



UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

“REPLICACION SIMETRICA, UN CASO AVANZADO DE LAS BASES DE DATOS ORACLE. UN EJEMPLO DE SU APLICACION”

T E S I S

QUE PARA OBTENER EL TITULO DE: INGENIERO EN COMPUTACION

P R E S E N T A N:

GARCIA MACIAS, KASUO CRISTOBAL

TAPIA CORONA RODRIGO

287664

DIRECTOR DE LA TESIS: DRA. ANA MARIA VAZQUEZ VARGAS



MEXICO D.F.

ENERO DE 2001



Universidad Nacional
Autónoma de México

Dirección General de Bibliotecas de la UNAM

Biblioteca Central



UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Agradecimientos

A Reyna y Gilberto con cariño, por su dedicación y cuidado de padres, siendo el pilar mas importante para conseguir esta meta.

A Vero, Lili y Forti por ser tan comprensivos con todos los que los rodean

A mi Abuelita y mis tías que siempre me consintieron y me dieron de todo desde que tengo memoria.

A Xochitl por su constante apoyo y por la felicidad que me brinda desde el día que la conocí.

A la Dra. Ana María Vázquez. Por el invariable apoyo, flexibilidad, consideración y asesoría otorgada bajo condiciones tan especiales.

A mi tío Francisco por ser el modelo que me inspiró desde pequeño para convertirme en ingeniero.

Rodrigo Tapia Corona.

Agradecimientos

A mis padres con mucho cariño por todos sus esfuerzos y sacrificios que realizaron para sacarme adelante, por sus horas de trabajo, por enseñarme a vivir, porque cuando los he necesitado siempre están ahí sabiendo como animarme y enseñándome el camino para seguir creciendo. Por ser mis padres, y por todas las cosas que me han dado y que nunca terminare de agradecerles.

A mi hermana. Por el ejemplo, por latosa y por ser mi hermana.

A mis Abuelos quienes me dieron unos padres excelentes y me brindaron su apoyo en todo momento.

A Angie. Por su ayuda incondicional y por las desveladas que compartimos Juntos.

A la Dra. Ana María Vázquez. Por el apoyo brindado en el desarrollo de este trabajo, por la presión ejercida, por los conocimientos que tan amablemente compartió conmigo, por su disponibilidad.

Kasuo Cristobal Garcia Macias

Contenido

Agradecimientos

Contenido

Introducción 1

Capítulo 1. Introducción a las Bases de Datos Distribuidas

1.1 Sistemas Manejadores de Bases de Datos Distribuidas 7

1.2 Reglas de Distribución para los Manejadores de Bases de Datos 10

Capítulo 2. Esquema Cliente / Servidor y Base de Datos Distribuidas en Oracle

- 2.1 Arquitectura Cliente / Servidor 19
- 2.2 SQL*NET 24
- 2.3 Comunicación entre Base de Datos 26
- 2.4 Arquitectura de las Bases de Datos Distribuidas 29
- 2.5 Base de datos distribuidas y replicación de Base de Datos 31

Capítulo 3. Replicación de Base de Datos en Oracle

- 3.1 La Replicación 33
- 3.2 Replicación de datos Síncrona 34
- 3.3 Replicación de datos Asíncrona 36
- 3.4 La Replicación Básica de Oracle 37
- 3.5 Distribución de la Información 38
- 3.6 Tablas Snapshot de solo lectura 40
- 3.7 Refresco de Sanpshots 43
- 3.8 Replicación Simetrica 51
- 3.9 Esquemas de la Replicación Simetrica 51
- 3.10 Manejo de la Replicación Simetrica 57
- 3.11 Resolución de Conflictos 62

Capítulo 4. Caso de Estudio “Voyager”

- 4.1 Descripción General de Voyager 65
- 4.2 Módulo de Información del Cliente 66
- 4.3 Módulo de Manejo de Problemas 69
- 4.4 Manejo de Envíos de Productos 72
- 4.5 Operación de Voyager en el área 74
- 4.6 Tiempos de Respuesta 77
- 4.7 Topología de la Red 78

Capítulo 5. Implementando la Replicación Simétrica

- 5.1 Análisis del Modelo Entidad Relación
- 5.2 Identificación de las tablas de solo lectura
- 5.3 Implementando la Replicación de solo lectura
- 5.4 Identificación de las tablas del Esquema Simétrico
- 5.5 Implementación de la Replicación Simétrica
- 5.6 Administración del Sistema Replicado

Capítulo 6. Conclusiones

- 6.1 Desventajas observadas después de implementar la replicación
- 6.2 Ventajas observadas después de la implementar la replicación
- 6.3 Conclusiones finales

Bibliografía.

Apéndices.

A. Commit de dos fases

B. Propiedades ACID de las transacciones

Introducción

Hoy día, varias empresas deben de contar con la capacidad de crear y mantener múltiples copias de información redundante, lo cual se conoce como la replicación de información. Estas copias redundantes son típicamente usadas para mejorar la disponibilidad de la información y el performance (la capacidad para desempeñar cierta tarea dada) de las aplicaciones en los sites¹ remotos, evitando el procesamiento de queries en los sistemas de producción, por otro lado estas copias también son alternativas de recuperación tipo warm-standby (base de datos con los cambios recientes lista para ser puesta a disposición de los usuarios). Si estas copias de datos no son consistentes, los usuarios puede realizar decisiones incorrectas.

¹ En el desarrollo esta tesis determinamos nodo de una red de computadoras como **site**

El mantener y administrar estas copias se torna complejo y consume gran cantidad de diversos recursos. Estas necesidades han sido cubiertas en el mercado por varios proveedores, que han desarrollado software de replicación el cual provee ayuda en el manejo de este tipo de copiado. Para reducir el riesgo, la información redundante que se almacena debe ser consistente con todas las copias en todos los tiempos o debe de tener un nivel aceptable de inconsistencia para lapsos cortos de tiempo. Una copia inconsistente puede hacerse consistente sincronizando la información con la que se ha designado como fuente primaria, esta es, la copia maestra, o resolviendo cualquier conflicto que exista entre las copias.

El mantener datos consistentes a través de todas las copias redundantes almacenadas es una tarea muy complicada, no obstante esta se puede alcanzar de manera eficiente y efectiva usando software de replicación. Una clave para usar estos productos de manera exitosa radica en analizar y distribuir la información en la empresa de manera estratégica, de este modo se optimiza la red, el hardware y el tiempo de administración.

Cualquier situación o ambiente que cumpla con alguno de los siguientes criterios, puede ser solucionada con el uso de software de replicación, por ejemplo:

- Se han distribuido información y/o procesos en un ambiente de sites múltiples y se desea mejorar la disponibilidad de la información y el performance de las aplicaciones.
- Se tiene la necesidad de contar con un esquema de recuperación warm-standby.

- Se tienen ambientes de distribución de información, los cuales se hacen consistentes vía una total extracción y refresco con el deseo de reducir el costo existente en la red WAN y mejorar el tiempo para que la información este disponible.
- Se tienen sistemas de soporte a decisiones OLAP (Online Analytical Process), que requieren sincronía de información, con sistemas tipo OLTP (Online transaction Process) de transacciones en línea, que trabajan en tiempo real.
- Se requiere consolidar información a partir de múltiples localidades que procesan y capturan información.

Como se puede observar la mayoría de compañías medianas y grandes pueden llegar ha encontrarse en estas situaciones y por ende a ser candidatas para servicios de replicación.

El caso de prueba de la presente tesis, trata sobre como implementar un esquema de replicación simétrica para que un sistema que opera en producción de manera remota y centralizada, pueda ser operado desde de forma distribuida entre diferentes localidades geográficas, mejorando los tiempos de respuesta para diferentes áreas y manteniendo copias de las bases de datos de manera distribuida a lo largo de la región.

Este caso de prueba se basa en una compañía proveedora de productos y servicios para el manejo de información, la cual centralizo en Estados Unidos, el servicio de soporte que originalmente se brindaba localmente en varios países latinoamericanos.

Esta decisión implicó que ciertas operaciones alrededor del servicio y de la compañía se concentraran y otras se distribuyeran entre el centro de soporte y con cada una de las subsidiarias en el país, una de estas operaciones, es lo que comprende las actividades realizadas por el área de relaciones con clientes, la cual fue transformada.

Por un lado los analistas del área fueron centralizados, mientras que por otro la base de datos concerniente a los clientes permaneció a nivel local en las oficinas de la subsidiaria en cada país.

Voyager 2000 es el sistema que mantiene y controla la información en la base de datos de cada uno de los clientes de la compañía en cada país, este sistema opera en modo cliente - servidor. Una vez con la creación del centro de soporte, el usuario del centro quedó muy distante del site donde reside la base de datos, esto implicó que los tiempos de respuesta fueran inadecuados disminuyendo la productividad y la eficiencia del área de relaciones con clientes.

El objetivo de la tesis es el demostrar como a través del uso de la Replicación Simétrica se pueden mejorar los tiempos de respuesta para diferentes áreas, manteniendo copias de las bases de datos replicadas en forma distribuida a lo largo de la región, y las implicaciones después de la implantación.

Para cumplir este objetivo, la tesis se organizó de la siguiente forma:

Capítulo 1, " Introducción a las Bases de Datos Distribuidas ", se define el marco teórico de las bases de datos distribuidas, el cual comprende la revisión de las doce reglas de distribución definidas por C. J. Date.

Capítulo 2, " Esquema Cliente / Servidor y Bases de Datos Distribuidas en Oracle ", se describen las características de las bases de datos distribuidas y el esquema Cliente / Servidor bajo Oracle.

Capítulo 3, " Replicación de Base de Datos Oracle ", se describen y analizan los diferentes tipos de replicación; síncrona y asíncrona, al igual que el manejo de la replicación en Oracle

Capítulo 4, " Caso de Estudio Voyager ", se describe el sistema y la utilización de mismo de manera distribuida a lo largo de Latino América y problemas relacionados con los tiempos de respuesta del sistema.

Capítulo 5, " Implementando la Replicación Simétrica ", se describe como se colocaron varias tablas a ser replicadas en modos de lectura y en modo de escritura, entre dos bases de datos remotas, configurando el esquema de Replicación Simétrica de Base de Datos.

Capítulo 6, " Conclusiones ", se analizan los resultados obtenidos con respecto a los tiempo de acceso anteriores y después de implementar el esquema de replicación.

1

Introducción a las Bases de Datos Distribuidas

1.1 **Sistemas Manejadores de Bases de Datos Distribuidas**

El uso de los ambientes de cómputo distribuido en la actualidad es bastante generalizado. En este tipo de ambientes los usuarios comparten periféricos, información y programas gracias a que la infraestructura, disponible de hoy en día, facilita el manejo de los sistemas a través de los diferentes

componentes de redes, hardware y software. Las bases de datos han sido una parte importante en la evolución de la distribución.

Una definición genérica de una base de datos distribuida¹ es que no esta almacenada totalmente en un solo *site*, sino al contrario, esta esparcida en varios sites geográficamente dispersos, los cuales están conectados a través de algún medio de comunicación.

Por ejemplo un banco determinado, ubicado en una ciudad A, que opera un sistema en el cual los registros de las cuentas de la ciudad A se mantienen en una base de datos en A, por otro lado los registros de las cuentas de la ciudad B se mantienen en otra base de datos ubicada en la ciudad B, y ambas bases de datos están conectadas para formar una sola base de datos "global" o distribuida.

Observando este ejemplo detalladamente, se identifica que un sistema de bases de datos distribuidas consiste de una colección de *sites* o nodos conectados a través de algún tipo de red de comunicación, en el cual cada *site* es un sistema de bases de datos en sí, pero los *sites* están configurados para trabajar de manera conjunta, de tal forma que cualquier usuario de cualquiera de los servidores, puede tener acceso a la información en la red, exactamente como si la información estuviera almacenada en el *site* o lugar donde el usuario en cuestión se encuentre.

¹ Base de datos distribuidas, Date vol2 [7.1]

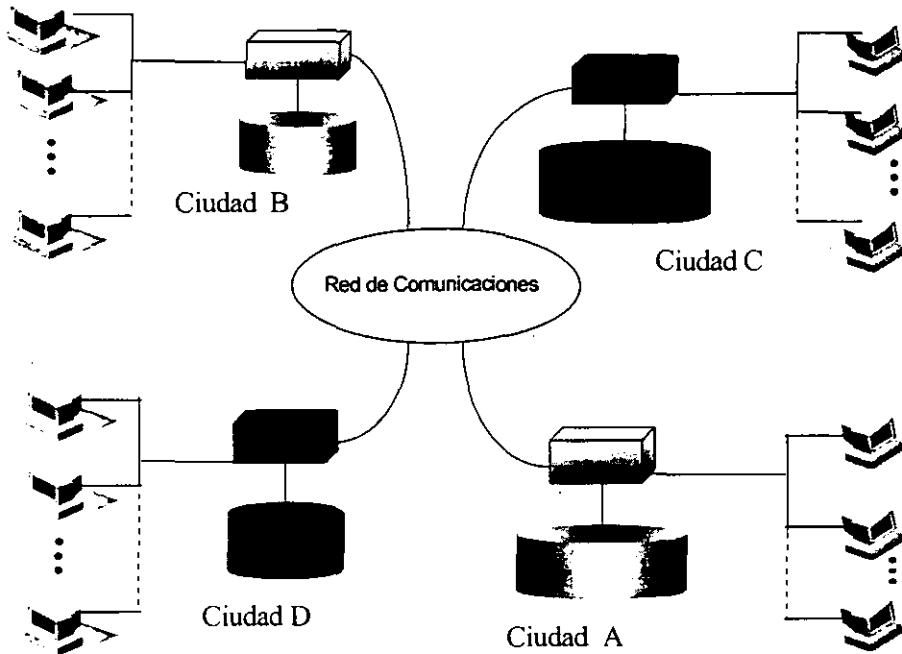


Figura 1.1 Un sistema típico de bases de datos distribuidas

De aquí se desprende que una "base de datos distribuida" es en realidad un tipo de objeto virtual, cuyas partes que lo componen están físicamente almacenadas en distintas bases de datos "reales" en distintos *sites* (de hecho, es la unión lógica de tales bases de datos reales).

La Fig. 1.1 muestra el ejemplo, del cual se puede observar que la información también está distribuida, ya que cada unidad o área dentro de la empresa o banco, etc., necesariamente mantendrá información que es relevante para su propia operación. De este modo, un sistema distribuido permite que la estructura de la base de datos refleje la estructura de la empresa: información local es guardada localmente, donde lógicamente pertenezca más, mientras que al mismo tiempo la información remota puede ser accedida cuando sea necesario.

1.2 Reglas de Distribución para los Manejadores de Bases de Datos

El principio fundamental de una base de datos distribuida es:

Un sistema distribuido debe de dar la apariencia como si NO fuera un sistema distribuido.

Este principio da pie a los siguientes postulados¹ o reglas que un sistema de bases de datos distribuidas debe de cumplir. A continuación se describen estos:

1. Autonomía Local.

Los *sites* en un sistema distribuido deben de ser autónomos. La autonomía local significa que todas las operaciones en un *site* dado son controladas por ese *site*; un *site* "x" no debería depender de otro *site* "y" para que su operación sea exitosa.

Autonomía Local también implica que la información local es poseída y manipulada localmente. Toda la información pertenece "realmente" a una base de datos local, aun si esta es accesible desde otros *sites* remotos. Temas como la seguridad e integridad de la información local debe de permanecer bajo control u jurisdicción del *site* local.

Actualmente, el objetivo de la autonomía local no es totalmente factible; existe un número de situaciones en las cuales un *site* "x" debe de delegar un cierto grado de control a otro *site* "y". El objetivo de la autonomía, de manera

¹ Principio Fundametal, Date vol 1 [21.2]

mas precisa, debería establecerse como: Los sites deben de ser autónomos al mayor nivel posible.

2. *No depender o confiar en un site central.*

Un sistema manejador de bases de datos distribuidas no deberá depender de un site central ya que esto crearía un punto muy susceptible de falla para todo el sistema. Este site central podría también convertirse en un cuello de botella con respecto al performance.

La autonomía local implica que "todos los sites deben ser tratados por igual", así que no debe de haber alguna dependencia de un site central o maestro para servicios como el procesamiento de consultas, manejo de transacciones o servidor de nombres, lo cual implicaría dependencia".

3. *Operación continua.*

Un sistema de bases de datos distribuidas nunca debería de requerir ser dado de baja o ser puesto fuera de operación.

Una ventaja de los sistemas distribuidos en general es que estos proveen mayor fiabilidad y mas disponibilidad.

- Fiabilidad (la cual puede ser definida como la probabilidad de que el sistema esta operando en cualquier momento dado). Debido a que los sistemas distribuidos son soluciones flexibles, estos pueden continuar operando (a cierto nivel) aun en la presencia de alguna falla de los sites.

- Disponibilidad (la cual puede ser definida como la probabilidad de que el sistema este funcionando durante un período específico), con respecto a la distribución, la posibilidad de replicar información mejora la disponibilidad de todo el sistema en general.

4. *Transparencia e independencia de ubicación.*

La idea básica es que los usuarios no debe de saber donde esta físicamente almacenada la información, no obstante el sistema debe dar la impresión — al menos desde el punto de vista lógico — como si toda la información estuviera en su site local. La independencia de ubicación es deseable ya que simplifica los programas de los usuarios y las interfaces finales. En particular, permite migrar información de site en site sin invalidar alguno de los programas o actividades. Tal capacidad de migración es deseable ya que permite que la información se transfiera en la red en respuesta a los requerimientos cambiantes de performance.

5. *Independencia de fragmentación*

Un sistema que soporta la fragmentación de información es aquel que dada una tabla, esta puede ser dividida en piezas o "fragmentos" para propósitos de almacenamiento físico. La fragmentación es conveniente por razones de performance. La información puede ser almacenada en el lugar donde es mas frecuentemente es usada, de tal forma que la mayoría de las operaciones son puramente locales y el tráfico en la red se minimiza.

Existe básicamente dos tipos de fragmentación, horizontal y vertical, correspondientes a las operaciones de restricción y proyección,

respectivamente. De manera mas general, un fragmento puede ser cualquier sub relación arbitraria que sea derivada a partir de la relación original a través de operaciones similares.

Un sistema que soporta la fragmentación de información debe también de soportar la independencia de fragmentación, (como la independencia de ubicación) es conveniente ya que simplifica la programación. En particular, permite que la información sea fragmentada en cualquier momento (y que los fragmentos sean distribuidos en cualquier momento) en respuesta a requerimientos cambiantes de performance, sin invalidar alguno de los programas.

La independencia de fragmentación implica presentar una vista de la información en la cual los fragmentos están lógicamente combinados a través de joins y uniones apropiadas. Es responsabilidad del sistema que optimiza el determinar que fragmentos necesitan tener acceso para satisfacer cualquier petición dada.

Por otro lado, el problema de soportar operaciones en tablas fragmentadas tiene ciertos puntos en común con el problema de soportar operaciones sobre vistas con join y uniones.

6. Independencia de Replicación.

Un sistema soporta la replicación de información si un fragmento dado puede ser representado por varias copias o replicas, almacenadas en los distintos sites. La replicación es conveniente por lo menos por dos razones: primero, puede significar un mejor performance (las aplicaciones pueden operar con copias locales en lugar de tener que comunicarse con sites remotos); segundo, también puede significar una mayor disponibilidad (un objeto replicado

dado permanece disponible para procesamiento mientras que al menos una copia permanezca disponible, al menos para propósitos de consulta). La mayor desventaja de la replicación, desde luego, es cuando un objeto replicado es actualizado, las actualizaciones deben de propagarse a todos los objetos.

La replicación como la fragmentación, idealmente debe ser transparente, un sistema que soporte replicación deberá de soportar también la independencia de replicación (también conocida como transparencia de replicación).

La independencia de replicación (al igual que la independencia de ubicación y fragmentación) es deseable ya que simplifica la programación; en particular permite la creación y destrucción de replicas en cualquier momento en respuesta a requerimientos cambiantes, sin invalidar alguno de los programas o actividades.

La independencia de replicación implica que es responsabilidad del sistema que optimiza el determinar que replicas físicamente requieren tener acceso para satisfacer la petición.

7. Procesamiento de queries distribuidos.

La optimización es mas importante en sistemas distribuidos que en un sistema centralizado. El punto básico es que, en un query que involucre a varios sites, existirán varias maneras posibles de mover la información en la red para satisfacer la petición, y es de crucial importancia que una estrategia eficiente sea elegida. Por ejemplo, un petición dígase la unión de una tabla Tx almacenada en el site X y una tabla Ty almacenada en el site Y puede llevarse acabo moviendo los datos de Rx a X o moviendo ambos a un tercer site Z, etc.,

con esta serie de combinaciones la optimización se torna crucial debido a la diferencia de tiempos de respuesta que cada una de las opciones pueden proporcionar.

8. Manejo de transacciones distribuidas.

Un sistema distribuido deberá soportar las propiedades ACID (ver apéndice A) de las transacciones. Deberá proveer soporte para peticiones remotas, unidades de trabajo remotas, unidades de trabajo distribuidas y peticiones distribuidas.

Existen dos aspectos principales en el manejo de transacciones, el control en la recuperación y el control en la concurrencia. Para comprender este tipo de controles es necesario primero entender lo que es un "agente". En un sistema distribuido, una sola transacción puede involucrar varias actualizaciones en varios sites. Dicho así cada transacción consiste de varios agentes, donde un agente es el proceso realizado en base a una transacción dada en un site dado. Y el sistema necesita conocer cuando dos agentes son ambos parte de la misma transacción, por ejemplo, dos agentes que son parte de la misma transacción obviamente deben de evitar colocarse mutuamente un "dead lock" (candado mortal).

Regresando al control en la recuperación. Para asegurar que una transacción es atómica (se realiza o toda o nada) en el sistema distribuido, entonces, el sistema debe de asegurarse que el conjunto de agentes para tal transacción deben de realizar "commit" (confirmación de la transacción) al unísono o "rollback" (deshacer una transacción realizada). Este resultado puede lograrse a través del mecanismo de "commit en dos partes" (ver apéndice).

El control en la concurrencia en la mayoría de los sistemas distribuidos esta basado en el bloqueo o la colocación de candados, al igual que en los sistemas no distribuidos.

9. *Independencia de hardware.*

Con respecto a esto, en las instalaciones comerciales, típicamente se involucran una multiplicidad de diversas maquinas IBM, Sun, HP, PCs, etc, la información en estos sistemas deben de presentar una sola imagen del todo sistema. Por lo tanto es conveniente el correr el mismo DBMS (Sistema que maneje las bases de datos) en diferentes plataformas, y mas aun el hacer partícipes por igual a todas maquinas en un sistema distribuido.

10. *Independencia del sistema operativo.*


Este principio es parcialmente un corolario del anterior. Es obviamente deseable, el no solo utilizar el mismo DBMS en diferentes plataformas, sino también que opere en diferentes sistemas operativos, de tal forma que sistemas operativos como Unix, MVS y Windows participen en el mismo sistema distribuido.

11. *Independencia de la Red.*

Este principio al igual que los anteriores se torna obvio, si el sistema puede soportar diferentes sites con diverso hardware y diversos sistemas operativos también deberá de soportar diversos tipos y protocolos de redes de comunicación.

12. Independencia de DBMS.

El soporte heterogéneo es definitivamente deseable. El hecho es que en el mundo real típicamente no solo se opera con maquinas diferentes sino con DBMS diferentes. Por ejemplo si ambos Oracle y Sybase soportan SQL que es el estándar oficial, entonces se puede concebir un sistema que en un lado se tenga Oracle y en otro site Sybase y que ambos se puedan comunicar mutuamente en el contexto de un sistema distribuido. En otras palabras, podría ser posible que el sistema distribuido sea heterogéneo hasta cierto grado.



2 Sistemas de Base de Datos Distribuidas en Oracle.

2.1 Arquitectura Cliente Servidor.

Si una organización es como la mayoría de las empresas de hoy en día, con los centros de información distribuidos geográficamente, al igual que los clientes y oportunidades de negocios, se hace necesaria una solución tecnológica en informática que cubra todos los factores críticos de éxito.

Con el paso de los años y los adelantos en la tecnología, la forma de procesar los datos dentro de las compañías y la forma de utilizar los resultados obtenidos ha tenido un constante cambio.

El éxito futuro de las compañías y su permanencia en el mercado está directamente relacionado con la capacidad de adecuación de estas nuevas tecnologías y su correcta utilización para satisfacer las necesidades de información dentro de su empresa.

2.1.1 La arquitectura cliente/servidor

La arquitectura cliente/servidor es una herramienta adecuada de solución, ya que no sólo es flexible y segura, sino que también protege la inversión en tecnología y permite manejar diferentes ambientes de computación, tal como se ilustra en la figura 2.1.1

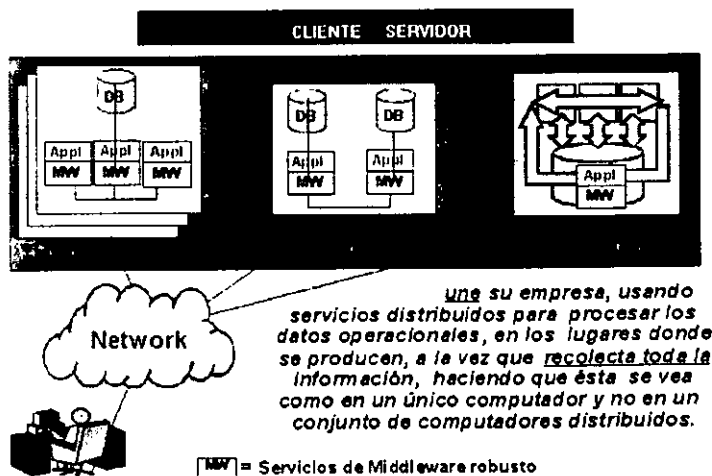


Figura 2.1.1

Flexibilidad : La infraestructura tecnológica debe soportar crecimientos y cambios rápidos, de manera que la empresa esté en capacidad de reaccionar, de manera oportuna, en el proceso de recolección y acceso de la información importante para su funcionamiento y crecimiento. Debe estar en capacidad de adicionar nuevas soluciones en forma efectiva, eficiente y tan transparente como sea posible.

Seguridad : La infraestructura informática debe ser segura contra fallas en componentes, pérdida de información, control de acceso, entre otros. Asimismo, se necesita un nivel de seguridad, como el que brindaban los mainframes, pero en ambientes de sistemas abiertos.

Protección de la inversión y control de costos : Es importante mantener la actual inversión en tecnología. La empresa no desea desechar tecnología que está actualmente trabajando y funcionando, así; como tampoco es deseable estar constantemente haciendo reingeniería de procesos, documentando y entrenando nuevamente.

2.1.2 Características del Cliente/Servidor

Dentro de las características que debe cumplir una aplicación cliente/servidor están las siguientes:

Balancear las cargas de trabajo entre los elementos de computación disponibles.

Manejo de mensajes, que le permite entrar en el modo conversación de un esquema Cliente/Servidor y en general de cualquier forma de paso de mensajes entre procesos.

Administración Global, como una sola unidad cómputo lógica.

Manejo de la consistencia entre los diferentes servidores de bases de datos principalmente en los procesos de OLTP.

Administración de la alta disponibilidad de la solución.

La tecnología Cliente/Servidor ha brindado gran ayuda en los niveles de grupos de trabajo y soluciones departamentales. Mediante el despliegue y aprovechamiento de los lenguajes de cuarta generación y las herramientas para desarrollos orientados por objetos se ha logrado un importante cambio en las aplicaciones Cliente/Servidor pasando de simples soluciones, usando bases de datos, a soluciones de mediana escala.

Esta combinación de bases de datos y herramientas son altamente efectivas para las aplicaciones que están orientadas a los datos. Donde la principal tarea de este modelo, es simplemente lograr que los datos sean alcanzados, y en el mejor de los casos hacer un despliegue de estos y/o una simple actualización. Esto se puede ver gráficamente en la figura 2.1.2.

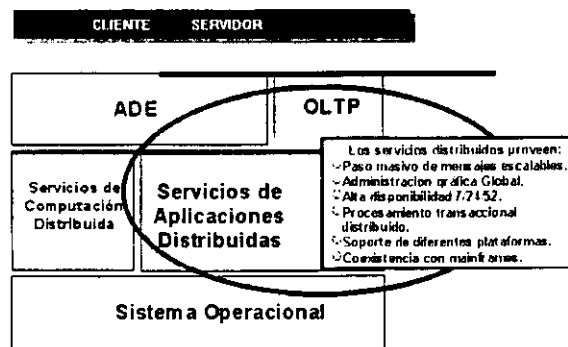


Figura 2.1.2

Aplicaciones más complejas, que van más allá del trabajo en grupo o del departamento, hacen grandes demandas de las bondades de la tecnología disponibles hoy en día. En estos altos niveles, los requerimientos están en alcanzar la información, moverla dentro de la empresa y utilizarla en forma adecuada.

Los beneficios para el negocio del uso de la tecnología Cliente/Servidor son bien conocidos: alta productividad de los programadores, bajo costo/alto rendimiento de las plataformas de sistemas y de las redes de comunicación y un incremento o en la habilidad para construir y entregar soluciones más efectivas para el negocio. Sin embargo, el reto está en poder construir soluciones Cliente/Servidor que logren pasar la barrera del simplemente alcanzar la información.

Frecuentemente la exitosa tecnología Cliente/Servidor falla cuando intenta hacer más de lo que está a su alcance o cuando no se están usando las herramientas adecuadas para llegar más allá de tomar la información; aún cuando se estén usando métodos estándares de construcción de aplicaciones cliente/servidor. La figura 2.1.2.1 ilustra como ayuda la arquitectura cliente servidor en el negocio.

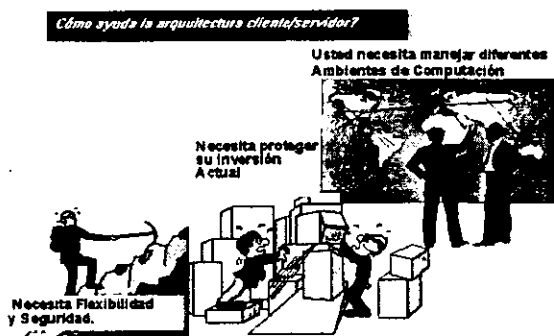


Figura 2.1.2.1

Uno de los elementos críticos que frecuentemente se olvida es el soporte para la creación y modelado de componentes complejos dentro del negocio, la estrategia para distribuir en forma transparente estos soportes en los puntos donde se necesita la información y la carencia de una infraestructura para el manejo en tiempo de ejecución de los componentes que se encuentran distribuidos.

2.2 SQL*NET

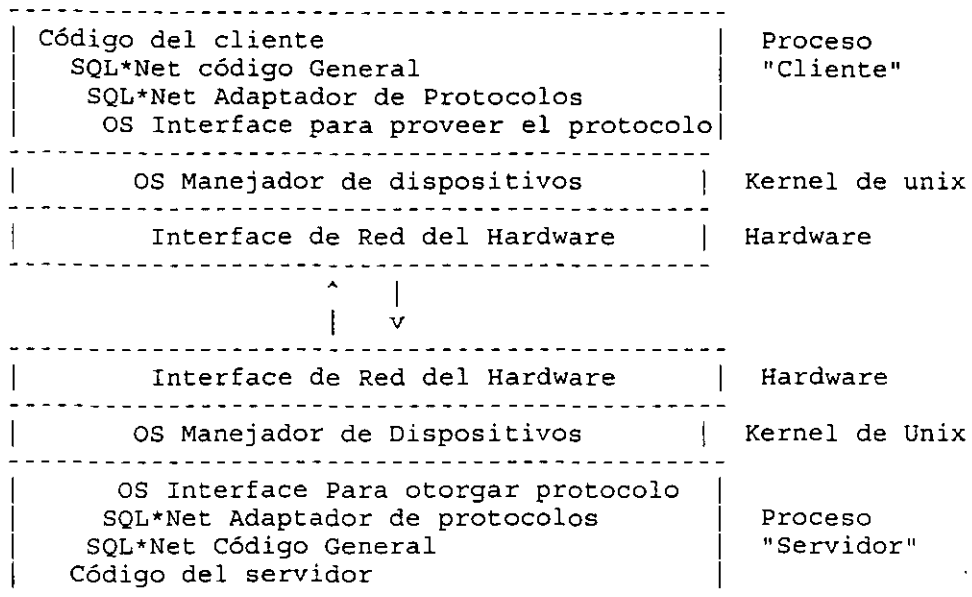
SQL*NET Es el módulo de comunicación sobre Oracle. Este capítulo describe la configuración de SQL*NET y su funcionamiento, a demás también explicaremos como detectar algunos posibles problemas presentados en SQL*NET.

SQL*NET sitúa comunicaciones entre los procesos clientes y servidores de Oracle, sobre algunos medios intermedios, usualmente utilizando un protocolo de red.

El adaptador de protocolos de SQL*NET provee de diferentes interfaces entre el código general de SQL*NET y las interfaces del sistema operativo dando como resultado un protocolo de red mostrado internamente. El "cliente" y el "servidor" mostrados aquí pueden ser procesos sobre ambos que finalmente están ligados a SQL*NET.

Figura 2.2

SQL*NET manejador de interfaces.



(OS = Sistema Operativo)

Sobre maquinas Unix todas las conexiones a la base de datos se realiza utilizando SQL*NET. Las conexiones locales de manera normal utilizan un adaptador de protocolo de SQL*NET llamado adaptador "Bequeath". Este adaptador utiliza pipes de Unix para comunicarse entre los procesos servidores de Oracle y el programa del cliente, en este caso no se presenta la interface del Hardware. Esto lo podemos visualizar en la figura de 2.2

2.2.1 COMPONENTES DE SQL*NET

Los componentes involucrados para estabilizar una conexión se muestran en la tabla 2.2.1. En la mayoría de los casos son requeridos tres componentes para establecer una conexión.

Tabla 2.2.1: Componentes de SQL*NET

Nombre:	Cliente	Listener	Server
Propósito:	Interface del usuario para comunicarse entre el usuario y el RDBMS.	Escuchar para establecer las conexiones.	Interpretar las operaciones de la base de datos y dirigirlas al SQL recividor.
Ejemplo:	sqlplus	tnslsnr	oracle
Archivos de configuración	tnsnames.ora sqlnet.ora	listener.ora	sqlnet.ora

2.3 Comunicación entre bases de datos

Una vez que se conoce la arquitectura cliente servidor y se ha hablado del sql*net el cual es la herramienta que nos va a permitir para el caso de Oracle la comunicación entre las bases de datos es importante hablar de como se realiza esta comunicación. A continuación hablaremos un poco de la configuración requerida para poder comunicar 2 bases de datos Oracle a través de la red.

2.3.1 Bases de datos y ligas.

Cada base de datos en un sistema distribuido es distinto de las otras bases de datos en el sistema y tiene su propio nombre Global. Oracle forma el nombre global de la base de datos con un prefijo del dominio de red con el nombre de la base de datos. Por ejemplo, La figura 2.3.1 ilustra un arreglo a través de la red.

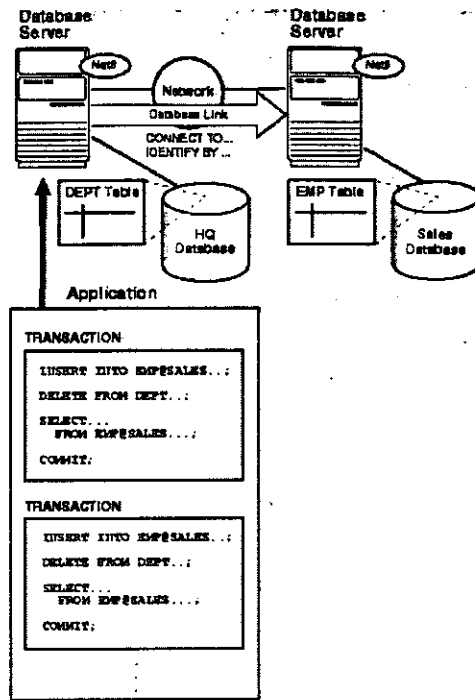


Figura 2.3.1

Mientras varias bases de datos pueden tener el mismo nombre individual, cada base de datos debe tener un único nombre global por ejemplo:

El dominio de red US.AMERICAS.ACME_AUTO.COM y UK.EUROPE.ACME_AUTO.COM cada dominio contiene una base de datos VENTAS.

VENTAS. US.AMERICAS.ACME_AUTO.COM

VENTAS. UK.EUROPE.ACME_AUTO.COM

2.3.2 Ligas de bases de datos.

Para facilitar las peticiones de la aplicación en un sistema de bases de datos distribuido se utilizan ligas hacia las bases de datos. Una liga es definida en un solo sentido. Y es la ruta de comunicación de una base de datos a otra.

Las ligas de bases de datos son esencialmente transparentes para los usuarios porque el nombre de la liga es el mismo que el nombre global de la base de datos al cual apunta la liga.

Por ejemplo, la siguiente sentencia de sql crea una liga en la base de datos local que describe la ruta remota VENTAS.US.AMERICAS.ACME_AUTO.COM

```
create database link VENTAS.US.AMERICAS.ACME_AUTO.COM ...;
```

Después de crear la liga las aplicaciones conectadas a la base de datos local pueden tener acceso a los datos en la base remota VENTAS.US.AMERICAS.ACME_AUTO.COM la siguiente sección explica como las aplicaciones pueden hacer referencia a objetos de esquemas remotos en una base de datos distribuida e incluye algunos ejemplos sobre como las sentencias utilizan las ligas de base de datos.

2.3.3 Resolución del nombre del objeto y esquema.

Para identificar el objeto al cual se esta haciendo referencia y el esquema en una base de datos simple, Oracle resuelve los nombres utilizando una jerarquía de acceso. Por ejemplo en una base de datos Oracle garantiza que cada esquema tiene un nombre único, cada objeto tiene un nombre único.

Como resultado , el nombre del objeto del esquema es siempre único en la base de datos. Por lo tanto Oracle puede fácilmente identificar el objeto referido.

En una base de datos distribuida un objeto es tiene acceso por todas las aplicaciones en el sistema Oracle simplemente extiende el modelo jerárquico agregando un nombre de base de datos global para crear nombres de objetos globales referidos a un esquema dentro de la base de datos distribuida. Por ejemplo, un query puede hacer referencia a una tabla remota especificando su calificador completo y la base de datos en la cual reside:

```
SELECT * FROM scott.emp @ ventas.us.americas.acme_auto.com;
```

Para completar la petición, el servidor local utiliza implícitamente una liga a la base de datos que conecta a la base de datos de ventas.

2.4 Arquitectura de las Base de Datos Distribuidas.

Una base de datos distribuida es un conjunto de bases de datos almacenadas sobre múltiples computadoras que típicamente aparecen para las aplicaciones como una sola base de datos. Consecuentemente una aplicación puede simultáneamente tener acceso y modificar datos en varias bases de dentro de la red. Cada base de datos en el sistema es controlada por su propio RDBMS.

Clientes y Servidores

Un servidor de base de datos es el software de que maneja una base de datos, y un cliente es una aplicación que pide información al servidor.

Cada computadora en el sistema es un nodo. Un nodo en un sistema de base de datos distribuida actúa como un cliente, un servidor, o ambos dependiendo de la situación. Por ejemplo en la figura 2.4 la computadora que maneja la base de datos HQ (Head Quaters) esta actuando como un servidor de base de datos cuando una sentencia es enviada contra los datos locales (por ejemplo, la segunda sentencia en cada transacción envía un query contra la tabla local DEPT), y esta actuando como un cliente cuando la sentencia es remota (por ejemplo, la primer sentencia en cada transacción es enviada contra la tabla remota EMP en la base de datos de ventas).

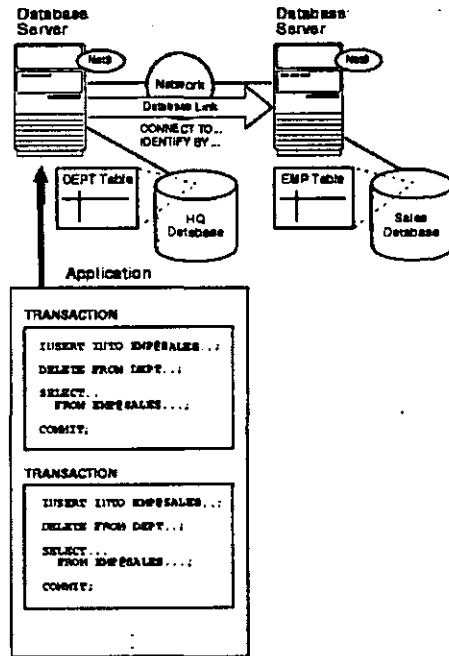


Figura 2.4

Proceso distribuido.

El procesamiento distribuido ocurre cuando una tarea a lo largo de un sistema distribuido es enviada a diferentes computadoras dentro de la red. Por ejemplo en el diseño de un proceso distribuido en una base de datos,

típicamente distribuye parte del proceso en la aplicación del usuario sobre los clientes, los cuales en su mayoría son PC, y por otro lado permite al servidor de base de datos manejar la otra parte del proceso, que estriba en los accesos compartidos a la base de datos en si. Consecuentemente, una aplicación de base de datos distribuida es mas comúnmente referida como "Cliente - Servidor"

2.5 Bases de datos distribuidas y replicación de base de datos

Los términos de sistemas de base de datos distribuidas y replicación de base de datos" están muy relacionados, no obstante son distintos. En una base de datos puramente distribuida (no replicada), el sistema maneja una copia simple de toda la información y objetos que conforman la base de datos. Típicamente, aplicaciones de base de datos distribuidas usan transacciones para tener acceso a los datos tanto de manera local como de forma remota y modificar la base de datos global de forma inmediata.

La replicación es el proceso de copiar y mantener objetos de la base de datos en múltiples bases de datos para simular un sistema distribuido. Mientras la replicación se basa sobre la tecnología de las bases de datos distribuidas, la replicación puede ofrecer beneficios a las aplicaciones que no son posibles cuando se tiene un sistema puramente distribuido.

Mas comúnmente la replicación es utilizada para mejorar el performance y proteger la disponibilidad de las aplicaciones porque alterna las opciones de accesos de datos. Por ejemplo, una aplicación puede normalmente tener acceso

a una base de datos local mientras que un servidor remoto minimiza el tráfico de la red y alcanza un menor performance además la aplicación puede continuar su funcionamiento si el servidor local falla. Pero otros servidores pueden replicar los datos si están accesibles.



3

Replicación de Información

3.1 La Replicación

La Replicación es el proceso de copiado y mantenimiento de los objetos de la base de datos en múltiples bases de datos que conformen un sistema de base de datos distribuido. La Replicación puede mejorar el performance y proteger la disponibilidad de las aplicaciones debido a que existen opciones alternas de acceso a los datos. Por ejemplo, una aplicación podría normalmente tener acceso una base de datos local en lugar de un servidor remoto para minimizar el tráfico en la red y lograr un performance máximo. Mas aun, la

aplicación puede continuar funcionando si el servidor local experimenta alguna falla ya que otros servidores con información replicada permanecen accesibles¹.

Actualmente no existen estándares en el área de replicación. Por lo que, en el medio cada proveedor de software utiliza sus propios términos para estar describir la replicación y sus funciones. Adicionalmente, estos difieren ampliamente en sus arquitectura.

En la investigación realizada se encontró que el término "replicación" se utiliza para identificar el mantenimiento automático de tablas relacionales replicadas en un ambiente distribuido. Esta definición implica la participación de un proceso de *commit*, para dejar todas las tablas sincronizadas de manera consistente. La replicación se clasifica básicamente en base a la cantidad de latencia que existe antes de que se consiga la consistencia entre las réplicas. Dentro de esta clasificación los tipos de replicación son la síncrona y la asíncrona.

3.2 Replicación de datos Síncrona

La replicación síncrona provee una "estrecha consistencia" entre las bases de datos. Esto es que el tiempo de latencia antes de lograr la consistencia es cero. La información en todas las replicas siempre es la misma, sin importar en que replica la actualización es originada. Esto se puede lograr únicamente a través del "commit en dos fases". Véase el apéndice B para mas detalles respecto a este tipo de commit.

La replicación síncrona puede ser provista por el DBMS o por un servidor / monitor de transacciones distribuidas. Este tipo de transacciones

¹ Oracle Concepts cap 31

muestran lo que se denomina las propiedades ACID cuando la operación es Atómica, los datos son Consistentes, la actividad es Independiente y los resultados son Durables. Véase el apéndice A para el detalle de estas propiedades.

3.2.1 Monitores Distribuidores para Procesamiento de Transacciones

Este tipo de herramientas como el CICS de IBM y Tuxedo, han sido usadas por mucho tiempo en *mainframes*. Hoy son parte de ambientes cliente/servidor robustos. Estos Monitores Distribuidores de Procesamiento de Transacciones proporcionan un mucho mayor rango de servicios que los proporcionados por un servicio de replicación.

Estos monitores proveen recuperación a nivel de proceso, el cual incluye la habilidad de balancear la carga de ejecución de transacciones y de iniciar nuevamente transacciones después de haber presentado alguna falla. Estos monitores administran y controlan los recursos para la ejecución de las transacciones ya sea en un solo servidor o en múltiples. Esto incluye la habilidad de cooperar con otros monitores en arreglos federados, lo cual permite un manejo global de transacciones a través de recursos heterogéneos destinados a las transacciones.

Cuando se usa un manejo global de transacciones, la consistencia en la información es mantenida en tiempo real a lo largo de todas las réplicas. Cualquier usuario con los privilegios apropiados puede colocar una transacción de actualización sincronizada y todas las réplicas participantes serán actualizadas.

La replicación síncrona o los monitores de transacciones deben de ser utilizados cuando los requerimientos del negocio demanden esta consistencia de información que únicamente un protocolo de commit global lo puede cumplir. El manejo de transacciones distribuidas solo significa el asegurar en tiempo real la consistencia a lo largo de 'n' servidores de transacciones ya sean homogéneos y/o heterogéneos. Si esta estrecha consistencia de información no es un requerimiento del negocio, entonces una técnica de replicación asíncrona que contemple el nivel requerido de integridad y consistencia secuencial de la transacción deberá ser considerada. La replicación asíncrona reduce el costo, mejora la concurrencia con el manejo de los recursos (los datos son bloqueados menos tiempo), y generalmente acorta el tiempo de ejecución la transacción en la base de datos.

3.3 Replicación de datos Asíncrona

La replicación asíncrona provee un tipo de consistencia mas relajada entre las instancias que almacenan información. Esto significa que el tiempo o latencia es mayor a cero antes que la información sea consistente. En otras palabras, siempre existe un grado de retraso entre la base de datos que originó el commit y cuando los efectos de la transacción están disponibles en cualquier replica. Si los recursos de infraestructura son suficientes, la latencia usualmente es medida en segundos.

La replicación asíncrona puede presentarse de dos maneras generales¹:

- Refresco completo o por incrementos,
- Propagación de eventos. La propagación de eventos implica un procesamiento casi en tiempo real. Esto se puede lograr de dos formas, a través de comunicación de base de datos a base de datos o comunicación de proceso a proceso.

Para un refresco completo trata sobre la programación y ejecución de extracciones de lo que se considera la fuente primaria de la información. Finalmente las réplicas destino son cargadas. Para los refrescos por incrementos, el mismo procesamiento ocurre contemplando únicamente los cambios que ocurrieron desde la última extracción. Para este tipo de refresco es que se tiene que realizar en *batch* se asume que el refresco tiene que ser completo.

3.4 La Replicación Básica de Oracle

Con la replicación básica, las réplicas de datos proveen acceso a la información de la tabla que proviene de un nodo primario o "maestro".

Las aplicaciones pueden consultar la información desde las réplicas locales para evitar el acceso a la red independientemente de la disponibilidad de la red. Sin embargo, las aplicaciones del sistema deben de tener acceso la información en el nodo primario cuando se requiera actualizar, la figura 3.4 ilustra la replicación básica.

¹ Replicación de Información, Burette p.p. 13

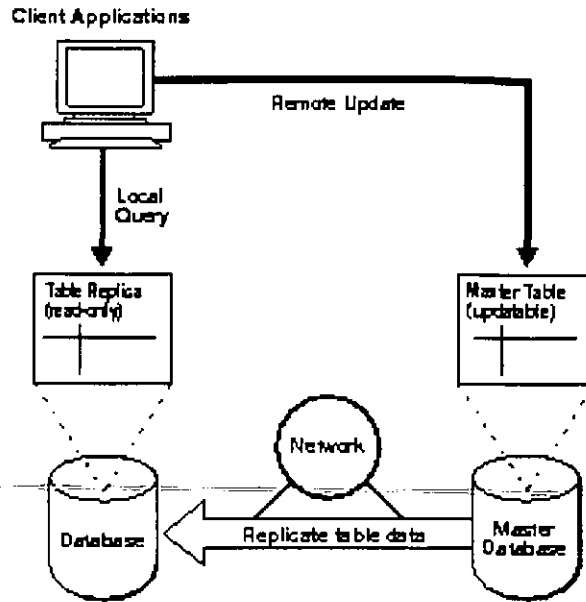


Figura 3.4 Replicación Básica de solo lectura

3.5 Distribución de la Información

La replicación básica es útil para la distribución de la información, Por ejemplo, considérese la operación de una departamental. En este caso, es crítico el asegurar que la información de los precios de los productos este siempre disponible, actualizado y consistente con los de las bodegas. Para lograr estas metas, cada tienda debe de tener su propia copia de la lista de precios de los productos que sea refrescada cada noche a partir de una tabla primaria de precios, la figura 3.5.1 ilustra esta distribución de información.

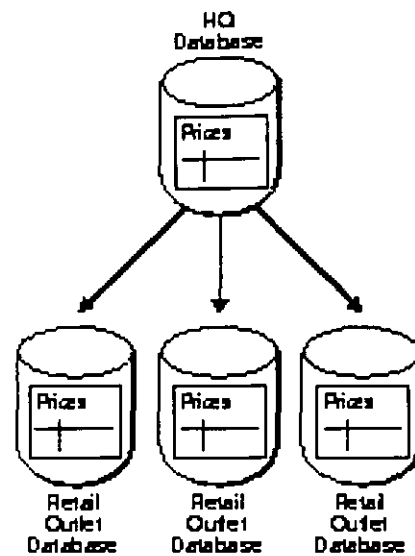


Figura 3.5.1 Carga de Información

La replicación básica es útil a manera de replicar bases de datos enteras o para carga de información en horarios fuera de operación.

Por ejemplo, cuando el performance de los sistemas, que procesan altos volúmenes de transacciones, es crítico, sería ventajoso el mantener una base de datos duplicada para aislar la demanda de las consultas generadas por las aplicaciones de soporte a decisiones, la figura 3.5.2 ilustra la carga de información

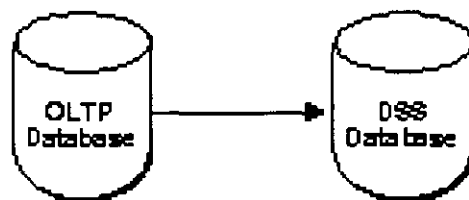


Figura 3.5.2

La replicación básica puede ser útil como un mecanismo de envío de información. Por ejemplo, la replicación puede periódicamente mover datos de una base de datos que procesa las transacciones en producción a un data warehouse (Base de datos para análisis de información).

3.6 Tablas Snapshots de solo lectura

Una tabla snapshot de solo lectura es una copia local de la información originada a partir de una o mas tablas maestras remotas. Una aplicación puede consultar los datos en una tabla snapshot, pero no puede insertar, actualizar o borrar renglones en el snapshot. La figura 3.6.1 ilustra los componentes de la replicación básica, snapshot de solo lectura, tablas maestras y bitácoras de los snapshots.

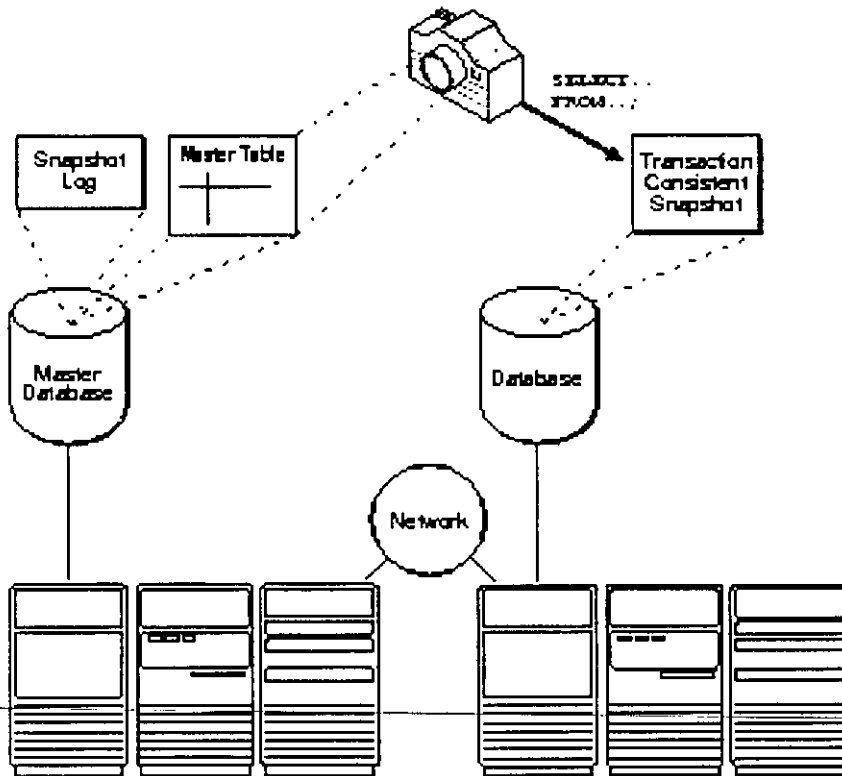


Figura 3.6.1

Arquitectura de Snapshots de solo lectura.

La replicación básica de Oracle se basa en su mecanismo de tablas snapshots. A continuación se explica esta arquitectura.

3.6.1 Query que define el Snapshot

La estructura lógica de información de las tablas snapshots esta definida por un *query* que hace referencia a la información en una o mas tablas maestras remotas. Un *query* que define el snapshot determinara la información que se contendrá en el snapshot.

Un query que define el snapshot será aquel a través del cual se establezca una correspondencia directa con un renglón o parte de un renglón en una sola tabla maestra. Específicamente, el query que define un snapshot no deberá contener las funciones DISTINCT, de agregación, cláusulas GROUP BY o CONNECT BY, *join*, *subqueries* de tipo restrictivo, u operaciones de conjuntos. El siguiente ejemplo muestra la definición de una tabla snapshot simple.

```
CREATE SNAPSHOT ventas.clientes AS
  SELECT * FROM ventas.clientes@matriz.compania.com
```

Obsérvese: en todos los casos, el query que define el snapshot debe de hacer referencia a todas las columnas de la llave primaria en la tabla maestra.

Un query que define un snapshot puede incluir ciertos tipos de subqueries que hagan referencia a múltiples tablas para filtrar los renglones del snapshot de la tabla maestra. Un snapshot basado en un subquery puede ser usado para

crear snapshots que contengan las referencias entre las tablas "hijo" y las tablas "padre", estas referencias podrían involucrar múltiples niveles.

Los siguientes ejemplos crean un snapshot simple basado en un subquery

```
CREATE SNAPSHOT ventas.ordenes AS
SELECT * FROM ventas.ordenes@matriz.compania.com o
WHERE EXISTS
( SELECT c_id FROM ventas.clientes@matriz.compania.com c
WHERE o.c_id = c.c_id AND codigo_postal = 19555);
```

Objetos internos que respaldan los **Snapshots**

Cuando se crea un snapshot nuevo y de solo lectura. Oracle crea varios objetos internos en la base de datos para operar los snapshots.

Oracle siempre crea una tabla base para almacenar los datos del snapshot. El nombre de la tabla base del snapshot es `SNAP$_nombredelsnapshot`.

Oracle crea un índice para la llave primaria en la tabla base del snapshot. El nombre del índice es el nombre del índice usado para forzar la restricción correspondiente de la LLAVE PRIMARIA en el nodo maestro.

Oracle siempre crea una vista para cada snapshot. La vista toma el nombre del snapshot y proporciona acceso de solo lectura al snapshot.

3.7 Refresco de Snapshots

Los datos del snapshot no necesariamente coincidirán con los datos recientes de su tabla maestra. Una tabla - snapshot es un reflejo de la consistencia de transacciones respecto a los datos maestros en un punto específico del tiempo. Para mantener los datos del snapshot relativamente actualizados con respecto a la información del maestro, Oracle periódicamente refresca el snapshot. Un refresco del snapshot es una eficiente operación en *batch* que hace que el snapshot refleje un estado mas similar al estado actual de su maestro.

Uno tiene que decidir como y cuando refrescar cada snapshot para que este mas actualizado. Por ejemplo snapshots que provienen de tablas maestras, usualmente actualizadas por las aplicaciones, requerirán refrescos mas frecuentes. En contraste, snapshots dependientes de tablas maestras relativamente estáticas, usualmente requerirán refrescos menos frecuentes. En suma, se tiene que analizar las características de la aplicación y los requerimientos para determinar los intervalos apropiados para el refresco del snapshot.

Para refrescar los snapshots, Oracle maneja diferentes tipos de refrescos para los grupos de snapshots, "completos" y "rápidos", al igual que refrescos "manuales" y "automáticos".

Oracle puede refrescar un snapshot usando ya sea un refresco completo o un refresco rápido.

3.7.1 Refrescos Completos

Para realizar un refresco completo de un snapshot, el servidor que maneja el snapshot ejecuta el query que lo define. El conjunto resultante del query reemplaza la información existente del snapshot para refrescar el snapshot. Oracle puede realizar un refresco completo para cualquier snapshot.

3.7.2 Refrescos Rápidos

Para realizar un refresco rápido, el servidor que maneja el snapshot primero identifica los cambios ocurridos en el maestro desde el refresco mas reciente del snapshot y posteriormente los aplica al snapshot. Los refresco rápidos son mas eficientes que los refrescos completos cuando existen pocos cambios en el maestro debido a que los nodos participantes y la red replica menos información. Refrescos rápidos para los snapshots solo son factibles cuando la tabla maestra tiene una bitácora del snapshot.

3.7.3 Bitácora de Snapshots

Cuando una tabla maestra corresponde a uno mas snapshots, es conveniente crear una bitácora de snapshot para la tabla de tal forma que los refrescos rápidos de los snapshots sean una opción.

Una bitácora de snapshot de la tabla maestra registra datos de refresco rápido para todos los snapshots, solo es posible una bitácora por tabla maestra. Cuando el servidor realiza un refresco rápido para un snapshot, usa los datos de la bitácora de la tabla maestra correspondiente para refrescar de manera eficiente el snapshot. Oracle automáticamente purga datos especificos de

refresco de la bitácora del snapshot después de que todos los snapshots sean refrescados de tal forma que los datos en la bitácora ya no sean necesarios.

Cuando se crea un snapshot para una tabla maestra, Oracle crea un tabla subyacente para manejar la bitácora del snapshot. La tabla de los registros del snapshot guarda las llaves primarias, el tiempo de registro, y opcionalmente los ROWIDs de los renglones que fueron actualizados en la tabla maestra por las aplicaciones.

Un snapshot puede también tener columnas de filtro para manejar los refrescos rápidos para los snapshots con subqueries. El nombre de una tabla de bitácora para un *snapshot* es `MLOG$_nombre_ de _tabla_maestra`.

3.7.4 Refresco por Grupo de Snapshots

Para conservar la integridad de referencias y la consistencia entre los snapshots de varias tablas relacionadas, Oracle organiza y refresca cada snapshot como parte de un refresco de grupo. Oracle refresca todos los snapshots en un grupo como una sola operación. Después de refrescar todos los snapshots en un refresco de grupo, la información en de los snapshots en el grupo corresponde a la misma transacción - consistente en un punto en el tiempo.

3.7.5 Refresco Automático de Snapshots

Cuando se crea un grupo de refresco de snapshots, los administradores usualmente configuran el grupo de tal forma Oracle automáticamente refresque

sus snapshots. De lo contrario, los administradores tendrían que manualmente refrescar el grupo cuando fuera necesario

Cuando se configura el grupo de refresco para refrescos automáticos, se debe:

Especificar un intervalo de refresco para el grupo.

Configurar el manejo del servidor de snapshots con uno o mas procesos de background SNP que están esperando para ser refrescados.

3.7.6 Intervalos automáticos de refresco.

Cuando se crea un grupo de refresco de snapshot se debe especificar un intervalo de refresco para el grupo. Es decir se debe especificar el intervalo de tiempo en que se realizara el refresco de los snapshots. Cuando se configura un grupo de refresco se deben tomar en cuenta las siguientes características:

La fecha o las fechas especifican los intervalos de refresco y deben ser evaluadas para tiempos posteriores.

El intervalo de refresco debe ser mayor que el tiempo en que se realiza el refresco.

Expresiones de fechas relativas evalúan puntos relativos en el tiempo, para las fechas mas recientes de refrescos. Si la red o el sistema falla interfiere con el grupo de refresco que se programaron, por lo tanto el siguiente refresco puede cambiar.

Expresiones de fechas explícitas evalúan puntos específicos en el tiempo con base en los refrescos mas recientes.

Normalmente Oracle intenta realizar un refresco rápido por cada snapshot en un grupo de refresco. Si no puede realizar el refresco rápido para un snapshot inválido, por ejemplo cuando una tabla maestra no tiene un snapshot log, Oracle realiza un refresco completo.

Procesos de background SNP

Oracle realiza los refrescos automáticos utilizando colas que programan la ejecución de procedimientos internos. Las colas de trabajos requieren al menos un proceso SNP que se este levantando. El proceso SNP revisa periódicamente si existen trabajos encolados para ser ejecutados.

Refrescos manuales

Los refrescos programados no siempre son lo mas adecuado. Por ejemplo después de una carga masiva de información sobre una tabla maestra, los snapshots no representan los datos de la tabla maestra. Y se tendría que esperar al próximo refresco programado. Para que esperar si podemos realizar el refresco de manera manual, para que inmediatamente se propaguen los cambios realizados en los snapshots asociados.

Oracle provee adicionalmente otras opciones dentro de la replicación como son:

- Snapshots complejos
- ROWID snapshots

Snapshots Complejos

Cuando el query definiendo al snapshot contiene alguna sentencia `distinct` o algún tipo de función como `GROUP BY` o sentencias de `CONNECT BY`, uniones, tipos de subqueries restringidos o un conjunto de operaciones el snapshot es definido como snapshot complejo. El siguiente ejemplo es una definición de un snapshot complejo:

```
CREATE SNAPSHOT scott.emp AS
SELECT ename, dname
FROM scott.emp@hq.acme.com a, scott.dept@hq.acme.com b
WHERE a.deptno = b.deptno
SORT BY dname
```

La desventaja primaria de un snapshot complejo es que Oracle no puede realizar un refresco rápido del snapshot; Oracle únicamente refrescos completos de snapshots complejos. Consecuentemente, el uso de snapshots complejos puede afectar al performance de la red durante el refresco del snapshot.

ROWID Snapshots

Los Snapshots de llave primaria, como se comentaron en la sección anterior de este capítulo, son el normal para Oracle. Oracle basa el snapshot sobre la llave primaria de la tabla maestra, porque con esta estructura se puede:

Reorganizar la tabla maestra de un snapshot sin completar un refresco completo del snapshot.

Crear un snapshot con una definición del *query* que incluya tipos de subqueries restringidos.

Conceptos de Replicación Simétrica.

En ambientes de replicación simétrica, los datos pueden ser replicados en cualquier parte de manera bi direccional. Y el acceso a los datos puede ser de lectura y escritura o ambos. La siguiente sección explica el concepto principal de un sistema de replicación simétrica.

Usos de la replicación simétrica.

Configuración de la replicación simétrica.

Replicando Objetos, Grupos, *Sites* y catálogos.

Arquitectura de la replicación simétrica de Oracle.

Administradores de la replicación. Propagadores y receptores.

Conflictos de la repliación.

Usos de la Replicación Simétrica.

La replicación simétrica es utilizada por muchos tipos de aplicaciones con requerimientos especiales.

Ambientes desconectados

La replicación simétrica es utilizada para el desarrollo de procesos de transacciones que operan con componentes desconectados. Por ejemplo, la fuerza típica de la automatización del departamento de ventas para una

compañía de seguros de vida. Cada vendedor debe visitar clientes regularmente con una computadora portátil y registrando órdenes en una base de datos personal mientras está desconectado de la computadora central de la corporación hasta que regrese a la oficina, cada vendedor debe enviar todas las órdenes a una base de datos centralizada de la corporación.

Esquema de fallas

La replicación simétrica puede ser utilizada para proteger la disponibilidad de una base de datos de misión crítica. Por ejemplo, un sistema de replicación simétrica puede replicar una base de datos completa para establecer un sistema de fallas. Si el site primario no se encuentra disponible, debido a fallas del sistema o interrupción de servicios de la red. En contraste con una base de datos en standby el site puede estar disponible mientras el site primario se encuentra abajo para soportar la aplicación.

Distribuyendo cargas en la aplicación

La replicación simétrica es útil para el proceso de transacciones de aplicaciones que requieren múltiples puntos de acceso a la información de la base de datos para propósitos de distribuir una carga pesada de datos en la aplicación, asegurando disponibilidad continua o permitiendo más el acceso de datos.

La replicación simétrica puede ser utilizada como un mecanismo de transporte de la información. Por ejemplo, un sistema de replicación simétrica puede periódicamente descargar datos de una base de datos con muy transaccional a un data *warehouse* o a un data mart.

3.8 La Replicación Simétrica de Oracle

La replicación simétrica de Oracle permite a las aplicaciones el actualizar las replicas de las tablas a través del sistema de bases de datos replicadas. Con la replicación simétrica, las replicas de datos pueden proveer acceso de lectura y de actualización, la figura 3.9 ilustra la replicación simétrica.

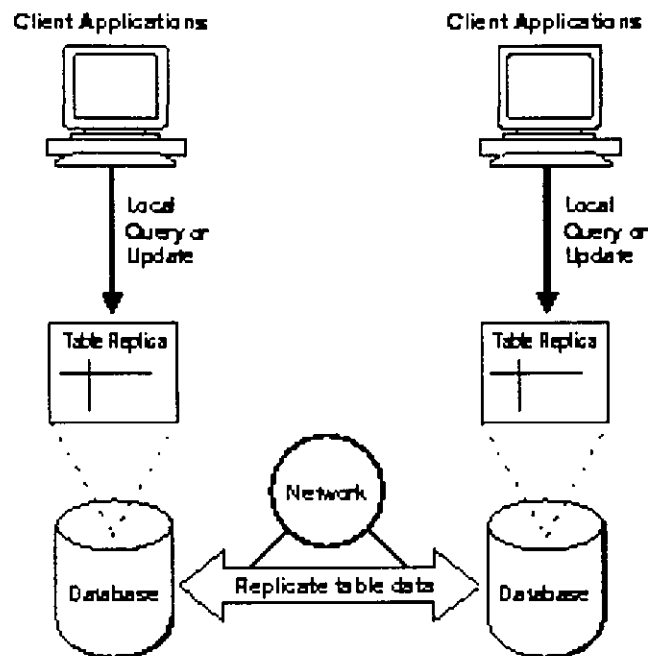


Figura 3.8 Replicación Simétrica

3.9 Esquemas de la replicación simétrica de Oracle

Configuración de la Replicación Simétrica

Oracle soporta los requerimientos de la replicación simétrica utilizando tanto esquemas *Multimasters* (Maestros Múltiples) como *master snapshot*.

Multi-master

La opción de replicación agrega un Método llamado multi-masters, Con la opción de replicación la mayoría de los objetos pueden ser registrados y replicados en *sites* remotos (todos los objetos de los esquemas, excepto secuencias *database links* y *clusters*). Un catálogo Global de objetos replicados es mantenido utilizado aplicaciones de PL/SQL que son ejecutadas desde el "*master definition site*" La replicación es automáticamente generada por cambios de registros en las tablas y llamadas de procedimientos.

Si una tabla es registrada en el catálogo de replicación un número de objetos replicados son generados. Un *trigger* es creado para capturar eventos que ocurrirán nuevamente sobre la tabla. Los *triggers* generados serán *after rows* es decir después de cada registro los cuales harán un llamado a la cola de las tablas retrasadas para realizar la acción correspondiente en el *site* remoto para cada registro individual. La llamada actual es situada dentro de la cola de las tabla atrasadas y será enviada en un paquete al *site* remoto. Los paquetes son automáticamente generados cuando la tabla es registrada dentro de la opción de replicación. Este contiene procedimientos para las transacciones de actualizaciones, inserciones y borrados, así como también contiene información para saber que las transacciones no serán replicadas nuevamente.

Un trabajo en la cola retrasos periódicamente envía transacciones para otro *site* remoto. Estos trabajos pueden ser calendarizados sobre las localizaciones con diferencia de intervalos pero es importante antes de ser enviados y después de ser recibida la información asegurarse de manejar la

resolución de conflictos para evitar cualquier tipo de problemas. Los paquetes soportados llaman rutinas de resolución de conflictos internas que manejan los conflictos como a ellos se les ocurre. El usuario tiene la capacidad de escoger que rutinas le gustaría invocar para conflictos que ocurran sobre una columna por columna fundamento. Algunas rutinas disponibles son: Escoger los valores con la ultima fecha de actualización, agregar valores juntos, escoger valores de la localización con diferentes prioridades, escoger el valor mas alto, escoger el valor mas anterior actualizado, descartar valores entrantes, sobre escribir valores existentes, etc.

Los usuarios pueden también escribir sus propios procedimientos los cuales pueden ser registrados con los métodos de resolución de conflictos por columna en la tabla. Los métodos también pueden ser escritos para proveer notificaciones electrónicas de detección de conflicto y resolución.

En conclusión *Multi-Master* es una configuración de tablas llenas, asomándose entre si, que emiten la replicación entre las bases de datos participantes. Todas las bases de datos son replicadas completamente. Cualquier cambio es emitido hacia todas las bases de datos participantes. Los cambios son colocados en la cola en el site donde se originaron los cambios y posteriormente son empujados hacia los diferentes sites. Múltiples colas de Múltiples *sites* pueden ser empujados al mismo tiempo. Las fallas durante el envío de un *site* no afectará a ningún otro site, y las transacciones son dejadas en la cola para ser propagadas después de la corrección de los errores. Los conflictos son resueltos en cada site de la base de datos, con cada uno utilizando la misma rutina para garantizar convergencias.

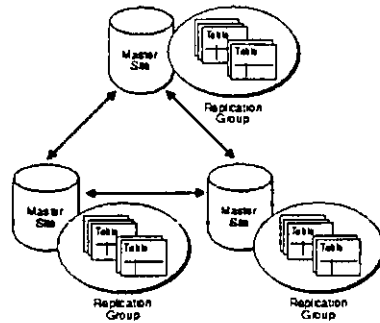


Figura 3.9.1 Sistema de Replicación Avanzado *Multimaster*

Snapshot sites y snapshots actualizable.

Los *Master sites* en un esquema de replicación simétrica pueden consolidar la información que las aplicaciones actualizan en el *snapshot site* remoto. La facilidad de la replicación simétrica de Oracle permite a las aplicaciones insertar, actualizar y borrar registros de tablas a través de snapshots actualizables, ver figura 3.10.2

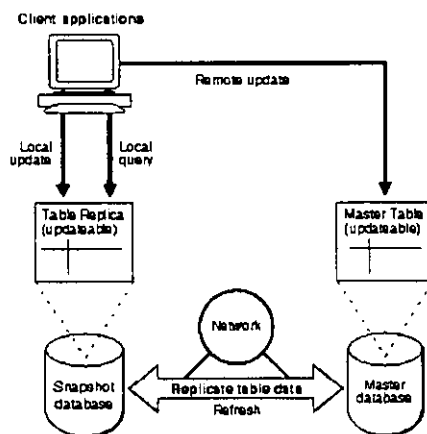


Figura 3.9.2 Sistema de Replicación Avanzado con Snapshots Actualizables

Los snapshots actualizables tienen las siguientes propiedades

Los snapshots actualizables son siempre simples, refrescos rápidos en las tablas de los snapshots.

Oracle propaga los cambios hechos a través de un snapshot actualizable hacia la tabla maestra del snapshot remoto. Si es necesario, la actualización, entonces se realiza en cascada hacia los otros *master sites*.

Oracle refresca un snapshot actualizable como parte de un grupo de refresco idéntico al de los snapshots de solo lectura.

Los snapshots actualizables tienen los mismos objetos implícitos, como tablas base, índices y vistas como los snapshots de solo lectura.

Adicionalmente, Oracle crea la tabla para soportar los snapshots actualizables.

USLOG\$_nombre_del_snapshot.

Configuraciones Híbridas

La replicación *multi-master* y los snapshots actualizables pueden ser combinados en configuraciones híbridas para encontrarse con diferentes requerimientos de las aplicaciones. Las configuraciones híbridas pueden tener cualquier número de *master sites* y múltiples snapshots para cada *master site*.

Por ejemplo, como se muestra en la siguiente figura 3.10.3 la replicación de *n* - caminos entre 2 *masters* puede soportar tablas completas replicadas entre las bases de datos que soportan dos regiones geográficas. Los snapshots pueden ser definidos sobre el *master* para replicar las tablas completas o sub conjuntos de tablas en cada región.

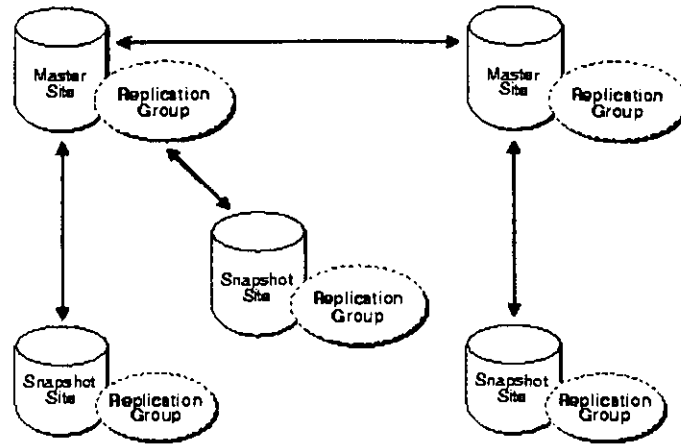


Figure 3.9.3 Configuración Híbrida

Algunas de las principales diferencias entre los snapshots actualizables y los *masters* replicados son las siguientes:

Los *masters* replicados deben contener datos para las tablas completas a ser replicadas, mientras que los snapshots pueden replicar sub conjuntos de datos de las tablas maestras.

La replicación *multi-master* permite replicar cambios para cada transacción como los cambios ocurren. Los refrescos de los snapshots son conjuntos orientados, propagando cambios para múltiples transacciones en una mas eficiente operación en *batch*, pero en intervalos. Si ocurren conflictos por cambios hechos de múltiples copias del mismo dato los *master sites* detectan y resuelven el conflicto.

3.10 La replicación Simétrica y el manejo de la replicación.

Replicación de objetos, grupos, sites y catálogos

La siguiente sección explica los componentes básicos de un sistema de replicación simétrica, incluyendo replicación de objetos, grupos, sites y catálogos.

Replicación de objetos

La replicación de objetos se basa en un objeto existente sobre múltiples servidores en un sistema de base de datos distribuida. La replicación simétrica de Oracle permite replicar tablas y soportar objetos como vistas, *triggers*, paquetes, índices y sinónimos.

Replicación de Grupos

En un ambiente de replicación simétrica, Oracle maneja la replicación de objetos utilizando grupos replicados. Para organizar objetos de bases de datos asociadas con un grupo de replicación, de esta manera se facilita la administración de muchos objetos juntos. Típicamente, se crea y utiliza un grupo de replicación para organizar los objetos necesarios dentro de un esquema y soportar una aplicación en particular. Pero no es necesario que los grupos y los esquemas deban de corresponder con uno u otros grupos o esquemas. Los objetos en un grupo de replicación pueden ser originarios de varios esquemas de bases de datos y el esquema puede contener objetos que son miembros de diferentes grupos de replicación. La restricción es que un objeto puede ser miembro de un solo grupo.

Replicación de sites

Un grupo de replicación puede existir en Múltiples *sites* replicados. Un ambiente de replicación simétrica soporta dos tipos básicos de *sites*. *Master sites* y *snapshots sites*.

Un *Master site* mantiene una copia completa de todos los objetos en un grupo de replicación . Todos los *master sites* en un ambiente de replicación *multi-master* se comunican directamente con otro para propagar los datos y los esquemas cambian en el grupo de replicación.

Un grupo de replicación en un *master site* es mas específicamente referido como un grupo maestro. Adicionalmente cada grupo de replicación tiene un y solo un *master definition site*. El *master definition site* es el lugar en donde se realizan todas las tareas de administración

Un *snapshot site* soporta *snapshots* simples (de solo lectura) y *snapshots* actualizables de datos de tablas asociados a *sites* maestros.

Una tabla del *snapshot* en el *snaphot site* puede contener todos o un sub conjunto de datos de una tabla del grupo de replicación. Por otra parte deben ser *snapshots* simples. Con correspondencias de uno a uno en tablas del *site* maestro.

Por ejemplo, un *snapshot site* puede contener *snapshots* para solo tablas seleccionadas en un grupo de replicación. Y un *snapshot* debe ser solo una porción seleccionada de una cierta tabla replicada. Un grupo de replicación en el *snapshot site* es mas comúnmente referido como *snapshot group* (grupo del

snapshot). Un *snapshot group* puede también contener otros objetos de replicación.

Catálogo de Replicación

Cada *master* y *snapshot site* en un ambiente de replicación simétrica tiene un catálogo de replicación. El catálogo del *site* de replicación es un conjunto de tablas y vistas que mantienen información administrativa acerca de objetos de la replicación y grupos de replicación en el *site*,

Cada servidor participante en el ambiente de replicación simétrica puede automatizar la replicación utilizando la información que se encuentra en el catálogo.

Manejo de la replicación API y las peticiones de administración

Para configurar y manejar un ambiente de replicación simétrica, cada servidor participante utiliza una aplicación de interfaz programada de Oracle (API por sus siglas en inglés *application programming interface*). El manejo de la replicación del servidor API es un conjunto de paquetes PL/SQL que encapsula procedimientos y funciones de administración pueden utilizarse para configurar los futuros de la replicación simétrica.

Una petición de administración es una llamada a un procedimiento o función en el manejo de la replicación de Oracle API por ejemplo, cuando se usa la herramienta gráfica (*Replication Manager*) para crear un nuevo grupo maestro, el *Replication Manager* completa las tareas haciendo una llamada a el procedimiento `DBMS_REPCAT.CREATE_MASTER_REPGROUP`.

Arquitectura de la replicación simétrica de Oracle

Oracle converge los datos de configuraciones típicas de replicación simétrica utilizando replicación de registros a bajo nivel con propagación de datos asíncrona. La siguiente sección explica como funciona este mecanismo.

Replicación a nivel de registros

El procesamiento de transacciones típicas de una aplicación modifica pequeños números de registros por transacción. Algunas aplicaciones que trabajan en un esquema de replicación simétrica usualmente dependerán de mecanismos de replicación a nivel registro. Con la replicación a nivel registro, las aplicaciones utilizan sentencias standard DML para modificar los datos de las réplicas locales. Cuando una transacción cambia los datos locales, el servidor automáticamente captura información acerca de las modificaciones y encola las transacciones diferidas para reenviar los cambios a los sites remotos.

Generación de objetos de replicación

Para soportar las transacciones de la replicación en un ambiente de replicación simétrica se deben generar uno o mas objetos internos en el sistema para soportar cada tabla replicada, paquete o procedimiento.

Cuando se replica una tabla, se pueden generar 2 paquetes correspondientes.

Oracle utiliza paquetes de tablas replicadas llamadas `tabla$RP` para replicar las transacciones envueltas en una tabla. Oracle utiliza paquetes de tablas replicadas `tabla$RR` para la resolución de conflictos envueltas en tablas.

En el replicado procedural se replica la especificación de un paquete la cual consta del encabezado y del cuerpo del paquete. Para soportar la replicación procedural, se fusionan paquete y cuerpo. Por default Oracle nombra la envoltura del paquete y el cuerpo utilizando el nombre del objeto con el prefijo "defer_".

Replicación de Datos Asíncrona (almacenamiento y envío)

La configuración típica de la replicación simétrica a nivel de registro propaga los cambios de los datos utilizando replicación asíncrona. La replicación asíncrona ocurre cuando una aplicación actualiza los datos de una tabla local replicada, utilizando del siguiente procedimiento:

- T0 Almacenamiento de la información a replicar en la cola local.
- T1 Envío de la información de la cola.
- T2 Verificación de la información.

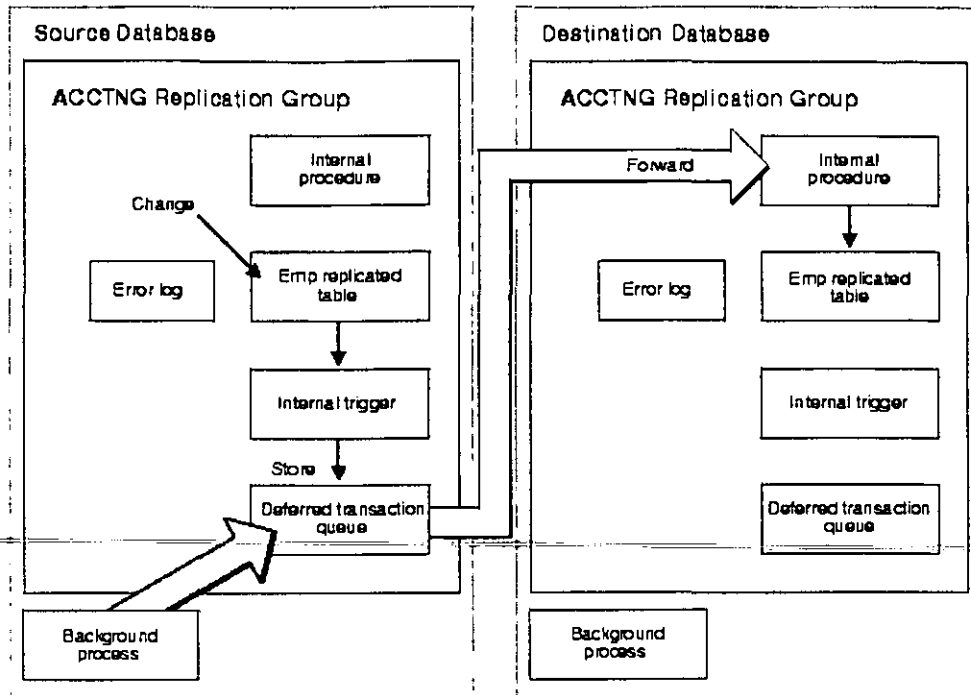


Figura 3.10.1 Mecanismo de Replicación de datos Asincrónica

Oracle utiliza sistemas internos de *triggers*, transacciones diferidas, colas de transacciones diferidas y colas de trabajos para propagar cambios a nivel de datos asincrónicamente entre *master sites* y un sistema de replicación simétrica.

Cuando una aplicación trabaja en un ambiente de replicación simétrica, Oracle utiliza *triggers* internos para capturar y almacenar la información acerca de las actualizaciones para los datos replicados. Los *triggers* construyen llamadas a procedimientos (RPCs) para reproducir los cambios de los datos hechos en los *sites* locales para replicar en los *sites* remotos.

3.11 Resolución de Conflictos

En ambientes de replicación Asíncronos y snapshots actualizables se debe tener en cuenta la posibilidad de conflictos dentro de la replicación, estos

quizás puedan ocurrir cuando dos transacciones que son realizadas en diferentes *sites* actualizan el mismo registro casi al mismo tiempo. Cuando hay conflicto entre los datos, necesitamos implementar un mecanismo para asegurar que los conflictos sean resueltos de acuerdo con las reglas del negocio y asegurar que los datos convergen correctamente en todos los *sites*.

4

Caso de Estudio

4.1 Descripción general de Sistema

Voyager 2000 es el nombre del sistema usado como caso de estudio, este es un sistema que trabaja en modo cliente/servidor y está compuesto por los siguientes módulos:

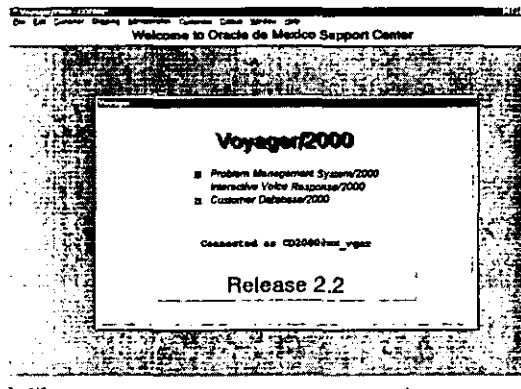


Figura 4.1.1

- Manejo de Problemas (Problem Management System *PMS*), este módulo maneja la información en solicitudes de asistencia técnica (Technical Assistance Request *TAR*) y también los *TARs* para el envío de producto (shipping tar).

 - Módulo de Respuesta Interactiva de Voz (Interactive Voice Response), este módulo maneja el procesamiento de la llamada, a partir del primer contacto del cliente hasta la transferencia de la llamada a un analista. (Este módulo actualmente no se encuentra en uso)
-
- Información del Cliente (Customer Data CD), el cual maneja los datos relacionados con el mantenimiento de contratos, licencias y renovaciones. Permite al personal de soporte el insertar, modificar y revisar varios tipos de registros de los clientes.

4.2 Módulo de Información del Cliente

El módulo esta compuesto de las siguientes aplicaciones:

- Perfil del Cliente
- Licencias del Cliente
- Proceso de Renovación
- Generación de Reportes

Ver figura 4.2.1

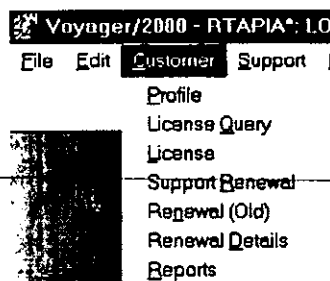


Figura 4.2.1

Perfil del Cliente (Customer Profile)

Esta aplicación permite la captura de los siguientes datos generales de la Compañía, como son los contactos, dirección, nombre, tipo de soporte, etc, ver figura 4.2.2.

The screenshot displays a software window titled "Company Profile" with a standard Windows-style title bar. The window is divided into two main sections: "Company" and "Contact".

Company Section:

- Fields: Name (NLS), Parent, Type (dropdown), Phone, Fax, Industry, AR ID.
- Options: Maintenance (Maint. Type dropdown), Reference (dropdown), Review Period (dropdown), Alert (checkbox), Global Support (checkbox).

Contact Section:

- Fields: Name, Title, Comm. Prefs. (dropdown), Phone, Fax, E Mail, Comments, Address, Renewal.
- Options: Salutation (dropdown), Deleted? (dropdown), Contact ID, Replacement ID.
- Radio buttons for Contact Type: Prime, Ship, Bill, Magazine, News, Announcement, Technical.

Figura 4.2.2

Licencias del Cliente

Esta aplicación permite el registro de los contratos y licencias del cliente, ver figura 4.2.3

Figura 4.2.3

Proceso de Renovación

Este proceso esta compuesto de la Pre - cotización, Cotización, Progreso del Trabajo e Historia de la cotización, ver figura 4.2.4

MID	Machine	Maintenance Type	Start Date	Value	Quoted Period	Closed Value	Closed Period	Payment Term Period	Remove From Quote

Figura 4.2.4

Generación de Reportes

Los reportes existentes son Compañía por Sistema Operativo, Detalles de la Licencia a partir de un compañías o varias, ver figura 4.2.5

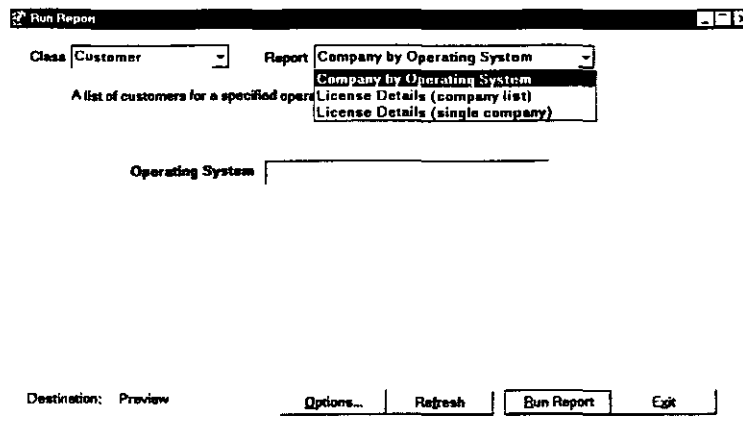


Figura 4.2.5

4.3 Módulo de Manejo de Problemas

Este módulo maneja la información relacionada con la TAR, permitiendo crear, modificar y revisar estas, este módulo esta compuesto por las siguientes aplicaciones, ver figura 4.3.1

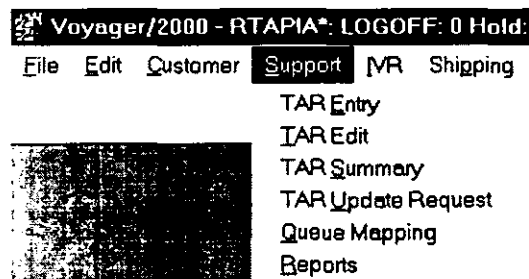


Figura 4.3.1

Petición de actualización de Tar

Voyager permite notificar vía correo electrónico cualquier cambio realizado a un TAR en particular o TARs de algún CSI o nombre de Compañía, ver figura 4.3.4

Figura 4.3.4

Reporte de Tars

En la figura 4.3.5 se muestra la pantalla desde donde se mandan ejecutar los reportes.

Figura 4.3.5

4.4 Solicitud de Actualización de Productos

Esta actividad está compuesta de varias tareas, tales como observar el progreso de las peticiones de actualización de software, generar el reporte de las peticiones de actualización, ver figura 4.4.1

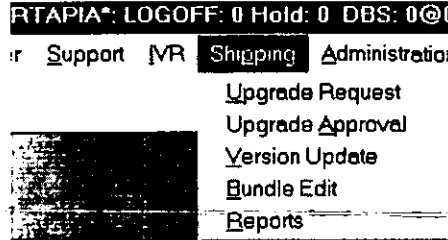


Figura 4.4.1

Cuando se genera una solicitud, el módulo PMS crea una Tar de envío (*Shipping TAR*). Cualquier actualización o cambio realizado a la solicitud es automáticamente reflejado a la *Shipping Tar* asociada, ver figura 4.4.2

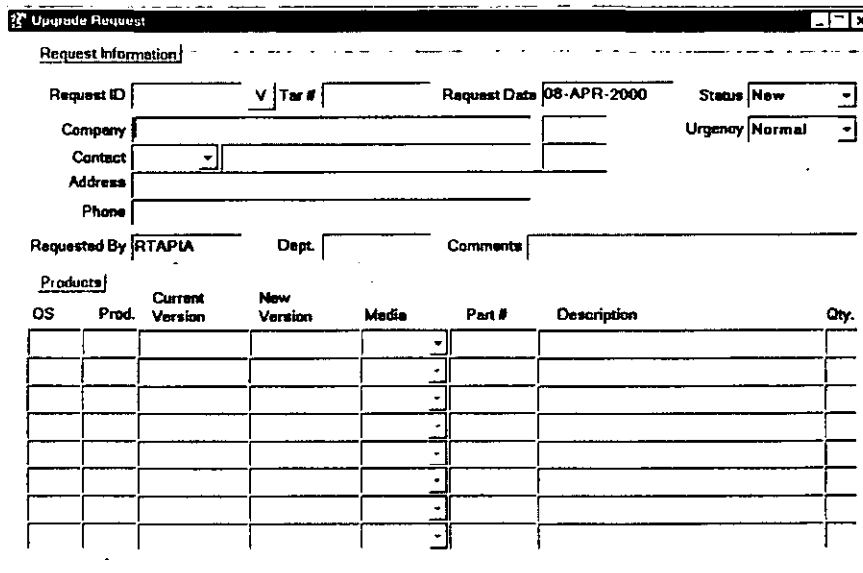


Figura 4.4.2

Revisión del Estado de una Solicitud Existente

Una vez creada la solicitud de actualización, se puede revisar a través de la siguiente pantalla, ver figura 4.4.3

The screenshot shows a window titled "Request Status Details" with the following sections:

- HOD Approval:** Includes fields for "Approved/Rejected By" and "Date", a "Close" button, and a "Comments" text area.
- Shipping:** Includes fields for "Shipped By", "Date", "PO #", and "DO #", and a "Comments" text area.
- Cancellation:** Includes fields for "Cancelled By" and "Date", and a "Comments" text area.

Figura 4.4.3

Aprobación de Solicitudes de Actualización.

La aplicación de aprobación de actualizaciones permite a aquellos usuarios autorizados a rechazar o aprobar las solicitudes de actualización, ya sea en grupo o de manera individual, ver figura 4.4.4

The screenshot shows a window titled "Update Approval" with the following elements:

- Query:** Includes a "Status" dropdown menu, a "Find" button, a "P Auto Query" button, and an "Approve All" button.
- Table:** A table with columns: "TAR #", "Company", "Comment", "Request Date", "Urgency", and "Status".
- Approval Controls:** Fields for "Requestor" and "HOD" with associated "Comments" text areas.
- Product Table:** A table with columns: "Product", "Curr. Version", "New Version", "Modifs", and "Qty".

Figura 4.4.4

Actualización de la Versión del Cliente.

El módulo CD2000 permite llevar el registro de que actualizaciones se le han enviado a que clientes. El proceso consiste en capturar la información del envío y actualizar la licencia del cliente con la nueva versión enviada, ver figura 4.4.5

The screenshot shows a software window titled "Product Version Details". It is divided into several sections:

- Request Information:** Contains fields for "TAR #", "Company", and "Status" (a dropdown menu).
- Shipping Information:** Contains fields for "DO #", "Delivery Date" (displaying "08-APR-2000"), and "Delivered By" (displaying "USER").
- Comments:** A text area for notes.
- Request Details:** A table with the following columns: "OS", "Product Requested", "Ver. Requested", "Bundle Shipped", and "Selective Update". The table contains several rows of data, though the text is small.

Figura 4.4.5

4.5 Operación de Voyager en el área

En base a la descripción realizada en los puntos anteriores, la utilización del sistema esta distribuida entre el Centro de Soporte en Orlando como en cada una de las diferentes subsidiarias. Las actividades se encuentran divididas de la siguiente forma:

En la subsidiaria se lleva a cabo:

- La información concerniente al cliente, licencias, etc, a través del módulo de Información del Cliente
- Cotizaciones y renovaciones de contratos.
- El seguimiento y dependiendo del caso, la autorización respectiva de la solicitud de actualización de productos.

Mientras por otro lado en el Centro de Soporte se lleva a cabo:

- La creación de solicitudes de actualización de productos
- La consulta de la información concerniente a los contactos, licencias, etc del cliente.

Lo anterior se ejemplifica a través de figura 4.5.1

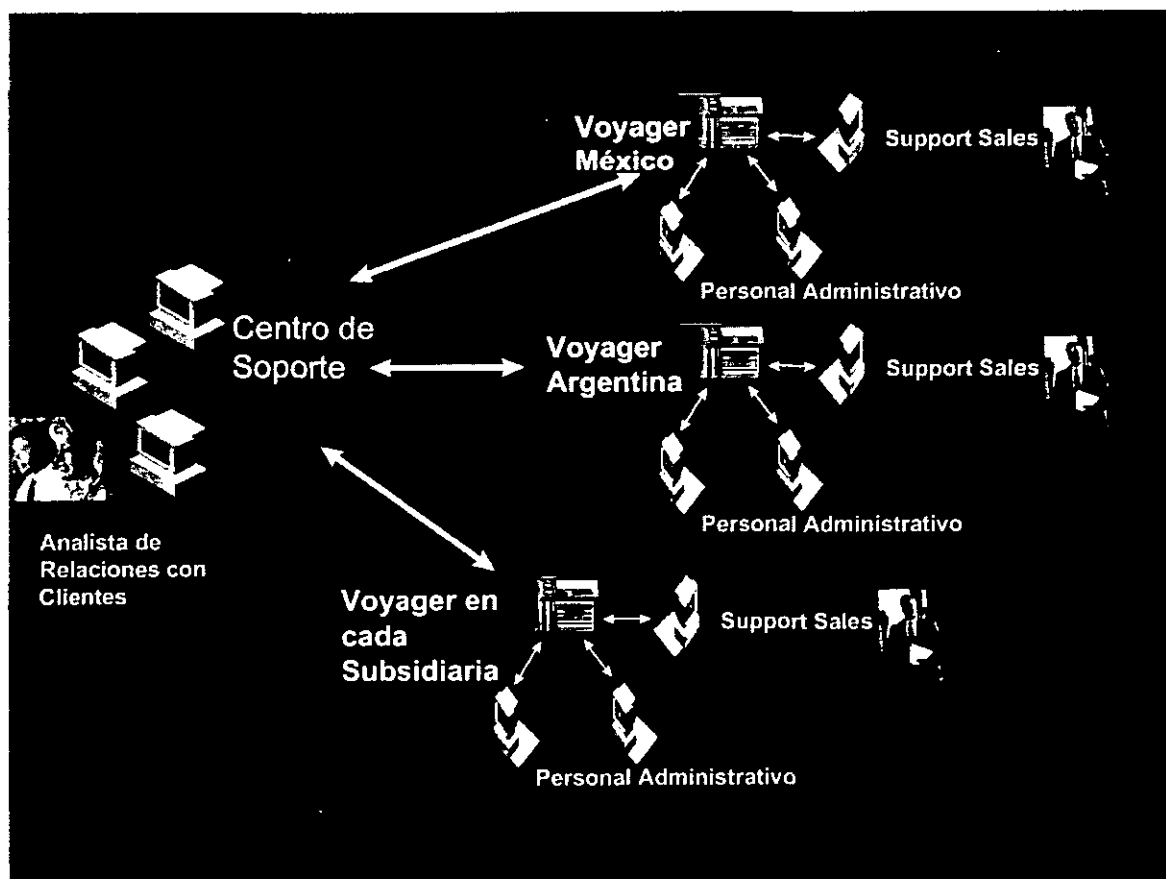


Figura 4.5.1

Debido a que cada una de las operaciones realizadas en el Centro de Soporte requiere de una conexión cliente / servidor a la subsidiaria, obsérvese figura 4.5.1 y 4.5.2, la cantidad de tiempo invertida en espera de respuesta del sistema resulta ser muy alta. Debido a esto se decidió tomar como caso de estudio esta situación y observar los resultados que se obtendrán una vez implementada la replicación simétrica de las tablas que soportan el sistema Voyager en particular.

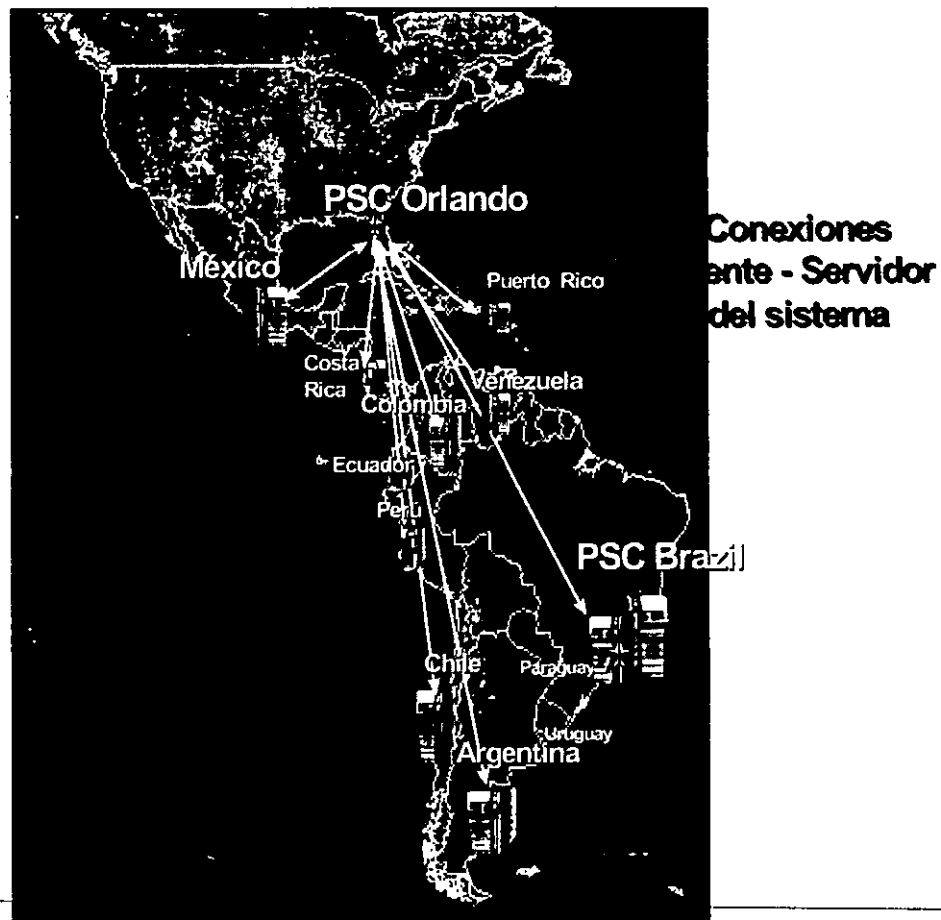


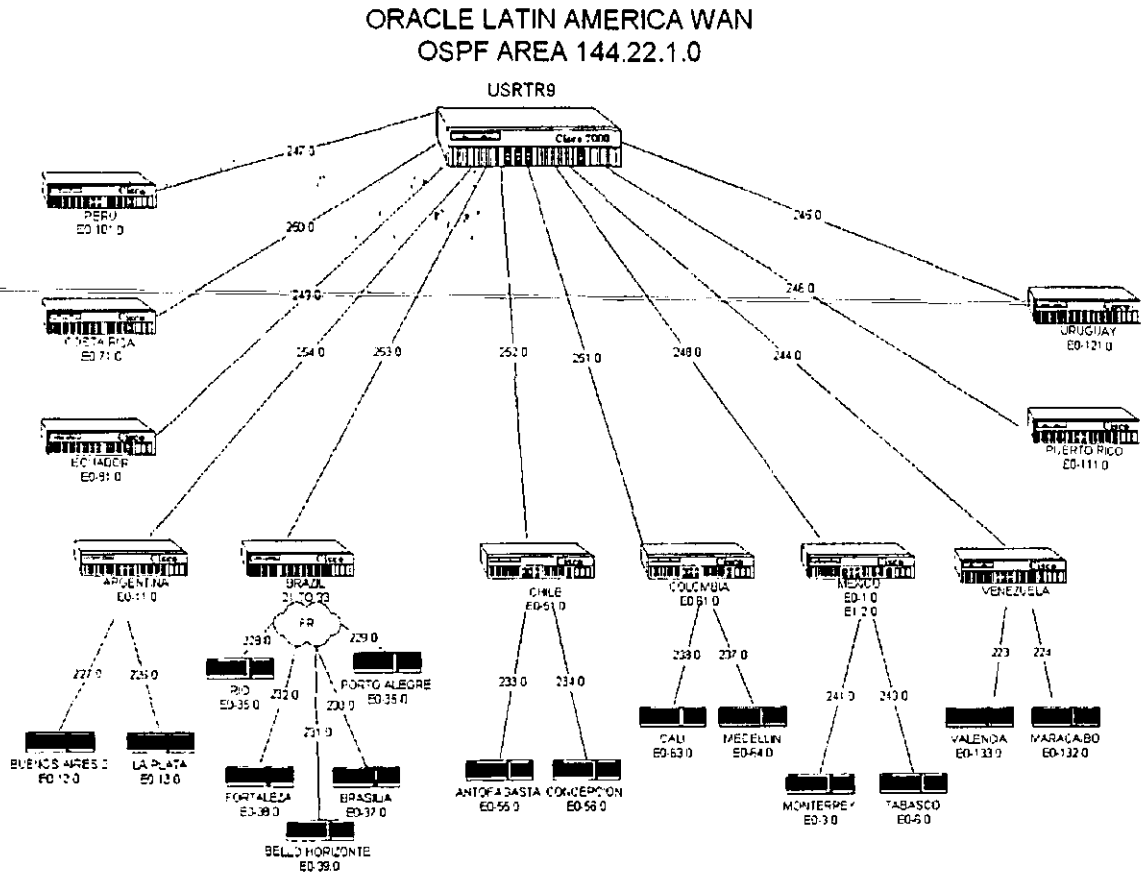
Figura 4.5.2

4.6 Tiempos de Respuesta

Tomando como base 10 consultas de los datos de un cliente en particular en lapsos de 1 hr a partir de las 9:00 hrs se obtuvieron los siguientes tiempos de respuesta.

Pais	Tiempo promedio por operación (Seg.)
Argentina	15
Brasil	10
Costa Rica	20
Colombia	17
Perú	35
Puerto Rico	20
Chile	17
México	7
Venezuela	20
Ecuador	45

4.7 Topología de Red



Pais	Ancho de banda (Bits)
Argentina	1024k
Brasil	1024k
Costa Rica	256k
Colombia	512k

Perú	256k
Puerto Rico	256k
Chile	512k
México	1536k
Venezuela	384k
Ecuador	128k

**ESTA TESIS NO SALE
DE LA BIBLIOTECA**

5

Implementando la Replicación Simétrica

5.1 Análisis del Modelo Entidad Relación

Como se mencionó en el capítulo anterior el sistema Voyager cuenta con los varios módulos y aplicaciones, Voyager como cualquier otro sistema basado en un esquema de base de datos relaciones, tiene que sustentarse en su

correspondiente modelo entidad relación. Este modelo contiene y refleja las definiciones del negocio.

A través del modelo entidad relación y de las aplicaciones desarrolladas alrededor del mismo se deberán de identificar las tablas que se considerará de solo lectura de actualización desde un punto de vista de operación local, una vez con esta clasificación se procederá a implementar la replicación de forma simétrica o de solo lectura.

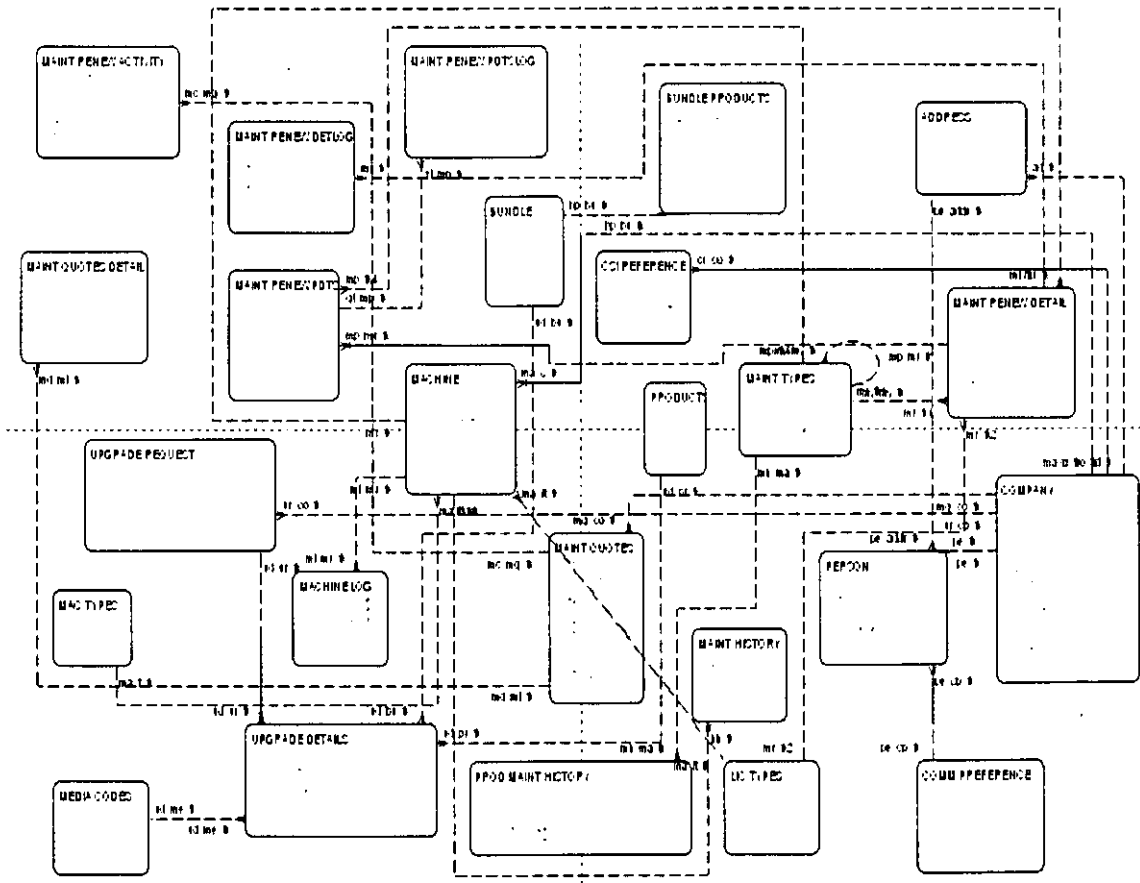


Figura 5-1 Modelo Entidad Relación

5.2 Identificación de las tablas de solo lectura y actualización

Aplicación	Tablas	Modo de Operación Lectura - Actualización
Perfil del Cliente	Company	Lectura
	Person	Lectura
	Address	Lectura
Licencias del Cliente	Machine	Lectura
	Products	Lectura
Proceso de Renovación	Prod_maint_history	Lectura
	Maint_history	Lectura
	Maint_renew_pdtlogs	Lectura
	Maint_renew_activity	Lectura
	Maint_renew_detail	Lectura
	Machine	Lectura
Solicitud de actualización	Upgrade_request	Actualización
	Upgrade_details	Actualización
Aprobación de actualización	Upgrade_request	Actualización
Edición de paquetes	Bundle	Lectura

5.3 Implementando la Replicación de solo lectura

A continuación se muestra la descripción de los pasos a seguir para colocar todas las tablas en ambas base de datos a replicar en modo de solo lectura.

En el snapshot site el acceso es local.

El snapshot es actualizado periódicamente mediante la operación de refresh, ya sea de forma automática o manual.

Para caso practico se trabajara únicamente sobre la tabla de "DEPT" mostrando la configuración para esta tabla, y asumiendo que la configuración es la misma para todas las tablas.

```
SQL> select * from dept;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

```
SQL> create snapshot log on dept;  
Snapshot log created.
```

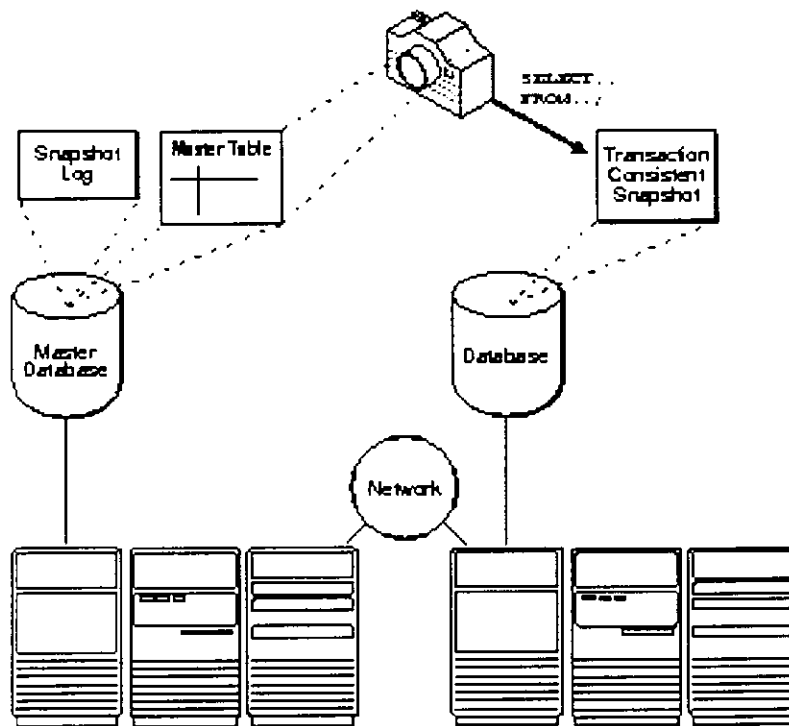
```
SQL> create snapshot dept  
refresh fast  
next sysdate + 1 / 1440  
start with sysdate  
with primary key  
as select *  
from dept@V81.WORLD;
```

```
SQL> select * from dept;
DEPTNO DNAME      LOC
-----
10 ACCOUNTING    NEW YORK
20 RESEARCH      DALLAS
30 SALES          CHICAGO
40 OPERATIONS    BOSTON
```

```
SQL> select job, what from user_jobs;
```

```
JOB WHAT
-----
```

```
445 dbms_refresh.refresh('SCOTT"."DEPT");
```



Snapshots de solo lectura

Ejemplo Configuración - Componentes

Ejemplo de Configuración Master

Primero nos conectamos al sistema con el usuario dueño de el objeto a replicar

```
SQL> conn scott/tiger@v81:world
```

Connected.

Creamos el Snapshotlog de la tabla

```
SQL> create snapshot log on dept  
2 with primary key;
```

Snapshot log created.

```
SQL> select master, log_table  
2 from user_snapshot_logs  
3 where master = 'DEPT';
```

MASTER	LOG_TABLE
DEPT	MLOG\$_DEPT

Nos conectamos con el usuario sys para revisar la configuración

```
SQL> conn sys/manager@v81.world
```

```
Connected.
```

```
SQL> select name, trigflag
```

```
2 from tab$ t, obj$ o
```

```
3 where t.obj# = o.obj#
```

```
4 and o.name = 'DEPT';
```

```
NAME                TRIGFLAG
```

```
-----  
DEPT                3
```

```
SQL> desc mlog$_dept
```

```
Name                Null?    Type  
-----  
DEPTNO              NUMBER(2)
```

```
(Primary Key de la tabla)
```

```
SNAPTIME$$          DATE
```

```
(Refresh Time)
```

```
DMLTYPE$$           VARCHAR2(1)
```

```
(Tipo de DML (D,U,I))
```

```
OLD_NEW$$           VARCHAR2(1)
```

```
(Información relativa a cuando se modifica PK (O,N,U))
```

```
CHANGE_VECTOR$$     RAW(255)
```

```
(Información de la columna que cambia. Usado también para  
subqueries y LOB)
```

Ejemplo Configuración Snapshot site

Nos conectamos con el usuario Dueño de la tabla

```
SQL> conn scott/tiger@v82.world
Connected.
```

Creamos la liga hacia el servidor (Master)

```
SQL> create database link v81.world
2 connect to scott identified by tiger;
```

Database link created.

Creamos el Snapshot indicando el intervalo de refresco y la tabla sobre la cual se creara.

```
SQL> create snapshot dept
2 refresh fast
3 next sysdate + 1 / 1440
4 start with sysdate
5 with primary key
6 as select *
7 from dept@V81.WORLD;
```

Snapshot created.

Revisamos la configuración

```
SQL> select name, table_name, updatable
2 from user_snapshots;
```

NAME	TABLE_NAME	UPD
DEPT	SNAP\$_DEPT	NO

```
SQL> select job, what, interval, next_date, next_sec
2 from user_jobs;
```

JOB WHAT	INTERVAL	NEXT_DATE	NEXT_SEC

448 dbms_refresh.refresh("SCOTT"."DEPT");	sysdate + 1 / 1440	22-JUL-99	17:50:54

5.4 Identificación de las tablas del Esquema Simétrico

Las siguientes tablas son las tablas que forman parte de las operaciones que son comunes entre el centro y la(s) subsidiaria (actualización)

SYSTEM_PRIVILEGE_MAP
TABLE_PRIVILEGE_MAP
STMT_AUDIT_OPTION_MAP
USER_PROFILE
AUDIT_ACTIONS
PSTUBTBL
DBMS_ALERT_INFO
DBJ_REP_OUTPUT
REPORT_TYPES
REPORT_CLASS_ROLES
REPORT_CLASSES
REPORT_PARAMS
LOV_TYPES
FILTERS
FILTER_CONDITIONS
TEMPLATES
ACTION
CALLS
STAFF
ACTION_TYPE_CODES

BUG_HEADER
BUG_TEXT
BUG_TRACKERS
CONTACT_STATUS
FAX_QUEUES
JOB_ITEMS
JOB_ITEM_ARGS
JOB_LOGS
JOB_PRIVS
JOB_STATUS_CODES
JOB_TYPES
JOB_DETAILS
SEVERITY_TYPES
SEVERITY_TEXT
RTSS_COUNTRY
SUPPORT_GROUP
TAR_UPDATE_REQUESTS
WORK_HOURS
IVR_TRANSACTION_LOGS
IVR_CODES
IVR_QUEUES
IVR_QUEUES_HISTORY
IVR_AVAILABLE_ANALYST
IVR_GLOBAL_SUPPORT
APPLICATION_PREFERENCES
SC_CSI_REFERENCE
APPLICATION_PRIVS
ERROR_CODES
CD_BUG_PRODUCTS

BUG_COMPONENTS
BUG_PRODUCTS
BUG_PRODUCT_LINES
ITS_STATUSES
ITS_ACTION_TYPES
ITS_QUEUES
ITS_REF_VALUES
ITS_COUNTRIES
CUSTOMIZE_MENU
JOB_DIST_LISTS
SORT_LISTS
ITS_XFER
ANALYST_STATUS_LOGS
QED_DETAIL
EMPLOYEE
INDUSTRY
MAINT_TYPES
COMPANY
ADDRESS
COMM_PREFERENCE
PERSON
MAC_TYPES
MEDIA_CODES
LIC_TYPES
PRODUCT_FAMILY
MACHINE
MACHINE_LOG
CSI_REFERENCE
PRODUCTS_GROUP
PRODUCTS

BUNDLE
BUNDLE_PRODUCTS
PROD_LIC
PRODUCT_LOG
MAINT_HISTORY
PROD_MAINT_HISTORY
VER_HISTORY
RENEWAL_STATUS
MAINT_RENEW_DETAIL
MAINT_QUOTES
MAINT_QUOTES_DETAIL
MAINT_RENEW_ACTIVITY
MAINT_RENEW_DETLOG
MAINT_RENEW_PDTS
MAINT_RENEW_PDTSLOG
DELETE_LOG
CD2000_SEQ
UPGRADE_REQUEST
UPGRADE_DETAILS
UPGRADE_STATUS
BILL_SOURCE
REV_ALLOC
PHONE
SITE_USE
DEFERRED_UPDATE
CORE_APPS_PERSON
CORE_APPS_PHONE
CD_GCDB_XFER
CD_GCDB_MAP

IVR_MENU
IVR_MENU_LINES
IVR_MENU_VERSIONS
IVR_VOICE_FILES
IVR_LOOKUP_TYPES
IVR_LOOKUPS
IVR_SYSTEM_DOWN
RA_CONTACTS_LOCAL
RA_PHONES_LOCAL
ROUTE_PROD_PLAT_ALGORITHM
ROUTE_GROUP_PROD_PLAT
ITS_MESSAGES

5.5 Implementación de la Replicación Simétrica

A continuación se muestra la descripción de los pasos a seguir para colocar las tablas, identificadas en ambas base de datos, a replicar en modo de simétrico.

Los pasos básicos a seguir para la configuración son los siguientes:

Resumen de Configuración

- Verificación de Global Names
- Creación de usuarios
 - Administrador de la replicación (REPADMIN)
 - Propagador (REPADMIN)
- Privilegios Administrador, Propagador, Owners
- Creación de Database Links
- Definición de Grupo Multimaster

- Inclusión de objetos en grupo Multimaster
- Definición de column groups (conflictos)
- Definición de rutinas de resolución de conflictos
- Generación de Soporte a Replicación
- Adición de Master Site
- Definición de Intervalos de Replicación

Habilitar Grupo de Replicación

Primero Procederemos con la configuración de los global names:

Los databases links que referencien a esta B.D. deben llamarse igual al global_name

Se recomienda crear un alias de sqlnet con el mismo global_name de la base de datos a que se haga referencia

Usado para seguridad en la consistencia de transacciones diferidas

```
SQL> conn system/manager@v81.world
```

```
Connected.
```

```
SQL> select * from global_name;
```

```
GLOBAL_NAME
```

```
-----
```

```
V81.WORLD
```

```
SQL> alter database rename global_name to ORLANDO.WORLD;
```

```
Database altered.
```

```
SQL> select * from global_name;
```

```
GLOBAL_NAME
```

```
-----
```

```
ORLANDO.WORLD
```

```
SQL> conn system/manager@v82.world
```

```
Connected.
```

```
SQL> alter database rename global_name to MIAMI.WORLD;
```

```
Database altered.
```

Ahora se procedera con la creación de usuarios:

• **Usuario Administrador de la Replicación:**

- **Configura y mantiene el ambiente de replicación**
- **Un administrador por cada grupo o un administrador global**

```
SQL> conn system/manager@ORLANDO.WORLD
```

```
Connected.
```

```
SQL> create user REPADMIN identified by repadmin;
```

```
User created.
```

```
SQL> conn system/manager@MIAMI.WORLD
```

```
Connected.
```

```
SQL> create user REPADMIN identified by repadmin;
```

```
User created.
```

• **Usuario Propagador de la Replicación:**

- **Se encarga de aplicar las transacciones diferidas en cada nodo remoto**
- **Este usuario puede ser el mismo administrador REPADMIN**

```
SQL> conn system/manager@ORLANDO.WORLD
```

```
Connected.
```

```
SQL> create user PROPAGADOR identified by propagador;
```

User created.

```
SQL> exec dbms_defer_sys.register_propagator(username =>
'PROPAGADOR')
```

```
SQL> conn system/manager@MIAMI.WORLD
```

Connected.

```
SQL>
```

```
SQL> create user PROPAGADOR identified by propagador;
```

User created.

```
SQL> exec dbms_defer_sys.register_propagator(username =>
'PROPAGADOR')
```

Ahora se procede con la configuración Multi Master

```
SQL> conn system/manager@ORLANDO.WORLD
```

Connected.

```
SQL> exec dbms_repcat_admin.grant_admin_any_repgroup('REPADMIN')
```

PL/SQL procedure successfully completed.

```
SQL> grant execute any procedure to PROPAGADOR;
```

```
SQL> conn system/manager@MIAMI.WORLD
```

Connected.

```
SQL> exec dbms_repcat_admin.grant_admin_any_repgroup('REPADMIN')
```

PL/SQL procedure successfully completed.

```
SQL> grant execute any procedure to PROPAGADOR;
```

Se otorgan los privilegios al dueño del esquema:

```
grant alter session to SCOTT;  
grant create cluster to SCOTT;  
grant create database link to SCOTT;  
grant create sequence to SCOTT;  
grant create session to SCOTT;  
grant create synonym to SCOTT;  
grant create table to SCOTT;  
grant create view to SCOTT;  
grant create procedure to SCOTT;  
grant create trigger to SCOTT;  
grant unlimited tablespace to SCOTT;  
grant create type to SCOTT;  
grant create any snapshot to SCOTT;  
grant alter any snapshot to SCOTT;  
grant execute on DBMS_DEFER to SCOTT;
```

Una vez otorgados los privilegios se procederá con la creación de los database links en el esquema multi master.

```
SQL> conn system/manager@ORLANDO.WORLD
```

Connected.

```
SQL> create PUBLIC database link MIAMI.WORLD  
2 using 'MIAMI.WORLD';
```

Database link created.

```
SQL> conn REPADMIN/REPADMIN@ORLANDO.WORLD
```

Connected.

```
SQL> create database link MIAMI.WORLD
  2 connect to REPADMIN identified by REPADMIN;
```

Database link created.

```
SQL> conn system/manager@MIAMI.WORLD
```

Connected.

```
SQL> create public database link ORLANDO.WORLD
  2 using 'ORLANDO.WORLD';
```

Database link created.

```
SQL> conn REPADMIN/REPADMIN@MIAMI.WORLD
```

Connected.

```
SQL> create database link ORLANDO.WORLD
  2 connect to REPADMIN identified by REPADMIN;
```

Database link created.

Una vez creados los database links procederemos a definir el grupo multimaster.

```
SQL> conn repadmin/repadmin@ORLANDO.WORLD
```

Connected.

```
SQL> exec dbms_repcat.create_master_repgroup('PERSONAL')
```


PL/SQL procedure successfully completed.

```
SQL> select gname, dblink, masterdef
2 from dba_repsites;
```

```
GNAME      DBLINK      M
-----
PERSONAL   ORLANDO.WORLD  Y
```

```
SQL> select job, what
2 from user_jobs;
```

```
      JOB WHAT
-----
425 dbms_repcat.do_deferred_repcat_admin('PERSONAL', FALSE);
```

Ahora procedemos con la creación del grupo incluyendo los objetos a replicar.

```
SQL> conn repadmin/repadmin@ORLANDO.WORLD
```

Connected.

```
SQL> begin
2  dbms_repcat.create_master_reobject(
3  gname => 'PERSONAL',
4  type => 'TABLE',
5  oname => 'DEPT',
6  sname => 'SCOTT',
7  use_existing_object => TRUE,
8  copy_rows => FALSE);
9 end;
10 /
```

PL/SQL procedure successfully completed.

```
SQL> select gname, sname, oname
2 from dba_reobject;
```

```
GNAME      SNAME      ONAME
-----
PERSONAL   SCOTT      DEPT
```

Una vez creado el grupo con los objetos a replicar se procede a crear el soporte a la replicación.

```
SQL> conn repadmin/repadmin@ORLANDO.WORLD
```

Connected.

```
SQL> begin
2  dbms_repcat.generate_replication_support(
3    sname => 'SCOTT',
4    oname => 'DEPT',
5    type => 'TABLE',
6    min_communication => TRUE);
7 end;
8 /
```

PL/SQL procedure successfully completed.

```
SQL> select gname, sname, oname, type
2 from dba_reobject;
```

GNAME	SNAME	ONAME	TYPE
PERSONAL	SCOTT	DEPT\$RP	PACKAGE
PERSONAL	SCOTT	DEPT	TABLE
PERSONAL	SCOTT	DEPT\$RP	PACKAGE BODY
PERSONAL	SCOTT	DEPT\$RR	PACKAGE
PERSONAL	SCOTT	DEPT\$RR	PACKAGE BODY

```
SQL> select sname, oname, type
  2 from dba_repgenerated;
```

SNAME	ONAME	TYPE
SCOTT	DEPT\$RP	PACKAGE
SCOTT	DEPT\$RP	PACKAGE BODY
SCOTT	DEPT\$RR	PACKAGE
SCOTT	DEPT\$RR	PACKAGE BODY

Una vez Generado el soporte de la replicación se incluye el master site al cual se replicara.

```
SQL> conn repadmin/repadmin@ORLANDO.WORLD
```

```
Connected.
```

```
SQL> begin
```

```
  2 dbms_repcat.add_master_database(
  3   gname => 'PERSONAL',
  4   master => 'MIAMI.WORLD',
  5   use_existing_objects => TRUE,
  6   copy_rows => FALSE,
```

```
7   propagation_mode => 'ASYNCHRONOUS');
8 end;
9 /
```

PL/SQL procedure successfully completed.

```
SQL> select gname, dblink, masterdef
2 from dba_repsites;
```

GNAME	DBLINK	M
PERSONAL	ORLANDO.WORLD	Y
PERSONAL	MIAMI.WORLD	N

Ahora procederemos a definir los intervalos en los cuales se deberá replicar.

```
SQL> conn repadmin/repadmin@ORLANDO.WORLD
```

Connected.

```
SQL> begin
```

```
2   dbms_defer_sys.schedule_push(
3     destination => 'MIAMI.WORLD',
4     next_date => sysdate,
5     interval => '*1:Mins*/ sysdate + 1/(60*24)',
6     stop_on_error => FALSE,
7     delay_seconds => 0,
8     parallelism => 1);
9 end;
```

10 /

PL/SQL procedure successfully completed.

```
SQL> select job, what  
2 from user_jobs;
```

JOB WHAT

```
-----  
425 dbms_repcat.do_deferred_repcat_admin('PERSONAL', FALSE);  
444 declare rc binary_integer; begin rc :=  
sys.dbms_defer_sys.push(destination=>  
    'MIAMI.WORLD', stop_on_error=>FALSE, delay_seconds=>0,  
parallelism=>1); end;
```

```
SQL> conn repadmin/repadmin@MIAMI.WORLD  
Connected.
```

```
SQL> begin  
2 dbms_defer_sys.schedule_push(  
3 destination => 'ORLANDO.WORLD',  
4 next_date => sysdate,  
5 interval => '/*5:Mins*/ sysdate + 5/(60*24)',  
6 stop_on_error => FALSE,  
7 delay_seconds => 0,  
8 parallelism => 1);  
9 end;  
10 /
```

Finalmente habilitamos la replicación

```
SQL> conn repadmin/repadmin@ORLANDO.WORLD
```

Connected.

```
SQL> begin
```

```
2  dbms_repcat.resume_master_activity(
```

```
3    gname => 'PERSONAL');
```

```
4  end;
```

```
5 /
```

PL/SQL procedure successfully completed.

```
SQL> select gname, status
```

```
2  from dba_repgroup;
```

GNAME	STATUS
PERSONAL	NORMAL

6

Conclusiones

6.1 Desventajas observadas después de implementar la replicación

Una de las desventajas más notorias fué la cantidad de recursos requeridos para la implementar el esquema de replicación, entre ellos estan:

- Un servidor de producción con 512 MB de memoria
- Un servidor de pruebas
- Un administrador responsable de los servidores, bases de datos, esquema de replicación y la aplicación.

6.2 Ventajas observadas después de la implementar la replicación

A continuación se muestra la tabla comparativa de tiempos de respuesta con acceso a la información de manera local, después de haber implementado el esquema de replicación simétrica.

País	Tiempo promedio por operación (seg.)	Tiempo promedio por operación (seg.)
	Antes	Después
Argentina	15	2
Brasil	10	3
Costa Rica	20	1
Colombia	17	2
Perú	35	1
Puerto Rico	20	1
Chile	17	2
México	9	3
Venezuela	20	2
Ecuador	45	1

6.3 Conclusiones finales

El hecho de requerir esquemas de replicación obedecen básicamente a cuatro factores:

1. Las compañías utilizan una amplia variedad de plataformas de hardware y software.

2. Las compañías se están volviendo más distribuidas día a día.
3. Las compañías requieren de esquemas para recuperarse de desastres.
4. El principio básico de que a los humanos no les gusta el hecho de compartir.

La replicación de información bajo un esquema distribuido reduce los costos de la red, debido a las consultas, y mejora la disponibilidad y el desempeño de la aplicación. Sin embargo por otro lado se tiene la perspectiva sobre la centralización masiva, esto es una solución no distribuida. Esta perspectiva se debe a un menor costo, mejor disponibilidad sobre un ancho de banda de red casi ilimitado, la centralización corporativa de información volverá de nuevo a ser una sólida alternativa. Consideramos que este hecho bajo la premisa de que es técnicamente factible, pero no refleja la dirección actual de las organizaciones.

Las compañías se están volviendo mas planas y están delegando el control en las manos de los que realmente manejan el negocio. Esto puede ser observado en la descentralización de las funciones de las empresas a un amplio espectro de localidades. Estos nuevos usuarios realmente son los creadores, dueños y controladores de la información. De tal forma que ellos demandan los beneficios de un ambiente distribuido no solamente acceso remoto a información centralizada, sino también toda la libertad asociada con la versatilidad. Ellos quieren realizar y manejar el negocio en base a su criterio, sin limitaciones tecnológicas. Para cumplir este requerimiento, la información debe de ser almacenada de manera redundante y algún tipo de mecanismo de replicación es necesario para mantener la consistencia.

La información solo debe de ser almacenada de manera redundante si la ganancia de performance y de disponibilidad justifican ampliamente el costo asociado al mantenimiento de la consistencia.

En el caso de la presente tesis, el costo asociado a implementar el esquema de replicación fue justificado con el incremento en la capacidad de respuesta del área de atención a clientes tanto en el volumen de llamadas como en la mejora del tiempo de respuesta.

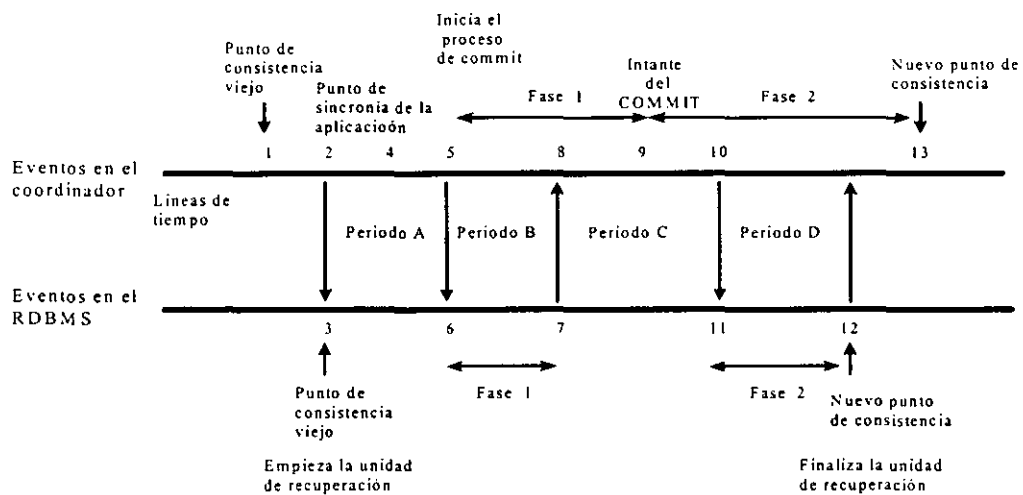
Bibliografía

1. An Introduction to Database Systems, Sixth Edition
C.J. Date
Addison Wesley, 12/1994
2. Data Replication : Tools and Techniques for Managing Distributed
Information
Buretta, Marie
John Wiley & Sons (Sd); 02/1997
3. Oracle8 Distributed Database Systems
Release 8.0
A58247-01

APÉNDICE

A

Commit de dos Fases



Explicación de un commit en dos fases:

- 1 Los datos en el commit coordinador se encuentran en un estado de consistencia.
- 2 El coordinador a través de una aplicación envía una sentencia para realizar una actualización.
- 3 La unidad de trabajo inicia. Este es el inicio de una unidad de recuperación.

- 4 El coordinador a través de una aplicación o proceso alcanza un punto de sincronización.
- 5-6 El coordinador inicia el procesamiento del commit. La Fase 1 del proceso commit inicia.
- 7 El RDBMS exitosamente completa el procesamiento de la Fase 1 y notifica al coordinador.
- 8 El coordinador recibe la notificación del RDBMS.
- 9 El coordinador exitosamente completa su Fase 1 de procesamiento. El coordinador registra el instante del commit, el coordinador inicia la Fase 2 del procesamiento.
- 10 El coordinador notifica al RDBMS que inicie su Fase 2.

- 11 El RDBMS inicia su Fase 2 del procesamiento.

- 12 El RDBMS exitosamente completa su procesamiento de la Fase 2 y entonces notifica al coordinador que ha finalizado.
- 13 El coordinador finaliza el procesamiento de la Fase 2. Los datos involucrados en esta unidad de trabajo ahora son consistentes.

Si una falla ocurre mientras el coordinador esta conectado al RDBMS, el RDBMS debe determinar durante el re inicio si tiene que realizar un commit o rollback a alguna unidad de recuperación. Para ciertas unidades de recuperación el RDBMS tiene suficiente información para realizar la decisión. Para otros tiene que obtener la información del coordinador cuando este restablezca su conexión.

Periodo *a* o *b* La falla ocurre antes de que el RDBMS completa su Fase 1. Durante el re inicio el RDBMS deshace las actualizaciones.

Periodo *c* La falla ocurre después de la fase 1 del RDBMS pero antes de que el RDBMS inicie la fase 2. Unicamente el coordinador sabe si la falla sucedió antes o después del instante del commit. Si sucediera antes, el RDBMS debe de deshacer sus actualizaciones; si sucediera después, el RDBMS deberá realizar el commit a las actualizaciones. Al inicio, el RDBMS espera por información por parte del coordinador antes de procesar esta unidad de recuperación.

Periodo *d* El RDBMS falla después de que empezó su propia fase 2 de procesamiento. Al re inicio, el RDBMS realiza commit a las actualizaciones.

APÉNDICE

B

Propiedades ACID de las Transacciones

Las Transacciones proveen un simple modelo de éxito o falla. Una transacción ya sea que totalmente realice un commit -todas sus acciones suceden o aborta de forma completa- ninguna de sus acciones se realizan. Esto puede se puede definir mejor a través de las llamadas propiedades ACID de las transacciones. ACID es un acrónimo para Atómica, Consistente, Isolated (Aislada) y Durable.

- Una transacción es atómica. Esto significa que tanto todas sus acciones suceden o ninguna de ellas se lleva a cabo.
- Una transacción es consistente. Esto significa que la transacción como un todo representa una correcta transformación de los manejos de los recursos involucrados. Pro ejemplo, lleva a la base de datos de un estado consistente a otro estado inconsistente.
- Una transacción es aislada (isolated). Esto significa que cada transacción ejecuta pensando en que no existen otras transacciones concurrentes.
- Una transacción es durable. Esto significa que los efectos de una transacción que realizó un commit sobrevive las fallas.

Tanto los sistemas manejadores de bases de datos y los sistemas TP proveen las propiedades ACID para las transacciones. Para lograr esto, ellos proveen tal funcionalidad como los mecanismos de bloqueo, bitácoras y del protocolo de commit en dos fases. Todo lo que el programador necesita hacer es proporcionar los símbolos de que demarquen a la transacción con palabras claves como BEGIN y COMMIT.

Hay que observar que los mecanismos de bloqueo permiten el aislamiento frente a la concurrencia.
