

TRANSACTION LOG

MS SQL SERVER

**Rafael Ernesto Sequeiros Alvarado**

Proyecto de Teoría de Base de Datos II:

Ing. Elvin Deras

San Pedro Sula, 24 de Agosto 2015

# **Introducción**

El registro de transacciones de SQL Server contiene la historia de cada acción que modifica cualquier cosa en la base de datos. Leyendo el registro es a menudo el último recurso cuando la investigación de cómo se produjeron ciertos cambios. Es una de las principales herramientas forenses a su disposición cuando tratan de identificar al autor de un cambio no deseado. Entender el registro y la excavación a través de él para obtener información es el núcleo muy duro y definitivamente no es para los débiles de corazón. Y el hecho de que la salida de :: fn\_dblog puede ir fácilmente en millones de filas no ayuda tampoco. Pero voy a tratar de dar algunos ejemplos prácticos sencillos que pueden ir una manera larga en ayudar a ordenar a través de toda la información y cavar lo que le interesa.

# **Objetivos**

## Objetivo general:

Poder comprender el Transacción Log en MSS SQL SERVER y saber su importancia para poder rollback tanto en Delete, Insert, Update y poder recuperar un estado de la base de datos que consideremos correcto.

## Objetivos específicos:

* Poder romper el [RowLog Content 0] teniendo conocimiento de su estructura.
* Poder realizar Conversiones de hexadecimal a los 12 datos especificados en el informe

# **Log de transacción de MS SQL SERVER**

Cada base de datos SQL Server tiene un registro de transacciones que registra todas las transacciones y las modificaciones de base de datos realizadas por cada transacción. El registro de transacciones debe ser truncado de forma regular para evitar que se llene. Sin embargo, algunos factores que pueden retrasar el truncamiento del registro, por lo que el tamaño del registro de monitoreo es importante. Algunas operaciones pueden ser mínimamente registran para reducir su impacto en el tamaño del registro de transacciones.

El registro de transacciones es un componente crítico de la base de datos y, si hay una falla en el sistema, el registro de transacciones puede ser necesario para traer a su base de datos a un estado coherente. El registro de transacciones nunca debe ser eliminado o movido a menos que usted entiende completamente las ramificaciones de hacer esto.

**La importancia de un Truncamiento en el Transacción Log**

El truncamiento del registro libera espacio en el archivo de registro para su reutilización por el registro de transacciones. Truncamiento de registro es esencial para mantener el registro de llenado. El truncamiento del registro elimina los archivos de registro virtuales inactivos desde el registro de transacciones lógico de una base de datos SQL Server, liberando espacio en el registro lógico para su reutilización por el registro de transacciones físicas. Si un registro de transacciones no se trunca, sería finalmente llenar todo el espacio en disco que se asigna a los archivos de registro físicos.

Para evitar este problema, a menos que el truncamiento del registro se está retrasando por alguna razón, el truncamiento se produce automáticamente después de los siguientes eventos:

Bajo el modelo de recuperación simple, después de un puesto de control.

Bajo el modelo de recuperación completa o el modelo de recuperación de registro masivo, si un puesto de control se ha producido desde la copia de seguridad anterior, el truncamiento se produce después de una copia de seguridad del registro (a menos que sea una única copia BACKUP LOG).

**¿Cómo acceder al Log de Transacciones?**

Tenemos que conseguir los registros borrados de sql server. Al utilizar el fn\_blog función de SQL Server estándar, podemos obtener fácilmente todo registro de transacciones (incluyendo borrado de datos Pero, necesitamos sólo los registros eliminados seleccionados del registro de transacciones. Así que hemos incluido tres filtros (Context, Operation & AllocUnitName).

El select especifico que usaremos en el proyecto será "SELECT [Transaction ID], [RowLog Contents 0] FROM ::fn\_dblog(NULL,NULL) where AllocUnitName = 'dbo.dateTimeTable' AND Context IN('LCX\_MARK\_AS\_GHOST', 'LCX\_HEAP') AND Operation in('LOP\_DELETE\_ROWS').

**Un select asterisco a fn\_log**

**Unos de los campos mas importantes que podria traernos un Select\*From** fn\_dblog(NULL,NULL contendria

|  |
| --- |
| Current LSN |
| Operation |
| Context |
| Transaction ID |
| Tag Bits |
| Log Record Fixed Length |
| Flag Bits |
| AllocUnitId |
| AllocUnitName |
| Description |
| RowLog Contents 0 |
| RowLog Contents 1 |
| RowLog Contents 2 |
| RowLog Contents 3 |
| RowLog Contents 4 |

# **Cómo podemos romper el RowLog Contents 0 desde fn\_blog**

1. Necesitamos obtener los registros borrados de sql server. Mediante el uso de la función estándar de fn\_blog SQL Server, podemos conseguir fácilmente todo el registro de transacciones (incluyendo borrado de datos. Pero, necesitamos sólo los registros eliminados seleccionados del registro de transacciones.

Ej:

Select [RowLog Contents 0] FROM sys.fn\_dblog(NULL,NULL) where AllocUnitName =‘dbo.Student’ AND Context IN(‘LCX\_MARK\_AS\_GHOST’, ‘LCX\_HEAP’) AND Operation in(‘LOP\_DELETE\_ROWS’)

Esta consulta devolverá el número de columnas que proporcionan información diferente, pero sólo tenemos que seleccionar la columna "RowLog contenido o, para obtener los datos eliminados.

La Columna "RowLog content 0" se verá algo así:“0x300018000100000000000000006B000056492020590000000500E001002800426F62206A65727279″

1. Ahora, hemos borrado datos, pero en valores hexadecimales, pero SQL mantiene estos datos en una secuencia específica para que podamos recuperarla fácilmente. El formato está definido de la siguiente manera:

* 1 Byte : Estado del Bit A
* 1 Byte : Estado del Bit B
* 2 Bytes : Tamaño de longitud fija
* n Bytes : Datos de longitud fija
* 2 Bytes : Número total de columnas
* n Bytes : NULL Bitmap (1 bit por cada columna , 1 indica que la columna esta nula y 0 indica que la columna no está nula
* 2 Bytes : Número de columnas de longitud variable
* n Bytes : Matriz de desplazamiento de columna (2x columna de longitud variable)
* n Bytes : Los datos de las columnas de longitud variable

Así que los datos hexadecimales de "RowLog content 0" es igual a:

“Estado del Bit A + Estado del Bit B + Tamaño de longitud fija + Datos de longitud fija + Número total de columnas + NULL Bitmap + Número

**3** Ahora, tenemos que romper el RowLog o contenido (valor hexadecimal de nuestros datos borrados) en la estructura definida anteriormente. [Códigos de color se utilizan únicamente como referencia]

[Longitud fija de datos] = substring (RowLog contenido 0, Status Bit Bit A + B + 1,2 Estado bytes)

[Total No de Columnas] = Subcadena (contenido RowLog 0, [longitud fija de datos] + 1,2 bytes)

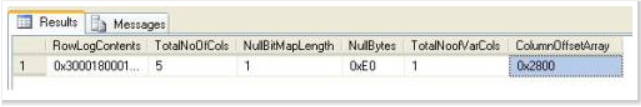
[Longitud Null Bitmap] = Techo ([Total No de Columnas] /8.0)

[Nulos Bytes] = Subcadena (RowLog contenido de 0, Status Bit A + B + Status Bit [Longitud fija de datos] 1, [longitud Null Bitmap])

Nº total de columnas variable = Subcadena (RowLog content 0, Status Bit A + B + Status Bit [Longitud fija de datos] 1, [Null Bitmap longitud] + 2)

Columna Offset Array = Subcadena (RowLog contenido de 0, Status Bit A + B + Status Bit [Longitud fija de datos] 1, [Null Bitmap longitud] + 2, Nº total de columnas variables \* 2)

Columna Variable Start = Estado Bit A + B + Status Bit [Longitud fija de datos] + [Null Bitmap longitud] + 2 + (Nº total de columnas variables \* 2).



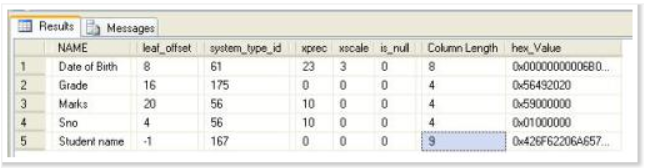
**4**. Ahora, tenemos los datos divididos, para que así podamos encontrar si el valor de la columna es nulo o no, mediante el uso de Null Bytes. Para lograrlo convertir Null Bytes (valor hexadecimal) en formato binario.

****

**5**. Ahora, así tenemos la división de datos principales (paso 3) y los valores nulos (paso 4). Después de eso tenemos que utilizar algunos fragmentos de código para obtener los datos de columna: como nombre de la columna, el tamaño de la columna, la precisión, la escala y lo más importante la hoja poco nulo (para asegurar que la columna se fija de datos (<= - 1) o tamaños de datos variables (> = 1)) de la tabla. Aquí algunos de los códigos que se pueden utilizar:

Select \* from sys.allocation\_units allocunits INNER JOIN sys.partitions partitions ON (allocunits.type IN (1, 3) AND partitions.hobt\_id = allocunits.container\_id) OR (allocunits.type = 2 AND partitions.partition\_id = allocunits.container\_id) INNER JOIN sys.system\_internals\_partition\_columns cols ON cols.partition\_id = partitions.partition\_id LEFT OUTER JOIN syscolumns ON syscolumns.id = partitions.object\_id AND syscolumns.colid = cols.partition\_column\_id

And join it with our collected data table (Step-1,2,3,4) on the basis of allocunits.[Allocation\_Unit\_Id].Till Ahora sabemos que la información sobre la tabla y los datos, por lo que necesitamos para utilizar estos datos para romper [RowLog Cesta 0] en datos de las columnas de la tabla pero en valor hexadecimal. Aquí tenemos que tener cuidado ya que los datos están ya sea en tamaño de columna fija o en tamaño de columna variable.



--NVARCHAR ,NCHAR

WHEN system\_type\_id IN (231, 239) THEN LTRIM(RTRIM(CONVERT(NVARCHAR(max),hex\_Value)))

--VARCHAR,CHAR

WHEN system\_type\_id IN (167,175) THEN LTRIM(RTRIM(CONVERT(VARCHAR(max),REPLACE(hex\_Value, 0x00, 0x20))))

--TINY INTEGER

WHEN system\_type\_id = 48 THEN CONVERT(VARCHAR(MAX), CONVERT(TINYINT, CONVERT(BINARY(1), REVERSE (hex\_Value))))

--SMALL INTEGER

WHEN system\_type\_id = 52 THEN CONVERT(VARCHAR(MAX), CONVERT(SMALLINT, CONVERT(BINARY(2), REVERSE (hex\_Value))))

-- INTEGER

WHEN system\_type\_id = 56 THEN CONVERT(VARCHAR(MAX), CONVERT(INT, CONVERT(BINARY(4), REVERSE(hex\_Value))))

-- BIG INTEGER

WHEN system\_type\_id = 127 THEN CONVERT(VARCHAR(MAX), CONVERT(BIGINT, CONVERT(BINARY(8), REVERSE(hex\_Value))))

--DATETIME

WHEN system\_type\_id = 61 Then CONVERT(VARCHAR(Max),CONVERT(DATETIME,Convert(VARBINARY(max),REVERSE (hex\_Value))),100)

--SMALL DATETIME

WHEN system\_type\_id =58 Then CONVERT(VARCHAR(Max),CONVERT(SMALLDATETIME,CONVERT(VARBINARY(MAX),REVERSE(hex\_Value))),100) --SMALL DATETIME

--- NUMERIC

WHEN system\_type\_id = 108 THEN CONVERT(VARCHAR(MAX), CAST(CONVERT(NUMERIC(18,14), CONVERT(VARBINARY,CONVERT(VARBINARY,xprec)+CONVERT(VARBINARY,xscale))+CONVERT(VARBINARY(1),0) + hex\_Value) as FLOAT))

--MONEY,SMALLMONEY

WHEN system\_type\_id In(60,122) THEN CONVERT(VARCHAR(MAX),Convert(MONEY,Convert(VARBINARY(MAX),Reverse(hex\_Value))),2)

--- DECIMAL

WHEN system\_type\_id = 106 THEN CONVERT(VARCHAR(MAX), CAST(CONVERT(Decimal(38,34), Convert(VARBINARY,Convert(VARBINARY,xprec)+CONVERT(VARBINARY,xscale))+CONVERT(VARBINARY(1),0) + hex\_Value) as FLOAT))

-- BIT

WHEN system\_type\_id = 104 THEN CONVERT(VARCHAR(MAX),CONVERT (BIT,CONVERT(BINARY(1), hex\_Value)%2))

--- FLOAT

WHEN system\_type\_id = 62 THEN RTRIM(LTRIM(Str(Convert(FLOAT,SIGN(CAST(Convert(VARBINARY(max),Reverse(hex\_Value)) AS BIGINT)) \* (1.0 + (CAST(CONVERT(VARBINARY(max),Reverse(hex\_Value)) AS BIGINT) & 0x000FFFFFFFFFFFFF) \* POWER(CAST(2 AS FLOAT), -52)) \* POWER(CAST(2 AS FLOAT),((CAST(CONVERT(VARBINARY(max),Reverse(hex\_Value)) AS BIGINT) & 0x7ff0000000000000) / EXP(52 \* LOG(2))-1023))),53,LEN(hex\_Value))))

--REAL

When system\_type\_id =59 THEN Left(LTRIM(STR(Cast(SIGN(CAST(Convert(VARBINARY(max),Reverse(hex\_Value)) AS BIGINT))\* (1.0 + (CAST(CONVERT(VARBINARY(max),Reverse(hex\_Value)) AS BIGINT) & 0x007FFFFF) \* POWER(CAST(2 AS Real), -23)) \* POWER(CAST(2 AS Real),(((CAST(CONVERT(VARBINARY(max),Reverse(hex\_Value)) AS INT) )& 0x7f800000)/ EXP(23 \* LOG(2))-127))AS REAL),23,23)),8)

--BINARY,VARBINARY

WHEN system\_type\_id In (165,173) THEN (CASE WHEN Charindex(0x,cast('' AS XML).value('xs:hexBinary(sql:column("hex\_value"))', 'varbinary(max)')) = 0 THEN '0x' ELSE '' END) +cast('' AS XML).value('xs:hexBinary(sql:column("hex\_value"))', 'varchar(max)')

# **Conclusiones**

* He comprendido en lo que consiste en Log de transacciones de MS SQL SERVER, para que funciona, en que nos beneficia el poder manejar este archivo.
* Se logró comprender que es lo que conlleva y maneja el RowLog Content 0
* Se ha logrado comprender y aprender a manejar cierta información del fn\_dblog tales como ser id de la transacción, tabla afectada, etc.

# **Recomendaciones**

* Al utilizar la función no documentada de fn\_dblog debemos de tener el debido cuidado de asegurarnos que la base de datos esté usando la opción de seguimiento de cambios completa.
* Tomar en cuenta que debe saber primero la estructura de su tabla antes de romper el hexadecimal.

# **Bibliografía**

# http://raresql.com/2011/10/22/how-to-recover-deleted-data-from-sql-sever/

http://www.mssqltips.com/sqlservertip/3076/how-to-read-the-sql-server-database-transaction-log/