Replicación de Datos en SQL Server

Introducción

La replicación de datos permite que ciertos datos de la base de datos sean almacenados en más de un sitio, y su principal utilidad es que permite aumentar la disponibilidad de los datos y mejora el funcionamiento de las consultas globales a la base de datos.

La replicación en SQL Server consiste, en el transporte de datos entre dos o más instancias de servidores. Para ello SQL Server brinda un conjunto de soluciones que permite copiar, distribuir y posiblemente modificar datos de toda la organización. Se incluyen, además, varios métodos y opciones para el diseño, implementación, supervisión y administración de la replicación, que le ofrecen la funcionalidad y flexibilidad necesarias para distribuir datos y mantener su coherencia

En la replicación se utiliza una metáfora de la industria de la publicación para representar los componentes y procesos de una topología de replicación. De esta forma el modelo se compone, básicamente, de los siguientes elementos: publicador, distribuidor, suscriptores, publicaciones, artículos y suscripciones

Componentes del modelo de replicación

Para representar los componentes y procesos de una topología de replicación se utilizan metáforas de la industria de la publicación. El modelo se compone de los siguientes objetos: el publicador, el distribuidor, el suscriptor, la publicación, el artículo y la suscripción; así como de varios agentes, que son los procesos responsabilizados de copiar los datos entre el publicador y el suscriptor.

La replicación de datos es un asunto exclusivamente entre servidores de datos, en nuestro caso hablamos de servidores SQL Server. Los servidores SQL Server pueden desempeñar uno o varios de los siguientes roles: publicador, distribuidor o suscriptor.

El publicador es un servidor que pone los datos a disposición de otros servidores para poder replicarlos. **El distribuidor** es un servidor que aloja la base de datos de distribución y almacena los datos históricos, transacciones y metadatos. **Los suscriptores** reciben los datos replicados.

Una publicación es un conjunto de artículos (este concepto: "artículo de una publicación", es diferente del concepto "artículo o registro de una base de datos", como

explicaremos más adelante) de una base de datos. Esta agrupación de varios artículos facilita especificar un conjunto de datos relacionados lógicamente y los objetos de bases de datos que desea replicar conjuntamente. Un artículo de una publicación puede ser una tabla de datos la cual puede contar con todas las filas o algunas (filtrado horizontal) y simultáneamente contar de todas las columnas o algunas (filtrado vertical), un procedimiento almacenado, una definición de vista, la ejecución de un procedimiento almacenado, una vista, una vista indizada o una función definida por el usuario.

Una suscripción es una petición de copia de datos o de objetos de base de datos para replicar. Una suscripción define qué publicación se recibirá, dónde y cuándo. Las suscripciones pueden ser de inserción o de extracción; y una publicación puede admitir una combinación de suscripciones de inserción y extracción. El publicador (en las suscripciones de inserción) o el suscriptor (en las suscripciones de extracción) solicitan la sincronización o distribución de datos de una suscripción.

El publicador puede disponer de una o más publicaciones, de las cuales los suscriptores se suscriben a las publicaciones que necesitan, nunca a artículos individuales de una publicación. El publicador, además, detecta qué datos han cambiado durante la replicación transaccional y mantiene información acerca de todas las publicaciones del sitio.

La función del distribuidor varía según la metodología de replicación implementada. En ocasiones se configura como distribuidor el mismo publicador y se le denomina distribuidor local. En el resto de los casos el distribuidor será remoto, pudiendo coincidir en algún caso con un suscriptor.

Los suscriptores además de obtener sus suscripciones, en dependencia del tipo y opciones de replicación elegidas, pueden devolver datos modificados al publicador. Además puede tener sus propias publicaciones

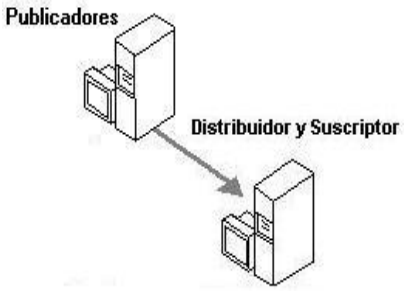
Escenarios típicos de la replicación

En una solución de replicación pudiera ser necesario utilizar varias publicaciones en una combinación de metodologías y opciones. En la replicación los datos o transacciones fluyen del publicador al suscriptor pasando por el distribuidor.

Por lo tanto en su configuración mínima una topología de replicación se compone de al menos dos o tres servidores SQL Server que desempeñan los tres roles mencionados.

Variando la ubicación del servidor distribuidor podríamos contar con las siguientes variantes:

1. El rol de distribuidor desempeñado por el publicador (Fig. 1.1).
2. El rol de distribuidor desempeñado por el suscriptor (Fig. 1.2)
3. Un servidor de distribución, independiente del publicador y del suscriptor (Fig. 1.3)



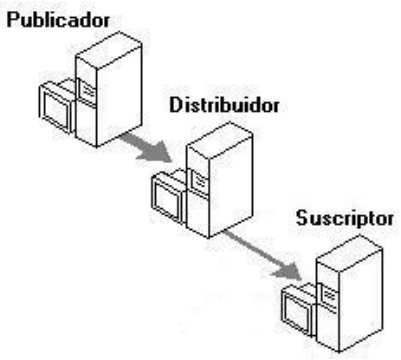


Fig.1 Publicador-Distribuidor Fig.2 Distribuidor-Suscriptor Fig. 3 Distribuidor independiente.

En la mayoría de las configuraciones, el peso fundamental de la replicación recae, sobre el servidor de distribución. Por tanto éste puede ser un criterio para determinar su ubicación, teniendo en cuenta las configuraciones (posibilidades físicas) de los servidores, así como otras responsabilidades que pueden estar desempeñando (servidor de dominio, servidor de páginas web entre otras.

Existe la posibilidad de contar con un servidor que se suscriba a una publicación y a la vez la publique para el resto de los suscriptores, esto puede ser muy útil cuando se cuente con una conexión muy costosa con el publicador principal. Por ejemplo el publicador principal en Madrid y los suscriptores en Ciudad Habana, Varadero, Cayo Coco, Cayo Largo, etc. En casos como este, se puede elegir un suscriptor, digamos el servidor de Ciudad Habana el cual se suscribe al publicador en Madrid y a la vez actúa como servidor de publicación para los servidores de Varadero, Cayo Coco, Cayo Largo y demás. Evidentemente en una configuración tal pueden nuevamente combinarse la ubicación de los dos distribuidores y aumentar el número de variantes que pueden presentarse pero las consideraciones para determinar la ubicación del servidor que fungirá como distribuidor son las ya mencionadas.

Tipos de replicación

Los tipos básicos de replicación son:

- replicación de instantáneas
- replicación transaccional
- replicación de mezcla

Para ajustarse aún más a los requerimientos de los usuarios se incorporan opciones como son la actualización inmediata en el suscriptor, la actualización en cola y la transformación de datos replicados

Replicación de instantáneas

En la replicación de instantáneas los datos se copian tal y como aparecen exactamente en un momento determinado. Por consiguiente, no requiere un control continuo de los cambios. Las publicaciones de instantáneas se suelen replicar con menos frecuencia que otros tipos de publicaciones.

Puede llevar más tiempo propagar las modificaciones de datos a los suscriptores. Se recomienda utilizar: cuando la mayoría de los datos no cambian con frecuencia; se replican pequeñas cantidades de datos; los sitios con frecuencia están desconectados y es aceptable un periodo de latencia largo (la cantidad de tiempo que transcurre entre la actualización de los datos en un sitio y en otro

Replicación transaccional

En este caso se propaga una instantánea inicial de datos a los suscriptores, y después, cuando se efectúan las modificaciones en el publicador, las transacciones individuales se propagan a los suscriptores. SQL Server 2000 almacena las transacciones que afectan a los objetos replicados y propaga esos cambios a los suscriptores de forma continua o a intervalos programados. Al finalizar la propagación de los cambios, todos los suscriptores tendrán los mismos valores que el publicador.

Replicación de mezcla

Permite que varios sitios funcionen en línea o desconectados de manera autónoma, y mezclar más adelante las modificaciones de datos realizadas en un resultado único y uniforme. La instantánea inicial se aplica a los suscriptores; a continuación SQL Server 2000 hace un seguimiento de los cambios realizados en los datos publicados en el

publicador y en los suscriptores. Los datos se sincronizan entre los servidores a una hora programada o a petición.

Fases generales para implementar y supervisar la replicación

A pesar de que existen varias formas de implementar y supervisar la replicación, y el proceso de replicación es diferente según el tipo y las opciones elegidas, en general, la replicación se compone de las siguientes fases:

- configuración de la replicación
- generación y aplicación de la instantánea inicial
- modificación de los datos replicados
- sincronización y propagación de los datos.

Transacciones

Una *transacción* es una colección de acciones que hacen transformaciones consistentes de los estados de un sistema preservando la consistencia del sistema. Una base de datos está en un estado *consistente* si obedece todas las restricciones de integridad definidas sobre ella. Los cambios de estado ocurren debido a actualizaciones, inserciones, y supresiones de información. Por supuesto, se quiere asegurar que la base de datos nunca entra en un estado de inconsistencia. Sin embargo, durante la ejecución de una transacción, la base de datos puede estar temporalmente en un estado inconsistente. El punto importante aquí es asegurar que la base de datos regresa a un estado consistente al fin de la ejecución de una transacción (Ver Figura 5.1)

Lo que se persigue con el manejo de transacciones es por un lado tener una transparencia adecuada de las acciones concurrentes a una base de datos y por otro lado tener una transparencia adecuada en el manejo de las fallas que se pueden presentar en una base de datos.

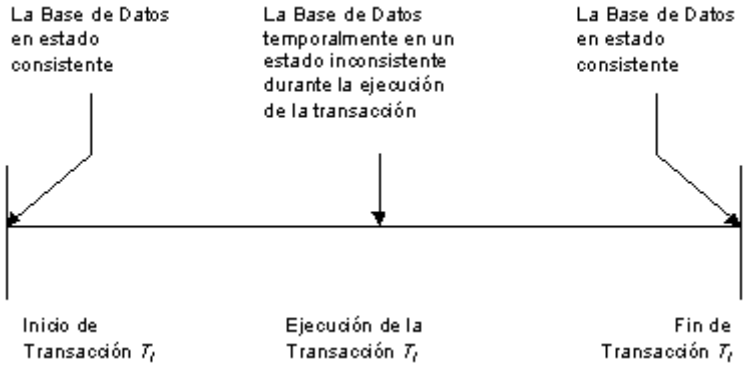


Figura 5.1. Un modelo de transacción.

Ejemplo 5.1. Considere la siguiente consulta en SQL para incrementar el 10% del presupuesto del proyecto CAD/CAM de la base de datos de ejemplo.

UPDATE J

SET BUDGET = BUDGET*1.1

WHERE JNAME = "CAD/CAM"

Esta consulta puede ser especificada, usando la notación de SQL, como una transacción otorgándole un nombre:

Begin_transaction ACTUALIZA_PRESUPUESTO
begin
UPDATE J
SET BUDGET = BUDGET*1.1
WHERE JNAME = "CAD/CAM"
end.

Condiciones de terminación de una transacción

Una transacción siempre termina, aun en la presencia de fallas. Si una transacción termina de manera exitosa se dice que la transacción hace un *commit* (se usará el término en inglés cuando no exista un término en español que refleje con brevedad el sentido del término en inglés). Si la transacción se detiene sin terminar su tarea, se dice que la transacción *aborta*. Cuando la transacción es abortada, su ejecución es detenida y todas sus acciones ejecutadas hasta el momento son deshechas (*undone*) regresando a la base de datos al estado antes de su ejecución. A esta operación también se le conoce como *rollback*.

Caracterización de transacciones

Observe que en el ejemplo anterior las transacciones leen y escriben datos. Estas acciones se utilizan como base para caracterizar a las transacciones. Los elementos de datos que lee una transacción se dice que constituyen el *conjunto de lectura* (*RS*). Similarmente, los elementos de datos que una transacción escribe se les denomina el *conjunto de escritura* (*WS*). Note que los conjuntos de lectura y escritura no tienen que ser necesariamente disjuntos. La unión de ambos conjuntos se le conoce como el *conjunto base* de la transacción ($BS = RS \cup WS$).

Formalización del concepto de transacción

Sea $O_{ij}(x)$ una operación O_j de la transacción T_i la cual trabaja sobre alguna entidad x. $O_j \in \{read, write\}$ y O_j es una operación atómica, esto es, se ejecuta como una unidad indivisible. Se denota por $OS_i = \cup_j O_{ij}$ al conjunto de todas las operaciones de la transacción T_i. También, se denota por N_i la condición de terminación para T_i, donde, $N_i \in \{abort, commit\}$.

La transacción T_i es un orden parcial, $T_i = \{ S_i, <_i \}$, donde

1. $S_i = OS_i \cup \{N_i\}$
2. Para cualesquiera dos operaciones, O_{ij}, $O_{ik} \in OS_i$, si $O_{ij} = R(x)$ y $O_{ik} = W(x)$ para cualquier elemento de datos x, entonces, ó $O_{ij} <_i O_{ik}$ ó $O_{ik} <_i O_{ij}$
3. $\forall O_{ij} \in OS_i, O_{ij} <_i N_i$

Propiedades de las transacciones

La discusión en la sección previa clarifica el concepto de transacción. Sin embargo, aun no se ha dado ninguna justificación para afirmar que las transacciones son unidades de procesamiento consistentes y confiables. Las propiedades de una transacción son las siguientes:

1. ***Atomicidad***. Se refiere al hecho de que una transacción se trata como una unidad de operación. Por lo tanto, o todas las acciones de la transacción se realizan o ninguna de ellas se lleva a cabo. La atomicidad requiere que si una transacción se interrumpe por una falla, sus resultados parciales deben ser deshechos. La

actividad referente a preservar la atomicidad de transacciones en presencia de abortos debido a errores de entrada, sobrecarga del sistema o interbloqueos se le llama *recuperación de transacciones*. La actividad de asegurar la atomicidad en presencia de caídas del sistema se le llama *recuperación de caídas*.

2. ***Consistencia***. La consistencia de una transacción es simplemente su correctitud. En otras palabras, una transacción es un programa correcto que lleva la base de datos de un estado consistente a otro con la misma característica. Debido a esto, las transacciones no violan las restricciones de integridad de una base de datos.
3. ***Aislamiento***. Una transacción en ejecución no puede revelar sus resultados a otras transacciones concurrentes antes de su *commit*. Más aún, si varias transacciones se ejecutan concurrentemente, los resultados deben ser los mismos que si ellas se hubieran ejecutado de manera secuencial (seriabilidad).
4. ***Durabilidad***. Es la propiedad de las transacciones que asegura que una vez que una transacción hace su *commit*, sus resultados son permanentes y no pueden ser borrados de la base de datos. Por lo tanto, los DBMS aseguran que los resultados de una transacción sobrevivirán a fallas del sistema. Esta propiedad motiva el aspecto de *recuperación de bases de datos*, el cual trata sobre como recuperar la base de datos a un estado consistente en donde todas las acciones que han hecho un commit queden reflejadas.

En resumen, las transacciones proporcionan una ejecución atómica y confiable en presencia de fallas, una ejecución correcta en presencia de accesos de usuario múltiples y un manejo correcto de réplicas (en el caso de que se soporten).

Tipos de Transacciones

Las transacciones pueden pertenecer a varias clases. Aun cuando los problemas fundamentales son los mismos para las diferentes clases, los algoritmos y técnicas que se usan para tratarlas pueden ser considerablemente diferentes. Las transacciones pueden ser agrupadas a lo largo de las siguientes dimensiones:

1. **Áreas de aplicación**. En primer lugar, las transacciones se pueden ejecutar en aplicaciones no distribuidas. Las transacciones que operan en datos distribuidos se les conoce como transacciones distribuidas. Por otro lado, dado que los resultados de una transacción que realiza un commit son durables, la única

forma de deshacer los efectos de una transacción con commit es mediante otra transacción. A este tipo de transacciones se les conoce como transacciones *compensatorias*. Finalmente, en ambientes heterogéneos se presentan transacciones *heterogéneas* sobre los datos.

2. **Tiempo de duración**. Tomando en cuenta el tiempo que transcurre desde que se inicia una transacción hasta que se realiza un commit o se aborta, las transacciones pueden ser de tipo batch o en línea. Estas se pueden diferencias también como transacciones de corta y larga vida. Las transacciones en línea se caracterizan por tiempos de respuesta muy cortos y por accesar un porción relativamente pequeña de la base de datos. Por otro lado, las transacciones de tipo batch toman tiempos relativamente largos y accesan grandes porciones de la base de datos.
3. **Estructura**. Considerando la estructura que puede tener una transacción se examinan dos aspectos: si una transacción puede contener a su vez subtransacciones o el orden de las acciones de lectura y escritura dentro de una transacción.

Aspectos relacionados al procesamiento de transacciones

Los siguientes son los aspectos más importantes relacionados con el procesamiento de transacciones:

1. Modelo de estructura de transacciones. Es importante considerar si las transacciones son planas o pueden estar anidadas.
2. Consistencia de la base de datos interna. Los algoritmos de control de datos semántico tienen que satisfacer siempre las restricciones de integridad cuando una transacción pretende hacer un commit.
3. Protocolos de confiabilidad. En transacciones distribuidas es necesario introducir medios de comunicación entre los diferentes nodos de una red para garantizar la atomicidad y durabilidad de las transacciones. Así también, se requieren protocolos para la recuperación local y para efectuar los compromisos (commit) globales.
4. Algoritmos de control de concurrencia. Los algoritmos de control de concurrencia deben sincronizar la ejecución de transacciones concurrentes bajo

el criterio de correctitud. La consistencia entre transacciones se garantiza mediante el aislamiento de las mismas.

5. Protocolos de control de réplicas. El control de réplicas se refiere a cómo garantizar la consistencia mutua de datos replicados. Por ejemplo se puede seguir la estrategia read-one-write-all (ROWA).

Un tercer componente que participa en el manejo de transacciones distribuidas es el *administrador de recuperación local* cuya función es implementar procedimientos locales que le permitan a una base de datos local recuperarse a un estado consistente después de una falla.

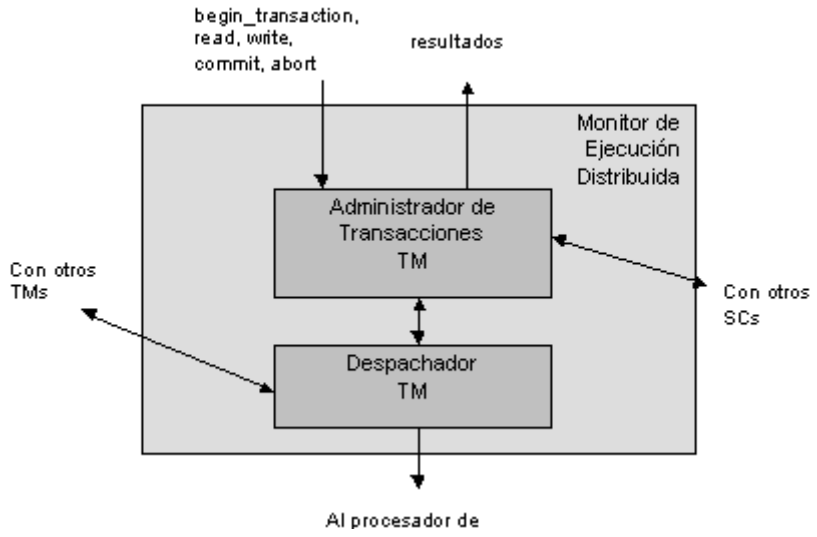


Figura 5.2. Un modelo del administrador de transacciones.

Los administradores de transacciones implementan una interfaz para los programas de aplicación que consiste de los comandos:

1. **Begin_transaction**.
2. **Read**.
3. **Write**.
4. **Commit**.
5. **Abort**.

En la Figura 5.3 se presenta la arquitectura requerida para la ejecución centralizada de transacciones. Las modificaciones requeridas en la arquitectura para una ejecución distribuida se pueden apreciar en las Figura 5.4. En esta última figura se presentan

también los protocolos de comunicación necesarios para el manejo de transacciones distribuidas.

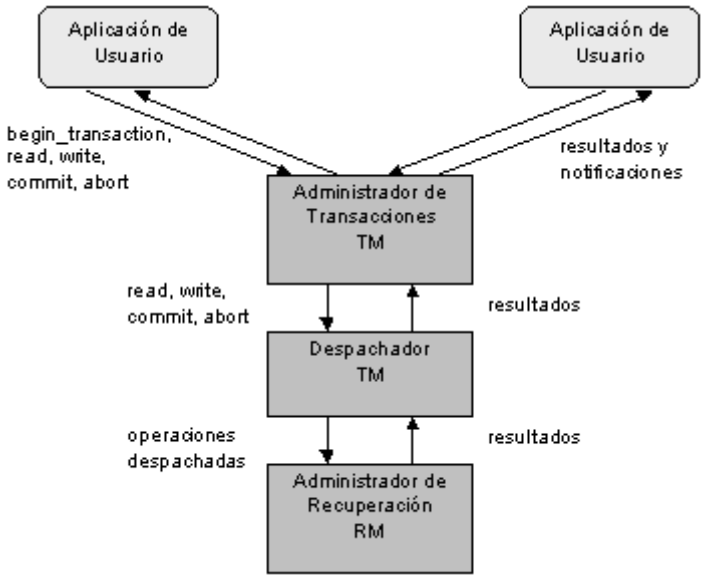


Figura 5.3. Ejecución centralizada de transacciones.

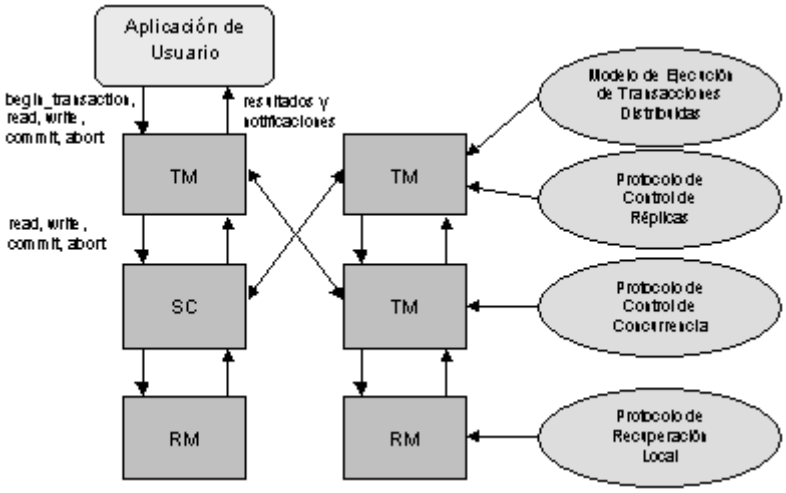


Figura 5.4. Ejecución distribuida de transacciones.

Data Warehouse

Que es un Data WareHouse?

Es un repositorio de datos de muy fácil acceso, alimentado de numerosas fuentes, transformadas en grupos de información sobre temas específicos de negocios, para permitir nuevas consultas, análisis, reporteador y decisiones.

Los objetivos fundamentales de un Data WareHouse son:

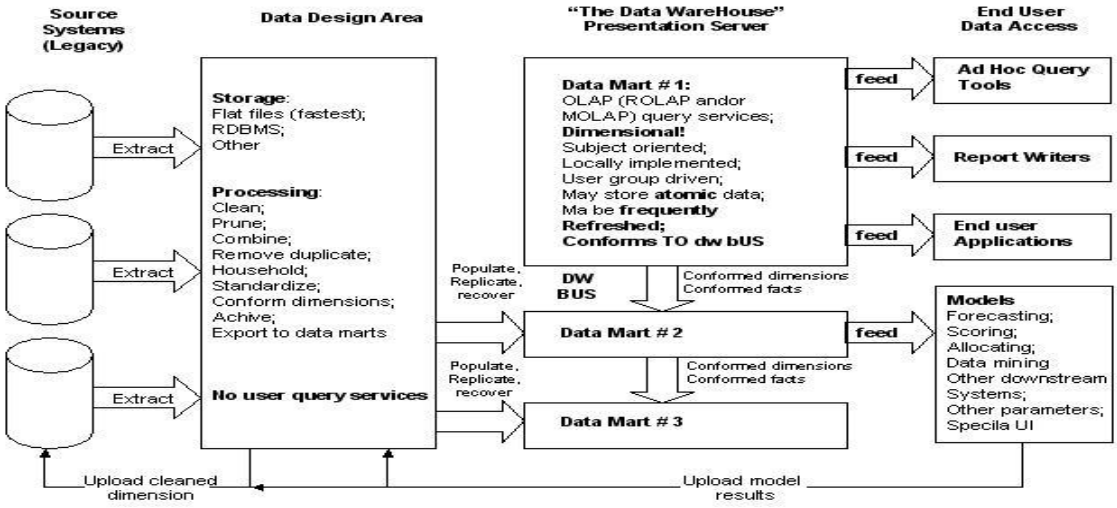
- Hace que la información de la organización sea accesible: los contenidos del Data WareHouse son entendibles y navegables, y el acceso a ellos son caracterizado por el rápido desempeño. Estos requerimientos no tienen fronteras y tampoco limites fijos. Cuando hablamos de entendible significa, que los niveles de la información sean correctos y obvios. Y Navegables significa el reconocer el destino en la pantalla y llegar a donde queramos con solo un clic. Rápido desempeño significa, cero tiempo de espera. Todo lo demás es un compromiso y por consiguiente algo que queremos mejorar.
- Hacer que la información de la organización sea consistente: la información de una parte de la organización puede hacerse coincidir con la información de la otra parte de la organización. Si dos medidas de la organización tienen el mismo nombre, entonces deben significar la misma cosa. Y a la inversa, si dos medidas no significan la misma cosa, entonces son etiquetados diferentes. Información consistente significa, información de alta calidad. Significa que toda la información es contabilizada y completada. Todo lo demás es un compromiso y por consiguiente algo que queremos mejorar.
- Es información adaptable y elástica: el Data WareHouse está diseñado para cambios continuos. Cuando se le hacen nuevas preguntas al Data WareHouse, los datos existentes y las tecnologías no cambian ni se corrompen. Cuando se agregan datos nuevos al Data WareHouse, los datos existentes y las tecnologías tampoco cambian ni se corrompen. El diseño de Data Marts separados que hacen al Data WareHouse, deben ser distribuidos e incrementados. Todo lo demás es un compromiso y por consiguiente algo que queremos mejorar.
- Es un seguro baluarte que protege los valores de la información: el Data WareHouse no solamente controla el acceso efectivo a los datos, si no que da a los dueños de la información gran visibilidad en el uso y abusos de los datos, aún después de haber dejado el Data WareHouse. Todo lo demás es un compromiso y por consiguiente algo que queremos mejorar.
- Es la fundación de la toma de decisiones: el Data WareHouse tiene los datos correctos para soportar la toma de decisiones. Solo hay una salida verdadera del Data WareHouse: las decisiones que son hechas después de que el Data WareHouse haya presentado las evidencias. La original etiqueta que preside el Data WareHouse sigue

siendo la mejor descripción de lo que queremos construir: un sistema de soporte a las decisiones.

Los elementos básicos de un Data WareHouse

- Sistema fuente: sistemas operacionales de registros donde sus funciones son capturar las transacciones del negocio. A los sistemas fuentes también se le conoce como Legacy System.
- Área de tráfico de datos: es un área de almacenamiento y grupo de procesos, que limpian transforman, combinan, remover los duplicados, guardan, archivan y preparan los datos fuente para ser usados en el Data WareHouse.
- Servidor de presentación: la maquina física objetivo en donde los datos del Data WareHouse son organizados y almacenados para queries directos por los usuarios finales, reportes y otras aplicaciones.
- Modelo dimensional: una disciplina específica para el modelado de datos que es una alternativa para los modelos de entidad – relación.
- Procesos de negocios: un coherente grupo de actividades de negocio que hacen sentido a los usuarios del negocio del Data WareHouse.
- Data Mart: un subgrupo lógico del Data WareHouse completo.
- Data WareHouse: búsquedas fuentes de datos de la empresa. Y es la unión de todos los data marts que la constituyen.
- Almacenamiento operacional de datos: es el punto de integración por los sistemas operacionales. Es el acceso al soporte de decisiones por los ejecutivos.
- OLAP: actividad general de búsquedas para presentación de texto y números del Data WareHouse, también un estilo dimensional especifico de búsquedas y presentación de información y que es ejemplificada por vendedores de OLAP.
- ROLAP: un grupo de interfaces de usuarios y aplicaciones que le dan a la base de datos relacional un estilo dimensional.
- MOLAP: un grupo de interfaces de usuarios, aplicaciones y propietarios de tecnología de bases de datos que tienen un fuerte estilo dimensional.
- Aplicaciones para usuarios finales: una colección de herramientas que hacen los queries, analizan y presentan la información objetivo para el soporte de las necesidades del negocio.
- Herramientas de acceso a datos por usuarios finales: un cliente de Data WareHouse.

- Ad Hoc Query Tool: un tipo especifico de herramientas de acceso a datos por usuarios finales que invita al usuario a formas sus propios queries manipulando directamente las tablas relacionales y sus uniones.
- Modelado de aplicaciones: un sofisticado tipo de cliente de Data WareHouse con capacidades analíticas que transforma o digiere las salidas del Data WareHouse.
- Meta Data: toda la información en el ambiente del Data WareHouse que no son así mismo los datos actuales.



Los procesos básicos del Data WareHouse (ETL)

- Extracción: este es el primer paso de obtener la información hacia el ambiente del Data WareHouse.
- Transformación: una vez que la información es extraída hacia el área de tráfico de datos, hay posibles paso de transformación como; limpieza de la información, tirar la basura que no nos sirve, seleccionar únicamente los campos necesarios para el Data WareHouse, combinar fuentes de datos, haciéndolas coincidir por los valores de las llaves, creando nuevas llaves para cada registro de una dimensión.
- Carga: al final del proceso de transformación, los datos están en forma para ser cargados.

Las razones básicas de porque una organización implementa Data WareHouse:

Para realizar tareas en los servidores y discos, asociados a queries y reportes en servidores y discos que no son utilizados por sistemas de proceso de transacciones.

Muchas de las empresas quieren instalar sistemas de procesos de transacciones para que haya una alta probabilidad de que las transacciones sean completadas en un tiempo razonable. Estos sistemas de procesos de transacciones hacen que las transacciones y peticiones sean más rápidas en menores tiempos dado a que los queries y reportes consumen mucho más de su límite permitido en los recursos de servidores y discos, por tal motivo las empresas han implementado una arquitectura de Data WareHouse que utiliza sus servidores y discos por separado para algunos de los queries y reportes.

Para utilizar modelos de datos o tecnologías de servidores que agilizan los queries y reportes, y que no son apropiados para los procesos de transacciones.

Existen maneras de modelar los datos que usualmente agilizan los queries y reportes (ejemplo: el esquema del modelo estrella) y que no son apropiados para los procesos de transacciones porque la técnica de modelado bajaría el rendimiento y complicaría el proceso de transacciones. También existen tecnologías que aceleran el proceso de queries y reportes pero baja la velocidad en el proceso de transacciones (ejemplo: la indexación de bitmaps) y tecnología de servidores que incrementan la velocidad en el proceso de transacciones, pero que disminuyen la velocidad del proceso de queries y reportes (ejemplo: La tecnología de recuperación de transacciones). Todo esto entonces esta en el cómo se hacen los modelos de datos y que tecnología se utiliza, inclusive que productos se adquieren para el impacto de los procesos de queries y reportes.

Para proveer un ambiente donde relativamente una muy poca cantidad de conocimiento de los aspectos técnicos de tecnología de bases de datos es requerida para escribir y mantener queries y reportes.

Frecuentemente un Data WareHouse puede ser instalado de manera que los queries y reportes puedan ser escritos por personal sin tanto conocimiento técnico, lo que hace que su mantenimiento y construcción se haga sin más complejidad.

Para proveer un repositorio del sistema de proceso de transacciones limpio que puede ser reportado y que no necesariamente requiere que se arregle el sistema de proceso de transacciones.

El Data WareHouse provee la oportunidad de limpiar los datos sin cambiar los sistemas de proceso de transacciones, sin embargo algunas implementaciones de Data WareHouse provee el significado para capturar las correcciones hechas a los datos del Data WareHouse y alimenta las correcciones hacia el sistema de proceso de

transacciones. Muchas veces hace más sentido hacer las correcciones de esta manera que aplicar las correcciones directamente al sistema de proceso de transacciones.

Para hacer los queries y reportes de datos básicamente más fácil de los múltiples procesos de transacciones y de las fuentes externas y de los datos que deben ser almacenados solamente para el propósito de hacer queries y reportes.

Desde hace mucho tiempo que las compañías necesitan reportes con información de múltiples sistemas y han hecho extracciones de datos para después correrlos bajo la lógica de búsqueda combinando la información de las extracciones con los reportes generados, lo que en muchas ocasiones es una buena estrategia. Pero cuando se tienen muchos datos y las búsquedas se vuelven muy pesadas y después limpiar la búsqueda, entonces lo apropiado sería un Data WareHouse.

Cubos de Datos

Los usuarios de los sistemas de apoyo a las decisiones a menudo ven los datos en forma de ***cubos de datos.*** El cubo se utiliza para representar los datos a lo largo de un cierto grado de interés. Aunque se llama un "cubo", puede ser de 2 dimensiones, 3 dimensiones, o más dimensiones. Cada dimensión representa algún atributo en la base de datos y las celdas en el cubo de datos representan la medida de su interés. Por ejemplo, podría contener un contador para el número de veces que se produce la combinación de atributos en la base de datos, o el mínimo, máximo, suma o valor medio de algún atributo. Las consultas se llevan a cabo en el cubo para recuperar información de soporte de decisiones.

Ejemplo: Tenemos una base de datos que contiene información sobre las operaciones relativas ventas de la compañía de una parte a un cliente en una tienda. El cubo de datos formado a partir de esta base de datos es una representación 3-dimensional, con cada célula *(p, c, s)* del cubo que representa una combinación de valores de una *parte, los clientes* y *ubicación de la tienda*-. Un cubo de datos de ejemplo de esta combinación se muestra en la Figura 1. El contenido de cada celda es el recuento del número de veces que un conjunto específico de valores se produce conjuntamente en la base de datos. Las células que aparecen en blanco, de hecho, tienen un valor de cero. El cubo se puede utilizar para recuperar información en la base de datos sobre, por ejemplo, que almacenan se debe dar una cierta parte para vender a fin de que la mayor venta.

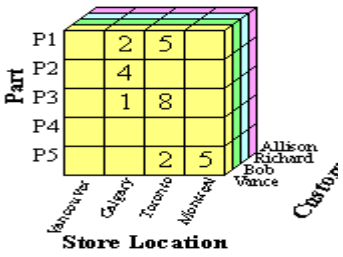


Figura 1 (a): Vista de frente de Muestra datos del cubo

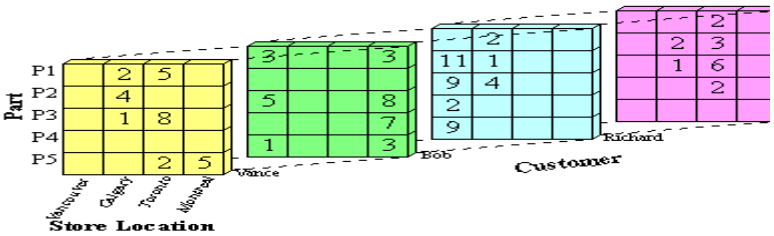


Figura 1 (b): Ver todo el de Muestra datos del cubo

Los cubos son elementos claves en OLAP (online analytic processing), una tecnología que provee rápido acceso a datos en un almacén de datos (data warehouse). Los cubos proveen un mecanismo para buscar datos con rapidez y tiempo de respuesta uniforme independientemente de la cantidad de datos en el cubo o la complejidad del procedimiento de búsqueda.

Los cubos son subconjuntos de datos de un almacén de datos, organizado y sumarizado dentro de una estructura multidimensional. Los datos se sumarian de acuerdo a factores de negocio seleccionados, proveyendo el mecanismo para la rápida y uniforme tiempo de respuesta de las complejas consultas.

La definición del cubo, es el primero de tres pasos en la creación de un cubo. Los otros pasos son, el especificar la estrategia de sumarización diseñando las agregaciones (elementos precalculados de datos), y la carga del cubo para procesarlo. Para definir un cubo, seleccione una tabla objetivo y seleccione las medidas (columnas numéricas de interés a los usuarios del cubo) dentro de esta tabla. Entonces seleccione las dimensiones, cada compuesta de una o más columnas de otra tabla. Las dimensiones proveen la descripción categórica por el cual las medidas son separadas para su análisis por los usuarios del cubo.

Dimensiones

Las Dimensiones son categorías descriptivas por los cuales los datos numéricos (Las Mediciones) en un cubo, son separados para su análisis. Por ejemplo, si una medición de un cubo es el conteo de la producción, y las dimensiones son Tiempo, localización de la fábrica y el producto, los usuarios del cubo, podrán separar el conteo de la

producción, dentro de varias categorías de tiempo, localización de la fábrica y productos.

Una dimensión puede ser creada para usarse en un cubo individual o en múltiples cubos. Una dimensión creada para un cubo individual, es llamada dimensión privada. Por el contrario si esta puede ser usada por múltiples cubos, se le llama dimensión compartida.

Estas podrán ser usadas dentro de todo cubo, en la base de datos, así se optimiza el tiempo y se evita el andar duplicando dimensiones privadas.

Las dimensiones compartidas, también habilitan la estandarización de las métricas de negocios entre cubos. Por ejemplo, el estandarizar las dimensiones compartidas para el tiempo y localización geográfica, aseguran que los datos analizados, desde diferentes cubos, estén organizados similarmente.

Medidas

Las Medidas, son datos numéricos de interés primario para los usuarios del cubo. Algunas medidas comunes son Ventas en unidades, ventasen pesos, costo de ventas, gastos, conteo de la producción, presupuesto, etc.

Estas son usadas por el procedimiento de agregación de los servicios de OLAP y almacenadas para su rápida respuesta a las peticiones de los usuarios.

Se puede crear una medida calculada y calcular miembros de dimensiones, combinando expresiones multidimensionales (MDX), fórmulas matemáticas y funciones definidas por el usuario (UDFs). Esta facilidad, habilita a usted a definir nuevas medidas y miembros de dimensión, basados sobre una sintaxis de fórmulas sencillas. Se pueden registrar adicionales bibliotecas de UDFs, para utilizarse en la definición de miembros calculados.

Propiedades de Miembros

Usted puede definir propiedades para los miembros de dimensión y usar datos para estas propiedades dentro de un cubo. Por ejemplo, si los miembros de la dimensión producto son sus número de partes, es lo mismo hacer varias propiedades asociadas con este

número de parte tales como, el tamaño, color, etc. Usted puede especificar tales propiedades, como una propiedad miembro y utilizarla en las búsquedas analíticas.

Cubos Virtuales

Usted puede juntar cubos, dentro de cubos virtuales, muy parecido al proceso de juntar tablas con vistas en las bases de datos relacionales. Un cubo virtual, provee acceso a los datos en los cubos combinados, si la necesidad de construir un nuevo cubo, mientras permite que se mantenga en mejor diseño en cada cubo individual.

Un cubo podrá ser actualizado, procesando solo los datos que han sido añadidos, en vez de hacerlo con el cubo entero, se puede usar la actualización incremental para actualizar un cubo mientras se este usando.

Agregaciones

Así se le llama al proceso de precalcular sumas de datos, para ayudar a disminuir los tiempos de respuestas, en los procesos de búsquedas de información.

Seguridad

Usando las facilidades de seguridad manejadas por Microsoft SQL Server OLAP services, usted puede controlar quien accesa los datos y los tipos de operaciones que los usuarios pueden ejecutar con los datos. OLAP services soporta el sistema de seguridad integrado que ofrece el sistema operativo Windows NT y permite que usted asigne permisos de acceso, a la base de datos y al cubo incluyendo a los cubos virtuales.

La seguridad es manejada vía los derechos de control de acceso que son manejados por los Roles, estos determinan el tipo de acceso a los datos. Los Roles definen, categorías de usuarios con los mismos controles de acceso.

Modos de Almacenaje

Para los cubos, se ofrece tres formas de almacenar su información:

1.- MOLAP - Multidimensional OLAP.

2.- ROLAP - Relacional OLAP.
3.- HOLAP - OLAP híbrido.

MOLAP :Los datos fuente del cubo son almacenados junto con sus agregaciones (sumarizaciones) en una estructura multidimensional de alto rendimiento. El almacenaje de MOLAP, provee excelente rendimiento y compresión de datos. Como se dice, todo va en el cubo.
Tiene el mejor tiempo de respuesta, dependiendo solo en el porcentaje y diseño de las agregaciones del cubo. En general este método, es muy apropiado para cubos con uso frecuente por su rápida respuesta.

ROLAP : Toda la información del cubo, sus datos, su agregación, sumas etc., son almacenados en una base de datos relacional. ROLAP no almacena copia de la base de datos, accesa las tablas originales cuando necesita responder a preguntas, es generalmente, mucho más lenta que las otras dos estrategias de almacenaje. Típicamente ROLAP se usa, para largos conjuntos de datos que no son frecuentemente buscados, tales como datos históricos de de los años más recientes.

HOLAP : combina atributos de MOLAP y ROLAP, la agregación de datos es almacenada en una estructura multidimensional usada por MOLAP, y la base de datos fuentes, en una base de datos relacional. Para procedimientos de búsqueda que accesan datos sumarizados, HOLAP es equivalente a MOLAP, por el contrario si estos procesos accesaran datos fuentes como los drill down, estos deben de buscar los datos en la base de datos relacional y esto no es tan rápido comparado a si los datos estuvieran almacenados en una estructura MOLAP.
Los cubos almacenados en como HOLAP, son más pequeños que los MOLAP y responden más rápidos que los ROLAP. HOLAP es generalmente usado para cubos que requieren rápida respuesta, para sumarizaciones basadas en una gran cantidad de datos.

TRANSACIONES Y REPLICAS COMPLEMENTO PRACTICO

Transacciones

Uno de los mayores riesgos al momento de interactuar con las bases de datos es que a la hora de realizar una operación (por medio del sistema) se produzca un error el cual no permita que dicha operación se efectué de manera completa. El ejemplo clásico de esto es una transferencia bancaria, en la que quitamos saldo a una cuenta y

lo añadimos en otra. Si no somos capaces de abonar el dinero en la cuenta de destino, no debemos quitarlo de la cuenta de origen.

Supongamos que al instante de realizar la transacción bancaria haya fallado el servicio de internet o se produjo un corte en la energía eléctrica, esto ocasionaría una situación de incertidumbre respecto a lo que realmente paso al momento de la operación: ¿se logro completar la operación bancaria o no? .Una de las maneras con la que podemos evitar que esto suceda es mediante el uso de las transacciones.

¿Qué es una transacción?

Una transacción representa un conjunto de instrucciones que se ejecutan de manera única, lo que implica que si falla una operación fallan todo el resto. Es decir deben de considerarse dos posibles escenarios:

1) Si la transacción tiene éxito, todas las modificaciones de los datos realizadas durante la transacción se confirman y se convierten en parte permanente de la base de datos.

2) Si una transacción encuentra errores y debe cancelarse o revertirse, se borran todas las modificaciones de los datos.

Las transacciones deben cumplir 4 propiedades fundamentales las cuales son comúnmente conocidas como ACID (atomicidad, coherencia, asilamiento y durabilidad).

1) **Atomicidad:** Una transacción debe ser una unidad atómica de trabajo, tanto si se realizan todas sus modificaciones en los datos, como si no se realiza ninguna de ellas.

2) **Coherencia:** Cuando finaliza, una transacción debe dejar todos los datos en un estado coherente. En una base de datos relacional, se deben aplicar todas las reglas a las modificaciones de la transacción para mantener la integridad de todos los datos. Todas las estructuras internas de datos, como índices de árbol B o listas doblemente vinculadas, deben estar correctas al final de la transacción.

3) **Aislamiento:** Las modificaciones realizadas por transacciones simultáneas se deben aislar de las modificaciones llevadas a cabo por otras transacciones simultáneas. Una transacción ve los datos en el estado en que estaban antes de que otra transacción simultánea los modificara o después de que la segunda transacción se haya concluido, pero no ve un estado intermedio. Esto se conoce como seriabilidad debido a que su resultado es la capacidad de volver a cargar los datos iniciales y reproducir una serie de transacciones para finalizar con los datos en el mismo estado en que estaban después de realizar las transacciones originales.

4) **Durabilidad:** Una vez concluida una transacción, sus efectos son permanentes en el sistema. Las modificaciones persisten aún en el caso de producirse un error del sistema.

Un ejemplo de una transacción utilizando el servidor de bases de datos SQL Server es una única sentencia SQL. Por ejemplo una sentencia como esta:

```
UPDATE Medicamento SET Stock = 45 WHERE CodMedicamento ='MED00120'
```

El código anterior representa una transacción autocompletada (autocommit). Cuando enviamos esta sentencia al SQL Server se escribe en el fichero de transacciones lo que va a ocurrir y a continuación realiza los cambios necesarios en la base de datos. Si hay algún tipo de problema al hacer esta operación el SQL Server puede leer en el fichero de transacciones lo que se estaba haciendo y si es necesario puede devolver la base de datos al estado en el que se encontraba antes de recibir la sentencia.

Este tipo de transacciones no requieren de ninguna intervención de nuestra parte puesto que el sistema se encarga de todo. Sin embargo si es necesario realizar varias operaciones y queremos que sean tratadas como una unidad tenemos que crear esas transacciones de manera explícita.

Tipos de transacciones

Como decíamos una transacción es un conjunto de operaciones tratadas como una sola. Este conjunto de operaciones debe marcarse como transacción para que todas las operaciones que la conforman tengan éxito o todas fracasen.

Existen dos tipos de transacciones las cuales proporcionan diferentes métodos para agrupar varias sentencias usando Transact SQL en una única transacción:

a) **Transacciones explícitas*:*** Cada transacción se inicia explícitamente con la instrucción **BEGIN TRANSACTION** y se termina explícitamente con una instrucción **COMMIT** o **ROLLBACK**.

b) **Transacciones implícitas*:*** Se inicia automáticamente una nueva transacción cuando se ejecuta una instrucción que realiza modificaciones en los datos, pero cada transacción se completa explícitamente con una instrucción **COMMIT** o **ROLLBACK**.

La sentencia que se utiliza para indicar el comienzo de una transacción es **BEGIN TRANSACTION** (o solo **BEGIN TRAN**). Si alguna de las operaciones de una transacción falla hay que deshacer la transacción en su totalidad para volver al estado inicial en el que estaba la base de datos antes de empezar. Esto se consigue con la sentencia **ROLLBACK TRAN**.

Si todas las operaciones de una transacción se completan con éxito hay que marcar el fin de una transacción para que la base de datos vuelva a estar en un estado consistente con la sentencia **COMMIT TRAN**.

Ahora veamos un ejemplo usando transacciones explicitas:

En este ejemplo hacemos uso de la base de datos Farmacia (creada para fines demostrativos).

```
USE Farmacia

Declaramos una variable que utilizaremos para almacenar un posible
código de error
DECLARE @Error int

Iniciamos la transacción
BEGIN TRAN
```

```
Ejecutamos la primera sentencia
UPDATE Medicamento SET Stock = 45 WHERE CodMedicamento ='MED00120'

Si ocurre algún error almacenamos su código en la variable @Error
y saltamos al bloque de código que se encargara de deshacer la
transacción
SET @Error = @@ERROR
IF (@Error<>0) GOTO ManejarError

Si la primera sentencia se ejecuto con éxito, pasamos a la segunda y
si ocurre algún error hacemos lo mismo que en la primera sentencia
UPDATE Medicamento SET Stock = 55 WHERE CodMedicamento ='MED00140'
SET @Error=@@ERROR
IF (@Error<>0) GOTO ManejarError

El llegar a este punto indica que las sentencias UPDATE utilizadas
anteriormente se han completado con éxito y podemos "guardar" la
transacción en la base de datos, con COMMIT TRAN damos por finalizada
la transacción
COMMIT TRAN

Si ocurre algún error se lo comunicamos al usuario y deshacemos la
transacción, en ese momento todo volverá a estar como si nada hubiera
ocurrido, es decir no se harán cambios en la base de datos
ManejarError:
If @@Error<>0 THEN
        BEGIN
        PRINT 'Error en tiempo de ejecución, transacción abortada'
        ROLLBACK TRAN
END
```

Para el caso de las transacciones implícitas simplemente podemos hacer uso de una sentencia predefinida para activar dicho tipo de transacción, veamos un haciendo uso del ejemplo anterior:

```
Hacemos uso de la base de datos Farmacia
USE Farmacia

Activamos el modo de transacciones implícitas, de tal manera que ya no
es necesario indicar el inicio de la transacción
SET IMPLICIT_TRANSACTIONS ON

Declaramos una variable que utilizaremos para almacenar un posible
código de error
DECLARE @Error int


Ejecutamos la primera sentencia
UPDATE Medicamento SET Stock = 45 WHERE CodMedicamento ='MED00120'

Si ocurre algún error almacenamos su código en la variable @Error
y saltamos al bloque de código que se encargara de deshacer la
transacción
SET @Error = @@ERROR
IF (@Error<>0) GOTO ManejarError

Si la primera sentencia se ejecuto con éxito, pasamos a la segunda y
si ocurre algún error hacemos lo mismo que en la primera sentencia
```

```
UPDATE Medicamento SET Stock = 55 WHERE CodMedicamento ='MED00140'
SET @Error=@@ERROR
IF (@Error<>0) GOTO ManejarError

El llegar a este punto indica que las sentencias UPDATE utilizadas
anteriormente se han completado con éxito y podemos "guardar" la
transacción en la base de datos, con COMMIT TRAN damos por finalizada
la transacción
COMMIT TRAN

Si ocurre algún error se lo comunicamos al usuario y deshacemos la
transacción, en ese momento todo volverá a estar como si nada hubiera
ocurrido, es decir no se harán cambios en la base de datos
ManejarError:
If @@Error<>0 THEN
        BEGIN
        PRINT 'Error en tiempo de ejecución, transacción abortada'
        ROLLBACK TRAN
END
```

Como se puede ver para cada sentencia que se ejecuta se verifica si se ha producido o no un error, y en caso de que detectemos un error ejecutamos el bloque de código que deshace la transacción.

La transacción sigue activa hasta que emita una instrucción **COMMIT** o **ROLLBACK**. Una vez que la primera transacción se ha confirmado o cancelado, se inicia automáticamente una nueva transacción la siguiente vez que la conexión ejecuta una instrucción para modificar datos.

Esto indica que la conexión continúa generando transacciones implícitas hasta que se desactiva el modo de transacciones implícitas (`SET IMPLICIT_TRANSACTIONS OFF`).

También existe una interpretación incorrecta sobre el funcionamiento de las transacciones, la cual es que al tener varias sentencias dentro de una transacción una de estas falla la transacción se aborta en su totalidad, lo que representa un grave error.

Analicemos el siguiente ejemplo:

```
USE Farmacia
BEGIN TRAN
UPDATE Medicamento SET Stock = 45 WHERE CodMedicamento ='MED00120'
UPDATE Medicamento SET Stock = 55 WHERE CodMedicamento ='MED00150'
COMMIT TRAN
```

Estas dos sentencias se ejecutarán como una sola. Supongamos que en medio de la transacción (después de la primera actualización y antes de la segundo) hay un corte de electricidad, cuando el SQL Server se recupere se encontrará en medio de una transacción y, en este momento hay dos alternativas: o bien la termina o bien la deshace, pero no se quedará a medias.

El error está en pensar que si la ejecución de la primera sentencia da un error se cancelará la transacción. El SQL Server sólo se preocupa de ejecutar las sentencias y mas no analiza si lo hacen correctamente o si la lógica de la transacción es correcta,

en otras palabras la responsabilidad de que la transacción actué de manera correcta dependerá de la manera en que controlemos esos errores.

Por ese motivo en el ejemplo presentado anteriormente para cada sentencia de nuestro conjunto averiguamos si se ha producido un error y si es así actuamos en consecuencia cancelando toda la operación.

Transacciones anidadas.

Otra de las posibilidades que nos ofrece el SQL Server es utilizar transacciones anidadas. Esto quiere decir que podemos tener transacciones dentro de transacciones, es decir, podemos empezar una nueva transacción sin haber terminado la anterior.

Podemos anidar varias transacciones. Cuando anidamos varias transacciones la instrucción **COMMIT** afectará a la última transacción abierta, pero **ROLLBACK** afectará a todas las transacciones abiertas.

Un hecho a tener en cuenta, es que, si hacemos **ROLLBACK** de la transacción superior se desharán también los cambios de todas las transacciones internas, aunque hayamos realizado **COMMIT** de ellas.

Asociada a esta idea de anidamiento existe una variable global @@**TRANCOUNT** que tiene valor 0 si no existe ningún nivel de anidamiento, 1 si hay una transacción anidada, 2 si estamos en el segundo nivel de anidamiento y así sucesivamente.

Veamos un ejemplo sencillo de transacciones anidadas:

```
Iniciamos la transacción
BEGIN TRAN
UPDATE Medicamento SET Stock = 15 WHERE CodMedicamento ='MED00150'

Anidamos una transacción
BEGIN TRAN
UPDATE Medicamento SET Stock = 15 WHERE CodMedicamento ='MED00150'

Escribimos la sentencia COMMIT que solo afectara a la transacción
actual, es decir la segunda
COMMIT TRAN

Ahora el ROLLBACK afectara a ambas transacciones
ROLLBACK TRAN
```

Ahora hay que analizar el comportamiento de las instrucciones **COMMIT** y **ROLLBACK** dentro de una transacción anidada:

- **ROLLBACK TRAN:** Dentro de una transacción anidada esta sentencia deshace todas las transacciones internas hasta la instrucción **BEGIN TRAN** más externa.

- **COMMIT TRAN:** Dentro de una transacción anidada esta sentencia únicamente reduce en 1 el valor de @@**TRANCOUNT**, pero no "finaliza" ninguna transacción ni "guarda" los cambios. En el caso en el que @@**TRANCOUNT**=1 (cuando estamos en la última transacción) **COMMIT**

TRAN hace que todas las modificaciones efectuadas sobre los datos desde el inicio de la transacción sean parte permanente de la base de datos, libera los recursos mantenidos por la conexión y reduce @@**TRANCOUNT** a 0.

Para entender mejor el uso de ROLLBACK y COMMIT analicemos los siguientes ejemplos:

```
Creamos una tabla para registrar el nivel de anidamiento de las
transacciones realizadas
CREATE TABLE PruebaTrans (Columna int)

Iniciamos la primera transacción y el valor de @@TRANCOUNT ahora es 1
BEGIN TRAN TransExterna

Mostramos el nivel de anidamiento y lo registramos en la tabla
   SELECT 'Nivel actual de anidamiento es', @@TRANCOUNT
   INSERT INTO Test VALUES (1)

Iniciamos la primera transacción a nivel interno y el valor de
@@TRANCOUNT cambia a 2
   BEGIN TRAN TransInterna1
      SELECT 'Nivel actual de anidamiento es', @@TRANCOUNT
      INSERT INTO Test VALUES (2)

Iniciamos la segunda transacción y el valor de @@TRANCOUNT ahora es 3
   BEGIN TRAN TransInterna2
            SELECT 'Nivel actual de anidamiento es', @@TRANCOUNT
            INSERT INTO Test VALUES (3)

Confirmamos la inserción de los datos y de esta manera se reduce el
contador de transacciones (@@TRANCOUNT = 2) mas no se registra en la
base de datos
   COMMIT TRAN TransInterna2
            SELECT 'Nivel actual de anidamiento es', @@TRANCOUNT
Igual que el anterior solo que ahora el valor de @@TRANCOUNT es 1
   COMMIT TRAN TransInterna1
   SELECT 'Nivel actual de anidamiento es', @@TRANCOUNT

Se lleva a cabo la transacción externa, con esto el @@TRANCOUNT se
reduce a 0
COMMIT TRAN TransExterna
SELECT 'Nivel actual de anidamiento es', @@TRANCOUNT

Y por ultimo mostramos la cantidad de transacciones registradas en la
tabla
SELECT * FROM Test
```

Ahora haciendo uso de ROLLBACK:

```
Iniciamos la primera transacción y el valor de @@TRANCOUNT ahora es 1
BEGIN TRAN TransExterna

Mostramos el nivel de anidamiento y lo registramos en la tabla
   SELECT 'Nivel actual de anidamiento es', @@TRANCOUNT
   INSERT INTO Test VALUES (1)

Iniciamos la primera transacción a nivel interno y el valor de
@@TRANCOUNT cambia a 2
   BEGIN TRAN TransInterna1
      SELECT 'Nivel actual de anidamiento es', @@TRANCOUNT
```

```
      INSERT INTO Test VALUES (2)

Iniciamos la segunda transacción y el valor de @@TRANCOUNT ahora es 3
   BEGIN TRAN TransInterna2
            SELECT 'Nivel actual de anidamiento es', @@TRANCOUNT
            INSERT INTO Test VALUES (3)

Con ROLLBACK TRAN deshacemos la transacción externa y todas las
internas, por lo tanto el valor de @@TRANCOUNT se convierte en 0
ROLLBACK TRAN
SELECT 'Nivel actual de anidamiento es', @@TRANCOUNT
SELECT * FROM Test
```

Puntos de guardado de transacciones

Los puntos de guardado permiten manejar las transacciones por pasos, ofrecen un mecanismo para deshacer partes de una transacción. Para crear un punto de guardado utilizamos la instrucción **SAVE TRANSACTION** *nombrePuntodeGuardado* y, a continuación, ejecute la instrucción **ROLLBACK TRANSACTION** *nombrePuntodeGuardado* para deshacer hasta el punto de guardado en vez de deshacer hasta el inicio de la transacción.

Los puntos de guardado son útiles en situaciones en que no es probable que se produzcan errores. El uso de un punto de guardado para deshacer parte de una transacción en caso de producirse un error no frecuente puede resultar más eficaz que hacer que cada transacción pruebe si una actualización es válida antes de realizarla. Las operaciones de actualizar y deshacer son costosas, por lo que los puntos de guardado son eficaces sólo si la probabilidad de encontrar un error es baja y el costo de comprobar la validez de una actualización de antemano es relativamente alto.

Para comprender mejor el uso de los puntos de guardado veamos el siguiente ejemplo:

```
CREATE TABLE Tabla1 (Columna1 varchar (50))
GO

Comenzamos la transacción, insertamos un registro en la tabla creada
al principio y declaramos el punto de salvado, insertamos otro
registro y terminamos
BEGIN TRAN
INSERT INTO Tabla1 VALUES ('Primer valor')
        SAVE TRAN Punto1
        INSERT INTO Tabla1 VALUES ('Segundo valor')

La instrucción ROLLBACK en este caso no deshará la transacción sino
que cancelara lo ocurrido desde SAVE TRAN nombrePuntodeGuardado hasta
ROLLBACK TRAN
        ROLLBACK TRAN Punto1
        INSERT INTO Tabla1 VALUES ('Tercer valor')
COMMIT TRAN

SELECT * FROM Tabla1
Columna1
```

Duplicación de base de datos (replicas)

La duplicación es un conjunto de tecnologías que le permiten mantener copias de los mismos datos en varios sitios, incluso a veces en centenares de sitios.

La duplicación utiliza un modelo de publicación y suscripción para distribuir datos:

- Un publicador es un servidor que es el origen de los datos que se van a duplicar. El publicador define un artículo por cada tabla u otro objeto de base de datos que se va a utilizar como origen de la duplicación. Uno o varios artículos relacionados de la misma base de datos se organizan en una publicación. Las publicaciones son formas adecuadas de agrupar datos y objetos relacionados que desea duplicar juntos.

- Un suscriptor es un servidor que recibe los datos duplicados por el publicador. El suscriptor define una suscripción a una publicación determinada. La suscripción especifica cuándo recibe el suscriptor la publicación del publicador y asigna los artículos a tablas y otros objetos de base de datos en el suscriptor.

- Un distribuidor es un servidor que realiza varias tareas cuando mueve artículos de los publicadores a los suscriptores. Las tareas reales que se llevan a cabo dependen del tipo de duplicación realizada.

Tipos de duplicación

Existen tres tipos de duplicación:

Duplicación de instantáneas

La duplicación de instantáneas copia los datos o los objetos de base de datos exactamente como existen en un momento dado. Las publicaciones de instantáneas se definen normalmente para que se produzcan a intervalos programados. Los suscriptores contienen copias de los artículos publicados tal y como existían en la última instantánea. La duplicación de instantáneas se utiliza donde el origen de datos sea relativamente estático, los suscriptores puedan no estar actualizados y la cantidad de datos que se va a duplicar sea pequeña.

Duplicación transaccional

En la duplicación transaccional, los suscriptores se sincronizan primero con el publicador, normalmente mediante una instantánea y, a continuación, a medida que los datos de la publicación se modifican, las transacciones se capturan y se envían a los suscriptores.

La integridad transaccional se mantiene en todos los suscriptores haciendo que el publicador realice todas las modificaciones y, a continuación, las duplique para los suscriptores. La duplicación transaccional se utiliza cuando tiene que duplicar los datos según se modifican, tiene que mantener las transacciones, y los publicadores y suscriptores se conectan de forma confiable o frecuente a través de la red.

Duplicación de mezcla

La duplicación de mezcla permite que varios sitios trabajen en modo autónomo con un conjunto de suscriptores y, a continuación, mezclen el trabajo combinado con el

publicador. Los suscriptores y el publicador se sincronizan con una instantánea. El seguimiento de cambios se realiza en los suscriptores y los publicadores. Posteriormente, los cambios se mezclan para formar una sola versión de los datos. Durante la mezcla, es posible encontrar algunos conflictos donde varios suscriptores han modificado los mismos datos.

La duplicación de mezcla admite la definición de resoluciones de conflictos, que son conjuntos de reglas que definen la forma de resolver dichos conflictos. Es posible escribir secuencias de comandos de resolución de conflictos personalizada para controlar la lógica que pueda hacer falta para resolver escenarios de conflictos complejos correctamente. La duplicación de mezcla se utiliza cuando es importante para los equipos suscriptores trabajar en modo autónomo (como un usuario con movilidad desconectado) o cuando múltiples suscriptores deben actualizar los mismos datos.

Ventajas de la duplicación

La duplicación ofrece diferentes ventajas según su tipo y las opciones elegidas, pero la ventaja común de la duplicación es la disponibilidad de los datos en el momento y lugar necesarios.

Otras ventajas son:

- Permitir que varios sitios conserven copias de los mismos datos. Esto resulta de utilidad si varios sitios necesitan leer los mismos datos o precisan servidores diferentes para las aplicaciones de informes.

- Separar las aplicaciones OLTP de las que hacen uso intensivo de lectura, como bases de datos de proceso analítico (OLAP), puestos o almacenes de datos.

- Permitir una mayor autonomía. Los usuarios pueden trabajar con copias de los datos mientras están desconectados y propagar los cambios realizados a otras bases de datos cuando se conecten.

- Ampliar horizontalmente los datos que se van a examinar, como consultar datos con aplicaciones basadas en Web.

- Aumentar el rendimiento acumulado de lectura.

- Acercar los datos a grupos o individuos. Esto ayuda a reducir los conflictos por las consultas y modificaciones de datos de usuarios múltiples, ya que los datos pueden distribuirse a través de la red y se pueden dividir los datos según las necesidades de las diferentes unidades de negocio o usuarios.

- Utilizar la duplicación como parte de una estrategia personalizada de servidor de reserva. La duplicación es una opción de la estrategia del servidor de reserva. Otras opciones de SQL Server 2000 incluyen el transvase de registros y clúster de conmutación por error, que proporcionan copias de los datos en el caso de un error en el servidor.

¿Cuándo se debe de utilizar la duplicación?

Debido a que las organizaciones utilizan diversos tipos de hardware y aplicaciones de software en entornos distribuidos, se hace necesario almacenar los datos de manera redundante. Además, las diferentes aplicaciones tienen necesidades distintas de autonomía y coherencia de datos.

La duplicación es una solución para un entorno de datos distribuidos si necesita:

- Copiar y distribuir los datos a uno o más sitios.
- Distribuir copias de datos en función de una programación.
- Distribuir las modificaciones en los datos a otros servidores.
- Permitir que varios sitios y usuarios efectúen cambios y después mezclar entre sí las modificaciones de los datos, y así poder identificar y resolver los conflictos.
- Construir aplicaciones de datos que tienen que utilizarse en entornos en línea y sin conexión.
- Construir aplicaciones web con las que los usuarios puedan examinar grandes cantidades de datos.

Bibliografía

- Manual de SQL Server 2005
- www.monografias.com
- www.bujarra.com