THEN 'Once'
            WHEN 4 THEN 'Minutes'
            WHEN 8 THEN 'Hours'
        END as char(16))
    END as 'Subday Frequency'
FROM dbo.sysjobs
LEFT OUTER JOIN dbo.sysjobschedules ON dbo.sysjobs.job_id = dbo.sysjobschedules.job_id
INNER JOIN dbo.sysschedules ON dbo.sysjobschedules.schedule_id = dbo.sysschedules.schedule_id
LEFT OUTER JOIN (SELECT job_id, max(run_duration) AS run_duration
    FROM dbo.sysjobhistory
    GROUP BY job_id) Q1
ON dbo.sysjobs.job_id = Q1.job_id
WHERE Next_run_time <> 0

ORDER BY [Start Date],[Start Time]

```

Recuperar información sobre operaciones de copia de seguridad y restauración

Para obtener la lista de todas las operaciones de copia de seguridad realizadas en la instancia de la base de datos actual:

```

SELECT sdb.Name AS DatabaseName,
    COALESCE(CONVERT(VARCHAR(50), bus.backup_finish_date, 120), '-') AS LastBackUpDateTime
FROM sys.sysdatabases sdb
    LEFT OUTER JOIN msdb.dbo.backupset bus ON bus.database_name = sdb.name
ORDER BY sdb.name, bus.backup_finish_date DESC

```

Para obtener la lista de todas las operaciones de restauración realizadas en la instancia de la base de datos actual:

```

SELECT
    [d].[name] AS database_name,
    [r].restore_date AS last_restore_date,
    [r].[user_name],
    [bs].[backup_finish_date] AS backup_creation_date,
    [bmf].[physical_device_name] AS [backup_file_used_for_restore]
FROM master.sys.databases [d]
    LEFT OUTER JOIN msdb.dbo.[restorehistory] r ON r.[destination_database_name] = d.Name
    INNER JOIN msdb.dbo.backupset [bs] ON [r].[backup_set_id] = [bs].[backup_set_id]
    INNER JOIN msdb.dbo.backupmediafamily bmf ON [bs].[media_set_id] = [bmf].[media_set_id]
ORDER BY [d].[name], [r].restore_date DESC

```

Encuentra cada mención de un campo en la base de datos.

```

SELECT DISTINCT
    o.name AS Object_Name, o.type_desc

```

```
FROM sys.sql_modules m
      INNER JOIN sys.objects o ON m.object_id=o.object_id
WHERE m.definition Like '%myField%'
ORDER BY 2,1
```

Encontrará menciones de `myField` en SProcs, Vistas, etc.

Lea **Recuperar información sobre la base de datos. en línea:** <https://riptutorial.com/es/sql-server/topic/697/recuperar-informacion-sobre-la-base-de-datos->

Capítulo 90: SCOPE_IDENTITY ()

Sintaxis

- SELECCIONA SCOPE_IDENTITY ();
- SELECCIONE SCOPE_IDENTITY () COMO [SCOPE_IDENTITY];
- SCOPE_IDENTITY ()

Examples

Introducción con ejemplo simple

SCOPE_IDENTITY () devuelve el último valor de identidad insertado en una columna de identidad en el mismo ámbito. Un ámbito es un módulo: un procedimiento almacenado, un activador, una función o un lote. Por lo tanto, dos declaraciones están en el mismo ámbito si están en el mismo procedimiento almacenado, función o lote.

```
Insertar en ([columna 1], [columna 2]) VALORES (8,9);  
IR  
SELECCIONE SCOPE_IDENTITY () COMO [SCOPE_IDENTITY];  
IR
```

Lea SCOPE_IDENTITY () en línea: <https://riptutorial.com/es/sql-server/topic/5326/scope-identity--->

Capítulo 91: Secuencias

Examples

Crear secuencia

```
CREATE SEQUENCE [dbo].[CustomersSeq]
AS INT
START WITH 10001
INCREMENT BY 1
MINVALUE -1;
```

Usa la secuencia en la tabla

```
CREATE TABLE [dbo].[Customers]
(
    CustomerID INT DEFAULT (NEXT VALUE FOR [dbo].[CustomersSeq]) NOT NULL,
    CustomerName VARCHAR(100),
);
```

Insertar en la mesa con secuencia

```
INSERT INTO [dbo].[Customers]
    ([CustomerName])
VALUES
    ('Jerry'),
    ('Gorge')

SELECT * FROM [dbo].[Customers]
```

Resultados

Identificación del cliente	Nombre del cliente
10001	alemán
10002	Garganta

Eliminar de e insertar nuevo

```
DELETE FROM [dbo].[Customers]
WHERE CustomerName = 'Gorge';

INSERT INTO [dbo].[Customers]
    ([CustomerName])
VALUES ('George')

SELECT * FROM [dbo].[Customers]
```

Resultados

Identificación del cliente	Nombre del cliente
10001	alemán
10003	Jorge

Lea Secuencias en línea: <https://riptutorial.com/es/sql-server/topic/5324/secuencias>

Capítulo 92: Seguridad a nivel de fila

Examples

Predicado del filtro RLS

La base de datos de Sql Server 2016+ y Azure Sql le permite filtrar automáticamente las filas que se devuelven en la declaración de selección utilizando algún predicado. Esta característica se llama **seguridad de nivel de fila** .

Primero, necesita una función con valores de tabla que contenga algún predicado que describa cuál es la condición que permitirá a los usuarios leer datos de alguna tabla:

```
DROP FUNCTION IF EXISTS dbo.pUserCanAccessCompany
GO
CREATE FUNCTION

dbo.pUserCanAccessCompany(@CompanyID int)

    RETURNS TABLE
    WITH SCHEMABINDING
AS RETURN (
    SELECT 1 as canAccess WHERE

    CAST (SESSION_CONTEXT (N'CompanyID') as int) = @CompanyID

)
```

En este ejemplo, el predicado dice que solo los usuarios que tienen un valor en `SESSION_CONTEXT` que coincide con el argumento de entrada pueden acceder a la empresa. Puede poner cualquier otra condición, por ejemplo, que verifique el rol de la base de datos o el ID de base de datos del usuario actual, etc.

La mayoría del código anterior es una plantilla que copiará y pegará. Lo único que cambiará aquí es el nombre y los argumentos del predicado y la condición en la cláusula `WHERE`. Ahora crea una política de seguridad que aplicará este predicado en alguna tabla.

Ahora puede crear una política de seguridad que aplicará predicado en alguna tabla:

```
CREATE SECURITY POLICY dbo.CompanyAccessPolicy
    ADD FILTER PREDICATE dbo.pUserCanAccessCompany (CompanyID) ON dbo.Company
    WITH (State=ON)
```

Esta política de seguridad asigna predicado a la tabla de la empresa. Cada vez que alguien intente leer datos de la tabla de la Compañía, la política de seguridad aplicará el predicado en cada fila, pasará la columna `CompanyID` como parámetro del predicado, y el predicado evaluará si esta fila se devuelve en el resultado de la consulta `SELECT`.

Alterar la política de seguridad RLS

La política de seguridad es un grupo de predicados asociados a tablas que pueden administrarse juntas. Puede agregar, eliminar predicados o activar / desactivar toda la política.

Puede agregar más predicados en las tablas en la política de seguridad existente.

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy
    ADD FILTER PREDICATE dbo.pUserCanAccessCompany (CompanyID) ON dbo.Company
```

Puedes eliminar algunos predicados de la política de seguridad:

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy
    DROP FILTER PREDICATE ON dbo.Company
```

Puede deshabilitar la política de seguridad

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy WITH ( STATE = OFF );
```

Puede habilitar la política de seguridad que fue deshabilitada:

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy WITH ( STATE = ON );
```

Previendo actualización usando predicado de bloque RLS

La seguridad a nivel de fila le permite definir algunos predicados que controlarán quién podría actualizar las filas en la tabla. Primero debe definir alguna función de valor de tabla que represente el predicado que controlará la política de acceso.

CREAR FUNCION

dbo.pUserCanAccessProduct (@CompanyID int)

```
RETURNS TABLE
WITH SCHEMABINDING
```

COMO VOLVER (SELECCIONE 1 como puedeAccesar DONDE

CAST (SESSION_CONTEXT (N'CompanyID ') como int) = @CompanyID

) En este ejemplo, el predicado dice que solo los usuarios que tienen un valor en SESSION_CONTEXT que coincide con el argumento de entrada pueden acceder a la empresa. Puede poner cualquier otra condición, por ejemplo, que verifique el rol de la base de datos o el ID de base de datos del usuario actual, etc.

La mayoría del código anterior es una plantilla que copiará y pegará. Lo único que cambiará aquí es el nombre y los argumentos del predicado y la condición en la cláusula WHERE. Ahora crea una política de seguridad que aplicará este predicado en

alguna tabla.

Ahora podemos crear una política de seguridad con el predicado que bloqueará las actualizaciones en la tabla de productos si la columna CompanyID en la tabla no satisface el predicado.

```
CREAR LA POLÍTICA DE SEGURIDAD dbo.ProductAccessPolicy AGREGAR PREDICACIÓN DE BLOQUE dbo.pUserCanAccessProduct (CompanyID) EN dbo.Product
```

Este predicado se aplicará en todas las operaciones. Si desea aplicar predicado en alguna operación, puede escribir algo como:

```
CREAR LA POLÍTICA DE SEGURIDAD dbo.ProductAccessPolicy AGREGAR BLOQUEAR PREDICAR dbo.pUserCanAccessProduct (CompanyID) EN dbo.Product DESPUÉS DE INSERTAR
```

Las opciones posibles que puede agregar después de la definición del predicado de bloque son:

```
[{AFTER {INSERT | ACTUALIZACIÓN}}  
| {ANTES de {ACTUALIZACIÓN | BORRAR } } ]
```

Lea Seguridad a nivel de fila en línea: <https://riptutorial.com/es/sql-server/topic/7045/seguridad-a-nivel-de-fila>

Capítulo 93: Si ... otra cosa

Examples

Declaración única de FI

Como la mayoría de los otros lenguajes de programación, T-SQL también admite sentencias `IF..ELSE`.

Por ejemplo, en el siguiente ejemplo, `1 = 1` es la expresión, que se evalúa como Verdadero y el control ingresa en el bloque `BEGIN..END` y la instrucción `Imprimir` imprime la cadena `'One is equal to One'`

```
IF ( 1 = 1)  --<-- Some Expression
BEGIN
    PRINT 'One is equal to One'
END
```

Múltiples declaraciones de IF

Podemos usar múltiples declaraciones `IF` para verificar múltiples expresiones totalmente independientes unas de otras.

En el siguiente ejemplo, se evalúa la expresión de cada instrucción `IF` y, si es verdadera, se ejecuta el código dentro del bloque `BEGIN...END`. En este ejemplo en particular, las expresiones Primera y Tercera son verdaderas y solo se ejecutarán esas declaraciones impresas.

```
IF (1 = 1)  --<-- Some Expression      --<-- This is true
BEGIN
    PRINT 'First IF is True'           --<-- this will be executed
END

IF (1 = 2)  --<-- Some Expression
BEGIN
    PRINT 'Second IF is True'
END

IF (3 = 3)  --<-- Some Expression      --<-- This true
BEGIN
    PRINT 'Thrid IF is True'           --<-- this will be executed
END
```

Una sola declaración `IF..ELSE`

En una sola instrucción `IF..ELSE`, si la expresión se evalúa como Verdadero en la instrucción `IF`, el control ingresa en el primer bloque `BEGIN..END` y solo se ejecuta el código dentro de ese bloque, el bloque `Else` simplemente se ignora.

Por otro lado, si la expresión se evalúa como `False` el bloque `ELSE BEGIN..END` se ejecuta y el

control nunca ingresa en el primer bloque `BEGIN..END` .

En el siguiente ejemplo, la expresión se evaluará como falsa y el bloque Else se ejecutará imprimiendo la cadena `'First expression was not true'`

```
IF ( 1 <> 1)  --<-- Some Expression
BEGIN
    PRINT 'One is equal to One'
END
ELSE
BEGIN
    PRINT 'First expression was not true'
END
```

Múltiples IF ... ELSE con declaraciones ELSE finales

Si tenemos varias declaraciones `IF...ELSE IF` , pero también queremos ejecutar algún fragmento de código si ninguna de las expresiones se evalúa como Verdadero, entonces podemos simplemente agregar un bloque `ELSE` final que solo se ejecuta si ninguna de las `IF` o `ELSE IF` expresiones `ELSE IF` se evalúan como verdaderas.

En el siguiente ejemplo, ninguna de las expresiones `IF` o `ELSE IF` son verdaderas, por lo que solo se ejecuta el bloque `ELSE` y se imprime `'No other expression is true'`

```
IF ( 1 = 1 + 1 )
    BEGIN
        PRINT 'First If Condition'
    END
ELSE IF ( 1 = 2 )
    BEGIN
        PRINT 'Second If Else Block'
    END
ELSE IF ( 1 = 3 )
    BEGIN
        PRINT 'Third If Else Block'
    END
ELSE
    BEGIN
        PRINT 'No other expression is true'  --<-- Only this statement will be printed
    END
```

Múltiples declaraciones de IF ... ELSE

En la mayoría de los casos, necesitamos verificar varias expresiones y realizar acciones específicas basadas en esas expresiones. Esta situación se maneja utilizando varias declaraciones `IF...ELSE IF` .

En este ejemplo, todas las expresiones se evalúan de arriba a abajo. Tan pronto como una expresión se evalúa como verdadera, se ejecuta el código dentro de ese bloque. Si ninguna expresión se evalúa como verdadera, nada se ejecuta.

```
IF ( 1 = 1 + 1 )
```

```
BEGIN
    PRINT 'First If Condition'
END
ELSE IF (1 = 2)
BEGIN
    PRINT 'Second If Else Block'
END
ELSE IF (1 = 3)
BEGIN
    PRINT 'Third If Else Block'
END
ELSE IF (1 = 1)      --<-- This is True
BEGIN
    PRINT 'Last Else Block'  --<-- Only this statement will be printed
END
```

Lea Si ... otra cosa en línea: <https://riptutorial.com/es/sql-server/topic/5186/si----otra-cosa>

Capítulo 94: Sobre la cláusula

Parámetros

Parámetro	Detalles
PARTICIÓN POR	El (los) campo (s) que sigue a PARTICIÓN POR es el que se basará en la "agrupación"

Observaciones

La cláusula OVER determina una ventana o un subconjunto de fila dentro de un conjunto de resultados de consulta. Se puede aplicar una función de ventana para establecer y calcular un valor para cada fila en el conjunto. La cláusula OVER se puede utilizar con:

- Funciones de clasificación
- Funciones agregadas

por lo tanto, alguien puede calcular valores agregados, como promedios móviles, agregados acumulados, totales acumulados o una N superior por resultados de grupo.

De una manera muy abstracta podemos decir que OVER se comporta como GROUP BY. Sin embargo, OVER se aplica por campo / columna y no a la consulta en su totalidad como lo hace GROUP BY.

Nota # 1: En SQL Server 2008 (R2) la cláusula ORDER BY no se puede usar con funciones de ventana agregadas ([enlace](#)).

Examples

Usando las funciones de agregación con OVER

Usando la [tabla de autos](#) , calcularemos la cantidad total, máxima, mínima y promedio de dinero que gastó cada cliente y muchas veces (COUNT) que ella trajo un auto para reparación.

Id CustomerId MechanicId Modelo Estado Costo Total

```
SELECT CustomerId,
       SUM(TotalCost) OVER(PARTITION BY CustomerId) AS Total,
       AVG(TotalCost) OVER(PARTITION BY CustomerId) AS Avg,
       COUNT(TotalCost) OVER(PARTITION BY CustomerId) AS Count,
       MIN(TotalCost) OVER(PARTITION BY CustomerId) AS Min,
       MAX(TotalCost) OVER(PARTITION BY CustomerId) AS Max
FROM CarsTable
WHERE Status = 'READY'
```

Tenga en cuenta que usar OVER de esta manera no agregará las filas devueltas. La consulta anterior devolverá lo siguiente:

Identificación del cliente	Total	Avg	Contar	Min	Max
1	430	215	2	200	230
1	430	215	2	200	230

La fila (s) duplicada (s) puede no ser tan útil para propósitos de reporte.

Si desea simplemente agregar datos, será mejor que use la cláusula GROUP BY junto con las funciones agregadas apropiadas, por ejemplo:

```
SELECT CustomerId,
       SUM(TotalCost) AS Total,
       AVG(TotalCost) AS Avg,
       COUNT(TotalCost) AS Count,
       MIN(TotalCost) AS Min,
       MAX(TotalCost) AS Max
FROM CarsTable
WHERE Status = 'READY'
GROUP BY CustomerId
```

Suma acumulativa

Usando la [Tabla de ventas de artículos](#) , intentaremos descubrir cómo aumentan las ventas de nuestros artículos a través de las fechas. Para ello, calcularemos la *suma acumulada* de las ventas totales por pedido de artículo antes de la fecha de venta.

```
SELECT item_id, sale_Date
       SUM(quantity * price) OVER(PARTITION BY item_id ORDER BY sale_Date ROWS BETWEEN
UNBOUNDED PRECEDING) AS SalesTotal
FROM SalesTable
```

Usando funciones de agregación para encontrar los registros más recientes

Usando la [base de datos de la biblioteca](#) , tratamos de encontrar el último libro agregado a la base de datos para cada autor. Para este ejemplo simple, asumimos una identificación siempre creciente para cada registro agregado.

```
SELECT MostRecentBook.Name, MostRecentBook.Title
FROM ( SELECT Authors.Name,
       Books.Title,
       RANK() OVER (PARTITION BY Authors.Id ORDER BY Books.Id DESC) AS NewestRank
FROM Authors
JOIN Books ON Books.AuthorId = Authors.Id
) MostRecentBook
WHERE MostRecentBook.NewestRank = 1
```

En lugar de RANK, se pueden usar otras dos funciones para ordenar. En el ejemplo anterior, el

resultado será el mismo, pero darán resultados diferentes cuando el pedido dé múltiples filas para cada rango.

- `RANK()` : los duplicados obtienen el mismo rango, el siguiente rango toma en cuenta el número de duplicados en el rango anterior
- `DENSE_RANK()` : los duplicados obtienen el mismo rango, el siguiente rango es siempre uno más alto que el anterior
- `ROW_NUMBER()` : dará a cada fila un 'rango' único, 'clasificando' los duplicados al azar

Por ejemplo, si la tabla tenía una columna `CreationDate` no única y el orden se realizó en base a eso, la siguiente consulta:

```
SELECT Authors.Name,
       Books.Title,
       Books.CreationDate,
       RANK() OVER (PARTITION BY Authors.Id ORDER BY Books.CreationDate DESC) AS RANK,
       DENSE_RANK() OVER (PARTITION BY Authors.Id ORDER BY Books.CreationDate DESC) AS
DENSE_RANK,
       ROW_NUMBER() OVER (PARTITION BY Authors.Id ORDER BY Books.CreationDate DESC) AS
ROW_NUMBER,
FROM Authors
JOIN Books ON Books.AuthorId = Authors.Id
```

Podría resultar en:

Autor	Título	Fecha de creación	RANGO	DENSE_RANK	NUMERO DE FILA
Autor 1	Libro 1	22/07/2016	1	1	1
Autor 1	Libro 2	22/07/2016	1	1	2
Autor 1	Libro 3	21/07/2016	3	2	3
Autor 1	Libro 4	21/07/2016	3	2	4
Autor 1	Libro 5	21/07/2016	3	2	5
Autor 1	Libro 6	07/04/2016	6	3	6
Autor 2	Libro 7	07/04/2016	1	1	1

Dividir datos en cubos igualmente particionados usando NTILE

Digamos que tiene puntajes de examen para varios exámenes y quiere dividirlos en cuartiles por examen.

```
-- Setup data:
declare @values table(Id int identity(1,1) primary key, [Value] float, ExamId int)
insert into @values ([Value], ExamId) values
(65, 1), (40, 1), (99, 1), (100, 1), (90, 1), -- Exam 1 Scores
(91, 2), (88, 2), (83, 2), (91, 2), (78, 2), (67, 2), (77, 2) -- Exam 2 Scores
```

```
-- Separate into four buckets per exam:
select ExamId,
       ntile(4) over (partition by ExamId order by [Value] desc) as Quartile,
       Value, Id
from @values
order by ExamId, Quartile
```

	ExamId	Quartile	Value	Id
1	1	1	100	4
2	1	1	99	3
3	1	2	90	5
4	1	3	65	1
5	1	4	40	2
6	2	1	91	9
7	2	1	91	6
8	2	2	88	7
9	2	2	83	8
10	2	3	78	10
11	2	3	77	12
12	2	4	67	11

`ntile` funciona muy bien cuando realmente necesita un número determinado de cubos y cada uno se llena aproximadamente al mismo nivel. Tenga en cuenta que sería trivial separar estas puntuaciones en percentiles simplemente usando `ntile(100)` .

Lea Sobre la cláusula en línea: <https://riptutorial.com/es/sql-server/topic/353/sobre-la-clausula>

Capítulo 95: SQL dinámico

Examples

Ejecutar la instrucción SQL proporcionada como una cadena

En algunos casos, necesitaría ejecutar una consulta SQL colocada en una cadena. EXEC, EXECUTE o el procedimiento del sistema sp_executesql puede ejecutar cualquier consulta SQL proporcionada como cadena:

```
sp_executesql N'SELECT * FROM sys.objects'
-- or
sp_executesql @stmt = N'SELECT * FROM sys.objects'
-- or
EXEC sp_executesql N'SELECT * FROM sys.objects'
-- or
EXEC('SELECT * FROM sys.columns')
-- or
EXECUTE('SELECT * FROM sys.tables')
```

Este procedimiento devolverá el mismo conjunto de resultados que la consulta SQL proporcionada como texto de declaración. sp_executesql puede ejecutar la consulta SQL proporcionada como literal de cadena, variable / parámetro, o incluso expresión:

```
declare @table nvarchar(40) = N'product items'
EXEC(N'SELECT * FROM ' + @table)
declare @sql nvarchar(40) = N'SELECT * FROM ' + QUOTENAME(@table);
EXEC sp_executesql @sql
```

Necesitas la función QUOTENAME para escapar de los caracteres especiales en la variable @table. Sin esta función, obtendría un error de sintaxis si la variable @table contiene algo como espacios, corchetes o cualquier otro carácter especial.

SQL dinámico ejecutado como usuario diferente

Puede ejecutar la consulta SQL como usuario diferente utilizando AS USER = 'nombre del usuario de la base de datos'

```
EXEC(N'SELECT * FROM product') AS USER = 'dbo'
```

La consulta SQL se ejecutará bajo el usuario de la base de datos dbo. Todas las verificaciones de permisos aplicables al usuario dbo se verifican en la consulta SQL.

Inyección de SQL con SQL dinámico

Las consultas dinámicas son


```
SET @sql = N'SELECT COUNT(*) FROM AppUsers WHERE Username = ''' + @user + ''' AND Password = ''' + @pass + ''''
EXEC (@sql)
```

Si el valor de la variable de usuario es mi nombre de **usuario " OR 1 = 1** , se ejecutará la siguiente consulta:

```
SELECT COUNT(*)
FROM AppUsers
WHERE Username = 'myusername' OR 1=1 --' AND Password = ''
```

Los comentarios al final del valor de la variable @nombre de usuario comentarán una parte final de la consulta y se evaluará la condición $1 = 1$. La aplicación que lo compruebe allí, al menos un usuario devuelto por esta consulta devolverá un conteo mayor que 0 y el inicio de sesión será exitoso.

Usando este método, el atacante puede iniciar sesión en la aplicación incluso si no conoce un nombre de usuario y contraseña válidos.

SQL dinámico con parámetros

Para evitar problemas de inyección y escape, las consultas de SQL dinámico deben ejecutarse con parámetros, por ejemplo:

```
SET @sql = N'SELECT COUNT(*) FROM AppUsers WHERE Username = @user AND Password = @pass
EXEC sp_executesql @sql, '@user nvarchar(50), @pass nvarchar(50)', @username, @password
```

El segundo parámetro es una lista de parámetros utilizados en la consulta con sus tipos, después de esta lista se proporcionan las variables que se utilizarán como valores de parámetros.

sp_executesql escapará de los caracteres especiales y ejecutará la consulta SQL.

Lea SQL dinámico en línea: <https://riptutorial.com/es/sql-server/topic/6871/sql-dinamico>

Capítulo 96: SQL Server Evolution a través de diferentes versiones (2000 - 2016)

Introducción

Estoy usando SQL Server desde 2004. Comencé con 2000 y ahora usaré SQL Server 2016. Creé tablas, vistas, funciones, activadores, procedimientos almacenados y escribí muchas consultas SQL, pero no usé muchas características nuevas de las siguientes versiones. Busqué en Google pero desafortunadamente no encontré todas las características en un solo lugar. Así que reuní y validé esta información de diferentes fuentes y la puse aquí. Solo estoy agregando la información de alto nivel para todas las versiones desde el 2000 hasta el 20

Examples

SQL Server versión 2000 - 2016

Las siguientes características agregadas en SQL Server 2000 de su versión anterior:

1. Se agregaron nuevos tipos de datos (BIGINT, SQL_VARIANT, TABLE)
2. En lugar de y para Triggers se introdujeron como avance a la DDL.
3. La integridad referencial en cascada.
4. Soporte XML
5. Funciones definidas por el usuario y vistas de partición.
6. Vistas indizadas (Permitir índice en vistas con columnas computadas).

Las siguientes características añadidas en la versión 2005 de su versión anterior:

1. Mejora en la cláusula TOP con la opción "WITH TIES".
2. Comandos de manipulación de datos (DML) y cláusula de SALIDA para obtener valores INSERTADOS y BORRADOS
3. Los operadores PIVOT y UNPIVOT.
4. Manejo de excepciones con bloque TRY / CATCH
5. Funciones de clasificación
6. Expresiones de tabla comunes (CTE)
7. Common Language Runtime (integración de lenguajes .NET para crear objetos como procedimientos almacenados, activadores, funciones, etc.)
8. Service Broker (Manejando el mensaje entre un remitente y un receptor de una manera libremente acoplada)
9. Cifrado de datos (capacidades nativas para admitir el cifrado de datos almacenados en bases de datos definidas por el usuario)
10. Correo SMTP
11. Puntos finales HTTP (Creación de puntos finales mediante una simple declaración T-SQL que expone un objeto para acceder a través de Internet)
12. Conjuntos de resultados activos múltiples (MARS). Esto permite que una conexión de base

de datos persistente de un solo cliente tenga más de una solicitud activa por conexión.

13. SQL Server Integration Services (se usará como una herramienta primaria de ETL (extracción, transformación y carga)
14. Mejoras en Analysis Services y Reporting Services.
15. Tablas e índices de particionamiento. Permite la partición de tablas e índices en función de los límites de partición según lo especificado por una FUNCIÓN DE PARTICIÓN con particiones individuales asignadas a grupos de archivos a través de un ESQUEMA DE PARTICIÓN.

Las siguientes características añadidas en la versión 2008 de su versión anterior:

1. Mejora en los tipos de datos existentes de DATE y TIME
2. Nuevas funciones como - SYSUTCDATETIME () y SYSDATETIMEOFFSET ()
3. Columnas de repuesto: para ahorrar una cantidad significativa de espacio en disco.
4. Tipos grandes definidos por el usuario (hasta 2 GB de tamaño)
5. Se introdujo una nueva función para pasar un tipo de datos de tabla a procedimientos y funciones almacenados
6. Nuevo comando MERGE para las operaciones INSERT, UPDATE y DELETE
7. Nuevo tipo de datos HierarchyID
8. Tipos de datos espaciales: para representar la ubicación física y la forma de cualquier objeto geométrico.
9. Consultas e informes más rápidos con GROUPING SETS: una extensión de la cláusula GROUP BY.
10. Mejora a la opción de almacenamiento de FILESTREAM

Las siguientes características agregadas en la versión 2008 R2 de su versión anterior:

1. PowerPivot - Para procesar grandes conjuntos de datos.
2. Generador de informes 3.0
3. Listo para la nube
4. StreamInsight
5. Servicios de datos maestros
6. Integración de SharePoint
7. DACPAC (paquetes de componentes de aplicación de nivel de datos)
8. Mejora en otras características de SQL Server 2008

Las siguientes características añadidas en la versión 2012 de su versión anterior:

1. Índices de almacén de columnas: reduce la E / S y la utilización de la memoria en consultas grandes.
2. Paginación: la paginación se puede hacer usando los comandos "OFFSET" y "FETCH".
3. Base de datos contenida - Gran característica para migraciones periódicas de datos.
4. Grupos de disponibilidad AlwaysOn
5. Soporte de Windows Server Core
6. Funciones de servidor definidas por el usuario
7. Soporte de Big Data
8. PowerView
9. Mejoras de SQL Azure

10. Modelo Tabular (SSAS)
11. Servicios de calidad de datos DQS
12. Tabla de archivos: una mejora de la función FILESTREAM que se introdujo en 2008.
13. Mejora en el manejo de errores, incluida la sentencia THROW
14. Mejora a la depuración de SQL Server Management Studio a. SQL Server 2012 introduce más opciones para controlar los puntos de interrupción. segundo. Mejoras a las ventanas en modo debug
do. Mejora en IntelliSense - como insertar fragmentos de código.

Las siguientes características añadidas en la versión 2014 de su versión anterior:

1. Motor OLTP en memoria: mejora el rendimiento hasta 20 veces.
2. Mejoras de AlwaysOn
3. Extensión de la piscina de búfer
4. Características de la nube híbrida
5. Mejora en los índices del almacén de columnas (como los índices del almacén de columnas actualizables)
6. Mejoras en el manejo de consultas (como SELECT INTO en paralelo)
7. Power BI para la integración de Office 365
8. Durabilidad retardada
9. Mejoras para copias de seguridad de bases de datos

Las siguientes características añadidas en la versión 2016 de su versión anterior:

1. Siempre encriptado: Always Encrypted está diseñado para proteger datos en reposo o en movimiento.
2. Analítica operacional en tiempo real
3. PolyBase en SQL Server
4. Soporte nativo de JSON
5. Tienda de consultas
6. Mejoras a AlwaysOn
7. OLTP en memoria mejorada
8. Varios archivos de base de datos TempDB
9. Estirar la base de datos
10. Seguridad a nivel de fila
11. Mejoras en la memoria

Mejoras de T-SQL o nuevas adiciones en SQL Server 2016

1. TABLA DE TRUNCATOS CON PARTICIÓN
2. GOTA SI EXISTE
3. Funciones STRING_SPLIT y STRING_ESCAPE
4. ALTER TABLE ahora puede alterar muchas columnas mientras la tabla permanece en línea, usando WITH (ONLINE = ON | OFF).
5. MAXDOP para DBCC CHECKDB, DBCC CHECKTABLE y DBCC CHECKFILEGROUP

6. ALTER DATABASE SET AUTOGROW_SINGLE_FILE

7. ALTER DATABASE SET AUTOGROW_ALL_FILES

8. Funciones COMPRESS y DECOMPRESS

9. Declaración de FORMATO

10. 2016 introduce 8 propiedades más con SERVERPROPERTY

a. InstanceDefaultDataPath

segundo. InstanceDefaultLogPath

do. ProductBuild

re. ProductBuildType

mi. ProductMajorVersion

F. ProductMinorVersion

sol. ProductUpdateLevel

h. ProductUpdateReference

Lea [SQL Server Evolution a través de diferentes versiones \(2000 - 2016\)](https://riptutorial.com/es/sql-server/topic/10129/sql-server-evolution-a-traves-de-diferentes-versiones--2000---2016-) en línea:
<https://riptutorial.com/es/sql-server/topic/10129/sql-server-evolution-a-traves-de-diferentes-versiones--2000---2016->

Capítulo 97: SQL Server Management Studio (SSMS)

Introducción

SQL Server Management Studio (SSMS) es una herramienta para administrar y administrar SQL Server y SQL Database.

Microsoft ofrece SSMS de forma gratuita.

[La documentación de SSMS](#) está disponible.

Examples

Actualizar el caché de IntelliSense

Cuando los objetos se crean o modifican, no están disponibles automáticamente para IntelliSense. Para que estén disponibles para IntelliSense, la memoria caché local debe actualizarse.

Dentro de una ventana del editor de consultas, presione `Ctrl + Shift + R` o seleccione `Edit | IntelliSense | Refresh Local Cache` desde el menú.

Después de esto, todos los cambios desde la última actualización estarán disponibles para IntelliSense.

Lea [SQL Server Management Studio \(SSMS\) en línea](https://riptutorial.com/es/sql-server/topic/10642/sql-server-management-studio--ssms-): <https://riptutorial.com/es/sql-server/topic/10642/sql-server-management-studio--ssms->

Capítulo 98: SQLCMD

Observaciones

O bien debe estar en la ruta donde existe SQLCMD.exe o agregarlo a su variable de entorno PATH.

Examples

SQLCMD.exe llamado desde un archivo por lotes o línea de comandos

```
echo off

cls

sqlcmd.exe -S "your server name" -U "sql user name" -P "sql password" -d "name of databse" -Q
"here you may write your query/stored procedure"
```

Los archivos por lotes como estos se pueden usar para automatizar tareas, por ejemplo, para hacer copias de seguridad de bases de datos a una hora específica (se pueden programar con el Programador de tareas) para una versión de SQL Server Express donde no se pueden usar los trabajos de agente.

Lea SQLCMD en línea: <https://riptutorial.com/es/sql-server/topic/5396/sqlcmd>

Capítulo 99: Subconsultas

Examples

Subconsultas

Una subconsulta es una consulta dentro de otra consulta SQL. Una subconsulta también se denomina consulta interna o selección interna y la declaración que contiene una subconsulta se denomina consulta externa o selección externa.

Nota

1. Las subconsultas deben estar entre paréntesis,
2. Un ORDER BY no se puede utilizar en una subconsulta.
3. El tipo de imagen, como BLOB, matriz, tipos de datos de texto, no están permitidos en subconsultas.

Las subconsultas se pueden utilizar con declaraciones de selección, inserción, actualización y eliminación dentro de dónde, desde, cláusula de selección junto con IN, operadores de comparación, etc.

Tenemos una tabla llamada ITCompanyInNepal en la que realizaremos consultas para mostrar ejemplos de subconsultas:

ID	CompanyName	CompanyAddress	Headquarter	NumberOfEmployee
1	CompanyOne	Kathmandu	USA	350
2	CompanyTwo	Kathmandu	USA	310
3	CompanyThree	Kathmandu	Nepal	300
4	CompanyFour	Kathmandu	Nepal	180
5	CompanyFive	Birgunj	Denmark	150
6	CompanySix	Janakpur	USA	100
7	CompanySeven	Janakpur	Australia	100
8	CompanyEight	Birganj	Australia	150
9	CompanyNine	Biratnagar	Canada	200
10	CompanyTen	Pokhara	India	85

Ejemplos: Subconsultas con declaración selecta

con el operador **In** y la cláusula **where** :

```
SELECT *
FROM ITCompanyInNepal
WHERE Headquarter IN (SELECT Headquarter
                      FROM ITCompanyInNepal
                      WHERE Headquarter = 'USA');
```

con **operador de comparación** y **donde** la cláusula


```
SELECT *
FROM ITCompanyInNepal
WHERE NumberOfEmployee < (SELECT AVG(NumberOfEmployee)
                           FROM ITCompanyInNepal
                           )
```

con cláusula **selecta**

```
SELECT  CompanyName,
        CompanyAddress,
        Headquarter,
        (Select SUM(NumberOfEmployee)
         FROM ITCompanyInNepal
         Where Headquarter = 'USA') AS TotalEmployeeHiredByUSAInKathmandu
FROM    ITCompanyInNepal
WHERE   CompanyAddress = 'Kathmandu' AND Headquarter = 'USA'
```

Subconsultas con declaración de inserción

Tenemos que insertar datos de la tabla IndianCompany en ITCompanyInNepal. La tabla para IndianCompany se muestra a continuación:

ID	CompanyName	CompanyAddress	Headquarter	NumberOfEmployee
1	CompanyA	Banglore	USA	450
2	CompanyB	Banglore	USA	500
3	CompanyC	Hyderabad	Denmark	480
4	CompanyD	Hyderabad	Australia	780
5	CompanyE	Delhi	Canada	790

```
INSERT INTO ITCompanyInNepal
SELECT *
FROM IndianCompany
```

Subconsultas con estado de actualización

Supongamos que todas las compañías cuya sede central es EE. UU. Decidieron despedir a 50 empleados de todas las compañías de Nepal basadas en EE. UU. Debido a algún cambio en la política de las compañías estadounidenses.

```
UPDATE ITCompanyInNepal
SET NumberOfEmployee = NumberOfEmployee - 50
WHERE Headquarter IN (SELECT Headquarter
                     FROM ITCompanyInNepal
                     WHERE Headquarter = 'USA')
```

Subconsultas con declaración de borrado

Supongamos que todas las empresas cuya sede central es Dinamarca decidieron cerrar sus empresas de Nepal.

```
DELETE FROM ITCompanyInNepal
WHERE Headquarter IN (SELECT Headquarter
```

```
FROM ITCompanyInNepal  
WHERE Headquarter = 'Denmark')
```

Lea Subconsultas en línea: <https://riptutorial.com/es/sql-server/topic/5629/subconsultas>

Capítulo 100: Tablas temporales

Observaciones

SQL Server 2016 presenta la compatibilidad con las tablas temporales con versión del sistema como una función de base de datos que brinda compatibilidad integrada para proporcionar información sobre los datos almacenados en la tabla en cualquier momento, en lugar de solo los datos que son correctos en el momento actual.

Una tabla temporal con versión del sistema es un nuevo tipo de tabla de usuario en SQL Server 2016, diseñada para mantener un historial completo de cambios en los datos y permitir un análisis fácil en el momento. Este tipo de tabla temporal se conoce como una tabla temporal con versión del sistema porque el sistema administra el período de validez de cada fila (es decir, el motor de la base de datos). Cada tabla temporal tiene dos columnas definidas explícitamente, cada una con un tipo de datos `datetime2`. Estas columnas se conocen como columnas de período. Estas columnas de período son utilizadas exclusivamente por el sistema para registrar el período de validez de cada fila siempre que se modifique una fila.

Examples

Crear tablas temporales

```
CREATE TABLE dbo.Employee
(
    [EmployeeID] int NOT NULL PRIMARY KEY CLUSTERED
    , [Name] nvarchar(100) NOT NULL
    , [Position] varchar(100) NOT NULL
    , [Department] varchar(100) NOT NULL
    , [Address] nvarchar(1024) NOT NULL
    , [AnnualSalary] decimal (10,2) NOT NULL
    , [ValidFrom] datetime2 (2) GENERATED ALWAYS AS ROW START
    , [ValidTo] datetime2 (2) GENERATED ALWAYS AS ROW END
    , PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo)
)
WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.EmployeeHistory));
```

INSERTOS: en un **INSERT** , el sistema establece el valor de la columna **ValidFrom** a la hora de inicio de la transacción actual (en la zona horaria UTC) según el reloj del sistema y asigna el valor de la columna **ValidTo** al valor máximo de 9999- 12-31. Esto marca la fila como abierta.

ACTUALIZACIONES: en una **ACTUALIZACIÓN** , el sistema almacena el valor anterior de la fila en la tabla de historial y establece el valor de la columna **ValidTo** en el momento de inicio de la transacción actual (en la zona horaria UTC) según el reloj del sistema. Esto marca la fila como cerrada, con un período registrado para el cual la fila era válida. En la tabla actual, la fila se actualiza con su nuevo valor y el sistema establece el valor de la columna **ValidFrom** a la hora de inicio de la transacción (en la zona horaria UTC) según el reloj del sistema. El valor de la fila actualizada en la tabla actual para la columna **ValidTo** sigue siendo el valor máximo de 9999-12-

31.

ELIMINAR : en **BORRAR** , el sistema almacena el valor anterior de la fila en la tabla de historial y establece el valor de la columna **ValidTo** en el momento de inicio de la transacción actual (en la zona horaria UTC) según el reloj del sistema. Esto marca la fila como cerrada, con un período registrado para el cual la fila anterior era válida. En la tabla actual, la fila se elimina. Las consultas de la tabla actual no devolverán esta fila. Solo las consultas que tratan con datos históricos devuelven datos para los que se cierra una fila.

MERGE : en un **MERGE** , la operación se comporta exactamente como si se ejecutaran hasta tres sentencias (un **INSERT** , un **UPDATE** y / o un **DELETE**), según lo que se especifique como acciones en la sentencia **MERGE** .

Consejo: Los tiempos registrados en las columnas datetime2 del sistema se basan en la hora de inicio de la transacción. Por ejemplo, todas las filas insertadas dentro de una sola transacción tendrán la misma hora UTC registrada en la columna correspondiente al inicio del período **SYSTEM_TIME** .

¿Cómo puedo consultar los datos temporales?

```
SELECT * FROM Employee
FOR SYSTEM_TIME
BETWEEN '2014-01-01 00:00:00.0000000' AND '2015-01-01 00:00:00.0000000'
WHERE EmployeeID = 1000 ORDER BY ValidFrom;
```

Devuelve el valor real especificado en el tiempo (FOR SYSTEM_TIME AS OF)

Devuelve una tabla con una fila que contiene los valores que eran reales (actuales) en el momento especificado en el pasado.

```
SELECT * FROM Employee
FOR SYSTEM_TIME AS OF '2016-08-06 08:32:37.91'
```

PARA SYSTEM_TIME ENTRE Y

Igual que arriba en la descripción de FOR SYSTEM_TIME FROM <start_date_time> TO <end_date_time>, excepto que la tabla de filas devueltas incluye filas que se activaron en el límite superior definido por el punto final <end_date_time>.

```
SELECT * FROM Employee
FOR SYSTEM_TIME BETWEEN '2015-01-01' AND '2015-12-31'
```

PARA SYSTEM_TIME FROM A

Devuelve una tabla con los valores para todas las versiones de fila que estaban activas dentro del rango de tiempo especificado, independientemente de si comenzaron a estar activas antes del valor del parámetro <start_date_time> para el argumento FROM o dejaron de estar activas después del valor del parámetro <end_date_time> para el A argumento. Internamente, se realiza

una unión entre la tabla temporal y su tabla de historial y los resultados se filtran para devolver los valores de todas las versiones de fila que estaban activas en cualquier momento durante el rango de tiempo especificado. Las filas que se activaron exactamente en el límite inferior definido por el punto final FROM se incluyen y los registros que se activaron exactamente en el límite superior definido por el punto final TO no se incluyen.

```
SELECT * FROM Employee
FOR SYSTEM_TIME FROM '2015-01-01' TO '2015-12-31'
```

PARA SYSTEM_TIME CONTENIDO EN (,)

Devuelve una tabla con los valores para todas las versiones de fila que se abrieron y cerraron dentro del rango de tiempo especificado definido por los dos valores de fecha y hora para el argumento CONTAINED IN. Se incluyen las filas que se activaron exactamente en el límite inferior o dejaron de estar activas exactamente en el límite superior.

```
SELECT * FROM Employee
FOR SYSTEM_TIME CONTAINED IN ('2015-04-01', '2015-09-25')
```

PARA SYSTEM_TIME ALL

Devuelve la unión de filas que pertenecen a la tabla actual y la tabla de historial.

```
SELECT * FROM Employee
FOR SYSTEM_TIME ALL
```

Creación de una tabla temporal con versión de sistema optimizada para la memoria y limpieza de la tabla de historial de SQL Server

Crear una tabla temporal con una tabla de historial predeterminada es una opción conveniente cuando desea controlar los nombres y aún depende del sistema para crear la tabla de historial con la configuración predeterminada. En el siguiente ejemplo, una nueva tabla temporal optimizada con memoria del sistema vinculada a una nueva tabla de historial basada en disco.

```
CREATE SCHEMA History
GO
CREATE TABLE dbo.Department
(
    DepartmentNumber char(10) NOT NULL PRIMARY KEY NONCLUSTERED,
    DepartmentName varchar(50) NOT NULL,
    ManagerID int NULL,
    ParentDepartmentNumber char(10) NULL,
    SysStartTime datetime2 GENERATED ALWAYS AS ROW START HIDDEN NOT NULL,
    SysEndTime datetime2 GENERATED ALWAYS AS ROW END HIDDEN NOT NULL,
    PERIOD FOR SYSTEM_TIME (SysStartTime, SysEndTime)
)
WITH
(
    MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_AND_DATA,
    SYSTEM_VERSIONING = ON ( HISTORY_TABLE = History.DepartmentHistory )
)
```

```
);
```

Limpiando la tabla de historial de SQL Server Con el tiempo, la tabla de historial puede crecer significativamente. Como no se permite insertar, actualizar o eliminar datos de la tabla de historial, la única forma de limpiar la tabla de historial es deshabilitar primero la versión del sistema:

```
ALTER TABLE dbo.Employee
```

```
SET (SYSTEM_VERSIONING = OFF); IR
```

Eliminar datos innecesarios de la tabla de historial:

```
DELETE FROM dbo.EmployeeHistory
```

```
WHERE EndTime <= '2017-01-26 14:00:29';
```

y luego volver a habilitar el control de versiones del sistema:

```
ALTER TABLE dbo.Employee
```

```
SET (SYSTEM_VERSIONING = ON (HISTORY_TABLE = [dbo]. [EmployeeHistory],  
DATA_CONSISTENCY_CHECK = ON));
```

La limpieza de la tabla de historial en las bases de datos de Azure SQL es un poco diferente, ya que las bases de datos de Azure SQL tienen soporte incorporado para la limpieza de la tabla de historial. Primero, la limpieza de la retención del historial temporal debe habilitarse en un nivel de base de datos:

```
ALTER DATABASE CURRENT
```

CONFIGURAR TEMPORAL_HISTORIAL_RETENCIÓN EN IR

A continuación, establezca el período de retención por tabla:

```
ALTER TABLE dbo.Employee
```

```
SET (SYSTEM_VERSIONING = ON (HISTORY_RETENTION_PERIOD = 90 DAYS));
```

Esto eliminará todos los datos en la tabla de historial con más de 90 días. Las bases de datos locales de SQL Server 2016 no son compatibles con `TEMPORAL_HISTORIAL_RETENTION` y `HISTORY_RETENTION_PERIOD` y cualquiera de las dos consultas anteriores se ejecutan en las bases de datos locales de SQL Server 2016, se producirán los siguientes errores.

Para `TEMPORAL_HISTORIAL_RETENTION` el error será:

```
Msg 102, Level 15, State 6, Line 34
```

Sintaxis incorrecta cerca de 'TEMPORAL_HISTORIAL_RETENTION'.

Para HISTORY_RETENTION_PERIOD el error será:

```
Msg 102, Level 15, State 1, Line 39
```

Sintaxis incorrecta cerca de 'HISTORY_RETENTION_PERIOD'.

Lea Tablas temporales en línea: <https://riptutorial.com/es/sql-server/topic/5296/tablas-temporales>

Capítulo 101: Tarea o trabajo programado

Introducción

El Agente SQL Server utiliza SQL Server para almacenar información de trabajo. Los trabajos contienen uno o más pasos de trabajo. Cada paso contiene su propia tarea, es decir: hacer una copia de seguridad de una base de datos. El Agente SQL Server puede ejecutar un trabajo de forma programada, en respuesta a un evento específico o a pedido.

Examples

Crear un trabajo programado

Crear un trabajo

- Para agregar un trabajo primero, tenemos que usar un procedimiento almacenado llamado [sp_add_job](#)

```
USE msdb ;
GO
EXEC dbo.sp_add_job
@job_name = N'Weekly Job' ; -- the job name
```

- Luego tenemos que agregar un paso de trabajo usando un procedimiento almacenado llamado [sp_add_jobstep](#)

```
EXEC sp_add_jobstep
@job_name = N'Weekly Job', -- Job name to add a step
@step_name = N'Set database to read only', -- step name
@subsystem = N'TSQL', -- Step type
@command = N'ALTER DATABASE SALES SET READ_ONLY', -- Command
@retry_attempts = 5, --Number of attempts
@retry_interval = 5 ; -- in minutes
```

- Dirigir el trabajo a un servidor

```
EXEC dbo.sp_add_jobserver
@job_name = N'Weekly Sales Data Backup',
@server_name = 'MyPC\data; -- Default is LOCAL
GO
```

Crear un horario utilizando SQL

Para crear un programa tenemos que usar un procedimiento almacenado del sistema llamado [sp_add_schedule](#)

```
USE msdb
GO
```



```
EXEC sp_add_schedule
    @schedule_name = N'NightlyJobs' , -- specify the schedule name
    @freq_type = 4, -- A value indicating when a job is to be executed (4) means Daily
    @freq_interval = 1, -- The days that a job is executed and depends on the value of
`freq_type`.
    @active_start_time = 010000 ; -- The time on which execution of a job can begin
GO
```

Hay más parámetros que se pueden usar con `sp_add_schedule` sobre los `sp_add_schedule` puede leer más en el enlace proporcionado anteriormente.

Adjuntar horario a un TRABAJO

Para adjuntar una programación a un trabajo de agente SQL, debe usar un procedimiento almacenado llamado [sp_attach_schedule](#)

```
-- attaches the schedule to the job BackupDatabase
EXEC sp_attach_schedule
    @job_name = N'BackupDatabase', -- The job name to attach with
    @schedule_name = N'NightlyJobs' ; -- The schedule name
GO
```

Lea Tarea o trabajo programado en línea: <https://riptutorial.com/es/sql-server/topic/5329/tarea-o-trabajo-programado>

Capítulo 102: Teclas de acceso directo de Microsoft SQL Server Management Studio

Examples

Ejemplos de atajos

1. Abra una nueva ventana de consulta con la conexión actual (`Ctrl + N`)
2. Alternar entre pestañas abiertas (`Ctrl + Tab`)
3. Mostrar / Ocultar panel de resultados (`Ctrl + R`)
4. Ejecutar consulta resaltada (`Ctrl + E`)
5. Hacer que el texto seleccionado sea mayúscula o minúscula (`Ctrl + Mayús + U` , `Ctrl + Mayús + L`)
6. Intellisense lista de miembros y palabra completa (`Ctrl + Espacio` , `Tabulador`)
7. Ir a la línea (`Ctrl + G`)
8. cerrar una pestaña en SQL Server Management Studio (`Ctrl + F4`)

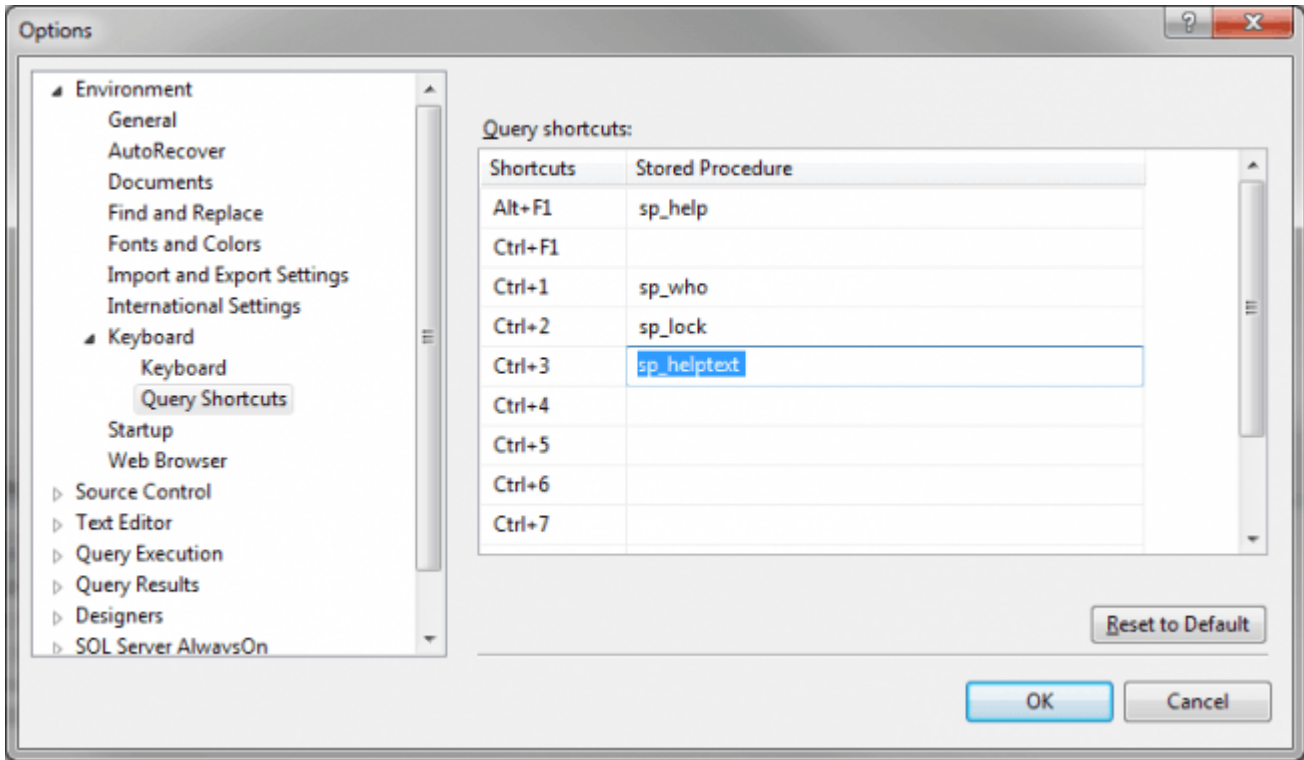
Accesos directos del teclado de activación del menú

1. Mover a la barra de menú de SQL Server Management Studio (`ALT`)
2. Activar el menú para un componente de herramienta (`ALT + HYPHEN`)
3. Mostrar el menú contextual (`MAYÚS + F`)
4. Mostrar el cuadro de diálogo Nuevo archivo para crear un archivo (`CTRL + N`)
5. Mostrar el cuadro de diálogo Abrir proyecto para abrir un proyecto existente (`CTRL + MAYÚS + O`)
6. Mostrar el cuadro de diálogo Agregar nuevo elemento para agregar un nuevo archivo al proyecto actual (`CTRL + MAYÚS + A`)
7. Mostrar el cuadro de diálogo Agregar elemento existente para agregar un archivo existente al proyecto actual (`CTRL + MAYÚS + A`)
8. Mostrar el Diseñador de consultas (`CTRL + MAYÚS + Q`)
9. Cerrar un menú o cuadro de diálogo, cancelando la acción (`ESC`)

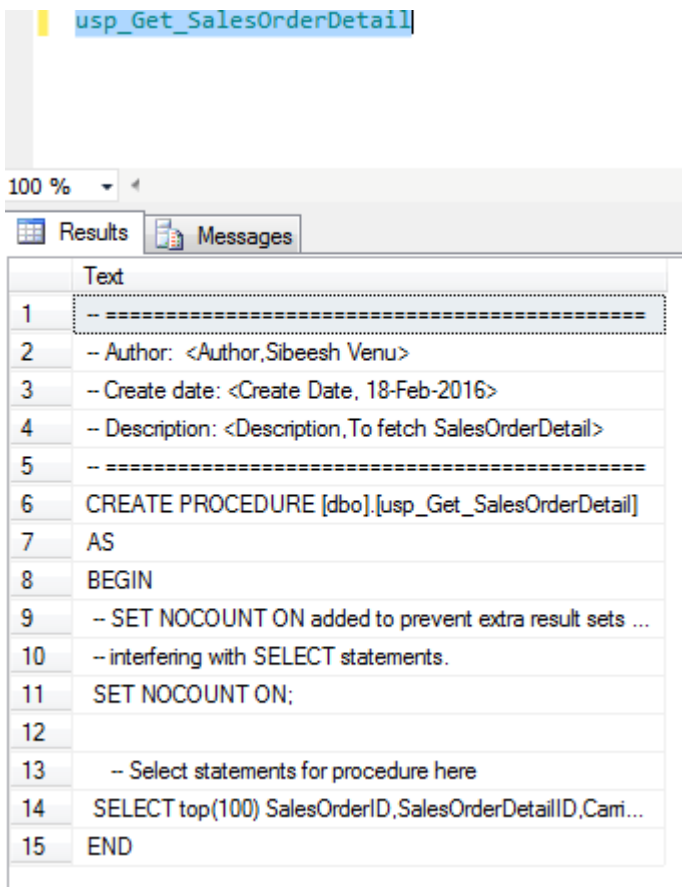
Atajos de teclado personalizados

Vaya a Herramientas -> Opciones. Vaya a Entorno -> Teclado -> Accesos directos de consulta

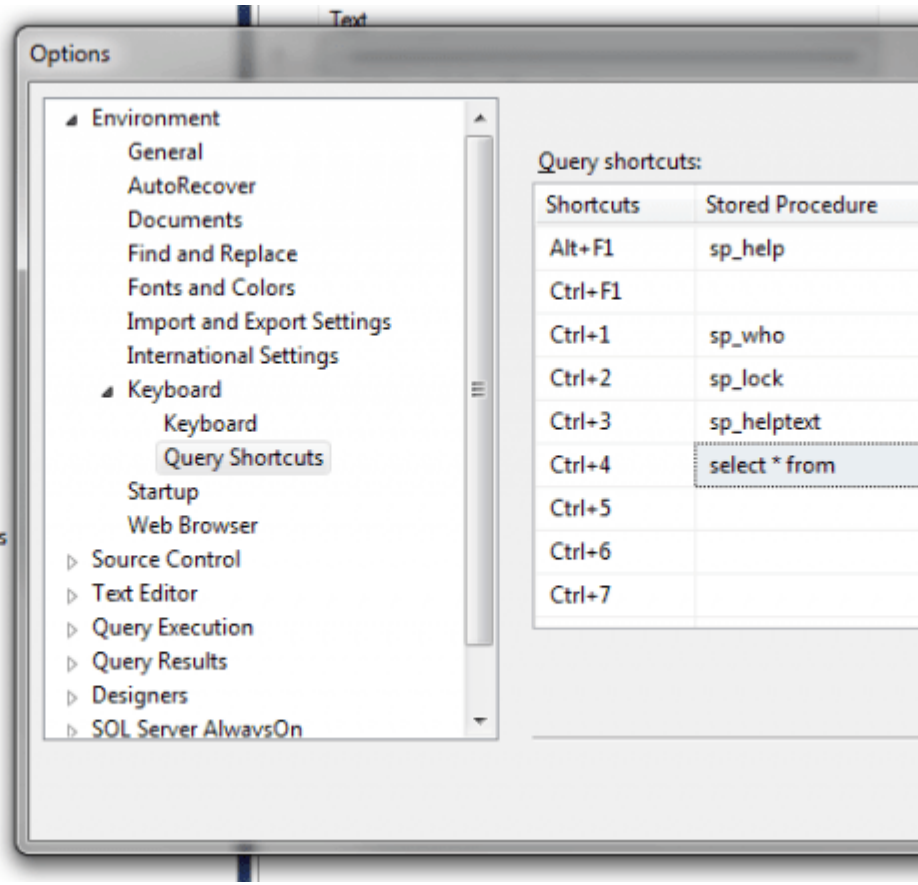
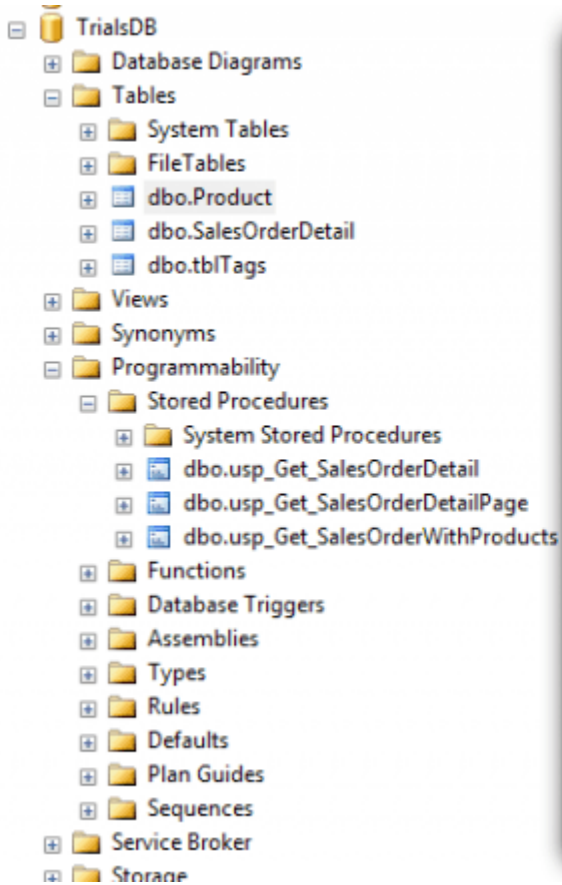
En el lado derecho puede ver algunos accesos directos que están predeterminados en SSMS. Ahora, si necesita agregar uno nuevo, simplemente haga clic en cualquier columna en la columna Procedimiento almacenado.



Haga clic en Aceptar. Ahora vaya a una ventana de consulta y seleccione el procedimiento almacenado, luego presione CTRL + 3, se mostrará el resultado del procedimiento almacenado.



Ahora, si necesita seleccionar todos los registros de una tabla cuando selecciona la tabla y presiona CTRL + 5 (puede seleccionar cualquier tecla). Puede hacer el acceso directo de la siguiente manera.



Ahora siga adelante y seleccione el nombre de la tabla en la ventana de consulta y presione CTRL + 4 (la tecla que seleccionamos), le dará el resultado.

Lea Teclas de acceso directo de Microsoft SQL Server Management Studio en línea:

<https://riptutorial.com/es/sql-server/topic/7749/teclas-de-acceso-directo-de-microsoft-sql-server-management-studio>

Capítulo 103: Tienda de consultas

Examples

Habilitar el almacén de consultas en la base de datos

El almacén de consultas se puede habilitar en la base de datos mediante el siguiente comando:

```
ALTER DATABASE tpch SET QUERY_STORE = ON
```

SQL Server / Azure SQL Database recopilará información sobre las consultas ejecutadas y proporcionará información en las vistas de `sys.query_store`:

- `sys.query_store_query`
- `sys.query_store_query_text`
- `sys.query_store_plan`
- `sys.query_store_runtime_stats`
- `sys.query_store_runtime_stats_interval`
- `sys.database_query_store_options`
- `sys.query_context_settings`

Obtenga estadísticas de ejecución para consultas / planes SQL

La siguiente consulta devolverá información sobre queries, sus planes y estadísticas promedio con respecto a su duración, tiempo de CPU, lecturas físicas y lógicas de io.

```
SELECT Txt.query_text_id, Txt.query_sql_text, Pl.plan_id,  
       avg_duration, avg_cpu_time,  
       avg_physical_io_reads, avg_logical_io_reads  
FROM sys.query_store_plan AS Pl  
JOIN sys.query_store_query AS Qry  
  ON Pl.query_id = Qry.query_id  
JOIN sys.query_store_query_text AS Txt  
  ON Qry.query_text_id = Txt.query_text_id  
JOIN sys.query_store_runtime_stats Stats  
  ON Pl.plan_id = Stats.plan_id
```

Eliminar datos del almacén de consultas

Si desea eliminar alguna consulta o plan de consulta del almacén de consultas, puede usar los siguientes comandos:

```
EXEC sp_query_store_remove_query 4;  
EXEC sp_query_store_remove_plan 3;
```

Los parámetros para estos procedimientos almacenados son el ID de consulta / plan recuperado de las vistas del sistema.

También puede simplemente eliminar las estadísticas de ejecución para un plan particular sin eliminar el plan de la tienda:

```
EXEC sp_query_store_reset_exec_stats 3;
```

Parámetro proporcionado a este ID de plan de procedimiento.

Forzar plan de consulta

El optimizador de consultas SQL seleccionará los posibles planes que puede encontrar para alguna consulta. Si puede encontrar algún plan que funcione de manera óptima para alguna consulta, puede forzar a QO a usar ese plan siempre utilizando el siguiente procedimiento almacenado:

```
EXEC sp_query_store_unforce_plan @query_id, @plan_id
```

Desde este punto, QO siempre usará el plan provisto para la consulta.

Si desea eliminar este enlace, puede utilizar el siguiente procedimiento almacenado:

```
EXEC sp_query_store_force_plan @query_id, @plan_id
```

A partir de este punto, QO intentará nuevamente encontrar el mejor plan.

Lea [Tienda de consultas en línea](https://riptutorial.com/es/sql-server/topic/7349/tienda-de-consultas): <https://riptutorial.com/es/sql-server/topic/7349/tienda-de-consultas>

Capítulo 104: Tipos de datos

Introducción

En esta sección se describen los tipos de datos que puede usar SQL Server, incluidos el rango, la longitud y las limitaciones de los datos (si corresponde).

Examples

Números exactos

Hay dos clases básicas de tipos de datos numéricos exactos: **Integer** , **Precisión fija y Escala** .

Tipos de datos enteros

- poco
- tinyint
- smallint
- En t
- Bigint

Los *enteros* son valores numéricos que nunca contienen una parte fraccionaria, y siempre usan una cantidad fija de almacenamiento. El rango y los tamaños de almacenamiento de los tipos de datos enteros se muestran en esta tabla:

Tipo de datos	Distancia	Almacenamiento
poco	0 o 1	1 bit **
tinyint	0 a 255	1 byte
smallint	-2^{15} (-32,768) a $2^{15}-1$ (32,767)	2 bytes
En t	-2^{31} (-2,147,483,648) a $2^{31}-1$ (2,147,483,647)	4 bytes
Bigint	-2^{63} (-9,223,372,036,854,775,808) a $2^{63}-1$ (9,223,372,036,854,775,807)	8 bytes

Tipos de datos de precisión y escala fija

- numérico
- decimal
- poco dinero
- dinero

Estos tipos de datos son útiles para representar números exactamente. Mientras los valores puedan caber dentro del rango de los valores almacenables en el tipo de datos, el valor no tendrá problemas de redondeo. Esto es útil para cualquier cálculo financiero, donde los errores de redondeo resultarán en locura clínica para los contadores.

Tenga en cuenta que **decimal** y **numérico** son sinónimos para el mismo tipo de datos.

Tipo de datos	Distancia	Almacenamiento
Decimal [(p [, s])] o Numérico [(p [, s])]	$-10^{38} + 1$ a $10^{38} - 1$	Ver tabla de precisión .

Al definir un tipo de datos *decimal* o *numérico*, es posible que deba especificar Precisión [p] y Escala [s].

La precisión es el número de dígitos que se pueden almacenar. Por ejemplo, si necesita almacenar valores entre 1 y 999, necesitará una Precisión de 3 (para mantener los tres dígitos en 100). Si no especifica una precisión, la precisión predeterminada es 18.

Escala es el número de dígitos después del punto decimal. Si necesita almacenar un número entre 0.00 y 999.99, deberá especificar una Precisión de 5 (cinco dígitos) y una Escala de 2 (dos dígitos después del punto decimal). Debe especificar una precisión para especificar una escala. La escala por defecto es cero.

La precisión de un tipo de datos *decimal* o *numérico* define el número de bytes necesarios para almacenar el valor, como se muestra a continuación:

Mesa de precisión

Precisión	Bytes de almacenamiento
1 - 9	5
10-19	9
20-28	13
29-38	17

Tipos de datos fijos monetarios

Estos tipos de datos están destinados específicamente para la contabilidad y otros datos monetarios. Este tipo tiene una escala fija de 4; siempre verá cuatro dígitos después del lugar decimal. Para la mayoría de los sistemas que trabajan con la mayoría de las monedas, será suficiente usar un valor *numérico* con una Escala de 2. Tenga en cuenta que no se almacena ninguna información sobre el tipo de moneda representada con el valor.

Tipo de datos	Distancia	Almacenamiento
dinero	-922,337,203,685,477.5808 a 922,337,203,685,477.5807	8 bytes
poco dinero	-214,748.3648 a 214,748.3647	4 bytes

Números aproximados

- flotar [(n)]
- real

Estos tipos de datos se utilizan para almacenar números de punto flotante. Dado que estos tipos están destinados a contener valores numéricos aproximados solamente, no deben usarse en los casos en que cualquier error de redondeo sea inaceptable. Sin embargo, si necesita manejar números muy grandes o números con un número indeterminado de dígitos después del lugar decimal, esta puede ser su mejor opción.

Tipo de datos	Distancia	tamaño
flotador	-1.79E + 308 a -2.23E-308, 0 y 2.23E-308 a 1.79E + 308	depende de n en la tabla de abajo
real	-3.40E + 38 a -1.18E - 38, 0 y 1.18E - 38 a 3.40E + 38	4 bytes

Tabla de valores **n** para números *flotantes* . Si no se especifica ningún valor en la declaración del flotador, se utilizará el valor predeterminado de 53. Tenga en cuenta que *float (24)* es el equivalente de un valor *real* .

valor n	Precisión	tamaño
1-24	7 dígitos	4 bytes
25-53	15 dígitos	8 bytes

Fecha y hora

Estos tipos están en todas las versiones de SQL Server

- fecha y hora
- tiempo pequeño

Estos tipos están en todas las versiones de SQL Server después de SQL Server 2012

- fecha

- datetimeoffset
- datetime2
- hora

Cadenas de caracteres

- carbonizarse
- varchar
- texto

Cadenas de caracteres Unicode

- nchar
- nvarchar
- ntext

Cuerdas binarias

- binario
- varbinary
- imagen

Otros tipos de datos

- cursor
- marca de tiempo
- hierarchyid
- identificador único
- sql_variant
- xml
- mesa
- Tipos espaciales

Lea Tipos de datos en línea: <https://riptutorial.com/es/sql-server/topic/5260/tipos-de-datos>

Capítulo 105: Tipos de tablas definidas por el usuario

Introducción

Los tipos de tabla definidos por el usuario (UDT, por sus siglas en inglés) son tipos de datos que permiten al usuario definir una estructura de tabla. Los tipos de tablas definidas por el usuario admiten claves primarias, restricciones únicas y valores predeterminados.

Observaciones

Los UDT tienen las siguientes restricciones:

- no se puede utilizar como una columna en una tabla o un campo en un tipo estructurado definido por el usuario
- no se puede crear un índice no agrupado en un UDT a menos que el índice sea el resultado de crear una restricción PRIMARIA KEY o UNIQUE en el UDT
- La definición de UDT NO SE PUEDE modificar después de su creación

Examples

creando un UDT con una sola columna int que también es una clave principal

```
CREATE TYPE dbo.Ids as TABLE
(
    Id int PRIMARY KEY
)
```

Creando un UDT con múltiples columnas

```
CREATE TYPE MyComplexType as TABLE
(
    Id int,
    Name varchar(10)
)
```

Creando un UDT con una restricción única:

```
CREATE TYPE MyUniqueNamesType as TABLE
(
    FirstName varchar(10),
    LastName varchar(10),
    UNIQUE (FirstName, LastName)
)
```

Nota: las restricciones en los tipos de tablas definidas por el usuario no pueden ser nombradas.

Creación de un UDT con una clave principal y una columna con un valor predeterminado:

```
CREATE TYPE MyUniqueNamesType as TABLE
(
    FirstName varchar(10),
    LastName varchar(10),
    CreateDate datetime default GETDATE()
    PRIMARY KEY (FirstName,LastName)
)
```

Lea Tipos de tablas definidas por el usuario en línea: <https://riptutorial.com/es/sql-server/topic/5280/tipos-de-tablas-definidas-por-el-usuario>

Capítulo 106: TRATA DE ATRAPARLO

Observaciones

TRY / CATCH es una construcción de lenguaje específica para T-SQL de MS SQL Server.

Permite el manejo de errores dentro de T-SQL, similar al visto en el código .NET.

Examples

Transacción en un TRY / CATCH

Esto hará retroceder ambas inserciones debido a una fecha y hora no válida:

```
BEGIN TRANSACTION
BEGIN TRY
    INSERT INTO dbo.Sale (Price, SaleDate, Quantity)
    VALUES (5.2, GETDATE(), 1)
    INSERT INTO dbo.Sale (Price, SaleDate, Quantity)
    VALUES (5.2, 'not a date', 1)
    COMMIT TRANSACTION
END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION -- First Rollback and then throw.
    THROW
END CATCH
```

Esto comprometerá ambas inserciones:

```
BEGIN TRANSACTION
BEGIN TRY
    INSERT INTO dbo.Sale (Price, SaleDate, Quantity)
    VALUES (5.2, GETDATE(), 1)
    INSERT INTO dbo.Sale (Price, SaleDate, Quantity)
    VALUES (5.2, GETDATE(), 1)
    COMMIT TRANSACTION
END TRY
BEGIN CATCH
    THROW
    ROLLBACK TRANSACTION
END CATCH
```

Aumento de errores en el bloque try-catch

La función RAISERROR generará un error en el bloque TRY CATCH:

```
DECLARE @msg nvarchar(50) = 'Here is a problem!'
BEGIN TRY
    print 'First statement';
    RAISERROR(@msg, 11, 1);
    print 'Second statement';
```

```

END TRY
BEGIN CATCH
    print 'Error: ' + ERROR_MESSAGE();
END CATCH

```

RAISERROR con un segundo parámetro mayor que 10 (11 en este ejemplo) detendrá la ejecución en TRY BLOCK y generará un error que se manejará en el bloque CATCH. Puede acceder al mensaje de error utilizando la función `ERROR_MESSAGE ()`. La salida de esta muestra es:

```

First statement
Error: Here is a problem!

```

Generar mensajes de información en el bloque try catch

RAISERROR con severidad (segundo parámetro) menor o igual a 10 no arrojará la excepción.

```

BEGIN TRY
    print 'First statement';
    RAISERROR( 'Here is a problem!', 10, 15);
    print 'Second statement';
END TRY
BEGIN CATCH
    print 'Error: ' + ERROR_MESSAGE();
END CATCH

```

Después de la instrucción **RAISERROR**, se ejecutará la tercera instrucción y no se invocará el bloque **CATCH**. El resultado de la ejecución es:

```

First statement
Here is a problem!
Second statement

```

Excepción de relanzamiento generada por RAISERROR.

Puede volver a lanzar el error que atrapó en el bloque **CATCH** utilizando la instrucción **THROW**:

```

DECLARE @msg nvarchar(50) = 'Here is a problem! Area: ''%s'' Line:''%i'''
BEGIN TRY
    print 'First statement';
    RAISERROR(@msg, 11, 1, 'TRY BLOCK', 2);
    print 'Second statement';
END TRY
BEGIN CATCH
    print 'Error: ' + ERROR_MESSAGE();
    THROW;
END CATCH

```

Tenga en cuenta que en este caso estamos generando errores con argumentos formateados (cuarto y quinto parámetro). Esto puede ser útil si desea agregar más información en el mensaje. El resultado de la ejecución es:

```
First statement
Error: Here is a problem! Area: 'TRY BLOCK' Line:'2'
Msg 50000, Level 11, State 1, Line 26
Here is a problem! Area: 'TRY BLOCK' Line:'2'
```

Excepción de lanzamiento en bloques TRY / CATCH

Puedes lanzar una excepción en el bloque try catch:

```
DECLARE @msg nvarchar(50) = 'Here is a problem!'
BEGIN TRY
    print 'First statement';
    THROW 51000, @msg, 15;
    print 'Second statement';
END TRY
BEGIN CATCH
    print 'Error: ' + ERROR_MESSAGE();
    THROW;
END CATCH
```

Las excepciones se manejan en el bloque CATCH y luego se vuelven a lanzar con el comando THROW sin parámetros.

```
First statement
Error: Here is a problem!
Msg 51000, Level 16, State 15, Line 39
Here is a problem!
```

THROW es similar a RAISERROR con las siguientes diferencias:

- La recomendación es que las nuevas aplicaciones deberían utilizar el THROW en lugar del RAISERROR.
- THROW puede usar cualquier número como primer argumento (número de error), RAISERROR puede usar solo los identificadores en la vista sys.messages
- THROW tiene gravedad 16 (no se puede cambiar)
- THROW no puede formatear argumentos como RAISERROR. Use la función FORMATMESSAGE como un argumento de RAISERROR si necesita esta función.

Lea **TRATA DE ATRAPARLO** en línea: <https://riptutorial.com/es/sql-server/topic/5189/trata-de-atraparlo>

Capítulo 107: Última identidad insertada

Examples

SCOPE_IDENTITY ()

```
CREATE TABLE dbo.logging_table(log_id INT IDENTITY(1,1) PRIMARY KEY,
                                log_message VARCHAR(255))

CREATE TABLE dbo.person(person_id INT IDENTITY(1,1) PRIMARY KEY,
                          person_name VARCHAR(100) NOT NULL)

GO;

CREATE TRIGGER dbo.InsertToADifferentTable ON dbo.person
AFTER INSERT
AS
    INSERT INTO dbo.logging_table(log_message)
    VALUES('Someone added something to the person table')
GO;

INSERT INTO dbo.person(person_name)
VALUES('John Doe')

SELECT SCOPE_IDENTITY();
```

Esto devolverá el valor de identidad agregado más recientemente producido en la misma conexión, dentro del alcance actual. En este caso, 1, para la primera fila en la tabla `dbo.person`.

@@IDENTIDAD

```
CREATE TABLE dbo.logging_table(log_id INT IDENTITY(1,1) PRIMARY KEY,
                                log_message VARCHAR(255))

CREATE TABLE dbo.person(person_id INT IDENTITY(1,1) PRIMARY KEY,
                          person_name VARCHAR(100) NOT NULL)

GO;

CREATE TRIGGER dbo.InsertToADifferentTable ON dbo.person
AFTER INSERT
AS
    INSERT INTO dbo.logging_table(log_message)
    VALUES('Someone added something to the person table')
GO;

INSERT INTO dbo.person(person_name)
VALUES('John Doe')

SELECT @@IDENTITY;
```

Esto devolverá la identidad agregada más recientemente en la misma conexión, independientemente del alcance. En este caso, sea cual sea el valor actual de la columna de identidad en `logging_table`, supongamos que no se está produciendo ninguna otra actividad en la

instancia de SQL Server y que no se activan otros activadores desde esta inserción.

IDENT_CURRENT ('nombre de tabla')

```
SELECT IDENT_CURRENT('dbo.person');
```

Esto seleccionará el último valor de identidad agregado en la tabla seleccionada, independientemente de la conexión o el alcance.

@ @ IDENTIDAD y MAX (ID)

```
SELECT MAX(Id) FROM Employees -- Display the value of Id in the last row in Employees table.
GO
INSERT INTO Employees (FName, LName, PhoneNumber) -- Insert a new row
VALUES ('John', 'Smith', '25558696525')
GO
SELECT @@IDENTITY
GO
SELECT MAX(Id) FROM Employees -- Display the value of Id of the newly inserted row.
GO
```

Los dos últimos valores de instrucciones SELECT son los mismos.

Lea Última identidad insertada en línea: <https://riptutorial.com/es/sql-server/topic/5674/ultima-identidad-insertada>

Capítulo 108: UNIÓN

Examples

Unión y unión todos

La operación de **unión** combina los resultados de dos o más consultas en un único conjunto de resultados que incluye todas las filas que pertenecen a todas las consultas en la unión e ignorará cualquier duplicado que exista. **Union** también hace lo mismo pero incluye incluso los valores duplicados. El concepto de operación sindical quedará claro en el siguiente ejemplo. Algunas cosas a considerar al usar la unión son:

- 1.El número y el orden de las columnas deben ser iguales en todas las consultas.
2. Los tipos de datos deben ser compatibles.

Ejemplo:

Tenemos tres tablas: Marksheet1, Marksheet2 y Marksheet3. Marksheet3 es la tabla duplicada de Marksheet2 que contiene los mismos valores que la de Marksheet2.

Tabla 1: Marksheet1

SubjectCode	SubjectName	MarksObtained
101	Physics	87
102	Chemistry	75
103	Maths	85
104	English	89
105	Computer	95

Tabla 2: Marksheet2

CourseCode	CourseName	MarksObtained
201	PhysicsII	82
202	ChemistryII	86
203	MathsII	95
204	EnglishII	70
205	ComputerII	86

Table3 : Marksheet3

SubjectCode	SubjectName	MarksObtained
201	PhysicsII	82
202	ChemistryII	86
203	MathsII	95
204	EnglishII	70
205	ComputerII	86

Unión sobre tablas Marksheet1 y Marksheet2

```
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet1
UNION
SELECT CourseCode, CourseName, MarksObtained
FROM Marksheet2
```

Nota: La salida para la unión de las tres tablas también será la misma que la unión en Marksheet1 y Marksheet2 porque la operación de unión no toma valores duplicados.

```
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet1
UNION
SELECT CourseCode, CourseName, MarksObtained
FROM Marksheet2
UNION
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet3
```

SALIDA

	SubjectCode	SubjectName	MarksObtained
1	101	Physics	87
2	102	Chemistry	75
3	103	Maths	85
4	104	English	89
5	105	Computer	95
6	201	PhysicsII	82
7	202	ChemistryII	86
8	203	MathsII	95
9	204	EnglishII	70
10	205	ComputerII	86

Unión de todos

```
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet1
UNION ALL
SELECT CourseCode, CourseName, MarksObtained
FROM Marksheet2
UNION ALL
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet3
```

SALIDA

	SubjectCode	SubjectName	MarksObtained
1	101	Physics	87
2	102	Chemistry	75
3	103	Maths	85
4	104	English	89
5	105	Computer	95
6	201	PhysicsII	82
7	202	ChemistryII	86
8	203	MathsII	95
9	204	EnglishII	70
10	205	ComputerII	86
11	201	PhysicsII	82
12	202	ChemistryII	86
13	203	MathsII	95
14	204	EnglishII	70
15	205	ComputerII	86

Notarás aquí que los valores duplicados de Marksheet3 también se muestran usando union all.

Lea **UNIÓN** en línea: <https://riptutorial.com/es/sql-server/topic/5590/union>

Capítulo 109: UNIR

Introducción

A partir de SQL Server 2008, es posible realizar operaciones de inserción, actualización o eliminación en una sola declaración utilizando la instrucción MERGE.

La declaración MERGE le permite unir un origen de datos con una tabla o vista de destino y, a continuación, realizar varias acciones contra el destino en función de los resultados de esa unión.

Sintaxis

- Según MSDN: <https://msdn.microsoft.com/en-us/library/bb510625.aspx> [WITH <common_table_expression> [, ... n]] MERGE [TOP (expresión) [PERCENT]] [INTO] <target_table> [WITH (<merge_hint>)] [[AS] table_alias] USING <table_source> ON <merge_search_condition> [WHEN MATCHED [AND <clause_search_condition>] THEN <merge_matched>] [... n] [WHEN NOT MATCHED [BY OBJETIVO] [AND <clause_search_condition>] LUEGO <merge_not_matched>] [CUANDO NO SE AJUSTE POR FUENTE [AND <clause_search_condition>] LUEGO <merge_matched>] [... n] [<output_clause>] [OPTION (<query_matched>. ...n)]; <target_table> :: = {[database_name. nombre_esquema. | nombre_esquema.] target_table} <merge_hint> :: = {{{<table_hint_limited> [, ... n]] [[,] INDEX (index_val [, ... n])]]} <table_source> :: = {table_or_view_name [[AS] table_alias] [<tablesample_clause>] [WITH (table_hint [[,] ... n])] | rowset_function [[AS] table_alias] [(bulk_column_alias [, ... n])] | user_defined_function [[AS] table_alias] | OPENXML <openxml_clause> | tabla_de_derivada [AS] table_alias [(column_alias [, ... n])] | <joined_table> | <pivoted_table> | <unpivoted_table>} <merge_search_condition> :: = <search_condition> <merge_matched> :: = {UPDATE SET <set_clause> | DELETE} <set_clause> :: = SET {column_name = {expresión | Por defecto | Nulo} | {udt_column_name. {{property_name = expresión | field_name = expresión} | nombre_método (argumento [, ... n])}} | column_name {.WRITE (expresión, @Offset, @Length)} | @variable = expresión | @variable = columna = expresión | nombre_columna {+ = | - = | * = | / = | % = | & = | ^ = | | =} expresión | @variable {+ = | - = | * = | / = | % = | & = | ^ = | | =} expresión | @variable = columna {+ = | - = | * = | / = | % = | & = | ^ = | | =} expresión} [, ... n] <merge_not_matched> :: = {INSERT [(column_list)] {VALUES (values_list) | VALORES POR DEFECTO}} <clause_search_condition> :: = <search_condition> :: = {[NOT] | (<search_condition>)} [{AND | O} [NO] {| (<search_condition>)}] [, ... n] :: = {expresión {= | <> | != | > | < | <= | != <} expresión | string_expression [NO] LIKE string_expression [ESCAPE 'escape_character'] | expresión [NO] ENTRE expresión Y expresión | expresión es [no] nula | CONTIENE ({columna | *}, '<contiene_condición_de_búsqueda>') | FREETEXT ({column | *}, 'freetext_string') | expresión [NOT] IN (subconsulta | expresión [, ... n]) | expresión {= | <> | != | > = | != > | < | <= | != <} {TODOS | Algunos | ANY} (subconsulta) | EXISTS (subquery)} <output_clause> :: = {[OUTPUT <dml_select_list> INTO

| > = | ! > | < | <= | ! <} expresión | string_expression [NO] LIKE string_expression [ESCAPE 'escape_character'] | expresión [NO] ENTRE expresión Y expresión | expresión es [no] nula | CONTIENE ({columna | *}, '<contiene_condición_de_búsqueda>') | FREETEXT ({column | *}, 'freetext_string') | expresión [NOT] IN (subconsulta | expresión [, ... n]) | expresión {= | <> | != | > = | != > | < | <= | != <} {TODOS | Algunos | ANY} (subconsulta) | EXISTS (subquery)} <output_clause> :: = {[OUTPUT <dml_select_list> INTO

```
{@table_variable | output_table} [(column_list)] [OUTPUT <dml_select_list>]
<dml_select_list> ::= {<column_name> | scalar_expression} [[AS]
column_alias_identifer] [, ... n] <column_name> ::= {DELETED | Insertado |
from_table_name}. {* | nombre_columna} | $ acción
```

Observaciones

Realiza operaciones de inserción, actualización o eliminación en una tabla de destino en función de los resultados de una unión con una tabla de origen. Por ejemplo, puede sincronizar dos tablas insertando, actualizando o eliminando filas en una tabla en función de las diferencias encontradas en la otra tabla.

Examples

MERGE para insertar / actualizar / eliminar

```
MERGE INTO targetTable

USING sourceTable
ON (targetTable.PKID = sourceTable.PKID)

WHEN MATCHED AND (targetTable.PKID > 100) THEN
    DELETE

WHEN MATCHED AND (targetTable.PKID <= 100) THEN
    UPDATE SET
        targetTable.ColumnA = sourceTable.ColumnA,
        targetTable.ColumnB = sourceTable.ColumnB

WHEN NOT MATCHED THEN
    INSERT (ColumnA, ColumnB) VALUES (sourceTable.ColumnA, sourceTable.ColumnB);

WHEN NOT MATCHED BY SOURCE THEN
    DELETE
; --< Required
```

Descripción:

- `MERGE INTO targetTable` - tabla a modificar
- `USING sourceTable` - fuente de datos (puede ser una tabla o una vista o una función con valores de tabla)
- `ON ...` : condición de unión entre `targetTable` y `sourceTable` .
- `WHEN MATCHED` - acciones a tomar cuando se encuentra una coincidencia
 - `AND (targetTable.PKID > 100)` - condición (es) adicional (es) que deben cumplirse para que la acción se realice
- `THEN DELETE` : eliminar registro coincidente de la `targetTable`
- `THEN UPDATE` : actualice las columnas del registro coincidente especificadas por `SET ...`
- `WHEN NOT MATCHED` : acciones a tomar cuando no se encuentra una `targetTable` en `targetTable`
- `WHEN NOT MATCHED BY SOURCE` : acciones a tomar cuando no se encuentra una `sourceTable` en la `sourceTable` de `sourceTable`

Comentarios:

Si no se necesita una acción específica, omite la condición, por ejemplo, eliminar `WHEN NOT MATCHED THEN INSERT` evitará que se inserten registros

La declaración de fusión requiere un punto y coma de terminación.

Restricciones:

- `WHEN MATCHED` no permite la acción `INSERT`
- `UPDATE` acción `UPDATE` puede actualizar una fila solo una vez. Esto implica que la condición de unión debe producir coincidencias únicas.

Combinar usando la fuente CTE

```
WITH SourceTableCTE AS
(
    SELECT * FROM SourceTable
)
MERGE
    TargetTable AS target
USING SourceTableCTE AS source
ON (target.PKID = source.PKID)
WHEN MATCHED THEN
    UPDATE SET target.ColumnA = source.ColumnA
WHEN NOT MATCHED THEN
    INSERT (ColumnA) VALUES (Source.ColumnA);
```

MERGE usando la tabla de fuente derivada

```
MERGE INTO TargetTable AS Target
USING (VALUES (1,'Value1'), (2, 'Value2'), (3,'Value3'))
    AS Source (PKID, ColumnA)
ON Target.PKID = Source.PKID
WHEN MATCHED THEN
    UPDATE SET target.ColumnA= source.ColumnA
WHEN NOT MATCHED THEN
    INSERT (PKID, ColumnA) VALUES (Source.PKID, Source.ColumnA);
```

Ejemplo de combinación: sincronizar origen y tabla de destino

Para ilustrar la declaración `MERGE`, considere las siguientes dos tablas:

1. **dbo.Producto** : esta tabla contiene información sobre el producto que la compañía está vendiendo actualmente
2. **dbo.ProductNew** : esta tabla contiene información sobre el producto que la compañía venderá en el futuro.

El siguiente T-SQL creará y llenará estas dos tablas

```
IF OBJECT_id(N'dbo.Product',N'U') IS NOT NULL
```

```

DROP TABLE dbo.Product
GO

CREATE TABLE dbo.Product (
ProductID INT PRIMARY KEY,
ProductName NVARCHAR(64),
PRICE MONEY
)

IF OBJECT_id(N'dbo.ProductNew',N'U') IS NOT NULL
DROP TABLE dbo.ProductNew
GO

CREATE TABLE dbo.ProductNew (
ProductID INT PRIMARY KEY,
ProductName NVARCHAR(64),
PRICE MONEY
)

INSERT INTO dbo.Product VALUES (1,'IPod',300)
, (2,'IPhone',400)
, (3,'ChromeCast',100)
, (4,'raspberry pi',50)

INSERT INTO dbo.ProductNew VALUES (1,'Asus Notebook',300)
, (2,'Hp Notebook',400)
, (3,'Dell Notebook',100)
, (4,'raspberry pi',50)

```

Ahora, supongamos que queremos sincronizar la tabla de destino `dbo.Product` con la tabla `dbo.ProductNew`. Aquí está el criterio para esta tarea:

1. El producto que existe tanto en la tabla de origen `dbo.ProductNew` como en la tabla de destino `dbo.Product` se actualizan en la tabla de destino `dbo.Product` con nuevos productos nuevos.
2. Cualquier producto en la tabla de origen `dbo.ProductNew` que no existe en la tabla de destino `dbo.Product` se inserta en la tabla de destino `dbo.Product`.
3. Cualquier producto en la tabla de destino de `dbo.Product` que no exista en la tabla de origen `dbo.ProductNew` debe eliminarse de la tabla de destino de `dbo.Product`. Aquí está la declaración `MERGE` para realizar esta tarea.

```

MERGE dbo.Product AS SourceTbl
USING dbo.ProductNew AS TargetTbl ON (SourceTbl.ProductID = TargetTbl.ProductID)
WHEN MATCHED
    AND SourceTbl.ProductName <> TargetTbl.ProductName
    OR SourceTbl.Price <> TargetTbl.Price
THEN UPDATE SET SourceTbl.ProductName = TargetTbl.ProductName,
    SourceTbl.Price = TargetTbl.Price
WHEN NOT MATCHED
THEN INSERT (ProductID, ProductName, Price)
    VALUES (TargetTbl.ProductID, TargetTbl.ProductName, TargetTbl.Price)
WHEN NOT MATCHED BY SOURCE
THEN DELETE
OUTPUT $action, INSERTED.*, DELETED.*;

```


Nota: el punto y coma debe estar presente al final de la sentencia MERGE.

	Saction	ProductID	ProductName	PRICE	ProductID	ProductName	PRICE
1	UPDATE	1	Asus Notebook	300.00	1	iPod	300.00
2	UPDATE	2	Hp Notebook	400.00	2	iPhone	400.00
3	UPDATE	3	Dell Notebook	100.00	3	ChromeCast	100.00

Fusionar utilizando EXCEPTO

Use EXCEPTO para evitar actualizaciones de registros sin cambios

```
MERGE TargetTable targ
USING SourceTable AS src
  ON src.id = targ.id
WHEN MATCHED
  AND EXISTS (
    SELECT src.field
    EXCEPT
    SELECT targ.field
  )
  THEN
    UPDATE
    SET field = src.field
WHEN NOT MATCHED BY TARGET
  THEN
    INSERT (
      id
      ,field
    )
    VALUES (
      src.id
      ,src.field
    )
WHEN NOT MATCHED BY SOURCE
  THEN
    DELETE;
```

Lea UNIR en línea: <https://riptutorial.com/es/sql-server/topic/4550/unir>

Capítulo 110: Unirse

Introducción

En el lenguaje de consulta estructurado (SQL), un JOIN es un método para vincular dos tablas de datos en una sola consulta, lo que permite que la base de datos devuelva un conjunto que contiene datos de ambas tablas a la vez, o usar datos de una tabla para usarlos como una Filtar en la segunda mesa. Hay varios tipos de JOIN definidos dentro del estándar ANSI SQL.

Examples

Unir internamente

`Inner join` devuelve solo aquellos registros / filas que coinciden / existen en ambas tablas en función de una o más condiciones (especificadas mediante la palabra clave `ON`). Es el tipo más común de unión. La sintaxis general para `inner join` es:

```
SELECT *
FROM table_1
INNER JOIN table_2
    ON table_1.column_name = table_2.column_name
```

También se puede simplificar como `JOIN` :

```
SELECT *
FROM table_1
JOIN table_2
    ON table_1.column_name = table_2.column_name
```

Ejemplo

```
/* Sample data. */
DECLARE @Animal table (
    AnimalId Int IDENTITY,
    Animal Varchar(20)
);

DECLARE @AnimalSound table (
    AnimalSoundId Int IDENTITY,
    AnimalId Int,
    Sound Varchar(20)
);

INSERT INTO @Animal (Animal) VALUES ('Dog');
INSERT INTO @Animal (Animal) VALUES ('Cat');
INSERT INTO @Animal (Animal) VALUES ('Elephant');

INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (1, 'Barks');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (2, 'Meows');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (3, 'Trumpets');
```

```

/* Sample data prepared. */

SELECT
    *
FROM
    @Animal
    JOIN @AnimalSound
        ON @Animal.AnimalId = @AnimalSound.AnimalId;

```

AnimalId	Animal	AnimalSoundId	AnimalId	Sound
1	Dog	1	1	Barks
2	Cat	2	2	Meows
3	Elephant	3	3	Trumpets

Uso de la combinación interna con la combinación externa izquierda (Sustituto por No existe)

Esta consulta devolverá los datos de la tabla 1, donde los campos que coinciden con la tabla 2 con una clave y los datos que no están en la Tabla 1 al comparar con la Tabla 2 con una condición y clave

```

select *
from Table1 t1
    inner join Table2 t2 on t1.ID_Column = t2.ID_Column
    left join Table3 t3 on t1.ID_Column = t3.ID_Column
where t2.column_name = column_value
    and t3.ID_Column is null
order by t1.column_name;

```

Cruzar

A `cross join` es una combinación cartesiana, es decir, un producto cartesiano de ambas tablas. Esta unión no necesita ninguna condición para unir dos tablas. Cada fila de la tabla izquierda se unirá a cada fila de la tabla derecha. Sintaxis para una unión cruzada:

```

SELECT * FROM table_1
CROSS JOIN table_2

```

Ejemplo:

```

/* Sample data. */
DECLARE @Animal table (
    AnimalId Int IDENTITY,
    Animal Varchar(20)
);

DECLARE @AnimalSound table (
    AnimalSoundId Int IDENTITY,
    AnimalId Int,
    Sound Varchar(20)
);

INSERT INTO @Animal (Animal) VALUES ('Dog');

```

```

INSERT INTO @Animal (Animal) VALUES ('Cat');
INSERT INTO @Animal (Animal) VALUES ('Elephant');

INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (1, 'Barks');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (2, 'Meows');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (3, 'Trumpet');
/* Sample data prepared. */

SELECT
    *
FROM
    @Animal
    CROSS JOIN @AnimalSound;

```

Resultados:

AnimalId	Animal	AnimalSoundId	AnimalId	Sound
1	Dog	1	1	Barks
2	Cat	1	1	Barks
3	Elephant	1	1	Barks
1	Dog	2	2	Meows
2	Cat	2	2	Meows
3	Elephant	2	2	Meows
1	Dog	3	3	Trumpet
2	Cat	3	3	Trumpet
3	Elephant	3	3	Trumpet

Tenga en cuenta que hay otras formas de aplicar CROSS JOIN. Esta es una combinación de *"estilo antiguo"* (en desuso desde ANSI SQL-92) sin condición, lo que resulta en una combinación cruzada / cartesiana:

```

SELECT *
FROM @Animal, @AnimalSound;

```

Esta sintaxis también funciona debido a una condición de unión "siempre verdadera", pero no se recomienda y debe evitarse, en favor de la sintaxis explícita de `CROSS JOIN`, para facilitar la lectura.

```

SELECT *
FROM
    @Animal
    JOIN @AnimalSound
        ON 1=1

```

Unión externa

Izquierda combinación externa

`LEFT JOIN` devuelve todas las filas de la tabla de la izquierda, que coinciden con las filas de la tabla de la derecha donde se cumplen las condiciones de la cláusula `ON`. Las filas en las que no se cumple la cláusula `ON` tienen `NULL` en todas las columnas de la tabla derecha. La sintaxis de un `LEFT JOIN` es:

```
SELECT * FROM table_1 AS t1
LEFT JOIN table_2 AS t2 ON t1.ID_Column = t2.ID_Column
```

Unión externa derecha

RIGHT JOIN devuelve todas las filas de la tabla derecha, que coinciden con las filas de la tabla izquierda donde se cumplen las condiciones de la cláusula **ON** . Las filas en las que no se cumple la cláusula **ON** tienen **NULL** en todas las columnas de la tabla izquierda. La sintaxis de un **RIGHT JOIN** es:

```
SELECT * FROM table_1 AS t1
RIGHT JOIN table_2 AS t2 ON t1.ID_Column = t2.ID_Column
```

Unión externa completa

FULL JOIN combina **LEFT JOIN** y **RIGHT JOIN** . Todas las filas se devuelven desde ambas tablas, independientemente de si se cumplen las condiciones de la cláusula **ON** . Las filas que no satisfacen la cláusula **ON** se devuelven con **NULL** en todas las columnas de la tabla opuesta (es decir, para una fila en la tabla izquierda, todas las columnas en la tabla derecha contendrán **NULL** y viceversa). La sintaxis de un **FULL JOIN** es:

```
SELECT * FROM table_1 AS t1
FULL JOIN table_2 AS t2 ON t1.ID_Column = t2.ID_Column
```

Ejemplos

```
/* Sample test data. */
DECLARE @Animal table (
    AnimalId Int IDENTITY,
    Animal Varchar(20)
);

DECLARE @AnimalSound table (
    AnimalSoundId Int IDENTITY,
    AnimalId Int,
    Sound Varchar(20)
);

INSERT INTO @Animal (Animal) VALUES ('Dog');
INSERT INTO @Animal (Animal) VALUES ('Cat');
INSERT INTO @Animal (Animal) VALUES ('Elephant');
INSERT INTO @Animal (Animal) VALUES ('Frog');

INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (1, 'Barks');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (2, 'Meows');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (3, 'Trumpet');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (5, 'Roars');
/* Sample data prepared. */
```

IZQUIERDA COMBINACIÓN EXTERNA

```
SELECT *
FROM @Animal As t1
```

```
LEFT JOIN @AnimalSound As t2 ON t1.AnimalId = t2.AnimalId;
```

Resultados para LEFT JOIN

AnimalId	Animal	AnimalSoundId	AnimalId	Sound
1	Dog	1	1	Barks
2	Cat	2	2	Meows
3	Elephant	3	3	Trumpet
4	Frog	NULL	NULL	NULL

JUSTE EXTERIOR DERECHO

```
SELECT *  
FROM @Animal As t1  
RIGHT JOIN @AnimalSound As t2 ON t1.AnimalId = t2.AnimalId;
```

Resultados de RIGHT JOIN

AnimalId	Animal	AnimalSoundId	AnimalId	Sound
1	Dog	1	1	Barks
2	Cat	2	2	Meows
3	Elephant	3	3	Trumpet
NULL	NULL	4	5	Roars

ÚNICAMENTE EN EL EXTERIOR

```
SELECT *  
FROM @Animal As t1  
FULL JOIN @AnimalSound As t2 ON t1.AnimalId = t2.AnimalId;
```

Resultados para FULL JOIN

AnimalId	Animal	AnimalSoundId	AnimalId	Sound
1	Dog	1	1	Barks
2	Cat	2	2	Meows
3	Elephant	3	3	Trumpet
4	Frog	NULL	NULL	NULL
NULL	NULL	4	5	Roars

Uso de unirse en una actualización

Las combinaciones también se pueden utilizar en una instrucción UPDATE :

```
CREATE TABLE Users (  
    UserId int NOT NULL,  
    AccountId int NOT NULL,  
    RealName nvarchar(200) NOT NULL  
)
```

```
CREATE TABLE Preferences (  
    UserId int NOT NULL,  
    SomeSetting bit NOT NULL  
)
```

Actualice la columna `SomeSetting` del `SomeSetting` de la tabla de `Preferences` por un predicado en la tabla de `Users` siguiente manera:

```
UPDATE p  
SET p.SomeSetting = 1  
FROM Users u  
JOIN Preferences p ON u.UserId = p.UserId  
WHERE u.AccountId = 1234
```

`p` es un alias para las `Preferences` definidas en la cláusula `FROM` de la declaración. Solo se actualizarán las filas con un `AccountId` coincidente de la tabla `Users`.

Actualización con sentencias de unión externa izquierda

```
Update t  
SET t.Column1=100  
FROM Table1 t LEFT JOIN Table12 t2  
ON t2.ID=t.ID
```

Actualizar tablas con función de unión interna y agregación

```
UPDATE t1  
SET t1.field1 = t2.field2Sum  
FROM table1 t1  
INNER JOIN (select field3, sum(field2) as field2Sum  
from table2  
group by field3) as t2  
on t2.field3 = t1.field3
```

Únete a una subconsulta

La incorporación a una subconsulta se usa a menudo cuando se desean obtener datos agregados (como `Count`, `Avg`, `Max` o `Min`) de una tabla secundaria / de detalles y mostrarlos junto con los registros de la tabla principal o de encabezado. Por ejemplo, es posible que desee recuperar la fila superior / primera secundaria en función de la fecha o la identificación o tal vez desee un recuento de todas las filas secundarias o un promedio.

Este ejemplo utiliza alias que facilitan la lectura de consultas cuando tiene varias tablas involucradas. En este caso, estamos recuperando todas las filas de las órdenes de compra de la tabla principal y recuperando solo la última fila secundaria (o la más reciente) de la tabla secundaria `PurchaseOrderLineItems`. Este ejemplo asume que la tabla secundaria usa identificadores numéricos incrementales.

```
SELECT po.Id, po.PODate, po.VendorName, po.Status, item.ItemNo,  
    item.Description, item.Cost, item.Price  
FROM PurchaseOrders po
```

```
LEFT JOIN
(
  SELECT l.PurchaseOrderId, l.ItemNo, l.Description, l.Cost, l.Price, Max(l.id) as Id
  FROM PurchaseOrderLineItems l
  GROUP BY l.PurchaseOrderId, l.ItemNo, l.Description, l.Cost, l.Price
) AS item ON item.PurchaseOrderId = po.Id
```

Auto unirse

Una tabla se puede unir a sí misma en lo que se conoce como una unión automática, combinando registros en la tabla con otros registros en la misma tabla. Las uniones automáticas se utilizan normalmente en consultas donde se define una jerarquía en las columnas de la tabla.

Considere los datos de muestra en una tabla llamada `Employees` :

CARNÉ DE IDENTIDAD	Nombre	Boss_ID
1	Mover	3
2	Jim	1
3	Sam	2

El `Boss_ID` cada empleado se asigna a la `ID` otro empleado. Para recuperar una lista de empleados con el nombre de su jefe respectivo, la tabla se puede unir sobre sí misma utilizando esta asignación. Tenga en cuenta que unir una tabla de esta manera requiere el uso de un alias (`Bosses` en este caso) en la segunda referencia a la tabla para distinguirse de la tabla original.

```
SELECT Employees.Name,
       Bosses.Name AS Boss
FROM Employees
INNER JOIN Employees AS Bosses
       ON Employees.Boss_ID = Bosses.ID
```

La ejecución de esta consulta producirá los siguientes resultados:

Nombre	Jefe
Mover	Sam
Jim	Mover
Sam	Jim

Eliminar usando Join

Las combinaciones también se pueden utilizar en una sentencia `DELETE` . Dado un esquema de la siguiente manera:


```

CREATE TABLE Users (
    UserId int NOT NULL,
    AccountId int NOT NULL,
    RealName nvarchar(200) NOT NULL
)

CREATE TABLE Preferences (
    UserId int NOT NULL,
    SomeSetting bit NOT NULL
)

```

Podemos eliminar filas de la tabla de `Preferences` , filtrando por un predicado en la tabla de `Users` siguiente manera:

```

DELETE p
FROM Users u
INNER JOIN Preferences p ON u.UserId = p.UserId
WHERE u.AccountId = 1234

```

Aquí `p` es un alias para las `Preferences` definidas en la cláusula `FROM` de la declaración y solo eliminamos las filas que tienen un `AccountId` coincidente de la tabla `Users` .

Accidentalmente convirtiendo una unión externa en una unión interna

Las combinaciones externas devuelven todas las filas de una o ambas tablas, más las filas coincidentes.

```

Table People
PersonID FirstName
    1 Alice
    2 Bob
    3 Eve

Table Scores
PersonID Subject Score
    1 Math    100
    2 Math    54
    2 Science 98

```

A la izquierda uniéndose a las mesas:

```

Select * from People a
left join Scores b
on a.PersonID = b.PersonID

```

Devoluciones:

```

PersonID FirstName PersonID Subject Score
    1 Alice          1 Math    100
    2 Bob            2 Math    54
    2 Bob            2 Science 98
    3 Eve            NULL NULL  NULL

```

Si desea devolver a todas las personas, con cualquier puntaje de matemáticas aplicable, un error común es escribir:

```
Select * from People a
left join Scores b
on a.PersonID = b.PersonID
where Subject = 'Math'
```

Esto eliminaría a Eve de sus resultados, además de eliminar el puntaje de ciencia de Bob, ya que Subject **es** NULL para ella.

La sintaxis correcta para eliminar registros no matemáticos y conservar a todos los individuos en la tabla People sería:

```
Select * from People a
left join Scores b
on a.PersonID = b.PersonID
and b.Subject = 'Math'
```

Lea Unirse en línea: <https://riptutorial.com/es/sql-server/topic/1008/unirse>

Capítulo 111: Uso de la tabla TEMP

Observaciones

Las tablas temporales son realmente muy útiles.

La tabla se puede crear en tiempo de ejecución y puede realizar todas las operaciones que se realizan en una tabla normal.

Estas tablas se crean en una base de datos tempdb.

Utilizado cuando?

1. Tenemos que hacer una operación de unión compleja.
2. Hacemos gran cantidad de manipulación de filas en procedimientos almacenados.
3. Puede reemplazar el uso del cursor.

Así aumenta el rendimiento.

Examples

Tabla de temperatura local

- Estará disponible hasta que la conexión actual persista para el usuario.

Se elimina automáticamente cuando el usuario se desconecta.

El nombre debe comenzar con # (#temp)

```
CREATE TABLE #LocalTempTable (  
    StudentID      int,  
    StudentName    varchar(50),  
    StudentAddress varchar(150))
```

```
insert into #LocalTempTable values ( 1, 'Ram','India');  
  
select * from #LocalTempTable
```

Después de ejecutar todas estas declaraciones, si cerramos la ventana de consulta y la abrimos de nuevo e intentamos insertarla y seleccionarla, aparecerá un mensaje de error.

```
"Invalid object name #LocalTempTable"
```

Tabla de temperatura global

- Comenzará con ## (## temp).

Se eliminará solo si el usuario desconecta todas las conexiones.

Se comporta como una mesa permanente.

```
CREATE TABLE ##NewGlobalTempTable(  
    StudentID      int,  
    StudentName    varchar(50),  
    StudentAddress varchar(150))  
  
Insert Into ##NewGlobalTempTable values ( 1, 'Ram', 'India');  
Select * from ##NewGlobalTempTable
```

Nota: Todos los usuarios de la base de datos pueden verlos, independientemente del nivel de permisos.

Tablas de temperatura de caída

Las tablas temporales deben tener ID únicos (dentro de la sesión, para tablas temporales locales, o dentro del servidor, para tablas temporales globales). Intentar crear una tabla con un nombre que ya existe devolverá el siguiente error:

```
There is already an object named '#tempTable' in the database.
```

Si su consulta produce tablas temporales y desea ejecutarla más de una vez, deberá eliminar las tablas antes de intentar generarlas nuevamente. La sintaxis básica para esto es:

```
drop table #tempTable
```

Intentar ejecutar esta sintaxis antes de que exista la tabla (por ejemplo, en la primera ejecución de su sintaxis) causará otro error:

```
Cannot drop the table '#tempTable', because it does not exist or you do not have permission.
```

Para evitar esto, puedes verificar si la tabla ya existe antes de soltarla, así:

```
IF OBJECT_ID ('tempdb..#tempTable', 'U') is not null DROP TABLE #tempTable
```

Lea **Uso de la tabla TEMP en línea**: <https://riptutorial.com/es/sql-server/topic/5328/uso-de-la-tabla-temp>

Capítulo 112: Utilidad bcp (programa de copia masiva)

Introducción

La utilidad del programa de copia masiva (bcp) copia datos en masa entre una instancia de Microsoft SQL Server y un archivo de datos en un formato especificado por el usuario. La utilidad bcp se puede usar para importar grandes cantidades de filas nuevas en tablas de SQL Server o para exportar datos de tablas a archivos de datos.

Examples

Ejemplo para importar datos sin un archivo de formato (usando formato nativo)

```
REM Truncate table (for testing)
SQLCMD -Q "TRUNCATE TABLE TestDatabase.dbo.myNative;"

REM Import data
bcp TestDatabase.dbo.myNative IN D:\BCP\myNative.bcp -T -n

REM Review results
SQLCMD -Q "SELECT * FROM TestDatabase.dbo.myNative;"
```

Lea Utilidad bcp (programa de copia masiva) en línea: <https://riptutorial.com/es/sql-server/topic/10942/utilidad-bcp--programa-de-copia-masiva->

Capítulo 113: Variables

Sintaxis

- DECLARE @VariableName DataType [= Value];
- SET @VariableName = Valor;

Examples

Declarar una variable de tabla

```
DECLARE @Employees TABLE
(
    EmployeeID INT NOT NULL PRIMARY KEY,
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    ManagerID INT NULL
)
```

Cuando creas una tabla normal, usas la sintaxis de `CREATE TABLE Name (Columns)`. Al crear una variable de tabla, utiliza la `DECLARE @Name TABLE (Columns)`.

Para hacer referencia a la variable de tabla dentro de una instrucción `SELECT`, SQL Server requiere que le dé un alias a la variable de tabla, de lo contrario obtendrá un error:

Debe declarar la variable escalar "@TableVariableName".

es decir

```
DECLARE @Table1 TABLE (Example INT)
DECLARE @Table2 TABLE (Example INT)

/*
-- the following two commented out statements would generate an error:
SELECT *
FROM @Table1
INNER JOIN @Table2 ON @Table1.Example = @Table2.Example

SELECT *
FROM @Table1
WHERE @Table1.Example = 1
*/

-- but these work fine:
SELECT *
FROM @Table1 T1
INNER JOIN @Table2 T2 ON T1.Example = T2.Example

SELECT *
FROM @Table1 Table1
WHERE Table1.Example = 1
```

Actualizando una variable usando SET

```
DECLARE @VariableName INT
SET @VariableName = 1
PRINT @VariableName
```

1

Usando `SET` , solo puedes actualizar una variable a la vez.

Actualizando variables usando SELECT

Usando `SELECT` , puedes actualizar múltiples variables a la vez.

```
DECLARE @Variable1 INT, @Variable2 VARCHAR(10)
SELECT @Variable1 = 1, @Variable2 = 'Hello'
PRINT @Variable1
PRINT @Variable2
```

1

Hola

Cuando se utiliza `SELECT` para actualizar una variable de una columna de la tabla, si hay varios valores, utilizará el *último* valor. (Se aplican las reglas de orden normales: si no se da una clasificación, el orden no está garantizado).

```
CREATE TABLE #Test (Example INT)
INSERT INTO #Test VALUES (1), (2)

DECLARE @Variable INT
SELECT @Variable = Example
FROM #Test
ORDER BY Example ASC

PRINT @Variable
```

2

```
SELECT TOP 1 @Variable = Example
FROM #Test
ORDER BY Example ASC

PRINT @Variable
```

1

Si no hay filas devueltas por la consulta, el valor de la variable no cambiará:

```
SELECT TOP 0 @Variable = Example
```

```
FROM #Test
ORDER BY Example ASC

PRINT @Variable
```

1

Declara múltiples variables a la vez, con valores iniciales.

```
DECLARE
    @Var1 INT = 5,
    @Var2 NVARCHAR(50) = N'Hello World',
    @Var3 DATETIME = GETDATE()
```

Operadores de asignación de compuestos

SQL Server 2008 R2

Operadores compuestos compatibles:

+= Añadir y asignar

-= Restar y asignar

***=** Multiplica y asigna

/= Divide y asigna

%= Modulo y asignar

&= Bitwise AND y asignar

^= Bitwise XOR y asignar

|= Bitwise OR y asignar

Ejemplo de uso:

```
DECLARE @test INT = 42;
SET @test += 1;
PRINT @test;    --43
SET @test -= 1;
PRINT @test;    --42
SET @test *= 2
PRINT @test;    --84
SET @test /= 2;
PRINT @test;    --42
```

Actualizando variables seleccionando desde una tabla

Dependiendo de la estructura de sus datos, puede crear variables que se actualicen dinámicamente.

```
DECLARE @CurrentID int = (SELECT TOP 1 ID FROM Table ORDER BY CreateDate desc)

DECLARE @Year int = 2014
DECLARE @CurrentID int = (SELECT ID FROM Table WHERE Year = @Year)
```

En la mayoría de los casos, querrá asegurarse de que su consulta devuelva solo un valor al usar este método.

Lea Variables en línea: <https://riptutorial.com/es/sql-server/topic/2566/variables>

Creditos

S. No	Capítulos	Contributors
1	Empezando con Microsoft SQL Server	Abhilash R Vankayala , Abhishek Jain , Ahmad Aghazadeh , Ahmar , Akshay Anand , alalp , Almir Vuk , Arthur D , ATC , Athafoud , BeaglesEnd , Bhanu , Biju jose , Blachshma , bluefeet , ChrisM , Christos , Community , cteski , D M , Darshak , Gidil , Gordon Bell , Greg Bray , Iztoksson , Jared Hooper , JerryOL , Job AJ , Joe Taras , John Odom , John Slegers , JonasCz , K48 , kafka , Lamak , Laughing Vergil , Mahesh Dahal , Malt , Martin Smith , Matt , Matt , Max , Mihai-Daniel Virna , Mudassir Hasan , n00b , Nick , Nikolay Kostov , onupdatecascade , OzrenTkalcecKrznic , Peter Tirrell , Phrancis , Prateek , Sam , Shaneis , Thuta Aung , Tony L. , Tot Zam , Uberzen1 , Umachandar - Microsoft , user_0 , user2314737 , VoidDemon , Zsuzsa
2	Administrar la base de datos SQL de Azure	Jovan MSFT
3	Agente de servicios	Ken S. , Matej , RamenChef
4	AGRUPAR POR	Andy , Edathadan Chief aka Arun , Jenism , juergen d , Julien Vavasseur , Kiran Ukande , Matas Vaitkevicius , ShlomiR
5	Almacenamiento de JSON en tablas SQL	Ed Harper , Jovan MSFT , RamenChef
6	Analizando una consulta	DForck42
7	aplicación cruzada	Hamza Rabah , Jovan MSFT , Tom V
8	Base de datos del sistema - TempDb	Anuj Tripathi , RamenChef
9	Cifrado	Rubenisme
10	Columnas calculadas	cnayak , Kannan Kandasamy
11	COLUMNSTORE CLUSTRADO	Jovan MSFT
12	Con la opción de	TheGameiswar

corbatas		
13	Conjunto de resultados de límite	alalp , chrisb , cteski , EriKE
14	Consejos de consulta	cteski , DARKOCEAN , Jovan MSFT , user_0
15	Consulta de resultados por página.	Pat
16	Consultas con datos JSON	bakedpatato , James , Jovan MSFT
17	Convertir tipos de datos	Ben O , Edathadan Chief aka Arun
18	Copia de seguridad y restauración de base de datos	Jones Joseph , Jovan MSFT
19	CREAR VISTA	Almir Vuk , cteski , Edathadan Chief aka Arun , Hadi , Josh B , Robert Columbia , Tot Zam
20	Cursores	Kane , Phrancis
21	Datos espaciales	cteski , Neil Kennedy , RamenChef , Vladimir Oselsky
22	DBCC	Jovan MSFT
23	DBMAIL	Phrancis
24	Declaración de caso	Laughing Vergil , RamenChef , Vikas Vaidya
25	Delimitando caracteres especiales y palabras reservadas.	bassrek
26	Desencadenar	Oluwafemi , The_Outsider , Zohar Peled
27	Eliminar palabra clave	Ignas , Jakub Ojmucianski , Justin Rohr , Max , scsimon
28	En memoria OLTP (Hekaton)	Akshay Anand , Behzad , Brandon , Jovan MSFT , Martijn Pieters
29	Enmascaramiento dinámico de datos	Jovan MSFT

30	Esquemas	Merenix
31	Exportar datos en el archivo txt utilizando SQLCMD	sheraz mirza
32	Expresiones de mesa comunes	Arif , bbrown , cteski , DForck42 , Jeffrey Van Laethem , Jovan MSFT , kafka , Keith Hall , Monty Wild , SQLMason
33	fechas	A_Arnold , Adam Porad , Akshay Anand , Bellash , cteski , Edathadan Chief aka Arun , JamieA , Jared Hooper , Kritner , Lamak , Mert Gülsoy , Nick , Phrancis , SHD , Siyual , Soukai , UnhandledExcepSean , Zohar Peled
34	Filestream	Raghu Ariga
35	Función de cadena dividida en el servidor SQL	Jibin Balachandran , Jovan MSFT , MasterBob , பரத் ப் , RamenChef
36	Funciones agregadas	Akshay Anand , cnayak , cteski , Jeffrey L Whittedge , Joe Taras , Vexator
37	Funciones agregadas de cadenas en SQL Server	Kannan Kandasamy
38	Funciones de cadena	A_Arnold , anon , cteski , FoxyBOA , Hadi , hatchet , Igor Micev , Jibin Balachandran , Jovan MSFT , mtb , Phrancis , Raidri , Ricardo C , Ross Presser , takrl , Zohar Peled
39	Funciones de clasificación	cteski , kolunar , New
40	Funciones de ventana	andyabel , feetwet , MarmiK
41	Funciones logicas	dd4711
42	Generando un rango de fechas	James , Siyual
43	Gobernador de recursos	Ako , RamenChef
44	Grupo de archivos	Behzad
45	Importación a granel	Jovan MSFT

46	Índice	Ahmad Aghazadeh , Akshay Anand , cteski , Henrik Staun Poulsen , Martin Smith , Tom V
47	Indización de texto completo	Edathadan Chief aka Arun
48	Insertar	Randall
49	INSERTAR EN	Abubakar Riaz , barcanoj , DVJex , Hari K M , intox , martinshort , Matas Vaitkevicius , Max , Michael Stum , n00b , Piotr Nawrot , Robert Columbia , Tot Zam , woony
50	Instalar SQL Server en Windows	Luis Bosquez
51	Instantáneas de la base de datos	Akash , Daryl , Jovan MSFT , Wolfgang
52	Instrucción SELECT	cteski , Jovan MSFT
53	Integración de Common Language Runtime	Jovan MSFT
54	JSON en Sql Server	Jovan MSFT , Mono
55	JUNTARSE	Bharat Prasad Satyal , Edathadan Chief aka Arun , Karthikeyan , Matej , scsimon , Tab Alleman
56	La función COSAS	Arthur D , bluefeet , Chetan Sanghani , dacoheii , Kiran Ukande , Luis Bosquez , MrE , user1690166
57	Lectura fantasma	Max
58	Llaves extranjeras	Jovan MSFT
59	Llaves primarias	Kritner
60	Manejo de transacciones	Metanormal
61	Mientras bucle	lord5et , Matas Vaitkevicius , podiluska , RamenChef , Wojciech Kazior
62	Migración	Matas Vaitkevicius
63	Modificar texto JSON	Jovan MSFT
64	Módulos compilados de forma nativa	bakedpatato , Jovan MSFT

(Hekaton)		
65	Mueve y copia datos alrededor de tablas	Nick.McDermaid
66	Niveles de aislamiento de transacciones.	Phrancis
67	Niveles de aislamiento y bloqueo.	RamenChef , sqlandmore.com
68	Nombres de alias en el servidor SQL	பரத் ப்
69	NULLs	Amir Pourmand , Hadi , Kannan Kandasamy , Kritner , Laughing Vergil , podiluska , Sean Branchaw , Zohar Peled
70	Opciones avanzadas	Ahmad Aghazadeh
71	OPENJSON	James , Jovan MSFT
72	Operaciones básicas de DDL en MS SQL Server	Matt
73	ORDEN POR	APH , beercohol , cteski , Gidil , RamenChef
74	Ordenar / ordenar filas	APH , OzrenTkalcecKrznic
75	Paginación	cteski , Jovan MSFT , Sender
76	PARA JSON	James , Jovan MSFT
77	PARA XML PATH	bluefeet , gotqn , Keith Hall , Wolfgang
78	Parámetros de tabla de valores	Zohar Peled
79	ParseName	Mani
80	Particionamiento	Dan Guzman , James Anderson , John Odom , Susang
81	Permisos de base de datos	Ben Thul
82	Permisos y seguridad	5arx

83	Pivot dinámico de SQL	Jesse
84	PIVOTE / UNPIVOT	Athafoud , bluefeet , DhruvJoshi , kolunar
85	Privilegios o permisos	Oluwafemi
86	Procedimientos almacenados	Bino Mathew Varghese , cnayak , cteski , Erik Oppedijk , Eugene Niemand , Hari K M , Jayasurya Satheesh , Matas Vaitkevicius , Nathan Skerl , Pirate X , scsimon
87	Puntos de vista	Benjamin Hodgson , Daniel Lemke , Max
88	Recupera información sobre tu instancia	Bino Mathew Varghese , feetwet , James Anderson , Kritner , LowlyDBA , S.Karras , scsimon
89	Recuperar información sobre la base de datos.	Andrea , Anuj Tripathi , Baodad , Brent Ozar , dario , feetwet , James Anderson , JamieA , Jasmin Solanki , Jeffrey Van Laethem , jyao , Kritner , Laughing Vergil , LowlyDBA , Mahendra , Moshiour , Phrancis , Rhumborl , scsimon , Shiva , spaghettidba , Tot Zam , TZHX , Umachandar - Microsoft
90	SCOPE_IDENTITY ()	Dheeraj Kumar
91	Secuencias	Josh Morel
92	Seguridad a nivel de fila	Carsten Hynne , Jovan MSFT
93	Si ... otra cosa	cteski , M.Ali , RamenChef
94	Sobre la cláusula	Athafoud , bluefeet , Brandon , DVT , gofr1 , Lamak , Paul Bambury , RamenChef , Rowland Shaw , Sam
95	SQL dinámico	Jovan MSFT
96	SQL Server Evolution a través de diferentes versiones (2000 - 2016)	Dan Guzman , M.Ali
97	SQL Server Management Studio (SSMS)	dd4711
98	SQLCMD	Eugene Niemand , Techie

99	Subconsultas	cnayak
100	Tablas temporales	Ahmad Aghazadeh , Akshay Anand , Ben O , Mspaja
101	Tarea o trabajo programado	Edathadan Chief aka Arun , Hadi
102	Teclas de acceso directo de Microsoft SQL Server Management Studio	Bino Mathew Varghese , cteski , Sibeesh Venu
103	Tienda de consultas	bakedpatato , Jovan MSFT
104	Tipos de datos	Laughing Vergil , Matas Vaitkevicius
105	Tipos de tablas definidas por el usuario	Jivan , Zohar Peled
106	TRATA DE ATRAPARLO	Jovan MSFT , ravindra , Uberzen1
107	Última identidad insertada	Jeffrey Van Laethem , sqluser , Tot Zam
108	UNIÓN	cnayak
109	UNIR	Abhilash R Vankayala , Abubakar Riaz , Alex , David Kaminski , dd4711 , Hari K M , Moshiour , Rogerio Soares , Serg
110	Unirse	4444 , Akshay Anand , Andy , APH , Bino Mathew Varghese , cteski , Dean Ward , DhruvJoshi , Dileep , Gajendra , HK1 , Iztoksson , Jeffrey Van Laethem , Joao Araujo , JonH , L J , Lamak , Laughing Vergil , LowlyDBA , mtb , Nikolay Kostov , OzrenTkalceckKrznaric , Phrancis , Ram Grandhi , SqlZim
111	Uso de la tabla TEMP	APH , New
112	Utilidad bcp (programa de copia masiva)	MarmiK
113	Variables	APH , Keith Hall , Phrancis