UCA
Pontificia Universidad Católica Argentina
Biblioteca digital
de la Universidad Católica Argentina

Vignolo, Alejandro ; Joffre, Andrés

Solución práctica para réplica asincrónica y auditoría de bases de datos

Tesis de Licenciatura en Sistemas y Computación
Facultad de Química e Ingeniería "Fray Rogelio Bacon"

Este documento está disponible en la Biblioteca Digital de la Universidad Católica Argentina, repositorio institucional desarrollado por la Biblioteca Central "San Benito Abad". Su objetivo es difundir y preservar la producción intelectual de la Institución.
La Biblioteca posee la autorización del autor para su divulgación en línea.

Cómo citar el documento:

Vignolo A, Joffre A. (2018). Solución práctica para réplica asincrónica y auditoría de bases de datos [en línea]. Tesis de Licenciatura en Sistemas y Computación. Universidad Católica Argentina. Facultad de Química e Ingeniería "Fray Rogelio Bacon". Disponible en: http://bibliotecadigital.uca.edu.ar/greenstone/cgi-bin/library.cgi?a=d&c=tesis&d=solucion-practica-replica-asincronica [Fecha de consulta:...]

Facultad de Química e Ingeniería del Rosario

Pontificia Universidad Católica Argentina

Cátedra de Seminario de Sistemas

Solución Práctica para Réplica Asincrónica y Auditoría de Bases de Datos

Presentado en cumplimiento parcial de los requisitos para la obtención del título de

LICENCIADO EN SISTEMAS Y COMPUTACIÓN

Vignolo Alejandro y Joffre Andrés

Rosario, 09/10/2018

Resumen

El siguiente trabajo de tesis se enmarca en el área de las Bases de Datos SQL con una orientación de aplicación comercial y se propone desarrollar una herramienta para la réplica de Bases de Datos y la auditoría de las modificaciones realizadas sobre éstas independientemente de cuál sea su tamaño. Para obtener el resultado esperado se desea utilizar técnicas de réplica unidireccional y asincrónica (de actualización periódica).

El desarrollo de la investigación se basa en las herramientas comerciales actuales más utilizadas para la réplica de Bases de Datos junto con las herramientas comerciales actuales más utilizadas para la auditoria de Bases de Datos.

En este trabajo se demuestra una solución, desarrollada, aplicada y funcionando correctamente. Esto no significa que sea la única solución ni que sea la más óptima. Por lo cual se concluye en algunas mejoras posibles a futuro que no fueron consideradas durante la primera versión funcional de la herramienta.

Índice

1. Introducción

La investigación y posterior desarrollo de la herramienta descripta en el resumen tiene su base en distintas razones que justifican y finalmente enmarcan esta tesis. Existe un interés personal por demostrar a la institución que nos educó y nuestras familias la posibilidad de una aplicación directa de los conocimientos obtenidos durante los años de aprendizaje, así como la búsqueda constante de formas de simplificar herramientas existentes. Por otro lado, existe un interés profesional de desarrollar una alternativa más eficiente y menos costosa al software disponible.

Se debe tener en cuenta el contexto comercial, geográfico e histórico. El área comercial local requiere soluciones que no encarezcan el producto final ni su implementación para poder mantener un perfil de competencia. En este aspecto podemos señalar la existencia de un gran rango de soluciones computacionales en el ambiente comercial, desde empresas que utilizan versiones anticuadas de software y hardware hasta empresas que utilizan lo último disponible. Dado que este rango se mantuvo en el tiempo, se desea una herramienta que pueda adaptarse a las fluctuaciones futuras.

Es necesario considerar que vivimos y nos desempeñamos en un país de gran extensión pero con una infraestructura de conexión que tiene margen para mejorar. La falta de confiabilidad y velocidad en la infraestructura de internet nos empuja a optimizar nuestra solución de forma que pueda ser aplicada en la vasta variedad de lugares que pueden requerir del servicio.

En base a lo mencionado previamente nos localizamos en un escenario basado en la ciudad de Rosario y su zona comercial de influencia. El rango de clientes que requieren servicios de réplica de Bases de Datos y su posterior Auditoría no se limita solo a PyMEs, y en general incluye empresas cuyos datos se encuentran distribuidos en distintos lugares geográficos (sucursales) dentro del espacio mencionado. En casos se observan sucursales que no cuentan con conexiones a internet óptimas dada la infraestructura disponible, y

deben ser estas las que influencien el desarrollo. En promedio se consideran Bases de Datos con tamaños de 5 GB a 100 GB, con un movimiento de cientos de MB diarios.

Los aspectos considerados en los párrafos anteriores nos llevan a identificar necesidades que sirven como punto de partida para la investigación:

- Analizar el estado de las herramientas disponibles y mayormente utilizadas.
- Adaptar las ideas a un marco económico y de infraestructura que refleje el estado actual del mercado local. Teniendo en cuenta el gran rango de poder adquisitivo que tienen los posibles usuarios junto con las variadas ofertas de infraestructura.
- Desarrollar la solución para ser adaptada a las necesidades de cada usuario sin ser finalmente una restricción más en el desarrollo del sistema final, sino una herramienta maleable y fácil de utilizar.

La tesina se organiza de la siguiente forma: El capítulo del *Marco Teórico* desarrolla brevemente conceptos necesarios para la comprensión general de la temática elegida. A continuación, el capítulo de *Estado del Arte*, describe alguna de las herramientas comerciales disponibles junto a sus ventajas y desventajas. Luego se detalla la *Solución Propuesta* en su propio capítulo en conjunto con el código necesario para implementarla. Sigue una *Evaluación de la Propuesta* que analiza y contrasta la efectividad de la solución del capítulo previo. Finalizando con la *Conclusión* que cierra este trabajo de tesina.

2. Marco Teórico

2.1 Sistemas distribuidos de Bases de Datos

En un *sistema distribuido de bases de datos*[1] se almacena la misma Base de Datos en varias computadoras utilizando distintos medios de comunicación para conectar los dispositivos que forman parte del sistema distribuido. Estos dispositivos no comparten ni memoria ni discos y pueden variar en sus características y función.

2.2 Bases de Datos distribuidas homogéneas y heterogéneas

Según Silberschatz, Korth y Sudarshan tenemos dos posibilidades de Bases de Datos distribuidas definidas por ellos a continuación[2]:

> *En las bases de datos distribuidas homogéneas todos los sitios tienen idéntico software de sistemas gestores de bases de datos, son conscientes de la existencia de los demás sitios y acuerdan cooperar en el procesamiento de las solicitudes de los usuarios. El software debe cooperar con los demás sitios en el intercambio de información sobre las transacciones.*

> *En las bases de datos distribuidas heterogéneas sitios diferentes puede que utilicen esquemas diferentes, diferente software de gestión de sistemas de bases de datos. Puede que unos sitios no sean conscientes de la existencia de los demás y puede que sólo proporcionen facilidades limitadas para la cooperación en el procesamiento de las transacciones.*

2.3 Réplica Completa de Base de Datos

La acción de realizar una réplica completa de Base de Datos se define como la copia electrónica idéntica de una Base de Datos localizada en un servidor hacia otro servidor u otra Base de Datos de forma periódica. La finalidad es obtener una Base de Datos

---

[1] Abraham Silberschatz; Henry F. Korth; S. Sudarshan, Fundamentos de Bases de Datos, Ed. McGraw-Hill/Interamericana de España, S.A.U., Madrid, 2002. Cuarta Edición. Página 455.

[2] Abraham Silberschatz; Henry F. Korth; S. Sudarshan, Fundamentos de Bases de Datos, Ed. McGraw-Hill/Interamericana de España, S.A.U., Madrid, 2002. Cuarta Edición. Página 463.

distribuida para que los usuarios de los distintos servidores puedan acceder a datos relevantes a sus tareas o para mantener una redundancia de los datos.

El problema básico[3] que existe al mantener una réplica de una Base de Datos es que cualquier operación sobre la Base de Datos debe ser propagada a la copia, y el dispositivo que mantiene la réplica puede no encontrarse disponible en el momento.

La réplica de una Base de Datos unidireccional se puede realizar de dos maneras, la primera por medio de la copia íntegra de la Base de Datos conocida como Réplica Snapshot de forma periódica. Y la segunda es una Réplica Transaccional donde se inicia con una copia completa de la Base de Datos y se la actualiza a medida que cambian los datos.

La Réplica Snapshot es útil cuando los datos cambian de forma poco frecuente, cuando ambos servidores no se encuentran sincronizados todo el tiempo, la Base de Datos tiene un tamaño pequeño, o cuando los datos cambian en grandes cantidades durante períodos de tiempo muy acotados. El tamaño de los datos replicados es directamente proporcional al tamaño de la Base de Datos con la que se trabaja.

La Réplica Transaccional se utiliza generalmente en tiempo real o períodos cortos de tiempo y los cambios en los datos son aplicados en la segunda Base de Datos en el mismo orden que en la original. Es preferible cuando se tiene un gran volumen de modificaciones sobre la Base de Datos original, y cuando la latencia entre servidores debe ser minimizada. El tamaño de los datos replicados es directamente proporcional a la cantidad de operaciones de manipulación realizadas en la Base de Datos en el período de tiempo elegido para la réplica.

[3] C.J. Date, Introducción a los Sistemas de Bases de Datos, Ed. Pearson Education, Méjico, 2001. Séptima Edición. Página 669.

2.4 Réplica Sincrónica / Asincrónica

La réplica sincrónica de Base de Datos tiene la ventaja de que la réplica creada se encuentra siempre y en todo momento actualizada. El problema que se presenta es que los DBM tradicionales como SQL Server no aceptan el cambio en la Base de Datos original hasta no recibir confirmación de la réplica. Esto, en función de la cantidad de operaciones y la velocidad de transporte de los datos, puede afectar el desempeño de la aplicación que utiliza la Base de Datos generando demoras. En general, es recomendable realizar réplicas sincrónicas con Data Centers a distancias menores a 50 km para mantener la latencia de la conexión lo más baja posible.

La réplica asincrónica elimina el problema de la latencia de la conexión de datos y no impacta negativamente la aplicación que utiliza la Base de Datos con la espera de una respuesta de la réplica. El problema de la réplica asincrónica en un DBM como SQL Server es que sin la conexión entre servidores se ve afectada por problemas de velocidad o se corta durante un período mayor al permitido, las Bases de Datos pierden su sincronización y no se puede recuperar.

2.5 Backup de Base de Datos

Se refiere a la copia y posterior archivo de una Base de Datos en un punto exacto del tiempo con la finalidad de tener un medio de recuperación en caso de una pérdida de datos por borrado o corrupción de la Base de Datos con la que se trabaja.

Un Backup requiere de una ventana de tiempo disponible para realizarlo, ya que genera una disminución en la performance del servidor y durante este período de tiempo no tendremos todos los recursos de software y hardware a nuestra disposición. También se ve limitado por el hardware del servidor, el dispositivo de almacenamiento y su conexión.

2.6 Diferencia entre Réplica de Base de Datos y Backup

La diferencia entre la réplica y el backup de base de datos es que el Backup se mantiene estático en el tiempo mientras que la réplica evoluciona y pierde cualquier dato histórico.

3. Estado del Arte

3.1 Réplica

3.1.1 SQL Server

SQL Server[4] posee un grupo de tecnologías destinadas a la copia y distribución de datos y objetos entre bases de datos, para luego sincronizarlas y mantener su coherencia. Esto es la Réplica de Base de Datos.

Podemos encontrar dos tipos principales de réplicas, una del tipo transaccional y una del tipo merge. Ambas enfocadas por Microsoft para distintos escenarios de uso. Siendo la réplica transaccional normalmente para uso de alto rendimiento entre servidores, y siendo la réplica merge principalmente para aplicaciones móviles o servidores distribuidos.

La ventaja de utilizar las herramientas provistas nativamente por SQL Server es que la réplica se encuentra de forma nativa en el software. Esta herramienta es confiable y soporta la interrupción de vínculo.

La primera desventaja que presenta es que la parametrización de la réplica se ve limitada por el desarrollador, lo que puede presentar problemas a futuro en caso de que el uso dado por el usuario evolucione más allá de lo considerado al implementar la solución. Además la falta de control sobre los recursos utilizados puede generar un conflicto al requerir su disponibilidad para otras tareas.

Una segunda desventaja es que el método que utiliza SQL Server para realizar las réplicas es crear “snapshots” de las tablas a ser replicadas, estos son una bajada a disco de los datos de cada tabla que se desea replicar. En consecuencia, en el momento de realizar la réplica como mínimo se duplica el espacio ocupado en disco. Esto puede convertirse en un

[4] Documentación de Microsoft SQL Server, disponible en https://msdn.microsoft.com/es-es/library/bb545450.aspx Consultado Mayo 2018.

problema serio al aumentar el tamaño de los archivos con el que se trabaja. Además, el resultado generado, ya sea uno o varios archivos, es comparado directamente con archivos en el destino. Dicha comparación se realiza con un proceso que requiere que los directorios sean compartidos a nivel NTFS con permisos de lectoescritura cruzados y lo realiza comparando línea a línea los archivos de texto plano. Siendo esto último un problema serio de seguridad.

Otro problema que presenta SQL Server es que frente a cortes de vínculo comienza a crear archivos adicionales temporales para generar información útil al momento de recuperar el vínculo. Esta solución para evitar la pérdida de datos genera una utilización de disco que crece con el tiempo. Además de esto, al recobrar el vínculo debe copiar todo sin importar el ancho de banda requerido y afectando negativamente la disponibilidad de conexión. A esto se suma que sí reconoce que el tiempo sin conexión es muy extenso para luego realizar las comparaciones, y con el fin de evitar errores, vuelve a arrancar desde un snapshot nuevo, utilizando muchos más recursos tanto en el cliente como en el servidor.

Es importante señalar que SQL Server recomienda utilizar la misma versión del software en ambos servidores ya que no todas las características son compatibles.

3.1.2 MySQL

MySQL[5] soporta la réplica asincrónica unidireccional, con un servidor como maestro y uno o más como esclavos. Para realizar la réplica, MySQL hace que el servidor maestro actualice un log binario y mantenga un índice de ficheros para rastrear los logs. Estos logs binarios son los que envía a los esclavos una vez que se conectan al maestro y, por medio de informar sobre la última actualización satisfactoria, “piden” los logs restantes para replicar los datos. Un esclavo puede ser al mismo tiempo maestro de otros servidores.

[5] Documentación de MySQL, disponible en https://dev.mysql.com/doc/ Consultado Mayo 2018.

La implementación unidireccional de la réplica presenta beneficios de robustez, velocidad y administración del sistema. Es robusto porque en caso de tener problemas con el maestro, se puede cambiar al esclavo de forma sencilla. Es más veloz porque se pueden separar las acciones que se realizan sobre cada servidor, siendo el maestro para comandos de creación o modificación y el esclavo para comandos de consulta sobre la tabla. En cuanto a administración del sistema, este esquema permite realizar copias de seguridad sobre el esclavo sin molestar al maestro.

El log binario del maestro es un registro que inicia una vez activada la opción, por lo que todos los esclavos deben tener una copia exacta de la base de datos al momento de iniciar el log, para de esta forma ejecutar las instrucciones paso a paso como las muestra el log binario sin generar una falla.

Una vez que la comunicación entre maestro y esclavo hace que el último tenga los datos de los logs binarios, se crea en el esclavo logs retardados para copiar las instrucciones generadas. Estos logs retardados se borran automáticamente una vez se terminan de utilizar.

Es importante resaltar que tanto el maestro como los esclavos deben tener instaladas versiones compatibles, o la misma, de MySQL para que todo funcione de forma correcta. La constante actualización de MySQL hace que esta compatibilidad sea mejor con cada iteración, pero la seguridad está en utilizar la misma versión en todos los servidores.

En cuanto a las consultas utilizadas, si se realizan consultas no deterministas, se puede obtener distintos resultados en maestro y en esclavo dado que queda a criterio del optimizador de consultas.

La falta de conexión entre maestro y esclavo genera que el esclavo se encuentre reintentando la conexión periódicamente.

La gran ventaja de MySQL es la capacidad de parametrización. Todas las opciones son personalizables a las necesidades del usuario, pero requieren de un usuario avanzado. Un cambio en los parámetros u omitir configurar un parámetro puede generar inestabilidad en la base de datos, por lo que se necesita asegurarse de que los cientos de parámetros posibles sean los correctos. Esta característica, mezclada con los errores que la comunidad soluciona semanalmente, puede terminar en problemas inaceptables para un usuario comercial.

3.1.3 SAP Sybase SQL Anywhere

SAP Sybase SQL Anywhere[6] es un administrador de base de datos que cuenta con Replication Server, una tecnología basada en conexiones para la réplica bidireccional de transacciones. Replication Server no está incluido en el paquete de SAP SQL Anywhere y debe ser adquirido por separado con la licencia de Log Transfer Management (LTM).

Esta tecnología permite usar procedimientos almacenados para realizar modificaciones en las tablas. Estos procedimientos ejecutan los comandos de actualización, inserción y eliminación.

Las bases de datos que necesiten utilizar Replication Server deben primero ser configuradas para dicho fin, ya que SAP SQL Anywhere no realiza la configuración automáticamente al crear una nueva base de datos. Esta configuración incluye, entre otras características, la selección de un usuario de mantenimiento y el nombre del servidor, y la configuración del idioma y el conjunto de caracteres que debe ser la misma en el servidor donde se realiza la réplica.

LTM admite replicar INSERT, DELETE y UPDATE y llamadas a procedimientos almacenados de dialectos Transact-SQL. El sistema replica únicamente los cambios

[6] Documentación de SAP Sybase SQL Anywhere disponible en http://dcx.sap.com/sa160/en/pdf/index.html Consultado Mayo 2018.

confirmados, lo que puede generar problemas de retraso, adicionales al retraso normal, cuando se realizan transacciones de gran longitud.

La tecnología de LTM acerca al usuario una amplia cantidad de opciones de configuración por medio de un archivo de configuración en texto plano. Este archivo tiene información tanto para el LTM como para el servidor SQL. Este archivo debe ser creado por el usuario previo a ejecutar el LTM.

El sistema permite la utilización de un búfer para los comandos a ser replicados para luego ser enviados en lotes al servidor de la réplica. Esto disminuye la cantidad de mensajes enviados y puede ayudar al rendimiento en general del sistema de réplica. El búfer tiene la capacidad de vaciarse con pérdidas una vez que se alcanza el máximo de comandos permitidos, o una vez que se queda sin memoria disponible. Por otro lado, se vacía correctamente al completar la transferencia al servidor de la réplica. Dada la capacidad del búfer de generar pérdidas del registro, SAP SQL Anywhere recomienda la realización periódica de back-ups para lo que tienen una herramienta a disposición del usuario.

Otra herramienta a disposición del usuario con el fin de mitigar errores es el espejo de registro de transacciones que se utiliza para guardar una copia idéntica del registro de transacciones en tiempo real. Esta herramienta es útil para detectar errores o para recuperarse completamente en caso de un fallo en el registro, pero presenta una penalización en el rendimiento de la base de datos ya que todo registro debe ser realizado dos veces. Esta última desventaja es directamente proporcional al volumen de tráfico de datos y a la configuración física del servidor.

SAP SQL Anywhere y su tecnología de Replication Server tiene algunas limitaciones que deben ser consideradas. En primer lugar requieren que la conexión entre servidores sea TCP/IP. Además la base de datos no debe tener eventos programados para evitar que estos generen diferencias entre los servidores. Y por último no puede replicarse desde un

servidor web, ya que al realizarse una conmutación por error se cambia la IP del servidor de la base de datos.

Una ventaja muy importante de este sistema es que permite el cifrado de los archivos a ser transportados .Permite AES para los archivos y RSA sobre la capa de transporte.

3.1.4 Oracle Advanced Replication

Las características Advanced Replication se encuentran totalmente integradas dentro de los servidores Oracle[7]. Dentro de sus opciones existen tres tipos de réplicas: Multimaster, De Vista Materializada y una híbrida entre las dos primeras.

La replicación Multimaster permite a varios servidores, actuando como pares iguales, administrar un grupo de objetos de base de datos replicados. Cada servidor es un maestro, y todos los maestros se comunican entre sí, pudiendo tener cada uno objetos replicados de la Base de Datos que a su vez pueden ser actualizados por otros maestros. El servidor Oracle se encarga de converger los datos de todas las réplicas automáticamente para garantizar la coherencia global de las transacciones y la integridad de los datos.

La replicación de Vista Materializada contiene una copia completa o parcial de un maestro de destino desde un único punto en el tiempo. El maestro de destino puede ser una tabla maestra en un sitio maestro o una vista maestra materializada en un sitio de vista materializada. Una vista materializada puede ser maestra si funciona como maestro de otra vista materializada, siendo la capa inferior una vista materializada de múltiples capas.

Las vistas materializadas tienen como beneficio que habilitan el acceso local a los datos, disminuyen las consultas al sitio maestro, y aumentan la seguridad de los datos al limitar

[7] Documentación de Oracle Advanced Replication disponible en https://docs.oracle.com/en/database/ Consultado Mayo 2018.

la cantidad de datos replicados del maestro. Además, es posible hacer que las vistas sean de tipo read, write y/o update.

Dadas las descripciones previas se entiende que en una réplica Multimaster las tablas replicadas son completas y actualizadas a medida que se realizan los cambios, mientras que las Vistas Materializadas pueden ser o no completas y reflejan una réplica en un punto de tiempo. Ambas se pueden combinar para obtener la réplica híbrida.

Tanto la replicación Multimaster como la de Vista Materializada presentan la posibilidad de conflictos de replicación al tener dos transacciones en sitios distintos que actualizan la misma fila al mismo tiempo. Oracle deja a cargo del usuario las reglas de resolución de estos conflictos entregando un conjunto de métodos pre construidos o la posibilidad de generar los propios.

3.2 Auditoría

3.2.1 ApexSQL Audit

ApexSQL Audit[8] es una herramienta de auditoría sobre SQL Server que permite capturar casi 200 eventos distintos y generar alertas en tiempo real. Toda la información auditada se encuentra en una Base de Datos central con control de integridad de datos.

La gran cantidad de filtros normales y avanzados que se le entregan al usuario vía una interfaz gráfica permite seleccionar que eventos de secuencias DDL o DML quiere auditar de cada tabla. De esta forma haciendo posible una auditoría adaptada a cada necesidad.

El sistema ofrece una encriptación SHA-256 sobre los datos archivados para aumentar la seguridad de los datos y evitar que sean realizadas modificaciones posteriores a su

[8] Documentación de ApexSQL Audit disponible en https://knowledgebase.apexsql.com/apexsql-audit-knowledgebase-table-contents/ Consultado Junio 2018.

captura. Dada esta característica, los datos y los informes generados a partir de estos solo pueden ser accedidos desde el software instalado.

Las versiones compatibles de ApexSQL Audit para SQL Server comienzan en la 2005 y de Windows a partir de Vista o Server 2008. Es necesario tener permisos de administrador para realizar la instalación del software en cualquiera de los casos.

La instalación de ApexSQL Audit requiere de 5 GB de espacio en disco solo para su instancia central que debe ser instalada junto a una versión compatible de SQL Server. Cada instancia distribuida que se conecta requiere de 1 GB adicional.

La documentación disponible online es limitada como también lo es su comunidad de usuarios. Por otro lado, su costo es superior a los mil dólares.

3.2.2 SQL Server Change Data Capture (CDC)

SQL Server cuenta con una característica a partir de su versión 2008 para la realización de auditorías que se llama Change Data Capture (CDC)[9] que registra toda la actividad de inserción, actualización o borrado que se realiza sobre una tabla de SQL Server. El resultado es un detalle de las modificaciones disponible en un formato relacional. Cabe destacar que CDC no se encuentra activado por defecto en SQL Server 2008 o mayor, sino que debe ser explícitamente activado para cada Base de Datos.

CDC utiliza el log de transacciones de SQL Server para capturar los datos que necesita. A medida que se aplican las modificaciones, se generan en el log nuevas entradas que las describen. CDC lee estas entradas en el log para generar la tabla de cambios asociada.

La tabla de cambios asociada tiene en primer lugar cinco columnas con metadatos que proveen información adicional relevante para los cambios generados. El resto de las columnas restantes son una copia en nombre y, generalmente, en tipo de las columnas de

[9] Documentación de SQL Server Change Data Capture (CDC) disponible en https://docs.microsoft.com/en-us/sql/relational-databases/track-changes/about-change-data-capture-sql-server/ Consultado Junio 2018.

la tabla sobre la que se están auditando los cambios. En estas columnas restantes se capturan la información.

Cada operación INSERT o DELETE sobre la tabla original genera una fila en la tabla de cambios, mientras que un UPDATE requiere dos filas para cubrir el valor previo y el posterior a la operación.

El tipo de auditoría de CDC es asincrónico, leyendo por lotes el log y generando luego la tabla de cambios. Este tipo de práctica que no es configurable por el usuario puede ser una ventaja o una desventaja dependiendo de la situación donde se encuentra aplicada la tecnología.

Previamente se mencionó que CDC funciona sobre SQL Server 2008 en adelante, pero es necesario aclarar que se necesita la versión Enterprise del software, hasta SQL Server 2016 que lo tiene por defecto en todas sus versiones.

3.2.3 IDERA SQL Compliance Manager

IDERA SQL Compliance Manager[10] soporta la auditoría de eventos en Bases de Datos sobre SQL Server o sobre Servidores de Bases de Datos Oracle. Puede capturar los eventos resultantes de inserciones, actualizaciones o borrados sobre tablas. En SQL Server tiene la posibilidad de tomar los datos desde eventos rastreables, eventos extendidos o logs de auditoría dependiendo de la versión de SQL Server sobre la que se está trabajando.

Permite la captura y auditoría de datos antes y después de una modificación en cualquier tabla, junto a la capacidad de generar alertas basadas en eventos, estados o columnas preseleccionadas.

El software garantiza que los datos generados no pueden ser alterados por un tercero, además de generar notificaciones frente a actividad sospechosa por mail o log de eventos.

10 Documentación de IDERA SQL Compliance Manager disponible en https://idera.com/productssolutions/sqlserver/sqlcompliancemanager/overview/ Consultado Junio 2018.

La compatibilidad del software se limita a Windows 7 SP1 o mayor, y Windows Server 2008 o mayor. Esto se debe a la tecnología utilizada para minimizar el impacto sobre los recursos de los servidores donde se trabaja.

Tiene un costo elevado comparado con otras opciones del mercado y no cuenta ni con una gran cantidad de documentación disponible online, ni con una gran comunidad de usuarios para resolver y analizar distintos problemas.

3.3 Conclusión del Estado del Arte

Basándonos en todas las opciones actuales del mercado, siendo las más populares las mencionadas previamente en este capítulo, se debe generar una discusión sobre la nueva herramienta que brinde la solución necesaria a nuestro problema.

Antes de analizar las opciones dadas sus ventajas y desventajas, el punto más importante es el de la implementación actual en el mundo real. Dado que una de las metas es no encarecer el costo final del producto, es necesaria una mirada rápida al escenario donde podría implementarse comercialmente la herramienta. Esta mirada es muy simple ya que entre los posibles clientes futuros hay un claro ganador en motores de bases de datos y es SQL Server.

Partiendo de la base de que se trabajará sobre SQL Server debemos considerar ahora las posibilidades para réplica y auditoría. En primer lugar tenemos la réplica nativa de SQL Server que presenta muchos problemas si uno no se mantiene actualizado siempre a la última versión. Este no es el caso común en nuestro escenario de potenciales clientes que trabajan sobre sistemas operativos desactualizados que no soportan las últimas versiones, junto a hardware funcional pero no óptimo para la tarea. Además, la forma de trabajo de la réplica nativa de SQL Server requiere una conexión estable con un buen ancho de banda disponible, y nuevamente no es algo que se pueda asegurar en nuestro escenario.

Por otro lado tenemos la auditoría de base de datos, para la cual no nos sirven las soluciones del mercado por falta de compatibilidad con versiones de SQL Server o Windows todavía en uso, o también por el elevado costo de la herramienta.

A partir de lo antes mencionado se propone crear una solución que tenga en cuenta para la réplica un alto nivel de parametrización, un bajo costo de uso de espacio en el disco sin contar la o las tablas a ser replicadas, una alta tolerancia a los cortes de conexión y compatibilidad entre distintas versiones de SQL Server. En cuanto a la auditoría de base de datos debe cubrir todas las actualizaciones posibles sobre una tabla, incluyendo los valores antes y después de los cambios, tener los metadatos necesarios para poder conocer qué, quién y cuándo se realizaron las actualizaciones, y todo sin olvidar que debe tener tolerancia a los cortes de conexión. En base a todo lo mencionado se considera la utilización de triggers para disminuir tanto el uso de hardware como la necesidad de un software externo. Otro punto a favor que tiene la utilización de triggers para la solución, es que toda la solución se desarrolla dentro de SQL Server por lo que el software utilizado por el usuario final es independiente y no necesita ser manipulado.

4. Solución Propuesta

4.1 Introducción

A partir de lo mencionado en el capítulo 3.3, sumado a la experiencia que conlleva el uso prolongado de algunas de las herramientas existentes, se consideró realizar un trigger con la capacidad de iniciar, controlar y mantener la réplica de la tabla original como también la de llenar la tabla de auditoría para ser ésta utilizada cuando sea necesario. En el servidor de destino se tiene un procedimiento almacenado para la ejecución de la réplica.

4.2 Base de Datos de Origen

La Base de Datos de Origen puede contener un número indefinido de tablas igual o superior a uno, sobre las cuales serán agregadas dos tablas y una copia del trigger por cada tabla sobre la que se desea operar. De ahora en adelante las tablas que tengan un trigger serán llamadas tablas de origen.

Las tablas de origen tienen una restricción en cuanto al tipo de datos, ya que el trigger no opera sobre datos del tipo image, text y ntext.

Las dos tablas que se agregan corresponden una a la tarea de Auditoría y otra a la tarea de Réplica. Ambas serán explicadas a continuación.

Dada la necesidad de que la Base de Datos de Origen sea modificada por el procedimiento almacenado en la Base de Datos de Destino, se debe configurar de forma manual la opción de Linked Server para generar la relación de confianza unidireccional entre los servidores.

4.3 Tabla de Auditoría

La tabla de Auditoría propuesta realiza la tarea de guardar en cada fila los datos relevantes a las modificaciones realizadas sobre cada campo al ejecutar un comando SQL Insert, Update o Delete sobre la o las tablas de origen. Cabe destacar que independientemente de la cantidad de tablas de origen en la Base de Datos, la tabla de Auditoría es única.

Las distintas operaciones pueden generar distintas cantidades de filas en la tabla Auditoría. Por ejemplo si se realiza un Update sobre un solo valor de una tabla de origen, se generará en la tabla Auditoría una sola fila, mientras que si se genera un Insert o un Delete se generarán tantas filas como columnas existan en una tabla de origen.

Otra de las capacidades de la tabla de Auditoría es la de reconocer cada sentencia independiente o bloque de sentencias en una transacción con un ID único, a fin de poder realizar una trazabilidad completa de cada modificación.

El fin de dicha tabla es poseer un historial detallado sobre los valores anteriores y posteriores a las modificaciones realizadas, especificando de forma detallada la tabla, las filas y las columnas a la que pertenece cada campo. También incluye quién, cómo y cuándo se realizó cada modificación. Una vez que tenemos todos estos datos, es posible volver atrás las modificaciones sobre cada tabla de origen de forma independiente.

4.3.1 Código SQL para la Tabla de Auditoría

```
CREATE TABLE [dbo].[Auditoria](
    [RowID] [numeric](18,0) IDENTITY(1,1) NOT NULL,
    [Tipo] [char] (1) NULL,
    [NombreTabla] [varchar] (128) NULL,
    [PK] [varchar] (1000) NULL,
    [NombreColumna] [varchar] (128) NULL,
    [ValorAnterior] [varchar] (1000) NULL,
    [ValorActual] [varchar] (1000) NULL,
    [FechaHora] [varchar] (21) NULL,
    [DBMSTransaction] [varchar] (255) NULL,
    [UsuarioOS] [varchar] (50) NULL,
    [UsuarioMotor] [varchar] (50) NULL,
    [NombreAplicacion] [varchar] (100) NULL,
    [NombreEquipo] [varchar] (50) NULL,
    [MacAddress] [varchar] (17) NULL,
    [IP] [varchar] (30) NULL,

CONSTRAINT [PK_Auditoria] PRIMARY KEY CLUSTERED
(
```

```
    [RowID] ASC
) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, FILLFACTOR = 90) ON [PRIMARY]
) ON [PRIMARY]
GO
```

4.4 Tabla de Réplica SQL

La tabla de Réplica SQL está diseñada para que cada trigger guarde en cada una de sus filas, los comandos SQL Insert, Updated y Delete que son ejecutados sobre la o las tablas de origen. De la misma forma que la tabla de Auditoría, la tabla de Réplica SQL es única independientemente de la cantidad de tablas de origen.

A diferencia de la tabla Auditoría, la tabla Réplica SQL solo guarda una fila por cada comando ejecutado sobre las tablas de origen.

La tabla de Réplica SQL incluye un campo que indica si la fila fue o no replicada por el procedimiento almacenado en la tabla de réplica. Lo cual permite no ejecutar dos veces la misma sentencia en el destino.

La finalidad de esta tabla es la de tener la capacidad de replicar cada uno de los comandos SQL, en el mismo orden que fueron ejecutados originalmente, en la tabla de réplica correspondiente a cada tabla de origen que se encuentra en la Base de Datos de destino.

4.4.1 Código SQL para la tabla de Réplica SQL

```
CREATE TABLE [dbo].[Replica_SQL](
    [id] [bigint] IDENTITY(1,1) NOT NULL,
    [fechahora] [datetime] NULL,
    [tipoevento] [varchar] (100) NULL,
    [parametros] [int] NULL,
    [sql] [nvarchar] (max) NULL,
    [procesado] [int] NULL,
    [fechahoraproc] [datetime] NULL,

CONSTRAINT [PK_auditoria_sql] PRIMARY KEY CLUSTERED
```

```
(
    [id] ASC
) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

ALTER TABLE [dbo].[Replica_SQL] ADD DEFAULT (getdate()) FOR [fechahora]
GO

ALTER TABLE [dbo].[ Replica_SQL] ADD DEFAULT ((0)) FOR [procesado]
GO
```

4.5 Trigger

El Trigger opera sobre la tabla especificada en su código y utiliza como destino las tablas agregadas de Auditoría y Réplica SQL. Este se activa al realizar una modificación sobre la tabla de origen al que corresponde.

Su tarea en primer lugar es capturar la sentencia SQL que lo invocó para guardarla en la tabla Réplica SQL en una nueva fila. Su segunda tarea es descomponer la sentencia para identificar la modificación realizada y crear las filas correspondientes en la tabla de Auditoría.

En su primera tarea, el Trigger, no realiza cambio alguno sobre la sentencia SQL. Simplemente toma la orden de modificación recibida y la guarda para ser utilizada posteriormente en una fila nueva de la tabla Réplica SQL.

En su segunda tarea, el Trigger, analiza la modificación generada y la descompone en qué se está modificando, quién lo está haciendo y desde dónde lo está haciendo. El “qué” incluye si es un Insert, Delete o Update junto con su valor anterior y valor actual del campo modificado. El “quién” incluye el usuario del motor o del sistema operativo. Y el “dónde” incluye el nombre del equipo, la IP, la MacAddress y el programa que realizó los cambios.

La utilización de este Trigger facilita la automatización de la copia de datos necesarios para luego realizar la réplica y genera en su totalidad la tabla de Auditoría. Su ventaja es que todo el proceso se realiza de forma local e inmediatamente.

Una limitación del Trigger en cuanto a la secuencia SQL en versiones anteriores al 2014 de SQL Server es que debe ser menor o igual a 4.000 caracteres de longitud ya que es la limitación del buffer del motor de base de datos DBCC INPUTBUFFER.

En el código siguiente se utiliza el Trigger sobre una tabla llamada “Tabla 1”.

4.5.1 Código SQL para el Trigger

```
ALTER trigger [dbo].[tr_Tabla_1] on [dbo].[Tabla_1] for insert, update, delete
AS

DECLARE @bit int,
   @field int,
   @maxfield int,
   @char int,
   @fieldname varchar(128),
   @PKCols varchar(1000),
   @sql varchar(2000),
   @PKSelect varchar(1000),
   @Tipo char(1),
   @NombreTabla varchar(128),
   @PK varchar(1000),
   @NombreColumna varchar(128),
   @ValorAnterior varchar(1000),
   @ValorActual varchar(1000),
   @FechaHora varchar(21),
   @DBMSTransaction varchar(255),
   @UsuarioOS varchar(50)
   @UsuarioMotor varchar(50),
   @NombreAplicacion varchar(100),
   @NombreEquipo varchar(50),
   @MacAdress varchar(17),
   @IP varchar(30),
   @Contexto varchar(128),
```

```
Set NOCOUNT ON

IF CAST(SUBSTRING(@@VERSION,21,5) as int) < 2014 [11]
  INSERT INTO Replica_SQL (tipoevento, parametros, sql) EXEC('DBCC INPUTBUFFER
          (@@spid)').
ELSE
  INSERT INTO Replica_SQL (sql) (SELECT ib.event_info FROM sys.dm_exec_sessions AS es CROSS
          APPLY sys.dm_exec_input_buffer(es.session_id, NULL) AS ib WHERE es.session_id
          = @@spid)

IF @@trancount>0
  EXECUTE sp_getbindtoken @DBMSTransaction OUTPUT
ELSE
  SET @DBMSTransaction = ''

IF PatIndex ( '%\%' ,SUSER_SNAME()) > 0
  BEGIN
    SET @UsuarioOS = SUSER_SNAME()
    SET @UsuarioMotor = ''
  END
ELSE
  BEGIN
    SET @UsuarioOS = ''
    SET @UsuarioMotor = SUSER_SNAME()
  END

SET @MacAdress = (SELECT net_address FROM master..sysprocesses WHERE
    spid=@@spid)
SET @MacAdress = substring (@MacAdress,1,2) + '-' + substring (@MacAdress,3,2) + '-' +
    substring (@MacAdress,5,2) + '-' + substring (@MacAdress,7,2) + '-' + substring
    (@MacAdress,9,2) + '-' + substring (@MacAdress,11,2)

SET @NombreAplicacion = (SELECT program_name FROM master..sysprocesses WHERE
    spid=@@spid)

SET @NombreEquipo = (SELECT hostname FROM master..sysprocesses WHERE
    spid=@@spid)
```

[11] El condicional se aplica para evitar que el Buffer se llene en versiones anteriores a SQL Server 2014.

```
SELECT @contexto = CAST(context_info AS varchar(128)) FROM master..sysprocesses
    WHERE spid=@@spid.

IF (PatIndex( '%LOGICAL_ADDRESS=%', @contexto) <> '') and (PatIndex(
    '%LOGICAL_ADDRESS=%', @contexto) > 0)
  SET @IP = substring (ltrim (substring (@contexto, PatIndex ( '%LOGICAL_ADDRESS=%',
      @contexto)+16,128)),1, charIndex( '///', ltrim(substring (@contexto, PatIndex (
      '%LOGICAL_ADDRESS=%', @contexto)+16,128)))-1)
ELSE
  SET @IP = ''

SELECT @NombreTabla = 'Tabla_1'

SELECT @FechaHora = convert(varchar(8), getdate(), 112) + ' ' + convert(varchar(12),
       getdate(), 114)
SET @IP = ''

IF EXISTS (SELECT * FROM inserted)
  IF EXISTS (SELECT * FROM deleted)
    SELECT @Tipo = 'U'
  ELSE
    SELECT @Tipo = 'I'
ELSE
  SELECT @Tipo = 'D'

SELECT * INTO #ins FROM inserted
SELECT * INTO #del FROM deleted

SELECT @PKCols = coalesce(@PKCols + ' and', ' on') + ' i.' + c.COLUMN_NAME + ' = d.' +
       c.COLUMN_NAME
FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS pk,
       INFORMATION_SCHEMA.KEY_COLUMN_USAGE c
WHERE pk.TABLE_NAME = @NombreTabla
AND CONSTRAINT_TYPE = 'PRIMARY KEY'
AND c.TABLE_NAME = pk.TABLE_NAME
AND c.CONSTRAINT_NAME = pk.CONSTRAINT_NAME

SELECT @PKSelect = coalesce(@PKSelect + '+', '') + ''''<' + COLUMN_NAME + '=' '+convert
       (varchar(100),coalesce(i.' + COLUMN_NAME + ',d.' + COLUMN_NAME + '))+''>'''
FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS pk
       INFORMATION_SCHEMA.KEY_COLUMN_USAGE c
```

```
WHERE pk.TABLE_NAME = @NombreTabla
AND CONSTRAINT_TYPE = 'PRIMARY KEY'
AND c.TABLE_NAME = pk.TABLE_NAME
AND c.CONSTRAINT_NAME = pk.CONSTRAINT_NAME

IF @PKCols IS NULL
BEGIN
  raiserror('No tiene PK la tabla %s', 16, -1, @NombreTabla)
  return
END

SELECT @field = 0, @maxfield = max(ORDINAL_POSITION) FROM
      INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = @NombreTabla
WHILE @field < @maxfield
BEGIN
  SELECT @field = min (ORDINAL_POSITION) FROM INFORMATION_SCHEMA.COLUMNS
  WHERE TABLE_NAME = @NombreTabla AND ORDINAL_POSITION > @field
  SELECT @bit = (@field – 1)% 8+1
  SELECT @bit = power(2,@bit - 1)
  SELECT @char = ((@field – 1) / 8) + 1
  IF substring (COLUMNS_UPDATED(), @char, 1) & @bit > 0 OR @Tipo in ('I','D')
  BEGIN
    SELECT @fieldname = COLUMN_NAME from INFORMATION_SCHEMA.COLUMNS
          WHERE TABLE_NAME = @NombreTabla AND ORDINAL_POSITION = @field
    SELECT @sql = 'insert Auditoria (Tipo ,NombreTabla ,PK ,NombreColumna
          ,ValorAnterior ,ValorActual ,FechaHora ,DBMSTransaction ,UsuarioOS
          ,UsuarioMotor ,NombreAplicacion ,NombreEquipo ,MacAdress ,IP )'
    SELECT @sql = @sql + ' select ''' + @Tipo + '''
    SELECT @sql = @sql + ',''' + @NombreTabla + '''
    SELECT @sql = @sql + ',' + @PKSelect
    SELECT @sql = @sql + ',''' + @fieldname + '''
    SELECT @sql = @sql + ',convert(varchar(1000),d.' + @fieldname + ')'
    SELECT @sql = @sql + ',convert(varchar(1000),i.' + @fieldname + ')'
    SELECT @sql = @sql + ',''' + @FechaHora + '''
    SELECT @sql = @sql + ',''' + @DBMSTransaction + '''
    SELECT @sql = @sql + ',''' + @UsuarioOS + '''
    SELECT @sql = @sql + ',''' + @UsuarioMotor + '''
    SELECT @sql = @sql + ',''' + @NombreAplicacion + '''
    SELECT @sql = @sql + ',''' + @NombreEquipo + '''
    SELECT @sql = @sql + ',''' + @MacAdress + '''
    SELECT @sql = @sql + ',''' + @IP + '''
```

```
      SELECT @sql = @sql + ' FROM #ins i FULL OUTER JOIN #del d'
      SELECT @sql = @sql + @PKCols
      SELECT @sql = @sql + ' WHERE i.' + @fieldname + ' <> d.' + @fieldname
      SELECT @sql = @sql + ' OR (i.' + @fieldname + ' is null and d.' + @fieldname + ' is not
                  null)'
      SELECT @sql = @sql + ' OR (i.' + @fieldname + ' is not null and d.' + @fieldname + ' is
                  null)'
      EXEC (@sql)
   END
END
```

4.6 Base de Datos de Destino

Dentro de la Base de Datos de Destino debemos tener creadas las tablas que vamos a replicar con la misma estructura con sus tipos de datos, y los índices que corresponden. A estas tablas las mencionaremos en adelante como tablas de destino. Las tablas de destino deben tener el mismo nombre que las tablas de origen de las que son replicadas.

Junto con las tablas de destino tendremos un procedimiento almacenado que se encarga de tomar los campos de la tabla Réplica SQL de la Base de Datos de Origen y ejecutarlos como secuencias SQL en la Base de Datos de Destino para replicar cada tabla de origen en el mismo orden que fue modificada.

4.7 Procedimiento Almacenado

El Procedimiento Almacenado no se ejecuta nunca por sí mismo sino que debe ser invocado de forma manual o de forma automática con una tarea de programación en el motor de la Base de Datos de Destino. La tarea de programación incluye una especificación de fecha y hora en la que debe ser ejecutado el procedimiento almacenado. Es posible configurarlo de forma recurrente y en un rango horario específico.

En primer lugar, el procedimiento almacenado, inicia una transacción y carga en un cursor todas las secuencias no procesadas de la tabla Réplica SQL de la Base de Datos de Origen. Una vez que tiene las secuencias SQL las ejecuta dentro de una transacción

compartida entre ambos motores de base de datos operando sobre la tabla de destino correspondiente y la de Réplica SQL. Cada vez que ejecuta una secuencia SQL en una tabla de destino vuelve a la tabla de Réplica SQL y modifica el campo “procesado” para registrarla como correcta junto con su fecha y hora de proceso. Una vez ejecutada la secuencia correctamente, y actualizada la tabla de Réplica SQL, genera un COMMIT y se mueve a la sentencia siguiente.

En caso de estar corriendo el proceso y no poder conectarse a la Base de Datos de Origen para generar el cambio en la tabla de Réplica SQL, se realiza un rollback sobre la última sentencia ejecutada, luego envía un correo con el error y finaliza el proceso hasta ser ejecutado nuevamente.

En caso de que la conexión no sea posible al ejecutar el procedimiento almacenado, envía un correo electrónico de error previo a cerrarse.

4.7.1 Código SQL para el Procedimiento Almacenado

```
ALTER PROCEDURE [dbo].[SP_Replica_SQL]
AS
BEGIN
   SET NOCONUNT ON;
   SET XACT_ABORT ON;

   DECLARE @ErrorCode int
   DECLARE @ErrorStep varchar(200)
   DECLARE @sql nvarchar(max)
   DECLARE @id bigint
   DECLARE @fechahora nvarchar(50)
   DECLARE @Return_Message varchar (1024)
   DECLARE @ParameterDefinition nvarchar(4000)

   SELECT @ParameterDefinition = ‘ @idpar bigint, @fechahorapar DATETIME ‘;
   SELECT @ErrorCode = @@ERROR

BEGIN TRY
   BEGIN TRAN
```

```
    SELECT @ErrorStep = 'Declaro Cursor '
    DECLARE CUR_Replica_Origen CURSOR FOR
      Select id, fechahora, sql
      FROM Replica_Origen_dbo.Replica_SQL aud
      WHERE aud.procesado = 0
      ORDER by aud.fechahora;

      SELECT @ErrorStep = 'Abro Cursor '

      OPEN CUR_Replica_Origen;

      SELECT @ErrorStep = 'Traigo Datos Cursor '
      FETCH NEXT FROM CUR_Replica_Origen INTO @id, @fechahora, @sql

      WHILE @@FETCH_STATUS = 0
      BEGIN

        SELECT @ErrorStep = 'Actualizando Replica Destino ';
        PRINT @sql
        EXEC sp_executesql @sql
        SELECT @ErrorStep = 'Update tabla Replica_SQL en Replica_Origen';
        SELECT @fechahora = CONVERT (nvarchar(30), GETDATE(), 121)
        SET @sql = 'UPDATE Replica_Origen.dbo.Replica_SQL set procesado = 1,
              fechahoraproc = @fechahorapar where id = @idpar'
        PRINT @sql
        EXEC sp_executesql @sql, @ParameterDefinition, @fechahorapar = @fechahora,
              @idpar = @id;
        FETCH NEXT FROM CUR_Replica_Origen INTO @id, @fechahora, @sql
      END

    CLOSE CUR_Replica_Origen;
    DEALLOCATE CUR_Replica_Origen;

  COMMIT TRAN
  SELECT @ErrorCode = 0; @Return_Message = 'Se actualizaron los datos en
          Replica_Destino' + CAST(GETDATE() as varchar(25))
  RETURN @ErrorCode
  SET XACT_ABORT OFF;

END TRY
```

```
BEGIN CATCH

   CLOSE CUR_Replica_Origen;
   DEALLOCATE CUR_Replica_Origen;

   IF @@TRANCOUNT > 0 ROLLBACK

   SELECT @ErrorCode = @@ERROR
   SELECT @ErrorCode = ERROR_NUMBER()
      , @Return_Message = @ErrorStep + ' – Fila Replica_SQL: '
      + CAST(@id as varchar(20)) + ' - NUMBER: '
      + CAST(ERROR_NUMBER() as varchar(20)) + ' - SEVERITY: '
      + CAST(ERROR_SEVERITY() as varchar(20)) + ' - STATE: '
      + CAST(ERROR_STATE() as varchar(20)) + ' - LINEA: '
      + CAST(ERROR_LINE() as varchar(20)) + ' - MENSAJE: '
      + ERROR_MESSAGE() + ' > SQL: ' + @sql
      + ERROR_PROCEDURE()

   EXEC msdb.dbo.sp_send_dbmail
      @recipients= 'contacto@tesis.com.ar',
      @subject = 'Error Replica',
      @body = @Return_Message,
      @body_format = 'HTML',
      @profile_name= 'SQLDBMailTesis'

   BEGIN TRAN
      INSERT INTO Replica_SQL_LOG (fechahora,error) VALUES (GETDATE(),
                                 @Return_Message)
   COMMIT TRAN
   SET XACT_ABORT OFF;
   RETURN @ErrorCode -- = 0 OK, <> 0 Error

END CATCH
END
GO
```

4.8 Log de errores

En el código SQL del procedimiento Almacenado puede notarse la utilización de una tabla llamada Replica_SQL_LOG que se utiliza a fin de guardar los errores que se

produzcan al ejecutar el proceso. La tabla es simple y su código de creación puede verse a continuación.

4.8.1 Código SQL para el Log de errores

```
CREATE TABLE [dbo].[Replica_SQL_LOG](
		[id] [bigint] IDENTITY (1,1) NOT NULL,
		[fechahora] [datetime] NULL,
		[error] [nvarchar](max) NULL,
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

5. Evaluación de la Propuesta

En la introducción de esta tesina se describen los problemas planteados que desean solucionarse con la propuesta previa, dentro de las necesidades se encontraba la idea de simplificar las herramientas existentes y mencionadas en el capítulo de Estado del Arte, considerar una solución que se adapte a software y hardware desactualizado que se presenta en la infraestructura de sistemas de las empresas locales que se consideran como posibles futuros clientes, y por último tener en cuenta que la infraestructura de conexión de internet local no es óptima fuera de lugares puntuales en grandes ciudades.

A partir de los problemas planteados, la propuesta se adapta simplificando la aplicación de la herramienta más utilizada, SQL Server, sin limitarse en cuanto a la versión o versiones más actuales, lo que hace posible que se adapte a Software y Hardware desactualizado, junto a requerir poco ancho de banda desde el diseño de la solución y la capacidad de configurar de acuerdo a las capacidades de cada usuario el tiempo y momento de la comunicación.

5.1 Problemas de la solución

La principal limitación de la solución descripta en el capítulo anterior, es una que se encuentra por la utilización de triggers en SQL Server ya que estos no soportan los formatos ntext, text e image pero la solución es cambiarlos por nvarachar(max), varchar(max) y varbinary(max) respectivamente. Esta limitación debe tenerse en cuenta previo a la implementación, y deben modificarse las tablas para no tener errores. Las modificaciones, dado que los nuevos formatos aceptan los anteriores, no afectan el funcionamiento del software utilizado por el cliente.

Una segunda limitación es que en versiones de SQL Server previas a la 2014 el buffer que guarda las sentencias SQL que se ingresan tenía una limitación de 4.000 caracteres, descartando las secuencias más largas.

Un posible problema del diseño actual es que el proceso almacenado ejecuta las secuencias SQL en orden del primero al último, sin considerar las transacciones realizadas en origen. De esta forma podemos tener inconsistencia en los datos si se corta la comunicación antes de finalizar la tarea, aunque se soluciona una vez la comunicación es restablecida y las sentencias SQL son ejecutadas.

Aunque no es un problema de la solución, es necesario tener en cuenta que debe configurarse de forma manual en la Base de Datos de Destino cada cuánto tiempo se desea realizar el chequeo de nuevos datos para replicar, y cada cuánto tiempo se debe volver a intentar si se encuentra un error.

5.2 Caso de aplicación

La solución propuesta se aplicó en un caso puntual con un cliente que tiene una casa central en el centro de Rosario y cuatro sucursales en Rosario y localidades aledañas. El cliente requiere la réplica de los cinco puntos de trabajo a un data center externo a fin de que todas las consultas se realicen sobre este servidor y para disponer de una conexión más confiable, y una capacidad de procesamiento mayor.

Las Bases de Datos son aproximadamente de 15 GB de tamaño en cada una de las sucursales, y de unos 100 GB de tamaño en la casa central. La casa central cuenta con una conexión a internet óptima para la tarea, pero las sucursales dependen de cooperativas de los pueblos o ciudades donde se encuentran y no son ni confiables, ni rápidas. Las versiones de SQL Server con las que se trabajan son 2014 en la casa central y en las sucursales encontramos versiones 2008 y 2005.

La solución propuesta en esta tesina se aplicó configurando las réplicas de forma que se realicen cada una hora, comenzando una hora antes del inicio de la jornada laboral diaria y terminando una hora después de la finalización. Frente a errores se configuró la solución para volver a intentar conectarse quince minutos luego de la falla e informar del error vía mail.

6. Conclusión

A partir de la Evaluación de la propuesta podemos concluir que el sistema no solo funciona de forma óptima sino que se encuentra en instancias de ser comercializado considerando su eficiencia y sus costos de desarrollo e implementación bajos. Lo que cumple con nuestras ideas al iniciar el proyecto. Aun así es posible plantear mejoras a futuro para la herramienta desarrollada.

Una posible primera mejora es simplificar la herramienta utilizando únicamente la tabla Auditoría y descartando la tabla Réplica SQL. Esto es posible, y teóricamente simple de implementar, pero fue una idea que llegó una vez finalizado el desarrollo y con la herramienta funcionando.

Otra mejora es la de también replicar la tabla de auditoría al destino, que en este momento se encuentra solo en el origen. La razón para mantenerla en el origen es que los datos de auditoría solo necesitan ser accedidos de forma eventual.

Finalmente existe la posibilidad de solucionar las posibles inconsistencias de la réplica agregando una columna a Réplica SQL con el código DBMS_Transaction y forzando que el proceso almacenado ejecute las secuencias SQL por lotes en caso de que sea lo correcto.

Concluimos mencionando que nuestra solución debe ser considerada una vez que el cliente cuenta con una implementación profesional de sistemas, y esta debería incluir una conexión por VPN a su servidor externo para evitar la posibilidad de que se filtren sus datos. De esta forma no es necesario cifrar la información enviada al replicar.

7. Bibliografía

- Abraham Silberschatz; Henry F. Korth; S. Sudarshan, Fundamentos de Bases de Datos, Ed. McGraw-Hill/Interamericana de España, S.A.U., Madrid, 2002. Cuarta Edición.
- C.J. Date, Introducción a los Sistemas de Bases de Datos, Ed. Pearson Education, Méjico, 2001. Séptima Edición.
- Documentación de Microsoft SQL Server, disponible en https://msdn.microsoft.com/es-es/library/bb545450.aspx Consultado Mayo 2018.
- Documentación de MySQL, disponible en https://dev.mysql.com/doc/ Consultado Mayo 2018.
- Documentación de SAP Sybase SQL Anywhere disponible en http://dcx.sap.com/sa160/en/pdf/index.html Consultado Mayo 2018.
- Documentación de Oracle Advanced Replication disponible en https://docs.oracle.com/en/database/ Consultado Mayo 2018.
- Documentación de ApexSQL Audit disponible en https://knowledgebase.apexsql.com/apexsql-audit-knowledgebase-table-contents/ Consultado Junio 2018.
- Documentación de SQL Server Change Data Capture (CDC) disponible en https://docs.microsoft.com/en-us/sql/relational-databases/track-changes/about-change-data-capture-sql-server/ Consultado Junio 2018.
- Documentación de IDERA SQL Compliance Manager disponible en https://idera.com/productssolutions/sqlserver/sqlcompliancemanager/overview/ Consultado Junio 2018.