UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA INFORMÁTICA DE GESTÍON

PROYECTO FINAL DE CARRERA

ANÁLISIS, DESARROLLO E IMPLEMENTACIÓN DE AUDITORÍA EN LA BASE DE DATOS MICROSOFT SQL SERVER 2005

AUTOR: CLEMENTE GONZÁLEZ PUERTO

TUTOR: ELENA CASTRO GALÁN

CO-DIRECTOR: IGNACIO-J SANTOS FORNER

MARZO 2010

AGRADECIMIENTOS

Siempre he tenido la ilusión de hacer una carrera universitaria, y después de un camino difícil pero muy satisfactorio he conseguido llegar a la meta. Pero no he estado sólo, así que quiero aprovechar esta página para expresar mi agradecimiento a todas las personas que han estado conmigo.

Tuve la fortuna de conocer a Ignacio Santos hace unos años, y digo fortuna porque sin su interés, apoyo y conocimientos me hubiera sido muy difícil acabar este proyecto. Gracias Profesor.

A Olga, Carmen, Cristina, Dani, Ana, Ani, habéis estado siempre a mi lado queriéndome, comprendiéndome y apoyándome. Sin vosotros no lo hubiera conseguido, así que este trabajo también es vuestro.

No me quiero olvidar de mis profesores, compañeros y amigos de estos años, cada uno de vosotros me ha enseñado algo y siempre estará en mí.

La mayor sabiduría que existe es conocerse a uno mismo.

Galileo Galilei (1564-1642) Físico y astrónomo italiano.

1 Contenido

2 Objetivo del Proyecto.

Tenemos como objetivo estudiar la monitorización y el rendimiento de una instalación de base de datos de *Microsoft SQL Server 2005* y la aplicación de la nueva tecnología llamada *Service Broker,* integrada en el sistema gestor de base de datos y cuya principal función es proporcionar colas de mensajes. Además, queremos auditar el funcionamiento del SGBD *Microsoft SQL Server 2005,* creando una serie buenas prácticas que permitan obtener unos resultados que nos lleven a tomar las mejores decisiones para que el rendimiento del SGBD y las instancias de bases de datos manejadas sean óptimos. En definitiva, tenemos ante nosotros el reto de conocer y manejar el SGBD *Microsoft SQL Server 2005* y las herramientas que dispone para realizar auditorías midiendo su comportamiento para determinar las ventajas y deficiencias en las distintas maneras de realizar auditoría que vamos a estudiar. *Microsoft SQL Server* es un sistema de gestión de bases de datos relacionales (SGBD), capaz de poner a disposición de mucho usuarios grandes cantidades de datos de manera simultánea. *Microsoft SQL Server* constituye la alternativa de *Microsoft* a otros potentes sistemas gestores de base de datos como son *Oracle, DB2, Sybase, Interbase, Firebird o MySQL.*

Microsoft SQL Server soporta transacciones; dispone del lenguaje *Transact SQL* como herramienta de desarrollo y explotación; tiene la ventaja de la escalabilidad, estabilidad y seguridad; puede soportar procedimientos almacenados; permite trabajar en modo cliente-servidor, donde la información y datos se alojan en el servidor y los clientes de la red sólo acceden a la información; aporta la posibilidad de administrar información de otros servidores de datos y además incluye un potente entorno gráfico de administración y monitorización. Gracias a estas y otra característica y ventajas, *Microsoft SQL Server* en la versión 2005 se ha convertido en un SGBD usado en importantes empresas en aplicaciones críticas en el Ministerio de Fomento, CEPSA, XEROX, Nasdac ([H1]). Así que estamos ante un SGBD de trabajo profesional integrado en un mercado de productos informáticos donde hay mucha competencia

comercial. Esto hace que se necesite evaluar continuamente si el SGBD satisface los requerimientos para el que se implantó, y si tiene las capacidades y cumple las funcionalidades esperadas; es decir tenemos que realizar la auditoría del sistema de gestión de base de datos.

3 Introducción a la Auditoría.

Existen varias definiciones del término Auditoría dependiendo del ámbito en el que se encuadre, en nuestro caso, será el ámbito de los sistemas de información y más concretamente el de los sistemas gestores de bases de datos. Por tanto a modo de descripción, diremos que la auditoría informática comprende:

La revisión, análisis y evaluación independiente y objetiva, por parte de personas independientes y técnicamente competentes.

Esa labor se desarrolla sobre el entorno informático de una entidad, abarcando todas o alguna de las áreas, como equipos, sistemas operativos y paquetes, aplicaciones y procesos de desarrollo, organización y funciones, las comunicaciones y la propia gestión de los recursos informáticos.

Se tiene en cuenta las políticas, estándares y procedimientos en vigor en la entidad, su idoneidad así como el cumplimiento de los objetivos fijados, los planes, los presupuestos, los contratos, las normas legales aplicables, la calidad y los controles existentes para analizar los riesgos. Además se valora el grado de satisfacción de los usuarios y directivos.

En la actualidad, la auditoría informática es una práctica habitual en muchas organizaciones, cuyo centro de atención es el sistema de información de la organización.

El auditor informático, figura central de la auditoría, se caracteriza por la imparcialidad, la objetividad y la independencia. Lo que conlleva una cualificación, experiencia y conocimientos en constante actualización que le

permitan analizar con precisión las complejas situaciones que se generan en los sistemas de información de las organizaciones.

Según sea el agente que realiza la función de auditoría, pueden distinguirse dos tipos de auditorías, externa o interna. En la auditoría externa, se confía la realización a una organización externa especializada en esta función a la que se contrata la realización del servicio. En la auditoría interna, existe una unidad o departamento dentro de la organización en cargado de su realización; en este caso, la posición del departamento de auditoría debe ser tal que garantice la independencia que es necesaria para realizar su cometido ([H6]).

3.1 *Necesidad de la auditoría*

Los Sistemas Informáticos deben estar controlados. Y algunas de los motivos por los que se debe controlar son los que seguidamente expondremos:

- Existe la posibilidad de procesar y difundir información errónea por parte de los servidores de datos, generando por tanto resultados incorrectos. Aquí entra la Auditoría Informática de Datos.
- Las estaciones de trabajo, servidores y en general, los Centros de Procesamiento de Datos (CPDs) están amenazados por el espionaje, la delincuencia y el terrorismo. Además se tiene constancia que una gran mayoría los actos de sabotaje son internos. En este caso interviene la Auditoría Informática de Seguridad.
- El cumplimiento de la Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal, en sus Articulo 1 "*tiene por objeto garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor e intimidad personal y familiar*" ([H2]).

Estos son solo algunos de los varios inconvenientes que puede presentar un Sistema Informático, por eso existe necesidad de auditoría.

Existe otra razón por la que es necesaria una auditoría y para ello tenemos que diferenciar *seguridad* de *auditoría*. Pongamos el ejemplo del robo de un dato: un usuario de una base de datos puede visualizar un dato que considera interesante gracias a que tiene permisos: *seguridad*. Gracias a la seguridad, hay usuarios que no tienen permisos para ver ciertos datos. Sin embargo, un usuario que sí tiene permisos puede usar ese dato para el beneficio personal. La necesidad, en este caso, de la auditoría, está en poder ver qué usuarios accedieron a ese dato en un momento concreto para poder así usarlo para el beneficio personal.

3.2 *Niveles de auditoria*

En este apartado describiremos un conjunto de niveles de auditoría, en qué consisten y qué abarcan. Posteriormente analizaremos sus puntos positivos y negativos.

Podemos recoger dos categorías; en cada categoría, varios niveles:

- Categoría de datos personales
- Categoría de datos sensibles para la organización

La categoría de los datos personales es un conjunto que está dentro de la categoría de los datos sensibles para la organización. Vamos a tomar un subconjunto de esta categoría ya que consideramos dos tipos de información, según el tipo de ataques que puedan recibir.

En la categoría de datos personales que no son sensibles para la organización, la operación más delicada que puede recibir es la consulta, pues está la Ley Orgánica de Protección de Datos defendiéndolos. Sin embargo, un cambio en esos datos no afectan a los intereses económicos, a los objetivos ni a las funciones de la organización, a excepción de la aplicación de dicha ley con las multas que pueda recibir y la mala imagen que pueda dar.

En la categoría de datos sensibles para la organización incluiremos, además de los anteriores, los datos que pueden afectar a los intereses económicos, a los objetivos y a las funciones de la organización.

3.2.1 **Categoría de datos personales**

En esta categoría, que comprenden todos los datos de carácter personal no sensibles para la organización, estableceremos una jerarquía de tres niveles, cada uno con sus requisitos, que se explican a continuación:

3.2.1.1 **Nivel básico**

Este nivel corresponde al nivel de seguridad más bajo. Es muy recomendable que cualquier base de datos cuente, como mínimo, con este nivel, pues cubre los aspectos más básicos de seguridad. Sus requisitos son:

- Se debe tener un Documento de Seguridad, ya sea en papel, en formato electrónico o en formato relacional, en el que queden reflejados qué elementos y qué operaciones sobre dichos elementos pueden realizar los diferentes tipos de usuarios, por roles y/o por usuarios particulares, según la conveniencia de la organización (ver artículo 11.1 de la LOPD ([H2])).

- Se debe restringir el acceso a los datos por parte de usuarios que no tengan los permisos pertinentes, según el Documento de Seguridad, ya sea mediante privilegios *SQL Server* o mediante otros mecanismos. Es tarea del administrador la gestión de la seguridad, como concepto descrito en el apartado 3.1 Necesidad de auditoría.

- Se debe controlar el acceso al sistema, por medio de auditoría genérica, auditando dicho acceso y controlando el número de intentos. Es la auditoría mínima que se debería hacer para controlar el acceso del personal. Teniendo esta información y los horarios de

trabajo de los usuarios, se podrán detectar fácilmente anomalías mediante alertas y procedimientos almacenados.

- Se debe tener un repositorio activado, donde guardar los registros de auditoría. Activando algunos de los modos de auditoría que se detallan en el apartado 5. Tipos de auditoría del gestor de este documento.

- Guardar registros de auditoría durante al menos 1 años (ver artículo 47.1 de la LOPD ([H2])).

3.2.1.2 **Nivel Medio**

El nivel medio está diseñado para organizaciones medianas y grandes, que manejan datos de importancia y cuya seguridad debe ser más fuerte. Sus requisitos son:

- Cumplir el nivel básico.

- Se debe designar un responsable de seguridad (ver artículo 16 de la LOPD ([H2])) distinto del administrador, con un ordenador distinto del servidor para realizar sus funciones, que son:

    - Evolucionar las políticas de auditoría.

    - Obtener información sobre hechos excepcionales.

    - Obtener información sobre datos de auditoría, tal y como se explicará en el apartado 5.2.1 Creación de una traza mediante *SQL Profiler*.

    - Realización de copias de seguridad y limpieza de las tablas de auditoría.

- Se debe auditar y vigilar los intentos de realización de operaciones, autorizadas o no, sobre objetos de la base de datos y/o sobre los

datos personales con el objetivo de limitar dichos intentos por parte de un usuario. Esto se podrá realizar con el modo de auditoría C2 activado.

- Se debe auditar los intentos fallidos de conexión del usuario auditor, único usuario que puede ver el contenido de las tablas de auditoría, usando el envío de mensajes tanto al propio auditor como al administrador de la base de datos cada vez que esto ocurra. Esto se podrá realizar con el modo de auditoría C2 activado.

- Separar el fichero de transacciones o fichero de log en otro disco dentro del servidor de base de datos para obtener mayor eficiencia, en el caso de ser una base de datos muy concurrente.

Guardar registros de auditoría durante al menos 3 años (ver artículo 47.1 de la LOPD ([H2])).

3.2.1.3 **Nivel Alto**

El nivel alto es de utilidad para grandes organizaciones, donde la información es su valor más importante. Y no sólo datos de la organización, sino también datos personales, datos de mercado o datos económicos. Sus requisitos son:

- Cumplir el nivel medio.

- Copias de seguridad guardadas en ubicaciones diferentes.

- Repositorio de tablas de auditoría en servidor externo al de la base de datos.

- Desarrollo de alertas para la detección de posibles irregularidades: procedimientos almacenados que vigilan los movimientos que el auditor cree más conveniente vigilar. Esto se podrá realizar con

auditoria de grano fino recogiendo información en una traza mediante el uso de *SQL Profiler*.

3.2.2 Categoría de datos sensibles para la empresa

En esta categoría, que comprenden, además de los datos personales, todos los datos sensibles para la organización ya sean de carácter personal o no, se ha establecido igualmente una jerarquía de tres niveles, cada uno con sus requisitos, que se explican a continuación:

3.2.2.1 **Nivel Básico**

Este nivel es similar al nivel básico de la categoría de datos personales, explicado en el punto 3.2.1.1 Nivel básico.

3.2.2.2 **Nivel Medio**

Este nivel es similar al nivel medio de la categoría de datos personales, explicado en el punto 3.2.1.2 Nivel medio, más las siguientes características:

- Se deben auditar los intentos de actualización de los datos personales de los usuarios sensibles para la organización por parte de los propios usuarios. Cada vez que se actualicen dichos datos con una frecuencia mayor de la esperada, el auditor deberá investigar este hecho sobre quién ha realizado las modificaciones, desde qué ordenador y cuándo, por medio de procedimientos almacenados, con el objetivo de detectar irregularidades dentro de la plantilla de la organización. Esto se podrá realizar con auditoria de grano fino recogiendo información en una traza mediante el uso de *SQL Profiler* ó usando disparadores *DML*, cuyo funcionamiento explicaremos en el punto 4.1.2.

- Se deben auditar los intentos de actualización y borrado de los datos sensibles para la organización por parte de los usuarios, ya sean autorizados o no autorizados. Cada vez que se actualicen dichos datos, sin frecuencia límite, el auditor deberá investigar este hecho sobre quién ha realizado las modificaciones, desde qué ordenador y cuándo, por medio de procedimientos almacenados, con el objetivo de detectar irregularidades dentro de la plantilla de la organización. Esto se podrá realizar con auditoria de grano fino recogiendo información en una traza mediante el uso de *SQL Profiler* ó usando disparadores *DML,* cuyo funcionamiento explicaremos en el punto 4.1.2.

3.2.2.3 **Nivel Alto.**

Este nivel es análogo al nivel alto de la categoría de datos personales, explicado en el punto 3.2.1.3 Nivel alto del presente documento.

3.3 *Ventajas y desventajas de los niveles de auditoría*

A continuación se enumerarán un conjunto de ventajas e inconvenientes de los niveles de auditoría expuestos en el punto anterior, nivel por nivel.

Las ventajas e inconvenientes del nivel básico están incluidas en el nivel medio, y las ventajas e inconvenientes del nivel medio están incluidas dentro del nivel alto.

3.3.1 **Nivel básico**

3.3.1.1 **Ventajas**

Una de las principales ventajas que tiene este nivel es la documentación. Es de gran importancia este hecho debido a que facilita la detección de errores de asignación en los permisos y la evolución de éstos con los continuos cambios.

La gestión de privilegios permite al administrador y al auditor constar en el sistema quién tiene acceso y quién no tiene acceso a los objetos, es decir, quién tiene derecho a ejecutar una sentencia SQL o el acceso a un objeto de otro usuario. En *SQL Server 2005* nos encontramos con tres niveles o capas en los cuales podemos gestionar la seguridad. El primero de ellos se encuentra a nivel de servidor, en él podemos gestionar quién tiene acceso al servidor y quién no, y además gestionamos que roles va a desempeñar. Para que alguien pueda acceder al servidor debe tener un inicio de sesión (login) asignado, y a éste se asignaremos los roles o funciones que puede realizar sobre el servidor. La gestión de privilegios permite tener controlado todo ello.

El control de acceso a la base de datos permite saber quién está y quién no está conectado en este momento, así como quién estuvo y quién no estuvo conectado a la base de datos en un momento determinado. Esto permite saber si un usuario estaba o no estaba conectado en el momento en que se produjo algún error o vulnerabilidad.

Además, permite detectar infracciones del tipo leve, descritas en el artículo 44.2 de la LOPD ([H2])

3.3.1.2 **Desventajas**

Si tenemos una base de datos con altos niveles de conexiones y desconexiones en se puede descontrolar este control. Además, debemos gestionar los registros ó log de auditoría. El administrador/auditor debe gestionar estos log por si ocurren fallos como llenado de espacios de tabla u otro tipo de inconvenientes.

3.3.2 Nivel medio

3.3.2.1 Ventajas

Con respecto al nivel básico, se obtiene un control casi absoluto de las operaciones por parte de los usuarios. Cada operación que el auditor considera importante es auditada en los registros de auditoría.

Además, hay una persona que gestiona todos estos datos, pudiendo detectar otras vulnerabilidades nuevas dentro del sistema mediante el estudio de la documentación y la evolución de ésta.

Además, permite detectar infracciones del tipo grave, descritas en el artículo 44.3 de la LOPD ([H2]), e infracciones del tipo muy grave, descritas en el artículo 44.4 de la LOPD ([H2]).

Por último, mejora el rendimiento del servidor debido a que se separan los ficheros de transacciones en discos diferentes.

3.3.2.2 Desventajas

La principal desventaja de este nivel es el gasto económico extra en personal y en infraestructura de auditoría. Por ello, este nivel está planteado para ser rentable a organizaciones con un nivel medio, alto o muy alto de flujo de información.

3.3.3 Nivel alto

3.3.3.1 Ventajas

La ventaja añadida del nivel alto con respecto al nivel medio es que facilita la conservación de los datos de auditoría en caso de pérdida de una de las copias, pudiendo ser dichos datos recuperados.

Además, el hecho de externalizar los log de auditoría mejora el rendimiento del servidor de base de datos, a no ser que el flujo de información entre el

servidor de la base de datos y los clientes sea tan alto que aumentar el flujo de información en la red con el servidor de los log de auditoría haga que se ralentice la red, lo que se convertiría en desventaja. Aquí es donde entra el uso de la tecnología *Service Broker* para la transmisión asincrónica de mensajes entre instancias *SQL Server*.

Sin embargo, al externalizar los registros de auditoría estamos facilitando la labor al auditor en caso de ser una base de datos distribuida, pues tendría toda la información de auditoría en un solo repositorio.

Por otro lado, se tiene un control de eventos sospechosos. Operaciones que pueden hacer vulnerable el sistema son detectadas gracias a los elementos de auditoría como políticas de auditoría y procedimientos almacenados.

Por último, el uso de múltiples alertas mejora la seguridad, ya que en caso de posible ataque, su detección sería prácticamente inmediata.

3.3.3.2 **Desventajas**

Puede ser una desventaja el gasto extra en infraestructura. Sin embargo, las empresas con un nivel alto o muy alto de información poseen, normalmente, diferentes servidores, incluso en diferentes edificios, por lo que el gasto económico disminuye. No es así con el gasto computacional.

En cuanto al uso de alertas, para su creación hay dos caminos: la creación de alertas de forma manual, que es costoso en tiempo y personal; y la creación de alertas usando herramientas de auditoría, con un coste económico relativamente alto.

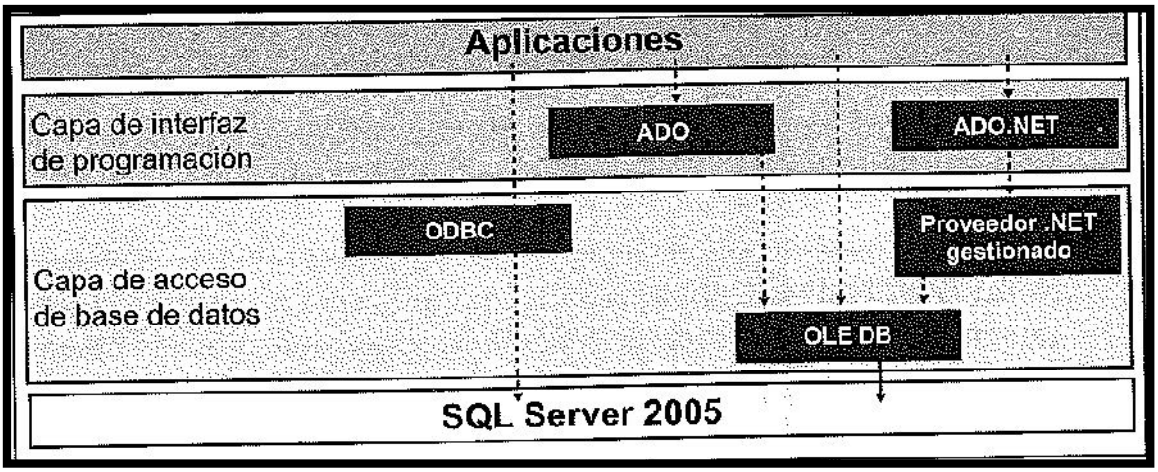
4 Auditoría de Bases de Datos

En este punto vamos a explicar las diferentes formas de auditoría informática. Por un lado hablaremos sobre la auditoría de cliente y, por otro, sobre la auditoría en el motor de la base de datos, cada una con sus diferentes alternativas y soluciones. Pero antes haremos un breve análisis de la arquitectura que fundamenta el sistema gestor de base de datos objeto de nuestro estudio.

En la actualidad muchos sistemas de bases de datos utilizan la arquitectura cliente/servidor que divide la carga de trabajo de las aplicaciones entre el servidor y el ordenador cliente; y *Microsoft SQL Server 2005* es uno de ellos. La arquitectura cliente/servidor tiene como principales características ([H3]):

- Está orientado a servicios. El servidor los ofrece y el cliente los consume.
- Se comparten recursos. Servicios ofrecidos a muchos clientes. Un servidor puede atender muchos clientes que solicitan esos servicios.
- Es transparente a la ubicación. El servidor es un proceso que puede residir en el mismo aparato que el cliente o en un aparato distinto a lo largo de una red. Un programa puede ser un servidor en un momento y convertirse en un cliente posteriormente.
- Mezcla e igualdad. Tal vez de las más importantes ventajas de este paradigma. Una aplicación cliente/servidor, idealmente es independiente del hardware y de sistemas operativos; mezclando e igualando estas plataformas.
- Interacción a través de mensajes, para envío y respuesta de servicios.
- Servicios encapsulados, exponiendo los servicios a través de interfaces, lo que facilita la sustitución de servidores son afectar los clientes; permitiendo a la vez una fácil escalabilidad.

Como partes constitutivas, además del Cliente y del Servidor nos encontramos con el *Middleware*; este elemento abarca todo el software que ocupa la parte intermedia del sistema cliente/servidor y es el enlace que permite que un cliente obtenga un servicio de un servidor. Su ámbito empieza en el módulo de *API* (Interfaz de programación de aplicaciones) de la parte del cliente que se emplea para invocar un servicio y comprende la transmisión de la solicitud por la red y la respuesta resultante. Pero no incluye al software que presta el servicio real; esto pertenece a los dominios del servidor. Tampoco a la interfaz del usuario ni a la lógica de la aplicación, en los dominios del cliente.



Arquitectura MDAC

Como ejemplo de interfaz de programación de aplicaciones estándar para utilizar SGBD tenemos ODBC (conectividad abierta de base de datos), cuyo objetivo principal es proporcionar un estándar genérico usado por varios lenguajes de programación, sistemas de bases de datos y sistemas operativos. Los principales sistemas operativos soportan el estándar OBDC, incluyendo *Microsoft Windows, UNIX, Mac OS X y Linux.* Hay también controladores para SGBD, incluyendo *MS SQL Server, MS Access, DB2, Oracle y Sybase*.

4.1 ***Auditoría de Cliente.***

La auditoría de cliente se realiza por parte del programador de la aplicación ó de la base de datos, no por el administrador ó auditor de la base de datos; por tanto no pertenece al motor de la base de datos. Tendremos dos

subtipos de auditoría de cliente: auditoría de aplicativo y auditoría de disparadores.

4.1.1 **Auditoría de aplicativo.**

La auditoría de aplicativo es el servicio que permite analizar las aplicaciones forma independiente a la base de datos, y su ámbito estará delimitado al trabajo diario de la aplicación y los procesos en los que participe. Esta forma de trabajar nos ofrece la ventaja de la independencia, puesto podemos desarrollar en el ámbito de nuestra aplicación informática los controles necesarios y adecuados a nuestras necesidades de auditoría sin necesidad de tener que acceder al motor de la base de datos; esta manera de auditar permite crear procesos de control más concretos y por tanto más eficaces. Por otra parte tendremos aplicaciones menos flexibles dado que el desarrollo de soluciones de control para la auditoria de bases de datos será más laborioso y tendremos más problemas para el mantenimiento y actualización de la aplicación.

Uno de los productos que encontramos alrededor de *SQL Server 2005* es la herramienta *Visual Studio 2005* ([H4]), que además de permitirnos desarrollar aplicaciones basadas en datos con *SQL Server 2005* permite crear extensiones de los servicios *SQL Server 2005*; desde funciones definidas por el usuario para las bases de datos relacionales y de análisis, o procedimientos almacenados para la base de datos relacional, hasta extensiones para acceso a datos, generación y entrega de informes. Por tanto, las empresas pueden desarrollar para el gestor de información aplicaciones que ayudan a gestionar sus datos ó crear una aplicación que permita gestionar operaciones complejas y con un requerimiento de seguridad importante, como ejemplo podemos hablar de los portales corporativos, de la banca por Internet, del comercio electrónico ó de las redes privadas virtuales.

4.1.2 **Auditoría de disparadores DML.**

Los disparadores se usan para añadir lógica o restricciones a la base de datos, por ejemplo pueden ser usados para establecer reglas de integridad con bases de datos externas (no grabar un pedido en la base de datos de pedidos si el cliente indicado no está dado de alta en la base de datos de pedidos por ejemplo); también se para mantener tablas de acumulados como por ejemplo la tabla que mantienen el stock de una determinada compañía o para guardar el acumulado de ventas en la ficha de un cliente. Nosotros nos vamos a centrar en el uso de los disparadores para auditar la información contenida en una tabla, registrando los cambios realizados, es decir, si un usuario inserta, actualiza o borra datos de una tabla se ejecutará un disparador que copiará la información a una tabla de auditoría para que posteriormente el auditor pueda analizar las operaciones. Usar este recurso nos aporta grandes ventajas, ya que de una manera muy sencilla y directa va quedando un registro de estas operaciones tan frecuentes en una base de datos, la única pega que se le podría poner es que aumenta la complejidad del sistema de base de datos puesto que es un código que se ejecuta.

Los *Triggers* o disparadores DML son piezas de código *Transact-SQL* que se ejecuta como respuesta a un evento producido por una operación de manipulación de datos, tales como sentencias *INSERT*, *UPDATE o DELETE*. DML es el acrónimo de *Data Manipulation Language* (Lenguaje de Manipulación de datos). Los disparadores son un tipo de objetos muy especiales en *SQL Server 2005*, ya que realmente son muy parecidos a lo que las rutinas de atención a eventos en código en cualquier lenguaje de programación como por ejemplo *.NET*. ([H5])

SQL Server 2005 dispone de unas tablas virtuales, llamadas *inserted y deleted*. Estas tablas especiales contienen la información de los registros que se han eliminado o insertado, con exactamente las mismas columnas que la tabla base que está sufriendo esa modificación. Las tablas *inserted y deleted* estarán o no rellenas de datos en función de cuál sea el tipo de operación que ha dado lugar

a su ejecución. Por ejemplo, un *Trigger* que se dispare por la inserción en una tabla tendrá tantos registros en la tabla virtual *inserted* como registros estén siendo insertados y cero registros en la tabla *deleted*; un disparador que se dispare por la eliminación de registros en una tabla tendrá cero registros en la tabla *inserted* y tantos registros en la tabla *deleted* como registros estén siendo eliminados, y un disparador que responda a una operación de *update*, tendrá el mismo número de registros en la tabla *inserted* y en la tabla *deleted* que además coincidirá con el número de registros actualizados en la tabla que da lugar al evento.

Los *Triggers* de tipo *Instead OF* son *Triggers* que se disparan en lugar de la operación que los produce, es decir, una operación de borrado de registros con la instrucción *delete* sobre una tabla que tiene un *Trigger* de tipo *INSTEAD OF* no se llega a realizar realmente, sino que *SQL Server 2005* cuando detecta esta operación invoca al *Trigger* que es el responsable de actuar sobre los registros afectados, en el ejemplo que estamos siguiendo, el *Trigger* sería el responsable de borrar los registros de la tabla que ha disparado el evento. Si el *Trigger* no se encarga de esta tarea, el usuario tendrá la sensación de que *SQL Server* no hace caso a sus comandos ya que por ejemplo una instrucción *DELETE* no borrará los registros.

Como ejemplo de *TRIGGER* de tipo *INSTEAD OF* vamos a ver como se implementaría la siguiente regla: ”no se pueden borrar los clientes cuyo Crédito Total sea mayor que cero, sin embargo si dentro de una operación de borrado hay clientes con Riesgo Total cero y otros con Riesgo Total distinto de cero, los que tengan cero si deben resultar eliminados”.

El TRIGGER que implementaría esa regla sería el siguiente ([H11]).

```
CREATE TRIGGER TR_BorradoSelectivo on Clientes INSTEAD OF
DELETE
AS
BEGIN
   DELETE c
   FROM Clientes C
   INNER JOIN DELETED d
      ON C.idCliente=D.idCliente
   WHERE C.CreditoTotal=0
END
```

4.2 *Auditoría en el Motor.*

La importancia de la auditoría del entorno de bases de datos radica en que es el punto de partida para poder realizar la auditoría de las aplicaciones que utilizan esta tecnología. Sin embargo, cada vez es mayor el interés de las organizaciones y por tanto de los auditores por tener conocimiento de las operaciones que cada usuario realiza sobre los objetos de una base de datos. Y llegamos así a establecer que existe una gran relación entre auditoría y seguridad de la información, creamos controles de acceso a los sistemas que después nos ofrecen una información auditable en términos de seguridad. Por tanto, en un sistema gestor de bases de datos se debe establecer procedimientos de explotación y mantenimiento que aseguren que los datos se tratan de forma congruente y que el contenido de los sistemas sólo se modifica mediante la autorización adecuada.

Como norma general el proceso de auditoría de bases de datos debe ser independiente de las aplicaciones que hacen uso de la información contenida en ella, así que a la hora de evaluar el coste de la realización de la auditoría sólo tendremos que tener en cuenta el coste que se genera en el proceso de recogida de la información y el coste del procesamiento de la información que sirve de base para el análisis y estudio del auditor. Valoraremos la relación coste/beneficio en la implantación de cada nivel de auditoría por la gran dependencia que tenemos de la tecnología del gestor. Debemos ser selectivos a la

hora de recolectar información ya que se dispararían las necesidades de almacenamiento y el rendimiento de la base de datos y del sistema bajaría. La recogida de datos se hace en paralelo con las operaciones del gestor, mientras que su procesamiento puede ser realizado con posterioridad.

En principio la auditoría de base de datos tiene como ámbito aquellas acciones que suceden dentro del gestor, e incluirá las modificaciones de la estructura de objetos de la base de datos.

No debería generar información sobre las operaciones que suceden fuera del gestor, actividad en ficheros, ejecución de programas, conexiones del sistema operativo. Aunque de manera general ningún gestor audita operaciones fronterizas, como el uso que se hace de copias de seguridad, o una recuperación de datos; si sería interesante buscar mecanismos y herramientas que auditen el producto de este tipo de operaciones ([H7]).

Vamos a mostrar los diferentes niveles de auditoría que podemos realizar dependiendo del modo en que recopilamos los datos; inicialmente diferenciaremos entre la recogida de datos agregada y el modo de recogida detallado.

En el agregado realizamos estadísticas sobre el número de operaciones realizadas sobre un objeto de la base de datos, y como es una estadística su medida se puede hacer con técnicas muéstrales o censales. En el censal el SGBD toma datos de todas las operaciones que recibe, esto se puede realizar en paralelo con las operaciones auditadas. En el caso muestral se tomarán de forma periódica los datos de las sesiones que se mantienen en ese momento en el SGBD.

En el modo detallado incluimos todas las operaciones realizadas sobre el objeto. Por una parte tendremos en cuenta las operaciones que produzcan cambios en el contenido de los datos, así que habrá que disponer de una imagen de los datos anteriores y posteriores a la operación de cambio, así mismo contemplamos los posibles cambios en la estructura de los objetos que componen

la instancia de base de datos. Y por otra parte vigilamos las operaciones de acceso al contenido de los datos, tanto en su nivel de operación que nos dice la sentencia SQL que se ejecutó, como en su nivel de resultado, esto es los datos que se vieron en la sentencia SQL ejecutada.

Tenemos además un nivel de auditoría relacionado con las operaciones fronterizas, como son las copias de seguridad y la reconstrucción de estados; aunque ya dijimos que estaban poco implantados los mecanismos para realizar este nivel de auditoría.

5 Tipos de auditoría del Gestor.

5.1 *Modo de Auditoría C2.*

Este modo fue diseñado para poder cumplir con la clase *C2 del Criterio de evaluación de confianza de un sistema computacional* creado por el Department of Defence de los Estados Unidos de América (DoD) ([H8]).

En *SQL Server 2005*, al habilitar el *C2 Audit Tracing* auditamos todos los intentos de obtener acceso a instrucciones y objetos. Esta información puede servirnos de ayuda para auditar la actividad del sistema y realizar un seguimiento de las posibles infracciones de las directivas de seguridad. Es, por tanto, un modo de seguridad del servidor y para poderse ejecutar se debe de tener el rol *sysadmin*, es decir administrador de la base de datos.

Los datos de *C2 Audit Mode* se guardan en un archivo en el directorio *..\MSSQL\Data* de las instancias predeterminadas y en el directorio *..\MSSQL$instancename\Data* de las instancias con nombre. Si el tamaño del archivo de registro de auditoría supera el límite de 200 megabytes (MB), *SQL Server 2005* creará un archivo nuevo, cerrará el antiguo y escribirá todos los registros de auditoría nuevos en el nuevo archivo. Este proceso continuará hasta que el directorio de datos de auditoría se llene o la función de auditoría se desactive.

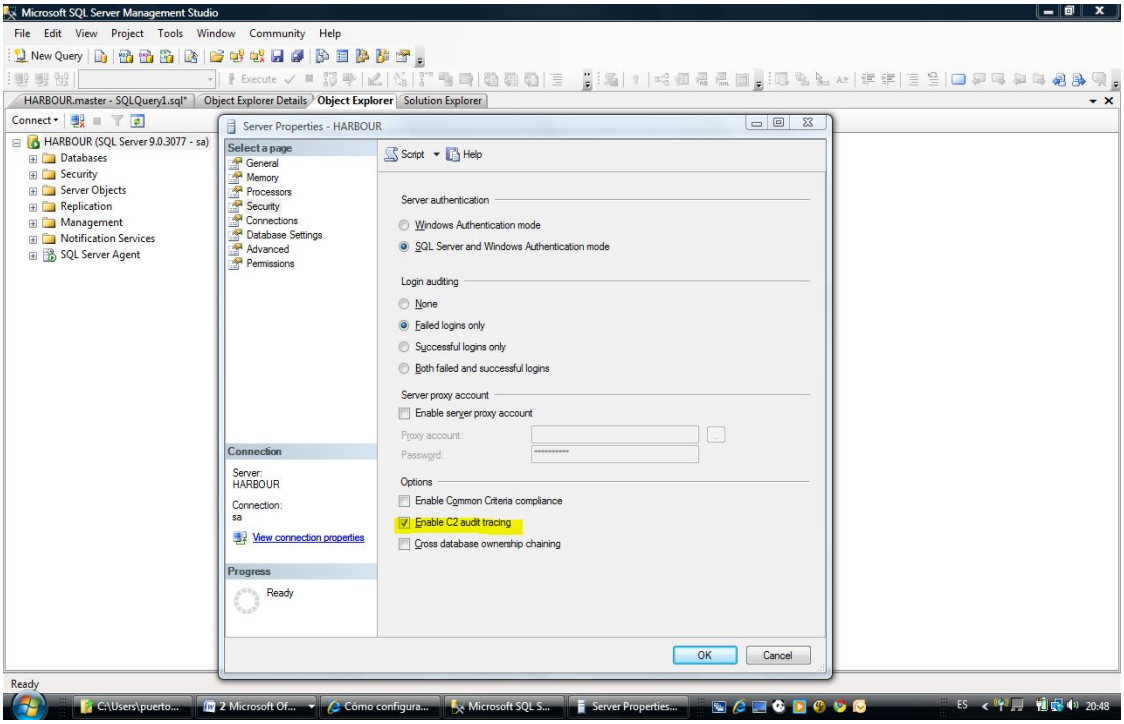
C2 Audit Mode puede configurarse mediante las opciones de configuración de *SQL Server Management Studio*, que es un entorno integrado para obtener acceso a todos los componentes de *SQL Server 2005*, configurarlos, administrarlos y desarrollarlos; o a través de la ejecución del procedimiento almacenado del sistema (*stored procedure)* llamado *sp_configure*, opción que explicaremos con más detalle. *C2 Audit Mode* guarda una gran cantidad de información de eventos en el archivo de registro ya que el servidor registrará tanto los intentos sin éxito como los intentos con éxito de acceso a instrucciones y objetos; es decir, registra el apagado y encendido del SGBD, los intentos de

inicio de sesión de los usuarios, el acceso de los usuarios a los diferentes objetos de la base de datos (tablas, vistas, registros, etc.) teniendo en cuenta los permisos que tienen, la sentencias de los lenguajes de manipulación de datos (DML) o de definición de datos (DDL) o de control de acceso (DCL); el hecho de recoger tanta información tiene como consecuencia el rápido crecimiento del tamaño del fichero de almacenamiento.

Si el directorio de datos en el que se guardan los registros se queda sin espacio, *SQL Server* se cerrará automáticamente. Si la auditoría se ha configurado para iniciarse automáticamente, deberá reiniciar la instancia con el indicador -f (que omite la auditoría) o liberar más espacio de disco para el registro de auditoría.

Configuración mediante *SQL Server Management Studio* ([H9])

- En el Explorador de objetos, haga clic con el botón secundario en un servidor y, a continuación, seleccione **Propiedades**.
- Seleccione la página **Seguridad**.
- Active la casilla de verificación **Habilitar traza de auditoría C2.**



Configuración mediante el procedimiento almacenado *sp_configure* ([H9]). Definiremos *stored procedure* como una sucesión ordenada de instrucciones *Transact-SQL* guardada en el servidor con un nombre, para luego poder ser invocada y ejecutada ([H10]). El procedimiento almacenado *sp_configure* se utiliza para ver o cambiar la configuración del servidor.

Inicialmente mediante una consulta al diccionario de la base de datos podemos obtener el modo en el que se encuentra:

```
SELECT * FROM sys.configurations

ORDER BY name ;
```

Configuration_id	name	value
544	C2 audit mode	0

Donde el valor *0* correspondiente a *value* significa que está desactivado el modo *C2* y el valor *1* significa que esta activado

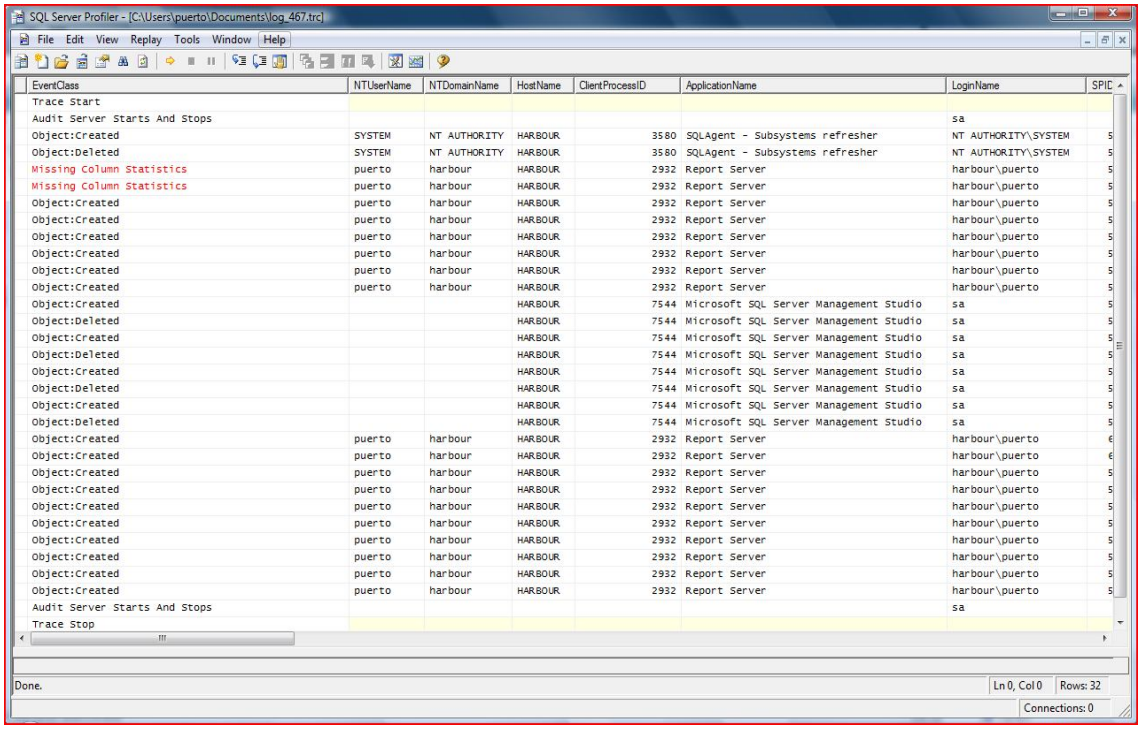
Primero hay que establecer en 1 el valor de *C2 Audit Mode*. A continuación, si se ejecutamos *sp_configure* sin parámetros, se mostrarán todas las opciones de configuración. Y ejecutamos la instrucción *RECONFIGURE* para instalar.

```
USE master;
GO
EXEC sp_configure 'C2 Audit Mode', '1';
```

```
Configuration option 'c2 audit mode' changed from 0 to 1. Run the RECONFIGURE statement to install.
```

```
RECONFIGURE;
EXEC sp_configure;
```

Hay dos maneras de ver el contenido de los registros del fichero de auditoría; la primera, mediante la herramienta *Profiler de SQL Server 2005*, abriendo directamente el fichero de traza.



La segunda, ejecutando siguiente consulta *T-SQL*.

```
SELECT *
 FROM ::fn_trace_gettable(
 'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\LOG\
log_467.trc', default
)
GO
```

Entre otros datos contiene la información de hora, el identificador de la cuenta que provocó el evento, el nombre del servidor de destino, el tipo de evento, sus resultados (éxito o fracaso), el nombre del usuario del servidor de aplicación y proceso de identificación de la conexión del usuario y el nombre de la base de datos.

SQL Server Management Studio nos ofrece varias formas de ver la información que contiene el fichero de auditoría; en texto plano, de forma tabular

o un fichero que posteriormente podemos editar en un programa de análisis como *MS Excel.*

La principal limitación que tiene esta forma de recopilar información para la auditoría de la base de datos es que reduce el rendimiento de SGBD *SQL Server 2005* puesto que se guarda todas las acciones en un fichero que además puede crecer mucho reduciendo el tamaño del espacio en disco. Según las especificaciones de *C2 Audit Mode*, sí nos quedamos sin espacio en el disco el SGBD se cerraría. Pero sí necesitamos tener una recolección de datos exhaustiva para poder realizar una auditoría detallada de la base de datos, *C2 Audit Mode* puede ser una buena opción.

5.2 *Herramienta SQL Server Profiler*

Es una herramienta incluida en *SQL Server 2005* cuyo principal objetivo es capturar eventos de una instancia de *SQL Server*. Los eventos se guardan generalmente en un archivo de traza ó en una tabla que posteriormente se puede analizar o utilizar para reproducir series concretas de pasos con el objetivo de identificar o diagnosticar ciertos problemas ([H12]).

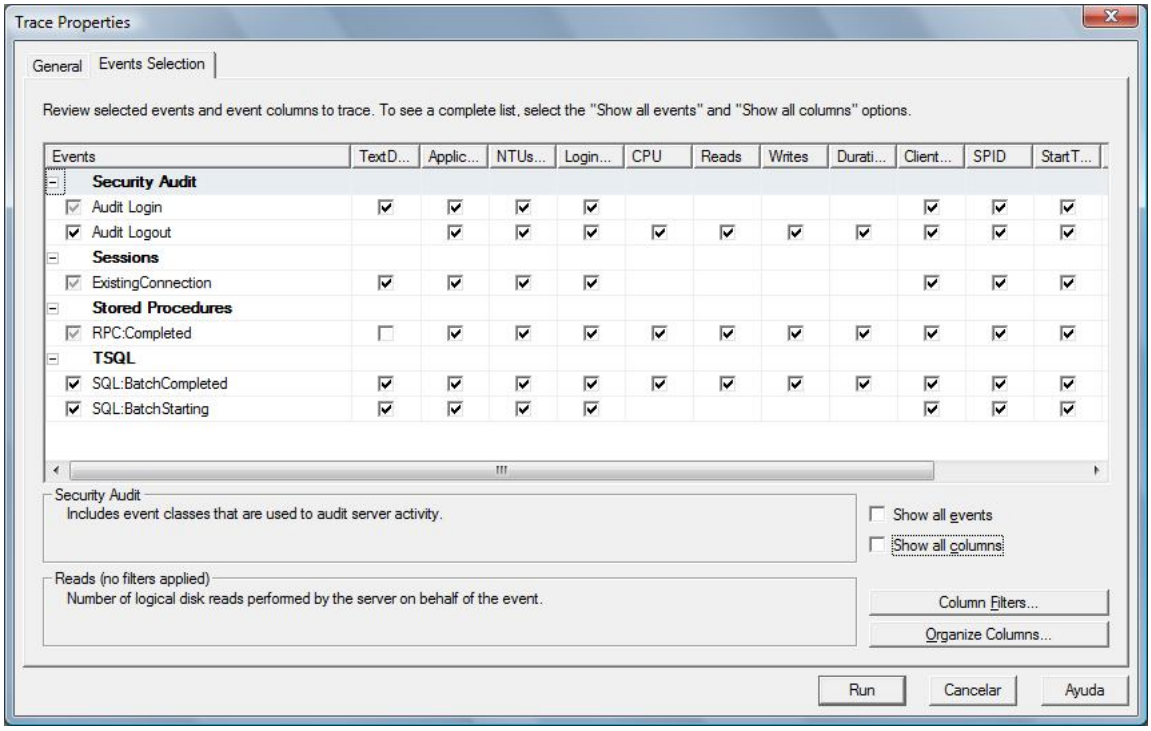
El principal uso que hacemos de la herramienta es para tener una visión de las peticiones que llegan al servidor en determinados periodos de tiempo, pero la herramienta se puede utilizar para muchas otras cosas:

- Depurar procedimientos almacenados e instrucciones de T-SQL.
- Realizar pruebas de carga en distintos servidores mediante la reproducción de trazas grabadas.
- Analizar las consultas guardando en las trazas el plan de ejecución.
- Supervisar el rendimiento de una instancia del motor relacional, *Analisys Server o Integration Services*.
- Como mecanismo de auditoría de seguridad.

Es necesario que el usuario que ejecute la traza tenga permisos ALTER TRACE. GRANT ALTER TRACE TO <inicio_de_sesion>

El GUI de *SQL Profiler* mide el tiempo en microsegundos, y esto afecta a la forma en que se representa el resultado el interfaz gráfico que lo hace en milisegundos. Así que tendremos que tener en cuenta esta peculiaridad si al almacenar el resultado de una traza se guarda como fichero de traza XML, o como tabla de base de datos.

Para hacer más sencilla la selección de columnas, en *SQL Server 2005*, con cada uno de los eventos, *SQL Profiler* nos muestra cuáles son las columnas que devuelven información para cada evento.



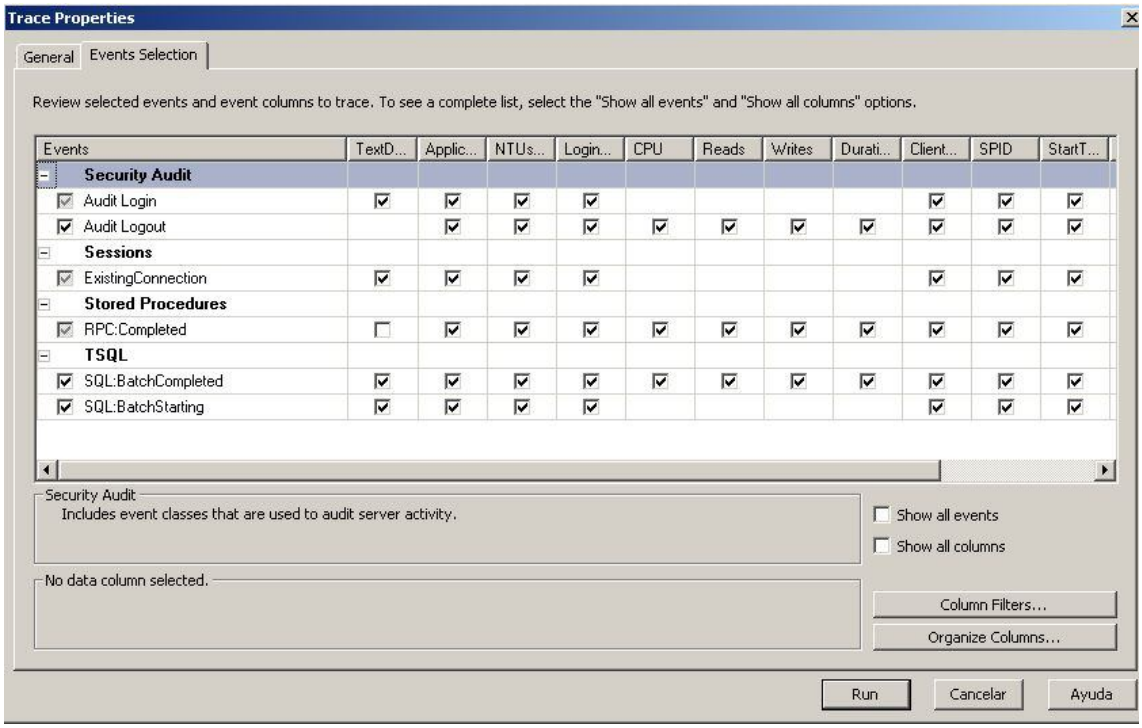
Como recomendación, no debe usarse *SQL Profiler* en el propio servidor de base de datos porque la ejecución de la herramienta causa sobrecarga en el servidor, que dependiendo del tipo de sistema será más ó menos asumible esta penalización en el rendimiento.

Existen varias formas de creación trazas y los eventos que captura; la primera que vamos a estudiar es el llamado modo *Profiler On-Line*. Se considera el uso del analizador del *SQL Server* de modo On-line en el momento que se define la captura de eventos en una traza. Inicialmente nos conectaremos al

servidor *SQL Server*, para después indicar las propiedades de la traza; como es el nombre, la plantilla que usamos, el destino (fichero o tabla) de la traza y el momento de la finalización de la captura. Además de nuevos eventos, columnas y filtros para la traza a analizar. De hecho se realiza mediante la utilización del interfaz gráfico que nos permite crear trazas basándonos en plantillas predefinidas. Inicialmente vamos a evaluar cuáles son los eventos a capturar en relación al objetivo de auditoría buscado. Nos apoyaremos en el análisis de las plantillas para creación de trazas integradas ([H13]).

a. Plantilla “Standard (default)”.

Captura un amplio rango de eventos y datos, y se suele usar como punto de partida para la creación de una nueva definición de traza ó para crear monitorizaciones de propósito general.



- “Sessions”. Incluye las clases de eventos que se producen cuando los clientes se conectan a una instancia de SQL Server y se desconectan de ésta.
- “Stored Procedure”. Contempla las clases de evento que se producen al ejecutarse un procedimiento almacenado.

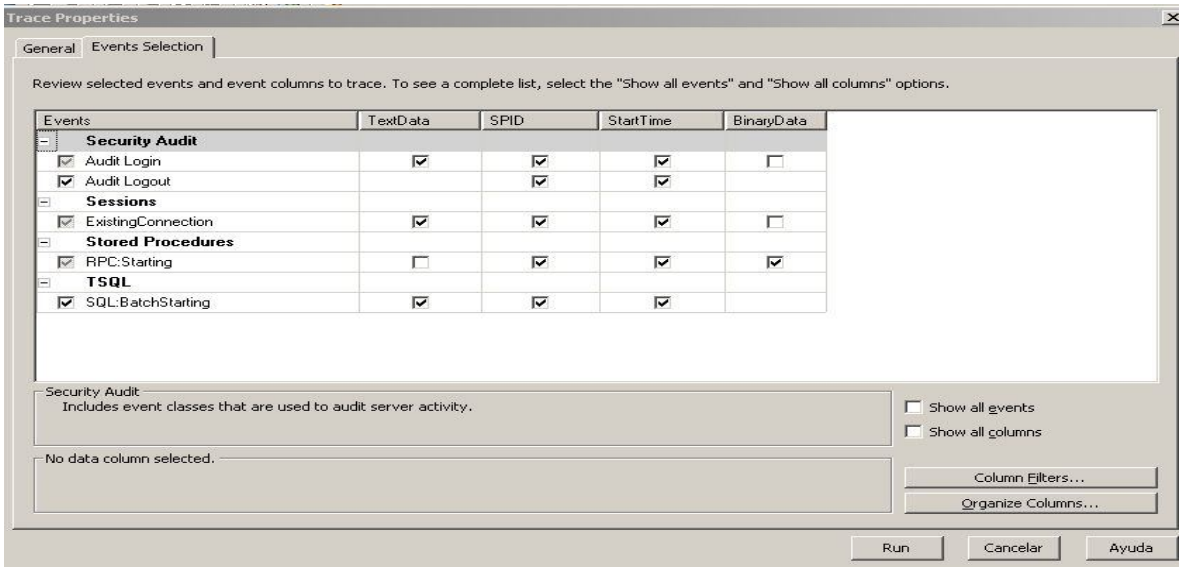
- “TSQL”. Incluye las clases de eventos que se producen al ejecutar instrucciones “Transact-SQL” pasadas a una instancia de SQL Server desde el cliente
- “Security audit”. Incluye las clases de evento que se utilizan para auditar la actividad de los servidores.

Desde el momento en que arrancamos la traza de monitorización almacenamos eventos, tales como, los intentos de conexión y desconexión del cliente de SQL Server, los datos la conexión activa (nombre del usuario Windows ó del entorno SQL Server), las llamadas a procedimientos remotos completados (tenemos columnas de datos referidas al coste de máquina que tiene el procedimiento almacenado), aquellas sentencias de T-SQL completadas y las sentencias que se iniciaron. En la mayoría de las clases de evento se almacena el momento de inicio y fin.

En resumen, estamos auditando quién está conectado, qué consultas y procedimientos ejecuta, cuánto cuesta ejecutarlo, y quién y cuándo se desconecta de la instancia de SQL Server.

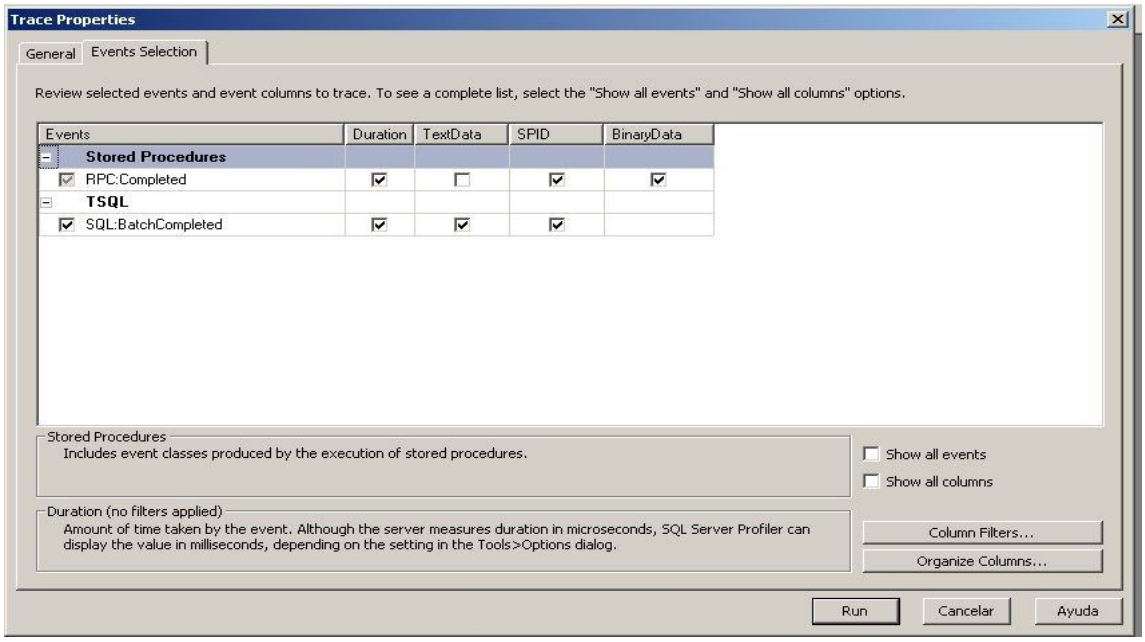
b. SQLProfilerTSQL

Captura eventos relacionados con “Transact SQL” y se usa para interceptar y verificar la depuración de sentencias SQL.



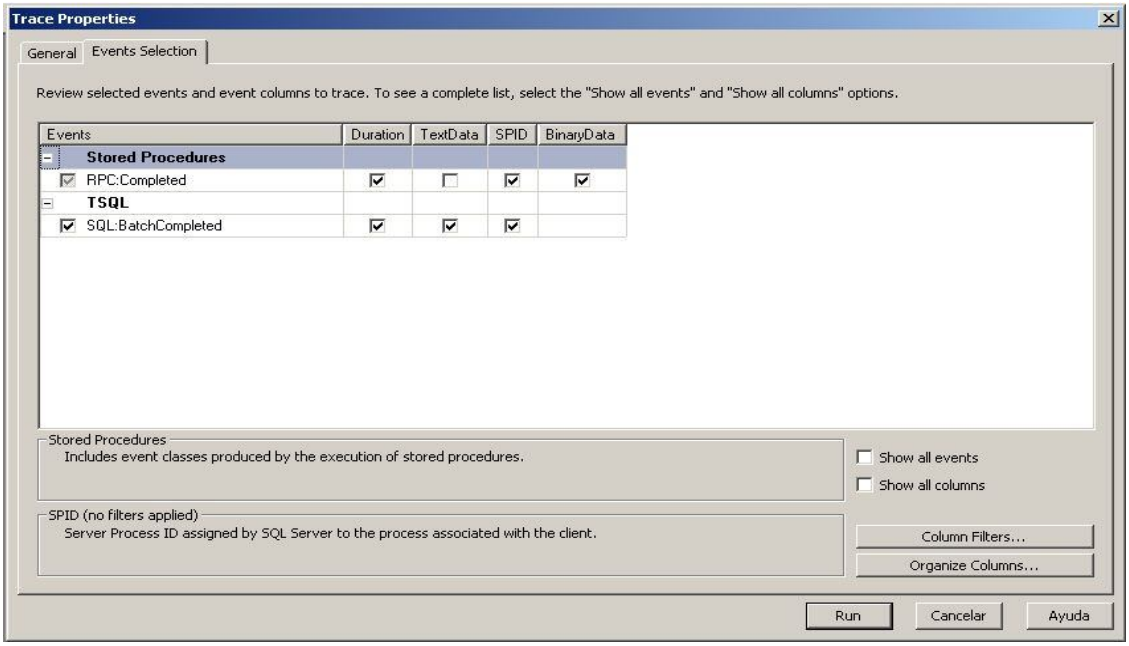
c. SQLProfilerTSQL_Duration

Eventos que permiten medir el coste de ejecución (duración en milisegundos) de la consultas T-SQL y de los procedimientos almacenados y las usamos para supervisar las consultas de alto coste de ejecución.



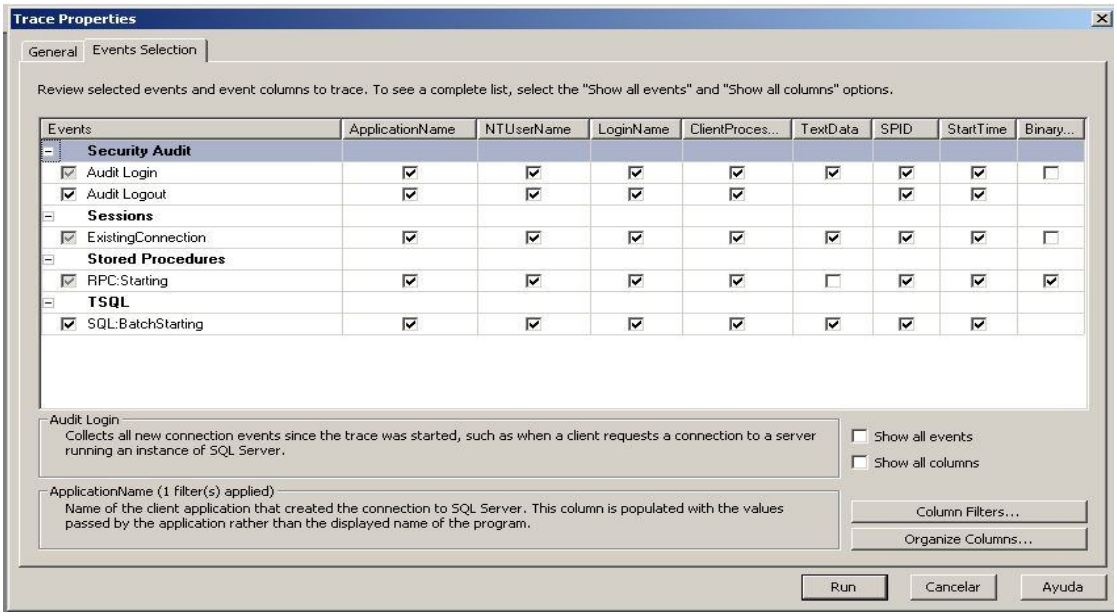
d. SQLProfilerSP_Counts

Contiene eventos que rastrean la frecuencia en que los procedimientos almacenados “SP” se ejecutan y lo podemos usar para identificar la “SP” más comúnmente usados con el fin de poderlos refinar y optimizar.



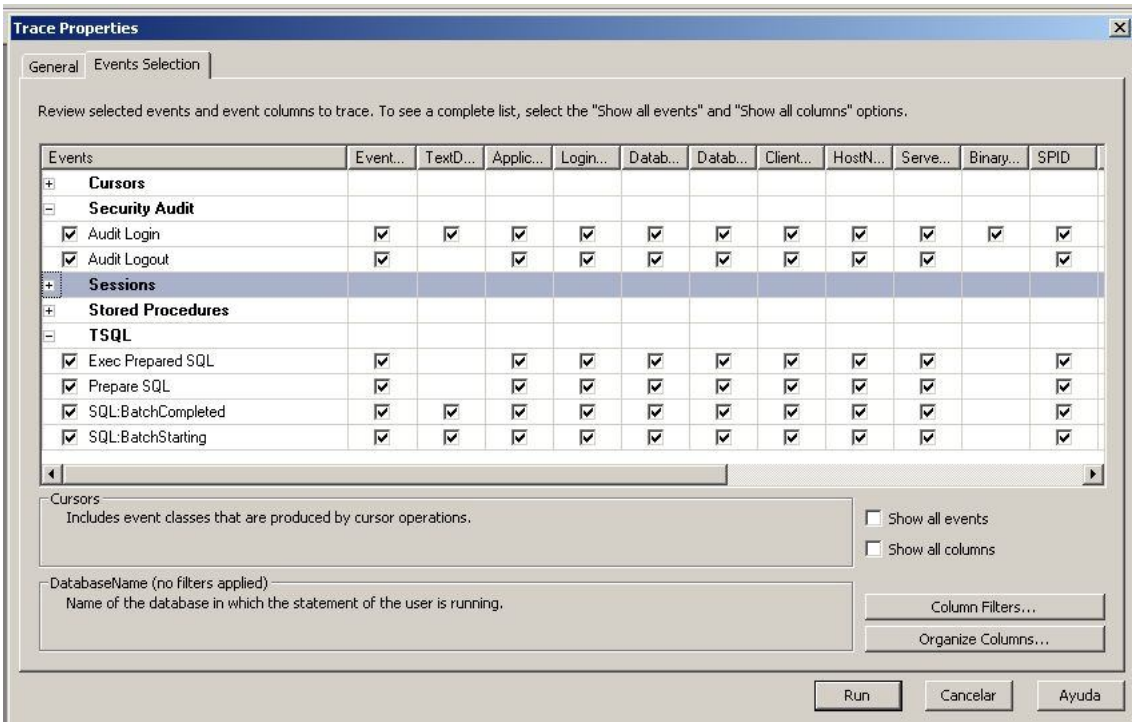
e. SQLProfilerTSQL_Grouped

Es similar a “SQLProfilerStandard”, pero con la información agrupada y organizada, y se suele usar como punto de partida para la creación de una nueva definición de traza ó para crear monitorizaciones de propósito general. Debido a que esta plantilla agrupa la información por usuario, puede ser utilizada para rastrear solicitudes específicas de un usuario problemático. Un ejemplo es una aplicación que utiliza autenticación de “Windows“ para conectar con “SQL Server”, y tiene problemas que se refieren a un usuario específico.



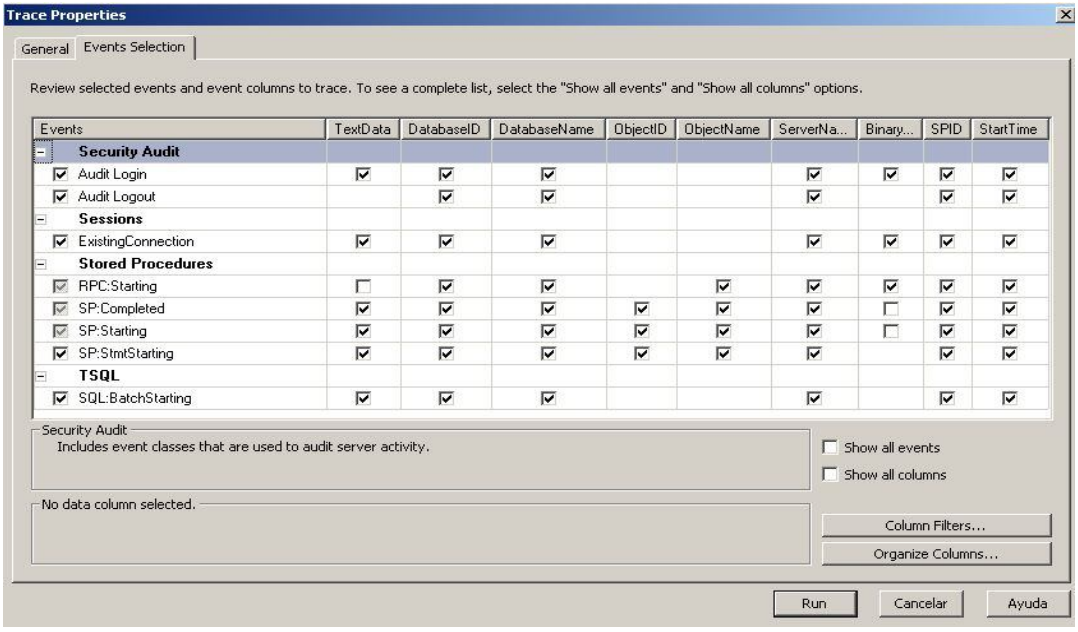
f. SQLProfilerTSQL_Replay

Captura eventos que pueden ser usados para reproducir de manera exacta una secuencia de pasos en un u otro momento, ó en otro servidor; y se suele usar para reproducir y resolver problemas.



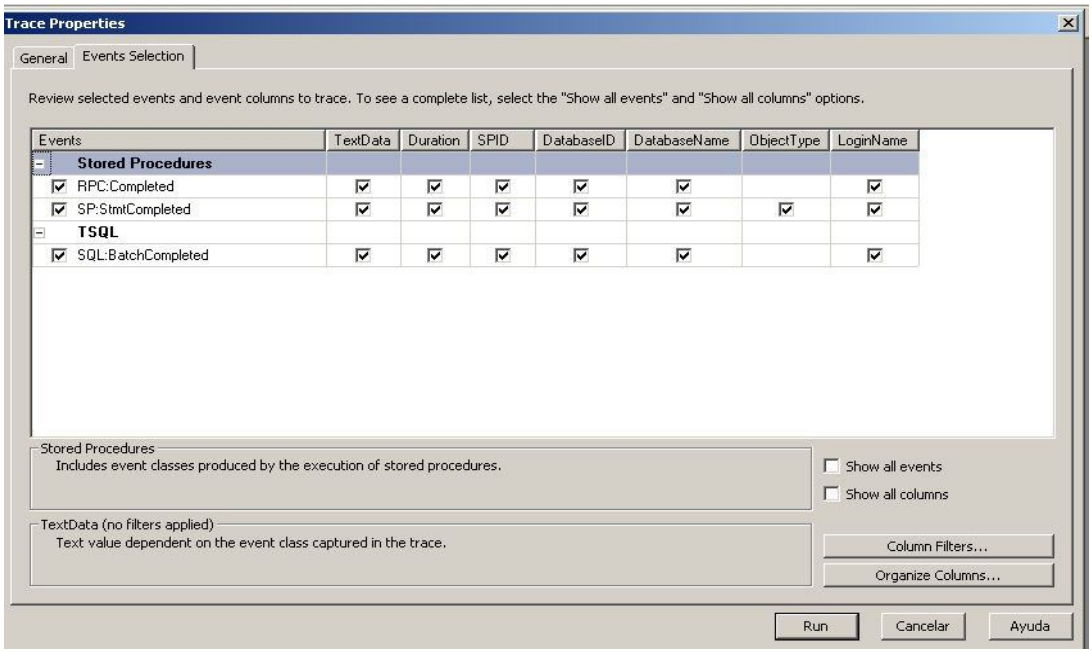
g. SQLProfilerTSQL_SPs

Proporciona un amplio registro de los procedimientos almacenados, incluyendo consultas ejecutadas dentro de procedimientos almacenados. Se usa para resolver problemas en “SP”(Stored Procedure).



h. SQLProfilerTuning

Contiene eventos necesarios para capturar el coste de ejecución y los datos relacionados, así se puede seguir el coste de una consulta.



Los grupos de eventos que se pueden observar son ([H14]):

- "Cursors": Colección de eventos que se producen al crear, usar ó borrar cursores.
- "Database": Clases de eventos producidos cuando se aumenta o reduce automáticamente el tamaño de los archivos de datos o del registro de transacciones.
- "Errors" y "Warnings". Eventos que se producen cuando SQL Server genera un error o una advertencia.
- "Locks". Incluye las clases de evento que se producen cuando se adquiere, cancela o libera un bloqueo, o bien cuando se realiza otro tipo de acción en un bloqueo.
- "Objects". Incluye las clases de eventos que se producen cuando se crean, abren, cierran, quitan o eliminan objetos de la base de datos.
- "Performance" (Rendimiento). Colección de eventos que se producen cuando se ejecutan comandos lenguaje de manipulación de datos (DML) de SQL.
- "Scans" (Recorridos). Son las clases de eventos que se producen cuando se recorren tablas e índices.
- "Security audit". Incluye las clases de evento que se utilizan para auditar la actividad de los servidores.
- "Server". Colección de eventos producidos por el control del servidor y los cambios de memoria.
- "Sessions". Incluye las clases de eventos que se producen cuando los clientes se conectan a una instancia de SQL Server y se desconectan de ésta.
- "Stored Procedure". Contempla las clases de evento que se producen al ejecutarse un procedimiento almacenado.
- "Transactions". Colección de eventos que se producen al ejecutar transacciones del Coordinador de transacciones distribuidas de Microsoft o al escribir en el log de transacciones.
- "TSQL". Incluye las clases de eventos que se producen al ejecutar instrucciones "Transact-SQL" pasadas a una instancia de SQL Server desde el cliente.
- Configurables por el usuario. Aquellas que puede definir el usuario.

En cada uno de los grupos ó categorías de eventos tenemos clases de eventos.

Probemos una de estas plantillas y veamos sus resultados.

5.2.1 Creacion de una traza mediante SQL Profiler

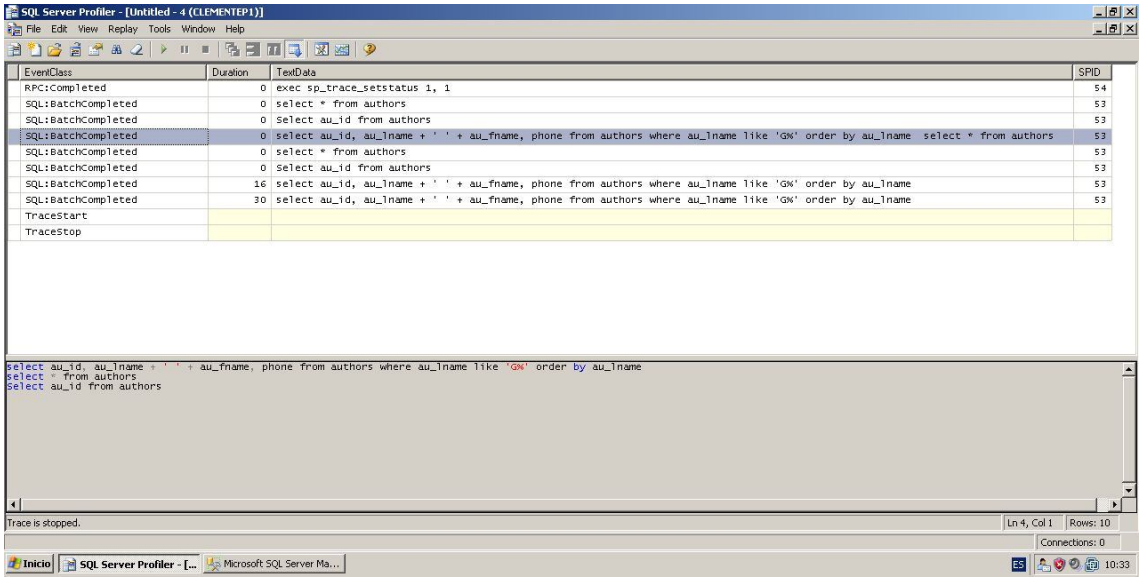
Creemos que tenemos una aplicación que presenta un problema de rendimiento que se produce por la ejecución de una consulta demasiado compleja. Para analizar el problema podemos comprobar la duración de la consulta con el *SQL Profiler* usando la plantilla "SQLProfilerTSQL_Duration". Por supuesto, podemos analizar la consulta de manera interactiva ó salvando los resultados a un fichero para su posterior análisis.

Ejecutamos la traza creada basada en la plantilla antes mencionada ([H15]). Ejecutamos las consultas sobre la base de datos "ADVENTUREWORKS".

select * from information_schema.tables

select * from person.address

Paramos la ejecución de la traza, y obtenemos el siguiente resultado…



Como se puede observar, el evento “SQL:BatchCompleted” en su propiedad “Duration” nos indica el tiempo en milisegundo que ha tardado cada sentencia en completarse.

En los siguientes pasos crearemos una fichero de traza con los resultados de la ejecución de una traza basada en una plantilla, para posteriormente importar los datos del fichero en una tabla y poder analizar su comportamiento.

En esta ocasión vamos a crear una traza a partir de la plantilla “SQLProfilerTSQL_Duration”, pero añadiendo la siguientes columnas de datos.

- CPU
- Reads
- StartTime
- Writes

Guardaremos los datos en C:\Mitraza.trc, ejecutamos las siguientes consultas sobre la base de datos “ADVENTUREWORKS”.

- select ContactID, FirstName + ' ' + LastName, EmailAddress from person.contact
  where LastName like 'G%' order by LastName

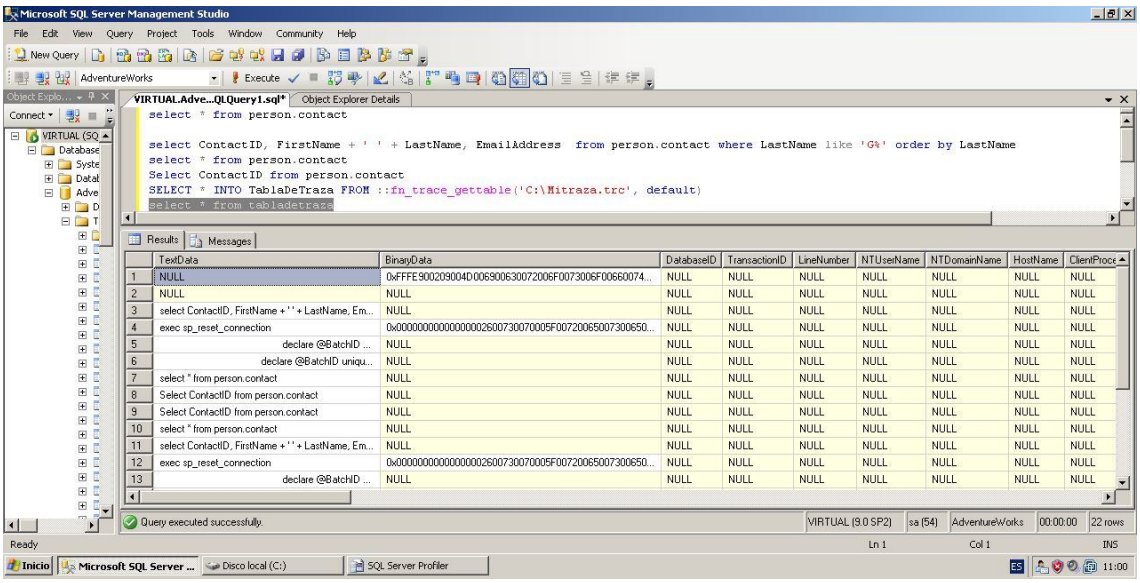
- select * from person.contact
- select ContactID from person.contact

Paramos la ejecución de la traza y en “SQL Query Analyzer” realizamos la consulta:

```
SELECT * INTO TablaDeTraza FROM
::fn_trace_gettable('C:\Mitraza.trc', default)
```

Su función es importar la información capturada en un tabla de la base de datos.

Sí se consulta los datos contenidos en la tabla se observa que contiene información de todas las columnas de datos de la traza, esta columnas se han convertido en campos de la tabla y hay registros sin información relevante.



Posteriomente la información recopilada se analizará usando consultas agregadas que nos ofrezcan medidas estadíticas (media, máximo, mínimo, suma) de la Duración.

5.2.2 Creación de trazas mediante sentencia Transact SQL

Existe una forma alternativa de crear trazas sin usar el interfaz gráfico del *SQL Profiler*, ejecutando sentencias *T-SQL*. Este mecanismo es menos intrusivo para el sistema, el script *T-SQL*, tras ejecutarse en el servidor, registra los eventos seleccionados en el destino que se haya seleccionado, normalmente un fichero de traza en unidad de red local al servidor ([H16]). Vamos a detallar los pasos a seguir para conseguir generar el código *T-SQL* a partir de la definición de una traza a través del interfaz gráfico.

Una vez definida la traza, lo que hay que hacer es guardar la definición de la traza. En lugar de elegir en el menú 'Save As...' un formato concreto (fichero XML, fichero .trc, tabla de base de datos, etc.), lo que haremos será elegir del

menú 'File', el menú 'Export', 'Script Trace Definition', y elegiremos 'For SQL Server 2005'. Lo que nos pedirá es guardar la traza como fichero T-SQL.

Ya podemos cerrar *SQL Profiler* porque ya no lo vamos a necesitar más para crear nuestra traza. Lo siguiente es personalizar la definición de la traza.

Ahora tenemos que editar el fichero con SQL Server Management Studio.

Abrimos el fichero .sql e 'identificaremos' varias secciones:

- Tamaño del fichero de captura

```
-- Create a Queue
declare @rc int
declare @TraceID int
declare @maxfilesize bigint
set @maxfilesize = 5
```

El valor @maxfilesize indica el tamaño que tendrá el fichero en MB. Por defecto, el tamaño del fichero es de 5MB. Lo personalizaremos para nuestras necesidades concretas, se aconseja crear ficheros del orden de 100 o 200MB para tener un periodo de actividad lo suficientemente grande, cuantos más eventos añadamos a la definición de la traza, más fácilmente se llenará el fichero; por ejemplo, si añadimos eventos relativos al plan de ejecución, el fichero se llenará rápidamente.

Independientemente del tamaño del fichero, SQL Server 'vuelca' al fichero de traza información en trozos de 128Kb, así que el tamaño del fichero va creciendo en múltiplos de 128.

- Ubicación donde se guardará el fichero de traza

-- Please replace the text InsertFileNameHere, with an appropriate

-- filename prefixed by a path, e.g., c:\MyFolder\MyTrace. The .trc extension

-- will be appended to the filename automatically. If you are writing from

-- remote server to local drive, please use UNC path and make sure server has

-- write access to your network share

exec @rc = sp_trace_create @TraceID output, 0, N'InsertFileNameHere', @maxfilesize, NULL

Reemplazaremos el texto 'InsertFileNameHere' por la ruta donde se guardará el fichero. Tendremos en cuenta que la ruta debe ser una ruta del mismo servidor para evitar el tráfico que red que comentamos anteriormente. El nombre del fichero no es necesario que tenga extensión .trc porque SQL Server automáticamente lo añade por nosotros.

- Columnas y Eventos a capturar

La siguiente sección muestra una serie de ejecuciones del procedimiento almacenado “sp_trace_setevent”, que lo que hace es definir en la traza las columnas y eventos que se quieren capturar.

….

exec sp_trace_setevent @TraceID, 10, 12, @on

exec sp_trace_setevent @TraceID, 10, 13, @on

exec sp_trace_setevent @TraceID, 10, 6, @on

exec sp_trace_setevent @TraceID, 10, 14, @on

exec sp_trace_setevent @TraceID, 12, 15, @on

exec sp_trace_setevent @TraceID, 12, 16, @on

exec sp_trace_setevent @TraceID, 12, 1, @on

exec sp_trace_setevent @TraceID, 12, 9, @on

exec sp_trace_setevent @TraceID, 12, 17, @on

….

- Definición de Filtros

La última sección incluye la definición de los filtros que se aplican; generalmente las trazas tendrán filtros del tipo:

Aplicación que se desea auditar

Nombre de usuario

Lecturas, escrituras, duración, uso de CPU mayor que cierto valor

Por ejemplo, el siguiente código excluye todos los eventos que vengan de la aplicación *"SQL Profiler":*

```
-- Set the Filters

declare @intfilter int

declare @bigintfilter bigint

exec sp_trace_setfilter @TraceID, 10, 0, 7, N'SQL Server Profiler'

-- Set the trace status to start

exec sp_trace_setstatus @TraceID, 1
```

Una vez modificado el código *T-SQL* de la traza, lo que queda es ejecutar la traza para que se ponga a capturar eventos. Para ello, lo único que tendremos que hacer es ejecutar el código *T-SQL* del fichero de traza, y listo. Si nos devuelve un valor mayor que cero indicará que la traza se ha iniciado correctamente; en caso contrario, habrá que verificar la definición de la traza para ver si es correcta.

- Cómo validar el estado de la traza

Para obtener información de las trazas, *SQL Server* dispone de una función definida (UDF) que muestra información de todas las trazas en funcionamiento:

```
SELECT * FROM ::fn_trace_getinfo(default)
```

Que devuelve los siguientes resultados:

traceid	property	value
1	1	2
1	2	C:\ProgramFiles\MicrosoftSQLServer\MSSQL.1\MSSQL\LOG\log_553.trc
1	3	20
1	4	NULL
1	5	1
2	1	0
2	2	c:\traza.trc
2	3	5
2	4	NULL
2	5	1

La columna “TraceId” muestra el identificador de la traza; en el ejemplo hay dos trazas en marcha, la traza 1 que es la traza por defecto de SQL Sever 2005, y la traza 2, que es la que hemos creado usando el “scritp” anterior. Hacemos notar que la “property” 2 indica donde se almacena la traza que podremos usar como TIP para identificar cual es la traza que creamos. Otro valor interesante es la “property” 3 que indica el tamaño máximo del fichero (en nuestro caso lo dejamos con 5MB). Para finalizar la “property” 5 indica el estado de la traza, en este caso 1 indica que se está ejecutando.

Para finalizar, si necesitamos cambiar el estado de la traza (por ejemplo pararla), utilizaremos el procedimiento almacenado de sistema "sp_trace_setstatus":

```
-- Parar la traza

EXEC sp_trace_setstatus id_traza, 0

-- Cerrar y borrar la traza

EXEC sp_trace_setstatus id_traza, 2
```

Donde "id_traza " es el identificador de la traza que hemos creado.

Por tanto, hemos conseguido crear trazas de "*SQL Profiler*" sin necesidad de tener en marcha la herramienta de captura de trazas. Dependiendo del servicio que tenga que atender tu servidor de bases de datos, estarás más o menos necesitado de que nuestros scripts de auditoría tengan más o menos impacto en el rendimiento del servidor. El paso siguiente será consultar la información capturada desde "*SQL Server Profiler*", o desde *T-SQL*.

Los ficheros de trazas se pueden abrir perfectamente desde *SQL Profiler*, pero en muchas ocasiones necesitaremos buscar información en la traza, y para ello es mejor utilizar *T-SQL*. El fichero de traza se puede consultar directamente usando la función definida por el sistema *fn_trace_gettable*, teniendo como argumento el nombre del fichero de traza a leer:

```
Select *
        from ::fn_trace_gettable (
        'D:\Compartido\traza_uno.trc',
        default )
```

En definitiva, estamos recogiendo un conjunto de datos que son el producto de unas preguntas que puede hacer un auditor. Así podemos hacernos preguntas del tipo.

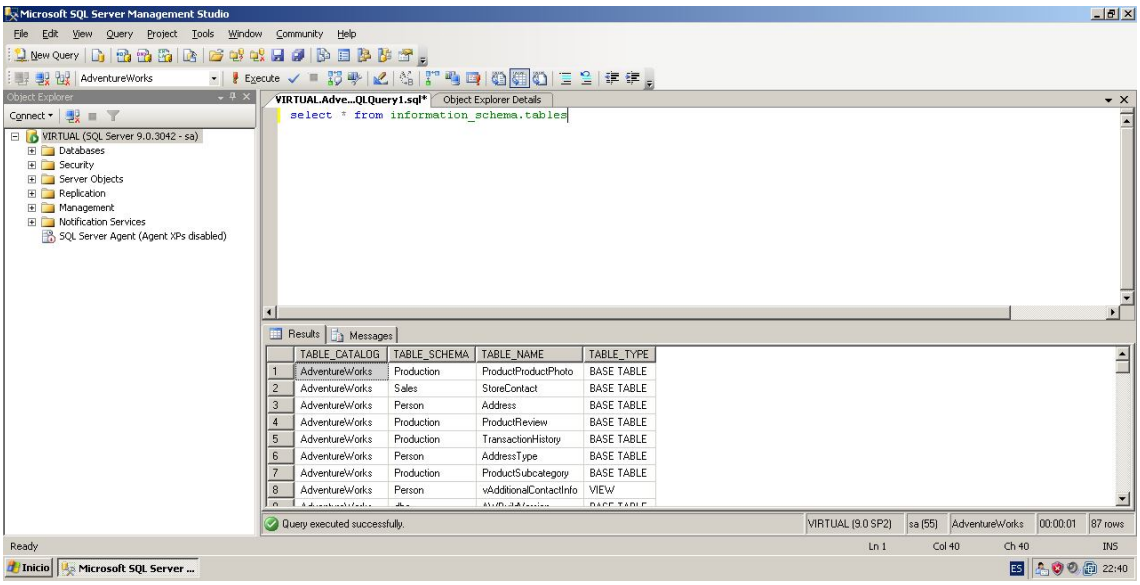
- ¿Quién hizo una operación entre dos instantes de tiempo?
- ¿Qué operaciones realizó el usuario dos instantes de tiempo?
- ¿Cuántas operaciones de un mismo tipo han sido realizadas por parte de un usuario determinado y en desde qué terminal?

Dando respuesta a estas preguntas podremos realizar informes de actividad de un usuario y analizar su productividad.

5.2.3 SQL Profiler y Performance Monitor.

Veremos el trabajo conjunto del Analizador de SQL Server y del monitor de rendimiento de *Microsoft Windows.* Se trata de cruzar una traza con el monitor de rendimiento ([H15]). El Monitor del sistema de Windows registra la actividad del sistema para contadores específicos en registros de rendimiento.

Queremos monitorizar bases de datos para evaluar el rendimiento de un servidor. Una supervisión efectiva implica tomar instantáneas periódicas del rendimiento actual para aislar los procesos que causan problemas y recopilar datos continuamente en el tiempo para hacer un seguimiento de las tendencias del rendimiento. El Analizador de SQL Server puede establecer correlaciones de contadores del Monitor del sistema de Microsoft Windows con eventos de SQL Server.



En *SQL Server 2005* se arranca “SQL Server Management Studio”, y nos conectamos al servidor, elegimos la base de datos de ejemplo “Adventureworks”, y comprobamos que la conexión es buena haciendo una consulta.

select * from information_schema.tables

Abrimos “SQL Server Profiler” desde Tools como usuario “sa”, creando una nueva traza “traza_monitor.trc”, a partir de la plantilla “standard”.

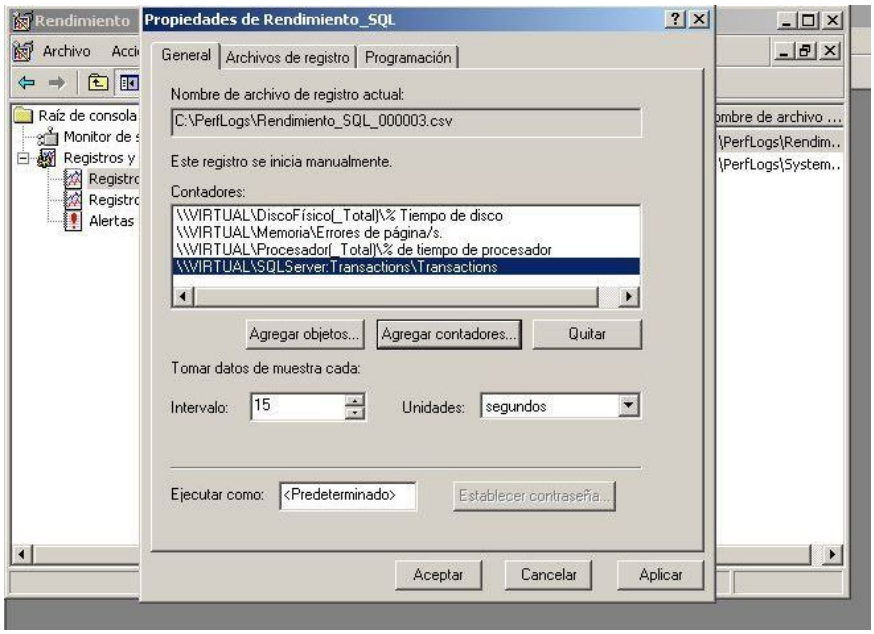
Ahora en menú “Tools”, tenemos la opción “Performance Monitor”, que abre la aplicación.

Hemos creado una nueva configuración para el registro de contador y lo llamamos “Rendimiento_SQL” y el monitor de rendimiento se almacenara en un fichero llamado así, “Rendimiento_SQL_00000x”.

Almacenamos con el registro de 4 contadores.

De procesador:

a. “% de tiempo de procesador”. % de tiempo de procesador se expresa como un porcentaje del tiempo que un procesador invierte ejecutando un subproceso activo. Se calcula midiendo durante cuánto tiempo el subproceso no activo está activo en el intervalo modelo. (Cada procesador tiene un subproceso inactivo que consume ciclos cuando otros subprocesos no están preparados para ejecutarse). Este contador es el indicador primario de la actividad del procesador, y muestra el porcentaje medio del tiempo ocupado observado durante el intervalo modelo. Se calcula monitorizando el tiempo que el servicio ha estado inactivo, y sustrayendo este valor a 100%.



Disco Físico (en la ventana de la derecha se puede seleccionar un disco o todos en caso de que hubiere):

a. “% Tiempo de Disco”.

% de tiempo de disco es el porcentaje de tiempo durante el cual la unidad de disco seleccionada estuvo ocupada atendiendo peticiones de lectura o escritura.

Memoria:

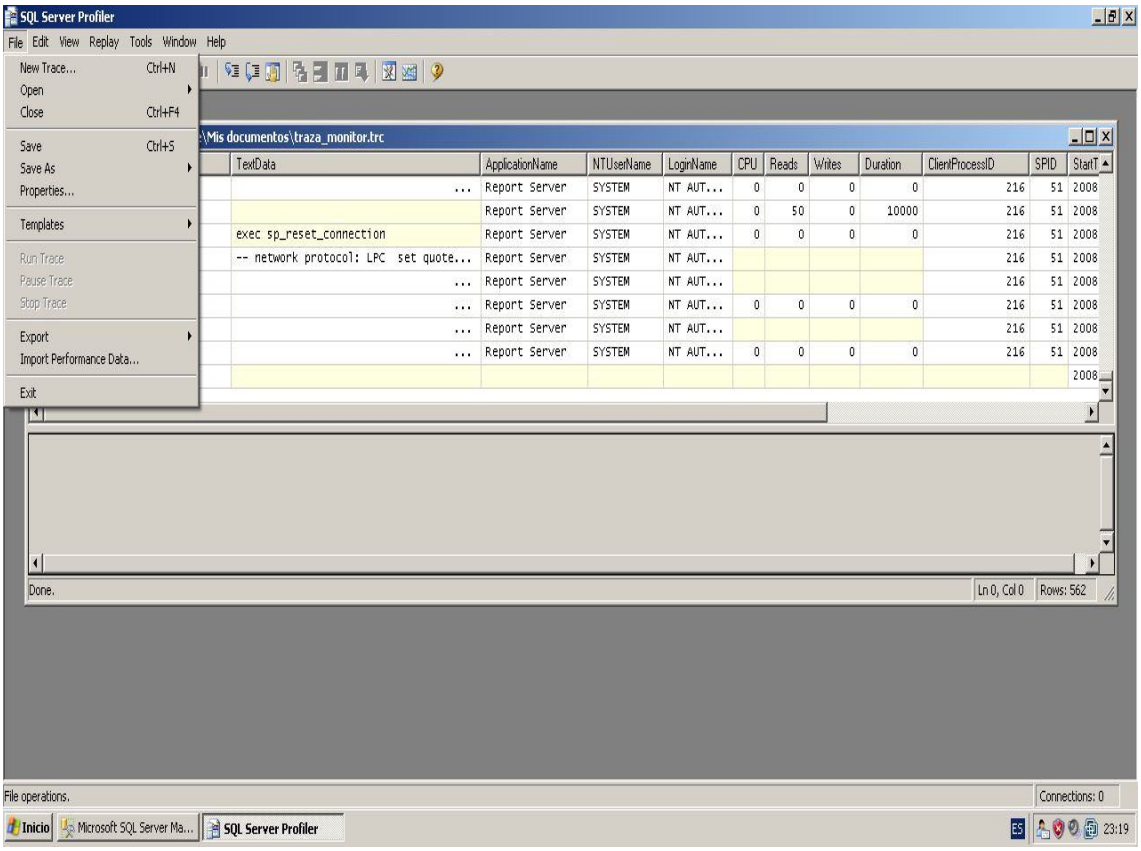
b. “Errores de página por segundo”.

Errores de página por segundo es el número de páginas fallidas por segundo. La medición se realiza en número de páginas fallidas por segundo porque sólo una página falla en cada operación fallida, por lo tanto, este es también igual al número de operaciones fallidas de páginas. Este contador incluye, tanto errores severos (aquellos que necesitan acceso al disco), como errores flexibles (donde la página con error se encuentra en algún otro sitio de la memoria física). Muchos procesos soportan gran número de errores flexibles sin consecuencias significantes. Sin embargo, los errores severos, que requieren acceso a disco, pueden producir retrasos importantes.

c. “SQL :Transactions”:

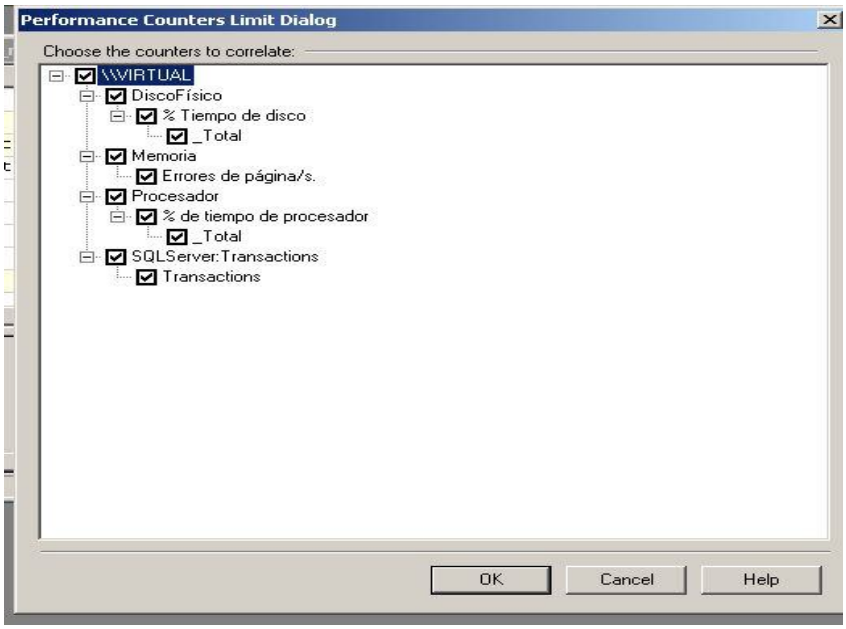
Número de transacciones activas actualmente de todos los tipos en la instancia de bases de datos

Arrancamos el monitor de rendimiento y arrancamos la traza.

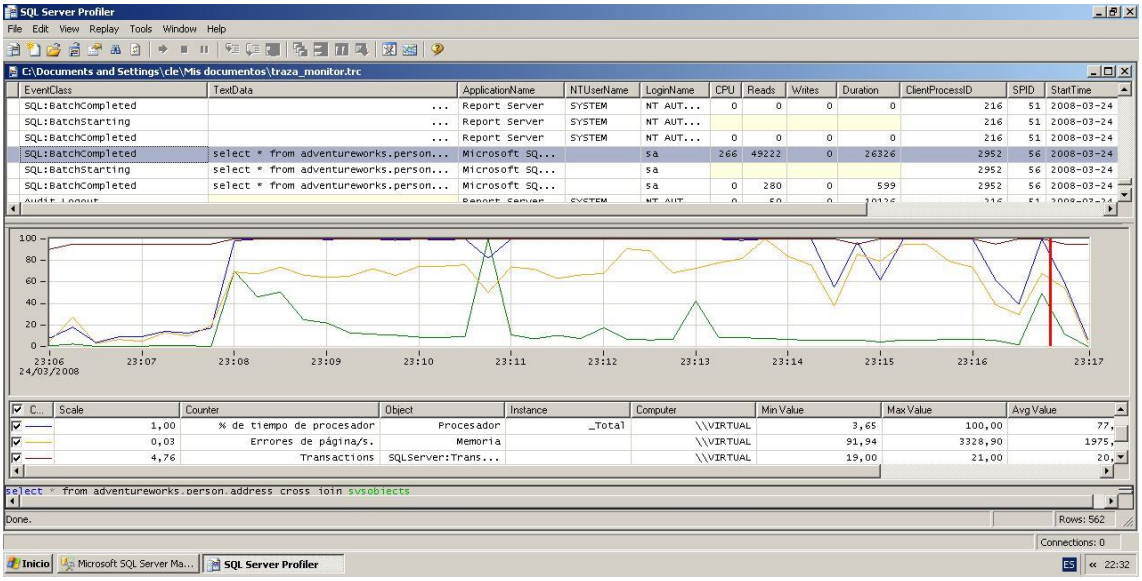


Ya estamos monitorizando que pasa en nuestro SQL Server.

Seleccionamos todos los contadores.



Obtenemos una sincronización de la traza capturada en “Profiler” y la captura del monitor de rendimiento.



En este caso la consulta:

select * from adventureworks.person.address cross join sysobjects.

El evento “SQL:BatchCompleted” requiere el 100 % del tiempo de procesador, 2264 páginas fallidas por segundo, 21 transacciones realizadas, y un porcentaje de uso del disco en operaciones de lectura ó escritura del 97,3

Ahora podemos ir pinchando en instrucciones y ver cuál era el consumo de recursos mientras se usaban y al revés, de esta forma no solo tenemos las consultas más costosas por que “Profiler” lo dice sino que además podemos ver el impacto que han tenido en nuestro servidor mientras se estaban ejecutando.

6 Auditar Sentencias DDL

En la medida que crece la necesidad de auditar la actividad en los servidores SQL, revisamos qué herramientas nativas traen las nuevas versiones del motor. Adicionalmente al *Profiler*, a las trazas en background, a los *triggers DML*, aparecen en la versión 2005 de SQL los *triggers DDL (Data Definition Language)* ([H17]).

6.1 *Descripción de los disparadores DDL.*

Frente a los *disparadores DML*, que hemos descrito en el punto 4.1.2, los *disparadores DDL* se utilizan finalidades distintas.

Los *disparadores DML* funcionan con las instrucciones *INSERT, UPDATE y DELETE,* y permiten exigir las reglas de la compañía y extender la integridad de los datos cuando se modifican datos en tablas o vistas.

Los *disparadores DDL* funcionan en las instrucciones *CREATE, ALTER, DROP* y otras instrucciones *DDL*. Se utilizan para realizar tareas administrativas y exigir las reglas de la compañía que afectan a las bases de datos. Se aplican a todos los comandos de un solo tipo en toda una base de datos o todo un servidor.

Los *disparadores DDL y DML* se crean, modifican y quitan con una sintaxis *Transact-SQL* similar y tienen un comportamiento parecido.

Al igual que los *disparadores DML*, los *DDL* pueden ejecutar código administrado empaquetado en un ensamblado creado en *Microsoft .NET Framework* y cargado en *SQL Server*. Se pueden crear varios *disparadores DDL* en la misma instrucción *Transact-SQL*. Asimismo, un *disparador DDL* y la instrucción que lo activa se ejecutan en la misma transacción. Esta transacción se puede revertir desde el desencadenador. Los errores graves pueden hacer que se revierta automáticamente la totalidad de una transacción. Si se ejecutan *disparadores DDL* desde un lote y se incluye explícitamente la instrucción *ROLLBACK TRANSACTION*, se cancelará todo el lote. Otra característica de

disparadores DDL es que se pueden anidar, estamos hablando de un disparador que se activa como resultado de la emisión de una instrucción emitida por otro disparador.

Al diseñar *disparadores DDL*, tendremos en cuenta lo que los diferencia de los *DML*:

- Sólo se ejecutan una vez completada una instrucción *Transact-SQL*. Los *disparadores DDL* no se pueden utilizar como *disparadores INSTEAD OF*.

- No crean las tablas **inserted** y **deleted**. La información acerca de un evento que activa un *disparador DDL* y las modificaciones posteriores provocadas por el mismo se capturan con la función *EVENTDATA()* ([H19]).

Los *trigger*s *DDL* pueden utilizarse para tareas administrativas como auditar y regular las operaciones de base de datos, siendo posible utilizarlos cuando:

- Deseemos evitar determinados cambios en el esquema de base de datos.

- Vayamos buscando que ocurra algún evento en la base de datos como respuesta a un cambio realizado en el esquema de base de datos.

- Tratemos de registrar cambios o eventos del esquema de base de datos.

6.2 *Diseño de los disparadores DDL*

A la hora de diseñar *trigger*s *DDL* debemos comprender el ámbito al que afecta su desencadenamiento y debemos determinar la instrucción *Transact-SQL* o el grupo de instrucciones que lo activan.

Respecto a la primera consideración, los *trigger*s *DDL* pueden activarse en respuesta a un evento de *Transact-SQL* procesado en la base de datos actual o en el servidor actual. El ámbito del disparador depende del evento. Por ejemplo, un

*trigger*s *DDL* creado para activarse como respuesta a un evento CREATE TABLE se activará siempre que se produzca un evento CREATE TABLE en la base de datos. Un *trigger*s *DDL* creado para activarse como respuesta a un evento CREATE LOGIN se activará siempre que se produzca un evento CREATE LOGIN en el servidor.

En el ejemplo siguiente, el *trigger*s *DDL* "seguro" se activará siempre que se produzca un evento DROP TABLE o ALTER TABLE en la base de datos:

```
CREATE TRIGGER seguro ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
  PRINT 'Debe desactivar el Trigger "seguro" para borrar, cambiar tablas!'
  ROLLBACK ;
```

En el ejemplo siguiente, un *trigger*s *DDL* imprime un mensaje si se produce algún evento CREATE DATABASE en la instancia de servidor actual. Utilizaremos la función EVENTDATA ([H19]) para recuperar el texto de la instrucción *Transact-SQL* correspondiente.

```
IF EXISTS (SELECT * FROM sys.server_triggers
   WHERE name = 'ddl_trig_database')
DROP TRIGGER ddl_trig_database
ON ALL SERVER;
GO
CREATE TRIGGER ddl_trig_database
ON ALL SERVER
FOR CREATE_DATABASE
AS
   PRINT 'Base de datos creada.'
         SELECT
      EVENTDATA().value('(/EVENT_INSTANCE/TSQLCommand/C
      ommandText)[1]','nvarchar(max)')
GO
DROP TRIGGER ddl_trig_database
ON ALL SERVER;
GO
```

Los *trigger*s *DDL* de ámbito de base de datos se almacenan como objetos en la base de datos donde se crean. Estos desencadenadores se pueden crear en la base de datos *master* y actuar como los creados en bases de datos diseñadas por el usuario. Se puede obtener información sobre los *trigger*s *DDL* en la vista de catálogo **sys.triggers** desde el contexto de base de datos donde se crean o bien mediante la especificación del nombre de base de datos como un identificador (por ejemplo, **master.sys.triggers**).

SELECT type, name, parent_class_desc FROM sys.triggers

Los *trigger*s *DDL* de ámbito de servidor se almacenan como objetos en la base de datos ***master***. Sin embargo, se puede obtener información sobre los *trigger*s *DDL* de ámbito de servidor desde la vista de catálogo **sys.server_triggers** en cualquier contexto de base de datos.

SELECT type, name, parent_class_desc FROM sys.server_triggers

Los *trigger*s *DDL* no se activan como respuesta a eventos que afectan a procedimientos almacenados y tablas temporales, ya sean locales o globales.

La función *EventData()* ([H19]) devuelve información acerca de los eventos del servidor ó de los eventos de la base de datos, esta información es devuelta en un formato tipo *XML (Extensible Markup Language*), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C).

En el siguiente diagrama se muestran los grupos de eventos que se pueden utilizar para activar un *trigger*s *DDL* ([H18]), las instrucciones *Transact-SQL* que abarcan y el ámbito donde se pueden programar (ON SERVER u ON DATABASE). No fijamos en la naturaleza inclusiva de los grupos de eventos, según indica la estructura de árbol. Por ejemplo, un *desencadenador DDL* que especifica FOR DDL_TABLE_EVENTS abarca las instrucciones CREATE TABLE, ALTER TABLE y DROP TABLE de *Transact-SQL*; un *trigger*s *DDL* que especifica FOR DDL_TABLE_VIEW_EVENTS abarca todas las

instrucciones *Transact-SQL* incluidas en *DDL_TABLE_EVENTS, DDL_VIEW_EVENTS, DDL_INDEX_EVENTS y DDL_STATISTICS_EVENTS.*

	Ámbito del servidor	Ámbito de la base de datos
DDL_SERVER_LEVEL_EVENTS (CREATE DATABASE, ALTER DATABASE, DROP DATABASE)	X	
DDL_ENDPOINT_EVENTS (CREATE ENDPOINT, ALTER ENDPOINT, DROP ENDPOINT)	X	
DDL_SERVER_SECURITY_EVENTS	X	
DDL_LOGIN_EVENTS (CREATE LOGIN, ALTER LOGIN, DROP LOGIN)	X	
DDL_GDR_SERVER_EVENTS (GRANT SERVER, DENY SERVER, REVOKE SERVER)	X	
DDL_AUTHORIZATION_SERVER_EVENTS (ALTER AUTHORIZATION SERVER)	X	
DDL_DATABASE_LEVEL_EVENTS		X
DDL_TABLE_VIEW_EVENTS		X
DDL_TABLE_EVENTS (CREATE TABLE, ALTER TABLE, DROP TABLE)		X
DDL_VIEW_EVENTS (CREATE VIEW, ALTER VIEW, DROP VIEW)		X
DDL_INDEX_EVENTS (CREATE INDEX, ALTER INDEX, DROP INDEX, CREATE XML INDEX)		X
DDL_STATISTICS_EVENTS (CREATE STATISTICS, UPDATE STATISTICS, DROP STATISTICS)		X
DDL_SYNONYM_EVENTS (CREATE SYNONYM, DROP SYNONYM)		X
DDL_FUNCTION_EVENTS (CREATE FUNCTION, ALTER FUNCTION, DROP FUNCTION)		X
DDL_PROCEDURE_EVENTS (CREATE PROCEDURE, ALTER PROCEDURE, DROP PROCEDURE)		X
DDL_TRIGGER_EVENTS (CREATE TRIGGER, ALTER TRIGGER, DROP TRIGGER)		X
DDL_EVENT_NOTIFICATION_EVENTS (CREATE EVENT NOTIFICATION, DROP EVENT NOTIFICATION)		X
DDL_ASSEMBLY_EVENTS (CREATE ASSEMBLY, ALTER ASSEMBLY, DROP ASSEMBLY)		X
DDL_TYPE_EVENTS (CREATE TYPE, DROP TYPE)		X
DDL_DATABASE_SECURITY_EVENTS		X
DDL_CERTIFICATE_EVENTS (CREATE CERTIFICATE, ALTER CERTIFICATE, DROP CERTIFICATE)		X
DDL_USER_EVENTS (CREATE USER, ALTER USER, DROP USER)		X
DDL_ROLE_EVENTS (CREATE ROLE, ALTER ROLE, DROP ROLE)		X
DDL_APPLICATION_ROLE_EVENTS (CREATE APPROLE, ALTER APPROLE, DROP APPROLE)		X
DDL_SCHEMA_EVENTS (CREATE SCHEMA, ALTER SCHEMA, DROP SCHEMA)		X
DDL_GDR_DATABASE_EVENTS (GRANT DATABASE, DENY DATABASE, REVOKE DATABASE)		X
DDL_AUTHORIZATION_DATABASE_EVENTS (ALTER AUTHORIZATION DATABASE)		X
DDL_SSB_EVENTS		X
DDL_MESSAGE_TYPE_EVENTS (CREATE MSGTYPE, ALTER MSGTYPE, DROP MSGYTPE)		X
DDL_CONTRACT_EVENTS (CREATE CONTRACT, DROP CONTRACT)		X
DDL_QUEUE_EVENTS (CREATE QUEUE, ALTER QUEUE, DROP QUEUE)		X
DDL_SERVICE_EVENTS (CREATE SERVICE, ALTER SERVICE, DROP SERVICE)		X
DDL_ROUTE_EVENTS (CREATE ROUTE, ALTER ROUTE, DROP ROUTE)		X
DDL_REMOTE_SERVICE_BINDING_EVENTS (CREATE REMOTE SERVICE BINDING, ALTER REMOTE SERVICE BINDING, DROP REMOTE SERVICE BINDING)		X
DDL_XML_SCHEMA_COLLECTION_EVENTS (CREATE XML SCHEMA COLLECTION, ALTER XML SCHEMA COLLECTION, DROP XML SCHEMA COLLECTION)		X
DDL_PARTITION_EVENTS		X
DDL_PARTITION_FUNCTION_EVENTS (CREATE PARTITION FUNCTION, ALTER PARTITION FUNCTION, DROP PARTITION FUNCTION)		X
DDL_PARTITION_SCHEME_EVENTS (CREATE PARTITION SCHEME, ALTER PARTITION SCHEME, DROP PARTITION SCHEME)		X

La tabla siguiente muestra los eventos que se pueden capturar.

SENTENCIA	Alcance: SERVER	Alcance: DATABASE
ADD_ROLE_MEMBER	X	X
ADD_SERVER_ROLE_MEMBER	X	
CREATE_APPLICATION_ROLE (Applies to CREATE APPLICATION ROLE statement and **sp_addapprole**. If a new schema is created, this event also triggers a CREATE_SCHEMA event.)	X	X
ALTER_APPLICATION_ROLE (Applies to ALTER APPLICATION ROLE statement and **sp_approlepassword**.)	X	X
DROP_APPLICATION_ROLE (Applies to DROP APPLICATION ROLE statement and **sp_dropapprole**.)	X	X
CREATE_ASSEMBLY	X	X
ALTER_ASSEMBLY	X	X
DROP_ASSEMBLY	X	X
ALTER_AUTHORIZATION_SERVER	X	
ALTER_AUTHORIZATION_DATABASE (Applies to ALTER AUTHORIZATION statement when ON DATABASE is specified, and **sp_changedbowner**.)	X	X
CREATE_CERTIFICATE	X	X
ALTER_CERTIFICATE	X	X
DROP_CERTIFICATE	X	X
CREATE_CONTRACT	X	X
DROP_CONTRACT	X	X
CREATE DATABASE	X	
ALTER DATABASE	X	X
DROP DATABASE	X	

Estos *trigger*s podrían registrar información útil como pista de auditoría como la cuenta Windows de la *session* que lo realize, desde qué *host,* usuario de BD, fecha, base, etc, tanto en una tabla que se haya creado para tal fin, como también escribir el evento en el *Error log de SQL.*

Los *trigger*s *DDL* se pueden diseñar para activarse después de ejecutar una o varias instrucciones *Transact-SQL* determinadas. En el ejemplo anterior, el *trigger*s *DDL* "seguro" se activa después de un evento DROP TABLE o ALTER TABLE.

No todos los eventos DDL pueden utilizarse para activar *trigger*s *DDL*. Algunos eventos están diseñados únicamente para instrucciones asincrónicas no transaccionales. Por ejemplo, no se puede utilizar un evento ADD_ROLE_MEMBER para activar un *trigger*s *DDL*. Para estos eventos debe utilizar notificaciones de eventos, de este tema trataremos más adelante en el capítulo dedicado a *Service Broker*.

Los *trigger*s *DDL* se pueden activar tras la ejecución de un evento de *Transact-SQL* que pertenezca a una agrupación predefinida de eventos similares. Por ejemplo, si desea activar un *desencadenador DDL* después de que se ejecute una instrucción CREATE TABLE, ALTER TABLE o DROP TABLE, puede especificar FOR DDL_TABLE_EVENTS en la instrucción CREATE TRIGGER. Después de ejecutar CREATE TRIGGER, los eventos incluidos en un grupo de eventos se agregan a la vista de catálogo **sys.trigger_events**.

De tal manera si queremos registrar la actividad de DDL relacionada con una base de datos, no es necesario definir un *trigger DDL* para las tablas por un lado, para las vistas por otro, otro trigger para definición *alter-drop* de *stored procedures*. Se define un solo *trigger* para toda la base de datos (DDL_DATABASE_LEVEL_EVENTS).

Vamos ahora por unos ejemplos. El primero es de *trigger DDL* cuyo ámbito es todo el Servidor SQL. En el mismo se graba un mensaje en el Log de SQL y en el Application Log de Windows, donde aparece como origen: MSSQLSERVER y EVENT: 17061.

```
CREATE TRIGGER servertrigger
ON ALL SERVER
FOR DDL_LOGIN_EVENTS
AS
DECLARE @data XML
DECLARE @MESSAGE varchar(255)
DECLARE @comando varchar(255)
DECLARE @Usuario NVARCHAR(100)
DECLARE @Server NVARCHAR(100)
SET @data = EVENTDATA()
SET @Usuario = SYSTEM_USER
SET @Server = @@servername
SELECT @ comando=
EVENTDATA().value('(/EVENT_INSTANCE/EventType)[1]','nvarchar(max)') +
EVENTDATA().value('(/EVENT_INSTANCE/ObjectName)[1]','nvarchar(max)')

SELECT @MESSAGE = 'DDL en SERVER: ' + @Server + ';' + @Usuario + ';' +
@comando
EXEC master.dbo.xp_logevent 50001, @MESSAGE, informational
GO
```

Ahora vamos por un ejemplo a nivel base de datos. En este caso se graba un registro en una tabla creada para tal fin y también se escribe en Logs. Se utiliza un evento global para capturar cualquier actividad de DDL que ocurra en la base.

```
USE [basex]
GO
CREATE TRIGGER [ddlenbasex]
ON DATABASE
FOR DDL_DATABASE_LEVEL_EVENTS
AS
DECLARE @data XML
DECLARE @MESSAGE varchar(255)
DECLARE @DBNAME NVARCHAR(128)
DECLARE @Usuario NVARCHAR(100)
DECLARE @comando NVARCHAR(128)
SET @DBNAME = DB_NAME()
SET @data = EVENTDATA()
SET @Usuario = SYSTEM_USER
SET @comando = @data.value('(/EVENT_INSTANCE/TSQLCommand)[1]',
'nvarchar(2000)')
INSERT DDL_log
(Fecha, Login, Usuario, Evento, Sentencia)
VALUES
(GETDATE(),
CONVERT(nvarchar(100), SYSTEM_USER),
CONVERT(nvarchar(100), CURRENT_USER ),
```

```
@data.value('(/EVENT_INSTANCE/EventType)[1]', 'nvarchar(100)'),
@data.value('(/EVENT_INSTANCE/TSQLCommand)[1]', 'nvarchar(2000)') )
SELECT @@MESSAGE = 'DDL en base de datos: ' + @DBNAME + ';' + @Usuario +
';' + @comando
EXEC master.dbo.xp_logevent 50001, @@MESSAGE, informational
GO
```

Podemos concluir este capítulo diciendo que los *disparadores DDL* son una nueva herramienta muy potente, que nos permitirá poder tener control sobre las sentencias DDL y así también poder realizar auditorías a las mismas. Con esta información podemos hacer informes que nos indiquen, por ejemplo:

- Cuales tablas han sufridos cambios recientemente.
- Cuales tablas no sufrieron cambios el pasado año.
- Cuales tablas jamás han sufrido cambios.
- Mostrar todos los cambios a las tablas por un usuario y periodo específico.
- Mostrar las tablas más activas en un determinado periodo.

De nuevo, la contrapartida es que sobrecargan el sistema ya que cuando se produce el evento que activa al disparador y se ejecutan las acciones programadas, los recursos del sistema dedicarán un tiempo variable a estas acciones. En el capítulo dedicado a la tecnología *Service Broker* trataremos de minimizar con su uso el impacto de sobre el rendimiento del sistema.

7 Rendimiento en la recolección de datos en SQL Server 2005

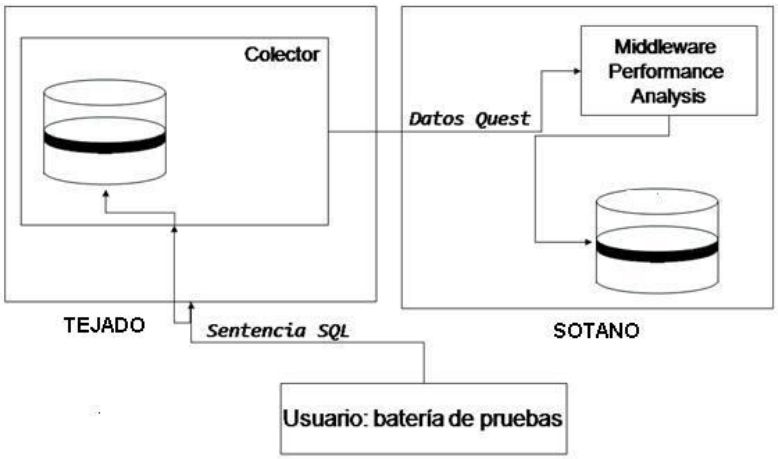
Describiremos como se han realizado una serie de pruebas efectuadas en una base de datos sobre *SQL Server 2005.* Buscamos mediciones que nos permitan analizar el rendimiento en diferentes situaciones.

a) Cuando no se audita nada.
b) Cuando está activado el modo de auditoría "C2".
c) Cuando se utiliza la herramienta *SQL Server Profiler.*

Vamos a disponer de una serie de ordenadores y sistemas en los que realizaremos las pruebas pensadas para conseguir obtener un alto grado de uso de sistema gestor de bases de datos *SQL Server 2005*, con estas pruebas tratamos de simular diferentes maneras de trabajar sobre la base de datos.

Usamos las pruebas realizadas cuando no auditamos nada como línea base, siendo la referencia para poder hacer comparaciones con las pruebas que se realizan con los dos tipos de auditoría anteriormente señalados.

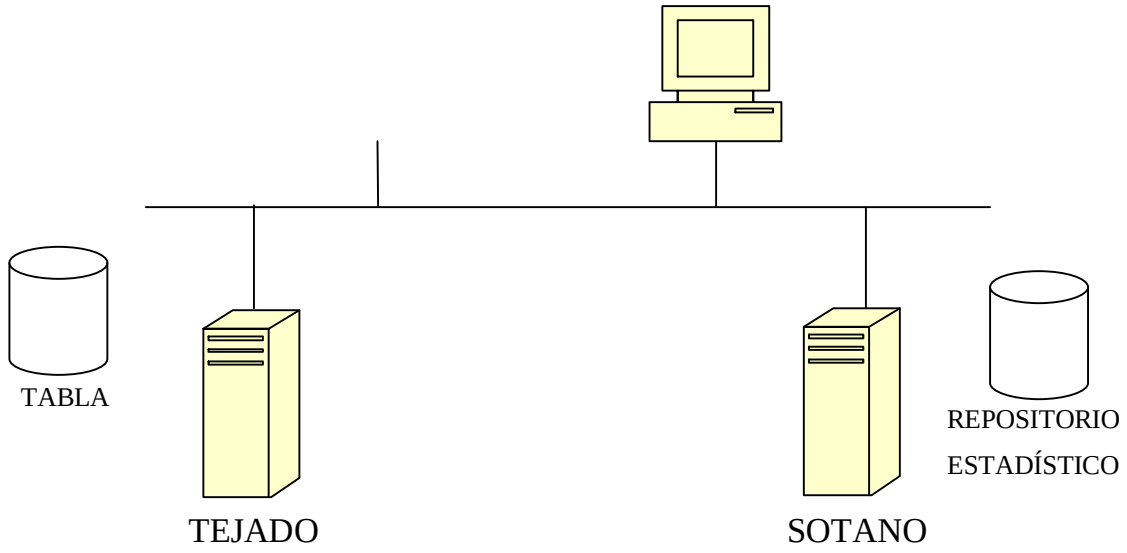
Seguidamente hacemos una descripción detallada de la infraestructura de la que disponemos, de las pruebas que vamos a realizar, y por último de los resultados y conclusiones de nuestro análisis.



7.1 *Infraestructura*

Hemos creado una infraestructura de tres ordenadores en la que cada una tiene una rol distinto; uno es el servidor de la base de datos y los otros dos son los clientes con funciones diferenciadas.

En la siguiente figura se muestra un esquema donde se reproduce la situación de las pruebas.



El ordenador llamado TEJADO realiza la función de servidor de la base de datos. El proyecto va destinado a medir el rendimiento en la plataforma Windows, elegimos por tanto un sistema operativo *Windows XP Profesional versión 2002 SP2*. La primera intención fue utilizar *Windows Server 2008* pero problemas de compatibilidad con la herramienta de medición de rendimiento *Performance Analysis* nos lo ha impedido. Se ha instalado SQL Server 2005 SP1, donde damos de alta un usuario *pfc_admin* que utiliza la base de datos *uc3m* objeto de la auditoría. La CPU consta de un procesador Intel® Pentium® 4 a 3,2 GHz con 3 GB de RAM y 80 GB de disco duro.

El ordenador llamado SOTANO realiza la función de cliente de la base de datos. Dispone de una instancia de *SQL Server 2*005 para almacenar el repositorio que necesita la herramienta *Performance Analysis*. El sistema

operativo *Windows XP Profesional versión 2002 SP2*. La CPU consta de un procesador Intel® Pentium® 4 a 3,4 GHz con 2 GB de RAM y 150 GB de disco duro.

En cuanto a la medición del rendimiento el equipo llamado SOTANO tendrá un proceso intermedio (*Middleware*) que recoge los datos de rendimiento enviados por el equipo llamado TEJADO y los presenta en forma gráfica con el *Performance Analysis*. Así mismo, TEJADO tiene instalado un proceso llamado Colector (*Collector*) que se encarga de medir la información de rendimiento y de enviarla a SOTANO.

7.2 *Pruebas realizadas en la infraestructura*

Las pruebas se dividen en 4 tipos dependiendo de la operación que se realiza. Con el objeto de realizar las operaciones desde una maquina cliente cualquiera dentro de la infraestructura vamos a utilizar el programa *sqlcmd* que nos permite ejecutar secuencias de comando o *scripts* sql incluidos en un archivo.

Prueba	**Operación**	**Acción**	**Objetivo**
1	Inserción	Carga de datos	1 millón de registros
2	Consulta	Obtener resultado	100 consultas seguidas
3	Modificación	Cambiar contenido registro	Modificar registro sin parar en una sesión
4	Borrado	Eliminar registros	Borrar sin parar en una sesión

Ahora, más detalladamente explicaremos cada una de las pruebas que hemos realizado para capturar la información que nos ha servido para analizar el rendimiento del sistema monitorizado en las situaciones anteriormente expuestas.

- Prueba 1: operación de inserción. Esta prueba simula una carga de datos. Para insertar de un volumen masivo de registros hemos utilizado el programa de línea de comando *bcp*; es un programa de SQL Server que podemos ejecutar en un ordenador cliente con conexión a la base de datos cuya función principal es mover datos formateados y masivos dentro y fuera de SQL Server (*INSERT BULK uc3m.dbo.datos*). Nuestro objetivo es insertar al menos un millón de registros y las limitaciones de memoria de nuestra infraestructura no permite más de doscientos cincuenta mil inserciones de registros sin parar desde *SQLCMD* en una única sesión.
- Prueba 2: operación de consulta. Consiste de realizar consultas continuas. Cada consulta tiene un número diferente de registros de retorno hasta un máximo de 100, para que el proceso de envío de datos no normalice el rendimiento del ordenador. Los resultados de las consultas se envían a un fichero, para evitar la demora de salida por pantalla.
- Prueba 3: operación de modificación. Se trata de realizar modificaciones sin parar desde SQLCMD en una única sesión.
- Prueba 4: operación de borrado. Consiste en realizar borrados sin parar desde SQLCMD en una única sesión.

La tabla sobre la que se crean las auditorías es de longitud media, en la que hay tipos de datos variados para simular bases de datos generales, como currículums, historiales, etc.

Tiene la siguiente definición:

```
create table datos (
id int primary key,
dni int,
num float,
nombre varchar2(50),
apellido1 varchar2(50),
email varchar2(100),
email2 varchar2(100),
direccion varchar2(200),
direccion2 varchar2(200),
descripcion varchar2(4000)
);
```

Las gráficas que se muestran de cada prueba son las que ofrecen *Performance Analysis* de *Quest* ([H20]). En cada prueba mostramos los resultados de las mediciones y los gráficos comparativos asociados, para finalizar con un análisis de los resultados.

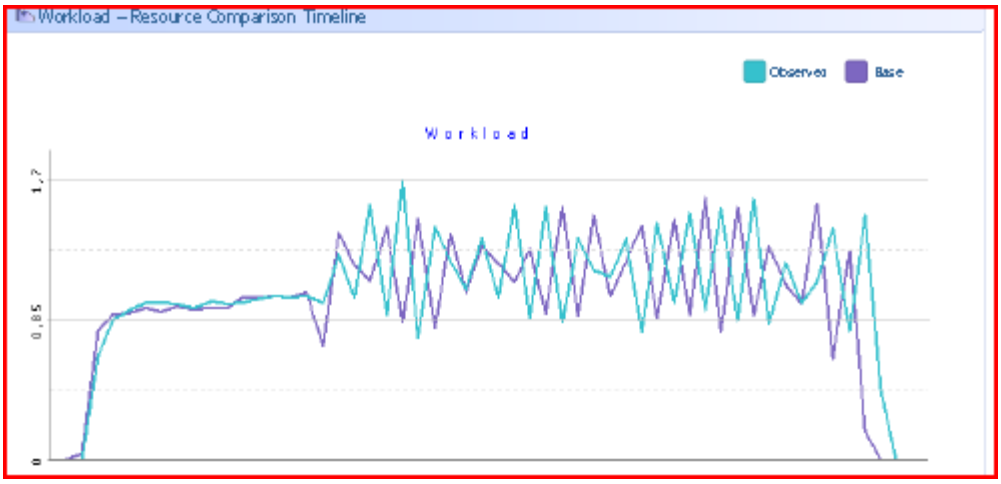
7.3 *Rendimiento del S.G.D.B con el modo de auditoría "C2"*

Hemos intentado ver las diferencias de rendimiento entre realizar las pruebas sin auditoría y realizar las pruebas con el modo de auditoría "C2" activado.

Como hemos comentados anteriormente al activar esta modalidad de motorización todos los eventos y columnas pertenecientes a cada evento se registraran en un fichero de traza.

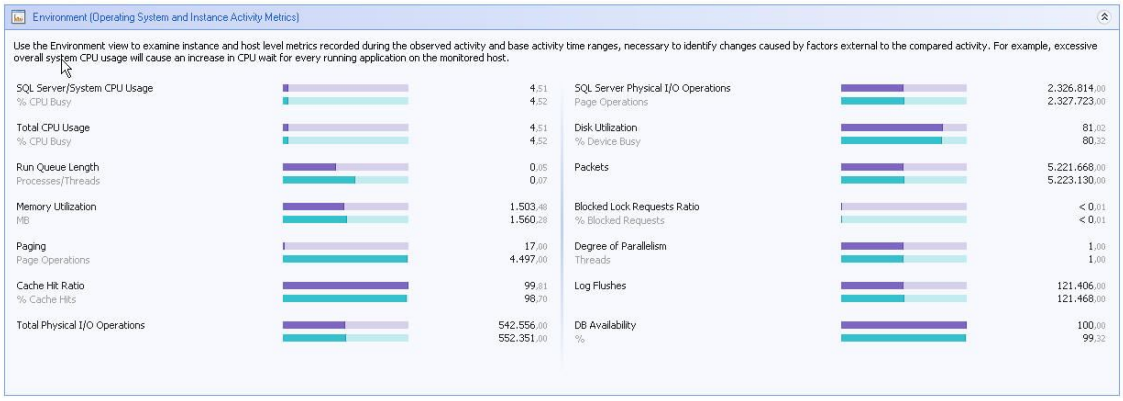
Mediante *Performance Analysis* generamos un informe comparativo de aquellos valores de rendimiento medidos durante el proceso de prueba. Tomamos como línea base de la comparación las pruebas realizadas sin tener activada la auditoría y la comparamos con las pruebas realizadas con el modo de auditoría "C2" activo.

Prueba 1.



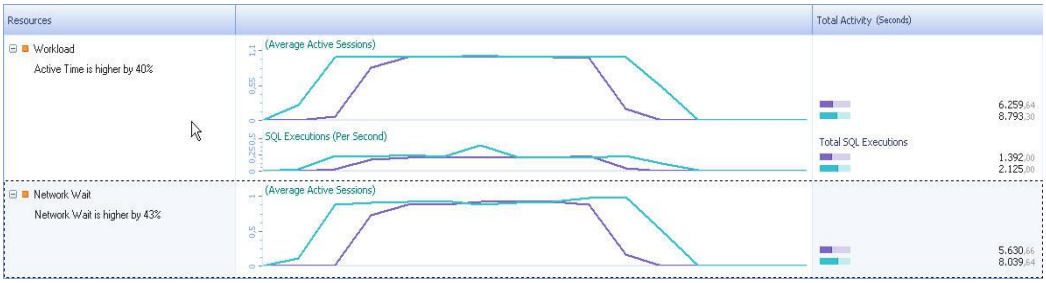
Global Environment Statistics Comparison

Statistic	Base	Observed	Change	
SQL Server/System CPU Usage % CPU Time	4,51	4,52	—	0,00
Total CPU Usage % CPU Time	4,51	4,52	—	0,00
Run Queue Length Processes/Threads	0,05	0,07	▲	0,02
Memory Utilization MB	1508,48	1560,28	▲	56,81
Paging Page Operations	17,00	4497,00	▲	4480,00
Cache Hit Ratio % Cache Hits	98,81	98,70	▼	-0,11
Total Physical I/O Operations	542536,00	552361,00	▲	9796,00
SQL Server Physical I/O Operations Page Operations	2326814,00	2327723,00	▲	909,00
Disk Utilization % Disk Time	81,02	80,32	▼	-0,70
Packets	5221688,00	5223130,00	▲	1462,00
Blocked Lock Requests Ratio % Blocked Requests	0,00	0,00	—	0,00
Degree of Parallelism Threads	1,00	1,00	—	0,00
Log Flushes	121406,00	121468,00	▲	62,00
DB Availability %	100,00	99,32	▼	-0,68



Hacemos notar como única diferencia importante el valor de paginación ó número de operaciones de paginación realizadas por el sistema operativo, se observa que con el modo “C2” activado aumenta considerablemente el valor, esto es motivado porque hay más operaciones de entrada y salida para almacenar la información de la monitorización. Normalmente este problema se soluciona aumentado la memoria RAM en el servidor y así conseguir que haya menos fallos de página y por tanto menos movimiento de I/O (entrada/salida) entre disco y memoria. Otro factor a tener en cuenta es al configuración que tengamos de la memoria de la instancia, en nuestro caso es dinámica, es decir el S.G.B.D. es el que va asignando memoria para que la operación de inserción se realice adecuadamente. Por tanto, hemos de tener cuidado si hacemos una asignación de memoria manual, porque si se asigna poca memoria SQL Server podría no gestionar las tareas en un periodo adecuado; y si se asigna demasiada podría ocupar recursos que serían necesarios para el funcionamiento del sistema, lo cual tendría un impacto directo en el rendimiento general de servidor, en este caso el equipo TEJADO.

Prueba 2.



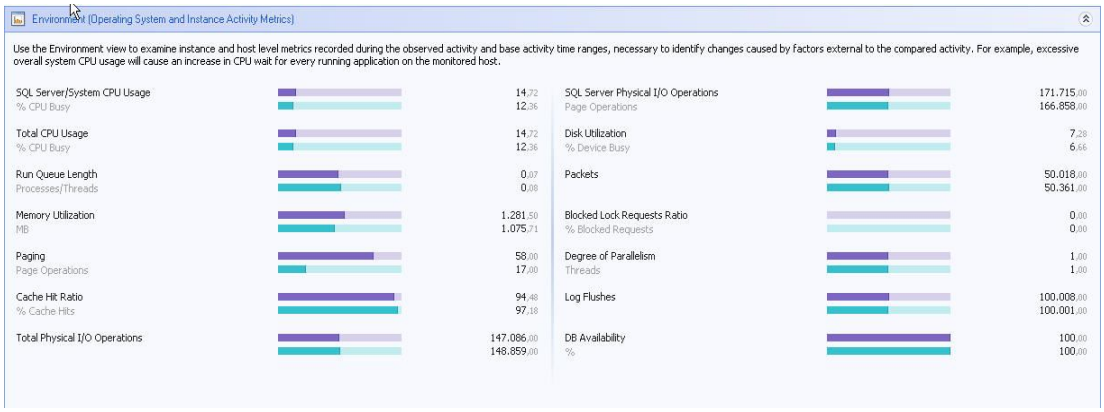
El primer dato que observamos es que la duración de la prueba aumenta un 40% y el tiempo de espera de red generados por las peticiones de consulta desde el cliente y respuestas del servidor aumenta un 43%. Estos dos porcentajes

son lo suficientemente altos para determinar lo negativo que es la activación del modo “C2” cuando se realizan operaciones de consultas sucesivas.

Prueba 3.

Global Environment Statistics Comparison

Statistic	Base	Observed	Change	
SQL Server/System CPU Usage % CPU Time	14,72	12,36	∨	-2,36
Total CPU Usage % CPU Time	14,72	12,36	∨	-2,36
Run Queue Length Processes/Threads	0,07	0,08	—	0,00
Memory Utilization MB	1281,50	1075,71	∨	-205,79
Paging Page Operations	58,00	17,00	∨	-41,00
Cache Hit Ratio % Cache Hits	94,48	97,18	∧	2,71
Total Physical I/O Operations	147086,00	148859,00	∧	1773,00
SQL Server Physical I/O Operations Page Operations	171716,00	166858,00	∨	-4857,00
Disk Utilization % Device Time	7,28	6,66	∨	-0,62
Packets	50018,00	50361,00	∧	343,00
Blocked Lock Requests Ratio % Blocked Requests	0,00	0,00	—	0,00
Degree of Parallelism Threads	1,00	1,00	—	0,00
Log Flushes	100008,00	100001,00	∨	-7,00
DB Availability %	100,00	100,00	—	0,00

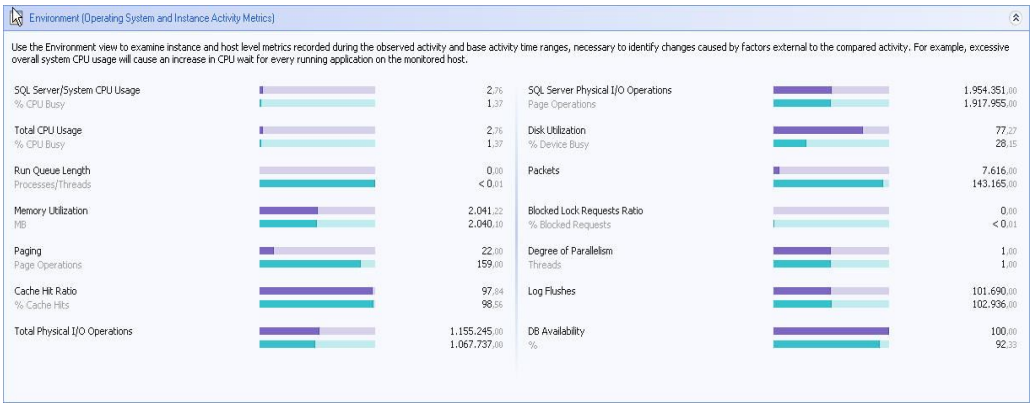


En la operación de modificación de registros, no se observa ningún valor que nos indique que la realización de la auditoria penalice negativamente el rendimiento del sistema. Consideramos que el incremento de un 14% en el tiempo de realización de la prueba no es significativo puesto que cualquier pequeña incidencia de espera de CPU, red ó disco en el cliente puede afectar a al tiempo de ejecución.

Prueba 4.

Global Environment Statistics Comparison

Statistic	Base	Observed	Change
SQL Server/System CPU Usage % CPU Busy	2,76	1,37	-1,39
Total CPU Usage % CPU Busy	2,76	1,37	-1,39
Run Queue Length Processes/Threads	0,00	0,00	0,00
Memory Utilization MB	2041,22	2040,10	-1,13
Paging Page Operations	22,00	159,00	137,00
Cache Hit Ratio % Cache Hits	97,84	98,56	0,72
Total Physical I/O Operations	1155245,00	1067737,00	-87508,00
SQL Server Physical I/O Operations Page Operations	1954351,00	1917955,00	-36396,00
Disk Utilization % Device Busy	77,27	28,15	-49,13
Packets	7616,00	143165,00	135549,00
Blocked Lock Requests Ratio % Blocked Requests	0,00	0,00	0,00
Degree of Parallelism Threads	1,00	1,00	0,00
Log Flushes	101690,00	102936,00	1246,00
DB Availability %	100,00	92,33	-7,67



Environment (Operating System and Instance Activity Metrics)

Use the Environment view to examine instance and host level metrics recorded during the observed activity and base activity time ranges, necessary to identify changes caused by factors external to the compared activity. For example, excessive overall system CPU usage will cause an increase in CPU wait for every running application on the monitored host.

SQL Server/System CPU Usage % CPU Busy	2,76 1,37		SQL Server Physical I/O Operations Page Operations	1.954.351,00 1.917.955,00	
Total CPU Usage % CPU Busy	2,76 1,37		Disk Utilization % Device Busy	77,27 28,15	
Run Queue Length Processes/Threads	0,00 < 0,01		Packets	7.616,00 143.165,00	
Memory Utilization MB	2.041,22 2.040,10		Blocked Lock Requests Ratio % Blocked Requests	0,00 < 0,01	
Paging Page Operations	22,00 159,00		Degree of Parallelism Threads	1,00 1,00	
Cache Hit Ratio % Cache Hits	97,84 98,56		Log Flushes	101.690,00 102.936,00	
Total Physical I/O Operations	1.155.245,00 1.067.737,00		DB Availability %	100,00 92,33	

La operación de borrado de afecta de forma negativa al rendimiento de sistema, primero porque ralentiza el proceso en 14%; después hay más cantidad de paquetes de red enviados, aunque este valor no es producido por la activación de la auditoría del modo “C2” sino por el tráfico generado por los procesos de actualización del *Performance Analysis.* Pero es el valor de paginación, como en la operación de inserción, el que se ve más afectado. Concluyendo que el modo de auditoría “C2” durante la operación de borrado afecta negativamente al rendimiento.

En cuanto al almacenamiento físico sabemos que los ficheros se almacenan en el directorio \MSSQL\Data, es decir en el mismo lugar donde está instalado el SQL Server y son archivos .trc de 200MB cada uno. Los archivos se van generando a medida que se van llenando, así que se arranca con un archivo, cuando llega a 200MB, se cierra y se abre uno nuevo. Corremos por tanto el peligro de llenar el disco y al almacenarse en la misma partición que la base de datos esta se podría bloquear.

Una vez analizadas las pruebas, estudiamos los ficheros de traza generados, observamos que contiene una cantidad grande de entradas repetidas lo que provoca que sea muy laborioso el filtrado y selección de aquellos eventos y columnas válidos para que un auditor pueda analizar y determinar los posibles problemas de una base de datos operativa. Por tanto podemos concluir que la activación del modo "C2" en un sistema que tenga un movimiento medio de datos va afectar de manera negativa al rendimiento y no se va a conseguir el objetivo de realizar un proceso de auditoría realmente bueno.

PRUEBA	**SIN AUDITORÍA**	**CON AUDITORÍA MODO "C2"**	**% VARIACIÓN DE TIEMPO**
1: inserción	1826	1865	2%
2: consulta	6171	8618	40%
3: modificación	97	111	14%
4: borrado	1399	1557	11%

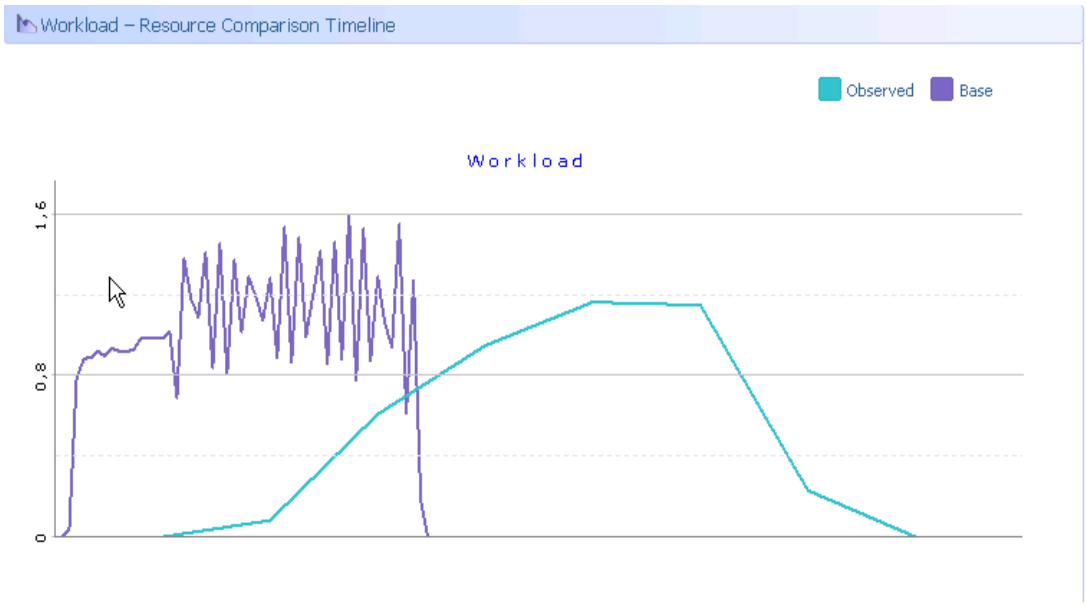
7.4 ***Rendimiento del S.G.D.B utilizando SQL Server Profiler***

Hemos intentado ver las diferencias de rendimiento entre realizar las pruebas sin auditoría y realizar las pruebas usando la herramienta *SQL Server Profiler*

Vamos a utilizar la plantilla *Standard* para la creación de la traza, cuya descripción se hizo en el punto 5.2. El fichero de traza se va a almacenar en el equipo SOTANO, de manera que minimizamos el trabajo en disco en el equipo TEJADO.

Mediante *Performance Analysis* generamos un informe comparativo de aquellos valores de rendimiento medidos durante el proceso de prueba. Tomamos como línea base de la comparación las pruebas realizadas sin tener activada la auditoría y la comparamos con las pruebas realizadas con el modo de auditoría *Profiler* activado.

Prueba 1.



Global Environment Statistics Comparison

Statistic	Base	Observed	Change	
SQL Server/System CPU Usage % CPU Busy	4,51	2,19	▼	-2,32
Total CPU Usage % CPU Busy	4,51	2,19	▼	-2,32
Run Queue Length Processes/Threads	0,05	0,03	▼	-0,02
Memory Utilization MB	1503,48	1294,54	▼	-208,94
Paging Page Operations	17,00	10682,00	▲	10665,00
Cache Hit Ratio % Cache Hits	99,81	99,44	▼	-0,37
Total Physical I/O Operations	542556,00	577214,00	▲	34658,00
SQL Server Physical I/O Operations Page Operations	2326814,00	2317297,00	▼	-9517,00
Disk Utilization % Device Busy	81,02	30,50	▼	-50,52
Packets	5221668,00	5423841,00	▲	202173,00
Blocked Lock Requests Ratio % Blocked Requests	0,00	0,00	—	0,00
Degree of Parallelism Threads	1,00	1,00	—	0,00
Log Flushes	121406,00	121229,00	▼	-177,00
DB Availability %	100,00	100,00	—	0,00

Environment (Operating System and Instance Activity Metrics)

Use the Environment view to examine instance and host level metrics recorded during the observed activity and base activity time ranges, necessary to identify changes caused by factors external to the compared activity. For example, excessive overall system CPU usage will cause an increase in CPU wait for every running application on the monitored host.

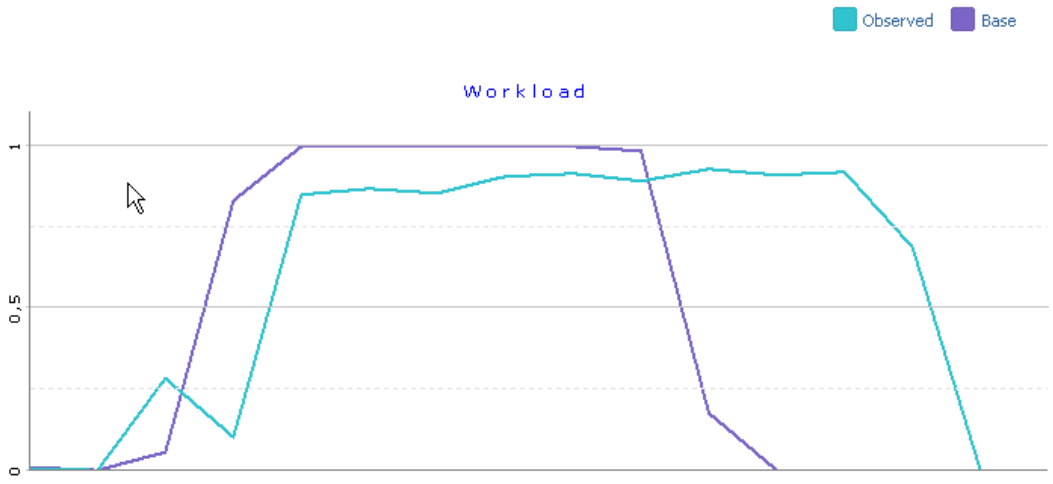
SQL Server/System CPU Usage % CPU Busy	4,51 2,19		SQL Server Physical I/O Operations Page Operations	2.326.814,00 2.317.297,00	
Total CPU Usage % CPU Busy	4,51 2,19		Disk Utilization % Device Busy	81,02 30,50	
Run Queue Length Processes/Threads	0,05 0,03		Packets	5.221.668,00 5.423.841,00	
Memory Utilization MB	1.503,48 1.294,54		Blocked Lock Requests Ratio % Blocked Requests	< 0,01 < 0,01	
Paging Page Operations	17,00 10.682,00		Degree of Parallelism Threads	1,00 1,00	
Cache Hit Ratio % Cache Hits	99,81 99,44		Log Flushes	121.406,00 121.229,00	
Total Physical I/O Operations	542.556,00 577.214,00		DB Availability %	100,00 100,00	

El parámetro más significativo en esta prueba es el aumento del tiempo de duración, que sube en un 28%. Esta información unida a los datos de paginación que también aumenta considerablemente nos permite deducir que la realización de la auditoría repercute de manera notable en el rendimiento de SOTANO cuando se realiza la operación de inserción. Curiosamente el dato de utilización

de disco disminuye, dado que el fichero de traza no se almacena sobre la unidad de almacenamiento de SOTANO debería no haber diferencias entre el resultado obtenido al hacer auditoría y el resultado de no realizarla; nos ayudamos con gráfica de carga de trabajo para observar que la línea de actividad tiene más picos en el caso de la prueba realizada sin auditoría, por tanto podemos deducir que algún proceso no provocado por la operación de inserción se ejecutó en SOTANO.

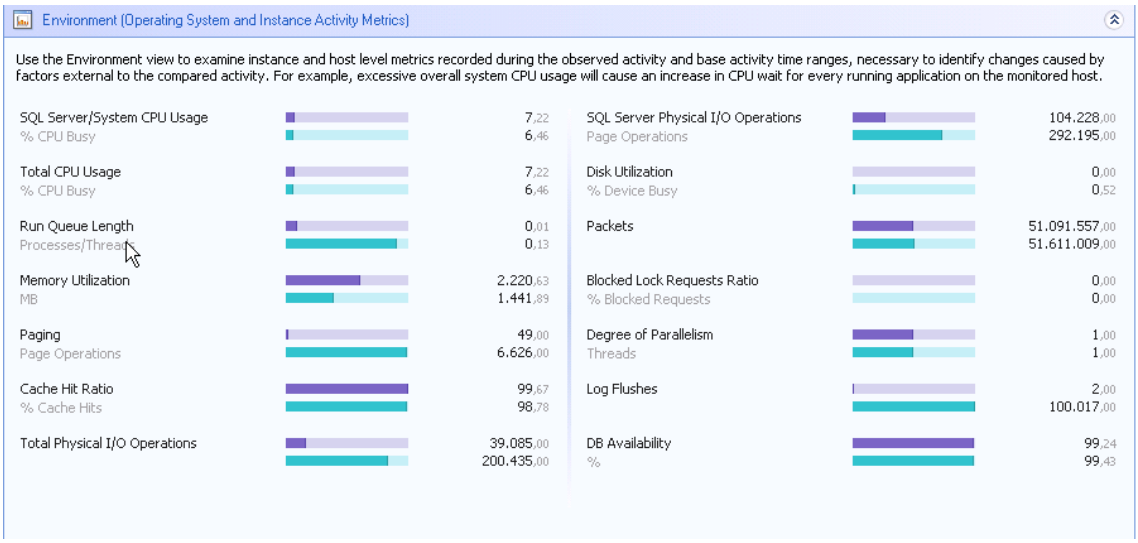
Prueba 2.

Workload – Resource Comparison Timeline



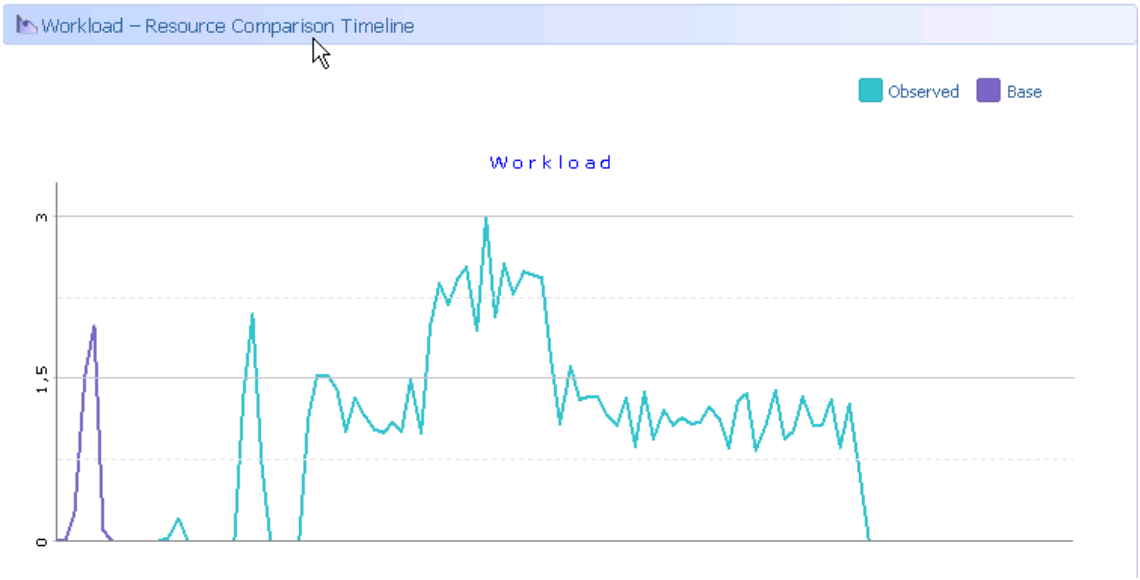
Global Environment Statistics Comparison

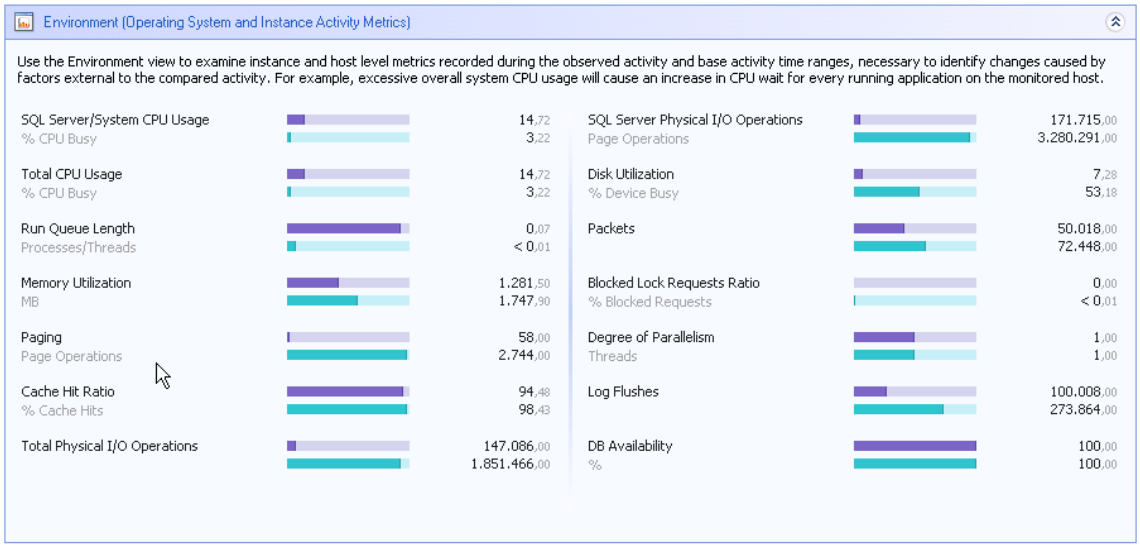
Statisti	Base	Observed	Change	
SQL Server/System CPU Usage % CPU Busy	7,22	6,46	▼	-0,76
Total CPU Usage % CPU Busy	7,22	6,46	▼	-0,76
Run Queue Length Processes/Threads	0,01	0,13	▲	0,12
Memory Utilization MB	2220,63	1441,89	▼	-778,74
Paging Page Operations	49,00	6626,00	▲	6577,00
Cache Hit Ratio % Cache Hits	99,67	98,78	▼	-0,89
Total Physical I/O Operations	39085,00	200435,00	▲	161350,00
SQL Server Physical I/O Operations Page Operations	104228,00	292195,00	▲	187967,00
Disk Utilization % Device Busy	0,00	0,52	▲	0,52
Packets	51091557,00	51611009,00	▲	519452,00
Blocked Lock Requests Ratio % Blocked Requests	0,00	0,00	—	0,00
Degree of Parallelism Threads	1,00	1,00	—	0,00
Log Flushes	2,00	100017,00	▲	100015,00
DB Availability %	99,24	99,43	▲	0,19

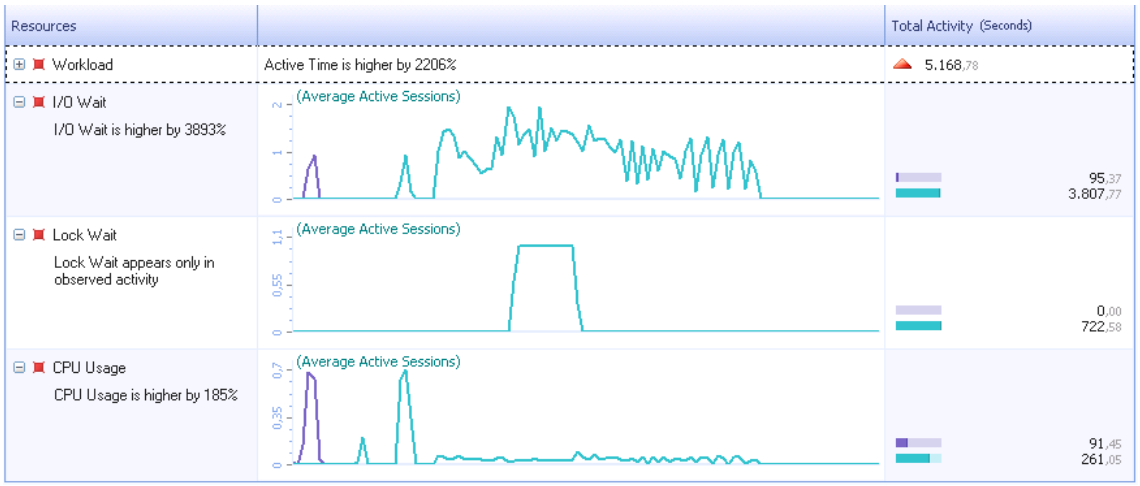


La visión de la comparación de los valores resultado de las pruebas realizadas con la operación de consulta nos hacen concluir que hay una gran penalización del rendimiento en el equipo SOTANO, dado que el tiempo de realización aumenta en un 26%; el número de operaciones físicas de E/S aumenta más de un 100%; la actividad del log de transacciones se dispara, este contador nos muestra el numero de accesos al fichero de transacciones por segundo, esto es debido a que el número de operaciones físicas de E/S del SQL Server durante la operación de consulta aumenta en más de 100.000 operaciones de página. Así pues cada uno de los contadores de rendimiento que presentan un resultado diferenciador en las pruebas realizadas, tienen interrelación y debemos tomarlos de manera conjunta para que nos sirve de apoyo en nuestras conclusiones.

Prueba 3.

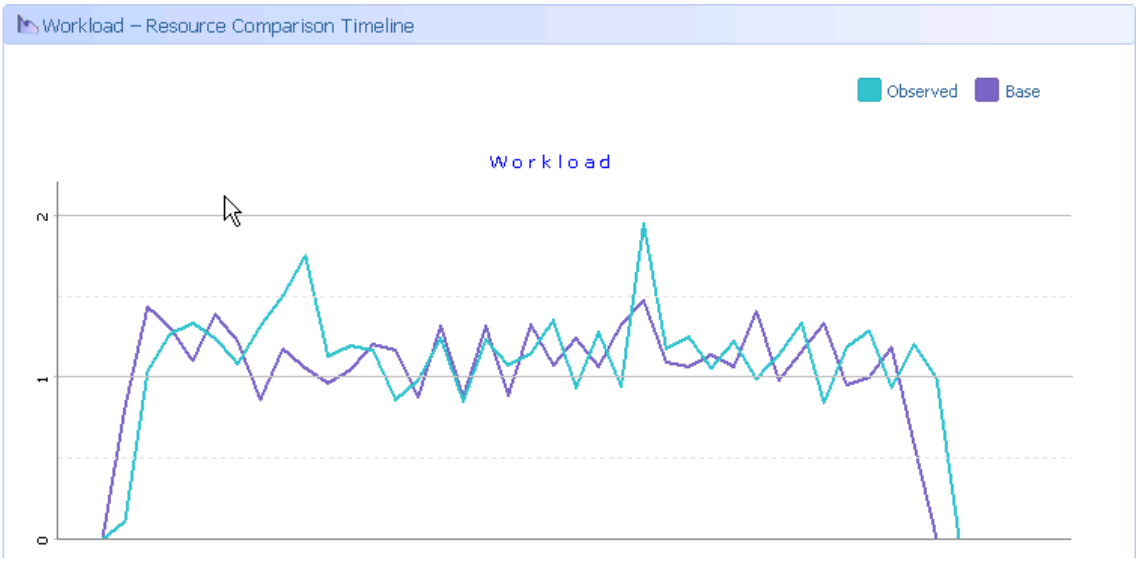

Workload – Resource Comparison Timeline
Observed
Base
Workload
3
1,5
0

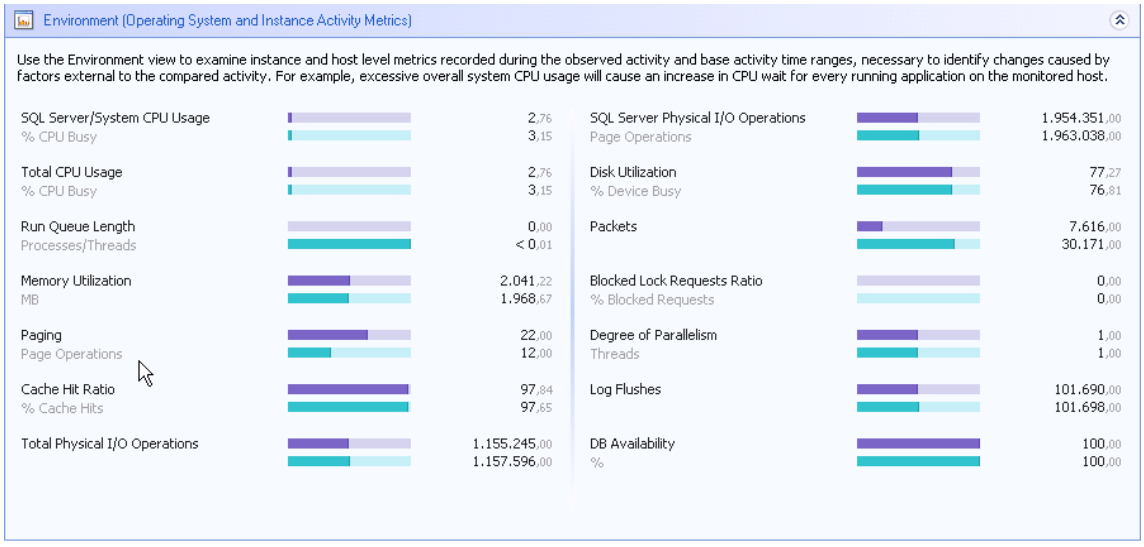

Environment (Operating System and Instance Activity Metrics)
Use the Environment view to examine instance and host level metrics recorded during the observed activity and base activity time ranges, necessary to identify changes caused by factors external to the compared activity. For example, excessive overall system CPU usage will cause an increase in CPU wait for every running application on the monitored host.
SQL Server/System CPU Usage
% CPU Busy
14,72
3,22
Total CPU Usage
% CPU Busy
14,72
3,22
Run Queue Length
Processes/Threads
0,07
< 0,01
Memory Utilization
MB
1.281,50
1.747,90
Paging
Page Operations
58,00
2.744,00
Cache Hit Ratio
% Cache Hits
94,48
98,43
Total Physical I/O Operations
147.086,00
1.851.466,00
SQL Server Physical I/O Operations
Page Operations
171.715,00
3.280.291,00
Disk Utilization
% Device Busy
7,28
53,18
Packets
50.018,00
72.448,00
Blocked Lock Requests Ratio
% Blocked Requests
0,00
< 0,01
Degree of Parallelism
Threads
1,00
1,00
Log Flushes
100.008,00
273.864,00
DB Availability
%
100,00
100,00


Resources
Total Activity (Seconds)
Workload
Active Time is higher by 2206%
5.168,78
I/O Wait
I/O Wait is higher by 3893%
(Average Active Sessions)
95,37
3.807,77
Lock Wait
Lock Wait appears only in observed activity
(Average Active Sessions)
0,00
722,58
CPU Usage
CPU Usage is higher by 185%
(Average Active Sessions)
91,45
261,05

Cuando hacemos modificaciones en los registros de nuestra tabla de datos, notamos que mejoramos en la duración de la prueba bajando a un 13%, porcentaje mejor que en la inserción y consulta. Pero en los contadores que se refieren al número de operaciones físicas de E/S totales y de SQL Server se mantiene un nivel alto de diferencia con la prueba sin auditoría. Además la paginación y la actividad en el registro de transacciones mantienen una importante carga, producto de penalización de rendimiento que supone monitorizar la actividad del servidor de bases de datos.

Prueba 4.





Global Environment Statistics Comparison

Statistic	Base	Observed	Change
SQL Server/System CPU Usage % CPU Busy	2,76	3,15	0,39
Total CPU Usage % CPU Busy	2,76	3,15	0,39
Run Queue Length Processes/Threads	0,00	0,00	0,00
Memory Utilization MB	2041,22	1968,67	-72,56
Paging Page Operations	22,00	12,00	-10,00
Cache Hit Ratio % Cache Hits	97,84	97,65	-0,19
Total Physical I/O Operations	1155245,00	1157596,00	2351,00
SQL Server Physical I/O Operations Page Operations	1954351,00	1963038,00	8687,00
Disk Utilization % Device Busy	77,27	76,81	-0,47
Packets	7616,00	30171,00	22555,00
Blocked Lock Requests Ratio % Blocked Requests	0,00	0,00	0,00
Degree of Parallelism Threads	1,00	1,00	0,00
Log Flushes	101690,00	101698,00	8,00
DB Availability %	100,00	100,00	0,00

En el borrado la diferencia de tiempo de duración de las pruebas es mínima, para el resto de los contadores de rendimiento, no se aprecian diferencias notables. Lo único que podemos reseñar es el aumento de paquetes transmitidos por la red, motivado por transmisión de información hacia el fichero de traza que se almacena en el equipo SOTANO.

El fichero de traza se mantiene en un tamaño adecuado ya que lo hemos limitado a 100 MBytes y además no se encuentra ubicado en el mismo disco que el sistema gestor de la base de datos; estos dos hechos ya por si sólo tiene una incidencia positiva en el rendimiento del equipo TEJADO comparado con los resultados de la auditoría modo “C2”. Por otra parte, la definición mediante plantilla de los eventos que se capturan ayuda a que la recolección sea menor y el posterior análisis de la traza sea más fácil. De hecho el uso de la herramienta

SQL Server Profiler de una manera adecuada tomando sólo aquellos eventos y propiedades significativos es el modo de auditoría que mejor rendimiento ofrece.

PRUEBA	**SIN AUDITORÍA**	**CON AUDITORÍA PROFILER**	**% VARIACIÓN DE TIEMPO**
1: inserción	1826	2344	28%
2: consulta	6171	7769	26%
3: modificación	97	110	13%
4: borrado	1399	1457	4%

8 Service Broker

Una parte de este trabajo se compone en aportar algún aspecto novedoso; queremos conseguir que la tecnología *Service Broker* integrada *SQL Server 2005* sea una herramienta útil para la auditoría de Bases de datos, es decir tenemos que conseguir mejorar la monitorización de una base de datos aprovechando capacidad para almacenar mensajes en colas e intercambiar estos mensajes mediante comunicación asíncrona asegurando la información.

Es una nueva tecnología que forma parte de *SQL Server 2005* que ofrece a los programadores de bases de datos la posibilidad de generar aplicaciones seguras, confiable y ampliables. *Service Broker* puede utilizarse tanto en las aplicaciones que usan una única instancia de *SQL Server* como en aquellas que usan varias instancias. En una única instancia de *SQL Server, Service Broker* proporciona un modelo sólido de programación asincrónica. Las aplicaciones de base de datos suelen utilizar programación asincrónica para reducir el tiempo de respuesta interactivo y aumentar el rendimiento general de la aplicación. *Service Broker* también proporciona mensajería confiable entre instancias de *SQL Server*.

Service Broker permite componer aplicaciones a partir de componentes independientes, denominados servicios. Las aplicaciones que necesitan la funcionalidad expuesta en estos servicios utilizan mensajes para interactuar con los servicios. Utiliza *TCP/IP* para intercambiar mensajes entre instancias, además incluye características para ayudar a evitar el acceso no autorizado desde la red y a cifrar los mensajes enviados por ésta. En los siguientes puntos vamos a detallar las características y los detalles de funcionamiento de *Service Broker*, para continuar con la realización de unas pruebas basadas en el uso de combinado de la auditoría de disparadores DML y del *Service Broker*. Y finalizaremos con las *Notificaciones de Eventos,* que son un tipo especial de objeto de base de datos que envía información sobre eventos de servidor y base de datos a *Service Broker* y constituiría otra forma de aplicar la tecnología del *Service Broker*.

Resumiendo, entre los beneficios que nos ofrece el *Service Broker* tenemos ([H23]):

- La integración de bases de datos.
- Ordenación y coordinación de mensajes.
- El acoplamiento flexible de las aplicaciones.
- El bloqueo de mensajes relacionados.
- La activación automática.

Y entre las aplicaciones de uso del *Service Broker* podemos tener:

- Desencadenadores asincrónicos
- Procesamiento confiable de consultas
- Recopilación confiable de datos
- Procesamiento distribuido en el servidor para aplicaciones cliente
- Consolidación de datos para aplicaciones cliente
- Procesamiento por lotes a gran escala

8.1 *Fundamentos*

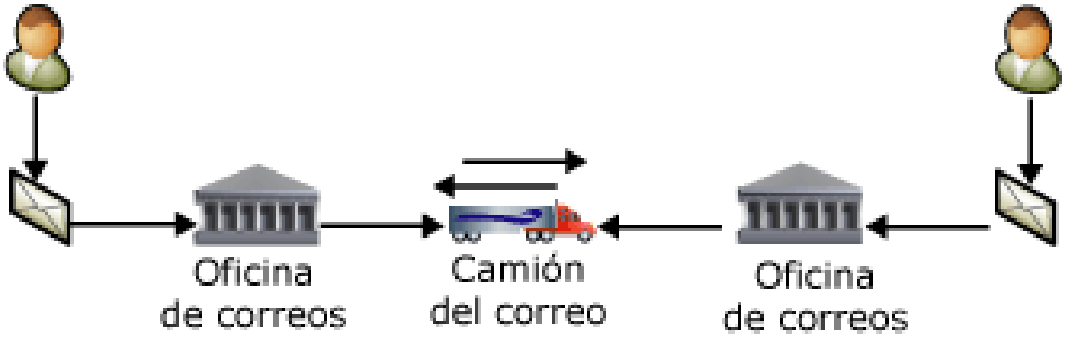
Detallaremos aspectos fundamentales de *Service Broker* para obtener una mejor visión de esta tecnología ([H24]).

Inicialmente explicaremos el concepto de conversación. *Service Broker* está diseñado según las funciones básicas de envío y recepción de mensajes. Cada mensaje forma parte de una conversación, que constituye un canal de comunicación confiable y persistente, cada mensaje y conversación tienen un

tipo específico que impone *Service Broker*. Tenemos entonces que una aplicación envía mensajes a un servicio, que es un nombre para un conjunto de tareas relacionadas; además la aplicación recibe mensajes de una cola, que es una vista de una tabla interna. Hay que tener en cuenta que los mensajes para la misma tarea forman parte de la misma conversación. En cada conversación se garantiza que una aplicación reciba cada mensaje exactamente una vez en el orden en el que se envió. El programa que implementa un servicio puede asociar conversaciones relacionadas para el mismo servicio en un grupo de conversación. Respecto a la seguridad, decimos que está basada en certificados, ofreciendo protección a los mensajes confidenciales y control sobre el acceso a los servicios.

Una forma de entender *Service Broker* consiste en compararlo con el servicio postal. Para mantener una conversación con un amigo que está lejos, puede comunicarse mediante cartas enviadas a través del servicio postal, que ordena y entrega las cartas. Ambos retiran las cartas de sus buzones, las leen, escriben respuestas y envían nuevas cartas, hasta que la conversación finaliza. La entrega de cartas sucede de forma "asincrónica", mientras los dos llevan a cabo otras tareas. En la analogía del servicio postal, las cartas son los mensajes. El servicio de *Service Broker* es la dirección a la que la oficina postal entrega las cartas. Las colas son los buzones que guardan las cartas una vez entregadas. Las aplicaciones reciben los mensajes, actúan sobre los mismos y envían respuestas.

Un programa que utiliza *Service Broker* mantiene conversaciones con otros programas de forma similar a la entrega postal. No hay que saber específicamente cuándo lee nuestro amigo el correo o escribe la respuesta. De igual forma, una aplicación que utiliza *Service Broker* no tiene que conocer cómo



otro servicio procesa el mensaje, cómo se entrega o cuándo la otra aplicación va a procesar el mensaje.

La segunda característica original es la programación asincrónica transaccional; en la infraestructura de *Service Broker*, la entrega de mensajes entre aplicaciones es transaccional y asincrónica. Como la mensajería de *Service Broker* es transaccional, si una transacción se revierte, todas las operaciones de *Service Broker* de la transacción también lo harán, incluidas las operaciones de envío y recepción. En una entrega asincrónica, el Motor de base de datos controla la entrega mientras la aplicación continúa ejecutándose. Para mejorar la escalabilidad, *Service Broker* proporciona mecanismos para iniciar automáticamente programas que procesan una cola cuando existen tareas pendientes. Así que podemos escribir aplicaciones que usan colas, estas permiten que la base de datos continúe dando respuesta a los usuarios interactivos a la vez que utilizan los recursos disponibles de forma eficaz.

Las colas permiten que una aplicación realice un trabajo en una transacción distinta de la transacción que solicita el trabajo. *Service Broker* amplía esta idea, de forma que las aplicaciones puedan llevar a cabo el trabajo en una instancia o en equipos diferentes. *Service Broker* es un gran recurso para el trabajo con bases de datos, ya que proporciona colas integradas en la base de datos y una mensajería transaccional confiable entre instancias.

Otra propiedad interesante es la compatibilidad con aplicaciones de acoplamiento flexible, estas aplicaciones están formadas por varios programas que envían y reciben mensajes de forma independiente. Tales aplicaciones deben contener las mismas definiciones para los mensajes intercambiados y definir la misma estructura global para la interacción entre los servicios. Sin embargo, las aplicaciones no tienen que ejecutarse al mismo tiempo, ni ejecutarse en la misma instancia de *SQL Server*, ni compartir los detalles de la implementación. Una aplicación no necesita conocer la ubicación física ni la implementación del otro participante de la conversación.

Por último, hacemos referencia a los tres tipos de componentes de *Service Broker*:

- **Componentes de conversación.** Los grupos de conversación, las conversaciones y los mensajes conforman la estructura de tiempo de ejecución de una aplicación de *Service Broker*. Las aplicaciones intercambian mensajes como parte de una conversación. Cada conversación forma parte de un grupo de conversación; un grupo de conversación puede contener varias conversaciones. Las conversaciones de *Service Broker* son diálogos, es decir, conversaciones entre dos participantes.

- **Componentes de definición de servicio.** Éstos son los componentes de tiempo de diseño que especifican la estructura básica de las conversaciones que utiliza la aplicación. Definen los tipos de mensajes, el flujo de conversación y el almacenamiento de base de datos de la aplicación.

- **Componentes de red y de seguridad.** Estos componentes definen la infraestructura para intercambiar mensajes fuera de una instancia de *Service Broker*. Para ayudar a los administradores de base de datos a administrar entornos cambiantes, *Service Broker* les permite configurar estos componentes independientemente del código de la aplicación.

Los componentes de definición de servicio, los componentes de red y los componentes de seguridad forman parte de los metadatos de la base de datos y de la instancia de *Service Broker*.

Los grupos de conversación, las conversaciones y los mensajes forman parte de los datos que contiene la base de datos.

8.2 *Arquitectura de Service Broker*

Las aplicaciones de *Service Broker* están compuestas por objetos de base de datos de *Service Broker* y una o más aplicaciones que utilizan dichos objetos. Describiremos cada uno de los objetos utilizados en una aplicación de *Service Broker*. Existen tres tipos de componentes ([H25]).

- Componentes de conversación. Definen la estructura de tiempo de ejecución de la conversación. Como ya hemos señalado las aplicaciones intercambian mensajes como parte de una conversación, de ahí su importancia.

- Objetos de definición de servicio. Son los componentes de tiempo de diseño que especifican el diseño básico de la aplicación. Estos componentes definen los tipos de mensajes, el flujo de conversación y el almacenamiento de base de datos de la aplicación.

- Componentes de enrutamiento y seguridad. Estos componentes definen la infraestructura para intercambiar mensajes fuera de una instancia de *SQL Server*.

8.3 *Arquitectura de la conversación*

Todas las aplicaciones que usan *Service Broker* se comunican a través de conversaciones, ya hemos dicho que son intercambios asincrónicos, confiables y de larga duración de mensajes. *Service Broker* utiliza los objetos siguientes para las conversaciones.

Mensajes

Los mensajes son la información intercambiada entre aplicaciones que utilizan *Service Broker*.

Cada mensaje forma parte de una conversación. Un mensaje tiene un tipo específico, que viene determinado por la aplicación que envía el mensaje. Cada

mensaje tiene una identidad de conversación única, así como un número de secuencia en la conversación. Cuando recibe mensajes, *Service Broker* utiliza la identidad de conversación y el número de secuencia del mensaje para ordenar los mensajes.

El contenido del mensaje viene determinado por la aplicación. Cuando se recibe un mensaje, *Service Broker* valida el contenido del mensaje para garantizar que es válido para el tipo de mensaje. Independientemente del tipo de mensaje, *SQL Server* almacena el contenido del mensaje como de tipo *varbinary(max)*. Por tanto, un mensaje puede contener cualquier dato que se pueda convertir al tipo *varbinary(*max).

Normalmente, una aplicación procesa el contenido de un mensaje basándose en el contrato y en el tipo de mensaje.

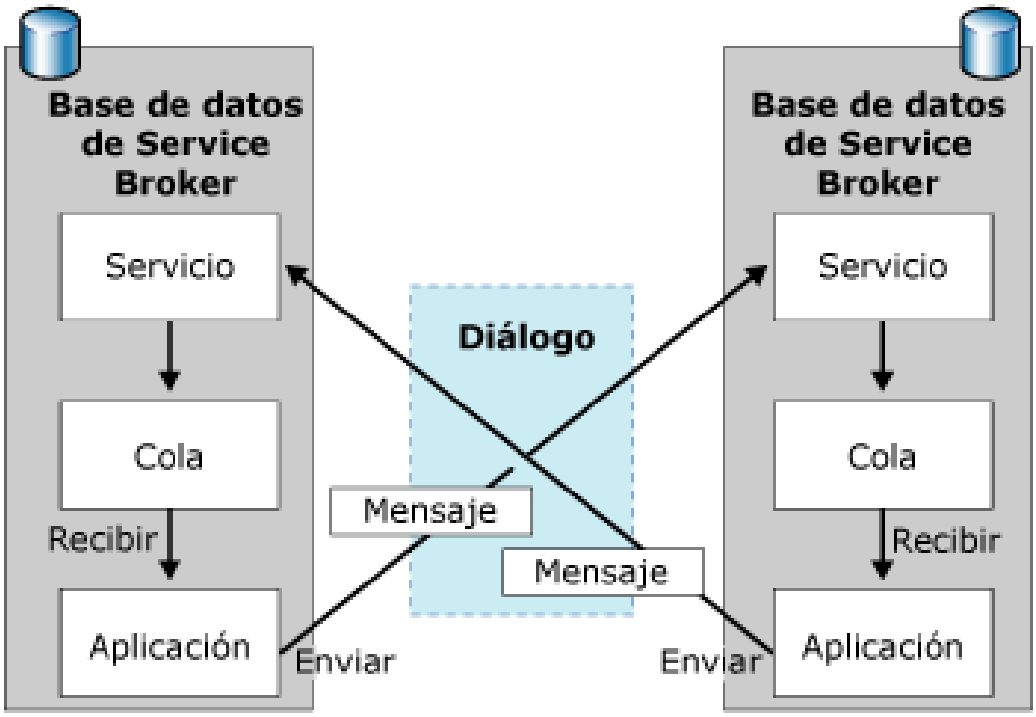
Conversaciones de diálogo

Todos los mensajes enviados por *Service Broker* forman parte de una conversación. Un diálogo es una conversación entre dos servicios. Es una secuencia bidireccional, persistente y confiable de mensajes entre dos servicios.

Los diálogos proporcionan entrega de mensajes EOIO (exactamente una vez por orden). Los diálogos utilizan el identificador de conversación y los números de secuencia incluidos en cada mensaje para identificar mensajes relacionados y entregar los mensajes en el orden correcto. Un diálogo es una secuencia persistente y confiable de mensajes entre dos servicios.

La conversación de diálogo está formada por dos participantes. El iniciador inicia la conversación. El destino acepta una conversación comenzada por el iniciador. Si un participante inicia la conversación, determina los mensajes que puede enviar, como se especifica en el contrato de la conversación.

El siguiente diagrama muestra el flujo de mensajes de un diálogo.



Las aplicaciones intercambian mensajes como parte del diálogo. Cuando *SQL Server* recibe un mensaje para un diálogo, entonces coloca el mensaje en la cola de dicho diálogo. La aplicación recibe el mensaje desde la cola y lo procesa según sea necesario. Como parte del proceso, es posible que la aplicación envíe mensajes al otro participante del diálogo.

Los diálogos incorporan reconocimientos automáticos de recepción de mensajes para garantizar una entrega confiable.

Se pueden intercambiar mensajes entre las aplicaciones mientras no caduca la duración del diálogo. La duración del diálogo se extiende desde el momento en que la instancia de *SQL Server* local crea el diálogo hasta que una aplicación lo finaliza explícitamente o recibe un mensaje de error asociado al diálogo. Cada participante es responsable de finalizar explícitamente la conversación cuando la aplicación recibe un mensaje que indica un error o el final de la conversación. En la mayoría de los servicios, un participante es

responsable de indicar que la conversación ha finalizado y es correcta finalizando la conversación sin errores. El propósito de la conversación es el que decide si el responsable es el destino o el iniciador.

El temporizador de conversación permite a una aplicación recibir un mensaje a una hora específica. Cuando el temporizador de conversación caduca, *SQL Server* inserta un mensaje para la conversación en la cola de la conversación, en el extremo que inició el temporizador de conversación. Una aplicación puede utilizar un temporizador de conversación con cualquier fin. Un uso común del temporizador de conversación consiste en responder a los retrasos a las respuestas desde el servicio remoto. Otro uso común es crear un servicio que envíe mensajes al servicio remoto en intervalos establecidos. Por ejemplo, un servicio puede utilizar un temporizador de conversación para informar del estado actual de *SQL Server* cada pocos minutos. Las aplicaciones también pueden utilizar un temporizador de conversación para activar un procedimiento almacenado en un momento determinado. Esto permite que *Service Broker* admita actividades programadas.

Cada participante de una conversación puede establecer un temporizador de conversación por conversación. El temporizador de conversación no se comparte con el otro participante y no afecta a la duración de la conversación. En su lugar, cuando el temporizador caduca, *Service Broker* local agrega un mensaje de tiempo de espera a la cola del servicio local.

Los mensajes de tiempo de espera tienen el tipo http://schemas.microsoft.com/SQL/ServiceBroker/DialogTimer.

Grupos de conversación

Los grupos de conversación identifican las conversaciones que trabajan conjuntamente para realizar la misma tarea. *Service Broker* utiliza los grupos de conversación para administrar el bloqueo de mensajes, lo que permite a los desarrolladores de software administrar la simultaneidad. Los programadores de

aplicaciones también utilizan grupos de conversación como ayuda en la administración del estado.

8.4 *Arquitectura de servicio*

En esta sección se describen los objetos de base de datos que especifican el diseño básico de una aplicación que utiliza *Service Broker*.

En tiempo de diseño, las aplicaciones de *Service Broker* especifican los siguientes objetos:

- ***Tipos de mensajes**:* Definen los nombres de los mensajes que se intercambian entre aplicaciones. Opcionalmente, proporcionan validación para los mensajes.

- ***Contratos***: Especifican la dirección y el tipo de mensajes de una conversación dada.

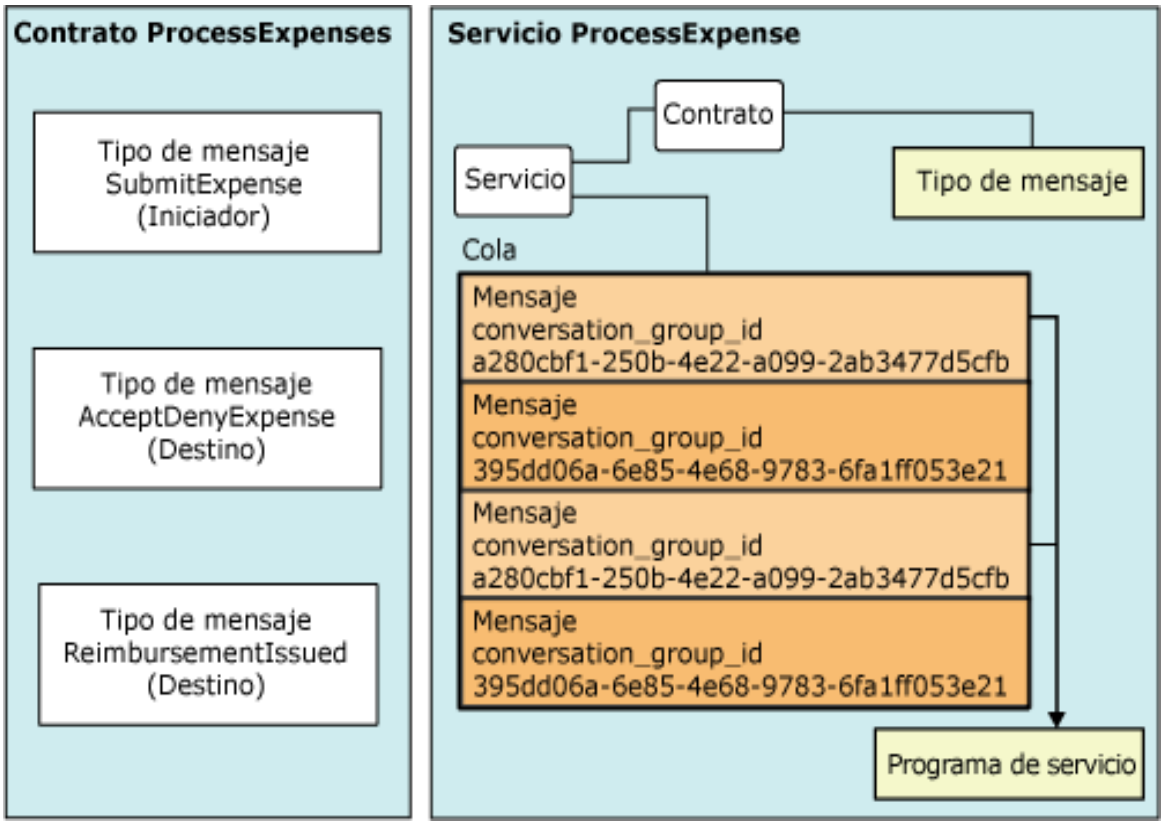
- ***Colas***: Almacenan mensajes. Este mecanismo de almacenamiento permite la comunicación asincrónica entre servicios. Las colas de *Service Broker* proporcionan otras ventajas, como el bloqueo automático de mensajes del mismo grupo de conversación.

- ***Servicios***: Son extremos con direcciones para las conversaciones. Los mensajes de *Service Broker* se envían desde un servicio a otro. Un servicio especifica una cola para retener mensajes y los contratos en los que el servicio puede ser el destino. Un contrato proporciona un servicio con un conjunto bien definido de tipos de mensajes.

Una aplicación de *Service Broker* usa los objetos de *SQL Server* de la lista anterior para dirigir una conversación. Cualquier programa que pueda ejecutar instrucciones *Transact-SQL* en *SQL Server* puede utilizar *Service Broker*. Las aplicaciones pueden ser procedimientos almacenados escritos en *Transact-SQL* o en un lenguaje compatible con *CLR* (Common Language

Runtime, es el lenguaje de programación de .NET Framework) o bien programas externos que se conectan con una instancia de *SQL Server*.

En el siguiente diagrama se muestra un servicio de *Service Broker* ([H26]).



Como se muestra en la ilustración, el contrato ***ProcessExpenses*** especifica tres tipos de mensajes: ***SubmitExpense***, ***AcceptDenyExpense*** *y* ***ReimbursementIssued***. El contrato enumera los tipos de mensajes necesarios para una conversación que lleva a cabo una tarea de reembolso de gastos. El contrato ***ProcessExpenses*** controla todas las conversaciones entre el servicio ***ProcessExpense*** y cualquier servicio que inicia una conversación con ***ProcessExpense***. El servicio ***ProcessExpense*** almacena mensajes entrantes y salientes en la cola ***ExpenseQueue***. El procedimiento almacenado ***ExpenseProcessing*** recibe mensajes de esta cola, los procesa y los envía de nuevo a la cola para enrutarlos al broker adecuado si se necesita respuesta.

8.4.1 **Tipos de mensajes**

Las aplicaciones que utilizan *Service Broker* se comunican mediante el envío de mensajes entre ellas como parte de una conversación. Los participantes de una conversación deben estar de acuerdo en el nombre y el contenido de cada mensaje. Un objeto de tipo de mensaje define un nombre para un tipo de mensaje y el tipo de datos que contiene el mensaje. Los tipos de mensajes permanecen en la base de datos donde se crean. Debe crear un tipo de mensaje idéntico en cada base de datos que participa en una conversación.

Cada tipo de mensaje especifica la validación que *SQL Server* realiza en los mensajes de ese tipo. *SQL Server* puede validar que el mensaje contenga XML válido, que contenga XML conforme a un esquema determinado o que no contenga ningún dato. En datos arbitrarios o binarios, el tipo de mensaje puede especificar que *SQL Server* no valide el contenido del mensaje.

La validación se lleva a cabo cuando el servicio de destino recibe el mensaje. Si el contenido del mensaje no coincide con la validación especificada, *Service Broker* devuelve un mensaje de error al servicio que envía el mensaje.

Es importante que independientemente de la validación especificada, una aplicación deba comprobar que el contenido de un mensaje sea adecuado para la aplicación antes de que el programa utilice los datos.

En un tipo de mensaje vacío, el cuerpo del mensaje no debe contener datos. En un tipo de mensaje que especifica XML correcto, el cuerpo del mensaje debe ser XML correcto. En un tipo de mensaje que especifica XML conforme a una colección de esquemas determinada, el cuerpo del mensaje debe contener XML correcto válido para uno de los esquemas de la colección. En un tipo de mensaje que no especifique ninguna validación, *SQL Server* acepta cualquier contenido de mensaje, incluidos datos binarios, XML o mensajes vacíos.

Service Broker ofrece un tipo de mensaje integrado denominado DEFAULT. Si el tipo de mensaje no se especifica en un comando SEND de *Service Broker* el sistema utilizará el tipo de mensaje DEFAULT.

8.4.2 Contratos

Un contrato define los tipos de mensajes que utiliza una aplicación para realizar una tarea determinada. Un contrato es un acuerdo entre dos servicios sobre el cual los mensajes de cada servicio realizan una tarea determinada. Las definiciones de contrato se almacenan en la base de datos donde se crea el tipo.

Debe crear un contrato idéntico en cada base de datos que participa en una conversación. Por ejemplo, si una aplicación de recursos humanos desea comprobar un Id. de empleado, el servicio que solicita la comprobación debe saber los tipos de mensajes que espera el otro servicio. El servicio que realiza la solicitud también debe saber los tipos de mensajes que puede recibir y estar preparado para procesarlos.

El contrato especifica los tipos de mensajes que se pueden utilizar para realizar la tarea deseada. El contrato también especifica qué participante de la conversación puede utilizar cada tipo de mensaje. Algunos tipos de mensajes pueden enviarse por cualquier participante; otros tipos de mensajes están restringidos y sólo los pueden enviar el iniciador o el destino. Un contrato debe contener un tipo de mensaje enviado por el iniciador o un tipo de mensaje enviado por algún participante; de lo contrario, el iniciador no podrá iniciar una conversación que utilice el contrato.

Service Broker también incluye un contrato integrado denominado DEFAULT. El contrato DEFAULT sólo contiene el tipo de mensaje **SENT BY ANY**. Si no se especifica ningún contrato en la instrucción BEGIN DIALOG, *Service Broker* utiliza el contrato DEFAULT.

Por ejemplo, un contrato puede tener tipos de mensajes ***SubmitRequest***, ***ProcessRequest*** y ***RequestStatus***. Sólo el extremo iniciador puede utilizar

SubmitRequest y sólo el extremo de destino puede enviar ***ProcessRequest.*** Cualquiera de los participantes en la conversación puede enviar el tipo de mensaje ***RequestStatus***. El tipo de mensaje ***RequestStatus*** permite al participante ver en qué parte del procesamiento se encuentra el destino o comprobar con el iniciador el estado de cualquier procesamiento en paralelo relacionado con esta solicitud.

8.4.3 Colas

Las colas almacenan mensajes. Cuando *Service Broker* recibe un mensaje para un servicio, lo inserta en la cola para ese servicio. Para obtener los mensajes enviados al servicio, una aplicación recibe mensajes de la cola. *Service Broker* administra las colas y presenta una vista de una cola similar a una tabla.

Cada servicio se asocia con una cola. Cuando llega un mensaje para un servicio, *Service Broker* coloca el mensaje en la cola asociada a este servicio.

Cada mensaje es una fila de la cola. La fila incluye el contenido del mensaje así como información sobre el tipo de mensaje, el servicio al que se destina el mensaje, el contrato que sigue el mensaje, la validación realizada en el mensaje, la conversación de la que forma parte el mensaje e información interna de la cola. Una aplicación utiliza la información de la fila de mensaje para identificar cada mensaje de forma exclusiva y procesarlo correctamente.

Las aplicaciones reciben mensajes de la cola para el servicio. En cada conversación, las colas devuelven los mensajes en el orden en que el remitente los ha enviado. Todos los mensajes devueltos desde una operación de recepción única forman parte de una conversación que pertenece a un grupo de conversación. De hecho, una cola contiene conjuntos de mensajes relacionados, un conjunto por cada grupo de conversación. La cola devuelve un conjunto de mensajes relacionados cada vez que la aplicación lleva a cabo una operación de recepción desde la cola. La aplicación puede elegir recibir los mensajes de una conversación específica o de un grupo de conversación específico. Las colas no

devuelven mensajes en orden FIFO estricto (*First In, First Out;* primero en llegar, primero en salir); en su lugar, las colas devuelven los mensajes de cada conversación en el orden en el que se enviaron. Así, una aplicación no necesita incluir código para recuperar el orden original de los mensajes.

Una cola puede estar asociada a un procedimiento almacenado. En este caso, *SQL Server* activa el procedimiento almacenado cuando hay mensajes para procesar en la cola. *SQL Server* puede iniciar más de una instancia de procedimiento almacenado hasta llegar a un máximo configurado.

8.4.4 Servicios

Un servicio de *Service Broker* es un nombre para una tarea empresarial o un conjunto de tareas empresariales específico. Las conversaciones se producen entre servicios. *Service Broker* utiliza el nombre del servicio para entregar mensajes a la cola correcta de una base de datos, para enrutar mensajes, para imponer el contrato de una conversación y para determinar la seguridad remota de una conversación nueva.

Cada servicio especifica una cola para contener los mensajes entrantes. Los contratos asociados con el servicio definen las tareas específicas para las que el servicio acepta nuevas conversaciones. Por tanto, un servicio de destino especifica uno o más contratos que deben seguir las conversaciones con el servicio. Un servicio que inicia conversaciones pero no recibe conversaciones nuevas de otros servicios no tiene que especificar contratos. Si el servicio puede recibir mensajes en el contrato **DEFAULT**, el contrato **DEFAULT** debe estar incluido en la definición de servicio.

Vamos a resumir un caso de éxito en el uso del *Service Broker* ([H27]).

MySpace decidió que la mejor manera de manejar el constante crecimiento de sus bases de datos relacionales, que actualmente suman más de 1 *petabyte*, era escalar horizontalmente y dividir la información a través de múltiples instancias de *SQL Server*. Para ayudar a garantizar la integridad de los datos mientras se

mantiene picos de demanda de servicio de hasta 4,4 millones de usuarios simultáneos, se necesitaba una solución eficiente de mensajería asincrónica entre sus 440 instancias de *SQL Server* y más de 1000 bases de datos.

MySpace creó una solución para que actúe como punto de coordinación para la entrega de mensajes a través de sus bases de datos distribuidas. La solución trabaja en un modelo de broadcast en la que el despachador de servicios asegura que un cambio originario de una base de datos se entrega al grupo de bases de datos destino relevante para la transacción mediante la utilización del *Service Broker*, lo que ha permitido a *MySpace* realizar la gestión de claves foráneas a través de sus 440 servidores de bases de datos, la activación y desactivación de cuentas de sus millones de usuarios.

MySpace también utiliza *Service Broker* administrativa para distribuir los nuevos procedimientos almacenados y otras actualizaciones en todos los 440 servidores de bases de datos a través del despachador que crearon.

8.5 *Ejemplo de funcionamiento*

Después de la descripción teórica, vamos a realizar un pequeño ejemplo que muestra cómo trabaja *Service Broker*. En este ejemplo se detallan los comandos que usaremos en una aplicación con *Service Broker;* además, introduciremos su uso en un sistema de auditoría.

Como ya hemos comentado la base del funcionamiento de esta tecnología es el intercambio de mensajes en un dialogo entre dos servicios. Iremos explicando los comandos básicos en el orden en que es necesario para la creación de los objetos de una conversación ó dialogo.

El primer objeto que se crea es un ***Message Type***, aquí tenemos su sintaxis.

```
CREATE MESSAGE TYPE message_type_name
    [ AUTHORIZATION owner_name ]
    [ VALIDATION = {
    NONE | EMPTY | WELL_FORMED_XML |
    VALID_XML WITH SCHEMA COLLECTION
    schema_collection_name
    } ]
```

Y crearemos un mensaje simple que se valide como un XML bien formado ejecutando el siguiente comando.

```
CREATE MESSAGE TYPE [//Audit/Message] VALIDATION =
    WELL_FORMED_XML
```

El segundo objeto que se crea es el contrato, y en él definiremos los tipos de mensajes permitidos en la conversación; la sintaxis para crear el objeto ***contract.***

```
CREATE CONTRACT contract_name
    [ AUTHORIZATION owner_name ]
    ( { { message_type_name | [ DEFAULT ] }
    SENT BY { INITIATOR | TARGET | ANY }
    } [ ,...n] )
```

Crearemos un objeto tipo ***contract*** que permite que un tipo ***message*** previamente creado sea enviado sólo por el iniciador (*INITIATOR*) de la conversación.

```
CREATE CONTRACT [//Audit/Contract] ([//Audit/Message] SENT BY
    INITIATOR)
```

En este paso creamos el objeto tipo ***Queue,*** una cola guarda los mensajes que llegan desde el otro lado de la conversación y su sintaxis tiene la siguiente definición.

```
CREATE QUEUE [ database_name. [ schema_name ] . | schema_name. ]
    queue_name
```

Vamos a crear una cola simple que tiene un Procedimiento almacenado que se ejecutará cada vez que un nuevo mensaje llegue a la cola, con un máximo de 50 ejecuciones concurrentes del procedimiento y el usuario bajo cuyo contexto se ejecuta el procedimiento almacenado.

```
CREATE QUEUE dbo.TargetAuditQueue

    WITH STATUS=ON,

    ACTIVATION (

    PROCEDURE_NAME = usp_WriteAuditData,

    MAX_QUEUE_READERS = 50,

    EXECUTE AS 'dbo' );
```

Cada cola está asociada a un servicio, así que el siguiente paso se crea el objeto tipo ***Service,*** y su sintaxis es la siguiente.

```
CREATE SERVICE service_name
    [ AUTHORIZATION owner_name ]
    ON QUEUE [ schema_name. ]queue_name
    [ ( contract_name | [DEFAULT] [ ,...n ] ) ]
```

Tenemos que crear un objeto tipo servicio con un contrato y una cola previamente creados.

```
CREATE SERVICE [//Audit/DataWriter]
    AUTHORIZATION dbo
    ON QUEUE dbo.TargetAuditQueue ([//Audit/Contract])
```

Una vez que sabemos cómo crear los objetos que son necesarios para que funcione *Service Broker,* vamos a crear la infraestructura para realizar la auditoría de una base de datos. Usaremos la auditoría de disparadores DML vista en el punto 4, y crearemos dos bases de datos cada una con su propio *Service Broker,* en la base de datos *MAuditD* almacenamos la información de la auditoría y la base de datos que se audita es *TDb1*. En este primer ejemplo no vamos a usar certificados en la comunicación entre los dos *Service Broker,* así que se configura las dos bases de datos con *SET TRUSTWORTHY ON*. También cada una de las bases de datos tiene su propia tabla de error donde se guardan los errores producidos en la comunicación de *Service Broker.*

Cuando insertamos, actualizamos o eliminamos el trigger o disparador toma los datos necesarios de las tablas lógicas *Inserted* y *Deleted,* los convierte a formato XML con que definimos el mensaje, y entonces es cuando hacemos uso

de *Service Broker* para enviar los datos al *Service Broker* de la base de datos MAuditD que guardará la información en la tabla de auditoría. Cada vez que un nuevo mensaje llega a la cola destino, un procedimiento almacenado se ejecuta insertando el mensaje que tenemos en cola en la tabla de auditoría. En sistemas muy ocupados un único procedimiento almacenado posiblemente no podría hacer frente a todos los mensajes entrantes, por este motivo configuramos la cola para que permita ejecuciones concurrentes. Tenemos que tener en cuenta que al ser la recepción de mensajes en la cola una operación asíncrona no afecta al rendimiento en el extremo del iniciador (INITIATOR) y la activación del disparador finaliza.

Este método de auditoría no afecta negativamente al rendimiento general del sistema. Únicamente la base de datos *MAuditD* puede crecer de tamaño, pero esto no afecta al rendimiento de la base de datos auditada *TDb1*.

8.6 *Aplicación en la auditoría de Bases de Datos*

El objetivo de nuestro proyecto es buscar formulas que mejoren el trabajo de auditar una base de datos, y uno de los aspectos más importantes es conseguir optimizar el rendimiento del sistema donde se realiza la auditoría. Como hemos observado en las pruebas detalladas en el punto 7 de este documento existe una importante carga de trabajo cuando se realiza la monitorización de las acciones que se ejecutan en la base de datos. Creemos que si podemos aprovechar las ventajas que nos ofrece *Service Broker* en cuanto a comunicación asíncrona, seguridad y escalabilidad; con toda seguridad conseguiremos mejorar el rendimiento de sistema.

8.6.1 **Notificación de Eventos**

Las notificaciones de eventos se ejecutan como respuesta a una variedad de instrucciones del lenguaje de definición de datos (DDL) *Transact-SQL* y eventos de Traza de SQL enviando información acerca de esos eventos a un

servicio de *Service Broker*. Las notificaciones de eventos se pueden usar para realizar lo siguiente:

- Registrar y revisar cambios o actividades que se producen en la base de datos.
- Realizar una acción en respuesta a un evento de una forma asincrónica en lugar de sincrónica.

Las notificaciones de eventos pueden ofrecer una alternativa de programación a los *Triggers DDL* y a la Traza SQL.

Las notificaciones de eventos se ejecutan asincrónicamente, fuera del alcance de una transacción. Por consiguiente, a diferencia de los *Triggers DDL*, las notificaciones de eventos se pueden usar dentro de una aplicación de bases de datos para responder a eventos sin utilizar los recursos definidos por la transacción inmediata. A diferencia de la Traza de SQL, las notificaciones de eventos se pueden utilizar para realizar una acción en una instancia de *SQL Server* como respuesta a un evento de Traza de SQL.

Cuando se crea una notificación de eventos, se abren una o más conversaciones de *Service Broker* entre una instancia de *SQL Server* y el servicio de destino que se especifica. Normalmente, las conversaciones permanecen abiertas mientras existe la notificación de eventos como objeto de la instancia de servidores. En algunos casos de error, las conversaciones se pueden cerrar antes de que se quite la notificación de eventos. Esas conversaciones nunca se comparten entre notificaciones de eventos. Cada notificación de eventos tiene sus propias conversaciones exclusivas. Al finalizar una conversación explícitamente se impide que el servicio de destino reciba más mensajes y la conversación no se vuelve a abrir la próxima vez que se activa la notificación de eventos.

La información de eventos se proporciona a *Service Broker* como una variable de tipo **XML** que proporciona información acerca de cuándo se produce un evento, el objeto de la base de datos afectado, la instrucción de lote *Transact-*

SQL implicada y otra información. Los datos de eventos pueden ser utilizados por aplicaciones que se ejecutan junto con SQL Server para realizar un seguimiento del progreso y tomar decisiones. Por ejemplo, la siguiente notificación de eventos envía un aviso a un servicio determinado cada vez que se emite una instrucción ALTER TABLE en la base de datos de ejemplo **AdventureWorks**.

```
USE AdventureWorks
GO
CREATE EVENT NOTIFICATION NotifyALTER_T1
ON DATABASE
FOR ALTER_TABLE
TO SERVICE '//Adventure-Works.com/ArchiveService' ,
   '8140a771-3c4b-4479-8ac0-81008ab17984';
```

Para diseñar una notificación de eventos es preciso determinar los siguientes aspectos:

- El ámbito de la notificación.
- La instrucción o grupo de instrucciones *Transact-SQL* que generan la notificación de eventos.

Podemos especificar una notificación de eventos para que se produzca como respuesta a una instrucción aplicada a todos los objetos de la base de datos actual o a todos los objetos de una instancia de *SQL Server*. El ámbito de las notificaciones de eventos especificadas en los eventos QUEUE_ACTIVATION y BROKER_QUEUE_DISABLED está limitado a colas individuales. No todos los eventos pueden producirse en todos los ámbitos. Por ejemplo, los eventos CREATE_DATABASE sólo se pueden producir en las instancias de servidor.

Por el contrario, las notificaciones de eventos creadas en un evento ALTER_TABLE pueden programarse para que se produzcan en todas las tablas de una base de datos o en todas las tablas de una instancia de servidor.

En el siguiente ejemplo se envía una notificación de una instrucción ALTER TABLE ejecutada en una instancia de servidor a la instancia de *Service Broker* de la base de datos actual.

```
CREATE EVENT NOTIFICATION log_ddl1
  ON SERVER
  FOR ALTER_TABLE
  TO SERVICE '//Adventure-Works.com/ArchiveService' , 'current database';
```

Las notificaciones de eventos pueden diseñarse para que se activen tras ejecutarse un procedimiento almacenado o una instrucción *Transact-SQL* determinados. Como se muestra en el ejemplo anterior, esa notificación de eventos se produce después de un evento ALTER_TABLE.

Las notificaciones de eventos pueden diseñarse para que se activen tras producirse un evento de traza de SQL. Por ejemplo, la siguiente notificación de eventos se activa después de producirse un evento **Object_Created** en el servidor.

```
CREATE EVENT NOTIFICATION log_ddl1
  ON SERVER
  FOR Object_Created
  TO SERVICE '//Adventure-Works.com/ArchiveService', 'current database' ;
```

Como puesta en práctica de estas nociones teóricas hemos diseñado un sistema de notificación de eventos que supervisa y responde a la actividad de una base de datos. Por un lado vamos a supervisar cuándo se crean y quitan las bases de datos en la instancia del servidor y también el momento en que se producen inicios de sesión, cierres de sesión y errores en los inicios de sesión en la instancia del servidor.

Las notificaciones de eventos envían datos XML sobre estos eventos al servicio *Microsoft Service Broker* especificado en el ejemplo. El ejemplo también define una cola para recibir los mensajes y una ruta que especifica la dirección del servicio. En este caso, la dirección es la base de datos local. Indicamos al servidor que debe esperar 60 segundos hasta que el servicio reciba el mensaje y lo convierta en **XML**. Posteriormente se prueban las notificaciones de eventos haciendo que los eventos en los que se crean ocurran en la instancia del servidor ([H23]).

Results | Messages

	message_body	status	priority	queuing_order	conversation_group_id	conversation_handle	message_sequence_number	service_name
1	<EVENT_INSTANCE><EventType>CREATE_DATABASE</Event...	1	0	3213	3506B61F-57F8-DE11-B716-000000000000	3606B61F-57F8-DE11-B716-000000000000	46	//Adventure-Works.com/I
2	<EVENT_INSTANCE><EventType>CREATE_DATABASE</Event...	1	0	3214	3506B61F-57F8-DE11-B716-000000000000	3606B61F-57F8-DE11-B716-000000000000	47	//Adventure-Works.com/I
3	<EVENT_INSTANCE><EventType>DROP_DATABASE</EventTy...	1	0	3215	3506B61F-57F8-DE11-B716-000000000000	3606B61F-57F8-DE11-B716-000000000000	48	//Adventure-Works.com/I
4	<EVENT_INSTANCE><EventType>DROP_DATABASE</EventTy...	1	0	3216	3506B61F-57F8-DE11-B716-000000000000	3606B61F-57F8-DE11-B716-000000000000	49	//Adventure-Works.com/I
5	<EVENT_INSTANCE><EventType>CREATE_DATABASE</Event...	1	0	3221	3506B61F-57F8-DE11-B716-000000000000	3606B61F-57F8-DE11-B716-000000000000	50	//Adventure-Works.com/I
6	<EVENT_INSTANCE><EventType>CREATE_DATABASE</Event...	1	0	3222	3506B61F-57F8-DE11-B716-000000000000	3606B61F-57F8-DE11-B716-000000000000	51	//Adventure-Works.com/I
7	<EVENT_INSTANCE><EventType>DROP_DATABASE</EventTy...	1	0	3223	3506B61F-57F8-DE11-B716-000000000000	3606B61F-57F8-DE11-B716-000000000000	52	//Adventure-Works.com/I
8	<EVENT_INSTANCE><EventType>DROP_DATABASE</EventTy...	1	0	3224	3506B61F-57F8-DE11-B716-000000000000	3606B61F-57F8-DE11-B716-000000000000	53	//Adventure-Works.com/I
9	<EVENT_INSTANCE><EventType>CREATE_DATABASE</Event...	1	0	3237	3506B61F-57F8-DE11-B716-000000000000	3606B61F-57F8-DE11-B716-000000000000	54	//Adventure-Works.com/I
10	<EVENT_INSTANCE><EventType>DROP_DATABASE</EventTy...	1	0	3238	3506B61F-57F8-DE11-B716-000000000000	3606B61F-57F8-DE11-B716-000000000000	55	//Adventure-Works.com/I
11	<EVENT_INSTANCE><EventType>CREATE_DATABASE</Event...	1	0	3239	3506B61F-57F8-DE11-B716-000000000000	3606B61F-57F8-DE11-B716-000000000000	56	//Adventure-Works.com/I
12	<EVENT_INSTANCE><EventType>DROP_DATABASE</EventTy...	1	0	3240	3506B61F-57F8-DE11-B716-000000000000	3606B61F-57F8-DE11-B716-000000000000	57	//Adventure-Works.com/I
13	<EVENT_INSTANCE><EventType>CREATE_DATABASE</Event...	1	0	3241	3506B61F-57F8-DE11-B716-000000000000	3606B61F-57F8-DE11-B716-000000000000	58	//Adventure-Works.com/I
14	<EVENT_INSTANCE><EventType>DROP_DATABASE</EventTy...	1	0	3242	3506B61F-57F8-DE11-B716-000000000000	3606B61F-57F8-DE11-B716-000000000000	59	//Adventure-Works.com/I
15	<EVENT_INSTANCE><EventType>CREATE_DATABASE</Event...	1	0	3245	3506B61F-57F8-DE11-B716-000000000000	3606B61F-57F8-DE11-B716-000000000000	60	//Adventure-Works.com/I
16	<EVENT_INSTANCE><EventType>DROP_DATABASE</EventTy...	1	0	3246	3506B61F-57F8-DE11-B716-000000000000	3606B61F-57F8-DE11-B716-000000000000	61	//Adventure-Works.com/I
17	<EVENT_INSTANCE><EventType>CREATE_DATABASE</Event...	1	0	3247	3506B61F-57F8-DE11-B716-000000000000	3606B61F-57F8-DE11-B716-000000000000	62	//Adventure-Works.com/I
18	<EVENT_INSTANCE><EventType>DROP_DATABASE</EventTy...	1	0	3248	3506B61F-57F8-DE11-B716-000000000000	3606B61F-57F8-DE11-B716-000000000000	63	//Adventure-Works.com/I

Tenemos, por tanto, una técnica que nos permite auditar eventos muy significativos que se producen en nuestra base de datos, siendo una auditoría con una capacidad de personalización muy alta ya que podemos elegir aquellos eventos que nos interesen, evitando recolecciones de datos innecesarias.

Al ser una tecnología integrada dentro de *SQL Server 2005*, aprovecha todos los recursos de S.G.B.D. y permite que el rendimiento del servidor no se vea penalizado por la realización de la auditoría. Esto se fundamenta en la forma de trabajar asíncrona de los eventos de notificación.

Hemos diseñado otro ejemplo incluye una implementación simple de un servicio que recibe y archiva mensajes de notificación de eventos. La diferencia con el anterior ejemplo es que almacena notificaciones de eventos en dos tablas diferentes. Cada tabla usa un formato diferente. Una tabla almacena los datos de la notificación de eventos en formato relacional. La otra almacena una parte de la información de la notificación de eventos en formato relacional, guardando el mensaje completo en formato XML. Describiremos a continuación el funcionamiento del script que se adjunta a este documento.

Como es preceptivo se activa la función que permite trabajar con los mensajes de *Service Broker*, si no estuviera activo. Se crea la cola para el servicio y se crea un servicio para recibir las notificaciones de los eventos. El servicio acepta conversaciones en base al contrato generado para las notificaciones de los eventos del sistema. Crearemos las tablas para almacenar la información de los eventos, como hemos dicho anteriormente.

Se crea un procedimiento almacenado para procesar los mensajes de los eventos y que lee cada mensaje de notificación de eventos desde la cola y desde el mensaje almacena la información dentro de ambas tablas.

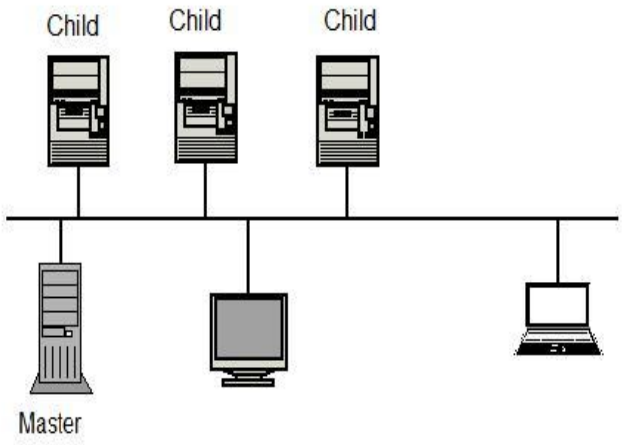
8.6.2 **Auditoria asíncrona centralizada a través de dos instancias y dos servidores.**

En esta implementación un servidor audita datos de otro servidor o más. Es un sistema escalable y reutilizable con los siguientes requisitos.

- Hay una instancia de base de datos en un servidor donde se almacenará toda la información que le llegue, es decir es el servidor que recolecta los datos para la auditoría. A esta instancia la llamaremos Master.
- Existirá otro servidor en el cual está la instancia de base de datos cuyo trabajo queremos auditar. A esta instancia la llamaremos

Child. Tendría que se posible añadir tantas nuevas instancias del tipo Child como queramos sin que se tuviera que modificar la instancia Master, para que sea un sistema escalable.

- Debe estar habilitada la comunicación segura entre las instancias.



Vamos a configurar la instancia de base de datos Master en el Servidor Master para que se cumplan los requisitos anteriores.

1. Crear Master key
2. Crear Certificate (CertificateAuditDataReceiver) que se cifra con la Master key y tiene Active for BEGIN_DIALOG = ON
3. Hacer copia de seguridad en fichero de Certificate, este fichero tendrá que ser copiado en el servidor Child, y poder usarlo para autenticar.
4. Crear el Service Broker Endpoint y autenticar con el Certificate creado en el paso 2.
5. Crear la base de datos MasterAuditDatabase y establecer ENABLE_BROKER a ON

Ahora, vamos a configurar la instancia de base de datos Child en el Servidor Child para que se cumplan los requisitos anteriores.

1. Crear un login nuevo y un nuevo usuario para en la instancia de base de datos Child.
2. Crear Master key.
3. Crear Certificate (CertificateAuditDataSender) que se cifra con la Master key y tiene Active for BEGIN_DIALOG = ON.
4. Crear Certificate (CertificateAuditDataReceiver) desde el fichero copiado en el Servidor Master y autorizarlo con el usuario (no el login) creado en el paso primero.
5. Crear el SB Endpoint y autenticar con el Certificate creado en el paso 3.
6. Conceder permisos de conexión sobre el Endpoint al login (no al usuario) creado en el paso 1.

Ahora tenemos configurada la seguridad para el transporte entre el Master Server y el Child Server.

La comunicación entre instancias de *Service Broker* depende de las rutas porque estas indican a los mensajes su destino. Para dos *Service Broker* en diferentes instancias que se comunican entre ellas, cada *Service Broker* debe tener la ruta correspondiente a otro *Service Broker*.

Al reutilizar “Dialog Conversation” , nos aseguramos que no tendremos que empezar una nueva conversación para cada mensaje, esto es bueno por motivos de rendimiento. Es decir, tener 1 conversación con 100 mensajes tomará menos recursos que 100 mensajes con 1 mensaje cada uno.

Debido a que en una auditoria hay un constante flujo de datos entre la instancia del Child Server y la instancia del Master Server elegimos tener una conversación por base de datos abierta constantemente.

Orden de ejecución de los scripts.

1. Ejecutar 1_Master_Server_Master_Database, (Data Receiver).
2. Ejecutar 2_Child_Server_Master_Database, (Data Receiver).
3. Copiar el certificado creado en el punto 1 a Child Server, (Data Sender).
4. Copiar el Service Broker Id devuelto en el punto 2 en el script 3_Master_Server_MasterAuditDatabase.
5. Ejecutar 3_Master_Server_MasterAuditDatabase en el Child server (Data Sender)
6. Cambiar la IP's en el script 4_Child_Server_Audited_Database, con las nuevas IP's de nuestros servidores.
7. Ejecutar 4_Child_Server_Audited_Database en el Child Server (Data senders).
8. Ejecutar 5_Test, tiene dos partes, una para cada instancia.

Las pruebas que hemos realizado y los resultado obtenidos, nos permiten afirmar que esta tecnología es una opción a tener en cuenta. Permite reducir la saturación en las comunicaciones de datos entre los servidores haciendo que el rendimiento del sistema mejore. Sin embargo, este estudio se ha realizado en un sistema experimental a pequeña escala y que requiere un desarrollo más profundo.

Los script y resultados están incluidos en el material anexo a este documento.

9 Conclusiones y líneas futuras

9.1 *Conclusiones*

Nuestro trabajo consiste en recopilar, organizar, analizar información de la actividad de un sistema informático para poder ofrecer un resultado que nos ayude a tomar decisiones con el objetivo de mejorar el rendimiento de este sistema informático. Además tenemos que tener en cuenta otras variables que afectan al rendimiento, siendo la más importante el nivel de seguridad que queremos tener en nuestro sistema. Así que tenemos que diseñar, teniendo en cuenta lo anteriormente expuesto, una sistema de bases de datos seguro, efectivo y eficiente.

Sabemos que existe gran dependencia de los profesionales experimentados, esto es a la vez una ventaja y un lastre para la evolución del campo de la auditoría informática de bases de datos. La labor de auditoría tiene la independencia como pilar fundamental. Debemos encontrar técnicas de trabajo que estandaricen el trabajo del auditor para conseguir resultados mejores, más rentables y más óptimos.

Usando las tecnologías y herramientas que el sistema gestor de base de datos SQL Server 2005 aporta en materia de almacenamiento y recuperación de datos para realizar auditoria podemos crear esas metodologías de trabajo independientes. Por otra parte, la mala utilización de estas tecnologías y herramientas pueden ser perjudiciales para el funcionamiento del sistema gestor, ralentizado o incluso bloqueado su actividad.

Durante la elaboración de este documento hemos estudiado las tecnologías y herramientas SQL Server 2005, investigando su funcionamiento, realizando pruebas e introduciendo nuevas líneas de uso; pretendiendo diseñar la infraestructura y actividades necesarias para llevar a cabo la labor de auditoría informática con SQL Server 2005. Hemos utilizado juegos de pruebas, programas de medición de rendimiento y análisis de datos para llegar a

configurar una infraestructura de dos servidores que optimizan de manera general el rendimiento del servidor de base de datos para aumentar la eficiencia del mismo. Para evitar que la red se vea colapsada por el tráfico de información de de un servidor otro hemos introducido el uso de una tecnología que permite gestionar el flujo de mensajes entre servidores, creando colas que almacenan dichos mensajes y que permiten el tratamiento de los mensajes en cada servidor de forma independiente evitando tiempos de espera. Tenemos en la infraestructura otro ordenador que será usado por un auditor informático para obtener información sobre datos de auditoría, sobre hechos excepcionales y desarrollar políticas de auditoría.

El objetivo de nuestra labor de investigación ha sido tratar que el grado de disponibilidad y eficiencia de los dos servidores sea el mayor posible dentro de las posibilidades técnicas que hemos tenido a nuestro alcance.

Para que una base de datos pueda tener un sistema de auditoría adecuado, hemos establecido unos niveles de auditoría en función de la seguridad requerida por la base de datos de la organización. Llegamos a la conclusión de que estos niveles son necesarios, ya que el rendimiento obtenido en la realización de las pruebas hacen ver que el coste de mantenimiento de la auditoría de base de datos varía con respecto al uso de la misma.

Establecimos unos niveles de auditoría en base a dos pilares que determinan la necesidad de auditoría de una base de datos, tanto si es de una gran organización como de una de tamaño pequeño.

- La ley: los diferentes niveles de infracciones establecidos en el artículo 44 de la LOPD [3].
- El sistema informático: la maximización de la disponibilidad y el rendimiento de la base de datos.

Tendremos tres niveles:

- Básico: abarca la protección de la base de datos para prevenir infracciones leves según el artículo 44.2 de la LOPD ([2]). Se basa en establecer y documentar los permisos concedidos a los usuarios para detectar posibles huecos de seguridad.

- Medio: abarca la protección de la base de datos para prevenir infracciones graves según el artículo 44.3 de la LOPD ([2]). Se basa en detectar posibles irregularidades en el funcionamiento de la base de datos.

- Alto: abarca la protección de la base de datos para prevenir infracciones muy graves según el artículo 44.4 de la LOPD ([2]). Consiste en aumentar aún más la seguridad, sin disminuir la disponibilidad de la base de datos.

La organización deberá establecer el nivel de seguridad que cree más conveniente para su correcto funcionamiento y aplicarlo.

Finalmente, hemos realizado un estudio profundo de la tecnología Service Broker integrada SQL Server 2005 con el objetivo que sea una herramienta útil para la auditoría de Bases de datos, ya que queremos mejorar la monitorización de una base de datos aprovechando capacidad para almacenar mensajes en colas e intercambiar estos mensajes mediante comunicación asíncrona asegurando la información. Las colas permiten que una aplicación realice un trabajo en una transacción distinta de la transacción que solicita el trabajo. *Service Broker* amplía esta idea, de forma que las aplicaciones puedan llevar a cabo el trabajo en una instancia o en equipos diferentes. *Service Broker* es un gran recurso para el trabajo con bases de datos, ya que proporciona colas integradas en la base de datos y una mensajería transaccional confiable entre instancias.

Hemos buscado formulas que puedan mejorar el trabajo de auditar una base de datos, siendo uno los aspectos más importantes conseguir optimizar el

rendimiento del sistema donde se realiza la auditoría. Hemos intentado aprovechar las ventajas que nos ofrece Service Broker en cuanto a comunicación asíncrona, seguridad y escalabilidad para mejorar el rendimiento de sistema. Dentro de esta dinámica de trabajo se he introducido el uso de las notificaciones de eventos para registrar y revisar cambios o actividades que se producen en la base de datos ó para realizar una acción en respuesta a un evento de una forma asincrónica en lugar de sincrónica.

Y como aplicación final de la tecnología se ha creado un sistema que realiza una auditoria asíncrona centralizada a través de dos instancias de base de datos SQL Server y dos servidores. Con lo que hemos podido demostrar que esta tecnología favorece el rendimiento del sistema implantado, aunque tiene la deficiencia de la complejidad de su aplicación ya que son necesarios unos conocimientos técnicos avanzados dada la novedad de la tecnología.

9.2 *SQL Server 2008. Nuevas herramientas de auditoría*

SQL Server 2008 introduce una capacidad de auditoría completa y profundamente integrada que ofrece importantes mejoras con respecto a las versiones anteriores del software de base de datos *Microsoft ® SQL Server ®*.

En *SQL Server 2005* la auditoría era realizada mediante una combinación de herramientas que hemos detallado a lo largo de este texto. Todas las herramientas descritas anteriormente permanecen en *SQL Server 2008*, y seguirá siendo útil para una amplia gama de necesidades. Sin embargo, *SQL Server 2008* nos trae una capacidad de auditoría nueva, más centrada, y más profundamente integrada.

En primer lugar, el objeto *SQL Server Audit* ([H21])va a sustituir a *SQL Trace* como la solución preferida de auditoría ya que está desarrollado exclusivamente para usos de auditoría; y en su diseño se ha tenido como objetivo el aumento de la seguridad, la facilidad de administración, la mejoría del rendimiento y la mejoría de acceso a los datos de auditoría. Es importante reseñar

que la nueva función de *SQL Server Audit* aporta ventajas de rendimiento y que los objetos de auditoría se pueden crear y manipular a través de instrucciones *DDL* o a través de *SQL Server Management Studio*.

Otro punto importante de mencionar acerca de *SQL Server Audit* es el nivel de detalle al que podemos llegar en la auditoría. La auditoría de la actividad de los usuarios, roles o grupos sobre cada uno de los objetos de base de datos se puede restringir hasta el nivel de tabla. Por ejemplo, podríamos registrar todos los *Update* realizados en la tabla **nómina** por el usuario DBO.

La auditoría es un componente en una estrategia de seguridad de base de datos y por tanto *SQL Server Audit* será seguro. Al ser un objeto de base de datos, el privilegio para crear, eliminar o modificar un objeto Audit será controlado en el modelo de permisos del motor de base de datos. Tenemos dos nuevos permisos a nivel de servidor y de base de datos, ALTER ANY SERVER AUDIT para el objeto *Server Audit Specification* y ALTER ANY DATABASE AUDIT para *Database Audit Specification.*

El registro de auditoría es otro activo que necesita protección. Debe prestarse especial atención con respecto a dónde enviar los datos de la auditoría. Idealmente, la información de la auditoría debe enviarse a una ubicación que no puede ser modificada o alterada, incluso por un administrador del sistema. Una estrategia para lograr esto es enviar los eventos de auditoría en un archivo compartido a la sólo tiene acceso de escritura la cuenta *SQL Server Service;* el auditor tendría acceso de lectura a este archivo, pero nadie más.

Otra opción es el registro de seguridad de Windows. Esta es una buena elección para alguien que quiere almacenar el registro de *SQL Server Audit* localmente y necesita el nivel de seguridad de *sysadmin* o de administrador local de Windows.

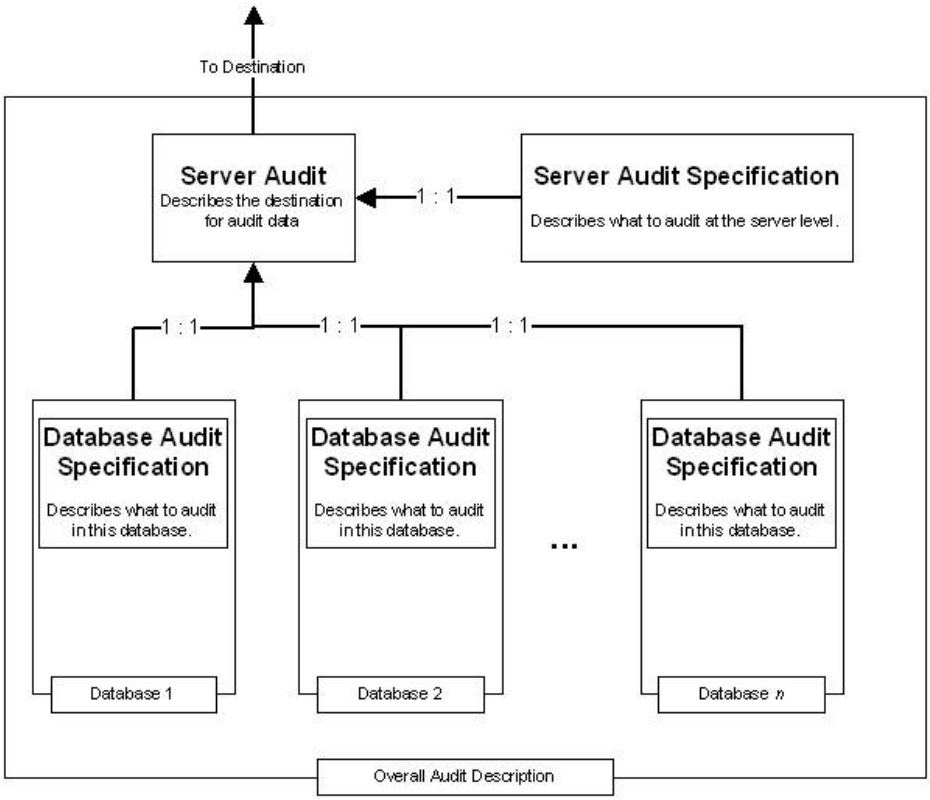
SQL Server 2008 introduce una nueva infraestructura de eventos de alto rendimiento llamada *SQL Server Extended Events*. *SQL Server Audit* provecha

las ventajas de rendimiento y proporcionan capacidad de escritura sincrónica y asincrónica; de forma predeterminada, los eventos de auditoría se escriben en el destino de auditoría de forma asincrónica para mejorar el rendimiento. La opción QUEUE_DELAY = 1000 de la sentencia de auditoría CREATE AUDIT DDL indica que *SQL Server Audit* situará en la cola el registro de auditoría un segundo antes de escribir en el destino.

Como resumen diremos que en la nueva versión de *SQL Server* disponemos de un nuevo sistema de auditoría, implementado a través del objeto *SQL Server Audit*. Este objeto, nos permite definir auditoría a nivel de instancia o a nivel de base de datos, y almacenar los resultados de la auditoría bien en ficheros, o bien en el Visor de Sucesos de Sistema Operativo, ya sea en el registro de Aplicación o en el registro de Seguridad, y que será el que analizaremos en este apartado.

El primer objeto que debemos de crear es un objeto *SQL Server Audit*, en él que definimos el ámbito de la auditoría de *SQL Server*, y configuramos es el destino de la auditoria. Una vez hemos creado el objeto *SQL Server Audit* debemos de especificar cuáles son los eventos que deseamos auditar. Estos eventos podemos especificarlos utilizando los denominados *Audit Groups*, que no son más que grupos de acciones, a dos niveles:

- A nivel de instancia, a través de la *Server Audit Specification*. Nos permite auditar eventos que ocurren a nivel de instancia de *SQL Server*, tales como inicios de sesión, copias de seguridad, ejecución de comandos DBCC, o eventos de sesiones de *Service Broker*.



- A nivel de base de datos, a través de la *Database Audit Specification*. Para aquellos eventos que deseemos auditar a nivel de objetos u operaciones de las bases de datos, debemos de utilizar los *Audit Groups* a nivel de base de datos, que nos permiten auditar eventos tales como los eventos de creación, borrado o eliminación de objetos, o cambios en la gestión de roles de usuarios.

El proceso general de creación y uso de una auditoría es el siguiente ([H22]):

1. Crear una auditoría y definir el destino.
2. Crear una especificación de auditoría de servidor o una especificación de auditoría de base de datos que se asigne a la auditoría, habilitando la especificación de auditoría.

3. Habilitar la auditoría.
4. Leer los eventos de auditoría mediante el **Visor de eventos** de Windows, el **Visor del archivo de registros** o la función *fn_get_audit_file*.

En el siguiente ejemplo, vemos como se puede crear una auditoría que registro los inicios de sesión fallidos:

```
--Creamos el objeto de Auditoria que define la ubicación

CREATE SERVER AUDIT AuditTest

TO FILE

(    FILEPATH = N'C:\temp\'

    ,MAXSIZE = 0 MB

    ,MAX_ROLLOVER_FILES = 2147483647

    ,RESERVE_DISK_SPACE = OFF

)

WITH

(    QUEUE_DELAY = 1000

    ,ON_FAILURE = CONTINUE



)

GO

--Comprobamos que se ha creado correctamente

select * from sys.server_audits

GO
```

```
--Inspeccionamos cuales son las acciones que podemos auditar

Select * from sys.dm_audit_Actions

GO

--Creamos una especificación a nivel de instanacia para registrar los inicios de sesión fallidos

CREATE SERVER AUDIT SPECIFICATION AuditServerSpecification

FOR SERVER AUDIT AuditTest

ADD (FAILED_LOGIN_GROUP)

GO

--comprobamos que se ha creado correctamente

select * from sys.server_audit_specifications

select * from sys.server_audit_specification_details

GO

--Activamos el objeto de Auditoria

ALTER SERVER AUDIT AuditTest WITH(STATE= ON)

GO

--Activamos la especificación

ALTER SERVER AUDIT SPECIFICATION AuditServerSpecification WITH(STATE= ON)

-- Leemos los datos del fichero de auditoria

select * from fn_get_audit_file('C:\Temp\*',DEFAULT, default)
```

Si quisiéramos crear una auditoria a nivel de base de datos que, por ejemplo, auditase las sentencias SELECT ejecutadas en una determinada tabla, podríamos hacerlo del siguiente modo:

```
USE AdventureWorks

GO

CREATE DATABASE AUDIT SPECIFICATION AuditSelect

FOR SERVER AUDIT [AuditTest]

ADD(SELECT ON HumanResources.EmployeePayHistory BY dbo)

WITH (STATE = ON)

GO
```

De este modo, podemos crear sistemas de auditoría muy completos, y a la vez, no intrusivos, por lo que el impacto en el rendimiento es menor que el de otras aproximaciones.

Para justificar esta última afirmación vamos a comparar las diferencias de rendimiento ofrecidas por las auditorías *SQL Trace* en modo *C2* activado *y SQL Server Audit* configurado para auditar eventos equivalentes; y almacenado los resultados en un fichero.

La prueba que realizaremos tiene 2 bases de datos con un tamaño que varía entre 64MB y 400 MB. Tienen unas 35 tablas con una 40 mil filas de media. Los tipos de datos de los campos son **bit**, **varbinary**, **int** y **nvarchar.** Se han ejecutado un millón y medio de sentencias.

Base Time (min)	SQL Trace (min)	SQL Server Audit (min)	Coste Vs. Base	Benf. vs. Trace
41.30	101.85	55.93	35.42%	45.09 %

Descripción del cálculo:

- COSTE VS BASE = (SQL AUDIT– BASE TIME) / BASE
- BENEF VS TRACE = (SQL TRACE – SQL AUDIT) / SQL TRACE.

En esta prueba se generan una gran cantidad de eventos de auditoría que realza la eficiencia ante los ficheros de E/S ([H21]) .

10 Bibliografía

[H1].Casos de Éxito de Microsoft SQL Server [en línea]. [España]: Microsoft, 2009 [ref. 10 de Diciembre de 2009]. Disponible en Web: http://www.microsoft.com/spain/sql/productinfo/casestudies/default.mspx

[H2]. España. Ley Orgánica 15/1999, de 13 de diciembre, de protección de datos de carácter personal [en línea]. Boletín Oficial del Estado, 14 de diciembre de 1999, núm.298, p. 43088-43099. Consultado Diciembre 2009. Disponible en Web: http://www.boe.es/boe/dias/1999/12/14/pdfs/A43088-43099.pdf

[H3]. Microsoft SQL Server [en línea]. [EEUU]: Wikipedia, 2009 [ref. 14 de Diciembre de 2009]. Disponible en http://en.wikipedia.org/wiki/Microsoft_SQL_Server

[H4].SQL Server 2005 y Visual Studio 2005: Mejor Juntos [en línea]. [España]: Microsoft, 2009 [ref. 11 de Diciembre de 2009]. Disponible en Web http://www.microsoft.com/latam/technet/prodtechnol/sql/themes/default.mspx

[H5]. R. Vieira. Fundamentos Programación SQL Server 2005. Ed. Anaya

[H6]. Miguel A. Ramos. UC3M. Apuntes de Auditoría Informática, 2007.

[H7]. I. Santos Forner. UC3M. Apuntes de Auditoria de Bases de Datos.

[H8]. SQL ServerCentral.com. Basics of C2 Auditing. Dinesh Asanka, 2004. [ref. 15 de Diciembre de 2009]. Disponible en Web http://www.sqlservercentral.com/articles/Monitoring/basicsofc2auditing/1547/

[H9].Cómo configurar la opción C2 audit mode. Libros en Pantalla [en línea]. [España]: Microsoft, 2009 [ref. 15 de Diciembre de 2009]. Disponible en Web: http://msdn.microsoft.com/en-us/library/ms189114(SQL.90).aspx

[H10].Procedimientos almacenados. Libros en Pantalla [en línea]. [España]: Microsoft, 2009 [ref. 14 de Diciembre de 2009]. Disponible en Web: http://msdn.microsoft.com/es-es/library/ms190782(SQL.90).aspx

[H11].Desencadenadores DML. Libros en Pantalla [en línea]. [España]: Microsoft, 2009 [ref. 10 de Diciembre de 2009]. Disponible en Web: http://msdn.microsoft.com/es-es/library/ms191524(SQL.90).aspx

[H12]. Introducción a SQL Server Profiler. Libros en Pantalla [en línea]. [España]: Microsoft, 2009 [ref. 10 de Diciembre de 2009]. Disponible en Web: http://msdn.microsoft.com/es-es/library/ms181091(SQL.90).aspx

[H13]. Plantillas del analizador SQL Server. Libros en Pantalla [en línea]. [España]: Microsoft, 2009 [ref. 10 de Diciembre de 2009]. Disponible en Web: http://msdn.microsoft.com/es-es/library/ms190176(SQL.90).aspx

[H14]. Referencia de las clases de evento de SQL Server. Libros en Pantalla [en línea]. [España]: Microsoft, 2009 [ref. 12 de Diciembre de 2009]. Disponible en Web: http://msdn.microsoft.com/es-es/library/ms175481(SQL.90).aspx

[H15]. Temas de procedimiento de supervisión de la actividad y rendimiento del servidor. Libros en Pantalla [en línea]. [España]: Microsoft, 2009 [ref. 12 de Diciembre de 2009]. Disponible en Web: http://msdn.microsoft.com/es-es/library/ms191511(SQL.90).aspx

[H16]. Cómo utilizar un procedimiento almacenado para supervisar las trazas de SQL Server 2005. Articulo id. 912914 [en línea]. [España]: Soporte Microsoft, 2007 [ref. 12 de Diciembre de 2009]. Disponible en Web: http://support.microsoft.com/kb/912914/es

[H17]. Desencadenadores DDL. Libros en Pantalla [en línea]. [España]: Microsoft, 2009 [ref. 18 de Diciembre de 2009]. Disponible en Web: http://technet.microsoft.com/es-es/library/ms190989.aspx

[H18]. todosql.com. TodoSQL El blog de SQL Server en español [ref. 17 de Diciembre de 2009]. Disponible en Web: http://www.todosql.com/blog/200801/auditando-con-ddl-triggers-en-sql-server-2005

[H19]. EVENTDATA() Función de Transact-SQL. Libros en Pantalla [en línea]. [España]: Microsoft, 2009 [ref. 18 de Diciembre de 2009]. Disponible en Web: http://msdn.microsoft.com/es-es/library/ms173781(SQL.90).aspx

[H20]. Quest Central for SQL Server.Performance Analisys. [ref. 19 de Diciembre de 2009]. Disponible en Web: http://www.quest.com/quest_central_for_sql_server/performance_analysis.aspx

[H21]. Auditing in SQL Server 2008. SQL Server Technical Article. [ref. 19 de Diciembre de 2009]. Disponible en Web: http://msdn.microsoft.com/en-us/library/dd392015.aspx

[H22]. Descripción de SQL Server Audit. Libros en Pantalla [en línea]. [España]: Microsoft, 2009 [ref. 28 de Diciembre de 2009]. Disponible en Web: http://msdn.microsoft.com/es-es/library/cc280386.aspx

[H23]. Notificación de Eventos, Ejemplos. Libros en Pantalla [en línea]. [España]: Microsoft, 2009 [ref. 28 de Diciembre de 2009]. Disponible en Web: http://msdn.microsoft.com/es-es/library/cc280386.aspx

11 Anexo I: Elementos incluidos en el CD.

datos.sql: muestra la tabla usada en las pruebas.

prueba1.txt: comando y resultado prueba 1 sin auditoría

prueba1A.txt: comando y resultado prueba 1 auditoría C2

prueba1B.txt: comando y resultado prueba 1 auditoría Profiler

prueba2.sql: script utilizado para la pruebas 2

prueba2A.txt: comando prueba 2 auditoría C2

prueba2B.txt: comando prueba 2 auditoría Profiler

prueba3.sql: script utilizado para la prueba 3

prueba3A.txt: comando prueba 3 auditoría C2

prueba3B.txt: comando prueba 3 auditoría Profiler

prueba4.sql: script utilizado para la prueba 4

prueba4A.txt: comando prueba 4 auditoría C2

prueba4B.txt: comando prueba 4 auditoría Profiler

scripts_SB_child: Carpeta que contiene los scripts que hay que ejecutar en el servidor Child de la implementación de Service Broker. También contiene las salidas de los resultados de la ejecución.

scripts_SB_master: Carpeta que contiene los scripts que hay que ejecutar en el servidor Master de la implementación de Service Broker. También contiene las salidas de los resultados de la ejecución.

Tutorial_ServiceBroker : Carpeta que contiene un proyecto de SQL Server con un tutorial para crear objetos de Service Broker

pruebaAuditoria1: Carpeta que contiene los scripts para activa, desactivar y leer la traza generada con modo auditoria C2