CAPÍTULO I

INTRODUCCIÓN A LAS BASES DE DATOS DISTRIBUIDAS

1.1. INTRODUCCIÓN TEÓRICA A LA DISTRIBUCIÓN DE DATOS.

El procesamiento de Bases de Datos Distribuida continúa en evolución y de ninguna forma estamos hablando de una disciplina en su plena madurez. Como advirtieron Bernstein y Goodman sobre un aspecto del procesamiento distribuido "El control de concurrencia distribuido, en comparación (con el no distribuido), esta en un estado de alta turbulencia. Para los DDBMS (Distribuited Data Base Manager System) se han propuesto más de 20 algoritmos de control de concurrencia, y varios de ellos han estado, o están en proceso de ser implementados. Tales algoritmos son complejos, complicados de comprender, su corrección es difícil de probar, y hay que hacer notar que muchos de ellos son incorrectos"[1].

Muchos de los problemas han sido identificados, pero se conocen pocas soluciones. Además, el tema es en extremo complejo, y la investigación se encuentra dividida por varias facetas de los problemas. Gran parte del trabajo ha sido teórico, por lo tanto, muchos aspectos de tipo práctico han sido ignorados.

Al mismo tiempo, los usuarios finales que cuentan con microcomputadoras y bases de

[1] *Philip A. Bersntein y Nathan Goodman "Concurreney Control in Distributed Database System" Surveys, Junio, pag. 185*

datos locales han aumentado la presión sobre el departamento MIS para que proporcione alguna forma de procesamiento distribuido. Los fabricantes de los DBMS (Database Manager System) han empezado a anunciar productos que se dicen DDBMS, la mayor parte de los cuales falla al tratar de resolver los problemas de procesamiento de Base de Datos Distribuida. Tales productos mejoraran a través de tiempo, y se desarrollan verdaderos productos DDBMS distribuidos.

El procesamiento de datos distribuida es el procesamiento de base de datos en el cual la ejecución de transacciones y la recuperación y actualización de los datos acontece a través de dos o más computadoras independientes. Por lo general separadas geográficamente. La siguiente figura muestra un sistema de datos distribuida que involucra cuatro computadoras.

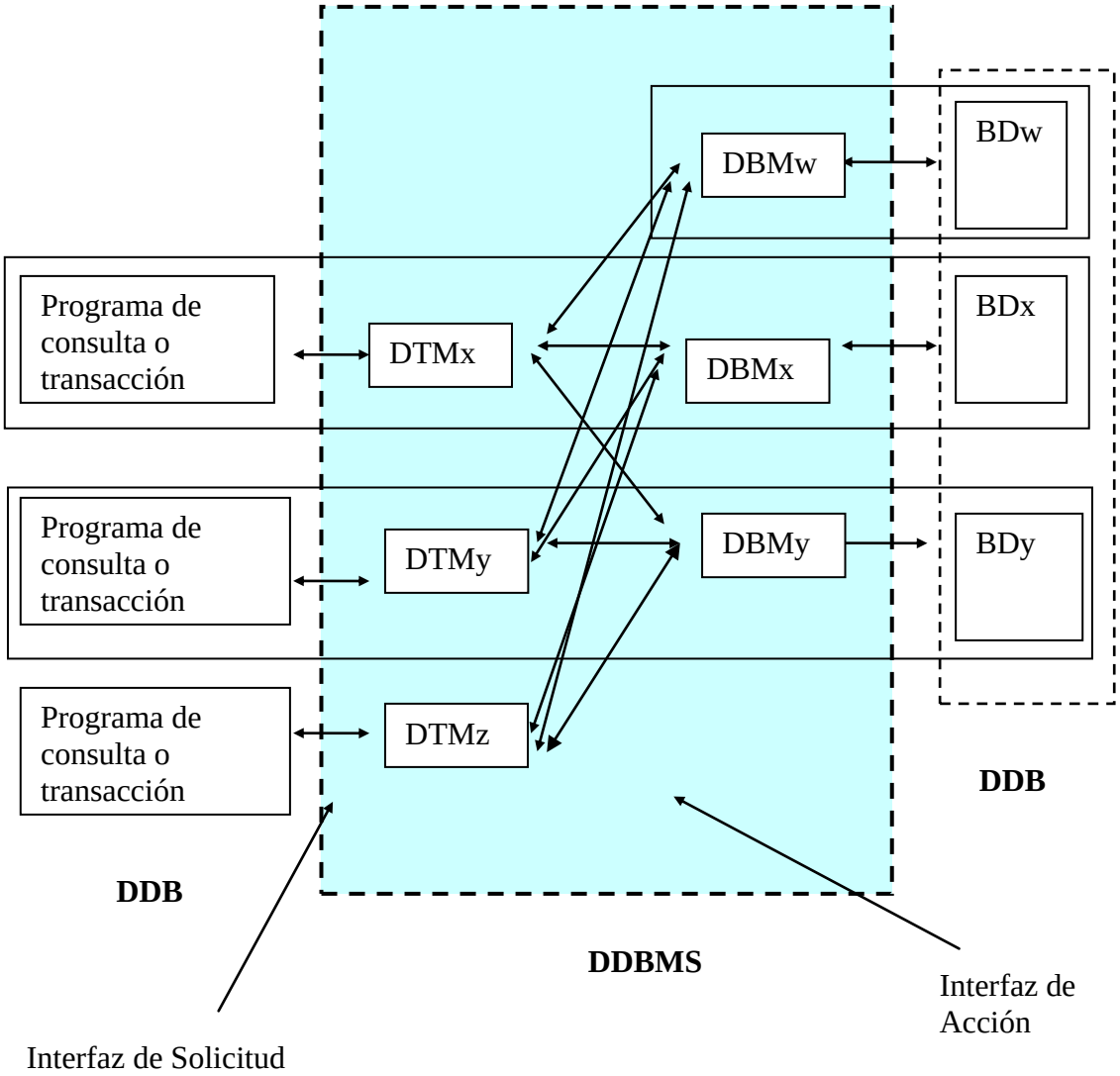


FIGURA Nro. 1
Arquitectura de Base de Datos Distribuida

1.1.1. Conceptos sobre bases de datos distribuidas

El sistema de administración de base de datos distribuida (Distributed Database Manager System DDBMS), esta formado por las transacciones y administradores de base de datos distribuidos de todas las computadoras. Tal y como se muestra, tal

DDBMS es un esquema genérico que implica un conjunto de programas que operan en diversas computadoras. Estos programas pueden ser subsistemas de un producto único DDBMS, concesionado por un solo fabricante, o también pudiera resultar una colección de programas de fuentes dispares: algunos concesionados por fabricantes, y algunos otros escritos en casa.

El administrador de transacciones distribuidas (Distributed Transaction Manager DTM) es un programa que recibe solicitudes de procesamiento de los programas de consultas o de transacciones y a su vez las traduce en acciones para los administradores de la base de datos. Una función importante del DTM es coordinar y controlar dichas acciones. Dependiendo de la aplicación del DDBMS, el DTM puede ser proporcionado como parte del DDBMS o puede desarrollarse en casa por la organización que pone en práctica el sistema distribuido. En otras aplicaciones menos complejas, una parte de sus funciones puede ser llevada a cabo por personas, siguiendo solo procedimientos manuales.

Un administrador de Base de Datos (Database Manager DBM) es un programa que procesa una porción de datos distribuida, como es el hecho de recuperar y actualizar los datos del usuario y generales, de acuerdo con los comandos de acción recibidos de los DTM. El DBM puede ser un conjunto de DDBMS, o ser también un DBMS comercial no distribuido. En algunos casos, el DDBMS pudiera contener diferentes productos DBMS.

Un nodo es una computadora que ejecuta un DTM, DBM, o inclusive ambos. Un nodo

de transacción procesa un DTM, y un nodo de base de datos procesa un DBM y su base de datos. En la figura, el nodo W de base de datos ejecutando DBMw y almacenando. El nodo X es tanto un nodo de transacción como de base de datos con un DTMx, DBMx y BDx. De modo similar, el nodo Y es tanto un nodo de transacción como de base de datos, pero el nodo Z es solamente un nodo de transacción.

Los programas de consulta o de transacción se comunican con los DTM a través de solicitudes parecidas a las solicitudes de acción del DBMS. Ejemplos son SELEC EMPLOYE WHERE E# EQ 123 O bien STORE DUE-DATE. Estas solicitudes operan sobre estructuras lógicas. El programa de consulta o de aplicación no se refiere a ninguna instancia física en particular de la estructura.

Los DTM se comunican con los DBM por medio de acciones a ejecutarse en ocurrencias específicas de datos. Por lo tanto, si la nueva ocurrencia de DUE-DATE debe almacenarse en BDx y en BDy, el DTM traducirá la solicitud STORE DUE-DATE en dos acciones. Una se dirigirá a DBMx para almacenar los nuevos datos, y la segunda se dirigirá a DBMy para a su vez almacenar tal información. En principio, las solicitudes las solicitudes y las acciones pueden también diferir en términos de su nivel abstracción. Por ejemplo, se pueden expresar una solicitud en términos de un objeto y puede ser traducida en acciones o expresada en términos de relaciones compuestas distribuidas o de archivo de archivo. A la fecha, no existe un DBMS como este.

1.1.2. Ventajas del Procesamiento Distribuido

Existen cuatro ventajas del procesamiento de base de datos distribuido:

- Puede dar como resultado un mejor rendimiento que el que se obtiene por un proceso centralizado. Los datos pueden colocarse cerca del punto de utilización, de forma que el tiempo de comunicación sea mas corto. Varias computadoras operando en forma simultánea pueden entregar más volumen de procesamiento que una sola computadora.

- Los datos duplicados aumentan su confiabilidad. Cuando falla una computadora, se pueden obtener datos extraídos de otras computadoras. Los usuarios no dependen de la disponibilidad de una sola fuente para sus datos.

- Los sistemas distribuidos pueden variar su tamaño de un modo más sencillo. Se pueden agregar computadoras adicionales a la red conforme aumentan el número de usuarios y su carga de procesamiento. A menudo es fácil y más barato agregar a una computadora única y centralizada. Después, si la carga de trabajo se reduce, el tamaño de la red también puede reducirse.

- Los sistemas distribuidos se pueden adecuar de una manera más sencilla a las estructuras de la organización de los usuarios. La figura 2 muestra la organización de un fabricante de distribuido geográficamente. Los gerentes

generales de cada una de las plantas poseen una enorme autoridad y libertad en la operación de sus instalaciones. Si tales plantas dependieran de una computadora única centralizada, la arquitectura de sistema estaría en conflicto con la filosofía y las políticas operacionales de la empresa. Incluso en organizaciones más centralizadas, el procesamiento distribuido ofrece una mayor flexibilidad para adecuarse a la estructura organizacional, de lo que permite el procesamiento centralizado.

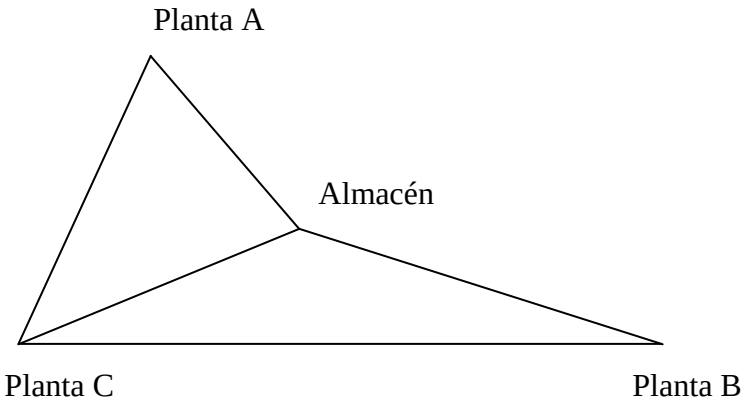


FIGURA Nro. 2
Un negocio Distribuido Geográficamente

1.1.3. Desventajas del Procesamiento Distribuido

Las primeras dos desventajas de las bases de datos distribuidas se relacionan con las dos primeras ventajas.

- El rendimiento puede ser peor para el procesamiento distribuido que para el procesamiento centralizado. Depende de la carga de trabajo, la red, el DDBMS y

las estrategias utilizadas de concurrencia y de falla, así como las ventajas del acceso local a los datos y de los procesadores múltiples, ya que éstos pueden ser abrumados por las tareas de coordinación y de control requeridas. Tal situación es probable cuando la carga de trabajo necesita un gran número de actualizaciones concurrentes sobre datos duplicados, y que deben estar distribuidos.

- El procesamiento de bases de datos distribuidas puede resultar menos confiable que el procesamiento centralizado. De nuevo, depende de la confiabilidad de las computadoras de procesamiento, de la red, del DDBMS, de las transacciones y de las tasas de error en la carga de trabajo. Un sistema distribuido puede estar menos disponible que uno centralizado.

- Estas dos desventajas indican que un procesamiento distribuido no es ninguna panacea. A pesar de que tiene la promesa de un mejor rendimiento y de una mayor confiabilidad, tal promesa no está garantizada.

- Su complejidad, a menudo se traduce en altos gastos de construcción y mantenimiento. Ya que existe componentes de hardware, hay más cantidad de cosas por aprender y más interfaces susceptibles de fallar. El control de concurrencia y recuperación de fallas puede convertirse en algo complicado y difícil de implementar, puede empujar a una mayor carga sobre programadores y personal de operaciones y quizá se requiera de personal más experimentado y más costoso.

1.1.4. Componentes de los Sistemas de Bases de Datos Distribuidas

Los componentes de los sistemas de base de datos distribuidos son confusos, ya que múltiples tipos de distintos de procesamiento entran dentro del término de procesamiento de base de datos distribuidas y pueden encajar en la arquitectura general de la figura 1. Considere, por ejemplo, el sistema de la figura 3(a). Cumple con la arquitectura mostrada en la figura 1 en la cual los nodos están especificados como macrocomputadoras. Es muy probable que para este sistema, el procesamiento este basado en la igualdad de colegas cooperando. Cada nodo de la base de datos (W, X, y Y), posee autoridad para insertar, modificar, suprimir, o bien leer cualquier dato a todo lo largo de la red. Los datos también se coordinan entre computadoras en tiempo real o algo tan cercano como sea posible.

Ahora vayamos a la figura 3(b), en la cual el nodo W es una macrocomputadora, los nodos X e Y son minicomputadoras , y el nodo Z es microcomputadora. En tal caso, las reglas del procesamiento podrían ser las siguientes: sólo el nodo W puede modificar la base de datos; los Nodos X e Y, que tienen copia de los datos del Nodo W, solo están autorizados para lectura, y el Nodo Z puede obtener datos nada más del Nodo Y. No se ha hecho ningún intento por conservar los datos actualizados en tiempo real. En lugar de ello, todos los días los Nodos X e Y son actualizados por el Nodo W, y una vez por semana el Nodo Y actualiza al Nodo Z.

La Figura 3(c) muestra un tercer ejemplo de la arquitectura de la Figura 1, en el cual el Nodo W es una macrocomputadora, y los Nodos X, Y y Z son microcomputadoras

conectadas a una red de área local. El Nodo X es un acceso la macrocomputadora y obtiene de W todos los datos de la base de datos que necesitan X, Y y Z y los almacena en su propia base de datos. Suponga que el Nodo Y requiere de un acceso frecuente a una parte de los datos, pero los procesa sólo en base de lectura. Cuando el Nodo X o el Nodo Z efectúan modificaciones a los datos, las efectúan sobre una copia de W. En forma periódica, W actualiza la base de datos de X con aquellos datos que hayan sido modificados.

Estos tres ejemplos cumplen con la arquitectura general de la Figura l, pero son del todo distintos. Tienen sus propios conjuntos de capacidades, así como su propio conjunto de problemas. Para poner orden en tal complejidad, consideraremos los siguientes cinco componentes de un sistema de base de datos distribuida: hardware, programas, datos, procedimientos y personal.

Figura 3
Tres Concurrencias de Arquitectura de la Figura 1

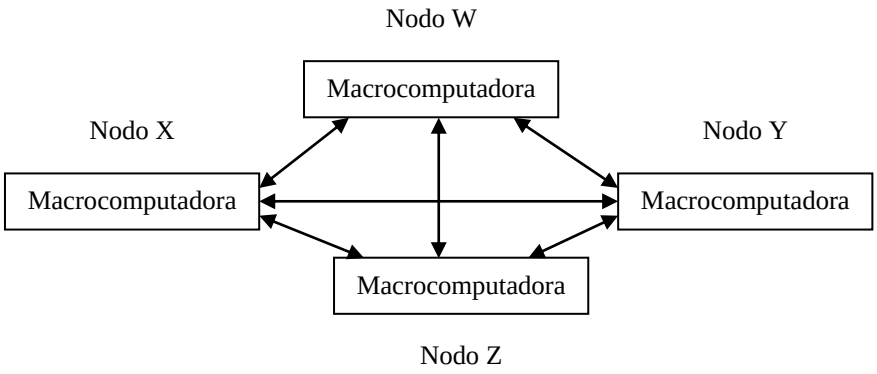


Figura 3 (a)
Sistema distribuido con cuatro macrocomputadoras

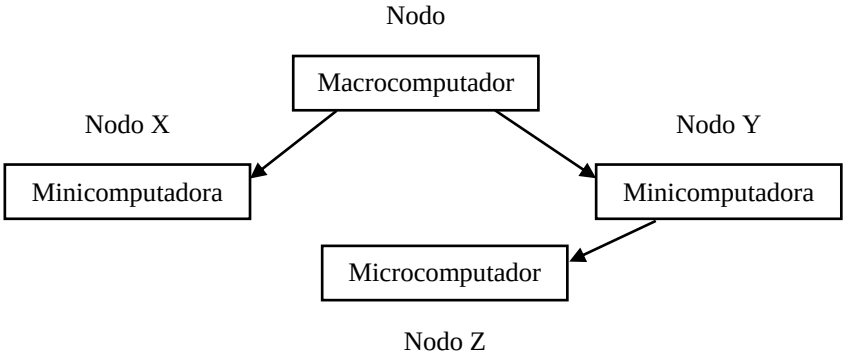


Figura 3 (b)
Sistema distribuido utilizando una macrocomputadora, dos minicomputadoras y una microcomputadora

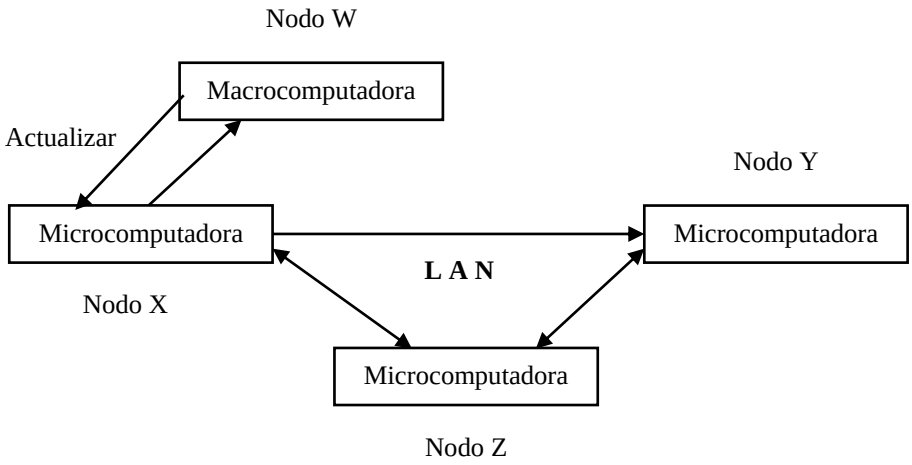


Figura 3(c)
Sistema distribuido utilizando una macrocomputadora, una LAN y tres microcomputadoras

1.1.4.1. Hardware

Como se muestra en la Figura 3 los nodos de procesamiento pueden estar formados por múltiples y distintos tipos de hardware. En algunos sistemas distribuidos, como en la figura 3(a), los nodos son homogéneos. En otros, como en la Figura 3(b) y (c) son heterogéneos. Al determinar las autorizaciones de proceso de los nodos deberán tomarse en cuenta las diferencias en las velocidades de procesamiento y las capacidades de almacenamiento.

1.1.4.2. Programas

El programa principal que necesitamos considerar en un sistema de base de datos distribuidas es el DDBMS. La arquitectura DDBMS que se muestra en la figura 1 es genérica. Los DTM y DBM pueden ser subsistemas de un solo producto DDBMS concesionado por un solo fabricante. En forma alterna y de manera más frecuente DDBMS es una amalgama de programas desarrollados en casa y de productos obtenidos por varios, fabricantes de software. En muchos casos, los DTM se escriben en casa, y los DBM son productos DBMS comerciales.

Considerando los ejemplos de la Figura 3. En el primero, las computadoras son del mismo tipo de máquinas (Macrocomputadora) y la empresa, con tal sistema, podría decidir adquirir la licencia de un DDBMS que sea un solo producto. De hacerlo así, los DTM y DBM serian proporcionados por el fabricante como subsistemas del DDBMS,. Cada DTM esperaría estar en comunicación con sólo el DBM proporcionado por el fabricante. X, un DDBMS prototipo desarrollado por IBM, es un ejemplo de un producto como este. Sin embargo, tales productos DDBMS son raros

Como se observa en la Figura 3(b). El sistema incluye una mezcla de tipos de hardware, y es poco probable que cualquier producto DDBMS comercial, pudiera funcionar con todos ellos. Los DBM podrían ser versiones de un producto DBMS comercial, siempre y cuando dicho producto se ejecute sobre todo tipo de hardware, Oracle, por ejemplo, se ejecutará en macrocomputadoras, microcomputadoras y

minicomputadoras. Los DTM no forman parte del producto comercial. En lugar de ello, serían programas de aplicación escritos en casa, con acceso a producto de una computadora exterior y descargando los datos. O se podrían utilizar como DBM varios productos DBMS, tal y como a continuación se describe.

En la Figura 3(c) se muestra una macrocomputadora que ha entrado en contacto con una red de área local de microcomputadoras. En tal caso, el DBM podría ser nada más un producto, como en el caso anterior, o una mezcla de productos. Si se tratara de una mezcla, el DBM del Nodo W pudiera ser un producto DBMS de una macrocomputadora como DB2, y los DBM de los Nodos X, Y y Z podrían ser versiones LAN multiusuario de un DBMS de microcomputadora como sería ,SQL Server o Informix. Es posible que un programa de extracción que obtiene datos de la base de datos en el Nodo X, fuera proporcionado por el vendedor del DBMS para microcomputadora. Entonces el DDBMS podría ser un solo producto o una amalgama de diferentes programas y productos.

1.1.4.3. Datos

Una de las claves de un sistema de base de datos distribuida consiste en que los datos se duplican o no. Una base de datos distribuida puede no estar duplicada, parcialmente duplicada o totalmente duplicada. Si no está duplicada, existe una y sólo una copia de cada elemento de datos. Los elementos de datos se asignan a nodos particulares, y pueden residir en su nodo asignado. Cualquier aplicación que necesite tener acceso a ese elemento de datos deberá obtenerlo del nodo oficialmente

designado.

Una base de datos parcialmente duplicada contiene ciertos elementos de datos que están duplicados y algunos que no lo están. Un directorio de sistema indica si un elemento de datos está duplicado y dónde se encuentra. Una base de datos distribuida totalmente duplicada, es aquella en la que la base de datos total se encuentra duplicada en dos o más nodos. Un nodo contiene o toda la base de datos o nada de ella (Ver anexo B “Tipos de Fragmentación”).

1.1.4.4. Procedimientos

Los sistemas de base de datos distribuidos contienen una multitud de componentes de procesamiento. El primer grupo corresponde a **derechos de procesamiento**. ¿Qué nodos pueden hacer qué y con qué datos?. En los sistemas distribuidos más sencillos, los datos no están duplicados, y sólo el nodo que almacena los datos pueden actualizarlos. (De hecho, en el sistema más sencillo de todos, los datos no se actualizan en lo absoluto, se obtienen de una fuente ajena, y se procesan como datos de lectura. Una situación así es rara.). En situaciones más complicadas, cualquier nodo puede emitir una solicitud de actualización para cualquier elemento de datos, por si mismo o para cualquier otro nodo. Si este elemento de datos está duplicado, todas sus copias serán modificadas. Al diseñar un sistema distribuido, los desarrolladores deberán determinar los derechos de procesamiento, los requerimientos y las capacidades del hardware y de los programas, el control de la concurrencia, así como otros factores.

Otro componente de los procedimientos es la **actualidad de los datos**. ¿Qué tan actuales deberán ser los datos? ¿Necesita cada uno de los nodos el valor más actual de todos los elementos de datos a los cuales tiene acceso? ¿Puede permitirse que algunas bases de datos se queden desactualizadas? En la Figura 3(a), todos los nodos poseen acceso a los datos más actuales. En la Figura 3(b) y (c), algunos de los nodos se encuentran procesando datos anteriores. Mientras más actuales sean los datos duplicados, más costoso será el sistema. Para controlar y coordinar el sistema de la Figura 3(a) se deberán dedicar enormes cantidades de ciclos de procesamiento y para procesar esta red se requieren CPU poderosas y costosas.

Muy relacionado con los temas de los derechos de procesamiento y de actualidad es el tema del flujo de los datos. ¿,Quién actualiza a quién? En la Figura 3(b), el Nodo W actualiza los datos de los Nodos X e Y, en tanto que en la Figura 3(c), el Nodo X actualiza los datos del Nodo W. Tales flujos quedan determinados por los requisitos y los derechos de procesamiento de los nodos.

Otro componente de procedimientos es el de control. Por lo que se refiere a las solicitudes de procesamiento conflictivas, ¿qué nodo resolverá el conflicto? En general, un control autoritario -por lo común puesto en práctica en sistemas como los de las Figuras 3(b) y (c) -es más fácil de instalar que esquemas basados en la igualdad -casi siempre puestos en práctica en sistemas parecidos a los de la Figura 3(a).

Para los sistemas distribuidos como el de la Figura 3(a), el control puede quedar

distribuido y difundido a todo lo largo de la red. No es necesario que ningún nodo quede a cargo. Las decisiones de control pueden ser llevadas a cabo por cualquiera de los nodos, dependiendo sólo del tema de control y del estado del sistema. Tal situación permite una mayor flexibilidad y, sin embargo, es mucho más compleja.

1.1.4.5. Personal

Los sistemas distribuidos varían en el conocimiento que exigen a las personas que trabajan con ellos. Los sistemas con un DDBMS complejo y poderoso imponen pocas demandas especiales sobre los usuarios. De hecho, los usuarios no saben que están procesando datos distribuidos. Sólo tienen acceso a sus aplicaciones, y todo el procesamiento distribuido lo efectúa el DDBMS. En sistemas menos complejos, los usuarios deben involucrarse.

Considere el sistema de la Figura 3(b). Los usuarios en los Nodos X e Y pudieran necesitar acudir a uno o más programas para hacer que los datos se descarguen de la macrocomputadora. En forma similar, los usuarios en el Nodo Z pudieran necesitar iniciar de manera manual programas a fin de traer los datos del Nodo Y. Dependiendo del diseño del sistema, los usuarios también pueden responsabilizarse de la inspección de los informes de proceso para determinar que los datos fueron recibidos sin error, y que por lo tanto fueron transmitidos los datos correctos.

En sistemas distribuidos muy primitivos, los usuarios deben incluso aceptar la carga de algunas de las responsabilidades del DTM. Por ejemplo, en algunos sistemas los

usuarios efectúan modificaciones a los datos sobre la base de datos local y a continuación de manera manual hacen que tales cambios se efectúen en los datos duplicados en los demás nodos. En los sistemas más primitivos, los usuarios emplean el método NIKE: se ponen sus zapatos tenis y corren por el pasillo, llevando disquetes de modificaciones de datos de una computadora a otra.

1.1.5. Cuatro Metas para un DBMS Distribuido

Traiger y sus colegas definieron cuatro metas para un DBMS distribuido, lo queda un excelente marco de referencia para una investigación de los temas, problemas y soluciones propuestos para las bases de datos distribuidas. [2]Cada una de estas metas involucra un aspecto de transparencia.

En un sistema de base de datos distribuida la transparencia significa que las opciones de consulta y los programas de transacciones quedan aislados de la administración de la base de datos distribuidos, de forma tal que obtengan las ventajas del procesamiento distribuido, sin por ello tener que involucrarse en detalles de la distribución de la base de datos. Los programadores y los usuarios pueden concentrarse en la naturaleza y en la lógica del problema de la información que necesitan resolver, no viéndose obligados a tratar asuntos que corresponden al DDBMS.

Las transacciones necesitan tener acceso a la base de datos vía un DDBMS que

[2] *Irving L. Traiger, Jim Gray, Cesare A. Galtieri y Bruce G. Lindsay, "Transactions and Consistency in Distributed Database Systems", Transactions on Database Systems, Septiembre, pp. 323-342*

proporcione los cuatro siguientes tipos de transparencia: localización de los datos, duplicación de los datos, concurrencia y falla. Lo cual significa que, en forma ideal, la transacción ni siquiera esté consciente que los datos se encuentran distribuidos. Los cuatro temas de distribución se manejan tras bambalinas.

1.1.5.1. Transparencia de Localización

Las transacciones necesitan ser independientes de la localización de un elemento de datos particular. De no ser así, las cuestiones de localización complicarían la lógica de la transacción. Considere usted la empresa manufacturera que se utilizó en la Figura 2. Si el gerente de inventarios desea mover refrigeradores de la Planta A a la Planta B, deberán modificarse dos registros de inventario. Suponga que los datos involucrados no están duplicados; pero que puedan estar almacenados en una computadora en cualquiera de las dos localizaciones. Si el programa que procesa esta transacción no es transparente en lo que se refiere a localización de los datos, tendrá que considerar cuatro casos: ambos registros en A, uno en A y uno en B, uno en B y el otro en A o ambos en B. La lógica de la transacción se confunde por la necesidad de considerar la localización de los datos. La lógica sería mucho más complicada para un ejemplo más complejo, en cualquier caso estas consideraciones son innecesarias e inapropiadas para un programa de aplicación.

Se puede conseguir la transparencia de localización si los administradores de transacciones distribuidas (los DTM en la Figura 1) son responsables de determinar la localización de los datos y de emitir las acciones a los DBM apropiados, lo cual se

puede llevar a cabo si los DTM poseen acceso a los directorios de las localizaciones de los datos si los datos se mueven, sólo el DTM necesita involucrarse. Todas las transacciones quedan aisladas de la modificación en la localización.

1.1.5.2. Transparencia de Duplicación

Las transacciones son accesibles a la duplicación si pueden procesarse sin saber cuántas veces, o incluso si los datos están duplicados. La transacción puede actuar como si todos los datos estuvieran almacenados sólo una vez en nada más un nodo. Con la transparencia de duplicación, se pueden crear nuevos duplicados, o los duplicados existentes pueden ser eliminarlos, sin provocar efecto alguno sobre la transacción del usuario o el procesamiento de la consulta.

Para proporcionar transparencia en la duplicación, los administradores de transacciones deben traducir las solicitudes de procesamiento de transacción en acciones para los administradores de la base de datos. Las lecturas son sencillas, el DTM selecciona uno de los nodos que almacena los datos y emite una acción para su lectura. Para facilitar la selección, el DTM pudiera conservar estadísticas sobre el tiempo que se requiere para leer datos de varios nodos, y seleccionar el nodo con el mejor rendimiento. Es más complicada la escritura de datos duplicados, porque el DTM deberá emitir una acción de escritura para cada uno de los DBM que almacena una copia de dichos datos.

Este análisis supone que cada DTM posee una copia exacta y actualizada de un

directorio que indique la localización de los datos. Sin embargo, aparecen problemas interesantes, si consideramos lo que ocurriría cuando el directorio deba modificarse para tomar en cuenta nuevas copias de datos o su eliminación. Resulta crítica la coordinación. Todos los directorios deberán ser instalados de forma que ningún DTM piense que los datos están disponibles antes que así sea (en el caso de lecturas) y que todos los DTM sepan que los datos están disponibles cuando lo estén (en el caso de escrituras). De lo contrario, un DTM pudiera solicitar datos que todavía no estén disponibles o dejar de emitir una orden de escritura a un DBM cuando los datos ya están disponibles. Véase a Bernstein y a Goodman para mayor información sobre el procesamiento de directorios. [3]

1.1.5.3. Transparencia de concurrencia

Aunque múltiples transacciones que involucran la base de datos distribuida se lleven a cabo al mismo tiempo, los resultados de las transacciones no deberán afectarse. El DDBMS proporciona transparencia de concurrencia si los resultados de todas las transacciones concurrentes son consistentes de manera lógica con los resultados que se habrían obtenido si las transacciones se hubieran ejecutado una por una, en algún orden serial arbitrario. Expresada de otra forma, la lógica de las transacciones procesadas en forma concurrente con otras transacciones deberá de ser la misma que sí la transacción se hubiera procesado sola.

[3] *Philip A. Bernstein y Nathan Goodman, "An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases", Transactions on Database Systems, December 1984, pp.696-615*

Se han desarrollado dos estrategias para proporcionar control de concurrencia. Una de ellas, conocida como **bloqueo distribuido en dos fases**. Un segundo método se llama **pedido con sello de recepción**. Ambos han sido implementados en productos DDBMS. Es más común el bloqueo distribuido en dos fases, y lo analizaremos con mayor detalle en la siguiente sección.

1.1.5.4. Transparencia de fallas

La cuarta meta del DDBMS es proporcionar transparencia de fallas, lo que significa que las transacciones sean procesadas de un modo correcto a pesar de fallas en la transacción, en el DDBMS, en la red y en la computadora. Frente a una falla, las transacciones deberán ser atómicas, esto es, ya sea que se procesen todas las transacciones o ninguna de ellas. Además, una vez comprometidos los resultados de las transacciones, serán permanentes.

La transparencia contra fallas es la meta más difícil entre las cuatro. Parte del problema es que existen muchos tipos distintos de fallas. En un extremo, habrá un nodo que jamás falla, a veces llamado un nodo perfecto. En el otro extremo aparece un nodo que falla de una manera del todo desconocida. Un nodo como éste pudiera comunicar basura a través de la red, o bien, en razón de su falla, pudiera enviar acciones inapropiadas, pero con formato válido por la red. Nodos como éstos se conocen como **nodos desquiciados**.

Otro tipo de falla corresponde a nodos que se convierten en malévolos, lo cual

significa que el nodo tiene el propósito expreso de llevar a cabo una actividad no autorizada, o de causar daños, en forma intencional. Es incluso posible considerar fallas donde los nodos conspiran unos con otros para hacer caer al sistema distribuido. A veces se conocen como **fallas bizantinas**.

Entre los extremos de los nodos perfectos y los nodos desquiciados están los nodos sensatos. Un nodo sensato es un nodo que puede fallar, pero sólo en una forma definida y conocida. El ejemplo más sencillo de un nodo sensato es uno que, o es perfecto, o deja de responder por completo.[4]

Otra razón por la cual la transparencia de falla es tan difícil es que el control de concurrencia es muy complicado. En un sentido, los problemas de control de concurrencia se resuelven por medio de recuperaciones de fallas. Es como una burbuja de aire bajo el tapiz que se empuja de una esquina sólo para volver a aparecer en otra. Los mecanismos de control de concurrencia funcionan siempre y cuando no ocurran fallas en ciertos momentos o en ciertos estados de la base de datos distribuida, o siempre y cuando se pueda garantizar la recuperación en una cierta forma, etc. En el caso general de bases de datos divididas y duplicadas quedan aún por resolver múltiples problemas teóricos y prácticos de puesta en práctica. Más adelante analizaremos la transparencia de las fallas con mayor detalle.

[4] *Héctor García Molina, Frank Pittelli y Susan Davidson, "Applications of Bysatine Agreement in Database Systems", Transactions on Database Systems, Marzo 1986, pp. 27-47.*

1.1.6. Control Distribuido de Concurrencia

Las bases de datos distribuidas se enfrentan a los mismos problemas de concurrencia que las bases de datos centralizadas. Aunque los problemas y sus soluciones son más complicados, porque existen varias computadoras independientes y datos potencialmente duplicados. Empezamos con un análisis de las anomalías que pueden suceder si el procesamiento no se controla en forma apropiada. Para el procesamiento centralizado. Nos enfrentaremos a ellas de una manera más formal, a fin de establecer la terminología necesaria para resolver este más difícil problema.

1.1.6.1. Anomalías de procesamiento concurrente

En la Figura 4(a) se ilustra la primera anomalía, llamada a veces anomalía de la actualización perdida. Suponga que las transacciones de este ejemplo corresponden al negocio distribuido del fabricante de la Figura 2. Suponga que las transacciones procesan el mismo elemento de datos no duplicados que se almacena en una computadora en la Planta A. Las transacciones pudieran provenir del mismo DTM o de distintos DTM. Cada transacción está reduciendo la cantidad a la mano de algún artículo del inventario. En estos ejemplos, no es necesario considerar la identidad del artículo.

La nomenclatura, $L_1(N_A)$, se refiere a una lectura de la transacción 1 del valor de N que está almacenado en la Planta A. La flecha y el valor indican el valor leído. De modo similar, $E_1(N_A)$ significa una escritura correspondiente a la transacción 1 de un

valor de N a la computadora en la Planta A. La flecha y el valor indican el valor escrito. Como usted puede ver en este ejemplo. E_1 se ha perdido, porque la escritura de la transacción 2 se ha sobrepuesto.

Una segunda anomalía se muestra en la Figura 4(b), que a veces se conoce como anomalía de lectura inconsistente, misma que ocurre cuando una transacción lee un elemento de datos al mismo tiempo que otra escribe en ella. En este ejemplo, una transacción (T_3) está moviendo cuatro unidades del almacén a la Planta B en tanto que otra transacción (T_4) está contando el número total de unidades en las Plantas A y B y en el almacén. Aunque en las tres localizaciones hay un total de 10 unidades, T_4 llega a la conclusión que solamente hay 6. Porque T_4 lee N_{alm} después que T_3 reduce las unidades pero lee N_B antes que T_3 las incremente.

1.1.6.2. Ejecuciones seriales y seriales equivalentes

Las situaciones ilustradas en la Figura 4 se juzgan como anomalías porque generan resultados que los usuarios no esperan. Generan resultados que son

Figura 4.
Ejemplo de anomalías causadas por concurrencia

T_1: Retirar 1 unidad del inventario de la planta A
T_2: Retirar 2 unidades del inventario de la planta A

Inicio	N_A
$L_1(N_A) \leftarrow 3$	3
$L_2(N_A) \leftarrow 3$	3
$E_1(N_A) \rightarrow 2$	2
$E_2(N_A) \rightarrow 1$	1

L=Leer
E=Escribir

3-1-2=1?

Figura 4(a)
Anomalía de: actualización perdida

T_3: Mover 4 unidades del almacén a la planta B
T_4: Contar las unidades en A, B y el almacén

		N_A	N_B	N_{alm}
Inicio		3	1	6
$L_3(N_{alm}) \leftarrow$		6	3	1
$E_3(N_{alm}) \rightarrow$		2	3	1
$L_4(N_{alm}) \leftarrow$		2	2	1
$L_3(N_B) \leftarrow$	1	1	1	2
$L_4(N_A) \leftarrow$	3	3	1	2
$L_4(N_B) \leftarrow$	1	3	1	2
$E_3(N_B) \rightarrow$	5	2	5	2

L=Leer
E=Escribir

Figura 4 (b)
Anomalía de lectura inconsistente

inconsistentes con resultados que hubieran producido si las transacciones se hubieran ejecutado una por una, es decir de un modo seriado. En la Figura 5 se

muestran dos ejecuciones seriales: una T_1 seguida por T_2 y una segunda T_3 seguida por T_4. En ambos casos los resultados son consistentes con lo que los usuarios esperarían.

Aunque generan resultados consistentes, las ejecuciones seriales prohíben la concurrencia y devienen en un bajo rendimiento. Por lo tanto una de las metas del control de concurrencia es permitir la concurrencia de forma que los resultantes de la ejecución concurrente sean consistentes con los de la ejecución serial.

Ejecución serial T_1T_2	Ejecución serial T_3T_4
$L_1(N\text{^})\leftarrow 3$	$L_3(N_{alm})\leftarrow 6$
$E\ (NA)\rightarrow 2$	$E_3(N_{alm})\rightarrow 2$
$L_2\ (N\text{^})\leftarrow 2$	$L_3(N_B)\leftarrow 1$
$E_2(NA)\rightarrow 0$	$E_3(N_B)\rightarrow 5$
	$L_4(N_A)\leftarrow 3$
	$L_4(N_{alm})\leftarrow 2$
	$L_4(N_B)\leftarrow 5$
L=Leer E=Escribir	

Figura 5
Ejemplos de ejecuciones seriales

1.1.6.2.1. Ejecuciones Equivalentes

Una ejecución de transacciones que no sea serial, pero que genere los mismos resultados que una ejecución serial particular se dice que es una ejecución equivalente a la serial.

En la Figura 6 se muestra una ejecución de T_3 y, de T_4 que resulta equivalente a una ejecución serial de T_3 seguida por T_4.

Planteado de una manera más formal: dos ejecuciones de una serie de transacciones se dicen equivalentes si se llenan dos condiciones: (1) Cada lectura de las dos ejecuciones lee valores de los elementos de datos producidos por la misma escritura en ambas ejecuciones, y (2) La escritura final de un elemento de datos es la misma en ambas ejecuciones? Si usted examina la Figura 6 encontrará que ambas condiciones se cumplen. Estas condiciones poseen sentido porque implican que las transacciones reciben las mismas entradas en ambas ejecuciones y que los valores de los elementos de datos finales también son los mismos.

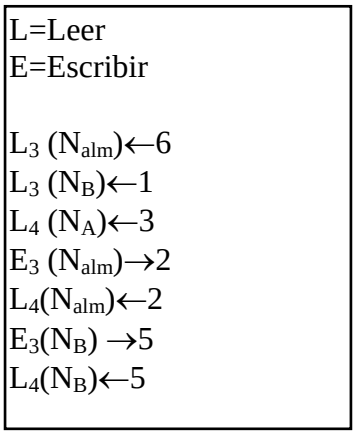


Figura 6

1.1.6.2.1.1. Definiciones

Antes de continuar, necesitamos definir distintos términos. Dos operaciones entran en conflicto si operan sobre el mismo elemento de datos y por lo menos una de las operaciones es una escritura. De lo anterior se concluye que existen dos tipos de conflictos. Sucede un conflicto de lectura-escritura cuando una

operación es una lectura y la otra es una escritura. Sucede un conflicto eje escritura-escritura cuando ambas operaciones son escrituras.

La Figura 7 muestra los conflictos intertransacción en las transacciones T_1 hasta T_4. Por ejemplo. $L_1(N_A)$ tiene un conflicto de lectura-escritura con $E_2(N_A)$ Y $E_1(N_A)$ tiene un conflicto de escritura-escritura con $E_2(N_A)$. Otros conflictos se muestran con las flechas. Observe usted que además de estos conflictos, existen conflictos en el interior mismo de las transacciones. Se supone que estos conflictos se administran mediante la lógica en el interior del programa de transacciones y no deben preocupar al DDBMS. Ejecución no serial de T_3, T_4 que es equivalente a la ejecución serial de T_3, T_4

La Figura 7 muestra una ejecución particular de las transacciones T_1 a T_4. Esta ejecución es una secuencia ordenada de las solicitudes al DTM, es decir un programa. El programa de la Figura 7 es uno de los múltiples programas posibles. Los programas que son equivalentes a los programas seriales se llaman programas consistentes. Considere la ejecución serial T_1 seguida por T_2.seguida por T_3 y seguida por T_4. Cualquier programa que sea equivalente a esta ejecución serial se dice que es consistente con dicha ejecución. Etiquetamos las solicitudes de acuerdo con la transacción que las genera, y por lo tanto SOL_1 se refiere a una solicitud genérica (ya sea una lectura o una escritura) que ocurra en el proceso de T_1.

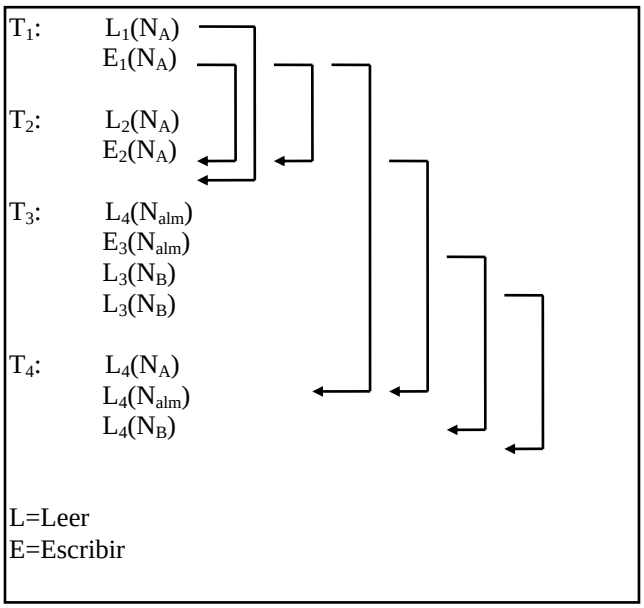


Figura 7
Conflictos intertransacción en T_1, T_2, T_3 y T_4

1.1.6.2.1.1.1. Serialización

La siguiente es una explicación del teorema fundamental de serialización: Suponga que tenemos una lista serializada de transacciones ordenadas, T_1, T_2 ..., T_n. Llame a esta lista ordenada T. Un programa particular **S**, es un programa consistente de **T** si para cualquiera dos solicitudes conflictivas, digamos SOL_i Y SOLj que ocurran de transacciones específicas T_i Y T_j, respectivamente, SOL_i anteceda a SOL_j en **S** si, y solamente si T_i anteceda a T_j en T.

Para comprender el ya mencionado teorema, hay que darse cuenta que está tratando únicamente con el orden de solicitudes conflictivas. Esto es, para que el programa sea consistente con T, el orden de las solicitudes conflictivas debe ser una imagen de espejo del orden de las transacciones que las hacen aparecer.

Por supuesto, no necesitamos preocuparnos con el orden de las solicitudes no conflictivas.

En la Figura 8 se muestra una ejecución consistente pero no serial de la ejecución serial T_1 seguida por T_2 seguida por T_3 seguida por T_4. Observe que, de hecho, el orden de las solicitudes conflictivas es una imagen de espejo del orden de las transacciones.

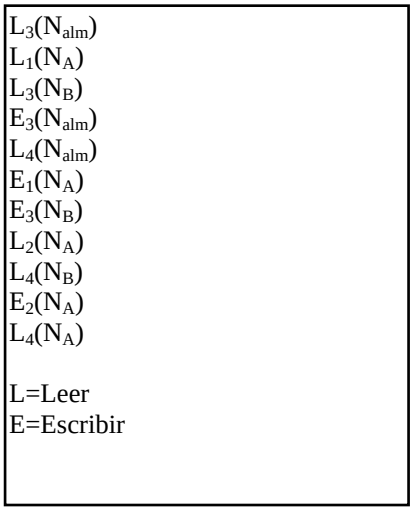


Figura 8
Programa consistente y no serial de T_1, T_2, T_3 y T_4

1.1.6.2.1.1.2. Serialización en Sistemas Distribuidos

Hasta aquí, este análisis se ha preocupado del teleprocesamiento centralizado de igual forma que del procesamiento distribuido. Nos podemos concentrar en el procesamiento distribuido de la siguiente forma: El orden de las solicitudes conflictivas deberán ser una imagen de espejo del orden de las transacciones independientemente de dónde procedan e independientemente de cuántas veces se procesen. En la Figura 9(a) se muestra un programa consistente y

concurrente de las transacciones correspondientes a la Figura 7. Tales transacciones se procesan en dos nodos sin duplicación. La Figura 9(b) muestra un programa consistente y concurrente de dichas transacciones, en el cual los datos del nodo B están duplicados.

Este análisis implica que todos los nodos están de acuerdo en un orden para las transacciones, sin embargo, dos nodos pudieran cada uno de ellos determinar que sus transacciones fueran las siguientes en procesarse: una situación que crea otros problemas. Para resolverlos, de vez en cuando pudiera ser necesario abortar una transacción en proceso y respaldar sus modificaciones.

El teorema fundamental formaliza los objetivos del mecanismo de control de concurrencia. De alguna forma estos mecanismos deben asegurar que las solicitudes conflictivas se procesan en el orden de las transacciones que las generaron. Existen otras muchas formas en las cuales los requerimientos de este teorema fundamental podrían cumplirse.

Figura 9
Ejemplos de programas distribuidos consistentes

Nodo que almacena datos de A y B	**Nodo que almacena datos del almacén**
$L_1(N_A)$	$L_3(N_{alm})$
$L_3(N_B)$	$E_3(N_{alm})$
$E_3(N_B)$	$L_4(N_{alm})$
$E_1(N_A)$	
$L_2(N_A)$	
$L_4(N_B)$	L=Leer
$E_2(N_A)$	E=Escribir
$L_4(N_A)$	

Figura 9 (a)
Programa consistente de T1, T2, T3 y T4 sobre dos nodos

Nodo que almacena datos de A y B	Nodo que almacena datos del almacén
$L_1(N_A)$	$L_3(N_{alm})$
$L_3(N_B)$	$E_3(N_B)$
$E_3(N_B)$	$E_3(N_{alm})$
$E_1(N_A)$	$L_4(N_{alm})$
$L_2(N_A)$	
$L_4(N_B)$	L=Leer
$E_2(N_A)$	E=Escribir
$L_4(N_A)$	

Figura 9 (b)
Programa consistente con datos replicados

1.1.6.3. Control de Concurrencia Utilizando Bloqueo Distribuido en Dos Fases

El método más común de imponer las limitantes del teorema fundamental de serialización se denomina **bloqueo distribuido en dos fases**. A través del uso de este método, los DTM son requeridos para conservar bloqueos antes de leer y describir datos. Antes de leer un elemento de datos, el DTM deberá recibir un bloqueo de lectura del DBM a partir del cual se empleen los datos, y antes de actualizar el elemento de datos, un DTM deberá obtener un bloqueo de escritura de todos los DBM que almacenan este elemento de datos.

1.1.6.3.1. Fases de Crecimiento y de Encogimiento

Los bloqueos se conceden con las siguientes restricciones: Se puede conceder un bloqueo de lectura siempre y cuando ninguna otra transacción posea un bloqueo de escritura sobre un elemento de datos, y un bloqueo de escritura puede ser concedido siempre y cuando ninguna otra transacción tenga un bloqueo de cualquier tipo sobre los elementos de datos. Los bloqueos de lectura pueden ser

compartidos con lecturas, los bloqueos de escritura no pueden ser compartidos con de forma alguna. Si no se puede conceder un bloqueo, la transacción será colocada en un estado de espera hasta que se conceda el bloqueo o se dé la transacción por abortada. Por ultimo, una vez que el DTM libera un bloqueo, no se le puede volver a conceder otro bloqueo. A partir de este punto lo único que se puede hacer es liberar bloqueos.

El término de dos-fases proviene de esta última limitación. Eswaran y sus asociados probaron que los bloqueos de lectura y de escritura descritos generaran programas consistentes si, y sólo si, las transacciones se procesan en dos fases. [6]Durante la primera fase, se les permite adquirir bloqueos sin liberar ninguno. Esto se conoce como fase de crecimiento. En cuanto una transacción libera un bloqueo, en un punto llamado el **punto de bloqueo**, las transacciones entran en la fase de encogimiento. Durante esta segunda fase, las transacciones pueden liberar bloqueos pero no pueden adquirirlos.

1.1.6.3.2. Bloqueo Distribuido

En el caso de bases de datos distribuidas, cada DBM deberá incluir un subsistema que conceda y libere bloqueos. Los DTM deberán ser programados para incorporar las reglas descritas. El DTM puede emitir una solicitud de lectura en cuanto tenga un solo bloqueo de lectura. Sin embargo, deberá obtener bloqueos de escritura de

[6] K. P. Eswaran, J. N. Gray, R. A. Lorie e Y. L. Traiger, "The Notion of Consistenteney and Predicate Locks in Database System", Communications of the ACM. Noviembre 1976, pp. 624-633

todos los nodos que almacenan el elemento de datos antes que emita los comandos de escritura.

Para comprender los bloqueos distribuidos, considere los casos de conflictos de lectura-escritura, de escritura-lectura y de escritura-escritura. Para el conflicto de lectura-escritura suponga que un elemento de datos, A, que reside en los nodos X, Y, y Z y que la transacción T_5 obtiene un bloqueo de lectura para A sobre X. Si otra transacción, digamos T_6 desea actualizar a A, deberá obtener un bloqueo para A sobre X, Y y Z. Los DBM de Y y de Z emitirán el bloqueo sin retardo, pero el DBM sobre X no concederá el bloqueo en tanto T_5 no termine. De esta forma, los conflictos de lectura-escriturase evitan.

Ahora veamos escritura-lectura. Suponga que T_6 posee un bloqueo de escritura para A sobre X, Y y Z. Si T_7 desea leer a A, será necesario que obtenga un bloqueo de lectura en X, Y o Z. Ninguno de los DBM de estos nodos concederán el bloqueo, sin embargo, hasta que T_6 libere los bloqueos. De esta forma, se evitan los conflictos de escritura-lectura.

Finalmente, considere los conflictos escritura-escritura. Suponga que T_8 tiene un bloqueo de escritura para A sobre X, Y y Z y que T_9 desea actualizar a A. T_9 deberá esperar a que todos los bloqueos de escritura se hayan liberado y obtener un bloqueo de escritura para A sobre X, Y y Z. De esta forma, se evitan los conflictos de escritura-escritura.

Para mostrar que esta estrategia genera programas consistentes, también es necesario comprobar que el orden de los bloqueos se conserva igual para un programa serial. Esto se lleva a cabo en la prueba de Eswaran, y es lo que produce la necesidad de las fases de crecimiento y encogimiento.

1.1.6.4. Procesamiento Distribuido de Interbloqueos

La desventaja del control de concurrencia con bloqueos es que se puede producir interbloqueos. La figura 10 ilustra una situación de interbloqueo entre tres transacciones que se ejecutan en tres nodos distintos, A, B y W. Cada nodo contiene un elemento de datos, N, que es la cuenta del número de unidades de algo en el inventario de dicha localización. T_1 está intentando transferir unidades de A a B; T_2 esta intentando transferir unidades de B a alm; y T_3 esta intentando transferir unidades de alm a A.

Para efectuar la transferencia, cada transacción lee primero los datos en las dos localizaciones involucradas. T_1, por ejemplo, lee N_A y N_B. Antes de leer, obtendrá un bloque de lectura sobre estos dos elementos. Cuando T_1 intente escribir en N_A, deberá obtener un bloqueo de escritura sobre él, pero deberá esperar porque T_3 ha obtenido un bloqueo de lectura sobre ese elemento de datos.

Nodo A	Nodo B	Nodo de alm
$L_1(N_A)$	$L_2(N_B)$	$L_3(N_{alm})$
$L_1(N_B)$	$L_2(N_{alm})$	$L_3(N_A)$
$E_1(N_A)$(esperar)	$E_2(N_B)$(esperar)	$E_3(N_{alm})$(esperar)

L=Leer E=Escribir

Figura 10
Ejemplo de un interbloqueo distribuido

Si usted examina, los bloqueos, encontrará que T_1 está esperando a T_3, T_3 está esperando a T_2 y T_2 está esperando a T_1 Esta situación queda diagramada en la gráfica de espera en la Figura 11. En esta gráfica, los nodos (círculos) representan transacciones, y los bordes (las líneas que conectan a los círculos) representan esperas. Una situación de **interbloqueo** existe siempre que existe un circulo es decir una trayectoria de un nodo y de vuelta a si mismo.

Al igual que en el caso de procesamiento centralizado, existen dos estrategias para abordar un interbloqueo. Un método es tener cuidado en la colocación de los bloqueos y no permitir la espera que puede llevar a una situación de interbloqueo. Tal estrategia se denomina **prevención del interbloqueo**. La segunda estrategia es colocar bloqueos sin restricción pero a continuación es fundamental detectar los interbloqueos cuando ocurran. Esto se conoce como **detección de interbloqueo**.

1.1.6.4.1. Prevención del Interbloqueo

Con los procedimientos de prevención de interbloqueo, los administradores del bloqueo en el DBM tienen cuidado al permitir que ocurran esperas. Cuando una transacción T_i pretende colocar un bloqueo que entre en conflicto con otro obtenido por una segunda transacción, T_j, el administrador de bloqueos evalúa la situación y no permite la espera si existe la posibilidad de un interbloqueo. Si existe dicha posibilidad, el administrador de bloqueo puede: rehusar la solicitud de T_i o bien abortar T_j y aprobar la solicitud. Si se niega la solicitud, la estrategia se conoce como **no preferente**. En este caso, T_i deberá abortar y volverse a iniciar. En el caso que T_j sea el abortado, la estrategia se conoce como **preferente**. En este caso, T_j es reiniciado. En ambas estrategias hay distintas variedades. La estrategia más sencilla no preferente es evitar cualquier tipo de espera. Si una transacción, T_i solicita un bloqueo que entra en conflicto 'con otro bloqueo que es propiedad de otra transacción, quedará abortada y reiniciada. Esta estrategia simplifica el procesamiento de bloqueos pero causa muchos reinicios.

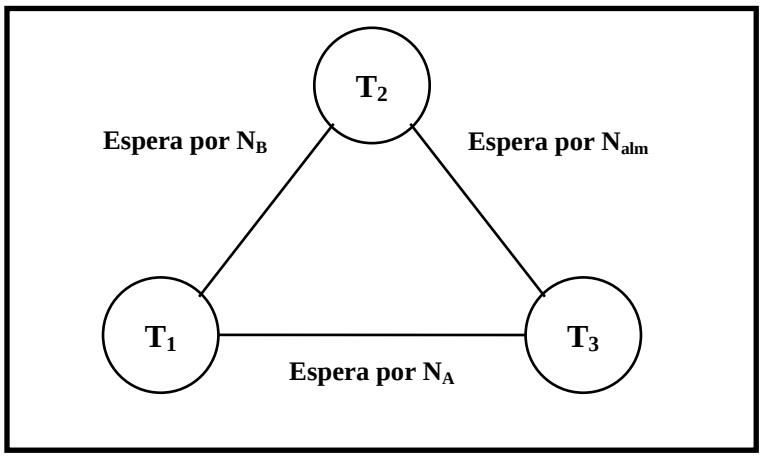


Figura 11
Ejemplo de gráfica de espera

Otra estrategia es asignar prioridades a las transacciones y permitir que la transacción con prioridad más alta tenga precedencia. Entonces cuando T_i solicite un bloqueo sobre un recurso en manos de T_j la respuesta no será no preferencial si la prioridad T_i es menor que T_j y de lo contrario será preferencial. Como todos los demás tipos de programación de prioridades existe el peligro que las transacciones de baja prioridad jamás se les permitirá terminar.

Una tercera estrategia toma en cuenta la edad de las transacciones. Cada transacción recibe una hora única de nacimiento. Cuando aparece un conflicto de bloqueo, las edades relativas se toman en cuenta. En una estrategia de **esperar o desaparecer**, cuando T_i solicita un bloqueo en los datos que mantiene T_j, al T_i se le permite esperar si es más joven que T_j. De lo contrario abortará. Esta es una estrategia no preferencial. Herir y esperar es una estrategia preferencial cuando T_i solicita un bloqueo sobre datos en poder de T_j, y T_i es más viejo que T_j, a T_i se le permite esperar. De lo contrario T_j es descartado y reiniciado. En ambos casos, es

importante que las transacciones conserven sus horas originales de nacimiento al reiniciarse.

Igual que en el caso del procesamiento centralizado de las bases de datos, también es posible evitar el interbloqueo haciendo que todas las acciones soliciten todos los bloqueos en un orden aceptado de antemano. Pero esta restricción viola la transparencia de concurrencia porque obliga a los programadores de las aplicaciones a tomar en cuenta aspectos de concurrencia al desarrollar los programas.

1.1.6.4.2. Detección del Interbloqueo

Con las estrategias de detección de interbloqueo, la espera se permite sin ninguna restricción. Se aceptan todas las solicitudes de bloqueo. Si los bloqueos entran en conflicto, se permite la espera de las transacciones. Hay dos formas en las cuales se detectan los interbloqueos. Una involucra recesos. Esto es, se permite que las transacciones esperen un cierto periodo de tiempo para que se libere un recurso. Cuando el tiempo de espera excede esta cantidad, se presume la existencia de un interbloqueo, y una de las transacciones involucradas en el bloqueo se aborta y se reinicia. Un problema con tal estrategia es que pueden resultar esperas debidas a causas distintas a interbloqueos, en especial en un sistema distribuido. Por lo tanto se abortarán y se reiniciarán en forma innecesaria las transacciones.

El segundo método de detección de interbloqueos es más preciso. Con el se

elaboran gráficas de espera como la que se muestra en la Figura 11, y se buscan los ciclos en dichas gráficas. Si se detectan tales ciclos, existe un interbloqueo, y una o más de las transacciones se abortan y se reinician. Las estructuras de datos para la representación de gráficas y los algoritmos para la identificación de ciclos son más bien conocidos.[7]

El problema de la identificación de ciclos en una base de datos distribuida es que las gráficas de espera requieren del conocimiento de todos los bloqueos en el sistema distribuido. Aunque un solo administrador de bloqueos no posee toda esta información, existen varios métodos de obtenerla. En un método, un nodo en particular se identifica como el detector global de interbloqueo, al cual todos los administradores de bloqueos envían sus datos de bloqueo. Este nodo construye la gráfica global de espera y determina si existen interbloqueos. Si existen, hará que terminen las transacciones apropiadas y se reinicien. *Consulte a Bernstein y a Goodman para un resumen de otros procedimientos de detección de interbloqueos.*[8]

Un ejemplo de control de concurrencia. Uno de los DBMS más antiguos y mejor conocidos es R*, una implementación operacional y prototipo desarrollada por el Almaden Research Center de IBM. Da soporte a bases de datos divididas y distribuidas pero no acepta redundancia. La concurrencia queda controlada por bloqueos distribuidos en dos fases. No se hace ningún intento de evitar el

[7] A.V. Aho, E. Hopcroft, y J.D Ullman, The Desing and Analysis of Computer Programs (Reading, MA: Addison-Wesley, 1975).

interbloqueo; en lugar de ello, se detecta el interbloqueo mediante algoritmos en los nodos distribuidos.

En R*, cada nodo es responsable de determinar sus propios interbloqueos locales. No existe un detector global de interbloqueos, y su detección es compartida entre dos nodos. Los nodos se encuentran programadas para detectar la posibilidad de un interbloqueo global y para enviar información de bloqueo a los nodos que están administrando transacciones que pudieran quedar bloqueadas. Cada nodo es responsable de procesar los datos de algún bloqueo potencial global que se le envía. Los proponentes de R* afirman que el algoritmo es tal que un solo nodo puede detectar un interbloqueo global particular y una vez detectado, se abortarán las transacciones locales.

1.1.7. Transparencia de Fallas

La cuarta meta de un DDBMS es proporcionar transparencia a las fallas. Como hemos visto, las fallas parecen provenir de una diversidad de fuentes. Los nodos sensatos fallan en forma predecible. Los nodos desquiciados fallan en forma impredecible y pueden difundir acciones de formato válido pero inapropiadas. Las fallas maliciosas, de tipo bizantino son provocadas por nodos con intención de engañar.

En este análisis suponemos el tipo más fácil de falla. En particular, suponemos que los nodos son sensatos, pero cuando fallan no hacen nada. Los nodos desquiciados y la

[8] Bernstein y Goodman, “Concurrency Control in Distributed Database System”, pp. 185-221

falla de tipo bizantino se salen del alcance de este análisis. También suponemos que el sistema distribuido está dividido limpiamente. Si el Nodo A no puede comunicarse con el Nodo B, suponemos que el Nodo B no puede comunicarse tampoco con el Nodo A.

Como ya hemos enunciado, la transparencia de las fallas deberá proporcionar transacciones atómicas; o bien se procesará toda, la transacción, o ninguna parte de ella. Una vez comprometida, los efectos de una transacción deberán, ser permanentes. En realidad, esta atomicidad no se puede conseguir en todos los casos, incluso en fallas sensatas. Si una parte demasiado sustancial de la red falla o porciones críticas fallan en momentos críticos, la recuperación con una atomicidad garantizada pudiera no ser posible. El administrador de recuperación de SDD-1, un DDBMS prototipo, reconoce lo anterior, con la definición de **catástrofes** del sistema[9]. Resultan estas catástrofes cuando fallan demasiados componentes. La recuperación de una catástrofe requiere la intervención manual y puede resultar en una base de datos que contenga fragmentos de transacciones. El análisis en esta sección supone que se lleva una bitácora de las actividades de las transacciones, por lo que es posible salirse de una transacción en cualquier momento antes del punto de compromiso.

En ciertos momentos es importante que el DDBMS cuente con la garantía de que una bitácora de control sobrevivirá en caso de falla. En tales casos, utilizamos el término escritura forzadam, lo cual significa que el DDBMS emite un comando de escritura y,

[9] M. Hammer y D. Shipman, "Reliability Mechanisms for SDD-1: A System for Distributed Database", Transactions on Database Systems. December 1980, pp. 431-466

antes de continuar, espera que el sistema operativo confirme que el registro ha sido escrito en un medio de almacenamiento permanente.

1.1.7.1 Necesidad de la administración de directorios

En una base de datos distribuida, cada DTM mantiene un directorio de la localización de los elementos de datos. El procesamiento de estos directorios es crucial: particularmente en un sistema con duplicación. Para comprender la razón, considere la situación que se muestra en la Figura 12.

Esta base de datos, duplicada y distribuida tiene cuatro nodos, etiquetados de A hasta D. El elemento de datos X está almacenado en los Nodos A y B, y el elemento Y en los Nodos C y D. La transacción 1 lee la copia de X que está en A (obteniendo primero un bloqueo de lectura), y la transacción 2 lee la copia de Y en D (también con un bloqueo de lectura). Entonces falla el Nodo A, seguido por el Nodo D. A continuación, la transacción 1 escribe Y en todos los nodos que contienen Y. Para ello, primero deber obtener un bloqueo de escritura en estos nodos. El único nodo tal es C, ya que el Nodo D ha fallado. En consecuencia, la transacción 1 es capaz de escribir Y aun cuando la transacción 2 tiene un bloqueo de lectura sobre éste. En forma similar, la transacción 2 escribe X en todos los nodos que contienen X. El único nodo como éste es B, ya que el Nodo A ha fallado. Así pues, la transacción 2 obtiene un bloqueo de escritura en B y escribe X en B, aun cuando la transacción 1 tenga un bloqueo en escritura en tal nodo.

Cuando subsecuentemente se recuperan los nodos A y D, la base de datos está en un estado no consistente. Los valores de X difieren en los Nodos A y B, y los valores de Y difieren en los Nodos C y D. La inconsistencia ha sido causada por los administradores de transacciones que fueron incapaces de detectar los bloqueos obtenidos por los nodos con falla. La situación puede ser procesada si los mismos directorios son sometidos a cuidadosos procedimientos de bloqueo. Estos procedimientos se presentan en un análisis del **algoritmo de copias disponibles.** [10](Esta referencia es también la fuente del presente ejemplo.)

L_1 (X_A) - con bloqueo de lectura en. X_A
L2 (Y_D) - con Bloqueo de lectura en Y_D
A falla
D falla
E_1 (Y_C) - con bloqueo de escritura en las copias disponibles de Y
E_2 (X_C) - con bloqueo de escritura en las copias disponibles de X
A se recupera
D se recupera
Ahora la base de datos es inconsistente

FIGURA 12
Inconsistencia generada por una falla de nodos inoportuna

1.1.7.2. Compromiso en sistemas de base de datos distribuidas

En sistemas no distribuidos, la atomicidad de las transacciones se consigue

[10] Bresntein y Goodman, "Algorithm for Concurrency Control and Recovery", pp. 596-615

retardando las modificaciones en la base de datos hasta que la transacción está o comprometida o abortada. Para los sistemas distribuidos, comprometer las modificaciones de los datos es más complicado.

Considere usted el siguiente resumen genérico del proceso de compromiso distribuido: a cada transacción se le asigna un espacio de trabajo privado durante su procesamiento. Conforme la transacción progresa, se llevan a cabo actualizaciones en el espacio de trabajo privado, pero no se compromete en la base de datos. Cuando la transacción termina, si todos los nodos que requieran actualizaciones para la transacción son capaces de comprometerlos en sus bases de datos respectivas, se efectúan las modificaciones. De lo contrario la transacción y todas sus modificaciones quedan abortadas.

Tome usted en cuenta un sistema distribuido como el que se muestra en la Figura 1. Cuando un DTM emite una solicitud de actualización a un DBM, el DBM coloca los datos actualizados en un espacio de trabajo privado que se mantiene para esa transacción. (Para una primera actualización, se creará el espacio de trabajo privado.) El DBM no escribirá la actualización en la base de datos, por ahora: Si un DTM emite una solicitud de lectura de un elemento de datos que la transacción ha modificado, el DBM proporcionar el valor actualizado proveniente del espacio privado de trabajo. Todo esto es sencillo. Sin embargo, cuando la transacción termina y el DTM solicita acciones para comprometer la información, aparece una complicación.

Suponga que tres DBM proponen modificaciones en una transacción particular. ¿En

el proceso de comprometer, qué pasa si una de ellas descubre que no puede comprometer sus modificaciones? A menos que se haga algo, dos de los tres DBM actualizar n sus bases de datos. Este resultado no es aceptable.

1.1.7.2.1. Compromiso en Dos Fases

Para resolver este problema, el compromiso distribuido se subdivide en un proceso de dos etapas llamado **compromiso en dos fases.** [11]Con este método, el DTM que esté en primer término procesando una transacción envía una acción de precompromiso a todos los DBM que mantienen datos actualizados correspondientes a dicha acción. Esta acción de precompromiso informa a los DBM que la transacción ha terminado, y solicita a los DBM que respondan SI o NO, según puedan comprometer o no las modificaciones sobre los datos que almacenan. Si todos los DBM responden SI, el DTM pedirá una acción de compromiso; de lo contrario, el DBM emitir una acción de aborto para todos los DBM y volver a iniciar la transacción.

Cuando un DBM recibe una acción de precompromiso, se asegurará que puede efectuar las modificaciones (dependiendo de la forma de control de concurrencia involucrado, el DBM pudiera ser incapaz de efectuar las modificaciones -la resolución de un interbloqueo es un ejemplo). Si el DBM puede efectuar las modificaciones, hará escrituras forzadas en los registros de la bitácora indicando

[11] Observe que el vocabulario de la tecnología de base de datos incluye tanto el término compromiso en dos fases y bloqueo en dos fases. No deben ser confundidos. El compromiso en dos fases corresponde

que se puede comprometer las modificaciones, aunque tales cambios no están escritos en la base de datos. A continuación el nodo responde SÍ al DTM. Por último, cuando el DTM permite la acción de comprometer, el DBM actualiza la base de datos.

Este análisis no considera casos en los cuales el DTM o los DBM fallan después que la acción de precompromiso ha sido aceptada por el DBM. Para comprender lo que ocurre en este caso, es necesario que seamos más específicos. Considere para ello el ejemplo de la Figura 13 (adaptado de un artículo escrito por Mohan, Lindsay y Obermarck[12]). Al estudiar este ejemplo, recuerde que una transacción puede comprometerse únicamente si todos los DBM involucrados tienen la capacidad de compromiso. Un voto NO por cualquier DBM es un veto.

1.1.7.2.2. Procesamiento de Compromiso en Dos Fases Sin Falla

Cuando el DBM recibe la acción de precompromiso, determina si puede comprometer la transacción. Si lo puede hacer, escribe forzado un registro PRE-COMMIT (Precomprometer) en la bitácora y envía un mensaje YES de regreso al DTM. Si el DBM es incapaz de comprometer la transacción, escribe forzado un registro ABORT en la bitácora y envía un mensaje NO. Si envía un NO, el DBM tiene la seguridad de que la transacción se abortará, y por lo tanto puede olvidarse de ella.

---

sólo al procesamiento de bases de datos distribuidas, en tanto que el bloqueo en dos fases corresponde a procesamiento de bases de datos tanto distribuidas como no distribuidas.

Cuando el DTM ha recibido respuesta de todos los DBM, examina los votos. Si cualquiera de los nodos respondió NO, el DBM escribe forzado un registro ABORT en su bitácora, aborta la transacción, Y envía acciones ABORT a todos los DDM que votaron YES (no es necesario que envíe acciones a aquellos que enviaron NO. Estos ya decidieron que la transacción falla). Si todos los DBM votaron YES, el DTM escribe forzado un registro COMMIT (Comprometer) en su bitácora y envía acciones COMMIT a todos los DBM.

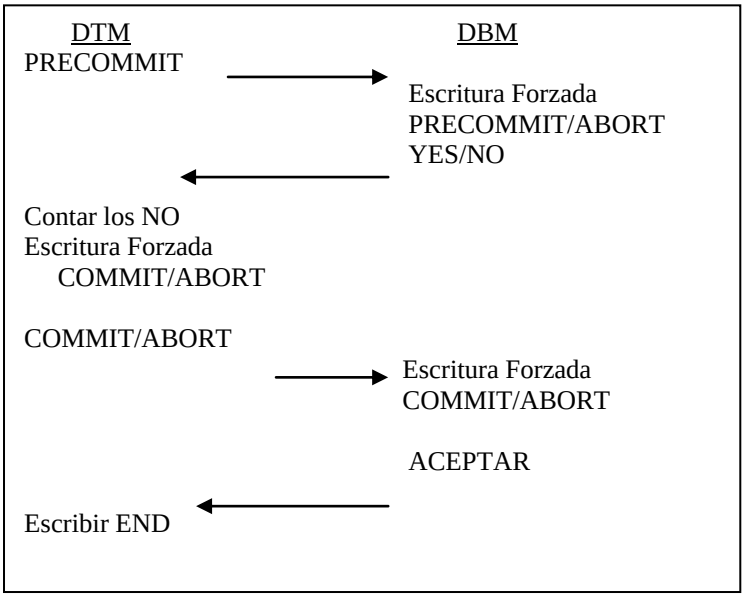


FIGURA 13
Resumen de un compromiso en dos fases

Si los DBM reciben una acción ABORT, escriben registros ABORT en sus bitácoras y abortan la transacción. Si el DBM recibe una acción COMMIT, escribe registros COMMIT en su bitácora y comprometen la transacción. En cualquiera de

[12] C. Mohan, B. Lindsay y R. Obermarck, "Transaction Management in R* Distributed Database Management System", Transactions on Database Systems, December 1986, pp. 378-396

los casos, los DBM envían un acuse de recibo a la acción ABORT o COMMIT que ha sido enviada por el DTM. Una vez que el DTM recibe los acuses de recibo (ACK) de todos los DBM, escribe un registro END en su bitácora y se olvida de la transacción.

1.1.7.2.3. Procesamiento de Compromiso en Dos Fases Bajo Falla

Ahora vea usted lo que ocurre cuando el DTM o un DBM falla en varios puntos de este proceso. Si un DBM falla, y, durante la recuperación, encuentra parte de una transacción en su bitácora, sin ningún registro PRE-COMMIT ni ABORT, puede ignorar dicha transacción. El nodo falló en la mitad de la transacción, y por lo tanto el DTM no pudo haber recibido un YES de este nodo. La transacción de seguro abortó. Pero si encuentra un registro ABORT en su bitácora, puede ignorar la transacción por razones similares.

Si el DBM encuentra un registro PRE-COMMIT sin un COMMIT, no sabrá qué hacer. Sabe que votó por YES, pero no conoce el resultado del voto de otros DBM. En este caso, el DBM deber preguntar al DTM sobre esta transacción. Una vez que reciba una respuesta (ya sea ABORT o un COMMIT) del DTM, puede procesar bajo condiciones normales. Si el DBM encuentra registros tanto PRE-COMMIT como COMMIT en su bitácora, sabrá que la transacción debería haberse comprometido, y se asegurará que así sea aplicando imágenes posteriores.

Ahora vea usted las fallas del DTM. Si el DTM falla antes de enviar cualquier

acción PRE-COMMIT, abona la transacción. Debido a que sabe que todos los DBM también abortarán (en algún momento) no necesita tomar ninguna otra acción. (A propósito, aquí aparece un caso de algo que se relaciona tanto con los aspectos de falla / recuperación como con los aspectos de control de concurrencia. Pudiera haber bloqueos sobre elementos de datos en los DBM. Tales bloqueos se conservarán en su lugar hasta que una situación de interbloqueo obligue al DBM a abortar la transacción, hasta que el DBM falle y su procesador de recuperación advierta la transacción incompleta o hasta que un programa de utilería depure la bitácora. Sería mejor para el DTM enviar acciones ABORT para una transacción como ,ésta durante una recuperación de tal forma que los DBM pudieran liberar sus bloqueos. Sin embargo, no existe nada en el algoritmo de procesamiento de fallas, que necesite de esto.)

Si el DTM encuentra un registro COMMIT en su bitácora sin un END correspondiente, falló antes de recibir todos los acuses de recibo, y periódicamente enviar acciones COMMIT a los DBM que no acusaron recibo. Una vez recibidos todos los acuses, escribir END en la bitácora. Si el DTM encuentra un registro ABORT en su bitácora sin un END correspondiente, enviar acciones ABORT a todos los DBM que no hayan acusado recibo y escribir un END en la bitácora.

Estudie la Figura 13 y asegúrese que comprende cómo este algoritmo proporciona atomicidad de transacción en relación con estos tipos de falla. La situación es algo más complicada de la descrita aquí, pero este análisis es suficiente para que comprenda el meollo de un compromiso en dos fases: si considera que las fallas

también pueden ocurrir en la mitad de la recuperación de fallas, puede usted empezar a darse cuenta de lo difícil que puede resultar la transparencia a las fallas.

1.1.7.3. Consistencia en redes divididas

El aspecto final de fallas que analizaremos se refiere al procesamiento de bases de datos distribuidas en redes divididas. Una división o partición es una subred creada cuando los nodos se desconectan en razón de una falla de un nodo o de una línea de comunicaciones. La Figura 14(a) ilustra una red distribuida, y la Figura 14(b) muestra una red se ha separado rota en dos divisiones o particiones en razón de una falla del Nodo E.

Cuando, existe una división o partición, si los datos no estén duplicados, las consecuencias, son sencillas, aunque no son deseadas. Una transacción puede operar si todos los datos que lee y que escribe quedan localizados en nodos de la partición en la cual la transacción ha sido iniciada. De lo contrario, la transacción deber esperar hasta que se recupere la red. Para el ejemplo de la Figura 14(b), las transacciones iniciadas en la Partición 1 pueden ejecutarse si los datos que leen y escriben están localizados en los nodos A, B, C o D. Las transacciones iniciadas en la Partición 2 pueden ejecutarse si los datos que leen y escriben están localizados en los Nodos F, G o H.

FIGURA 14
Divisiones de ejemplo

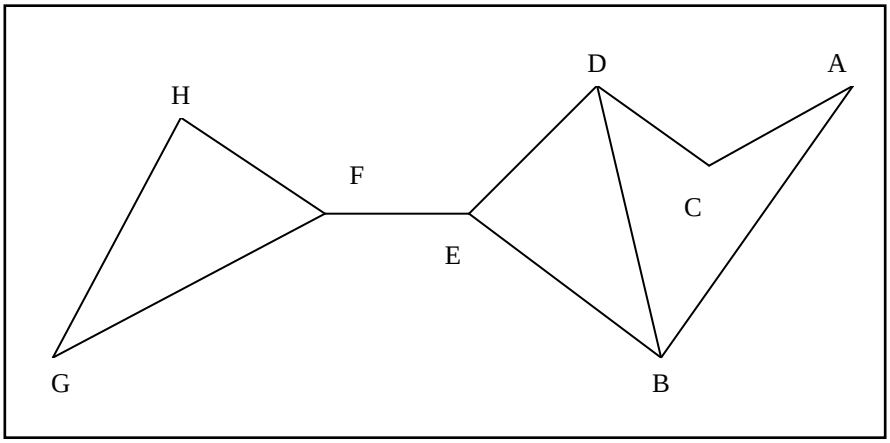


Figura 14 (a)
Red distribuida de ejemplo

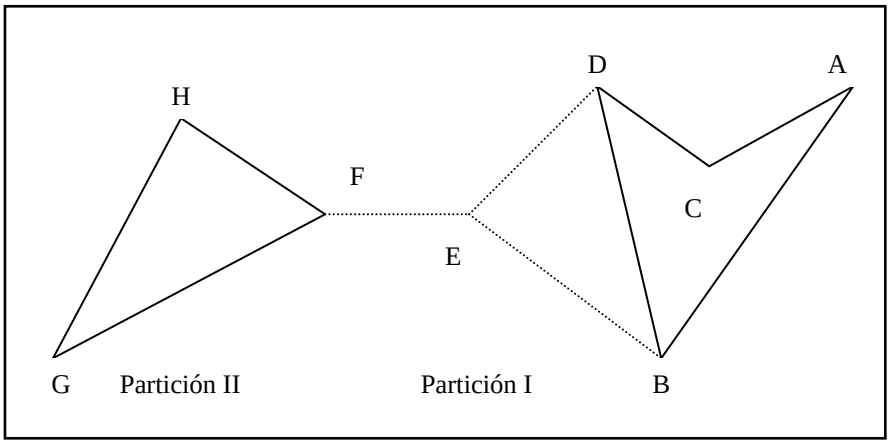


Figura 14 (b)
Particiones causadas por la falla del nodo E.

Como ya hemos dicho, en ocasiones las organizaciones deciden duplicar datos a fin de aumentar la confiabilidad y el rendimiento. Cuando los datos están duplicados, el procesamiento durante una división debe controlarse con mucha atención, y la recuperación es más difícil. Si la red de la Figura 14 da soporte a la entrada de pedidos y si los datos de inventarios están almacenados en ambas partes de la

división, podría ser posible vender el último artículo a dos clientes distintos. Adicionalmente, una vez recuperada la red, los dos conjuntos de datos de inventario procesados por separado deberán combinarse para producir un gran conjunto único y consistente.

1.1.7.4. Exactitud en comparación con disponibilidad en las redes divididas

Puede ocurrir una amplia variedad de anomalías al procesar datos duplicados en una red dividida. En la Figura 15 se ilustran dos de ellas. En el primer ejemplo, ambos Nodos A y H, en dos particiones distintas, venden collares de diamantes. Empiezan con la misma cuenta de cuatro. El Nodo A vende tres collares, y el Nodo H vende dos a un cliente y otro a un segundo cliente. En el punto de recuperación, los registros en ambos nodos indican que existe un collar restante. De hecho, se vendieron un total de seis collares cuando sólo cuatro estaban disponibles. Durante la recuperación, los Nodos A y H tienen la misma cuenta de elementos. Este ejemplo muestra que el proceso de recuperación debe de hacer más que asegurar que los elementos de datos de particiones separadas contengan el mismo valor.

FIGURA 15
Anomalías causadas por divisiones

Partición I	Partición II
(Nodo A)	(Nodo H)
Cuenta de collar de diamantes = 4	Cuenta de collar de diamantes = 4
Vende 3 collares de diamantes	Vende 2 collares de diamantes Vende 1 collar de diamantes
Cuenta de collar de diamantes = 1	Cuenta de collar de diamantes = 1

RECUPERACIÓN

Figura15 (a)
Anomalía de actualización causada por la partición

Partición I	Partición II
(Nodo A)	(Nodo H)
Límite de crédito del Cliente 100 = $5000	Límite de crédito del Cliente 100= $500O

Saldo del Cliente 100= $4500 Vende $400 en artículos al Cliente100 Saldo del Cliente 100= $4900	Saldo del Cliente 1 O0 = $4500 Vende $300 en artículos al Cliente 1 O0 Saldo del Cliente 100 = $4800

RECUPERACION

Recalcular el saldo del Cliente 100 = $5200

Figura 15 (b)
violación de limitante causada por la partición

En el segundo ejemplo, el procesamiento en dos particiones viola una limitante. El límite de créditos de un cliente es $5000. y su balance inicial es $4500. El Nodo A le vende a ese cliente $400 de mercancías, y Nodo H le vende $300. Después de la recuperación, el saldo de este cliente es $5200, lo cual excede por $200 su límite de crédito.

Como apuntan Davidson, García-Molina y Skeen, el procesamiento en redes divididas requiere de un compromiso entre los extremos de la exactitud y de la disponibilidad.[13] La exactitud absoluta es lo más fácil de proporcionar si no se permite procesamiento de datos duplicados durante una partición. Uno podría argumentar que, en cualquier caso, para qué preocuparse en duplicar los datos desde

[13] S. B. Davidson, H. García-Molina y D. Skeen, "Consistency in Partitioned Networks", Computing Surveys, Septiembre 1985, pp. 341-370.

el principio? La única razón sería un rendimiento mejorado durante los periodos en que la red no estuviera dividida.

En el otro extremo, la disponibilidad es máxima si no se colocan restricciones en el procesamiento de datos duplicados durante las particiones. Esto es factible en el caso de algunas aplicaciones. En vista de que las aerolíneas aceptan un excedente de reservaciones para sus vuelos, que diferencia habría si, de un modo ocasional, durante estas divisiones, el mismo asiento se vendiera a distintos clientes? Tanto las aerolíneas como los pasajeros prefieren contar con una alta disponibilidad de las reservaciones a tener una precisión absoluta. El sistema bancario mundial toma una actitud distinta en relación con este asunto. De nuevo volvemos a la necesidad de comprender los requisitos de los sistemas.

1.1.7.5. Estrategia para el procesamiento de datos duplicados divididos

Múltiples estrategias han sido propuestas para el procesamiento de datos duplicados en redes divididas. El artículo recién citado: Davidson, García-Molina y Skeen, proporciona una excelente investigación de muestra. Analizamos sólo una de las estrategias de mayor importancia, denominada el protocolo optimista.

Esta estrategia utiliza gráficas, iguales a las gráficas de espera de la Figura 11, para llevar un control de las dependencias entre las modificaciones de datos. Mientras está dividida la red, se permiten modificaciones sin restricción (de ahí el nombre optimista), y se mantienen gráficas de precedencia, indicando cuáles fueron las

transacciones que han leído y escrito cuáles datos. Durante la recuperación, las gráficas de todas las particiones se combinan y analizan. Si se han desarrollado inconsistencias, algunas de las transacciones son motivo de recuperación regresiva y estas modificaciones se eliminan.

La Figura 17 ilustra el ejemplo de una gráfica de precedencia combinada involucrando dos particiones (esta figura es una adaptación de la Figura 5 en otro artículo de Davidson[14]). El significado de tal figura es como sigue: T_{ij} es una transacción número j que se ejecuta en la partición número i. Las fechas punteadas indican que en una transacción escribió un elemento de datos que fue leído por la transacción subsecuente. Una flecha continua indica que una transacción leída por un elemento de datos más adelante fue modificada por otra transacción. Los elementos de datos leídos por la transacción están por encima de la línea al lado de la transacción, y los elementos escritos aparecen por debajo de dicha línea. Por lo tanto, T_{11} leyó los elementos f y g y escribió el elemento f.

Davidson probó que la base de datos es consistente si, y únicamente si no existen ciclos en la gráfica de precedencia. La gráfica de la Figura 17 contiene ciclos: fechas que vuelven a un nodo, y por lo tanto la base de datos no es consistente. Existen aquí varias inconsistencias. Una, T_{11}, lee un valor viejo del elemento de datos f. A pesar que el elemento había sido modificado por T_{23}, T_{11} no está reconociendo la modificación en razón de la división de la red.

[14] S. B. Davidson, "Optimism and Consistency in Partitioned Distributed Database Systems", Transactions on database Systems, September 1984, pp. 456-482.

La fuerza de esta estrategia es su solidez. Puede detectar las inconsistencias, y existen suficientes datos disponibles para tener la capacidad de corregir la base de datos mediante la recuperación regresiva. Sin embargo, tiene dos desventajas de importancia: deberán llevarse bitácoras de todas las actividades de lectura y de escritura y muchas, si no es que todas las escrituras en bitácora deberán ser forzadas, esto generaría un problema de rendimiento.

Segundo, una vez completada la recuperación regresiva, las transacciones comprometidas violan la atomicidad de las transacciones. Por esta estrategia, los cambios comprometidos no son necesariamente permanentes. Para los datos de la Figura 17, T_{22} quizá deba ser devuelto una vez que las salidas reales hayan sido generadas por la transacción. Acaso se les hayan prometido a los clientes collares de diamantes que no puedan ser entregados. El hecho es que más tarde recibirán una disculpa y una explicación y el argumento de que dicho collar no existía en el momento de la venta, no sirve de consuelo. A pesar de estas desventajas, la estrategia de protocolo optimista muestra cierta promesa para las redes que requieren de un procesamiento consistente y correcto a pesar de divisiones en la misma.

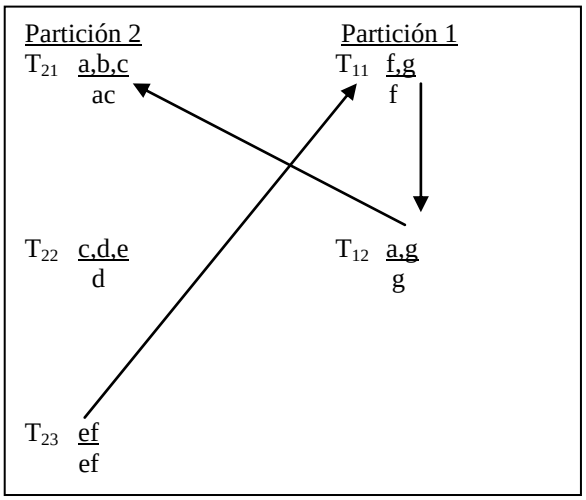


FIGURA 17
Gráfica de precedencia combinada
para dos particiones

1.2. SOBRE QUE SISTEMAS OPERATIVOS FUNCIONARÁN LAS BASES DE DATOS

1.2.1. Breve Estudio del Sistema Operativo Linux

1.2.1.1. Introducción

Linux es un sistema operativo desarrollado por Linus Torvalds, en base a minix, que es un sistema operativo didáctico, enseñado en casi todas las universidades del mundo que dictan cátedras de sistemas operativos en carreras de informática y afines. Minix es un subconjunto del poderoso sistema operativo UNIX, es decir tiene casi toda la funcionalidad de este, y Linux por ende ya no es un subconjunto de UNIX, sino que es ya una verdadera versión de UNIX con todas sus características técnicas y operativas.

1.2.1.2. Datos Técnicos

El código fuente para Linux es GNU (GNU is not UNIX[15]), es decir que es gratuito en general, y como cualquier UNIX es escrito en Lenguaje C. Cuando instalamos Linux en un PC, los códigos fuentes se ubican en el directorio "/usr/src/linux", y se puede reescribir y recompilar para fines específicos, es decir Linux es un sistema operativo que podemos ajustar a nuestras necesidades.

[15] Término bien documentado en la WebSite de Open Source para GNU www.gnu.org

Entonces ahora tratemos a Linux como un UNIX, y lo analizaremos como tal. Unix es un sistema operativo multiusuario y multitarea, se fundamenta en los procesos a base de hebras, es decir cuando se desencadena un proceso padre, los procesos ligados a este se encadenan como hebras hijas de este proceso, hablando entonces de procesos multihebras, y a su vez cada proceso hijo puede generar otras hebras más. En Linux los procesos principales se cargan como módulos del mismo kernel, y para optimizar el sistema, reescribiremos el código fuente del kernel para eliminar los módulos que nunca utilizaremos para ocupar el espacio en memoria que serían utilizados por estos procesos, en otras operaciones más provechosas para nuestro sistema; un ejemplo de estos son los módulos de las tarjetas de red. Otros módulos son cargados como *daemons*, o demonios que son programas que se cargan en memoria y receptan peticiones y ejecutan tareas, estos son cargados en el momento del arranque del sistema; un ejemplo de estos son los protocolos de red.

Linux actualmente se ha convertido en un sistema operativo de uso diario y ha ganado muchos adeptos en los últimos años; incluso se puede tener conectividad transparente con equipos Windows utilizando SAMBA que es un utilitario UNIX para conectividad con Windows.

1.2.1.3. Configuración de TCP/IP

TCP/IP se carga en Linux como un módulo del sistema operativo, en su versión IPV4, actualmente IPV6 esta en etapa experimental; se puede configurar directamente en los archivos de arranque de Linux ubicados en "/etc/rc.d". Aunque

TCP/IP se configura inicialmente al instalar Linux en nuestro computador, se puede utilizar el utilitario netconfig (para las distribuciones RedHat o basadas en esta) para cambiar los parámetros de: dirección IP principal del adaptador de red, máscara de subred, dirección IP del encaminador y dominio.

1.2.1.4. Servidor SAMBA para Linux

SAMBA es un conjunto de herramientas, que permiten utilizar el «Session Message Block (SMB) Protocol» (también llamado NetBIOS o protocolo LanManager), que es el protocolo que utilizan los servidores Windows para compartir recursos como Discos e Impresoras, por lo tanto también podemos utilizar el árbol de directorios UNIX como si fuera un disco compartido de Windows, o también una impresora conectada a una máquina UNIX como una compartida en Windows; Por otro lado, también podemos compartir desde UNIX, los discos e impresoras compartidos en servidores Windows.

El conjunto de herramientas SAMBA, se detalla a continuación:

- smbd es el demonio que ejecuta el servidor SMB en los servidores UNIX.
- nmbd es el demonio que ejecuta el "Windows Name Resolution", que será descrito a continuación.
- smbclient programa que se ejecuta desde Linux para compartir recursos en servidores Windows y otros servidores UNIX que ejecuten SAMBA, a manera de programas FTP.

- smbrun programa script sencillo, que nos ayuda a correr las aplicaciones en un servidor UNIX que ejecute SAMBA.
- smbprint programa script para imprimir en impresoras compartidas en otros servidores
- testparm programa para verificar el archivo de configuraciones para SAMBA en servidores Linux smb.conf.

1.2.1.4.1. Configuración de SAMBA

Los servicios SMB, en Linux, se configuran mediante el script smb.conf, que se encuentra ubicado en "/etc ". A continuación se muestra un pequeño ejemplo:

```
/etc/smb.conf
;
; Make sure and restart the server after making changes to this file
; /etc/rc.d/init.d/smb stop
; /etc/rc.d/init.d/smb start

[global]
; Uncomment this if you want a guest account
; guest account = nobody
log file = /var/log/samba-log.%m
lock directory = /var/lock/samba
share modes = yes
```

[homes]

comment = Home Directories

browseable = no

read only = no

create mode = 0750

[tmp]

comment = Temporary file space

path = /tmp

read only = no

public = yes

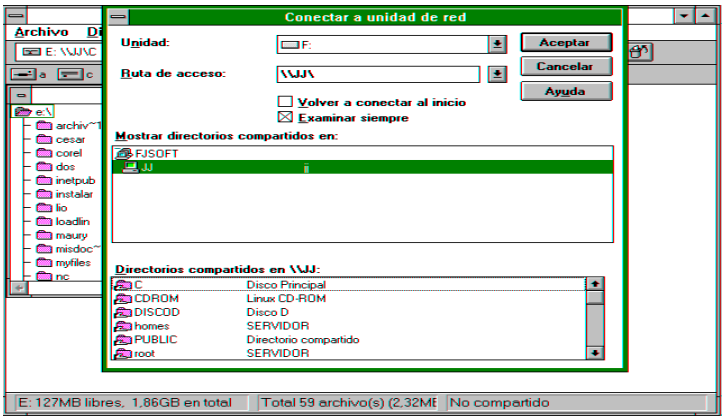
El propósito de este trabajo no es mostrar la configuración de servidores en sí, por lo que se deja al lector que profundice más en este tema[16]

1.2.1.4.2. Compartir recursos en un servidor Linux desde Windows 3.11 para Trabajo en Grupo

Para compartir un directorio, ejecutamos el Administrador de Archivos, escogemos el comando Conectarse a Unidad de Red, y vemos primeramente el

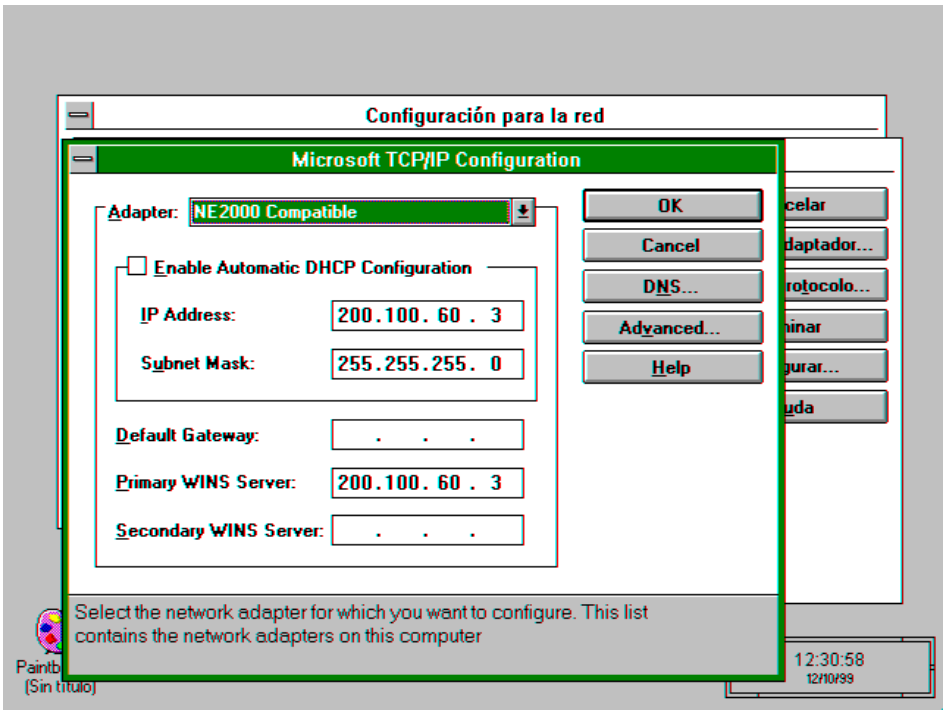
[16]HOWTO para Linux, también ejecute en Linux los siguientes comandos: man smbd, man nmbd, man smb.conf, Revista SOLO PROGRAMADORES #40, Revista PC Actual.

Nombre del Servidor, y luego los directorios compartidos en este, escogemos lo



requerido. En cambio para compartir impresoras realizamos el mismo procedimiento, pero en el Administrador de Impresoras.

***Nota:** En las pruebas con Windows 98, se tuvo problemas, ya que al momento de intentar acceder al servidor Linux, se nos pide el password para el recurso compartido IPC$.



1.2.1.5. Fortalezas y Debilidades de Linux

1.2.1.5.1. Fortalezas

- Como ya se expresó, Linux es un verdadero UNIX, con todas sus características técnicas y operativas, convirtiéndolo así en una poderosa herramienta a nuestro alcance.

- No exige requerimientos excesivos de hardware, se ha probado en el laboratorio de pruebas con una PC 80386 SX de 33MHz, con 10Mb de RAM y 500Mb de disco duro, aunque hay quienes aseguran que funciona desde 4Mb de RAM.

- Actualmente ya existe bastante información tanto en revistas especializadas en la rama, como en libros y en la Internet.

- Su código fuente como es abierto ha sido compilado para varios tipos de procesadores como: Intel x86, ALPHA, PowerPC, RISC de estaciones SUN, entre otros.

- El costo está al alcance de todos, por que no cuesta nada, únicamente tendríamos que pagar una tarifa si lo utilizamos para fines particulares

- Se puede encontrar fácilmente, podemos descargarlo de la Internet, o podemos encontrarlo en el disco compacto que se incluye en alguna revista de informática o comprarlo en un paquete con libros acerca de Linux, entre otras maneras de distribución.

- Existen empresas comerciales y organizaciones que se dedican a depurarlo y agregarle nuevas herramientas, como: Red Hat Software, S.U.S.E, Caldera Inc., entre otras. A estas se les ha sumado la colaboración de otras multinacionales como IBM o Hewlett Packard, para desarrollos conjuntos para el uso en servidores de red que se instalaría Linux. Aparte de esto en todo el mundo, existen colaboradores independientes a este proyecto que aportan nuevas ideas y herramientas, incluso el mismo Linus Torvalds, es

así que se lanzo la nueva versión del kernel 2.2 el año de 1999, desarrollado con colaboración de Alan Cox un empleado de Red Hat, incluso para el 2000 se espera la versión 2.4 del kernel (actualmente ya liberado).

- Es un sistema que ha surgido vertiginosamente en el poco tiempo que tiene de vida.

- Como también ya se dijo, lo podemos ajustar a nuestras necesidades específicas.

1.2.1.5.2. Debilidades.

- No es un sistema operativo para novatos, definitivamente hay que tener conocimiento en el manejo de UNIX para poder utilizar Linux, y no hablemos de su instalación, que necesita de una persona que tenga buenos conocimientos de computación.

- No existían paquetes comerciales para este sistema operativo, pero al parecer esto está cambiando, ya que grandes empresas de informática han puesto sus ojos en este, como Corel Corporation, Informix, entre otras.

- Para un futuro se está pensando cambiar la ideología de la licencia GNU para uno de sus principales agregados, como lo son las X Windows, es

decir que tendrán un costo, por cierto es algo que ha disgustado mucho a su creador Linus Torvalds.

1.2.2. Breve Estudio del Sistema Operativo Windows NT 4.0

1.2.2.1. Introducción

Este sistema operativo desarrollado por Microsoft desde los años 80, basado en la filosofía de los sistemas operativos Windows de Microsoft, su principal objetivo es dar la mayor facilidad al usuario final, pero en Windows NT ya se le da verdaderas características de un sistema operativo completo, como servidor y mayores privilegios de usuarios.

Al terminar de escribir estas líneas, es posible que ya este en el mercado la versión de Windows 2000.

1.2.2.2. Datos Técnicos

El núcleo de Windows NT fue escrito en el lenguaje de programación C++, es un sistema operativo multitarea y multiusuario.

Así como Windows 95 utiliza librerías DLL (Dynamic Library Linking o traducido al español como librerías de acceso dinámico en tiempo de ejecución) y las nuevas

librerías VxD (Literalmente, controlador virtual de cualquier cosa. Un componente de software de bajo nivel que gestiona un recurso concreto, tal como una pantalla de visualización o un puerto serie, posiblemente en nombre de muchas hebras concurrentes[17]. Se las encuentra como archivos con la extensión VCX, de ahí que también se las conoce con el nombre de librerías VCX) y otros programas ejecutables residentes en memoria, para que estas y estos recepten peticiones del sistema y realicen tareas, unas tan comunes como dibujar un píxel en la memoria de video, y otras complejas como resolver algoritmos para evitar colisiones de envío de paquetes de red. Windows NT utiliza un método más estructurado y efectivo, además de utilizar las anteriormente mencionadas, como los son los servicios del sistema.

En este documento se va a tener un mayor énfasis en los servicios de comunicaciones, que los encontramos como servicios de redes.

A veces para que un servicio funcione, necesita de otros servicios, como por ejemplo, el servicio de TCP/IP necesita del servicio de Servidor y viceversa, en el servicio del Internet Information Server (IIS, controla los servidores de HTTP, FTP y Gopher) necesita del servicio de TCP/IP. Es decir hay peticiones de TCP/IP al servicio de Servidor, y peticiones de IIS al servicio de TCP/IP.

El que interpreta e intercepta las peticiones al sistema es el kernel de Windows NT y

[17] Windows 95; Adrian King; McGraw Hill; primera edición; pag. 368.

decide que servicio utilizar, muchas de las llamadas al sistema son realizadas por el shell de Windows NT (explorer.exe), que se denominan llamadas del sistema del usuario, como por ejemplo cuando el usuario hace un click en el mouse, hay una librería VCX que controla las peticiones del mouse, esta acción es interpretada por el kernel y envía un mensaje al explorer.exe o el programa que este activo en ese momento, y si por ejemplo se hace un click en la estafeta del Entorno de Red, se desencadena una compleja lista de tareas a realizar, como llamar al servicio de Examinador de Equipos.

En palabras sencillas y ejemplos se trata de indicar como es el funcionamiento interno de Windows NT.

Windows NT soporta varios protocolos de red, insertados al sistema como servicios, tales como IPX/SPX y el más importante del momento como lo es TCP/IP, o protocolos de más alto nivel como el HTTP o FTP, que funcionan sobre TCP.

1.2.2.3. Configuración de TCP/IP

La forma de configurar TCP/IP en Windows NT, es muy similar a la forma como se indicó en Windows 95 o Windows para Trabajo en Grupo 3.11, por lo que no hay necesidad de entrar en explicaciones complejas.

Al instalar el servicio Internet Information Server, se pide que se lo configure, también se instalan documentos basados en HTML y Windows Help, de la forma

precisa de los pasos a seguir y los diferentes parámetros a utilizar.

1.2.2.4. Fortalezas y Debilidades de Windows NT

1.2.2.4.1. Fortalezas

- Windows NT 4.0 es un sistema operativo de configuraciones bastante intuitivas y con gran cantidad de documentación, por lo tanto no se necesita de un usuario experto para configurarlo.

- Una gran cantidad de usuarios lo usan en el mundo, por lo que existe suficiente capacitación y documentación adquiribles comercialmente y con bajo costo. Como consecuencia de esto, hay bastantes profesionales capacitados para administrar e instalar Windows NT.

- Tiene un sistema avanzado de tecnología Plug and Play.

- Su costo no es exageradamente caro, es bastante comercial por lo que se lo puede conseguir fácilmente.

- Sirve como servidor para redes Windows, bastante difundidas en nuestro medio, ya que la principal elección para instalar el sistema operativo de una PC de escritorio es Microsoft Windows como lo son Windows para Trabajo

en Grupo 3.11, Windows 95, Windows 98 y Windows NT 4.0 Workstation (para un futuro no muy lejano tendremos Windows 2000 Workstation).

- Tiene un sistema relativamente seguro en lo que se refiere a seguridades de acceso al sistema, inclusive para Internet tenemos varios paquetes comerciales de firewalls que funcionan en Windows NT.

- Existen una gran cantidad de aplicaciones comerciales que funcionan en Windows NT, y es de la casa más famosa del mundo en lo que se refiere a software para PC, es parte de la familia Microsoft Back Office.

1.2.2.4.2. Debilidades

- Windows NT 4.0 aún está propenso a caídas inesperadas del sistema, producidas por el volcamiento de la pila de memoria, produciéndonos la famosa pantallita azul, en pruebas de laboratorio encontramos este error al intentar acceder a un disco compacto de solo lectura que tuvo fallas al ser grabado en una unidad CDRW.

- Necesita de elevados recursos de memoria para ser instalado, es así que Windows NT 4.0 Server requiere de por lo menos 32 MB en RAM y como recomendado de 64 MB mínimo, y si utilizamos motores de bases de datos, con 128 MB de RAM (para la nueva versión, Windows 2000 Server se

necesita como requerimiento mínimo 64 MB de RAM y recomendado 128 MB, ya que solo para la carga del sistema ocupa 52 MB de RAM).

- Fue diseñado para procesadores Intel 80486 DX o superiores, o compatibles, también para procesadores ALPHA.

- No puede utilizar FAT de 32 bits, únicamente es compatible con particiones FAT de DOS y NTFS.

- El método de Dominios y Relaciones de Confianza, nunca acabo de convencer a los usuarios, configuradores y administradores de este difundido sistema operativo, es así que para la nueva versión Windows 2000 Microsoft opto por el método utilizado en Internet que es el DNS (Domain Name Service traducido al español Servicio de Nombres de Dominio). En el Service Pack 5.0 para Windows NT 4.0 ya se introduce al DNS.

- El costo comparado con Linux, es bastante superior en relación a las ventajas que nos da sobre este.

- Ha tenido fuertes críticas por parte del creador de Linux, Linus Torvalds.

1.3. ELEGIR UNA ESTRATEGIA PARA EL ACCESO A DATOS[18]

Cuando elija una tecnología de acceso a datos debe considerar las siguientes preguntas:

- ¿Se trata de un nuevo diseño o de una modificación a una aplicación existente que utiliza una tecnología de acceso a datos "antigua"? Para una modificación, es tentador seguir con los métodos antiguos de acceso a datos de la aplicación. Para el futuro inmediato, esto parece una decisión razonable y de costo reducido. Sin embargo, la parte negativa es la dificultad de programación a medida que la programación tiene que utilizar nuevos orígenes de datos diferentes y la eventual obsolescencia. Para un diseño nuevo, debe utilizar ADO[19].

- ¿Dónde están los datos? ¿Están en World Wide Web, en un servidor remoto o almacenados localmente en el sistema del usuario? Si los datos están almacenados localmente en el sistema del usuario, la necesidad de crear un servidor independiente para administrarlos puede parecer excesivo. Si los datos son remotos, ¿qué pasa con la administración de conexión? ¿Qué ocurrirá cuando su aplicación no se pueda conectar? ¿Debe utilizar una tecnología de acceso a datos asíncrona como ADO o RDO[20]?

[18]Tomado de Microsoft Developer Network Library para Visual Studio 6.0

[19]ADO AciveX Data Object, se definirá luego.

[20]RDO Remote Data Object, se definirá luego

- ¿Qué tecnología de acceso a datos dominan los programadores? ¿Ya tienen experiencia con ADO, RDO, DAO[21] u ODBC[22]? ¿Vale la pena el costo reducido y el esfuerzo de entrenar a toda la plantilla en el uso de ADO? Si empieza a utilizar ADO, ¿puede anticipar una reducción del costo de mantenimiento en un futuro cercano?

- ¿Requiere su aplicación tener acceso a datos de orígenes relacionales y no relacionales? ¿Tiene un proveedor OLE DB para cada uno? Si es así, debe utilizar ADO.

- ¿Tiene la intención de utilizar Microsoft® Transaction Server (MTS)? Si este es el caso, debe elegir una de las tecnologías de acceso a datos que se puedan ejecutar en el servidor y actuar como un "administrador de recursos " (un término de MTS para un componente que implementa su conjunto de interfaces de administrador de recursos). Por ejemplo ADO, RDO y ODBC pueden actuar como administradores de recursos de MTS. La interfaz de DAO no puede actuar como un administrador de recursos. También debe considerar si el componente debe ofrecer seguridad para subprocesos (como en ADO y RDO), ya que éste es un requisito para la mayoría de los componentes administrados por MTS si espera rendimiento y uso de recursos razonables.

[21]DAO Data Access Object, se definirá luego

[22]ODBC Open DataBase Connectivity

- ¿Ya utilizan todas las aplicaciones la API de ODBC?. Si continúa con ODBC, ¿cómo puede tener acceso su aplicación a otros tipos de orígenes de datos en el futuro?

Puede utilizar las diferenciadas tecnologías de acceso a datos para implementar muchas estrategias de acceso a datos y de comunicación de aplicaciones. Se muestran en la tabla siguiente.

CUADRO DE COMPARATIVO DE METODOS DE ACCESO A DATOS		
Su mejor elección es □	Si su aplicación requiere □	Comentarios
ADO	Datos de grandes sistemas o comunicaciones entre programas.	Con Microsoft® SNA Server puede configurar proveedores de datos OLE DB para orígenes de datos de grandes sistemas, como archivos VSAM, CICS, IMS y AS/400.
	Reingeniería.	Para aplicaciones existentes, debe considerar la posibilidad de volver a programar con ADO. Como alternativa, puede continuar con los métodos de acceso a datos anteriores.

CUADRO DE COMPARATIVO DE METODOS DE ACCESO A DATOS		
	Nueva programación.	Para cualquier programación nueva debe utilizar la tecnología de acceso a datos ADO de Microsoft.
	Acceso uniforme a varios orígenes de datos y varios tipos de datos.	ADO es una interfaz común para todos los requisitos de acceso a datos.
	Programación rápida.	ADO ayuda a minimizar los costos de programación porque es uniforme, coherente y fácil de utilizar. Puede entrenar a sus programadores una sola vez y aprovechar las ventajas.
	Alto rendimiento.	ADO es muy rápido.
	Páginas Active Server (ASP) de Internet Information Server (IIS).	Si su aplicación utiliza IIS con ASP para generar HTML independiente del explorador desde bases de datos, debe utilizar ADO.
OLE DB	Acceso a archivos personalizado.	Puede programar proveedores de datos OLE DB personalizados para prácticamente cualquier origen de datos. A continuación, puede usar ADO como la tecnología de acceso a datos.

CUADRO DE COMPARATIVO DE METODOS DE ACCESO A DATOS		
RDO	Acceso rápido a datos ODBC existentes.	RDO es muy rápido.
ODBCDirect	Acceso a datos ODBC.	ODBCDirect proporciona una mejora de rendimiento con respecto a tecnología de acceso a datos DAO, más antigua.
DAO	Mejoras en el acceso a datos de DAO existente.	DAO proporciona un modelo de programación coherente para situaciones donde algunos de los servicios de acceso a datos se deben proporcionar con Microsoft Jet. Si ya tiene gran cantidad de código DAO y desea pasar por alto las ventajas de diseño, programación y rendimiento proporcionadas por ADO, no tiene que modificarlo.
	Ejecución en un entorno de 16 bits.	DAO es la única elección.
ODBC	Acceso rápido a datos ODBC existentes.	Si desea programar y mantener código complejo con la API de ODBC API, ésta es una buena elección.

1.3.1. Acceso a Datos Mediante Activex Data Objects (ADO)[23]

ActiveX Data Objects (ADO) se ha diseñado como una interfaz a nivel de aplicación fácil de usar para cualquier proveedor de datos OLE DB, incluyendo bases de datos relacionales y no relacionales, sistemas de correo electrónico y de archivos, texto y gráficos, y objetos empresariales personalizados, así como orígenes de datos ODBC existentes. Prácticamente todos los datos disponibles en la empresa están disponibles mediante el uso de la tecnología de acceso a datos de ADO.

ADO es fácil de utilizar, independiente del lenguaje, se implementa con una pequeña huella, utiliza un tráfico de red mínimo y posee menos capas entre la aplicación cliente y el origen de datos (de manera a proporcionar un acceso a datos ligero y de alto rendimiento).

Las características generales de ADO son:

- Fácil de utilizar.
- Alto rendimiento.
- Control de cursores mediante programación.
- Tipos complejos de cursores, incluyendo cursores de proceso por lotes y

[23]Tomado de Microsoft Developer Network Library para Visual Studio 6.0

cursores de cliente y servidor.

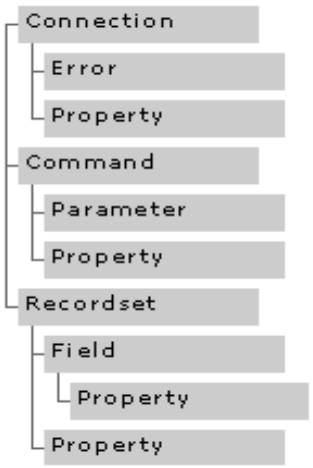
- Capacidad de devolver múltiples conjuntos de resultados desde una única consulta.
- Ejecución de consulta síncrona, asíncrona o controlada por eventos.
- Objetos reutilizables y de propiedades modificables.
- Administración avanzada de caché de recordsets.
- Flexibilidad: funciona con tecnologías existentes de base de datos y con todos los proveedores OLE DB.
- Excelente detección de errores.

La semántica sencilla de ADO y su aplicación universal implica mínimo entrenamiento de los programadores, programación rápida de aplicaciones y bajo costo de mantenimiento.

1.3.1.1. Descripción del modelo de objetos de ADO

El modelo de objetos de ADO define una colección de objetos programables que admiten el Modelo de objetos componentes (COM) y Automatización de OLE para aprovechar la eficaz tecnología denominada OLE DB. El modelo de objetos de ADO (cuando se compara con otros objetos de acceso a datos como RDO o DAO) es más plano (tiene menos objetos) y más sencillo de utilizar.

El dibujo siguiente muestra el modelo de objetos de ADO.



En el modelo de objetos de ADO hay siete objetos, como se describe en la lista siguiente.

- Objeto Command Mantiene información acerca de un comando, como una cadena de consulta, definiciones de parámetro, etcétera. Puede ejecutar una cadena de comando en un objeto Connection de una cadena de consulta como parte de la apertura de un objeto Recordset, sin definir un objeto Command. El objeto Command es útil cuando desee definir parámetros de consultas o ejecutar procedimientos almacenados que devuelva parámetros de resultados. Los objetos Command admiten varias propiedades para describir el tipo y el propósito de la consulta y para ayudar a ADO a optimizar la operación.

- Objeto Connection Mantiene información de conexión, como el tipo de cursor,

la cadena de conexión, el tiempo de espera de la consulta, el tiempo de espera de la conexión y la base de datos predeterminada.

- Objeto Error Contiene información ampliada acerca de condiciones de error producida por el proveedor de datos. Como una única instrucción puede generar dos o más errores, la colección Errors puede contener más de un objeto Error al mismo tiempo.

- Objeto Field Contiene información acerca de una única columna de datos de un recordset. El objeto Recordset utiliza la colección Fields para que contenga a todos sus objetos Field. Esta información de Field incluye el tipo de datos, la precisión y la escala numérica.

- Objeto Parameter Es un parámetro individual asociado a un objeto Command. El objeto Command utiliza la colección Parameters para que contenga a todos sus objetos Parameter. Los objetos Parameter de ADO se pueden generar automáticamente enviando consultas a la base de datos. Sin embargo, también puede generar esta colección mediante programación para mejorar el rendimiento en tiempo de ejecución.

- Objeto Property Una característica definida por el proveedor de un objeto ADO. Los objetos ADO tienen dos tipos de características: integradas y dinámicas. Las

propiedades integradas son las que están implementadas en ADO y se encuentran disponibles para cualquier objeto nuevo de ADO. Las propiedades dinámicas están definidas por el proveedor de datos subyacente y aparecen en la colección Properties para el objeto de ADO apropiado. Por ejemplo, una propiedad puede indicar si un objeto Recordset admite transacciones o actualizaciones. Ésta es una de las principales características de ADO, que permite al proveedor de servicios de ADO presentar interfaces especiales.

- Objeto Recordset Un conjunto de filas devueltas de una consulta, que incluye un cursor en las filas. Puede abrir un objeto Recordset (es decir, ejecutar una consulta) sin tener que abrir de forma explícita un objeto Connection. Sin embargo, si crea un objeto Connection en primer lugar, puede abrir múltiples objetos Recordset en la misma conexión.

1.3.1.2. Acceso a datos mediante ADO

ADO es la interfaz de datos individual que necesita para todas las soluciones para Web y cliente-servidor basadas en acceso a datos. Uno de los puntos fuertes de ADO es que puede exponer y utilizar las propiedades únicas de cada proveedor de datos. Independientemente del origen de datos utilizado, ADO es totalmente flexible y adaptable a los requisitos de acceso a datos de su aplicación.

ADO (como RDO) también incluye una biblioteca de cursores de proceso por lotes

que admite actualizaciones optimistas por lotes. Con las actualizaciones por lotes, puede crear un conjunto de resultados, modificar los datos si es necesario y realizar todas las modificaciones sucesivas con el método de actualización por lotes. Esto reduce la sobrecarga del servidor y la red y mejora el rendimiento.

Una característica importante del uso de ADO es la disponibilidad de la administración avanzada de caché de recordset con Remote Data Services (RDS). RDS proporciona almacenamiento de datos en caché en la estación de trabajo cliente de forma opcional. Con RDS, puede controlar fácilmente datos entre el servidor y el cliente. Por ejemplo, su aplicación puede usar un gran conjunto de resultados en el cliente. Esto reduce el número de peticiones de datos desde la aplicación cliente al servidor, mejorando así el rendimiento real y el rendimiento percibido de la aplicación cliente. Además, puede abrir y rellenar un objeto Recordset desconectado de forma asíncrona. Esto mejora el rendimiento al dejar al cliente libre para ejecutar otras tareas mientras aún se están devolviendo los registros.

Una aplicación típica basada en ADO utiliza las operaciones siguientes para tener acceso a un origen de datos.

- Crear el objeto Connection Especifica la cadena de conexión con información como el nombre del origen de datos, la identificación del usuario, la contraseña, el tiempo de espera de la conexión, la base de datos predeterminada y la

ubicación del cursor. Un objeto Connection representa una sesión única con un origen de datos. Puede incluso controlar transacciones a través del objeto Connection mediante los métodos BeginTrans, CommitTrans y RollbackTrans.

- Abrir la conexión Abre la conexión de ADO con el origen de datos.

- Ejecutar una instrucción SQL Cuando la conexión esté abierta, puede ejecutar una consulta. Puede ejecutar esta consulta de forma asíncrona y elegir también procesar el conjunto de resultados de la consulta de forma asíncrona; ADO señala al controlador del cursor para rellenar el conjunto de resultados como operación de fondo. Esto permite que su aplicación realice otros procesos sin tener que esperar.

- Utilizar el conjunto de resultados El conjunto de resultados se encuentra ahora disponible para su aplicación. En función del tipo de cursor, puede examinar y modificar los datos de la fila en el servidor o en el cliente.

- Finalizar la conexión Cancela la conexión con el origen de datos.

Aunque los objetos de ADO tienen muchas propiedades y muchos métodos, el uso de ADO es en realidad tan sencillo como parece. ADO representa el futuro de la tecnología

de acceso a datos.

1.3.1.3. Cuándo se debe utilizar ADO

ADO es la principal tecnología de acceso a datos de Microsoft. Esta tecnología junto con OLE DB forma la solución recomendada para todos los accesos a datos. Si está programando una nueva aplicación, debe utilizar sin duda ADO.

Si va a migrar a ADO, debe decidir si las características y las ventajas de ADO son suficientes para justificar el costo de convertir el software existente. El código anterior programado en RDO y DAO no se convertirá de forma automática en código de ADO. Sin embargo, cualquiera que sea la solución que haya programando anteriormente con otras estrategias de acceso a datos pueden implementarse con ADO. Con el tiempo debe cambiar a ADO.

1.3.2. Acceso a Datos Mediante Remote Data Objects (RDO)[24]

Remote Data Objects (RDO) está diseñado específicamente para tener acceso a orígenes de datos relacionales remotos y facilita el uso de ODBC sin código de aplicación complejo. RDO es un medio principal de tener acceso a bases de datos de SQL Server, Oracle o a cualquier base de datos relacional expuesta con un controlador

[24]Tomado de Microsoft Developer Network Library para Visual Studio 6.0

ODBC.

Las características generales de RDO son:

- Simplicidad (en comparación con la API de ODBC).
- Alto rendimiento con los orígenes de datos ODBC remotos.
- Control de cursores mediante programación.
- Cursores complejos, incluyendo cursores de proceso por lotes.
- Capacidad de devolver múltiples conjuntos de resultados a partir de una única consulta.
- Ejecución de consultas síncrona, asíncrona o controlada por eventos.
- Objetos reutilizables con propiedades modificables.
- Capacidad de exponer controladores ODBC subyacentes (para aquellas funciones que no controla RDO).
- Detección de errores excelente.

En comparación con la tecnología más antigua Data Access Objects (DAO), los objetos de RDO son una alternativa más pequeña, rápida y sofisticada. RDO tiene la capacidad específica de generar y ejecutar consultas en procedimientos almacenados y de controlar todo tipo de conjuntos de resultados, incluyendo aquellos generados por

múltiples procedimientos de conjuntos de resultados, aquellos que devuelven argumentos de resultados y estado de retorno y los que requieren parámetros complejos de entrada.

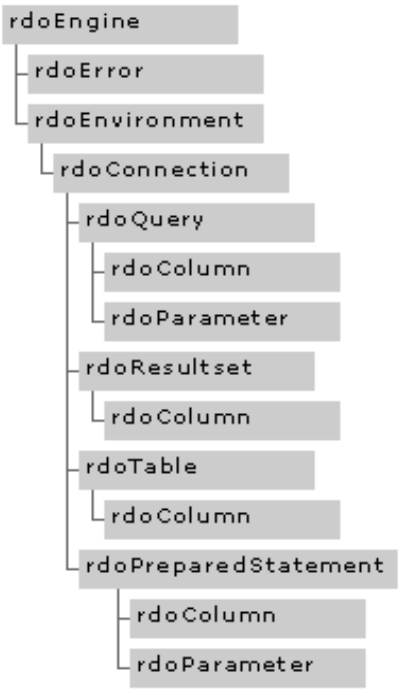
1.3.2.1. Descripción del modelo de objetos de RDO

Remote Data Objects (RDO) es una capa delgada de modelo de objetos sobre la interfaz de programación de aplicaciones (API) de ODBC. RDO depende del controlador ODBC y el motor de base de datos para gran parte de su funcionalidad. El acceso a datos mediante RDO está pensado únicamente para bases de datos relacionales de ODBC.

Nota Aunque es técnicamente posible utilizar RDO y controladores ODBC para tener acceso a bases de datos de Microsoft Jet e ISAM, requiere programación compleja y no es recomendable.

El modelo de objetos RDO incluye varios objetos, propiedades y métodos diseñados específicamente para funcionar con procedimientos almacenados y sus argumentos, parámetros de entrada y salida y valores de estado de retorno.

El dibujo siguiente muestra el modelo de objetos de RDO.



Hay diez objetos en el modelo de objetos de RDO, como se describe en la lista siguiente.

- **Objeto rdoEngine** El objeto base de RDO. Este objeto se crea automáticamente la primera vez que tiene acceso a RDO en su aplicación.

- **Objeto rdoError** Controla todos los errores y mensajes de ODBC generados por RDO. Este objeto se crea automáticamente.

- **Objeto rdoEnvironment** Define un conjunto lógico de conexiones y de

alcance de transacciones para un nombre de usuario determinado. Este objeto contiene conexiones abiertas y asignadas (pero sin abrir), proporciona mecanismos para transacciones simultáneas y define un contexto de seguridad para operaciones en la base de datos. Este objeto se crea automáticamente.

- **Objeto rdoConnection** Representa una conexión abierta con un origen de datos remoto o una conexión asignada que aún no se ha abierto.

- **Objeto rdoQuery** Una instrucción de consulta SQL con cero o más parámetros.

- **Objeto rdoColumn** Representa una columna de datos, que incluye el tipo de datos y las propiedades comunes.

- **Objeto rdoParameter** Representa un parámetro asociado a un objeto rdoQuery. Los parámetros de consulta pueden ser de entrada, de salida o ambos.

- **Objeto rdoResultset** Un conjunto de filas devueltas de una consulta.

- **Objeto rdoTable** Representa la definición de base de datos almacenada de una tabla o vista.

- **Objeto rdoPreparedStatement** El objeto rdoPreparedStatement contiene una instrucción SQL preparada y una colección de objetos rdoParameter. El objeto rdoPreparedStatement es obsoleto (aunque aún se admite). Debe sustituirlo por el objeto rdoQuery.

1.3.2.2. Acceso a Datos Mediante RDO

RDO es una manera popular de generar un acceso a datos eficaz a bases de datos relacionales ODBC. Con RDO, puede crear conjuntos de resultados sencillos sin cursor, o con cursores de proceso por lotes más complejos o con cursores de cliente. Puede limitar el número de filas devueltas y supervisar todos los mensajes y errores generados por el origen de datos remoto sin comprometer la ejecución de la consulta.

Una de las características más importantes de RDO no disponible en DAO es la capacidad de administrar consultas y procedimientos almacenados que devuelven múltiples conjunto de resultados. Esta característica se utiliza normalmente cuando hay que cargar múltiples controles (como cuadros combinados) con datos de múltiples tablas. Esta característica elimina el procesamiento redundante y la sobrecarga de tráfico de red debido a usar muchas consultas independientes.

Note Como los modelos de objetos son muy similares, puede convertir fácilmente sus aplicaciones existentes que utilizan el acceso a datos de DAO para que utilicen RDO.

La mayoría de los métodos de RDO se pueden ejecutar de forma síncrona, asíncrona o mediante el uso de procedimientos de evento para notificar al código cuándo se han completado las operaciones o cuándo están a punto de ejecutarse. Con las operaciones asíncronas y los procedimientos de eventos, su aplicación puede realizar otros trabajos mientras se ejecutan consultas de larga duración.

Aunque RDO y cualquier controlador ODBC específico pueden aprovechar funciones únicas de origen de datos exponiendo directamente las funciones de la API de ODBC, puede que otros controladores no admitan estas mismas funciones. Si su aplicación se ha diseñado para su uso con varias bases de datos, estas funciones directas deben utilizarse con cuidado o no utilizarse.

RDO tiene la capacidad de disociar y volver a asociar un objeto rdoQuery de su conexión. Esto es especialmente útil porque puede asociar la consulta con otras conexiones y aplicar por lo tanto la misma consulta a través de múltiples orígenes de datos sin tener que volver a crear las conexiones individuales.

RDO (como ADO) también incluye una biblioteca de cursores de proceso por lotes

de cliente que admiten actualizaciones por lotes optimistas. Con las actualizaciones por lotes, puede crear un conjunto de resultados, modificar los datos según sea necesario y realizar todas las modificaciones posteriores con el método de actualización por lotes. Esto mejora el rendimiento al reducir la sobrecarga del servidor, la red y ODBC.

Una aplicación basada en RDO utiliza las operaciones siguientes para tener acceso a un origen de datos.

- Establecer el controlador de entorno Identifica la ubicación de la memoria para datos globales e información de estado para las conexiones definidas.

- Abrir la conexión Especifica la cadena de conexión con información como el nombre del origen de datos, la identificación del usuario, la contraseña, la base de datos predeterminada, el nombre de red del servidor origen de datos y el nombre del controlador del origen de datos.

- Abrir el conjunto de resultados Ejecuta una consulta y crea un conjunto de resultados.

- Utilizar el conjunto de resultados El conjunto de resultados se encuentra ahora

disponible para su aplicación. En función del tipo de cursor, puede examinar y modificar los datos del elemento en el servidor o en el cliente.

- Cerrar la conexión Cancela la conexión con el origen de datos.

- Liberar el controlador de entorno Cancela los datos globales y libera toda la memoria asociada.

Con RDO puede crear código independiente de la base de datos que se adapte automáticamente a varias bases de datos de ODBC.

1.3.2.3. Cuándo se debe utilizar RDO

Si tiene una aplicación RDO que funcione bien en la actualidad, no hay razón para modificarla. Pero si necesita ampliar su aplicación para tener acceso a otros tipos de datos, debe considerar la posibilidad de reprogramarla para que utilice ADO.

Si está programando una aplicación nueva, debe considerar la utilización de ADO.

1.3.3. Acceso a Datos Mediante Data Access Objects (DAO)[25]

Data Access Objects (DAO) proporciona acceso a datos de bases de datos nativas de

[25]Tomado de Microsoft Developer Network Library para Visual Studio 6.0

Microsoft Jet (archivos .mdb), base de datos de ISAM seleccionadas y a cualquier origen de datos ODBC. Históricamente, DAO es una solución popular para utilizar orígenes de datos de tipo Microsoft® Access (.mdb) o ISAM, como Btrieve, FoxPro, Paradox y dBase.

Las características generales de DAO son:

- Dificultad de programación.
- Flexibilidad, con facilidades para tener acceso a muchos orígenes de datos diferentes.
- Rendimiento de apropiado a lento.
- Funcionalidad Lenguaje de datos dinámicos (DDL).
- Compatibilidad con cursores complejos.

Si se compara con las tecnologías ActiveX Data Objects (ADO) o Remote Data Objects (RDO), Data Access Objects (DAO) es una alternativa de acceso a datos más lenta y de menor capacidad. DAO (y su compañero, el motor de base de datos Microsoft Jet) se diseñó para controlar administrar el acceso a datos remotos de ISAM. DAO está unido al motor Microsoft Jet porque utiliza los procesadores de consultas y conjuntos de resultados de Microsoft Jet.

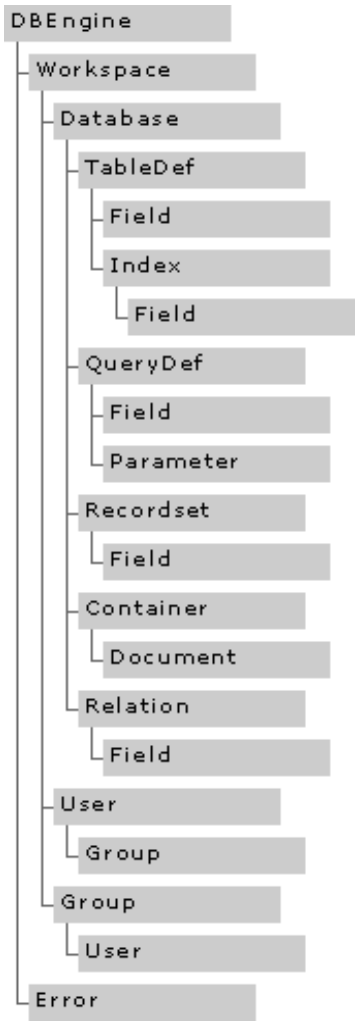
Una variación de DAO llamada ODBCDirect utiliza un modelo de objetos similar pero pasa por alto la sobrecarga del motor Microsoft Jet y proporciona un acceso más rápido y directo a bases de datos ODBC.

1.3.3.1. Descripción del modelo de objetos de DAO

El modelo de objetos de DAO es una colección de objetos que modelan la estructura de un sistema de base de datos relacional. Con las propiedades y los métodos proporcionados por los objetos de DAO, puede realizar todas las operaciones necesarias para administrar el sistema; permiten crear bases de datos, definir tablas, campos e índices, establecer relaciones entre tablas y examinar y consultar la base de datos.

El motor de base de datos Microsoft Jet traduce las operaciones sobre objetos de acceso a datos en sofisticadas operaciones en los archivos de la base de datos, controlando toda la mecánica de interfaz con las diferentes bases de datos admitidas.

El dibujo siguiente muestra el modelo de objetos de DAO.



Hay 15 objetos en el modelo de objeto DAO, como se indica a continuación.

- Objeto DBEngine Este objeto de base de DAO contiene a los demás objetos y mantiene las opciones de motor.

- Objeto Workspace Define y administra la sesión actual del usuario. Este

objeto contiene información acerca de bases de datos abiertas y proporciona mecanismos para realizar transacciones simultáneas.

- Objeto Database Representa una base de datos con al menos una conexión abierta. Puede ser una base de datos de Microsoft Jet o un origen de datos externo.

- Objeto TableDef Contiene objetos Field e Index para describir las tablas de la base de datos.

- Objeto QueryDef Representa una instrucción de consulta SQL almacenada, con cero o más parámetros, mantenida en una base de datos de tipo Microsoft Jet.

- Objeto Recordset Representa el resultado de una consulta con un cursor. DAO tiene cinco tipos de objetos Recordset: tabla, dynaset, snapshot, sólo hacia delante y dinámico.

- Objeto Contenedor Representa un conjunto determinado de objetos de una base de datos para la que puede asignar permisos en un grupo de trabajo seguro. Además de los objetos Container proporcionados por DAO, una

aplicación puede definir sus propios objetos Container (como formularios guardados, módulos, informes o macros de secuencia de comandos).

- Objeto Relation Representa una relación entre campos de tablas y consultas. Puede utilizar el objeto Relation para crear, eliminar o cambiar el tipo de relación y determinar qué tablas suministran los campos participantes, para forzar la integridad referencial o para realizar actualizaciones en cascada y eliminaciones.

- Objeto Field Representa un campo de una tabla, una consulta, un índice, una relación o un recordset. Un objeto Field contiene datos y puede utilizarlo para leer datos de un registro o escribir datos en un registro.

- Objeto Index Representa un índice en una tabla de la base de datos.

- Objeto Parameter Representa un valor asociado a un objeto QueryDef. Los parámetros de consulta pueden ser de entrada, de salida o ambos.

- Objeto Document Contiene información acerca de objetos individuales de la base de datos (como tablas, consultas o relaciones).

- Objeto User Representa una cuenta de usuario con permisos de acceso específicos.

- Objeto Group Representa un grupo de cuentas de usuario que tienen permisos de acceso comunes en una área de trabajo determinada.

- Objeto Error Contiene información acerca de un error producido durante una operación de DAO. Cuando se produce más de un error durante una única operación de DAO, cada error individual se representa mediante un objeto Error independiente.

Cada objeto Workspace tiene una colección de objetos Database. Cada objeto Database representa todos los objetos que pueden estar presentes en una base de datos. De estos objetos, los objetos Recordset son los más utilizados y proporcionan los medios para ejecutar instrucciones SQL y manipular el conjunto de resultados. Los objetos TableDef proporcionan un acceso sencillo a las tablas y a sus campos e índices. El modelo de objetos de DAO es bastante complicado debido a que proporciona mucha funcionalidad a través del motor Microsoft Jet para muchos orígenes de datos diferentes.

1.3.3.2. Acceso a datos con DAO

Hay tres categorías de bases de datos que son accesibles a través de DAO y del motor Microsoft Jet, como se describe en la lista siguiente.

- Bases de datos nativas de Microsoft Jet Estos archivos de base de datos utilizan el mismo formato que las bases de datos de Microsoft Access. Estas bases de datos se crean y manipulan directamente mediante el motor de Microsoft Jet y proporcionan máxima flexibilidad y velocidad.

- Base de datos externas Éstas son las bases de datos Indexed Sequential Access Method (ISAM) en varios formatos populares, como Btrieve, dBASE III, dBASE IV, Microsoft FoxPro versiones 2.0 y 2.5 y Paradox versiones 3.x y 4.0. Puede crear o manipular todos estos formatos de base de datos en Visual Basic. También puede tener acceso a bases de datos de archivos de texto y de hojas de cálculo de Microsoft Excel o Lotus 1-2-3.

- Bases de datos ODBC Incluyen bases de datos relacionales compatibles con el estándar ODBC, como Microsoft SQL Server.

El motor de base de datos Microsoft Jet traduce las operaciones sobre objetos DAO a operaciones sofisticadas en los mismos archivos de la base de datos, controlando todos los mecanismos de interfaz con las muchas bases de datos que admite.

Una aplicación basada en DAO utiliza las siguientes operaciones para tener acceso a un origen de datos:

- Crear el área de trabajo Define la sesión del usuario, incluyendo la identificación del usuario, la contraseña y el tipo de base de datos (como Microsoft Jet u ODBC).

- Abrir la base de datos Especifica una cadena de conexión para un objeto Workspace determinado, con información como el nombre del origen de datos y la tabla de la base de datos.

- Abrir el recordset Ejecuta una consulta SQL (con o sin parámetros) y llena el recordset.

- Utilizar el recordset El resultado de la consulta está ahora disponible para su aplicación. En función del tipo de cursor, puede examinar y modificar los datos de la fila.

- Cerrar el recordset Cancela los resultados de la consulta y cierra el recordset.

- Cerrar la base de datos Cierra la base de datos y libera la conexión.

Con DAO puede trabajar directamente con tablas e índices de ISAM. Al principio, ésta fue una ventaja que ofrecía la utilización del modelo de acceso a datos de DAO, pero ADO, con proveedores OLE DB, también puede proporcionar la misma funcionalidad.

Puede utilizar DAO para realizar operaciones DDL (Lenguaje de definición de datos) que afectan la estructura de su base de datos. Por ejemplo, puede crear, eliminar y modificar definiciones de tabla.

Como se trata de una tecnología de acceso a datos más antigua, DAO está limitada a almacenes de datos que puede controlar el motor Microsoft Jet. Si su aplicación requiere tener acceso a otros tipos de almacenes de datos, DAO no se lo puede proporcionar. Además, DAO no puede generar consultas mediante cursores de servidor. La utilización de DAO implica una gran pérdida de rendimiento debido a que utiliza el motor de base de datos Microsoft Jet.

1.3.3.3. Cuándo se debe utilizar DAO

DAO es la única tecnología de acceso a datos que admite operaciones de 16 bits. Si su aplicación debe ejecutarse en un entorno de 16 bits, DAO es la única opción.

Si su aplicación debe tener acceso a recursos nativos de Microsoft Jet y ODBC, DAO proporciona un modelo de programación coherente (aunque debe considerar la utilización de proveedores de datos OLE DB y el modelo uniforme de acceso a datos que proporciona ADO).

Si su aplicación debe tener acceso a orígenes remotos de datos, DAO y su compañero (el motor de base de datos Microsoft Jet) representan una elección deficiente porque son lentos y consumen muchos más recursos que las nuevas tecnologías de acceso a datos (como ADO o RDO).

Si tiene experiencia en el uso de DAO y tiene una gran cantidad de código DAO o sólo necesita ampliar una aplicación existente que utiliza una base de datos de tipo Microsoft Jet, DAO puede servirle. La desventaja es que si su aplicación requiere otros tipos de orígenes de datos, DAO no puede proporcionar el acceso a datos. Eventualmente, deseará aprovechar las ventajas de diseño, programación y rendimiento proporcionadas por ADO.

1.3.4. Acceso a Datos Mediante ODBCDirect[26]

ODBCDirect es un modo alternativo de Data Access Objects (DAO) que tiene acceso directamente a orígenes de datos ODBC, pasando por alto el motor Microsoft Jet y aprovechando completamente las posibilidades de procesamiento de orígenes de datos remotos.

Sugerencia ODBCDirect no es más que Remote Data Objects (RDO) con los nombres de los objetos de DAO. Cuando ODBCDirect está habilitado, DAO no carga el motor de base de datos Microsoft Jet, sino RDO 2.0.

Cuando se compara con Data Access Objects (DAO) estándar, ODBCDirect tiene varias ventajas, como se muestra a continuación.

- Mejor rendimiento
- Requisitos de recursos reducidos (al pasar por alto el motor Microsoft Jet)
- Acceso a la funcionalidad específica del servidor (como especificar valores de parámetros para procedimientos almacenados)
- Actualización optimista por lotes de cambios de recordset almacenados localmente

[26]Tomado de Microsoft Developer Network Library para Visual Studio 6.0

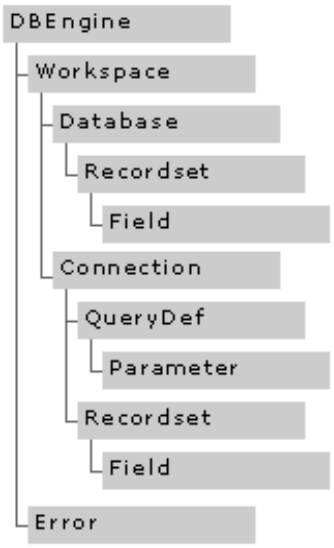
- Consultas asíncronas

ODBCDirect es una mejora con respecto a DAO estándar, que utiliza siempre el motor Microsoft Jet. Sin embargo, cuando se compara con las tecnologías ActiveX Data Objects (ADO) o Remote Data Objects (RDO), ODBCDirect es una alternativa de acceso a datos más lenta y de menor capacidad.

1.3.4.1. Descripción del modelo de objetos ODBCDirect

El modelo de objetos de ODBCDirect es esencialmente un subconjunto del modelo de objetos de Data Access Objects (DAO). Sin embargo, no contiene ninguno de los objetos específicos de bases de datos de Microsoft Jet (User, Group, Container, Document), no contiene los objetos de definición de bases de datos (TableDef y Relation) y tiene un objeto nuevo, Connection. Con este objeto, su aplicación puede abrir múltiples conexiones asíncronas con múltiples bases de datos y completar otras tareas mientras la conexión está ocupada.

El dibujo siguiente muestra el modelo de objetos de ODBCDirect.



Hay nueve objetos en el modelo de objetos de ODBCDirect, como se indica a continuación.

- Objeto DBEngine Este objeto de base de ODBCDirect contiene a todos los demás objetos del modelo y mantiene opciones del motor.

- Objeto Workspace Define y administra la sesión actual del usuario. Contiene información sobre bases de datos abiertas y proporciona mecanismos para transacciones simultáneas.

- Objeto Database Representa una base de datos ODBC con al menos una conexión abierta.

- Objeto Connection Representa una conexión con una base de datos ODBC, incluyendo el origen de datos, el nombre de usuario, la contraseña y la base de datos predeterminada.

- Objeto QueryDef Representa una instrucción almacenada de consulta SQL, con cero o más parámetros, mantenida en una base de datos ODBC.

- Objeto Recordset Representa un conjunto de resultados de una consulta con cursor. ODBCDirect tiene cinco tipos de objetos Recordset: tabla, dynaset, snapshot, sólo hacia delante y dinámico.

- Field Representa un campo de una tabla, una consulta, un índice, una relación o un recordset. Un objeto Field contiene datos y lo puede utilizar para leer datos de un registro o escribir datos en un registro.

- Parameter Representa un valor asociado a un objeto QueryDef. Los parámetros de consulta pueden ser de entrada, de salida o ambos.

- Error Contiene información acerca de un error producido durante una operación de ODBCDirect. Cuando se produce más de un error durante una única operación de ODBCDirect, cada error individual se representa con un objeto

Error independiente.

1.3.4.2. Acceso a datos mediante ODBCDirect

Cuando su aplicación utiliza ODBCDirect, en realidad está utilizando RDO. Puede preguntar con razón: "¿por qué no utilizar simplemente RDO?". La respuesta es que una conversión completa a RDO requiere muchos más cambios que los pocos ajustes sencillos que requiere la utilización de ODBCDirect.

Merece la pena tener en cuenta que ODBCDirect no es exactamente directo. Tener acceso a un origen de datos requiere una negociación con objetos de DAO, la conversión a través de una capa de traducción de ODBCDirect, una llamada ODBC de RDO y la presentación al controlador de ODBC, que finalmente traduce la petición de acceso a datos a código de motor de base de datos.

A diferencia de DAO estándar, no puede utilizar ODBCDirect para realizar operaciones DDL (Lenguaje de definición de datos) que afecten a la estructura de su base de datos, pero puede conseguir el mismo efecto mediante el uso de instrucciones SQL. Además, ODBCDirect no puede realizar combinaciones heterogéneas con tablas de múltiples orígenes de datos (uno de los puntos fuertes de DAO).

Una aplicación basada en ODBCDirect utiliza las operaciones siguientes para tener acceso a un origen de datos.

- Crear el área de trabajo Define la sesión de usuario, incluyendo la identificación de usuario, la contraseña y el tipo de base de datos.

- Abrir la conexión Especifica una cadena de conexión para una área de trabajo determinada con información como el nombre del origen de datos y la tabla de la base de datos.

- Abrir el recordset Ejecuta una consulta SQL (con o sin parámetros) y llena el recordset.

- Utilizar el recordset Los resultados de la consulta están ahora disponibles para su aplicación. En función del tipo del cursor, puede examinar y cambiar los datos de la fila.

- Cerrar el recordset Cancela los resultados de la consulta y cierra el recordset.

- Cerrar la base de datos Cierra la base de datos y libera la conexión.

Aunque ODBCDirect utiliza RDO, hay algunas características importantes que se controlan de manera diferente. Por ejemplo, con ODBCDirect debe crear un objeto Workspace nuevo para cada cursor diferente en una conexión específica. Las transacciones se coordinan a nivel del área de trabajo, no a nivel de la conexión o de la base de datos. Finalmente, aunque ODBCDirect admite operaciones asíncronas, debe solicitar que se complete (no hay devolución de llamada).

En comparación con DAO, una de las principales mejoras de ODBCDirect es la actualización optimista por lotes. Con la actualización optimista por lotes su aplicación puede almacenar datos localmente y actualizar el servidor en el modo de proceso por lotes. Esto tiende a ser muy rápido, porque reduce de forma significativa el tráfico hacia delante y hacia atrás en la red. La referencia a "optimista" tiene que ver con la suposición subyacente de que los registros modificados localmente no han sido actualizados simultáneamente por otro proceso.

ODBCDirect es una buena tecnología de acceso a datos limitada a almacenes de datos de ODBC. Si su aplicación requiere tener acceso a otros tipos de almacenes de datos, ODBCDirect no se lo puede proporcionar.

1.3.4.3. Cuándo se debe utilizar ODBCDirect

ODBCDirect es una elección aceptable si su aplicación debe ejecutar consultas o procedimientos almacenados en una base de datos relacional ODBC o si su aplicación requiere solamente las capacidades especificas de ODBC, como actualización por lotes o consultas asíncronas. Tenga en cuenta que todas las características de ODBCDirect se encuentran también disponibles en ADO.

Si tiene conocimiento del funcionamiento de ODBCDirect (o de DAO), y tiene grandes cantidades de código de ODBCDirect o sólo necesita ampliar una aplicación existente que ya lo utiliza, ODBCDirect le será de utilidad. El inconveniente es que si su aplicación requiere otros orígenes de datos que no sean de ODBC, ODBCDirect no puede proporcionar el acceso a los datos. Eventualmente, querrá aprovechar las ventajas de diseño, programación y rendimiento que proporciona ADO.

1.3.5. Acceso a Datos Mediante Conectividad Abierta De Bases De Datos (ODBC)

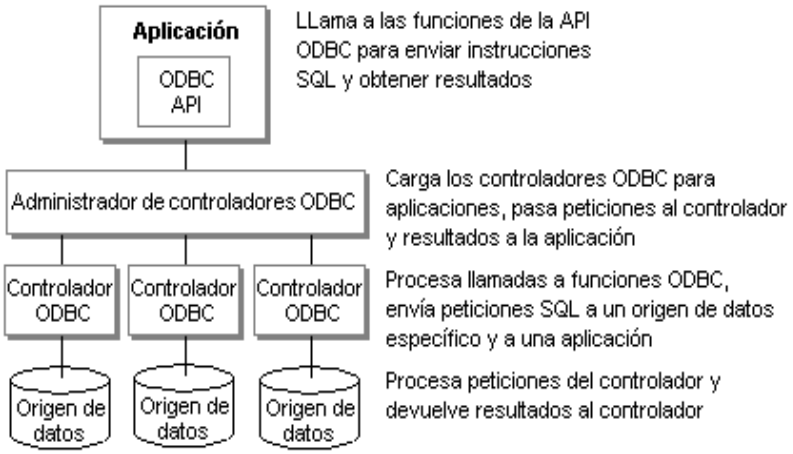
La Conectividad abierta de bases de datos (ODBC[27]) proporciona una interfaz de programación de aplicaciones (API) de conectividad universal de bases de datos que permite a las aplicaciones tener acceso a una amplia gama de bases de datos propietarias. Basada en la especificación X/Open SQL Access Group's Call Level Interface (CLI), ODBC es una manera abierta, independiente de proveedor, de tener acceso uniforme a datos almacenados en diferentes formatos y con diferentes motores de base de datos.

ODBC es la interfaz más utilizada para datos relacionales. También es muy rápida, pero puede pagar el acceso rápido con código de aplicación complejo.

Las características generales de ODBC son:

- Rendimiento muy eficaz.
- Dificultad de programación.
- Requisitos de memoria razonables.
- Compatibilidad con tecnologías existentes de base de datos.

[27]Tomado de Microsoft Developer Network Library para Visual Studio 6.0



- Portabilidad entre muchas plataformas de sistemas operativos.
- Un modelo de conexión que admite diferentes redes, sistemas de seguridad y opciones de base de datos.

Como interfaz estándar para datos relacionales, ODBC permite a su aplicación tener acceso a una gran cantidad de datos. Sin embargo, ODBC requiere que sus datos parezcan una base de datos relacional, por lo que no siempre es la mejor manera de exponer datos. Si no tiene una base de datos relacional, puede ser muy difícil escribir un controlador ODBC para exponer sus datos porque tiene que escribir un motor relacional sobre la estructura de datos existente.

1.3.5.1. Descripción de la arquitectura ODBC

- La arquitectura ODBC consta de cuatro componentes, como se describe en la lista siguiente.

- Interfaz de programación de aplicaciones (API) Llama a las funciones de ODBC para conectar con un origen de datos, enviar y recibir datos y desconectar.

- Administrador de controladores Proporciona información a una aplicación (como una lista de orígenes de datos disponibles), carga controladores dinámicamente cuando sean necesarios y proporciona comprobación de argumentos y transiciones de estados.

- Controlador Procesa llamadas de funciones de ODBC y administra todos los intercambios entre una aplicación y una base de datos relacional especifica. En caso de que sea necesario, el controlador puede traducir la sintaxis estándar SQL a SQL nativo del origen de datos de destino.

- Origen de datos Consta de los datos y su motor de base de datos asociado.

Su aplicación utiliza la API de ODBC para conectar con un origen de datos, enviar instrucciones SQL, buscar datos y desconectar. Un administrador de controladores está entre la aplicación y los controladores ODBC, decide qué controlador se debe cargar y administra las comunicaciones a medida que se llama a funciones del controlador. Finalmente, los controladores implementan las funciones de la API de ODBC para la base de datos. El dibujo siguiente muestra como interactúan estas funciones.

La arquitectura ODBC significa para su aplicación poder tener acceso a diferentes orígenes de datos ODBC, en ubicaciones diferentes, mediante las mismas llamadas de función disponibles en la API de ODBC. Cuando tenga código para tener acceso a un origen de datos relacional, el código se puede extender fácilmente para tener acceso a otros orígenes de datos.

1.3.5.2. Acceso a datos mediante ODBC

ODBC proporciona una API uniforme para tener acceso a todos los orígenes de datos relacionales. Como ODBC ofrece amplia compatibilidad con proveedores de aplicaciones y bases de datos, el resultado es una única API que proporciona toda la funcionalidad que necesitan los programadores de aplicaciones. Esta arquitectura de acceso a datos uniforme asegura la interoperabilidad y una aproximación común al acceso a datos para los muchos orígenes de datos relacionales diferentes.

Una aplicación utiliza las siguientes llamadas de función y procesos para tener acceso a un origen de datos mediante la utilización de la API de ODBC.

- Asignar controlador de entorno Identifica la ubicación en la memoria para datos globales e información de estado para las conexiones definidas.

- Asignar conexión Identifica la ubicación en la memoria para datos sobre una

conexión determinada.

- Conectar Especifica información de autorización de conexión (como el nombre del origen de datos, la identificación del usuario y la contraseña).

- Asignar instrucción Asocia una instrucción SQL a una conexión. Pueden asociarse a una conexión muchas instrucciones SQL diferentes, pero solo una cada vez.

- Ejecutar instrucción SQL Procesa la instrucción SQL con el motor de base de datos.

- Buscar conjunto de resultados Recibe los resultados de la instrucción SQL (como todas las filas, o sólo la primera, la última, la siguiente o la anterior) y también obtiene información acerca de los resultados (como el número de filas o el número de columnas).

- Liberar instrucción Elimina la instrucción de la conexión. Ahora puede asociar alguna otra instrucción SQL a la misma conexión.

- Desconectar Quita de la conexión el nombre del origen de datos e información

de autorización.

- Liberar conexión Elimina la conexión.

- Liberar controlador de entorno Elimina los datos globales y libera toda la memoria asociada.

Al programar con la API de ODBC, puede crear código independiente de la base de datos que se adapte automáticamente a una gran variedad de bases de datos. Sin embargo, existe una consideración importante al adoptar esta aproximación. Mientras cualquier controlador ODBC específico puede aprovechar las funciones de origen de datos únicas, puede que otros controladores no admitan las mismas funciones. Si su aplicación se ha diseñado para su uso a través con varias bases de datos, debe usar con cuidado estas funciones ampliadas o no usarlas.

1.3.5.3. Cuándo se debe utilizar ODBC

Varios factores influyen en la elección de la aproximación ODBC. Incluyen la necesidad de alto rendimiento, más control granular sobre la interfaz y una pequeña huella.

La API de ODBC es considerablemente más difícil de programar que las interfaces basadas en objetos, pero proporciona un control más sensible del origen de datos. A diferencia de otras tecnologías de acceso a datos (como ADO, RDO u ODBCDirect), la API de ODBC no está hecha "a prueba de balas". Aunque es bastante fácil producir errores de ODBC durante la programación, la API del ODBC proporciona un control de errores excelente con mensajes de error detallados. En general, programar, depurar y dar soporte a una aplicación basada en la API de ODBC requiere muchos conocimientos, experiencia y muchas líneas de código. Como regla general, los programadores prefieren tener acceso a datos mediante la utilización de una interfaz de objetos de más alto nivel y más sencilla, como ADO.

ODBC no es apropiado para datos no relacionales, como ISAM (Indexed Sequential Access Method) porque no tiene interfaces para buscar registros, establecer intervalos o examinar índices. ODBC no se ha diseñado para tener acceso a datos de tipo ISAM. Aunque puede utilizar el controlador ODBC de Microsoft Jet para controlar datos de ISAM y datos nativos del motor Microsoft Jet, lo que realmente pasa es que el motor de base de datos Microsoft Jet convierte los datos de ISAM a datos relacionales y proporciona funcionalidad limitada de tipo ISAM. El rendimiento en esta situación es lento debido a la capa adicional impuesta por el motor Microsoft Jet. Si su aplicación requiere un acceso muy rápido a datos ODBC existentes y desea escribir muchas líneas de código complejo (o ya tiene un lote de código ODBC disponible para reutilizarlo), ODBC es una buena elección.

1.3.6. Acceso a Datos Mediante Java Database Connectivity (JDBC)

Otra alternativa para la conectividad a Bases de Datos es Java DataBase Connectivity, que es un método usado por aplicaciones diseñadas en Lenguaje Java. Este tipo de dispositivo de software sirve para conectarse a Bases de Datos relacionales y no relacionales, además si desea diseñar una aplicación para la Web, la mejor opción de conectividad a datos es JDBC.

JDBC está basada en una completa interfase java, la cual accede a la base de datos directamente, por ejemplo, la clase getParseError.class, implementada en una interfaz de un JDBC, genera errores sintácticos de una instrucción SQL. Luego esta interfaz traduce las llamadas nativas a la base de datos, a funciones estándar de java, en las clases de java.sql.

Un driver JDBC es denominado como una dirección Internet, ejemplos de estos tenemos:

org.postgresql.Driver -> JDBC PostgreSQL

com.Sybase.jdbc.SybDriver -> JDBC Sybase Adaptive Server

Com.ibm.db2.jdbc.net.DB2Driver -> IBM DB2

oracle.jdbc.driver.OracleDriver -> Oracle 8i

Y para acceder a estas direcciones internet, lo hacemos mediante URLs, como por ejemplo un URL para acceder a PostgreSQL sería: jdbc:postgresql:<base de datos>,

entonces todo URL para acceder a JDBC se compone de: jdbc:<Nombre DBMS>:<...>.

Además la empresa Sun, nos ofrece un driver JDBC, para realizar un puente JDBC ODBC, y actualmente es la única forma para acceder a herramientas Microsoft, este driver es: sun.jdbc.odbc.JdbcOdbcDriver, y su URL es: jdbc:odbc:<nombre conexión ODBC>.

1.4 Conclusiones

En conclusión, tanto Linux como Windows NT, son sistemas operativos que pueden ser utilizados como servidores de Bases de Datos, ya que tienen las características necesarias para este propósito, luego se estudiarán DBMS que operan sobre estos sistemas operativos. Para escoger que tipo de acceso a datos se utilizará, depende de varios factores, por lo que no es prudencial decir este o cual es el mejor método.

Entre los factores que intervienen en la elección del método, el más primordial es el soporte por la herramienta de desarrollo al tipo de acceso a datos, por ejemplo Microsoft Visual Basic 5.0 utiliza DAO, entonces un programador que trabaje en esta herramienta esta forzado a utilizar DAO. Existen otros métodos de acceso a datos no discutidos en este documento, y los lectores pueden profundizar en este tema.

CAPITULO II

ESTUDIO Y EVALUACIÓN DE DBMSS QUE POSEEN LA CAPACIDAD DE DISTRIBUIR DATOS

2.1. TECNOLOGÍA SYBASE

Sybase es una empresa dedicada al desarrollo de bases de datos y herramientas para desarrollo de aplicaciones clientes servidor, y aquí se muestra un análisis de tres de sus productos: Sybase Adaptive Server Enterprise versión 11.x, Sybase SQL Anywhere versión 5.0 y SQL Remote.

Una replicación de datos mucha más compleja entre servidores se puede realizar mediante el Replication Server.

2.1.1. Sybase Adaptive Server Enterprise versión 11.x

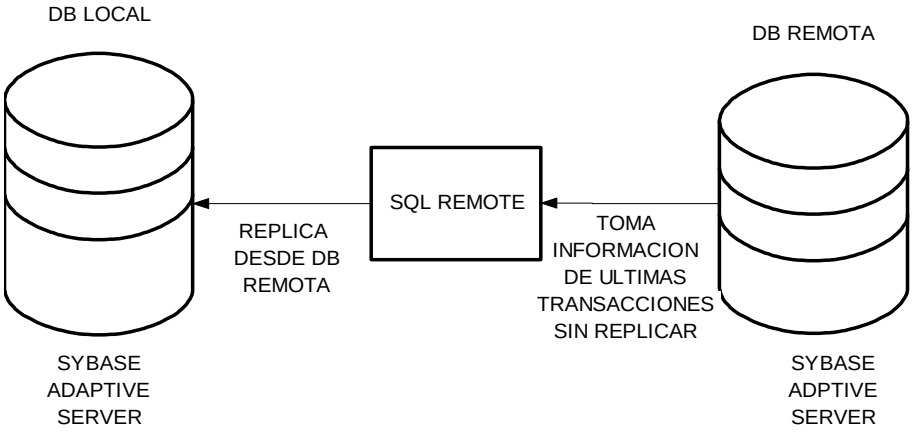
Definición

Sybase Adaptive Server Enterprise es un servidor de bases de datos SQL relacional, que opera bajo varios sistemas operativos de servidores como Sun Solaris o IBM AIX.

2.1.2. Sybase SQL Anywhere versión 5.0

Definición. Sybase SQL Anywhere es un pequeño servidor de bases de datos SQL relacional, que opera bajo varios sistemas operativos tanto de servidores como de clientes, por ejemplo Microsoft Windows 95.

2.1.3. Replicación de Datos con SQL Remote



2.1.3.1. ¿Qué es SQL Remote?

SQL Remote es una utilería diseñada para Sybase Adaptive Server y Sybase SQL Anywhere para la replicación de datos entre una base de datos consolidada y varias bases de datos remotas.

2.1.3.2. ¿Cómo opera SQL Remote?

Básicamente SQL Remote replica datos entre una base de datos consolidada y una o varias remotas.

Se denomina base de datos consolidada a la base principal, en la cual están todas las tablas y se encuentra almacenada toda la información que recoge de los servidores remotos desde los cuales se replican los datos. En cambio las bases de datos remotas pueden contener todas las tablas o un subconjunto de la principal, y tiene únicamente los datos necesarios para su correcto funcionamiento y generalmente en estas bases se recepta la información que luego se replicará.

Se debe tomar en cuenta dos conceptos que Sybase utiliza en SQL Remote, publicación y suscripción que se definirá a continuación.

Se entiende por publicación, al proceso que SQL Remote utiliza para enviar los datos, como su nombre lo dice, a publicar estos datos. Para esto se define una publicación, la cual contiene información de las columnas comunes de las tablas entre la base de datos consolidada y las bases de datos remotas. Se debe definir un usuario para que actúe como publicador en la base de datos consolidada, que es el que envía los datos, y este usuario debe ser el subscriptor de las bases de datos remotas que recibe la información de la base de datos consolidada.

Se entiende por suscripción, al proceso que SQL Remote utiliza para recibir la

información remota. Se define un usuario remoto, que a la vez funcionará como el subscriptor de la base de datos, para esto, este usuario se debe suscribir a una publicación. Este usuario como ya se mencionó, recibe información que envía el publicador de una base de datos remota, y lo mismo ocurre en la base de datos remota, pero con el cambio que recibe información que envía el publicador de la base de datos consolidada.

Para todo este proceso de replicación Sybase se basa en tablas de control las cuales llevan un registro de transacciones con todas las tablas involucradas en la replicación.

2.1.3.3. Configurar la Base de Datos para replicar datos

NOTA: En los ejemplos que se citarán a continuación se utilizará un servidor Linux en el cual está instalado el motor de base de datos Sybase Adaptive Server Enterprise versión 11.9.2 y en Windows NT el motor de base de datos Sybase SQL Anywhere versión 5.0 .

a) Con anterioridad se debe haber creado una base de datos con una estructura definida para utilizarla como la base de datos consolidada. Ej.: se creará la base de datos y una tabla definida en la siguiente instrucción SQL:

* Creando la base de datos denominada *remoto*.

```
create database remoto
```

```
go
```

* Utilizando la base de datos *remoto*.

```
use remoto
go
```

* Creando la tabla denominada *dremotos* en la base de datos.

```
create table dremotos (
        codigo   char(10)  not null,
        nombre  char(10),
        primary key (codigo))
go
```

b) En los otros servidores en los cuales se vaya a replicar datos o desde los cuales se replique los datos, se deben también haber creado las bases de datos con las mismas características de la base de datos consolidada.

c) Se creará un usuario, el cual servirá para enviar los datos a replicar en los servidores remotos. Ej.: se creará un usuario definido en la siguiente instrucción Transact SQL de Sybase, el usuario s_remoto con el login sqlremoto, en la base de datos remoto:

* Ejecutando el procedimiento *sp_addlogin* para agregar el usuario *s_remoto* en la base de datos con el login *sqlremoto*.

```
exec sp_addlogin s_remoto, sqlremoto, remoto
go
```

* Ejecutando el procedimiento *sp_role* para asignar permisos de administrador al usuario *s_remoto*.

```
exec sp_role 'grant', sa_role, s_remoto
go
```

* Utilizando la base de datos *remoto*.

```
use remoto
go
```

* Ejecutando el procedimiento *sp_adduser* para agregar el usuario *s_remoto* en la base de datos.

```
exec sp_adduser s_remoto
go
```

d) Se ejecutarán los scripts ssremote.sql que instala las tablas y procedimientos necesarios para la replicación de datos, stableq.sql para instalar la tabla en la que se registran los últimos cambios realizados en la base de datos.

* Ejecutando la utilería isql para ejecutar el script ssremote.sql

```
isql -S <nombre del servidor> -U sa -P -D remoto -i ssremote.sql
```

* Ejecutando la utilería isql para ejecutar el script stableq.sql

```
isql -S <nombre del servidor> -U sa -P -D remoto -i stableq.sql
```

e) Será necesario crear el tipo de mensaje, para enviar los mensajes necesarios para la replicación, hay cuatro tipos diferentes de mensajes: por un directorio compartido, vía ftp, por e-mail, Microsoft Mail (mapi) o Vendor Independent Messaging (vim) que es usado por Lotus Notes en algunas versiones de Lotus cc:Mail. Como ejemplo se usará el directorio compartido, y es necesario ejecutar el procedimiento que se instaló con ssremote.sql.

* Utilizando la base de datos *remoto*.

```
use remoto
go
```

* Ejecutando el nuevo procedimiento *sp_remote_type* para crear un mensaje de tipo file en la base de datos.

```
exec sp_remote_type file, remoto
go
```

f) Como siguiente paso será necesario crear un usuario que servirá de subscriptor, que tendrá la función de usuario de la base de datos remota.

* Ejecutando el procedimiento *sp_addlogin* para agregar el usuario *u_remoto* en la base de datos con el login *uremoto*.

```
exec sp_addlogin u_remoto, uremoto, remota
go
```

* Ejecutando el procedimiento *sp_adduser* para agregar el usuario *s_remoto* en la base de datos.

```
exec sp_adduser u_remoto
go
```

* Utilizando la base de datos *remoto*.

```
use remoto
go
```

* Ejecutando el nuevo procedimiento *sp_grant_remote* para asignar el usuario *u_remoto* como subscriptor de la base de datos, utilizando el tipo de mensaje file, definiendo el directorio en la variable de entorno definida en Sybase como *directorio*.

```
exec sp_grant_remote u_remoto u_remoto, file, directorio
go
```

g) Será necesario asignar el usuario que creamos anteriormente como el publicador de la base de datos. Ej.:

* Utilizando la base de datos *remoto*.

```
use remoto
```

```
go
```

* Ejecutando el nuevo procedimiento *sp_publisher* para asignar el usuario *s_remoto* como publicador de la base de datos

```
exec sp_publisher s_remoto
go
```

h) Crear una publicación en blanco. Ej. creamos la publicación en blanco con el nombre publicación, y añadimos la tabla dremotos como tabla para replicación:

* Utilizando la base de datos *remoto*.

```
use remoto
go
```

* Ejecutando el nuevo procedimiento *sp_create_publication* para crear una publicación en blanco denominada *publicación*.

```
exec sp_create_publication publication
go
```

* Ejecutando el nuevo procedimiento *sp_add_remote_table* para agregar la tabla *dremotos* para que tenga atributo de tabla remota.

```
exec sp_add_remote_table dremotos
go
```

* Ejecutando el nuevo procedimiento *sp_add_article_publication* para agregar la tabla remota *dremotos* a la publicación *publicación.*

```
exec sp_add_article publication dremotos
go
```

i) Crear la suscripción. Ej. creamos la suscripción para el usuario remoto u_remoto con el valor de rep1:

* Utilizando la base de datos *remoto*.

```
use remoto
go
```

* Ejecutando el nuevo procedimiento *sp_subscription* para agregar el usuario remoto *u_remoto* como subscriptor de la publicación *publicación*, con el valor de rep1.

```
exec sp_subscription 'create' ,  publication,  u_remoto, 'rep1'
go
```

Todo lo anteriormente se ejecutó desde la utilidad *isql*, pero lo expuesto lo podemos configurar en Sybase Central, para lo cual revisaremos los puntos a seguir.

- Creamos la base de datos denominada *remota*.
- Creamos una tabla denominada *dremotos* en la base de datos *remota*, con los campos codigo char(10) y nombre char(30), codigo como clave primaria.
- Creamos un usuario denominado *s_remoto* en la base de datos que nos servirá

como publicador.

- Instalamos las utilidades de SQL Remote para la replicación de datos en la base de datos, con los scripts ssremote.sql y stableq.sql.
- Definimos el directorio compartido en el mensaje tipo file.
- Creamos el usuario remoto denominado *u_remoto*, y le asignamos el tipo de mensaje file y definimos cual será el directorio compartido.
- Asignamos al usuario *s_remoto* que creamos con anterioridad, como publicador.
- Agregamos la tabla *dremotos* como una tabla remota.
- Creamos una publicación en blanco denominada *publicación*, a la cual le asignamos la tabla remota *dremotos*.
- Asignamos el usuario remoto *u_remoto* como subscriptor de la publicación denominada *publicación*.

2.1.3.4. Replicando datos

El proceso de replicación de datos se ejecuta mediante la utilería ssremote para Sybase Adaptive Server y dbremote para Sybase SQL Anywhere.

ssremote|dbremote -c "eng=<nombre del servidor>;dbn=<nombre de la base de datos>;uid=<usuario>;pwd=<password del usuario>"

Primeramente se debe sincronizar las bases de datos y se realiza bajo la utilería isql, para Sybase Adaptive Server se utiliza el nuevo procedimiento *sp_subscription* y

para Sybase SQL Anywhere con *SYNCHRONIZE SUBSCRIPTION*.

sp_subscription 'synchronize',<publicación>,<usuario remoto>

SYNCRONIZE SUBSCRIPTION TO <publicación>

Para crear las bases de datos remotas SQL Anywhere desde una base de datos consolidada utilizamos la utilería ssxtract para Sybase Adaptive Server y dbxtract para Sybase SQL Anywhere.

ssxtract|dbxtract -c "eng=<nombre del servidor>;dbn=<nombre de la base de datos>;uid=<usuario>;pwd=<password del usuario>" <directorio de salida de nueva base de datos> <usuario remoto>

Para crear la nueva base de datos remota, creamos una nueva base de datos SQL Anywhere, abrimos la utilería isql y llamamos al script que se creo con la utilería ssxtract o dbxtract, por lo general con el nombre *reload.sql*, aunque se puede cambiar el nombre con la opción -r <nombre del nuevo archivo>. Este proceso también se lo puede ejecutar desde Sybase Central de una manera más sencilla.

También podremos crear las bases de datos remotas siguiendo los mismos pasos que utilizamos para crear la base de datos consolidada, teniendo en cuenta que para estas bases de datos remotas, el publicador será el usuario remoto de la base de datos consolidada y el usuario remoto el publicador de la consolidada. Este es el único

método si deseamos utilizar otro Adaptive Server como base de datos remota.

Una vez este todo listo, para iniciar la replicación de datos ejecutamos las utilerías ssremote o dbremote según el caso.

2.2. POSTGRESQL

El mínimo de memoria requerido para PostgreSQL puede ser de 8MB, hay mejoras de velocidad notables cuando la memoria es de 96MB o más. La regla es que usted nunca puede tener demasiada memoria.

Cheque que usted tiene espacio del disco suficiente. Usted necesitará aproximadamente 30 Mbytes para el árbol de la fuente durante la recopilación y aproximadamente 5 Mbytes para el directorio de instalación. Un banco de datos vacío toma aproximadamente 1 Mbyte, por otra parte ellos toman sobre cinco veces la cantidad de espacio que un archivo del texto con los mismos datos. Si usted corre que la regresión prueba necesitará temporalmente un extra de 20MB.

Para verificar el espacio del disco, se lo realiza de la siguiente manera:

```
> df -k
```

Considerando los precios para un disco duro, si consigue un disco grande y rápido probablemente debe estar en sus planes una cantidad grande de datos.

2.2.1. Procedimiento de la instalación

Instalación de PostgreSQL

Para instalar o actualizar versiones anteriores de PostgreSQL:

1. Cree la cuenta PostgreSQL. Con esta cuenta correremos la Base de Datos. Para el uso de la base de Datos se debe crear una cuenta por separado, considerando algunos privilegios (se usa comúnmente la cuenta ***postgres***). Si usted no tiene acceso como ***root***, con una cuenta de usuario es suficiente para poder utilizarla

   Al correr PostgreSQL como ***root***, o cualquier otra cuenta con privilegios especiales, es un riesgo de seguridad; es mejor no hacerlo. ***Postmaste***r no se podrá ejecutar como ***root***.

2. Crear el directorio ***/usr/local/pgsql***, este y los siguientes pasos se realizarán como usuario ***roo***t. En donde se instalará la Base de Datos

   Se desempaqueta los archivos de los instaladores de versión 6.3.1 en ***/usr/src*.**

   Ejecutar el script ***./configure***

   seguido por cualquier opción que usted podría querer darlo. Para un completo la

lista de opciones, digite:

>. /configure –help

Algunas de las opciones que comúnmente se utilizan son las siguientes:

--prefix=BASEDIR

Selecciona un directorio bajo diferente para la instalación de PostgreSQL. El valor por defecto ***es /usr/local/pgsql*.**

--enable-locale

Si se quiere usar locales.

--enable-multibyte

Permite el uso de caracteres codificados. Esto es, para los idiomas como japonés, coreano, o chino.

--with-perl

Para trabajar con interfaz Perl y con extensión del lenguaje de plperl. Para utilizar esta opción, se necesita la instalación de los módulos de Perl (comúnmente se encuentra en ***/usr/lib/perl***), como se mencionó anteriormente

estos pasos se los realizará con la cuenta root .

--with-odbc

Para trabajar con ODBC.

with-tcl

Para trabajar con bibliotecas y programas que requieren Tcl/Tk, libpgtcl incluyendo, pgtclsh, y pgtksh.

Escogemos entre varios sistemas **UNIX**, **SCO UNIX**, **SPARC**, **SOLARIS, etc.**; escogemos Linux_elf, para linux, este script crea los archivos make que son necesarios para la compilación del motor de base de datos y otras utilerías.

Al realizar varias pruebas de compilación se observó que hubo un error al crear la aplicación ***psql*** (que es la principal interfase desde linux con el motor de la Base de Datos). Para corregir este error editamos **Makefile.global** buscamos la sección **RDFLAGS** y modificamos estas líneas: cambiamos **-lcurses** por **-lncurses** o únicamente le agregamos y ejecutamos **make all.** PARA LA VERSIÓN 7.2.X NO HAY NECESIDAD DE REALIZAR ESTE CAMBIO.

3. Ejecutamos

>make install

4. Modificamos el archivo ***/etc/profile*** se agrega las siguientes líneas:

   PATH=$PATH:/usr/local/pgsql/bin

   MANPATH=$MANPATH:/usr/local/pgsql/man

   PGLIB=/usr/local/pgsql/lib

   PGDATA=/usr/local/pgsql/data

   LD_LIBRARY_PATH=$PGLIB

   export PATH MANPATH PGLIB PGDATA LD_LIBRARY_PATH

5. Creamos el usuario ***postgres*** en el sistema para la administración de la Base de Datos.

6. Ejecutar el comando ***cd /usr/local/pgsql.***

   Damos permisos al directorio y a todos los subdirectorios anidados de **usr/local/pgsql** al usuario **postgres** con el comando ***chown***, por ejemplo: ***chown postgres /usr/local/pgsql/bin***.

7. Para crear el espacio para la base de Datos ejecutamos las siguientes líneas, como usuario root:

   > mkdir /usr/local/pgsql/data

   > chown postgres / usr/local/pgsql/data

> su postgres

> /usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data

La opción de -D especifica el directorio donde se almacenaran los datos. Se puede usar cualquier directorio, no tiene que ser el directorio de la instalación. Sólo el directorio que crea el usuario en este caso el directorio es ***/usr/local/pgsql/data***, y por defecto se usará este directorio, que se definió en la variable de entorno PGDATA, y es posible no usar la opción –D.

8. Arrancamos como usuario ***postgres*.**

Se ejecuta la instrucción ***initdb***, si ya no se ha realizado (crea los espacios para la base de datos), y arrancamos el motor de la base de datos como un daemon con la siguiente instrucción:

nohup postmaster –D /usr/local/pgsql/data –i –p 5432 >server.log 2>&1 &

Podemos optimizar este daemon ejecutando al inicio del sistema, poniéndolo en los archivos de inicio (en /etc/rc.d/init.d para la mayoría de los sistemas Linux).

Para poderse conectar a la base de datos en otros servidores o estaciones de trabajo se modifica el archivo ***pg_hba.conf*** que se encuentra en ***/usr/local/pgsql/data*** bajo la línea siguiente:

Al modificar en este archivo podremos dar permisos a usuarios o máquinas conectados a las base de datos.

Para comenzar a experimentar con Postgresql, arrancamos el servidor como se explico en el paso 9. Y procedemos a crear la base de Datos:

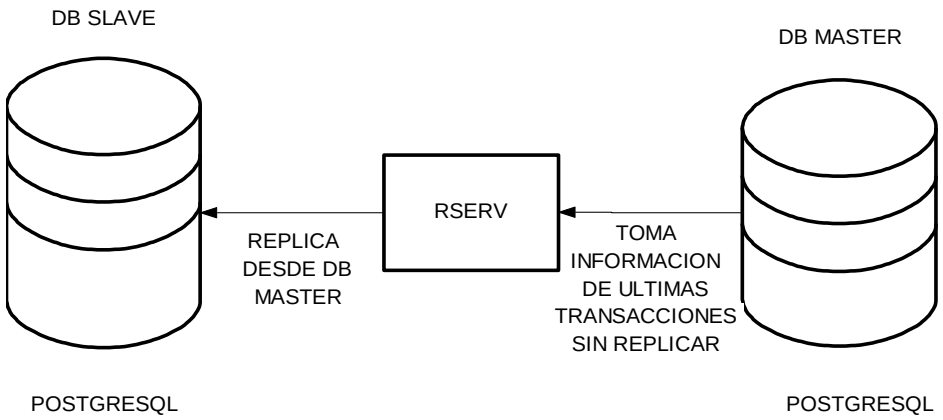
>***createdb prueba***

Presionamos enter

>***psql prueba***

Entonces nos hemos conectado a la base de datos prueba. Procedemos a utilizar los comandos SQL y comenzamos a crear tablas, procedimientos, etc.

2.2.2. Replicación de Datos con PostgreSQL



Para la replicación de datos con PostgreSQL, se tiene el paquete **rserv** versión 0.1.

Previamente se debe tener instalada el DBMS PostgreSQL versión 7.0.x o superior en un sistema linux, además de instalar la interface de PostgreSQL con perl (se debe tener instalado en el sistema el paquete perl5), y la interface tcl (opcional, para el ejemplo demostrativo que viene en el paquete rserv). También es necesario tener los fuentes de PostgreSQL (para este caso estos fuentes están ubicados en el directorio /usr/src/postgresql-7.0.3).

2.2.2.1. ¿Cómo funciona la replicación de PostgreSQL con rserv?

El paquete **rserv** interactúa con el DBMS mediante la librería para linux rserv.so (librería de enlace dinámico UNIX). Además este paquete contiene una serie de utilerías para configurar y replicar bases de datos de PostgreSQL, estas utilerías

están diseñadas en perl. Se define una base de datos maestra, a la cual se le ingresa la información, y replica a una base de datos denominada esclava. El tipo de replicación es asincrónica.

El funcionamiento interno, es similar al usado por SQLRemote de Sybase, en el cual se crean tablas de control para interceptar las transacciones realizadas en las tablas configuradas para replicar.

2.2.2.2. Instalación de rserv en un sistema Linux

El paquete **rserv** se distribuye como GNU/GPL, lo que para instalarlo, es necesario primeramente compilar los códigos fuentes, por lo que será necesario desempaquetar el contenido del archivo tgz en el que se distribuye en el directorio /usr/src.

Como usuario root, creamos el directorio /usr/local/rserv (directorio destino de los códigos fuentes), y el directorio /usr/local/rserv/bin. Luego nos ubicamos en el directorio de los fuentes (/usr/src/rserv), modificamos el archivo Makefile las líneas DESTDIR=, y le cambiamos lo que continua con /usr/local/rserv, y a la línea SRCDIR=, le cambiamos lo que continua con /usr/src/postgresql-7.0.3/src. La primera línea DESTDIR es para el directorio donde se instalará el paquete rserv, la línea SRCDIR es para el directorio donde se encuentran los fuentes de PostgreSQL. Una vez realizado esto, ejecutamos primero el comando "make" y luego el comando "make install", y el paquete rserv estará ahora listo para usar.

2.2.2.3. Configuración de la base de datos.

La base de datos maestra es un subconjunto de la base de datos esclava, configuramos en dos servidores la base de datos que utilizó en el ejemplo de Sybase.

Tabla Replica

Campo	Tipo	Longitud
codigo	caracter	10
nombre	caracter	40

clave primaria = codigo

Ejemplo: Se configuran dos servidores linux, el uno llamado linux1, y el otro linux2, en ambos deberá estar instalado el DBMS PostgreSQL versión 7.0.x o superior, además también se instalará el paquete rserv en cada uno. Creamos una base de datos replica1 en linux1, con la tabla replica descrita anteriormente, y realizamos el mismo procedimiento en linux2 con la base de datos replica2, todo esto como usuario postgres.

Para configurar la base de datos maestra se utilizan las utilerías MasterInit, que instala las tablas de control, procedimientos y triggers, y MasterAddTable que define las tablas que participarán en la replicación. La sintaxis para estos comandos es el siguiente:

MasterInit [--host=<servidor donde reside la base de datos>]
[--username=<usuario de la base de datos>]
[--password=<password para el usuario>]
<nombre base de datos maestra>

MasterAddTable [--host=<servidor donde reside la base de datos>]
[--username=<usuario de la base de datos>]
[--password=<password para el usuario>]
<nombre base de datos maestra>
<nombre de la tabla a replicar>
<campo clave>

Para el ejemplo ejecutariamos los siguientes comandos desde el directorio /usr/local/rserv/bin en el servidor linux1:

```
postgres@linux1:bin# ./MasterInit replica1

postgres@linux1:bin# ./MasterAddTable replica1 replica codigo
```

Para configurar la base de datos esclava se utilizan las utilerías SlaveInit, que instala las tablas de control, procedimientos y triggers, y SlaveAddTable que define las tablas que participarán en la replicación. La sintáxis para estos comandos es el siguiente:

SlaveInit [--host=<servidor donde reside la base de datos>]
[--username=<usuario de la base de datos>]
[--password=<password para el usuario>]
<nombre base de datos maestra>

SlaveAddTable [--host=<servidor donde reside la base de datos>]
[--username=<usuario de la base de datos>]
[--password=<password para el usuario>]
<nombre base de datos maestra>
<nombre de la tabla a replicar>
<campo clave>

Para el ejemplo ejecutariamos los siguientes comandos desde el directorio /usr/local/rserv/bin en el servidor linux2:

postgres@linux2:bin# ./SlaveInit replica1

postgres@linux2:bin# ./SlaveAddTable replica1 replica codigo

Para la replicación utilizamos la utilería Replicate, que replica datos desde la base de datos maestra hacia la esclava. La sintáxis de este comando es:

Replicate [--host=<servidor de base de datos remoto>]
[--user=<usuario de la base de datos remota>]

[--password=<password del usuario>]

[--masterhost=<servidor de la base de datos maestra>]

[--masteruser=<usuario de la base de datos maestra>]

[--masterpassword=<password del usuario>]

[--slavehost=<servidor de la base de datos esclava>]

[--slaveuser=<usuario de la base de datos esclava>]

[--slavepassword=<password del usuario>]

<base de datos maestra>

<base de datos esclava>

En el ejemplo, ingresaríamos datos a la base datos replica2, y ejecutamos el siguiente comando desde el servidor linux2:

postgres@linux2:bin# ./Replicate --masterhost=linux1 --masteruser=postgres --password=postgres replica1 replica2

Para replicar en sentido inverso, ejecutamos el mismo procedimiento cambiando el orden de configuración.

Como nota final, es necesario tener permisos de lectura/escritura en el directorio que se ejecute la utilería Replicate para el usuario que ejecute el comando.

2.2.3. Creación de triggers en PostgreSQL.

Para la creación de triggers en PostgreSQL, se debe seguir los siguientes pasos:

Primero: creación del código fuente (en lenguaje c), de la función que ejecutará el trigger.

En el código fuente en c, se debe incluir los archivos de cabecera *"executor/spi.h"* y *"commands/trigger.h"*, que tienen las definiciones de las funciones utilizadas para la manipulación de triggers en PostgreSQL, además la función deberá retornar un valor *HeapTuple* definida en trigger.h.

Segundo: compilación del código fuente.

Se utilizará el compilador de c (gcc) con las siguientes opciones:

-I[directorio de fuentes de PostgreSQL]/include -I[directorio de fuentes de PostgreSQL]
-O2 -Wall -Wmissing-prototypes -Wmissing-declarations -fpic -c

Se puede guardar esto en una variable de entorno CFLAGS, por lo que el comando para compilar el código sería el siguientes:

gcc $CFLAGS [nombre del archivo de código fuente].c

Luego se tiene que crear una librería dinámica para Linux, con la siguiente línea de comando:

```
gcc -shared -o [nombre del archivo de código fuente].so
[nombre del archivo de código fuente].o
```

Y se tiene como resultado una librería que tiene la función que ejecutará el trigger que se defina.

Tercero: añadir el procedimiento a la base de datos.

Para esto, se utiliza la instrucción *CREATE FUNCTION* de PostgreSQL, se la puede ejecutar desde la interfaz de usuario *psql*, o realizar un programa personal que ejecute la instrucción. A continuación se lista la sintáxis:

```
CREATE FUNCTION [nombre del procedimiento almacenado](...parámetros...)
RETURNS tipo_retorno
AS '[ubicación de la librería]' LANGUAGE 'c';
```

Cuarto: añadir el trigger a la base de datos.

Para añadir un trigger a la base de datos se utiliza la instrucción *CREATE TRIGGER*, que tiene la siguiente sintáxis:

```
CREATE TRIGGER [nombre del trigger]
```

AFTER | BEFORE [evento1] [OR [evento2] [OR ...]]

ON [tabla] FOR EACH ROW | STATEMENT

EXECUTE PROCEDURE [procedimiento almacenado] (...argumentos...);

2.3. TECNOLOGÍA MICROSOFT

Microsoft, al lanzar el servidor SQL Server 7.0 de Bases de Datos, como una herramienta de BackOffice, implementa la replicación de datos, para usar Microsoft Access como bases de datos móviles o replicar con otros SQL Server.

SQL Server de Microsoft es basado en SQL Server de Sybase, por lo que también apunta a los método de replicación de Sybase (pero nada comparable con el Sybase Replication Server), como en SQL Remote, por lo que la configuración es similar, aunque existe la diferencia de que el paquete de replicación esta integrado en el Motor del DBMS de SQL Server 7.0 y las bases de datos de Access solo pueden actuar como bases de datos remotas y no como bases de datos consolidadas.

Ver anexo C (Desarrollar aplicaciones móviles: comparación entre MS SQL Server 7.0 y Sybase Adaptive Server Anywhere 6.0).

2.4. TECNOLOGÍA INFORMIX

2.4.1. Introducción

Informix junto a Oracle son los proveedores más importantes de DBMS, y tienen complejos modelos de DDBMS en sus productos.

Informix Dynamic Server es un servidor de bases de datos relacional multihebra que puede operar simétricamente sobre sistemas multiprocesador y obviamente sobre sistemas monoprocesador. Dispone de las siguientes características:

- Arquitectura Cliente/Servidor
- Escalabilidad
- Alto rendimiento
- Tolerancia a fallos y alta disponibilidad
- Sistema de administración dinámico
- Consultas de datos distribuidos
- Seguridad en los servidores de bases de datos

Informix Dynamic Server implementa sistemas de replicación de datos mediante el "Enterprise Replication", herramienta que está disponible directamente desde de DBMS.

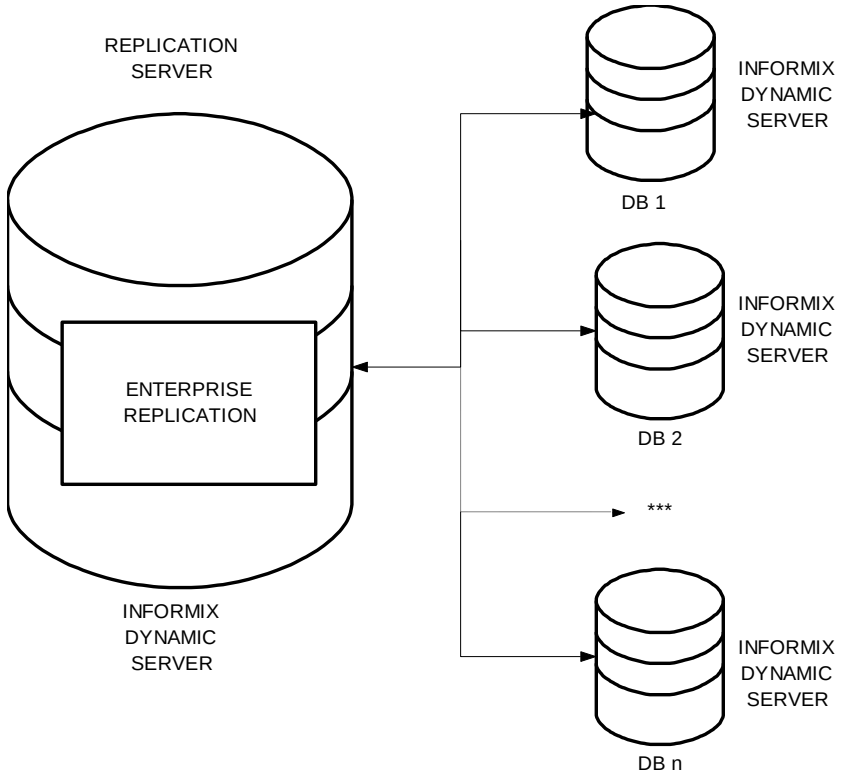
2.4.2. Características del Enterprise Replication

Algunos productos de replicación de datos usan los siguientes elementos claves satisfactoriamente:

- Alto rendimiento
- Alta disponibilidad
- Consistencia de información
- Arquitectura flexible
- Administración sencilla y centralizada

Enterprise Replication de Informix puede realizar tanto replicaciones sincrónicas y asincrónicas.

2.4.3. Configurando Sistemas de replicación con Enterprise Replication



Enterprise Replication puede configurar tres tipos de sistemas: sistema de replicación Primary-Target (como maestro – esclavo), sistema de replicación Workflow (cuando sea necesario) y sistema de replicación Update-Anywhere (replicación sincrónica).

2.4.3.1. Primary-Target

En este sistema de replicación, un servidor primario puede actualizar varios servidores remotos, esta actualización es unidireccional, las reglas del negocio son las llamadas a resolver los conflictos producidos por la actualización. Tiene los siguientes modelos: diseminación de datos, consolidación de datos y partición por

sobrecarga.

Este sistema es utilizado cuando la información ingresa por un nodo principal, y los nodos remotos se actualizan para información, más no para actualizar al primario.

2.4.3.2. Workflow

Es el típico sistema de replicación asincrónica, especialmente usado para bases de datos móviles

2.4.3.3. Update-Anywhere

Este es un complejo sistema de replicación, es sincrónico y se pueden establecer varias reglas para la repetición de datos.

2.5. CONCLUSIONES

Después de haberse realizado un breve estudio de cómo algunas DBMS, realizan los procedimientos de replicación de datos, se ha llegado a la conclusión, que la mejor manera de implementar un DTM, para soluciones pequeñas, es tomar el modelo utilizado por SQL Remote de Sybase o la implementación de SQL Server 7.0 de Microsoft, ya que estos métodos se basan en una tabla de usuario, que lleva una bitácora de las transacciones realizadas en ese servidor de Base de Datos, y esto puede ser implementado en funciones de usuario mediante triggers.

Como se detallo en el capítulo I, para aplicaciones sencillas, la mejor solución es implementar el mismo usuario del DTM, para mayor información analizar con detalle la información del anexo D, que realiza un estudio más exhaustivo sobre estos dos motores de bases de datos.

Para implementaciones más grandes, se recomendaría usar Informix u Oracle ya que tienen una avanzada implementación de un DDBMS y para pequeñas organizaciones, si tuviesen los medios económicos suficientes es bastante recomendable usar tecnología Sybase o Microsoft.

También el mercado ofrece otras soluciones de software middleware[28], o tecnologías de distribución de información (como CORBA), para poder implementar un DTM robusto.

Por último, queda acotar que el usuario final es el que toma la última palabra en la elección de la herramienta más adecuada (aunque a veces no es la solución más ideal), ya que es él quien va a trabajar en el programa, mas no el programador o asesor de sistemas.

[28] Se refiere así al software de distribución de información, y los programas de replicación de datos también entran en esta clasificación.

CAPITULO III

ALGORITMOS

3.1. INTRODUCCIÓN

3.1.1. Convenciones utilizadas en este capítulo

Para destacar la lexicografía de los algoritmos, se utilizará las siguientes convenciones:

PALABRA_RESERVADA: para destacar las palabras reservadas que luego se las aplicará a un lenguaje de programación para la implementación (Por ejemplo **SI ... ENTONCES**, en lenguaje C++ se define como if(...)). En negrillas y mayúsculas.

ACCIONES: para destacar una acción en un algoritmo, en los diferentes lenguajes pueden ser implementadas como procedimientos, funciones o comandos, o a su vez una serie de estos, si la acción lo implicase o fuese compleja (Por ejemplo ASIGNAR (variable) VALOR (valor), que asigna una variable denominada *variable*, el valor *valor*, en lenguaje C++ se define variable=valor). En mayúsculas.

variable: para destacar una variable o valor a utilizar. En minúsculas.

función(lista de parámetros): para destacar una función o un procedimiento. Esta función o procedimiento puede estar implementada en una librería, en caso de no

serlo, tendrá que ser implementada como función o procedimiento de usuario. En minúsculas cursivas.

Además, todo el texto que se escriba luego de → se tomará en cuenta como comentario, y si una línea del algoritmo termina con un punto y coma, significa que la línea del algoritmo continua en la siguiente. El texto entre «...», corresponde al nombre de una tabla de una base de datos, y «...».nombre_campo, para acceder a la variable almacenada en dicho campo de la tabla en el registro que se encuentre posicionado.

3.1.2. Replicación Sincrónica y Asincrónica

Se dice que una replicación es sincrónica, cuando los datos son actualizados inmediatamente después de realizar una transacción (mediante commit de varias fases[29]), para lo cual los servidores en los cuales residen las bases de datos deben estar sincronizados. El proceso de replicación sincrónico, es ejecutado por un proceso, y no manualmente.

Se dice que una replicación es asincrónica, cuando se actualiza en un servidor remoto, y la repetición de datos se realiza luego, la latencia de repetición depende de las reglas del negocio y de los requerimientos del usuario[30]. Este proceso puede realizarse manualmente o mediante un proceso.

[29] El método más utilizado en los DDBMS es la tecnología de two-face-commit, esta es aplicada si todas las transacciones en las bases de datos remotas son aceptadas. Para más información consultar en "Guide to Informix – Enterprise Replication" capítulo 1, pág. 1-3.

3.1.3. Operación en Línea y Fuera de Línea

Se dice que se tiene operación en línea, cuando los servidores de las bases de datos siempre están comunicados, en cambio la operación fuera de línea, significa que los servidores de bases de datos operan independientemente sin conexión, y se establece comunicación al momento de la replicación.

Según este concepto, los sistemas de replicación sincrónica nunca pueden trabajar fuera de línea.

3.1.4. Definiciones

Antes de describir los algoritmos, se debe aclarar primeramente, que son algoritmos para replicaciones asincrónicas.

Se denominará base de datos local, a la base de datos que se encuentre en el servidor de bases de datos local, y base de datos remota, como su nombre lo indica, reposa sobre un servidor de bases de datos remoto, esto depende desde el punto de vista que se mire, ya que una base de datos local, puede ser también una base de datos remota, dependiendo de la ubicación en donde el usuario se encuentre.

Entonces, la replicación se realizará desde la base de datos remota, hacia la base de datos local, y se puede aplicar las políticas de preferencia de la ejecutora de la

[30] Para más información consultar en “Guide to Informix – Enterprise Replication” capítulo 1, pág. 1-5.

replicación, es decir que la preferencia la tiene la base de datos desde la cual se replica, por lo general la base de datos remota, o la política de validación total, que valida la integridad referencial de la base de datos resultante.

Para llevar una bitácora de las transacciones realizadas en la base de datos remota, es preciso instalar una tabla de control, en la cual se almacenará dicha bitácora. Para la política de validación total es necesario instalar la tabla de control tanto en la base de datos remota como en la base de datos local.

3.1.5. Políticas de Replicación Asincrónica

Como se mencionó anteriormente, para la replicación asincrónica se definen dos políticas de actualización de la base de datos: preferencia a la ejecutora de la replicación y validación total.

La política de **preferencia de la ejecutora de la replicación**, consiste en que la base de datos desde la cual se realiza la replicación, tiene la preferencia para actualizar las transacciones que se han realizado sobre esta, a la otra base de datos. Las transacciones realizadas en la otra base de datos la cual no tiene la preferencia, no afectan a la base de datos principal. La replicación se ejecuta en un solo sentido. Por ejemplo, si en la base de datos local, insertamos un registro en la tabla “x” con clave “c” y datos “ab”, también en la base de datos remota, insertamos en la tabla “x” con clave “c” y datos “ad”, y definimos con preferencia de ejecutora de la replicación la base de datos remota, al ejecutar la replicación, los datos “ab” del registro con clave

"c" en la tabla "x" de la base de datos local, cambiaría por los datos "ad".

La política de **validación total**, consiste en validar totalmente ambas bases de datos, y tener una completa integridad referencial, tomando en cuenta la concurrencia de las transacciones realizadas en las dos bases de datos, para esto es necesario que los relojes de los servidores de bases de datos estén sincronizados. La transacción que tiene la preferencia es la última en llegar. La replicación se ejecuta en ambos sentidos.

3.2. ALGORITMOS

3.2.1. El Algoritmo Base

Toda transacción que ejecute una aplicación front-end, hacia la base de datos, debe validar si esta se ha realizado con éxito o no, para esto, en los diferentes lenguajes de las herramientas de desarrollo, se ha implementado comandos de transact sql (SQL para transacciones), los cuales son los siguientes:

INICIAR TRANSACCIÓN (*begin transaction*), inicia un bloque de transacción.

TERMINAR TRANSACCIÓN (*end transaction*), finaliza un bloque de transacción.

ACEPTAR TRANSACCIÓN (*commit*), hace que el DBMS, guarde definitivamente los cambios de la transacción en el dispositivo de base de datos (DBSpace).

ANULAR TRANSACCIÓN (*rollback)*, cancela la transacción cuando ha ocurrido un

error.

Además de esto, las tres operaciones básicas de transacciones son: INSERTAR, ACTUALIZAR y ELIMINAR registros de una tabla de la base de datos. Estos comandos están implementados en SQL con *insert*, *update* y *delete*. La sintaxis de cada uno de estos comandos depende del paquete de desarrollo que elijamos para el desarrollo de nuestra aplicación front-end, aunque llevan el estándar de SQL nivel 2 si son aplicadas como instrucciones SQL embebidas en el lenguaje.

Por lo tanto una sentencia típica para una transacción sería:

```
INICIAR TRANSACCIÓN
EJECUTAR TRANSACCIÓN
SI (falla transacción) ENTONCES
ANULAR TRANSACCIÓN
CASO CONTRARIO
ACEPTAR TRANSACCIÓN
FIN SI
FINALIZAR TRANSACCIÓN
```

Se puede implementar este algoritmo dentro de un bucle, para intentar aceptar la transacción, hasta que el error se resuelva (puede ser un error temporal de comunicación del front-end con el DBMS o back-end), o se puede cancelar mediante una acción del usuario (por ejemplo cuando la transacción se demore mucho en aceptar, el usuario pulsa la tecla ESC y se cancela la transacción), o sea un error grave (error permanente de comunicación o fallo en la aplicación front-end).

```
MIENTRAS (verdadero)
  INICIAR TRANSACCIÓN
  EJECUTAR TRANSACCIÓN
  SI (falla transacción) ENTONCES
    ANULAR TRANSACCIÓN
    FINALIZAR TRANSACCIÓN
    SI ((usuario cancela) O (error grave)) ENTONCES
      ABORTAR MIENTRAS
    FIN SI
  CASO CONTRARIO
    ACEPTAR TRANSACCIÓN
    FINALIZAR TRANSACCIÓN
    ABORTAR MIENTRAS
  FIN SI
FIN MIENTRAS
```

De esta manera se soluciona el problema de ejecutar el commit en la base de datos local, y el commit de la base de datos remota, se resuelve mediante el procedimiento de replicación.

3.2.2. Algoritmos de Replicación

3.2.2.1. La tabla de control y de tablas a replicar

La tabla de control se utilizará para llevar la bitácora de las transacciones realizadas en la base de datos. Para guardar la información en esta tabla, se utiliza un

procedimiento de los DBMS denominado **disparadores** (triggers), que se ejecuta cuando ocurre una acción en la base de datos, por lo general estos disparadores actúan al insertar, actualizar o eliminar registros en una tabla de la base de datos.

Se definirá la tabla de control con la siguiente estructura:

Tabla de control

campo	tipo	longitudprecisión	Observaciones
tabla	carácter	20	Almacena el alias de la tabla en la cual se ejecuta la transacción
valor	variable		Almacena el valor de la clave del registro que se actualiza
Valorant	variable		Almacena el valor de la clave antes de la modificación del registro
operación	carácter	1	Almacena que operación realizó: “I” insertar, “U” actualizar y “D” eliminar.
fecha	fecha-hora		Almacena la fecha y hora en la que se realizó la transacción
otros	carácter	20	Para cualquier otro uso
Semaforo	lógico	1	Variable utilizada por el procedimiento de replicación.

Se tomará en cuenta que tanto los campos valor como valorant, son de tipo carácter variable, de tal manera, que si la clave de la tabla sea de otro tipo de dato, es necesario transformar desde dicho tipo de dato a carácter.

Además también se definirá una tabla, para tener información de las tablas que participan en la replicación, la denominaremos tabla de tablas a replicar, se define con la estructura:

Tabla de tablas a replicar

campo	tipo	longitudprecisión	observaciones
Tabla	carácter	20	Alias de la tabla que participará en la replicación.
clave	variable		Clave primaria
tipodato	carácter	1	Tipo de dato de la clave
longitud	numérico	3 0	Tamaño máximo de la clave

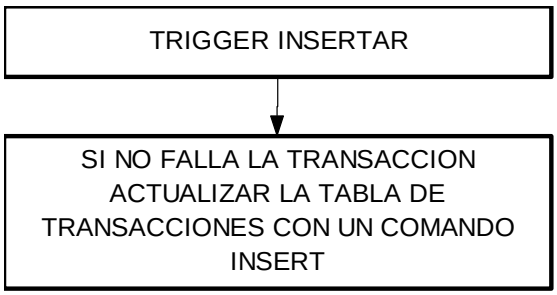
3.2.2.2. Disparadores

El disparador (trigger) es un procedimiento que utilizan las DBMS para ejecutar una acción cuando se produce un evento en la base de datos. Estos eventos son insertar, actualizar o eliminar un registro de una tabla en una base de datos. Por lo general la acción que se produce, es ejecutar código que afecta a la base de datos.

Para actualizar la bitácora que registra las transacciones realizadas en las tablas a replicar en la base de datos, se utiliza estos disparadores para los tres eventos: insertar, actualizar y eliminar. La función de este disparador es insertar un registro en la tabla de control, especificando que tipo de transacción es, en qué tabla, la hora

y fecha, y la clave principal afectada antes y después de la actualización del registro. Por lo tanto se debe definir el algoritmo para cada uno de los disparadores.

El algoritmo para el disparador insertar es:



DEFINIR (mi_nombre_clave)→nombre de la clave de la tabla

DEFINIR (mi_tabla)→variable para almacenar el nombre de la tabla

DEFINIR (mi_tipo_dato)→variable para almacenar el tipo de dato de la clave

DEFINIR (mi_clave)→variable para almacenar el valor de la clave en carácter

ASIGNAR (mi_tabla) VALOR (nombre de la tabla en la que se produce el disparador)

BUSCAR EN TABLA («tablas a replicar») POR EL CAMPO (tabla) VALOR (mi_tabla)

ASIGNAR (mi_nombre_clave) VALOR («tablas a replicar».clave)

ASIGNAR (mi_tipo_dato) VALOR *tipo_de_dato*(mi_nombre_clave)

SELECCIÓN MÚLTIPLE

SELECCIÓN mi_tipo_dato="C"→tipo carácter

ASIGNAR (mi_clave) VALOR («*valor*(mi_tabla)».*valor*(mi_nombre_clave))

SELECCIÓN mi_tipo_dato="N"→tipo numérico

ASIGNAR (mi_clave) VALOR;

numérico_a_carácter(«*valor*(mi_tabla)».*valor*(mi_nombre_clave))

SELECCIÓN mi_tipo_dato="D"→tipo fecha o fecha tiempo

ASIGNAR (mi_clave) VALOR;

fecha_a_carácter(«*valor*(mi_tabla)».*valor*(mi_nombre_clave))

SELECCIÓN NO LISTADA

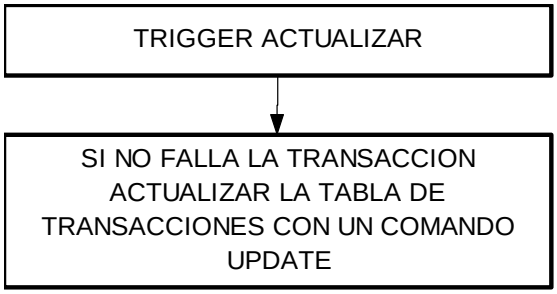
ASIGNAR (mi_clave) VALOR («*valor*(mi_tabla)».*valor*(mi_nombre_clave))

FIN SELECCIÓN MÚLTIPLE

INSERTAR EN TABLA («control») DATOS ;

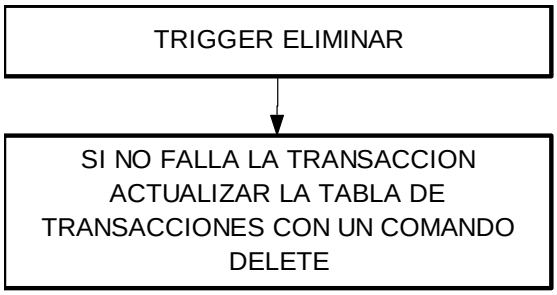
(mi_tabla,mi_clave,"","I",*fecha_tiempo_actual*(),*espacios*(20),FALSO)

El algoritmo para el disparador actualizar es:



```
DEFINIR (mi_nombre_clave)→nombre de la clave de la tabla
DEFINIR (mi_tabla)→variable para almacenar el nombre de la tabla
DEFINIR (mi_tipo_dato)→variable para almacenar el tipo de dato de la clave
DEFINIR (mi_clave)→variable para almacenar el valor de la clave en carácter
DEFINIR (mi_clave_ant)→variable para almacenar el valor de la clave anterior a la actualización
ASIGNAR (mi_tabla) VALOR (nombre de la tabla en la que se produce el disparador)
BUSCAR EN TABLA («tablas a replicar») POR EL CAMPO (tabla) VALOR (mi_tabla)
ASIGNAR (mi_nombre_clave) VALOR («tablas a replicar».clave)
ASIGNAR (mi_tipo_dato) VALOR tipo_de_dato(mi_nombre_clave)
SELECCIÓN MÚLTIPLE
  SELECCIÓN mi_tipo_dato="C"→tipo carácter
    ASIGNAR (mi_clave_ant) VALOR;
      valor_antes_actualizar(«valor(mi_tabla)».valor(mi_nombre_clave))
    ASIGNAR (mi_clave) VALOR («valor(mi_tabla)».valor(mi_nombre_clave))
  SELECCIÓN mi_tipo_dato=”N”→tipo numérico
    ASIGNAR (mi_clave_ant) VALOR numérico_a_carácter(;
      valor_antes_actualizar(«valor(mi_tabla)».valor(mi_nombre_clave)))
    ASIGNAR (mi_clave) VALOR;
      numérico_a_carácter(«valor(mi_tabla)».valor(mi_nombre_clave))
  SELECCIÓN mi_tipo_dato=”D”→tipo fecha o fecha tiempo
    ASIGNAR (mi_clave_ant) VALOR fecha_a_carácter(;
      valor_antes_actualizar(«valor(mi_tabla)».valor(mi_nombre_clave))
    ASIGNAR (mi_clave) VALOR;
      fecha_a_carácter(«valor(mi_tabla)».valor(mi_nombre_clave))
  SELECCIÓN NO LISTADA
    ASIGNAR (mi_clave_ant) VALOR;
      valor_antes_actualizar(«valor(mi_tabla)».valor(mi_nombre_clave))
    ASIGNAR (mi_clave) VALOR («valor(mi_tabla)».valor(mi_nombre_clave))
FIN SELECCIÓN MÚLTIPLE
INSERTAR EN TABLA («control») DATOS ;
  (mi_tabla,mi_clave,mi_clave_ant,”U”,fecha_tiempo_actual(),espacios(20),FALSO)
```

El algoritmo para el disparador eliminar es:



DEFINIR (mi_nombre_clave)→nombre de la clave de la tabla

DEFINIR (mi_tabla)→variable para almacenar el nombre de la tabla

DEFINIR (mi_tipo_dato)→variable para almacenar el tipo de dato de la clave

DEFINIR (mi_clave)→variable para almacenar el valor de la clave en carácter

ASIGNAR (mi_tabla) VALOR (nombre de la tabla en la que se produce el disparador)

BUSCAR EN TABLA («tablas a replicar») POR EL CAMPO (tabla) VALOR (mi_tabla)

ASIGNAR (mi_nombre_clave) VALOR («tablas a replicar».clave)

ASIGNAR (mi_tipo_dato) VALOR *tipo_de_dato*(mi_nombre_clave)

SELECCIÓN MÚLTIPLE

SELECCIÓN mi_tipo_dato="C"→tipo carácter

ASIGNAR (mi_clave) VALOR («*valor*(mi_tabla)».*valor*(mi_nombre_clave))

SELECCIÓN mi_tipo_dato="N"→tipo numérico

ASIGNAR (mi_clave) VALOR;

numérico_a_carácter(«*valor*(mi_tabla)».*valor*(mi_nombre_clave))

SELECCIÓN mi_tipo_dato="D"→tipo fecha o fecha tiempo

ASIGNAR (mi_clave) VALOR;

fecha_a_carácter(«*valor*(mi_tabla)».*valor*(mi_nombre_clave))

SELECCIÓN NO LISTADA

ASIGNAR (mi_clave) VALOR («*valor*(mi_tabla)».*valor*(mi_nombre_clave))

FIN SELECCIÓN MÚLTIPLE

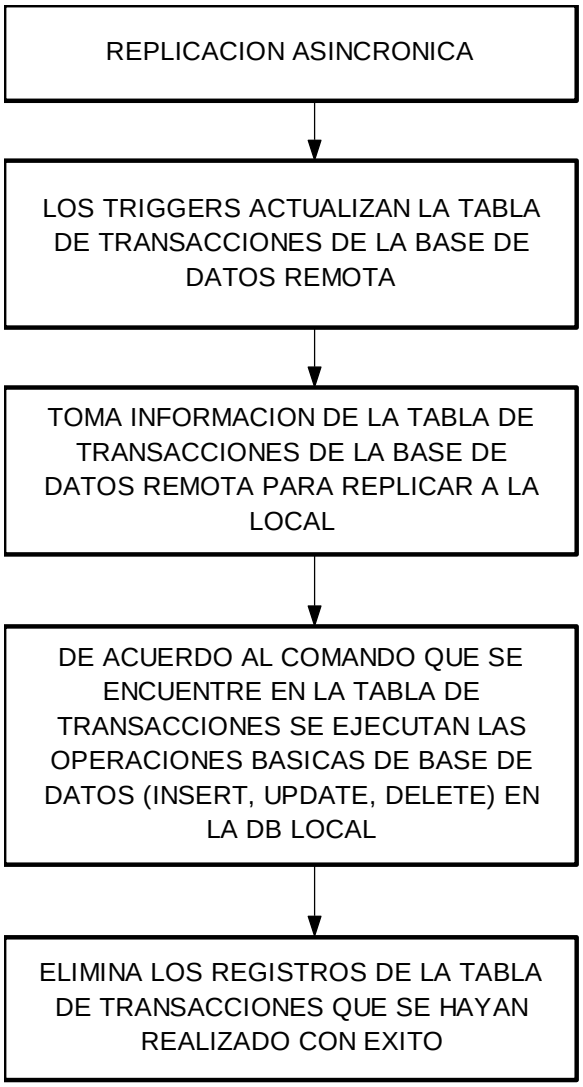
INSERTAR EN TABLA («control») DATOS ;

(mi_tabla,mi_clave,"","D",*fecha_tiempo_actual*(),*espacios*(20),FALSO)

Como se puede observar estos tres algoritmos son casi idénticos, con la distinción de que al actualizar la tabla de control, insertamos con el tipo de transacción “I” para insertar, “U” para actualizar y “D” para eliminar.

Estos algoritmos deben ser implementados en las tablas que replicaran de la base de datos. El lenguaje de implementación depende del DBMS.

3.2.2.3. Algoritmo para replicar mediante la política de preferencia de la ejecutora de la replicación



Como en el texto de esta política se detalla, las transacciones que tienen la preferencia son las de las tablas desde la base de datos que se realiza la replicación, por lo tanto, el primer paso a realizar es establecer conexión con la base de datos local y la base de datos remota. La replicación se realiza desde la base de datos remota hacia la local.

Para utilizar este algoritmo, es necesario que estén implementadas las tablas de tablas a replicar y de control y además los disparadores antes detallados en la base de datos remota.

Las transacciones a replicar son insertar, actualizar y eliminar, las cuales se implementarán en el algoritmo siguiente.

```
→Definiendo variables
DEFINIR con1,con2→identificadores de la conexión
DEFINIR ncont,ncont1→variables para almacenar número de registros de una tabla
DEFINIR pase→variable lógica para verificar las transacciones con éxito
DEFINIR cont1→contador
DEFINIR comandosql→variable para almacenar el comando sql a ejecutar

→Estableciendo conexión con las bases de datos, con1 será utilizado para la base de datos
→remota y con2 para la base de datos local
ESTABLECER CONEXIÓN CON (base de datos remota);
  ASIGNAR A IDENTIFICADOR (con1)
ESTABLECER CONEXIÓN CON (base de datos local);
  ASIGNAR A IDENTIFICADOR (con2)
```

→Crear vista temporal para la tabla «tablas a replicar»

CREAR VISTA SQL (vista_tablas_a_replicar) MEDIANTE INSTRUCCIÓN SQL;

"SELECT * FROM «tablas a replicar»";

DESDE IDENTIFICADOR DE CONEXIÓN (con1)

ACCEDER A VISTA SQL (vista_tablas_a_replicar)

IR AL PRIMER REGISTRO DE VISTA SQL (vista_tablas_a_replicar)

MIENTRAS (no se encuentre fin de vista sql vista_tablas_a_replicar)

→Bucle para examinar toda la vista sql vista_tablas_a_replicar

→Crear vista sql vista_control para examinar transacciones realizadas en la base

→de datos remota

CREAR VISTA SQL (vista_control) MEDIANTE INSTRUCCIÓN SQL;

"SELECT * FROM «control» WHERE tabla=*valor*(vista_tablas_a_replicar.tabla)";

DESDE IDENTIFICADOR DE CONEXIÓN (con1)

ACCEDER A VISTA SQL (vista_control)

IR AL PRIMER REGISTRO DE VISTA SQL (vista_control)

MIENTRAS (no se encuentre fin de vista sql vista_control)

→Bucle para ir escogiendo transacción por transacción registradas en la tabla de control

→Crear vista sql vista_registro_remoto, para extraer el registro a insertar o modificar

SI vista_tablas_a_replicar.tipodato="C" **ENTONCES**

→vista_control.valor es tipo carácter, entonces va entre comillas

ASIGNAR (comandosql) VALOR;

"SELECT * FROM «vista_tablas_a_replicar.tabla»;

WHERE (vista_tablas_a_replicar.clave) = '(vista_control.valor)' "

CASO CONTRARIO

→vista_control.valor es un tipo diferente de carácter, por lo tanto no va entre comillas

ASIGNAR (comandosql) VALOR;

"SELECT * FROM «vista_tablas_a_replicar.tabla»;

WHERE (vista_tablas_a_replicar.clave) = (vista_control.valor)"

FIN SI

CREAR VISTA SQL (vista_registro_remoto);

MEDIANTE INSTRUCCIÓN SQL (comandosql);

DESDE IDENTIFICADOR DE CONEXIÓN (con1)

ACCEDER A VISTA SQL (vista_registro_remoto)

CONTAR REGISTROS DE VISTA SQL (vista_registro_remoto) EN (ncont1)

IR AL PRIMER REGISTRO DE VISTA SQL (vista_registro_remoto)

→Crear vista sql vista_registro_local, para verificar la existencia de este en la tabla local,

→ para actualizarlo o eliminarlo

SI vista_tablas_a_replicar.tipodato=”C” **ENTONCES**

→vista_control.valor es tipo carácter, va entre comillas

ASIGNAR (comandosql) VALOR;

“SELECT * FROM «vista_tablas_a_replicar.tabla»;

WHERE (vista_tablas_a_replicar.clave) = ’*valor*(vista_control.valor)’ ”

CASO CONTRARIO

→vista_control.valor no es tipo carácter, no debe ir entre comillas

ASIGNAR (comandosql) VALOR;

“SELECT * FROM «vista_tablas_a_replicar.tabla»;

WHERE (vista_tablas_a_replicar.clave) = *valor*(vista_control.valor)”

FIN SI

CREAR VISTA SQL (vista_registro_local) MEDIANTE INSTRUCCIÓN SQL (comandosql);

DESDE IDENTIFICADOR DE CONEXIÓN (con2)

ACCEDER A VISTA SQL (vista_registro_local)

CONTAR REGISTROS DE VISTA SQL (vista_registro_local) EN (ncont)

ASIGNAR (pase) VALOR FALSO

SELECCIÓN MÚLTIPLE

SELECCIÓN vista_control.operación=”I”

→Transacción insertar

SI (ncont=0) **ENTONCES**

→Insertar registro, ncont en 0, no existe el registro en tabla local

INSERTAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con2);

DESDE REGISTRO ACTUAL DE VISTA SQL (vista_registro_remoto)

ASIGNAR (pase) VALOR VERDADERO

CASO CONTRARIO

→Significa que ncont está en 1, por lo tanto ya está insertado y se actualizará

ACTUALIZAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con2);

DE ACUERDO A CLAVE (*valor*(vista_control.valor));

POR REGISTRO ACTUAL DE VISTA SQL (vista_registro_remoto)

ASIGNAR (pase) VALOR VERDADERO

FIN SIN

SELECCIÓN vista_control.operación=”U”

→Transacción actualizar

SELECCIÓN MÚLTIPLE

SELECCIÓN ncont=0 Y vista_control.valor<>vista_control.valorant

→ncont=0 y cambio de valor en la clave, cambio el campo primario, se actualiza

ACTUALIZAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con2);

DE ACUERDO A CLAVE (*valor*(vista_control.valorant));

POR REGISTRO ACTUAL DE VISTA SQL (vista_registro_remoto)

ASIGNAR (pase) VALOR VERDADERO

SELECCIÓN ncont=1

→ncont=1, no cambio la clave, actualización de otros campos

ACTUALIZAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con2);

DE ACUERDO A CLAVE (*valor*(vista_control.valor));

POR REGISTRO ACTUAL DE VISTA SQL (vista_registro_remoto)

ASIGNAR (pase) VALOR VERDADERO

FIN SELECCIÓN MÚLTIPLE

SELECCIÓN vista_control.operación=”D”

→Transacción eliminar

SI (ncont=0) **ENTONCES**

ELIMINAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con2);

DE ACUERDO A CLAVE (*valor*(vista_control.valor))

```
            ASIGNAR (pase) VALOR VERDADERO
          FIN SI
        FIN SELECCIÓN MÚLTIPLE
        SI (pase) ENTONCES
          ACTUALIZAR CAMPO (semaforo) DE TABLA («tabla de control») DE (con1);
            POR VALOR (VERDADERO);
            PARA (tabla=vista_tablas_a_replicar.tabla)
        FIN SI
        ACCEDER A VISTA SQL (vista_control)
        RECORRER UN REGISTRO EN VISTA SQL (vista_control)
      FIN MIENTRAS
      ACCEDER A VISTA SQL (vista_tablas_a_replicar)
      RECORRER UN REGISTRO EN VISTA SQL (vista_tablas_a_replicar)
    FIN MIENTRAS

    →Eliminando los registros de transacciones que se replicaron con éxito
    ELIMINAR REGISTROS DE TABLA («tabla de control») DE (con1);
      PARA (semaforo=VERDADERO)

    CERRAR CONEXIÓN (con1)
    CERRAR CONEXIÓN (con2)
```

A continuación se mostrará el mismo algoritmo, pero tomando en cuenta las fallas de comunicación con las bases de datos. Es esta sección se corregirá el problema de integridad de la base de datos y problemas con la concurrencia. Al haber un fallo, no se replicarían todas las transacciones realizadas en la base de datos remota, por lo tanto es necesario volver a ejecutar el algoritmo, para intentar replicar las transacciones faltantes, y las transacciones que ya fueron replicadas no tendrían ningún problema, ya que se actualizarían con el mismo valor, sin que esto influya.

Además en el algoritmo anterior se supone que en la base de datos local no existe la tabla de control, por lo que para nuestro algoritmo siguiente, la base de datos local también tendrá la tabla de control, para poder realizar una replicación de retorno (es decir replicar desde la base de datos local hacia la base de datos remota, por lo que se cambiarían los papeles) y poder cumplir en cierta forma la replicación por la política de validación total, que se discutirá luego.

Las transacciones registradas, mientras dure el proceso de replicación, en la tabla de control de la base de datos local deberán ser eliminadas, para evitar que se repliquen al retorno, ya que los disparadores se ejecutarán también en la base de datos local mientras dure el procedimiento, y sería redundante replicar de retorno estas transacciones, ya que en la base de datos remota ya estarán realizadas estas transacciones.

```
→Definiendo variables
DEFINIR con1,con2→identificadores de la conexión
DEFINIR ncont,ncont1→variables para almacenar número de registros de una tabla
DEFINIR pase→variable lógica para verificar las transacciones con éxito
DEFINIR cont1→contador
DEFINIR comandosql→variable para almacenar el comando sql a ejecutar
DEFINIR horatope→variable para almacenar el tiempo de inicio del proceso

→Estableciendo conexión con las bases de datos, con1 será utilizado para la base de datos
→remota y con2 para la base de datos local
ESTABLECER CONEXIÓN CON (base de datos remota);
  ASIGNAR A IDENTIFICADOR (con1)
→Validando la conexión hacia la base de datos remota
```

SI (falla conexión con1) **ENTONCES**

ABORTAR ALGORITMO

FIN SI

ESTABLECER CONEXIÓN CON (base de datos local);

ASIGNAR A IDENTIFICADOR (con2)

→Validando la conexión hacia la base de datos local

SI (falla conexión con2) **ENTONCES**

CERRAR CONEXIÓN (con1)

ABORTAR ALGORITMO

FIN SI

ASIGNAR (horatope) VALOR *fecha_hora_actual*()

→Crear vista temporal para la tabla «tablas a replicar»

CREAR VISTA SQL (vista_tablas_a_replicar) MEDIANTE INSTRUCCIÓN SQL;

"SELECT * FROM «tablas a replicar»";

DESDE IDENTIFICADOR DE CONEXIÓN (con1)

→Verificando si se pudo realizar la consulta

SI (falla consulta) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ACCEDER A VISTA SQL (vista_tablas_a_replicar)

IR AL PRIMER REGISTRO DE VISTA SQL (vista_tablas_a_replicar)

MIENTRAS (no se encuentre fin de vista sql vista_tablas_a_replicar)

→Bucle para examinar toda la vista sql vista_tablas_a_replicar

→Crear vista sql vista_control para examinar transacciones realizadas en la base de datos

→remota

CREAR VISTA SQL (vista_control) MEDIANTE INSTRUCCIÓN SQL;

"SELECT * FROM «control» WHERE tabla=*valor*(vista_tablas_a_replicar.tabla)";

DESDE IDENTIFICADOR DE CONEXIÓN (con1)

→Verificando si se pudo realizar la consulta

SI (falla consulta) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ACCEDER A VISTA SQL (vista_control)

IR AL PRIMER REGISTRO DE VISTA SQL (vista_control)

MIENTRAS (no se encuentre fin de vista sql vista_control)

→Bucle para ir escogiendo transacción por transacción registradas en la tabla de control

→Crear vista sql vista_registro_remoto, para extraer el registro a insertar o modificar

SI vista_tablas_a_replicar.tipodato=”C” **ENTONCES**

→vista_control.valor es tipo carácter, entonces va entre comillas

ASIGNAR (comandosql) VALOR;

“SELECT * FROM «vista_tablas_a_replicar.tabla»;

WHERE (vista_tablas_a_replicar.clave) = ’(vista_control.valor)’ ”

CASO CONTRARIO

→vista_control.valor es un tipo diferente de carácter, por lo tanto no va entre comillas

ASIGNAR (comandosql) VALOR;

“SELECT * FROM «vista_tablas_a_replicar.tabla»;

WHERE (vista_tablas_a_replicar.clave) = (vista_control.valor)”

FIN SI

CREAR VISTA SQL (vista_registro_remoto);

MEDIANTE INSTRUCCIÓN SQL (comandosql);

DESDE IDENTIFICADOR DE CONEXIÓN (con1)

→Verificando si se pudo realizar la consulta

SI (falla consulta) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ACCEDER A VISTA SQL (vista_registro_remoto)

CONTAR REGISTROS DE VISTA SQL (vista_registro_remoto) EN (ncont1)

IR AL PRIMER REGISTRO DE VISTA SQL (vista_registro_remoto)

→Crear vista sql vista_registro_local, para verificar la existencia de este en la tabla local

→para actualizarlo o eliminarlo

SI vista_tablas_a_replicar.tipodato=”C” **ENTONCES**

→vista_control.valor es tipo carácter, va entre comillas

ASIGNAR (comandosql) VALOR;

“SELECT * FROM «vista_tablas_a_replicar.tabla»;

WHERE (vista_tablas_a_replicar.clave) = ’*valor*(vista_control.valor)’ ”

CASO CONTRARIO

→vista_control.valor no es tipo carácter, no debe ir entre comillas

ASIGNAR (comandosql) VALOR;

“SELECT * FROM «vista_tablas_a_replicar.tabla»;

WHERE (vista_tablas_a_replicar.clave) = *valor*(vista_control.valor)”

FIN SI

CREAR VISTA SQL (vista_registro_local) MEDIANTE INSTRUCCIÓN SQL (comandosql);

DESDE IDENTIFICADOR DE CONEXIÓN (con2)

→Verificando si se pudo realizar la consulta

SI (falla consulta) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ACCEDER A VISTA SQL (vista_registro_local)

CONTAR REGISTROS DE VISTA SQL (vista_registro_local) EN (ncont)

ASIGNAR (pase) VALOR FALSO

SELECCIÓN MÚLTIPLE

SELECCIÓN vista_control.operación=”I”

→Transacción insertar

SI (ncont=0) **ENTONCES**

→Insertar registro, ncont en 0, no existe el registro en tabla local

INSERTAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con2);

DESDE REGISTRO ACTUAL DE VISTA SQL (vista_registro_remoto)

→Verificando si se pudo realizar la inserción

SI (falla inserción) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ASIGNAR (pase) VALOR VERDADERO

CASO CONTRARIO

→Significa que ncont está en 1, por lo tanto ya está insertado y se actualizará

ACTUALIZAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con2);

DE ACUERDO A CLAVE (*valor*(vista_control.valor));

POR REGISTRO ACTUAL DE VISTA SQL (vista_registro_remoto)

→Verificando si se pudo realizar la actualización

SI (falla actualización) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ASIGNAR (pase) VALOR VERDADERO

FIN SIN

SELECCIÓN vista_control.operación=”U”

→Transacción actualizar

SELECCIÓN MÚLTIPLE

SELECCIÓN ncont=0 Y vista_control.valor<>vista_control.valorant

→ncont=0 y cambio de valor en la clave, cambio el campo primario, se actualiza

ACTUALIZAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con2);

DE ACUERDO A CLAVE (*valor*(vista_control.valorant));

POR REGISTRO ACTUAL DE VISTA SQL (vista_registro_remoto)

SI (falla actualización) **ENTONCES**

ABORTAR ALGORITMO

FIN SI

ASIGNAR (pase) VALOR VERDADERO

SELECCIÓN ncont=1

→ncont=1, no cambio la clave, actualización de otros campos

ACTUALIZAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con2);

DE ACUERDO A CLAVE (*valor*(vista_control.valor));

POR REGISTRO ACTUAL DE VISTA SQL (vista_registro_remoto)

→Verificando si se pudo realizar la actualización

SI (falla actualización) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ASIGNAR (pase) VALOR VERDADERO

FIN SELECCIÓN MÚLTIPLE

SELECCIÓN vista_control.operación=”D”

→Transacción eliminar

SI (ncont=0) **ENTONCES**

ELIMINAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con2);

DE ACUERDO A CLAVE (*valor*(vista_control.valor))

→Verificando si se pudo realizar la eliminación

SI (falla eliminación) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ASIGNAR (pase) VALOR VERDADERO

FIN SI

FIN SELECCIÓN MÚLTIPLE

SI (pase) **ENTONCES**

ACTUALIZAR CAMPO (semaforo) DE TABLA («tabla de control») DE (con1);

POR VALOR (VERDADERO);

PARA (tabla=vista_tablas_a_replicar.tabla)

→Verificando si falla actualización

SI (falla actualización) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

FIN SI

ACCEDER A VISTA SQL (vista_control)

RECORRER UN REGISTRO EN VISTA SQL (vista_control)

FIN MIENTRAS

ACCEDER A VISTA SQL (vista_tablas_a_replicar)

RECORRER UN REGISTRO EN VISTA SQL (vista_tablas_a_replicar)

FIN MIENTRAS

→Eliminando los registros de transacciones que se replicaron con éxito

ELIMINAR REGISTROS DE TABLA («tabla de control») DE (con1);

PARA (semaforo=VERDADERO)

→Verificando si falla eliminación

SI (falla eliminación) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

→Eliminando registros de transacciones realizadas mientras dura el procedimiento de
→replicación
ELIMINAR REGISTROS DE TABLA («tabla de control») DE (con2);
PARA (fecha>=horatope)

CERRAR CONEXIÓN (con1)
CERRAR CONEXIÓN (con2)

3.2.2.4. Algoritmo para replicar mediante la política de validación total

Para implementar este algoritmo, es necesario que estén instaladas las tablas de tablas a replicar y control, además de los disparadores, tanto en la base de datos local y remota.

Este algoritmo consiste en actualizar las transacciones tanto desde la base de datos remota hacia la local y viceversa, por lo tanto aplicaremos el algoritmo anterior, tanto de ida como de venida.

→Definiendo variables
DEFINIR con1,con2→identificadores de la conexión
DEFINIR ncont,ncont1→variables para almacenar número de registros de una tabla
DEFINIR pase→variable lógica para verificar las transacciones con éxito
DEFINIR cont1→contador
DEFINIR comandosql→variable para almacenar el comando sql a ejecutar
DEFINIR horatope→variable para almacenar el tiempo de inicio del proceso

→Estableciendo conexión con las bases de datos, con1 será utilizado para la base de datos

→ remota y con2 para la base de datos local

ESTABLECER CONEXIÓN CON (base de datos remota);

ASIGNAR A IDENTIFICADOR (con1)

→Validando la conexión hacia la base de datos remota

SI (falla conexión con1) **ENTONCES**

ABORTAR ALGORITMO

FIN SI

ESTABLECER CONEXIÓN CON (base de datos local);

ASIGNAR A IDENTIFICADOR (con2)

→Validando la conexión hacia la base de datos local

SI (falla conexión con2) **ENTONCES**

CERRAR CONEXIÓN (con1)

ABORTAR ALGORITMO

FIN SI

ASIGNAR (horatope) VALOR *fecha_hora_actual*()

→Replicación desde la base de datos remota hacia la local

→Crear vista temporal para la tabla «tablas a replicar»

CREAR VISTA SQL (vista_tablas_a_replicar) MEDIANTE INSTRUCCIÓN SQL;

"SELECT * FROM «tablas a replicar»";

DESDE IDENTIFICADOR DE CONEXIÓN (con1)

→Verificando si se pudo realizar la consulta

SI (falla consulta) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ACCEDER A VISTA SQL (vista_tablas_a_replicar)

IR AL PRIMER REGISTRO DE VISTA SQL (vista_tablas_a_replicar)

MIENTRAS (no se encuentre fin de vista sql vista_tablas_a_replicar)

→Bucle para examinar toda la vista sql vista_tablas_a_replicar

→Crear vista sql vista_control para examinar transacciones realizadas en la base de datos

→remota

CREAR VISTA SQL (vista_control) MEDIANTE INSTRUCCIÓN SQL;

"SELECT * FROM «control» WHERE tabla=*valor*(vista_tablas_a_replicar.tabla)";

DESDE IDENTIFICADOR DE CONEXIÓN (con1)

→Verificando si se pudo realizar la consulta

SI (falla consulta) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ACCEDER A VISTA SQL (vista_control)

IR AL PRIMER REGISTRO DE VISTA SQL (vista_control)

MIENTRAS (no se encuentre fin de vista sql vista_control)

→Bucle para ir escogiendo transacción por transacción registradas en la tabla de control

→Crear vista sql vista_registro_remoto, para extraer el registro a insertar o modificar

SI vista_tablas_a_replicar.tipodato=”C” **ENTONCES**

→vista_control.valor es tipo carácter, entonces va entre comillas

ASIGNAR (comandosql) VALOR;

“SELECT * FROM «vista_tablas_a_replicar.tabla»;

WHERE (vista_tablas_a_replicar.clave) = ’(vista_control.valor)’ ”

CASO CONTRARIO

→vista_control.valor es un tipo diferente de carácter, por lo tanto no va entre comillas

ASIGNAR (comandosql) VALOR;

"SELECT * FROM «vista_tablas_a_replicar.tabla»;

WHERE (vista_tablas_a_replicar.clave) = (vista_control.valor)"

FIN SI

CREAR VISTA SQL (vista_registro_remoto);

MEDIANTE INSTRUCCIÓN SQL (comandosql);

DESDE IDENTIFICADOR DE CONEXIÓN (con1)

→Verificando si se pudo realizar la consulta

SI (falla consulta) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ACCEDER A VISTA SQL (vista_registro_remoto)

CONTAR REGISTROS DE VISTA SQL (vista_registro_remoto) EN (ncont1)

IR AL PRIMER REGISTRO DE VISTA SQL (vista_registro_remoto)

→Crear vista sql vista_registro_local, para verificar la existencia de este en la tabla local,

→para

→actualizarlo o eliminarlo

SI vista_tablas_a_replicar.tipodato="C" **ENTONCES**

→vista_control.valor es tipo carácter, va entre comillas

ASIGNAR (comandosql) VALOR;

"SELECT * FROM «vista_tablas_a_replicar.tabla»;

WHERE (vista_tablas_a_replicar.clave) = '*valor*(vista_control.valor)' "

CASO CONTRARIO

→vista_control.valor no es tipo carácter, no debe ir entre comillas

ASIGNAR (comandosql) VALOR;

"SELECT * FROM «vista_tablas_a_replicar.tabla»;

WHERE (vista_tablas_a_replicar.clave) = *valor*(vista_control.valor)"

FIN SI

CREAR VISTA SQL (vista_registro_local) MEDIANTE INSTRUCCIÓN SQL (comandosql);

DESDE IDENTIFICADOR DE CONEXIÓN (con2)

→Verificando si se pudo realizar la consulta

SI (falla consulta) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ACCEDER A VISTA SQL (vista_registro_local)

CONTAR REGISTROS DE VISTA SQL (vista_registro_local) EN (ncont)

ASIGNAR (pase) VALOR FALSO

SELECCIÓN MÚLTIPLE

SELECCIÓN vista_control.operación=”I”

→Transacción insertar

SI (ncont=0) **ENTONCES**

→Insertar registro, ncont en 0, no existe el registro en tabla local

INSERTAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con2);

DESDE REGISTRO ACTUAL DE VISTA SQL (vista_registro_remoto)

→Verificando si se pudo realizar la inserción

SI (falla inserción) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ASIGNAR (pase) VALOR VERDADERO

CASO CONTRARIO

→Significa que ncont está en 1, por lo tanto ya está insertado y se actualizará

→Verificando si la inserción en la tabla remota se realizó primero

→Creando vista sql vista_registros_locales_control para buscar el registro de

→inserción en la tabla local

CREAR VISTA SQL (vista_registros_locales_control);

MEDIANTE INSTRUCCIÓN SQL;

"SELECT * FROM «control» WHERE tabla=*valor*(vista_tablas_a_replicar.tabla);

AND valor=*valor*(vista_control.valor) AND operación="I"";

DESDE IDENTIFICADOR DE CONEXIÓN (con2)

→Verificando si se realizó la consulta

SI (falla consulta) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ACCEDER A VISTA SQL (vista_registros_locales_control)

IR AL PRIMER REGISTRO DE VISTA SQL (vista_registros_remotos_control)

SI (vista_registros_locales_control.fecha<vista_control.fecha) **ENTONCES**

→Si la inserción se realizó primero en la tabla local, se actualiza, caso contrario no

→se actualiza

ACTUALIZAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con2);

DE ACUERDO A CLAVE (*valor*(vista_control.valor));

POR REGISTRO ACTUAL DE VISTA SQL (vista_registro_remoto)

→Verificando si se pudo realizar la actualización

SI (falla actualización) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

FIN SI

ASIGNAR (pase) VALOR VERDADERO

FIN SIN

SELECCIÓN vista_control.operación="U"

→Transacción actualizar

SELECCIÓN MÚLTIPLE

SELECCIÓN ncont=0 Y vista_control.valor<>vista_control.valorant

→ncont=0 y cambio de valor en la clave, cambio el campo primario, se actualiza

ACTUALIZAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con2);

DE ACUERDO A CLAVE (*valor*(vista_control.valorant));

POR REGISTRO ACTUAL DE VISTA SQL (vista_registro_remoto)

SI (falla actualización) **ENTONCES**

ABORTAR ALGORITMO

FIN SI

ASIGNAR (pase) VALOR VERDADERO

SELECCIÓN ncont=1

→ncont=1, no cambio la clave, actualización de otros campos

ACTUALIZAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con2);

DE ACUERDO A CLAVE (*valor*(vista_control.valor));

POR REGISTRO ACTUAL DE VISTA SQL (vista_registro_remoto)

→Verificando si se pudo realizar la actualización

SI (falla actualización) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ASIGNAR (pase) VALOR VERDADERO

FIN SELECCIÓN MÚLTIPLE

SELECCIÓN vista_control.operación=”D”

→Transacción eliminar

SI (ncont=0) **ENTONCES**

ELIMINAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con2);

DE ACUERDO A CLAVE (*valor*(vista_control.valor))

→Verificando si se pudo realizar la eliminación

SI (falla eliminación) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ASIGNAR (pase) VALOR VERDADERO

FIN SI

FIN SELECCIÓN MÚLTIPLE

SI (pase) **ENTONCES**

ACTUALIZAR CAMPO (semaforo) DE TABLA («tabla de control») DE (con1);

POR VALOR (VERDADERO);

PARA (tabla=vista_tablas_a_replicar.tabla)

→Verificando si falla actualización

SI (falla actualización) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

FIN SI

ACCEDER A VISTA SQL (vista_control)

RECORRER UN REGISTRO EN VISTA SQL (vista_control)

FIN MIENTRAS

ACCEDER A VISTA SQL (vista_tablas_a_replicar)

RECORRER UN REGISTRO EN VISTA SQL (vista_tablas_a_replicar)

FIN MIENTRAS

→Eliminando los registros de transacciones que se replicaron con éxito

ELIMINAR REGISTROS DE TABLA («tabla de control») DE (con1);

PARA (semaforo=VERDADERO)

→Verificando si falla eliminación

SI (falla eliminación) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

→Replicación desde la base de datos local hacia la remota

→Crear vista temporal para la tabla «tablas a replicar»

CREAR VISTA SQL (vista_tablas_a_replicar) MEDIANTE INSTRUCCIÓN SQL;

"SELECT * FROM «tablas a replicar»";

DESDE IDENTIFICADOR DE CONEXIÓN (con2)

→Verificando si se pudo realizar la consulta

SI (falla consulta) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ACCEDER A VISTA SQL (vista_tablas_a_replicar)

IR AL PRIMER REGISTRO DE VISTA SQL (vista_tablas_a_replicar)

MIENTRAS (no se encuentre fin de vista sql vista_tablas_a_replicar)

→Bucle para examinar toda la vista sql vista_tablas_a_replicar

→Crear vista sql vista_control para examinar transacciones realizadas en la base de datos

→local

CREAR VISTA SQL (vista_control) MEDIANTE INSTRUCCIÓN SQL;

"SELECT * FROM «control» WHERE tabla=*valor*(vista_tablas_a_replicar.tabla)";

DESDE IDENTIFICADOR DE CONEXIÓN (con2)

→Verificando si se pudo realizar la consulta

SI (falla consulta) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ACCEDER A VISTA SQL (vista_control)

IR AL PRIMER REGISTRO DE VISTA SQL (vista_control)

MIENTRAS (no se encuentre fin de vista sql vista_control)

→Bucle para ir escogiendo transacción por transacción registradas en la tabla de control

→Crear vista sql vista_registro_local, para extraer el registro a insertar o modificar

SI vista_tablas_a_replicar.tipodato=”C” **ENTONCES**

→vista_control.valor es tipo carácter, entonces va entre comillas

ASIGNAR (comandosql) VALOR;

“SELECT * FROM «vista_tablas_a_replicar.tabla»;

WHERE (vista_tablas_a_replicar.clave) = ’(vista_control.valor)’ ”

CASO CONTRARIO

→vista_control.valor es un tipo diferente de carácter, por lo tanto no va entre comillas

ASIGNAR (comandosql) VALOR;

“SELECT * FROM «vista_tablas_a_replicar.tabla»;

WHERE (vista_tablas_a_replicar.clave) = (vista_control.valor)”

FIN SI

CREAR VISTA SQL (vista_registro_local);

MEDIANTE INSTRUCCIÓN SQL (comandosql);

DESDE IDENTIFICADOR DE CONEXIÓN (con2)

→Verificando si se pudo realizar la consulta

SI (falla consulta) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ACCEDER A VISTA SQL (vista_registro_local)

CONTAR REGISTROS DE VISTA SQL (vista_registro_local) EN (ncont1)

IR AL PRIMER REGISTRO DE VISTA SQL (vista_registro_local)

→Crear vista sql vista_registro_remoto, para verificar la existencia de este en la tabla

→ remota, para actualizarlo o eliminarlo

SI vista_tablas_a_replicar.tipodato=”C” **ENTONCES**

→vista_control.valor es tipo carácter, va entre comillas

ASIGNAR (comandosql) VALOR;

“SELECT * FROM «vista_tablas_a_replicar.tabla»;

WHERE (vista_tablas_a_replicar.clave) = ’*valor*(vista_control.valor)’ ”

CASO CONTRARIO

→vista_control.valor no es tipo carácter, no debe ir entre comillas

ASIGNAR (comandosql) VALOR;

“SELECT * FROM «vista_tablas_a_replicar.tabla»;

WHERE (vista_tablas_a_replicar.clave) = *valor*(vista_control.valor)”

FIN SI

CREAR VISTA SQL (vista_registro_remoto);

MEDIANTE INSTRUCCIÓN SQL (comandosql);

DESDE IDENTIFICADOR DE CONEXIÓN (con1)

→Verificando si se pudo realizar la consulta

SI (falla consulta) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ACCEDER A VISTA SQL (vista_registro_remoto)

CONTAR REGISTROS DE VISTA SQL (vista_registro_remoto) EN (ncont)

ASIGNAR (pase) VALOR FALSO

SELECCIÓN MÚLTIPLE

SELECCIÓN vista_control.operación=”I”

→Transacción insertar

SI (ncont=0) **ENTONCES**

→Insertar registro, ncont en 0, no existe el registro en tabla local

INSERTAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con1);

DESDE REGISTRO ACTUAL DE VISTA SQL (vista_registro_local)

→Verificando si se pudo realizar la inserción

SI (falla inserción) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ASIGNAR (pase) VALOR VERDADERO

CASO CONTRARIO

→Significa que ncont está en 1, por lo tanto ya está insertado y se actualizará

→Verificando si la inserción en la tabla remota se realizó primero

→Creando vista sql vista_registros_locales_control para buscar el registro de

→inserción en la tabla local

CREAR VISTA SQL (vista_registros_locales_control);

MEDIANTE INSTRUCCIÓN SQL;

"SELECT * FROM «control» WHERE tabla=*valor*(vista_tablas_a_replicar.tabla);

AND valor=*valor*(vista_control.valor) AND operación="I"";

DESDE IDENTIFICADOR DE CONEXIÓN (con2)

→Verificando si se realizó la consulta

SI (falla consulta) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ACCEDER A VISTA SQL (vista_registros_locales_control)

IR AL PRIMER REGISTRO DE VISTA SQL (vista_registros_remotos_control)

SI (vista_registros_locales_control.fecha<vista_control.fecha) **ENTONCES**

→Si la inserción se realizó primero en la tabla local, se actualiza, caso contrario no

→se actualiza

ACTUALIZAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con2);

DE ACUERDO A CLAVE (*valor*(vista_control.valor));

POR REGISTRO ACTUAL DE VISTA SQL (vista_registro_local)

→Verificando si se pudo realizar la actualización

SI (falla actualización) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

FIN SI

ASIGNAR (pase) VALOR VERDADERO

FIN SIN

SELECCIÓN vista_control.operación=”U”

→Transacción actualizar

SELECCIÓN MÚLTIPLE

SELECCIÓN ncont=0 Y vista_control.valor<>vista_control.valorant

→ncont=0 y cambio de valor en la clave, cambio el campo primario, se actualiza

ACTUALIZAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con1);

DE ACUERDO A CLAVE (*valor*(vista_control.valorant));

POR REGISTRO ACTUAL DE VISTA SQL (vista_registro_local)

SI (falla actualización) **ENTONCES**

ABORTAR ALGORITMO

FIN SI

ASIGNAR (pase) VALOR VERDADERO

SELECCIÓN ncont=1

→ncont=1, no cambio la clave, actualización de otros campos

ACTUALIZAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con1);

DE ACUERDO A CLAVE (*valor*(vista_control.valor));

POR REGISTRO ACTUAL DE VISTA SQL (vista_registro_local)

→Verificando si se pudo realizar la actualización

SI (falla actualización) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ASIGNAR (pase) VALOR VERDADERO

FIN SELECCIÓN MÚLTIPLE

SELECCIÓN vista_control.operación=”D”

→Transacción eliminar

SI (ncont=0) **ENTONCES**

ELIMINAR REGISTRO EN TABLA («vista_tablas_a_replicar.tabla») DE (con1);

DE ACUERDO A CLAVE (*valor*(vista_control.valor))

→Verificando si se pudo realizar la eliminación

SI (falla eliminación) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

ASIGNAR (pase) VALOR VERDADERO

FIN SI

FIN SELECCIÓN MÚLTIPLE

SI (pase) **ENTONCES**

ACTUALIZAR CAMPO (semaforo) DE TABLA («tabla de control») DE (con2);

POR VALOR (VERDADERO);

PARA (tabla=vista_tablas_a_replicar.tabla)

→Verificando si falla actualización

SI (falla actualización) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

FIN SI

ACCEDER A VISTA SQL (vista_control)

RECORRER UN REGISTRO EN VISTA SQL (vista_control)

FIN MIENTRAS

ACCEDER A VISTA SQL (vista_tablas_a_replicar)

RECORRER UN REGISTRO EN VISTA SQL (vista_tablas_a_replicar)

FIN MIENTRAS

→Eliminando los registros de transacciones que se replicaron con éxito

ELIMINAR REGISTROS DE TABLA («tabla de control») DE (con2);

PARA (semaforo=VERDADERO)

→Verificando si falla eliminación

SI (falla eliminación) **ENTONCES**

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

ABORTAR ALGORITMO

FIN SI

→Eliminando registros de transacciones realizadas mientras dura el procedimiento de

→replicación

ELIMINAR REGISTROS DE TABLA («tabla de control») DE (con1);

PARA (fecha>=horatope)

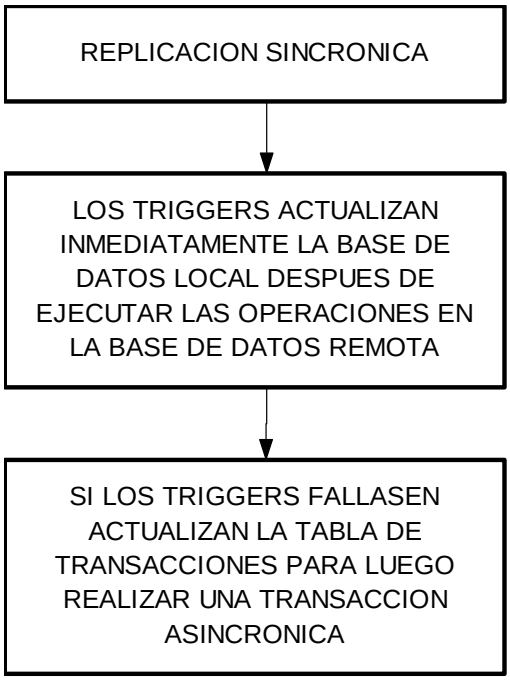
→Cerrando la conexión con las bases de datos

CERRAR CONEXIÓN (con1)

CERRAR CONEXIÓN (con2)

3.2.2.5. Replicación Sincrónica

Para la replicación sincrónica, las dos bases de datos, tanto remota como local, deben estar en permanente comunicación.



Para este cometido utilizamos los algoritmos de las operaciones INSERT, UPDATE y DELETE, de manera que actualicen la base de datos local, en el momento que se realicen las operaciones en la base de datos remota.

Para completar el algoritmo, las transacciones que no se puedan replicar se enviaran a la tabla de transacciones de la base de datos remota, para luego replicarlas asincrónicamente con los algoritmos anteriormente expuestos.

CAPÍTULO IV

APLICACIÓN PRÁCTICA

4.1. BREVE ESTUDIO DE HERRAMIENTAS FRONT-END Y BACK-END

4.1.1. Herramientas para desarrollo de aplicaciones Cliente Servidor

Los algoritmos descritos en el capítulo tres, son prácticos, por lo tanto aplicables. La mejor forma de implementarlos sería, no en un lenguaje de programación dado, sino más bien en una herramienta para desarrollo de aplicaciones Cliente-Servidor.

Por criterio de los autores, enseñanza en centros de formación medio y superior de informática, se escogieron las siguientes empresas de desarrollo informático, para el estudio de sus herramientas de desarrollo: Sybase, Borland Inprise, Symantec y Microsoft.

Sybase, a inicios de los noventa adquirió una empresa llamada Powersoft, que era quien desarrollaba de Power Builder, una de las principales herramientas de la época para desarrollo de aplicaciones Cliente-Servidor. A principio era incierto lo que pasaría con Power Builder ya que paso a manos de Sybase, pero continuo con la tradición de ser la mejor herramienta para el desarrollo de este tipo de aplicaciones ya que actualmente se distribuye con Sybase SQL Anyhware, que es un pequeño motor

de Base de datos (DBMS). Power Builder posee famosos Asistentes (Wizards) para la generación de formularios y Reportes, tiene su propio lenguaje de programación funciones que hacen llamadas a procedimientos únicos en los DBMS de Sybase, como pro ejemplo llamadas a replicación de datos en SQL Remote, y muchas otras características impresionantes. Además, Sybase pensando en los programadores de otros lenguajes de desarrollo Power C++ y PowerJ, con las mismas características de Power Builder pero con lenguaje de programación C++ y java respectivamente. El problema con estas herramientas es el importe, sobre los cuatrocientos USD ($400) para versiones estándar, mil quinientos USD ($1500) para las versiones profesionales y sobre los tres mil USD ($3000) las versiones corporativas, también hay que tomar en cuenta el costo de bibliografía y capacitación, aparte de ser muy escasa y muy costoso.

Borland Inprise, es por excelencia la empresa que nos ha dado las mejores herramientas de desarrollo, especialmente para las áreas de aprendizaje y capacitación de lenguajes de programación como Turbo Pascal y Borland C. Esta empresa tiene tres poderosas herramientas para el desarrollo de aplicaciones Cliente-Servidor que son las siguientes: CBuilder, Delphi y JBuilder. Tienen generadores de aplicaciones, formularios, reportes, acceso a Base de Datos nativos y mediante ODBCs, CORBA, JDBC. CBuilder programa en C++, Delphi en Pascal y JBuilder en Java, pero como en el caso anterior el problema es el costo: ochocientos USD ($800) para versiones estándar (sin acceso a Base de Datos es decir para desarrollo de aplicaciones de propósito general), mil quinientos USD ($1500) para versiones profesionales y sobre los tres mil USD ($3000) para versiones empresariales. La bibliografía y capacitación

de estas herramientas son escasas, a excepción de Delphi, que es muy difundida la enseñaza de esta herramienta en los centros de educación superior.

Symantec tiene una herramienta muy poderosa que es tan buena o superior a las anteriormente mencionadas denominada Visual Café (por su programación en java), pero lamentablemente no ha sido muy difundida en nuestro medio, pero en otros países están popular como Visual J++ de Microsoft o JBuilder de Borland Inprise.

Microsoft siempre nos ha sorprendido con sus paquetes integrados para oficinas como el mismo Office y Back Office, y nos presenta su Office de programación denominado Microsoft Visual Studio, que consta de las siguientes herramientas: Visual Basic, Visual FoxPro, Visual C++, Visual J++ y Visual Interdev. Las más populares de esta caja de herramientas son Visual FoxPro y Visual Basic. Visual Basic fue una de las primeras herramientas de desarrollo visual, de ahí su nombre, posee generadores de reportes de terceros (es decir de otra empresa, no de Microsoft), se programa en Basic, pero tiene un problema con incompatibilidad de programación con sus versiones anteriores, Visual FoxPro programa en xbase, y a parte de ser una herramienta visual es netamente orientada a objetos tiene generadores para bastantes operaciones rutinarias, como de formularios, reportes, menús, incluso generadores de documentación. Tiene un poco de incompatibilidad de programación con sus versiones anteriores, y algunas funciones cuando son mal aplicadas funcionan de una manera desastrosa sin dar un error de programación, teniendo que depurar, por suerte tiene un excelente programa de depuración. El costo de estas herramientas es relativamente bajo: doscientos cincuenta USD ($250) por versiones didácticas de cada

una de estas herramientas, setecientos USD ($700) en versiones profesionales y mil doscientos USD ($1200) en versiones empresariales. Tiene otra ventaja, que se puede comprar todo el paquete de Visual Studio por el precio de dos herramientas y tenemos las licencias de todas de una vez. Existe bastante bibliografía y son muy difundidos en los centros de Educación Superior.

Las herramientas anteriormente descritas operan bajo sistemas operativos Microsoft Windows, Power Builder tiene versiones de sus programas para OS2, Jbuilder también opera bajo Solaris, Linux y Macintosh, cabe destacar que recientemente Borland Inprise saco al mercado Killix, que es una poderosa herramienta que opera bajo Linux.

Cabe mencionar que existen otras herramientas en el mercado que no se detallan en este documento, pero son importantes, no se discutieron ya que no son tan difundidas en nuestro medio, estos paquetes son difíciles de conseguir y la bibliografía es escasa, como de empresas como IBM y Computer Associated.

Para demostrar la aplicabilidad de los algoritmos del capítulo anterior se debe escoger una de las herramientas anteriormente descritas, lo que no significa que se puedan desarrollar en otros lenguajes de programación u otras herramientas, y la opción más viable es Microsoft Visual FoxPro, porque aparte de los pro y contras, tiene otras ventajas, como, que su lenguaje es de sencilla comprensión y es didáctico, inclusive hay una revista norteamericana (Programmer FoxPro), dedicada única y exclusivamente a la programación en FoxPro (cuesta doce USD ($12) mensuales). La

bibliografía es extensa y no muy costosa. Incluso se pueden migrar pequeñas aplicaciones de negocios desarrolladas en CAClipper de Computer Associated, de FoxBase de FoxBase Corp. Y Dbase de Dbase Corp. Es una herramienta bastante difundida en centros de educación superior, tal es que para carreras de tecnologías informáticas, se debe aprobar asignaturas de FoxPro, por lo tanto es común entre programadores locales.

Ver anexo D (Cuadro comparativo de las Herramientas de Desarrollo más Populares).

4.1.2. Motores de Base de Datos Relacionales (DBMS, BackEnds)

Un motor de base datos es un conjunto de herramientas de software que manipulan grandes cantidades de datos almacenados en una base de datos física (archivos), está compuesta de tablas, tablas índices y procedimientos almacenados.

En el mercado se puede encontrar poderosos motores de base de datos como Sybase Adaptive Server de Sybase, Informix Dynamic Server de Informix, Microsoft SQL Server de Microsoft, etc., pero tienen un inconveniente, su precio y capacitación. Para pequeñas empresas, se acostumbra a usar tablas sueltas o archivos de datos en formato texto o binario, pues el volumen de información y la seguridad que implica no justifica la compra de un DBMS de altas prestaciones.

Para este estudio, se tomó como referencia tres motores, que si bien tienen un costo relativamente bajo, tienen altas prestaciones, y han demostrado ser unos excelentes

manejadores de base de datos.

Sybase SQL Anyware, utilizado por las tecnologías Sybase para bases de datos móviles, es la plataforma BackEnd de desarrollo para Power Builder (con acceso nativo), tiene todas las características para no envidiar a ningún gran DBMS, como por ejemplo la creación de tipos de datos definidos por el usuario, se puede instalar en Windows 9x, Windows NT y Linux. Para el acceso, se realiza mediante ODBC con protocolos TCP/IP, IPX, SMB y otros. El precio es de ochocientos USD ($800) para licencia de diez usuarios.

Microsoft Visual FoxPro, que es el DBMS que viene integrado con Visual FoxPro, tiene grandes mejoras con respecto a sus predecesoras, que eran las tablas con formato Dbase, por ejemplo que es una base de datos compacta, anteriormente si había una falla de energía, incluso se tenían que correr utilidades para reparar las tablas afectadas en el suceso, actualmente ya no existe ese problema. Funciona como un motor de base de datos relacional, por lo tanto la posibilidad de almacenar procedimientos para disparadores. Para ser utilizado por otras herramientas de desarrollo que no sea Visual FoxPro (que tiene acceso nativo), se utiliza ODBC, la base de datos (que consta de archivos DBF que contiene las tablas, CDX tablas de índice, FPT campos memo, y tres archivos más para la información de la base de datos), puede ser copiada en un directorio de cualquier plataforma como Windows o UNIX, que manipulen el protocolo de red de Bloque de Mensajes de Servidor (SMB), no tiene un mecanismo de seguridad para acceder a sus tablas, por lo tanto esto se maneja con la seguridad de acceso desde el sistema operativo al directorio en el que se

haya copiado los archivos. Para el uso de esta base de datos es necesario tener la licencia de Visual FoxPro profesional como mínimo.

PostgreSQL, es un motor experimental (aunque en la actualidad ya no), desarrollado por la Universidad de Berkeley en EEUU, y actualmente existe una organización sin fines de lucro que lleva su nombre, compuesta por programadores de todo el mundo, para su mantenimiento y mejoramiento, tiene una licencia GNU con código abierto, pero esta organización cobra un importe para cuando este DBMS sea utilizado comercialmente. Es un excelente motor de base de datos, ya que tiene todas las características de un DBMS comercial, incluso para procedimiento almacenados se pueden utilizar diferentes lenguajes de programación, como C o perl. Su código se compila en plataformas UNIX, especialmente en Linux, incluso varias revistas del medio informático como Linux Actual o Linux Journal, han nombrado a PostgreSQL como la mejor base de datos para Linux, también hay la posibilidad de compilarla en Windows NT. Para su acceso, en Linux se puede hacer nativamente desde programas hechos en C mediante la librería libpq y ecgp, una interfaz de libpq con perl, pgtcl para TCL, también se realiza el acceso mediante ODBC con protocolo TCP/IP.

Para la aplicabilidad de los algoritmos descritos en al capítulo anterior, podemos utilizar cualquiera de estos DBMS, a continuación se demuestra la implementación en Visual FoxPro y PostgreSQL.

IMPLEMENTACION PARA VISUAL FOXPRO

Código del trigger insert.

```
PROCEDURE inserta
  LOCAL micodigo,micodigostr,mialias,micursor,mitipodato
  mialias=ALIAS()
  micursor=mialias+"01"
  IF SELECT(micursor)=0
    SELECT * FROM rp_tablas WHERE tabla=mialias INTO CURSOR (micursor)
    SELECT (micursor)
    GO TOP
  ELSE
    SELECT (micursor)
  ENDIF
  micodigostr=ALLTRIM(clave)
  SELECT (mialias)
  mitipodato=TYPE(micodigostr)
  DO CASE
  CASE mitipodato="C"
    micodigo=EVALUATE(micodigostr)
  CASE mitipodato="N"
    micodigo=STR(EVALUATE(micodigostr))
  CASE mitipodato="D" OR mitipodato="T"
    micodigo=DTOC(EVALUATE(micodigostr))
  CASE mitipodato="L"
    micodigo=IIF(EVALUATE(micodigostr),".T.",".F.")
  OTHERWISE
    micodigo=EVALUATE(micodigostr)
  ENDCASE
  INSERT INTO rp_control VALUES(mialias,micodigo,"","I",DATETIME(),SPACE(20),.F.)
RETURN .T.
```

Código del trigger update.

```
PROCEDURE actualiza
  LOCAL micodigo,micodigostr,micodigoant,mialias,micursor,mitipodato
  mialias=ALIAS()
  micursor=mialias+"01"
  IF SELECT(micursor)=0
    SELECT * FROM rp_tablas WHERE tabla=mialias INTO CURSOR (micursor)
    SELECT (micursor)
    GO TOP
  ELSE
    SELECT (micursor)
  ENDIF
  micodigostr=ALLTRIM(clave)
  SELECT (mialias)
  mitipodato=TYPE(micodigostr)
  micodigoant=""
  DO CASE
  CASE mitipodato="C"
    micodigoant=OLDVAL(micodigostr)
    micodigo=EVALUATE(micodigostr)
  CASE mitipodato="N"
    micodigoant=STR(OLDVAL(micodigostr))
    micodigo=STR(EVALUATE(micodigostr))
  CASE mitipodato="D" OR mitipodato="T"
    micodigoant=DTOC(OLDVAL(micodigostr))
    micodigo=DTOC(EVALUATE(micodigostr))
  CASE mitipodato="L"
    micodigoant=IIF(OLDVAL(micodigostr),".T.",".F.")
    micodigo=IIF(EVALUATE(micodigostr),".T.",".F.")
  OTHERWISE
    micodigoant=OLDVAL(micodigostr)
    micodigo=EVALUATE(micodigostr)
  ENDCASE
  INSERT INTO rp_control
 VALUES(mialias,micodigo,micodigoant,"U",DATETIME(),SPACE(20),.F.)
RETURN .T.
```

Código del trigger delete.

```
PROCEDURE elimina
  LOCAL micodigo,micodigostr,mialias,micursor,mitipodato
  mialias=ALIAS()
  micursor=mialias+"01"
  IF SELECT(micursor)=0
    SELECT * FROM rp_tablas WHERE tabla=mialias INTO CURSOR (micursor)
    SELECT (micursor)
    GO TOP
  ELSE
    SELECT (micursor)
  ENDIF
  micodigostr=ALLTRIM(clave)
  SELECT (mialias)
  mitipodato=TYPE(micodigostr)
  DO CASE
  CASE mitipodato="C"
    micodigo=EVALUATE(micodigostr)
  CASE mitipodato="N"
    micodigo=STR(EVALUATE(micodigostr))
  CASE mitipodato="D" OR mitipodato="T"
    micodigo=DTOC(EVALUATE(micodigostr))
  CASE mitipodato="L"
    micodigo=IIF(EVALUATE(micodigostr),".T.",".F.")
  OTHERWISE
    micodigo=EVALUATE(micodigostr)
  ENDCASE
  INSERT INTO rp_control VALUES(mialias,micodigo,"","D",DATETIME(),SPACE(20),.F.)
RETURN .T.
```

IMPLEMENTACION PARA POSTGRESQL.

Código de la librería replica.so para implementar los triggers insertar, actualizar y eliminar en PostgreSQL.

```
/* replica.c v1.0ca */


/* Cabeceras necesarias para el desarrollo de trigger's sobre PostgreSQL. */

#include "executor/spi.h"
#include "commands/trigger.h"

/* Otras cabeceras.                                    */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Definición de las funciones a utilizar.                    */

void ing_pila(char *elem_pila);
char *sacar_pila(void);
void ing_pilas(char *elem_pila);
char *sacar_pilas(void);
void suma(void);
void executep(char p);
void finexp(void);
void nada(void);
char *replicap_evaluate(char *str_evaluate,char tipo_exp,HeapTuple ht,TupleDesc td);
```

```
HeapTuple insertar(void);
HeapTuple actualizar(void);
HeapTuple eliminar(void);

/* Definición de variables globales.                    */

char **pila,**pilas;
char *funcion;
char mi_tipo_exp;
int spila,spilas,texp;

/* Funciones para la pila de sintáctica.                */

void ing_pila(char *elem_pila)
{
   pila[++spila]=(char*)malloc(256);
   sprintf(pila[spila],"%s",elem_pila);
}

char *sacar_pila(void)
{
   char *ret_pila;

   ret_pila=(char*)malloc(256);
   sprintf(ret_pila,"%s",pila[spila]);
   free((char*)pila[spila--]);
   return(ret_pila);
}

/* Funciones para la pila semántica.                    */
```

```
void ing_pilas(char *elem_pila)
{
   pilas[++spilas]=(char*)malloc(256);
   sprintf(pilas[spilas],"%s",elem_pila);
}

char *sacar_pilas(void)
{
   char *ret_pila;

   ret_pila=(char*)malloc(256);
   sprintf(ret_pila,"%s",pilas[spilas]);
   free((char*)pilas[spilas--]);
   return(ret_pila);
}

/* Funciones sintácticas.                                    */

void suma(void)
{
   char *buftmp,*buftmp1,*res;
   int i,t;
   float f1,f2;

   f1=0;
   f2=0;
   buftmp=(char*)malloc(256);
   buftmp1=(char*)malloc(256);
   res=(char*)malloc(256);
   sprintf(res,"%s",sacar_pila());
   if(mi_tipo_exp=='N')
```

```
    f1=atof(res);
  if(spila>=1)
  {
    t=spila;
    for(i=0;i<t;i++)
    {
      sacar_pila();
      ++i;
          sprintf(buftmp1,"%s",sacar_pila());
          if(mi_tipo_exp=='N')
            f2=atof(buftmp1);
          sprintf(buftmp,"%s",res);
          if(mi_tipo_exp=='N')
          {
            f1=f1+f2;
            sprintf(res,"%f",f1);
          }
          else
            sprintf(res,"%s%s",buftmp1,buftmp);
    }
  }
  ing_pilas(res);
  free((char*)buftmp);
  free((char*)buftmp1);
  free((char*)res);
}

void executep(char p)
{
  int i,t;
  char *buftmp,*buftmp1,*res;
```

```
float f1,f2;



funcion[++texp]=p;
buftmp=(char*)malloc(256);
buftmp1=(char*)malloc(256);
res=(char*)malloc(256);
strcpy(res,"");
f1=0;
f2=0;
t=spila;
for(i=0;i<t;i++)
{
  sacar_pila();
  ++i;
  sprintf(buftmp1,"%s",sacar_pila());
  if(mi_tipo_exp=='N')
        f2=atof(buftmp1);
  sprintf(buftmp,"%s",res);
  if(mi_tipo_exp=='N')
  {
        f1=f1+f2;
        sprintf(res,"%f",f1);
  }
  else
        sprintf(res,"%s%s",buftmp1,buftmp);
}
if(t>=0)
  ing_pilas(res);
free((char*)buftmp);
free((char*)buftmp1);
```

```
   free((char*)res);
}

void finexp(void)
{
   char *buftmp;
   float f1;

   buftmp=(char*)malloc(256);
   switch(funcion[texp])
   {
      case 'V':
         suma();
         sprintf(buftmp,"%s",sacar_pilas());
         f1=atof(buftmp);
         sprintf(buftmp,"%f",f1);
         ing_pilas(buftmp);
         break;
      case 'S':
         suma();
         sprintf(buftmp,"%s",sacar_pilas());
         ing_pilas(buftmp);
               break;
      case 'C':
         suma();
         sprintf(buftmp,"%s",sacar_pilas());
         ing_pilas(buftmp);
         break;
   }
   --texp;
   free((char*)buftmp);
```

```
}

void nada(void)
{
  /* No ejecuta nada. */
}

/* Código para la función replicap_evaluate.                    */

char *replicap_evaluate(char *str_evaluate,char tipo_exp,HeapTuple ht,TupleDesc td)
{
  char *buftmp,*str_eval,*ret_str_eval;
  char **tokens;
  char tok[256];
  int i,j,k,t,r;
  float f1,f2;

  mi_tipo_exp=tipo_exp;
  str_eval=(char*)malloc(256);
  ret_str_eval=(char*)malloc(256);
  buftmp=(char*)malloc(256);
  spila=-1;
  pila=(char**)malloc(200);
  spilas=-1;
  pilas=(char**)malloc(200);
  texp=-1;
  tokens=(char**)malloc(200);
  funcion=(char*)malloc(200);

  /* Análisis léxico. */
  j=-1;
```

```
k=-1;
for(i=0;i<strlen(str_evaluate);i++)
{
  tok[++j]=str_evaluate[i];
  switch(tok[j])
  {
    case '+':
      if(str_evaluate[i-1]!=')')
      {
        tokens[++k]=(char*)malloc(256);
        tok[j]='\0';
        sprintf(tokens[k],"%s",tok);
      }
      tokens[++k]=(char*)malloc(256);
      sprintf(tokens[k],"+");
          j=-1;
      break;
    case '(':
      if(i-1>=0)
        if(str_evaluate[i-1]!='+' && str_evaluate[i-1]!=' ')
        {
          tok[j]='\0';
          tokens[++k]=(char*)malloc(256);
          sprintf(tokens[k],"%s",tok);
            }
      tokens[++k]=(char*)malloc(256);
      sprintf(tokens[k],"(");
      j=-1;
      break;
    case ')':
      tok[j]='\0';
```

```
        j=-1;
        tokens[++k]=(char*)malloc(256);
        sprintf(tokens[k],"%s",tok);
        tokens[++k]=(char*)malloc(256);
             sprintf(tokens[k],")");
        break;
      case ' ':
        if(i>0)
        {
          if(str_evaluate[i-1]!=' ')
          {
            if(j<0)
            {
                        tok[j]='\0';
              tokens[++k]=(char*)malloc(256);
              sprintf(tokens[k],"%s",tok);
              --j;
            }
            else
              --j;
          }
          else
            --j;
        }
             else
          --j;
        break;
    }
  }

  if(str_evaluate[i-1]!=')')
```

```
{
   tok[++j]='\0';
   tokens[++k]=(char*)malloc(256);
   sprintf(tokens[k],"%s",tok);
}

/* Análisis sintáctico. */

j=-1;
for(i=0;i<=k;i++)
{
   if(strcmp(tokens[i],"(")==0)
      nada();
   else if(strcmp(tokens[i],")")==0)
   {
      if(i<k && texp<0)
         suma();
      if(texp>=0)
         finexp();
   }
   else if(strcmp(tokens[i],"+")==0)
         ing_pila(tokens[i]);
   else if(strcmp(tokens[i],"val")==0)
      executep('V');
   else if(strcmp(tokens[i],"str")==0)
      executep('S');
   else if(strcmp(tokens[i],"ctod")==0)
      executep('C');
   else
   {
      r=SPI_fnumber(td,tokens[i]);
```

```
        ing_pila(SPI_getvalue(ht,td,r));
    }
  }
  if(spila>=0)
    suma();

  /* Análisis semántico. */
  t=spilas;
  strcpy(str_eval,"");
  f1=0;
  for(i=0;i<t+1;i++)
    if(mi_tipo_exp=='N')
    {
            sprintf(buftmp,"%s",sacar_pilas());
            f2=atof(buftmp);
       f1=f1+f2;
            sprintf(str_eval,"%f",f1);
    }
    else
    {
            sprintf(buftmp,"%s",str_eval);
            sprintf(str_eval,"%s%s",sacar_pilas(),buftmp);
    }
  for(i=0;i<k+1;i++)
    free((char*)tokens[i]);
  free((char**)tokens);
  free((char**)pila);
  free((char**)pilas);
  free((char*)buftmp);
  if(mi_tipo_exp=='C')
    sprintf(ret_str_eval,"'%s'",str_eval);
```

```
  else
    sprintf(ret_str_eval,"%s",str_eval);
  free((char*)str_eval);
  elog(NOTICE,ret_str_eval);
  return(ret_str_eval);
}
```

/* Código para el trigger insertar. */

```
HeapTuple insertar(void)
{
  HeapTuple retTuple;
  TupleDesc tupleDesc;
  int retSPI;
  char *Query,*mi_nombre_clave,*mi_tabla,*mi_tipo_dato,*mi_clave;

  retTuple=CurrentTriggerData->tg_trigtuple;
  tupleDesc=CurrentTriggerData->tg_relation->rd_att;
  Query=(char*)malloc(256);
  mi_tabla=(char*)malloc(256);
  mi_nombre_clave=(char*)malloc(256);
  mi_tipo_dato=(char*)malloc(256);
  mi_clave=(char*)malloc(256);
  if(SPI_connect()==SPI_OK_CONNECT)
  {
    sprintf(mi_tabla,SPI_getrelname(CurrentTriggerData->tg_relation));
    sprintf(Query,"SELECT * FROM rp_tablas WHERE
      tabla='%s'",mi_tabla);
    retSPI=SPI_exec(Query,1);
    if(retSPI==SPI_OK_SELECT && SPI_processed==1)
    {
```

```
        sprintf(mi_nombre_clave,"%s",SPI_getvalue(SPI_tuptable->vals[0],
          SPI_tuptable->tupdesc,2));
        sprintf(mi_tipo_dato,"%s",SPI_getvalue(SPI_tuptable->vals[0],
          SPI_tuptable->tupdesc,3));
        sprintf(mi_clave,"%s",replicap_evaluate(mi_nombre_clave,
          mi_tipo_dato[0],retTuple,tupleDesc));
        elog(NOTICE,mi_clave);
        sprintf(Query,"INSERT INTO rp_control
          VALUES('%s',%s,'','I',CURRENT_TIMESTAMP,'',false)",
          mi_tabla,mi_clave);
        retSPI=SPI_exec(Query,1);
        if(!(retSPI==SPI_OK_INSERT && SPI_processed==1))
          elog(NOTICE,"No se pudo insertar en rp_control.");
      }
      else
        elog(NOTICE,"No se pudo encontrar la relación en rp_tablas.");
      retSPI=SPI_finish();
    }
    else
      elog(NOTICE,"No se pudo establecer conexión con la base de datos.");
    free((char*)Query);
    free((char*)mi_tabla);
    free((char*)mi_nombre_clave);
    free((char*)mi_tipo_dato);
    free((char*)mi_clave);
    return(retTuple);
}

/* Código para el trigger actualizar.                    */

HeapTuple actualizar(void)
```

```
{
  HeapTuple retTuple;
  TupleDesc tupleDesc;
  int retSPI;
  char *Query,*mi_nombre_clave,*mi_tabla,*mi_tipo_dato,*mi_clave;
  char *mi_clave_ant;

  retTuple=CurrentTriggerData->tg_newtuple;
  tupleDesc=CurrentTriggerData->tg_relation->rd_att;
  Query=(char*)malloc(256);
  mi_tabla=(char*)malloc(256);
  mi_nombre_clave=(char*)malloc(256);
  mi_tipo_dato=(char*)malloc(256);
  mi_clave=(char*)malloc(256);
  mi_clave_ant=(char*)malloc(256);
  if(SPI_connect()==SPI_OK_CONNECT)
  {
    sprintf(mi_tabla,SPI_getrelname(CurrentTriggerData->tg_relation));
    sprintf(Query,"SELECT * FROM rp_tablas WHERE
      tabla='%s'",mi_tabla);
    retSPI=SPI_exec(Query,1);
    if(retSPI==SPI_OK_SELECT && SPI_processed==1)
    {
      sprintf(mi_nombre_clave,"%s",SPI_getvalue(SPI_tuptable->vals[0],
        SPI_tuptable->tupdesc,2));
      sprintf(mi_tipo_dato,"%s",SPI_getvalue(SPI_tuptable->vals[0],
        SPI_tuptable->tupdesc,3));
      sprintf(mi_clave,"%s",replicap_evaluate(mi_nombre_clave,
        mi_tipo_dato[0],retTuple,tupleDesc));
      sprintf(mi_clave_ant,"%s",replicap_evaluate(mi_nombre_clave,
        mi_tipo_dato[0],CurrentTriggerData->tg_trigtuple,tupleDesc));
```

```
        sprintf(Query,"INSERT INTO rp_control
          VALUES('%s',%s,%s,'U',CURRENT_TIMESTAMP,'',false)",
          mi_tabla,mi_clave,mi_clave_ant);
        retSPI=SPI_exec(Query,1);
        if(!(retSPI==SPI_OK_INSERT && SPI_processed==1))
          elog(NOTICE,"No se pudo insertar en rp_control.");
      }
      else
        elog(NOTICE,"No se pudo encontrar la relación en rp_tablas.");
      retSPI=SPI_finish();
    }
    else
      elog(NOTICE,"No se pudo establecer conexión con la base de datos.");
    free((char*)Query);
    free((char*)mi_tabla);
    free((char*)mi_nombre_clave);
    free((char*)mi_tipo_dato);
    free((char*)mi_clave);
    free((char*)mi_clave_ant);
    return(retTuple);
}
```

/* Código para el trigger eliminar. */

```
HeapTuple eliminar(void)
{
    HeapTuple retTuple;
    TupleDesc tupleDesc;
    int retSPI;
    char *Query,*mi_nombre_clave,*mi_tabla,*mi_tipo_dato,*mi_clave;
```

```
retTuple=CurrentTriggerData->tg_trigtuple;
tupleDesc=CurrentTriggerData->tg_relation->rd_att;
Query=(char*)malloc(256);
mi_tabla=(char*)malloc(256);
mi_nombre_clave=(char*)malloc(256);
mi_tipo_dato=(char*)malloc(256);
mi_clave=(char*)malloc(256);
if(SPI_connect()==SPI_OK_CONNECT)
{
  sprintf(mi_tabla,SPI_getrelname(CurrentTriggerData->tg_relation));
  sprintf(Query,"SELECT * FROM rp_tablas WHERE
    tabla='%s'",mi_tabla);
  retSPI=SPI_exec(Query,1);
  if(retSPI==SPI_OK_SELECT && SPI_processed==1)
  {
    sprintf(mi_nombre_clave,"%s",SPI_getvalue(SPI_tuptable->vals[0],
      SPI_tuptable->tupdesc,2));
    sprintf(mi_tipo_dato,"%s",SPI_getvalue(SPI_tuptable->vals[0],
      SPI_tuptable->tupdesc,3));
    sprintf(mi_clave,"%s",replicap_evaluate(mi_nombre_clave,
      mi_tipo_dato[0],retTuple,tupleDesc));
    sprintf(Query,"INSERT INTO rp_control
      VALUES('%s',%s,'','D',CURRENT_TIMESTAMP,'',false)",
      mi_tabla,mi_clave);
    retSPI=SPI_exec(Query,1);
    if(!(retSPI==SPI_OK_INSERT && SPI_processed==1))
      elog(NOTICE,"No se pudo insertar en rp_control.");
  }
  else
    elog(NOTICE,"No se pudo encontrar la relación en rp_tablas.");
  retSPI=SPI_finish();
```

```
  }
  else
    elog(NOTICE,"No se pudo establecer conexión con la base de datos.");
  free((char*)Query);
  free((char*)mi_tabla);
  free((char*)mi_nombre_clave);
  free((char*)mi_tipo_dato);
  free((char*)mi_clave);
  return(retTuple);
}
```

4.2. IMPLEMENTACIÓN DE UN SISTEMA DE REPLICACIÓN EN MICROSOFT VISUAL FOXPRO

4.2.1. Implementación del algoritmo de replicación por preferencia de la ejecutora de la replicación en Visual FoxPro

Como se mencionó anteriormente, se escogió como herramienta para desarrollo Microsoft Visual FoxPro, y a continuación se lista el código que se implemento en base del algoritmo de replicación por preferencia de la ejecutora de la replicación.

```
  LOCAL ncont,pase,fintabla,cont1,comandosql,exp1,exp2,ncont1,ncont2

  fintabla=DATETIME()

  namecon1="conexion1"
  namecon2="conexion2"
  con1=SQLCONNECT(namecon1)
```

```
IF con1<=0
  =MESSAGEBOX("Fallo Conexión",64,"Replicación")
  RETURN
ENDIF
con2=SQLCONNECT(namecon2)
IF con2<=0
  =MESSAGEBOX("Fallo Conexión",64,"Replicación")
  =SQLDISCONNECT(con1)
  RETURN
ENDIF

=SQLEXEC(con1,"SELECT * FROM rp_tablas ORDER BY primario","tmp3")
* Tablas que participan en la replicación
SELECT tmp3
GO TOP
DO WHILE NOT EOF()
  * Transacciones que se realizaron en la base de datos primaria
  =SQLEXEC(con1,"SELECT * FROM rp_control WHERE tabla=?tmp3.tabla","tmp4")
  SELECT tmp4
  GO TOP
  DO WHILE NOT EOF()
    * Escogiendo transacción por transacción
    comandosql="SELECT * FROM "+ALLTRIM(tmp3.tabla)+;
    " WHERE "+ALLTRIM(tmp3.clave)+"="+IIF(tmp3.tipodato="C","'","");
    +IIF(tmp3.tipodato="C",LEFT(tmp4.valor,tmp3.longitud),;
    ALLTRIM(tmp4.valor))+IIF(tmp3.tipodato="C","'","")
    =SQLEXEC(con1,comandosql,"tmp2")
    SELECT tmp2
    COUNT TO ncont1
    GO TOP
    comandosql="SELECT * FROM "+ALLTRIM(tmp3.tabla)+;
```

```
" WHERE "+ALLTRIM(tmp3.clave)+"="+IIF(tmp3.tipodato="C","'","");
  +IIF(tmp3.tipodato="C",LEFT(EVALUATE(tmp3.clave),tmp3.longitud),;
ALLTRIM(STR(EVALUATE(ALLTRIM(tmp3.clave)))))+IIF(tmp3.tipodato="C","'","")
=SQLEXEC(con2,comandosql,"tmp1")
SELECT tmp1
COUNT TO ncont
pase=.F.
DO CASE
  CASE tmp4.operacion="I"
    * Transacción insertar
    IF tmp3.primario
      * Tablas con clave única
      IF ncont=0
        * Insertar registro
        comandosql="INSERT INTO "+ALLTRIM(tmp3.tabla)+" ("+FIELD(1,"tmp1")
        FOR cont1=2 TO FCOUNT("tmp1")
          comandosql=comandosql+","+FIELD(cont1,"tmp1")
        NEXT
        DO CASE
        CASE TYPE(FIELD(1,"tmp1"))="N" AND ncont1=0
          comandosql=comandosql+") VALUES(0"
        CASE (TYPE(FIELD(1,"tmp1"))="D" OR TYPE(FIELD(1,"tmp1"))="T") AND ncont1=0
          comandosql=comandosql+") VALUES(CTOD('  /  /    ')"
        CASE TYPE(FIELD(1,"tmp1"))="L" AND ncont1=0
          comandosql=comandosql+") VALUES(.F."
        OTHERWISE
          comandosql=comandosql+") VALUES(?tmp2."+FIELD(1,"tmp1")
        ENDCASE
        FOR cont1=2 TO FCOUNT("tmp1")
          DO CASE
          CASE TYPE(FIELD(cont1,"tmp1"))="N" AND ncont1=0
```

```
            comandosql=comandosql+",0"
          CASE (TYPE(FIELD(cont1,"tmp1"))="D" OR ;
          TYPE(FIELD(cont1,"tmp1"))="T") AND ncont1=0
            comandosql=comandosql+",CTOD('  /  /    ')"
          CASE TYPE(FIELD(cont1,"tmp1"))="L" AND ncont1=0
            comandosql=comandosql+",.F."
          OTHERWISE
            comandosql=comandosql+",?tmp2."+FIELD(cont1,"tmp1")
          ENDCASE
        NEXT
        comandosql=comandosql+")"
        IF SQLEXEC(con2,comandosql)<0
          DO fallo WITH con1,con2
          RETURN
        ENDIF
        pase=.T.
      ELSE
          * Actualización si ya está insertado
          exp1=IIF(tmp3.tipodato="C",'"',"")+;
            IIF(tmp3.tipodato="C",LEFT(tmp4.valor,tmp3.longitud),;
            ALLTRIM(tmp4.valor))+IIF(tmp3.tipodato="C",'"',"")
          comandosql="UPDATE "+ALLTRIM(tmp3.tabla)+;
            " SET "+FIELD(1,"tmp1")+"=?tmp2."+FIELD(1,"tmp1")
          FOR cont1=2 TO FCOUNT("tmp1")
            comandosql=comandosql+","+FIELD(cont1,"tmp1")+"=?tmp2."+FIELD(cont1,"tmp1")
          NEXT
          comandosql=comandosql+" WHERE "+ALLTRIM(tmp3.clave)+"="+exp1
          IF SQLEXEC(con2,comandosql)<0
            DO fallo WITH con1,con2
            RETURN
          ENDIF
```

```
            pase=.T.
          ENDIF
        ENDIF
      CASE tmp4.operacion="U"
        * Transacción actualizar
        IF tmp3.primario
          * Tablas con clave única
        exp1=IIF(tmp3.tipodato="C","'","")+;
          IIF(tmp3.tipodato="C",LEFT(tmp4.valor,tmp3.longitud),;
          ALLTRIM(tmp4.valor))+IIF(tmp3.tipodato="C","'","")
          exp2=IIF(tmp3.tipodato="C","'","")+;
          IIF(tmp3.tipodato="C",LEFT(tmp4.valorant,tmp3.longitud),;
          ALLTRIM(tmp4.valorant))+IIF(tmp3.tipodato="C","'","")
        DO CASE
        CASE ncont=0 AND exp1#exp2
          * Cambio el campo primario
          comandosql="UPDATE "+ALLTRIM(tmp3.tabla)+" SET "+FIELD(1,"tmp1")+;
            "=?tmp2."+FIELD(1,"tmp1")
          FOR cont1=2 TO FCOUNT("tmp1")
            comandosql=comandosql+","+FIELD(cont1,"tmp1")+"=?tmp2."+FIELD(cont1,"tmp1")
          NEXT
          comandosql=comandosql+" WHERE "+ALLTRIM(tmp3.clave)+"="+exp2
          IF SQLEXEC(con2,comandosql)<0
            DO fallo WITH con1,con2
            RETURN
          ENDIF
          pase=.T.
        CASE ncont=1
          * Actualización de otros campos
           comandosql="UPDATE "+ALLTRIM(tmp3.tabla)+" SET "+FIELD(1,"tmp1")+;
             "=?tmp2."+FIELD(1,"tmp1")
```

```
            FOR cont1=2 TO FCOUNT("tmp1")
              comandosql=comandosql+","+FIELD(cont1,"tmp1")+"=?tmp2."+FIELD(cont1,"tmp1")
            NEXT
            comandosql=comandosql+" WHERE "+ALLTRIM(tmp3.clave)+"="+exp1
            IF SQLEXEC(con2,comandosql)<0
              DO fallo WITH con1,con2
              RETURN
            ENDIF
            pase=.T.
          ENDCASE
          ENDIF
        CASE tmp4.operacion="D"
  * Transacción eliminar
          IF tmp3.primario
            * Tablas con clave única
            IF ncont=0
              comandosql="DELETE FROM "+ALLTRIM(tmp3.tabla)+;
              " WHERE "+ALLTRIM(tmp3.clave)+"="+;
                IIF(tmp3.tipodato="C","'","")+IIF(tmp3.tipodato="C",;
                LEFT(tmp4.valor,tmp3.longitud),ALLTRIM(tmp4.valor))+IIF(tmp3.tipodato="C","'","")
                IF SQLEXEC(con2,comandosql)<0
                  DO fallo WITH con1,con2
                  RETURN
                ENDIF
                pase=.T.
              ENDIF
            ENDIF
        ENDCASE
        IF pase
          IF SQLEXEC(con1,;
          "UPDATE rp_control SET semaforo=.T. WHERE tabla=?tmp3.tabla AND fecha=?tmp4.fecha")<0
```

```
            DO fallo WITH con1,con2
            RETURN
          ENDIF
        ENDIF
        SELECT tmp4
        SKIP
      ENDDO
      SELECT tmp3
      SKIP
    ENDDO

    * Eliminando los registros de transacciones que se replicaron con éxito
    IF SQLEXEC(con1,"DELETE FROM rp_control WHERE semaforo")<0
      =MESSAGEBOX("Fallo Crítico",64,"Replicación")
      =SQLDISCONNECT(con1)
      =SQLDISCONNECT(con2)
      RETURN
    ENDIF

    * Eliminando los registros de transacciones que se crearon al realizar la replicación
    IF SQLEXEC(con2,;
    "DELETE FROM rp_control WHERE fecha>=?fintabla AND HOUR(fecha)>=HOUR(?fintabla)")<0
      =MESSAGEBOX("Fallo Crítico",64,"Replicación")
      =SQLDISCONNECT(con1)
      =SQLDISCONNECT(con2)
      RETURN
    ENDIF

    =SQLDISCONNECT(con1)
    =SQLDISCONNECT(con2)
```

```
CLOSE ALL

PROCEDURE fallo
PARAMETERS xcon1,xcon2
  =MESSAGEBOX("Fallo de Actualización de Registro",16,"Replicación")
  IF SQLEXEC(xcon1,"DELETE FROM rp_control WHERE semaforo")<0
    =MESSAGEBOX("Fallo Crítico",64,"Replicación")
    =SQLDISCONNECT(xcon1)
    =SQLDISCONNECT(xcon2)
    RETURN
  ENDIF
  IF SQLEXEC(xcon2,;
  "DELETE FROM rp_control WHERE fecha>=?fintabla AND HOUR(fecha)>=HOUR(?fintabla)")<0
    =MESSAGEBOX("Fallo Crítico",64,"Replicación")
    =SQLDISCONNECT(xcon1)
    =SQLDISCONNECT(xcon2)
    RETURN
  ENDIF
  =SQLDISCONNECT(xcon1)
  =SQLDISCONNECT(xcon2)
RETURN
```

«conexión1» y «conexión2» son orígenes de datos ODBC a las bases de datos remota y local respectivamente.

4.2.2. Un sistema completo de replicación

Para tener un sistema completo de replicación, el primer paso es tener un proyecto que funcione correctamente desarrollado en Visual FoxPro, creamos las tablas de control y

de tablas a replicar (en nuestro aplicativo están con los nombres de rp_control y rp_tablas respectivamente) en la base de datos de la aplicación, escribimos las funciones para los disparadores en los procedimientos almacenados de la base de datos, definimos los triggers en las tablas que vayan a participar en la replicación, y por último ingresamos la información de estas tablas en rp_tablas. Instalamos la aplicación en nodos remotos (físicamente distantes y no conectados), e iniciamos a trabajar. Para replicar datos, establecemos conexión con los nodos remotos, realizamos ODBC a las bases de datos locales y remotas, y ejecutamos el programa anteriormente descrito.

Hay que tener en cuenta que las claves no pueden duplicarse en las bases de datos, un típico ejemplo es un sistema de facturación, en el que el número de factura es la clave principal, entonces debemos definir que cada nodo remoto tenga su rango de número de facturas. Aquí intervienen las reglas del negocio, que son definidas por el administrador o normas de la empresa, por ejemplo, una sucursal trabaja con números de factura de 1 a 100.000, en cambio en el principal usará de 1'000.000 a 1'100.000 por ejemplo, así evitamos que se dupliquen las claves.

4.2.3. Replicap

Replicap, es un paquete diseñado por los autores de este trabajo, basado en los algoritmos anteriormente descritos, se está desarrollando como software comercial, para aplicaciones basadas en Bases de Datos de Visual FoxPro o PostgreSQL.

Replicap instala los procedimientos almacenados, triggers, tablas de control para replicación, en bases de datos Visual FoxPro o PostreSQL, preparándolas para que puedan replicar datos.

Además replicap utiliza cuatro métodos para replicar datos: desde Visual FoxPro hacia Visual FoxPro, desde Visual FoxPro hacia PostgreSQL, desde PostgreSQL hacia Visual FoxPro y desde PostgreSQL hacia PostgreSQL.