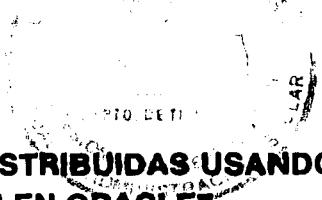


24021
22



UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
"ACATLAN"



BASES DE DATOS DISTRIBUIDAS USANDO REPLICACION EN ORACLE7.

T E S I S

QUE PARA OBTENER EL TITULO DE:

**LICENCIADO EN MATEMATICAS
APLICADAS Y COMPUTACION**

P R E S E N T A :

OLIVER SALVADOR JIMENEZ HERNANDEZ

ASESOR: M. en C. JUDITH JARAMILLO LOPEZ



TESIS CON
FALLA DE ORIGEN

ACATLAN EDO. DE MEX.

MARZO DE 2003



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 mis mejores amigos, quienes no dejaron de confiar en mí, siempre me dieron su apoyo sin condiciones. Me enseñaron los valores que me llevan por la vida y nunca me dejaron caer aun cuando parecía que no había para más. Gracias papás.

A mi papá, porque siempre me impulsaste y me diste todo lo que podías, por enseñarme a ser una persona responsable con tu ejemplo, y por siempre estar ahí. Gracias te quiero.

A mi mamá gracias por estar siempre conmigo, por esperarme por las noches, por todos los sacrificios que haces por darnos todo lo que esta a tu alcance y por creer siempre en mí. Te quiero Lupita.

A Israel mi hermano, porque tu compañía hizo el camino menos difícil. Porque con tus acciones siempre me demuestras tu valor como persona y porque siempre estoy aprendiendo de ti. A sus nenitas que le han llenado de luz la mirada.

A Eder, hermanito porque eres una persona brillante y capaz, porque tienes el coraje de aprender de tus errores y demostrármelo. Por tu ayuda y paciencia, y porque sé que me seguirás llenando de orgullo.

A Deyito, por ser y estar, por tu energía y alegría de vivir, por ser siempre quien me impulsa a ser mejor, me apoyas y confías en lo que hago. Te quiero linda.

A mi tía Paty por tu apoyo, por estar pendiente, por siempre brindar tu ayuda de manera desinteresada. A mis tíos y abuelos por siempre preguntar, por dar apoyo y aliento de diferentes maneras, Gracias.

A mis amigos en la Universidad, con quienes compartí muy buenos momentos y experiencias. Porque crecí junto con ellos. Gracias.

Agradezco a la Universidad, por darme la formación para ser un profesional responsable y capaz. A profesores como Judith, por el conocimiento compartido y por enseñarnos a confiar en nosotros mismos y demostrarnos que sí podíamos.

A Dios por permitirme estar vivo y compartir con ustedes este momento de alegría.

B

TESIS CON
FALLA DE ORIGEN

Bases de datos distribuidas usando replicación en Oracle7

**TESIS CON
FALLA DE ORIGEN**

Objetivo:

Analizar las propiedades que debe cumplir una base de datos distribuida e implementar un esquema de replicación utilizando uno de los métodos disponibles en Oracle7 como alternativa al uso de las bases de datos centralizadas.

Índice.

INTRODUCCIÓN.	v
CAPÍTULO I. Introducción a las bases de datos distribuidas.	1
1.1 Bases de datos relacionales y sistemas manejadores de bases de datos	2
1.2 Two-phase commit. Protocolos de compromiso de dos y tres fases	10
1.3 Características y terminología de las bases de datos distribuidas	23
1.4 Particionamiento de la información	30
CAPÍTULO II. Cliente-Servidor y sistemas distribuidos.	35
2.1 Arquitectura Cliente-Servidor	36
2.2 Diseño de un ambiente distribuido	41
2.3 Consideraciones de infraestructura y administración del sistema distribuido	52
CAPÍTULO III. Arquitectura del Oracle7 Server y SQL*Net.	59
3.1 Introducción y estructuras de datos del Oracle7 Server	60
3.2 Arquitectura del Oracle7 Server	64
3.3 Arquitectura del SQL*Net	70
CAPÍTULO IV. Replicación de datos.	75
4.1 Introducción a la replicación de datos	76
4.2 Snapshots de sólo lectura y actualizables	81
4.3 Replicación simétrica y solución de conflictos	88
CAPÍTULO V. Implementación de un esquema de replicación.	99
5.1 Definición del caso de estudio	100
5.2 Construcción del esquema de replicación	104
5.3 Implementación de una base de datos distribuida usando replicación	113
5.4 Revisión de resultados	129
CONCLUSIONES.	133
APÉNDICES.	
A. Modelo Entidad-Relación	135
B. Vistas para el manejo de replicación del diccionario de datos de Oracle7	143
GLOSARIO.	157
BIBLIOGRAFÍA.	163

Introducción.

Las bases de datos distribuidas surgen por la necesidad de acceso a datos compartidos de las organizaciones de hoy en día. Su crecimiento normal las obliga a repartir procesos y operaciones entre varias localidades como pueden ser almacenes, plantas de producción y sucursales. Esta repartición frecuentemente se da más allá de los límites de una red de área local (LAN) y pueden estar geográficamente dispersas formando una red de área amplia (WAN)

La actual arquitectura Cliente-Servidor que sigue siendo muy usada, se creó para optimizar el uso de recursos al dividir las tareas de procesamiento en ambientes LAN, se basa en un intercambio importante de mensajes que significa tráfico en la red, pero permite a la vez el uso extensivo de la interfaz gráfica en las aplicaciones y de otros recursos compartidos. Sin embargo, con la separación geográfica de los servidores de bases de datos de las localidades donde se ejecutan las aplicaciones de usuario, fuera de los límites de la red LAN, se genera un problema importante que es un pobre tiempo de respuesta y aumento en el uso de los recursos de red.

La idea general de una base de datos es mantener información consistente para todos los usuarios. Este principio hizo que el paso de una base de datos centralizada con información consistente, a una base de datos distribuida manteniendo la misma consistencia, no fuera algo fácil de alcanzar.

Tal ha sido la importancia por lograr esta distribución de datos que han sido definidas una serie de reglas y estándares que indican cuales deben ser las propiedades que debería tener una base de datos distribuida. Esto ha sido acompañado por nuevas metodologías para el diseño y para la colocación de datos en las diferentes localidades.

Las formas de repartición de datos que fueron identificadas por C.J. Date en sus reglas de las bases de datos distribuidas, incluyen la posibilidad de fragmentar una fuente única de datos entre varias localidades, y permitir además la existencia y distribución de copias de datos entre diferentes servidores.

Las reglas de las bases de datos distribuidas y la necesidad de las organizaciones por distribuir datos, o en ocasiones consolidar información en un lugar específico, impulsaron el desarrollo y adecuación de tecnologías existentes en manejadores de bases de datos comerciales, de tal manera que permitieran su interacción en un ambiente distribuido. Además de la tecnología, se comenzaron a desarrollar API's gráficos especiales para la administración e implementación del ambiente distribuido.

El objetivo general de esta tesis es *"Analizar las propiedades que debe cumplir una base de datos distribuida e implementar un esquema de replicación utilizando uno de los métodos disponibles en Oracle7 como alternativa al uso de las bases de datos centralizadas"*

La tesis plantea dos hipótesis que se buscan probar y son:

"Al acercar la fuente de datos a las aplicaciones de usuario, usando una base de datos distribuida se cumplen las propiedades que las definen y se mejora el tiempo de procesamiento de datos en comparación con el que se tiene con una base de datos centralizada."

"La construcción de un esquema de replicación, con mecanismos existentes a partir de la versión Oracle7, se facilita usando un API gráfico de administración e implementación de bases de datos distribuidas."

Introducción.

Para el desarrollo de la tesis se eligió el Oracle7 por las siguientes razones. En la actualidad los ambientes corporativos pueden estar formados por múltiples plataformas y diferentes manejadores de bases de datos, con enfoques de datos diferentes que pueden ser o no relacionales. Una base de datos distribuida usando Oracle7 como RDBMS puede comunicarse con otros nodos que no son Oracle e inclusive pueden no ser relacionales. Sin importar tampoco la plataforma usada o el tipo de red donde la base remota se encuentra.

Si bien es posible la integración con diferentes manejadores de bases de datos, la tesis se desarrollara usando solamente el RDBMS de Oracle.

Además de ser una versión muy estable el Oracle7 es la primera versión que incluye las características y estructuras para replicación y distribución de datos que encontramos en versiones posteriores como lo son Oracle8, Oracle8i y Oracle9i. Los principios que encontramos en las versiones recientes están basados en la arquitectura introducida en el Oracle7. La tesis no dejará de mencionar algún cambio significativo o mejora entre las diferentes versiones.

Por lo anterior y por permitir el uso de una interfaz grafica de Oracle llamada *Administrador de replicación* para administrar y construir una base de datos distribuida es que se decidió el uso de la versión Oracle7.

La tesis esta dividida en los siguientes capítulos, siendo el primero **Introducción a las bases de datos distribuidas**, que revisa los principios de las bases de datos distribuidas. Define conceptos y principios básicos para distribución de datos en una base de datos relacional y como se comunican en un ambiente distribuido.

El capítulo segundo **Cliente-Servidor y sistemas distribuidos**, analiza las diferencias entre el procesamiento distribuido del Cliente-Servidor y la base de datos distribuida, que si bien se encuentran relacionados son diferentes. Analiza además la comunicación entre los RDBMS's en un ambiente distribuido. Aborda el diseño del ambiente distribuido y que consideraciones deberfan tomarse antes de la implementación.

Ya definido el uso del Oracle7, el capítulo tercero **Arquitectura del Oracle7 Server y SQL*Net**, revisa las características principales del RDBMS Oracle7, la arquitectura y la interacción de los mecanismos involucrados con la distribución de y replicación de datos. Se introduce la arquitectura del SQL*Net, como componente responsable para comunicar clientes con el RDBMS para procesamiento Cliente-Servidor, y diversos RDBMS's en un ambiente distribuido.

El capítulo cuarto llamado **Replicación de datos**, esta dedicado a los diferentes tipos de replicación que pueden ser implementados con Oracle, la replicación elegida dependerá de los requerimientos de distribución y deben ser completamente transparentes para el usuario final. La replicación entre los nodos podrá implementarse en tiempo real o de manera diferida y serán señaladas las ventajas y desventajas de usar cada uno de estos métodos.

El último capítulo **Implementación de un esquema de replicación**, define un caso de estudio que nos permitiría validar o no las hipótesis planteadas en la tesis. Se buscara pasar de una base de datos centralizada a una base de datos distribuidas usando un ambiente de replicación usando Oracle7.

Capítulo I.

Introducción a las bases de datos distribuidas.

1.1 Bases de datos relacionales y sistemas manejadores de bases de datos

1.1.1 Antecedentes.

Con el uso más serio de las computadoras en aplicaciones comerciales, tanto el software como el hardware han estado en constante evolución buscando cubrir las crecientes y cada vez más variadas necesidades de los usuarios y negocios.

En sus inicios, las computadoras y el software para el desarrollo de aplicaciones, se limitaban al procesamiento de datos. Es decir, muchos de los procesos dentro de las empresas se automatizaron. Esta automatización consistió en la creación de sistemas independientes para cada una de las áreas de la organización. También fueron conocidos como sistemas de manejo de archivos.

Las características principales de este tipo de sistemas eran:

- Estaban enfocados a las necesidades individuales de procesamiento de datos de cada uno de los departamentos.
- Los archivos de datos usados por estos sistemas eran únicos y pertenecían sólo a un departamento.
- La lógica del programa usada dependía del formato y descripción de los datos.

De esta manera, se tenía un sistema para la facturación de la empresa, y se tenía otro independiente para el área de recepción de pedidos. La diferencia en la lógica de los programas y la manera en la que los datos eran procesados y organizados para cada departamento, se debe a que estos sistemas se desarrollaban con lenguajes de tercera generación (3GL) como lo es el Cobol o el C. Por consecuencia, cualquier cambio en las necesidades de manejo de información del departamento requería un cambio en los sistemas de procesamiento, mismo que era muy tardado y costoso.

Si cada área de la empresa contaba con un sistema para procesar sus datos y los manejaba de manera única, el intercambio de información entre departamentos se volvía prácticamente imposible pues no había manera de controlar la repetición y poca exactitud de los datos que cada área manejaba.

Con la automatización de cada vez más procesos en las organizaciones, y dado que en la mayoría de las ocasiones estos archivos y programas de aplicación se desarrollaron en un periodo largo de tiempo, muy probablemente por distintos programadores, era de esperar que sus archivos de datos tuvieran distintos formatos y que los programas estuviesen escritos en varios lenguajes de programación.

De lo anterior se marcan las siguientes desventajas [KOSI93]:

- **Redundancia e inconsistencia de los datos.** Es posible que diferentes archivos de datos y programas de aplicación, al ser creados por distintos programadores, contengan las mismas piezas de información. Esta redundancia aumenta los costos de almacenamiento y acceso, además de incrementar la posibilidad de que exista inconsistencia en la información, es decir que las copias de la misma información no concuerden entre sí.

- **Dificultad para tener acceso a los datos.** La capacidad para obtener diferentes formatos de la información almacenada se ve limitada dado que la aplicación no tiene la capacidad de recuperar la información requerida en forma eficiente.
- **Aislamiento de los datos.** Dado que los datos están repartidos en varios archivos, y éstos pueden tener diferentes formatos, la dificultad para escribir nuevos programas de aplicaciones para obtener los datos apropiados es mucho mayor.
- **No permite acceso concurrente.** La necesidad de acceso de varios usuarios a la misma información, en un mismo periodo de tiempo se fue convirtiendo en una necesidad para esas aplicaciones, sin embargo, este tipo de sistemas no contaba con ningún mecanismo que garantizara las actualizaciones concurrentes de los datos, esto sólo puede resultar en información inconsistente.
- **Problemas de seguridad.** No es recomendable que todos los usuarios del sistema tengan acceso a toda la información. Los niveles de seguridad que se podían definir en estas aplicaciones era la brindada por el sistema operativo, no era posible controlarlo basándose en la responsabilidad que cada usuario tenía dentro de la aplicación.
- **Problemas de integridad de datos.** Los datos en el sistema deben cumplir muchas veces con algo llamado reglas del negocio. En un principio estas reglas o propiedades de los datos pueden limitarse por la aplicación misma; el problema viene al agregar nuevas reglas o modificarlas implica cambiar los programas que hacen el acceso a los datos.

A pesar de estos inconvenientes, el uso comercial de este tipo de sistemas siguió en aumento. Se comenzó a concebir la idea de la sistematización de las aplicaciones, este nuevo tipo de aplicaciones debía ser capaz de usar archivos de datos comunes a todas las áreas de una organización, en otras palabras de una base de datos que proveyera de información a las diferentes aplicaciones.

La consecuencia de esto fue que a principios de la década de los 70's, el grupo de DBTG (Data Base Task Group) de CODASYL (Conference On Data Systems Languages) presentó los principios para el desarrollo de los **Sistemas Administradores de Bases de Datos (DBMS por sus siglas en inglés)**, cuya premisa sería el manejar los datos de cualquier organización bajo una estructura homogénea, que evitara la redundancia de la información y garantizara su fácil acceso.

De la misma forma, [DATE95] en 1972 un grupo de estudio fue establecido por un comité de ANSI (American National Standard Institute) para hacer una recomendación para un modelo de sistemas de manejo de bases de datos. Este grupo fue llamado SPARC (Standards and Planning Architecture Requirements Committee) El grupo publicó un par de reportes en 1975 y en 1978, los cuales definen la **arquitectura ANSI / SPARC** para sistemas manejadores de bases de datos. Actualmente es la arquitectura usada como el estándar y define un modelo de base de datos dividido en tres niveles:

1. **El nivel externo**
2. **El nivel conceptual**
3. **El nivel interno**

- **Nivel externo**, es la vista del usuario / aplicación, definen la manera en la que el usuario observa la base de datos; por ejemplo, pantallas de captura y / o despliegue de información.
- **Nivel conceptual**, es como la empresa u organización ve la base de datos. Equivale a la concepción que tiene el administrador de la base de datos, es del nivel conceptual donde es posible generar múltiples presentaciones de los mismos datos para los usuarios del nivel externo.
- **Nivel interno**, son especificados los tipos de datos, y la manera en la que serán almacenados físicamente. No importa el modelo de datos que va a ser usado ya que internamente los datos no son almacenados en forma de tablas o relaciones (en el caso del enfoque relacional), físicamente son un conjunto de registros, apuntadores y otras estructuras de datos.

1.1.2 Definición de una base de datos.

Una base de datos [KOSI93], se define como un conjunto de datos relacionados entre sí, almacenados de manera homogénea, que son no redundantes y de fácil acceso. El objetivo primordial de una base de datos es el permitir un ambiente en que sea posible guardar y recuperar información de manera eficiente y segura.

En una base de datos cada usuario y cada programa autorizado podrá tener acceso a los datos que requiera. Al ser datos comunes para ellos la redundancia de la información es minimizada y la exactitud de los datos es mayor.

Asociado a la base de datos esta el software usado para manejar y tener acceso a los datos, esta pieza de software es llamada DBMS (DataBase Management System) o sistema manejador de base de datos.

Los sistemas de bases de datos están diseñados para manejar grandes cantidades de información. El manejo de los datos incluye la definición de las estructuras para el almacenamiento y los medios para el manejo de la información. Conceptualmente el software para el manejo de bases de datos esta diseñado para:

1. *Un usuario ejecuta una solicitud de datos, utilizando algún lenguaje entendido por el DBMS.*
2. *El DBMS recibe la petición del usuario y la analiza.*
3. *El DBMS revisa el esquema externo para el usuario, lo relaciona con la representación conceptual; posteriormente el esquema conceptual es traducido al nivel interno.*
4. *El DBMS ejecuta las operaciones internas y presenta los resultados al usuario.*

Revisando más a detalle, el DBMS es capaz de entender las definiciones de datos (existentes en el nivel externo) y como se relaciona con los niveles conceptual e interno para entonces cambiar la definición por los objetos físicos correspondientes. Podemos decir entonces que el DBMS incluye un lenguaje de procesamiento para cada una de las definiciones de datos o 'Data Definition Language' (DDL por sus siglas en inglés)

El DBMS entiende las definiciones DDL, por ejemplo: sabe que el registro externo EMPLEADO incluye un campo SALARIO; entonces es capaz de usar este conocimiento y responder a las consultas de los usuarios como lo son "los empleados con un salario menor a \$5,000."

Además de contar con el lenguaje DDL, el DBMS puede manejar peticiones del usuario para recuperar, actualizar o borrar datos de la base de datos, o agregarlos si son nuevos. Esto se agrupa en otro grupo de instrucciones llamadas lenguaje de manipulación de datos o 'Data Manipulation Language' (DML por sus siglas en inglés)

Otras dos propiedades adicionales del software DBMS es que permiten monitorear las peticiones de los usuarios y rechazar cualquier intento de violar la seguridad y reglas de integridad definidas por el administrador de la base de datos. Estos niveles de seguridad son independientes al sistema operativo de la máquina donde reside el DBMS. También es capaz de controlar la concurrencia de varios usuarios usando los mismos datos en un al mismo tiempo.

Los datos en la base son homogéneos por la similitud que tienen. En el momento en que se captura la información pasa a formar parte de los datos existentes y al ser consultados por una aplicación de usuario o cualquier otro proceso intermedio, generará información útil. Esta integración implica que cualquier redundancia entre los de datos será completa o parcialmente eliminada.

Los sistemas de bases de datos que residen en grandes equipos de computo son generalmente sistemas multiusuario, mientras que los sistemas residentes en equipos pequeños son llamados sistemas de un sólo usuario. Los sistemas para un sólo usuario permiten acceso a los datos a un usuario a la vez; mientras que los sistemas multiusuario permiten el acceso a los datos de múltiples usuarios de manera concurrente.

Por regla general, los sistemas multiusuario cumplen con otra propiedad, además de poder mantener datos integrados, que es el permitir acceso compartido. Se definen como compartidos por ser vistos como piezas individuales de datos que pueden ser manipulados entre varios usuarios diferentes. En el momento en que cada uno de estos usuarios puede tener acceso a la misma sección de datos al mismo tiempo, se convierte en un **acceso concurrente**. El compartir datos, concurrentemente o no, depende en gran parte de la propiedad de integración de la base de datos.

El mantener los datos en un sitio común donde cumplen con propiedades como lo es la persistencia, integración y acceso concurrente; facilita de manera importante la administración y mantenimiento de la información en la base.

Este control es responsabilidad del **Administrador de la base de datos (DBA** por sus siglas en inglés) Las funciones del DBA incluyen el decidir que datos y como es que estos deben almacenarse en la base, y establecer políticas para el mantenimiento y la seguridad, definiendo el tipo de operaciones que cada usuario puede ejecutar y en que circunstancias. El DBA es responsable además de asegurar que el sistema opere con un desempeño adecuado y de respaldar los datos existentes.

Otro componente importante asociado a toda base de datos es el **Catálogo o Diccionario de datos**. Puede definirse como una base de datos dentro de la base de datos, es un conjunto de estructuras manejadas en su totalidad por el DBMS y contiene información útil para la

administración de cada objeto definido (incluyendo usuarios, recursos, espacio físico usado por los datos, etc.) El DBA es el único usuario con permiso de sólo lectura a este conjunto de información.

Destaquemos algunos de los beneficios de los sistemas de bases de datos, frente a los sistemas de manejo de archivos.

- **Reducción en la redundancia e inconsistencia de los datos.**
Anteriormente las aplicaciones tenían archivos de datos privados. Esto puede traer problemas con la redundancia de datos, que significa un desperdicio del espacio disponible de almacenamiento. Al compartir los mismos datos, se mantiene una sola versión de ellos eliminando la redundancia.
- **Los datos pueden ser compartidos.**
El compartir significa que las aplicaciones existentes pueden compartir los mismos datos, además podemos incluir nuevas aplicaciones que pueden operar usando los datos ya existentes sin impactar en lo más mínimo las aplicaciones que están en uso.
- **Restricción de acceso a los datos.**
Con un completo control sobre la base de datos, el DBA puede asegurarse que el acceso a los datos se hace únicamente a través de los medios permitidos, restringiendo el acceso a la base y el tipo de operaciones que pueden realizarse sobre datos importantes del negocio. Es posible el establecer diferentes políticas de seguridad que limiten el acceso a un cierto grupo de usuarios, incluyendo el registrar el tipo de operaciones que cada usuario realiza en la base de datos.
- **La integridad de los datos se mantiene.**
Esto se logra cuando los datos en la base son exactos. Inconsistencia o diferencias entre dos entradas a la base de datos que deberían tener el mismo significado es un ejemplo de falta de integridad. El tener un control centralizado permite definir reglas que verifiquen, al agregar o modificar datos, que estos cumplan con las reglas de la organización.

1.1.3 Modelado de bases de datos y enfoques de datos.

El punto de inicio para el desarrollo de bases de datos es el modelo conceptual de datos. Ocupa la parte superior del **Análisis Descendente** y se ejecuta durante la fase de análisis de requerimientos, es muy importante para la construcción de la base de datos. El objetivo, en esta etapa, es desarrollar el **diagrama Entidad-Relación** que representa de manera gráfica los requerimientos de información del negocio. Es independiente de la aplicación para lo que la base será usada, en este punto la preocupación primordial es: ¿Cuáles son los datos? Y no: ¿De qué manera serán usados?.

La **Figura 1.1** abajo, muestra el flujo del análisis descendente que inicia con la revisión de los requerimientos de la organización. Con la información obtenida a través de entrevistas con los usuarios y revisión de procesos se comienza la definición de las entidades que generan y requieren datos. Además se definen cómo están interrelacionadas. Se pueden definir las primeras reglas de integridad del negocio que deben cumplir los datos. Esto es llamado el modelo conceptual. Esta etapa es base para el diseño de la base de datos donde son definidos, y

dependiendo del enfoque que se vaya a utilizar, los objetos de la base, los usuarios y los niveles de acceso que cada grupo de usuarios tendrá. La última etapa es la construcción de los objetos definidos que formarán parte de la base de datos operacional.

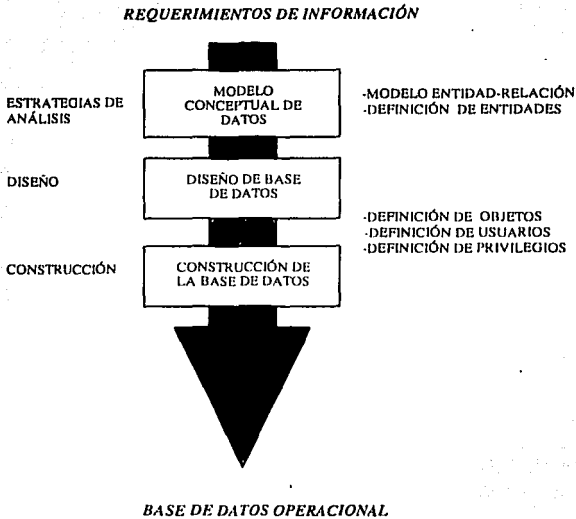


Figura 1.1 Análisis Descendente

Durante las primeras etapas del diseño es necesario mantenerse tan independiente de la aplicación que hará uso de los datos como sea posible. Al inicio del análisis no se sabe cuales son todos los posibles usos que los datos pueden tener además de no afectar el modelo conceptual de datos por futuros requerimientos de la aplicación.

Colocando esto en términos de la arquitectura ANSI / SPARC [DATE95], se busca obtener el esquema conceptual, que es un diseño lógico cuyas propiedades principales son:

- Independiente del hardware.
- Independiente del sistema operativo
- Independiente del DBMS
- Independiente del lenguaje
- Independiente del usuario

Es vital establecer completamente los requerimientos de la información durante la etapa del modelado de datos. Los cambios de los requerimientos durante etapas finales en el ciclo de vida

TESIS CON
FALLA DE ORIGEN

de desarrollo pueden ser muy costosos. Decisiones de diseño hechas al nivel físico tendrán impacto al nivel lógico. Por esta razón se recomienda realizar múltiples revisiones en la etapa de diseño.

La manera correcta de hacer diseño de base de datos es generando un diseño lógico de manera correcta sin prestar atención o preocuparse por la estructura física, como lo pueden ser consideraciones de desempeño o tamaño. Resumiendo hay que completar el "Lógico, entonces físico".

La parte del modelado conceptual consta de los siguientes pasos:

- **Análisis de requerimientos.** Determinado por entrevistas entre analistas y usuarios, produciendo especificaciones formales de requerimientos que incluyen:
 - * *Requerimientos y especificación de los tipos de dato.*
 - * *Relaciones entre los datos.*
 - * *Posibles medios de implementación.*
- **Diseño lógico.** Es la representación esquematizada de los requerimientos de información. Es una forma efectiva para integrar y documentar las relaciones entre datos, la representación más usada es el diagrama Entidad-Relación, que permite entre otras cosas:
 - * *Elimina redundancia de datos al organizar de manera clara y precisa las relaciones entre estos.*
 - * *Es de fácil entendimiento para los usuarios.*
 - * *Permite integrar múltiples aplicaciones y desarrollar otros proyectos.*
 - * *Es posible usar vistas o subconjuntos de un diagrama Entidad-Relación que generen diferentes enfoques de los mismos.*

Una de las mejores y más usadas metodologías para modelar la información es la llamada Entidad-Relación, introducida por Peter Chen en el año de 1976, que fue extendida y mejorada en años posteriores. En 1979 E. F. Codd presento una extensión al modelo Entidad-Relación, también conocido como modelo relacional extendido [DATE95]

De manera general, el modelo Entidad-Relación realiza analogías para todos los objetos semánticos generados en la etapa de análisis. Las cosas de importancia (**Entidades**) dentro de la organización, las propiedades de estas cosas (**Atributos**) y la manera en la que se relacionan unas con otras (**Relaciones**) Es una representación del "¿Qué es hecho?", Y "¿Cómo?" Dentro de la organización. El apéndice A trata de manera más detallada las propiedades del modelo Entidad-Relación y su normalización.

El modelo conceptual de datos es independiente del hardware o del software usados para la implementación. Un modelo Entidad-Relación puede ser utilizado para implementar una base de datos de red, jerárquica o relacional.

1.1.4 Bases de datos relacionales.

1.1.4.1 Definición de RDBMS.

El modelo relacional como herramienta para el modelado de datos. Representa los componentes de la organización en forma de entidades y relaciones. El siguiente paso es la implementación de la base de datos que puede hacerse siguiendo varios enfoques entre los que destaca el **enfoque relacional**.

El enfoque relacional de datos esta basado en el **álgebra relacional** propuesta por E. F. Codd, donde los datos son representados de manera lógica en forma de tablas a partir de relaciones y son manipuladas por operadores especiales que trabajan sobre estas relaciones [DATE95]

El enfoque relacional se caracteriza por:

1. Los datos son percibidos por los usuarios como **TABLAS**; y
2. Los operadores disponibles para los usuarios, para la recuperación de datos por ejemplo, son operadores que generan nuevas tablas a partir de las originales.

Los operadores mínimos del modelo relacional son **PROYECCIÓN**, **SELECCIÓN** (también conocido como **RESTRICCIÓN**) y **JUNTA**. La propiedad de generar otra tabla como resultado de cada operación sobre otras tablas se conoce como la propiedad de **CERRADURA**.

- La operación de **SELECCIÓN** es capaz de recuperar renglones o registros específicos de una tabla.
- La operación de **PROYECCIÓN** recupera determinadas columnas de una tabla.
- La operación **JUNTA** es usada para unir dos tablas basándose en la igualdad de valores de una columna común a las dos tablas.

Como se acaba de explicar, los operadores del álgebra relacional producirán como resultado nuevas tablas, una tabla es una estructura lógica, lo que significa que físicamente no se generara una nueva tabla con los resultados de la operación, el sistema determina cual es la mejor estructura de datos para almacenar y organizar la información, y él mismo se encargada de hacer la representación, en forma de tablas, de esas estructuras físicas.

El software usado para manejar las estructuras físicas es llamado **Sistema Manejador de Bases de Datos Relacionales (RDBMS)** por sus siglas en Inglés) ya que es un DBMS que puede aplicar las operaciones del álgebra relacional sobre el elemento llamado tabla.

Desde finales de la década de los 70s la mayor parte de los productos desarrollados para el manejo de bases de datos se han basado en el enfoque relacional. Actualmente es el enfoque dominante en el mercado de bases de datos. Un ejemplo de esto es el manejador de bases de datos Oracle llamado Oracle7 Server (nuevas versiones del producto son Oracle8, Oracle8i y Oracle9i)

El Oracle Server es un sistema para el manejo de bases de datos basado en el enfoque relacional, cuenta con un lenguaje de manipulación de datos llamado **SQL** (Structured Query Language) que representa los operadores del álgebra relacional. Además de los comandos SQL el Oracle7 Server cuenta con un lenguaje de procedimientos llamado **PL/SQL** (Procedural Language/SQL)

que permite a los desarrolladores controlar el flujo de sentencias SQL, usar variables y escribir rutinas para el manejo de errores.

Es importante señalar que las sentencias SQL son consideradas no procedurales pues solamente es necesario especificarles "¿qué se quiere?", No se preocupan en el "¿cómo?". Las operaciones indican que es lo que necesitan sin tener que especificar que procedimiento usar para hacerlo. La tarea de navegar por las estructuras de datos para generar los resultados es realizada de manera automática por el RDBMS.

[BOCH93] El Oracle Server tiene un par de configuraciones que le permiten cumplir con los requerimientos de cualquier organización sin importar su tamaño o el de su base de datos, las cuales son:

- Oracle7
- Oracle7 Enterprise Edition

Las dos configuraciones proveen una manera muy confiable para el manejo de información que varían desde aplicaciones para departamentos hasta sistemas con alto volumen de transacciones en línea o sistemas para la toma de decisiones. Además de contar con muchas herramientas para la administración del RDBMS y una gran flexibilidad para distribuir datos entre bases de datos diferentes de manera segura y eficiente.

Ambas configuraciones del Oracle Server se basan en el mismo código y pueden coexistir de manera transparente en ambientes distribuidos.

Sus características de seguridad, desempeño y la habilidad de funcionar con otras bases de datos en ambientes distribuidos, casi en cualquier plataforma de sistema operativo; por la gran variedad de productos para administración e implementación, han resultado en que Oracle sea el software de manejo de base de datos relacionales más usado en sistemas de producción.

1.2 Two-phase commit. Protocolos de compromiso de dos y tres fases.

1.2.1 Protección de datos.

Antes de iniciar la definición de las bases de datos distribuidas se revisara un aspecto importante de los sistemas manejadores de base de datos relacionales (distribuidos o no) que esta relacionado con los mecanismos para la protección de datos de cualquier peligro, ya sea deliberado o accidental.

El sistema de base de datos debe proveer un conjunto de controles para proteger los datos de cualquier riesgo, el mecanismo provisto es la **RECUPERACIÓN**. Su trabajo es traer la base de datos a un estado sabido como correcto después de una falla del sistema. [DATE95]

El principio fundamental de la recuperación es la redundancia, es decir, es la manera de asegurar que la información en la base de datos es recuperable, sabiendo que cada pieza de información puede ser reconstruida a partir de información almacenada, de manera redundante, en otra parte del RDBMS.

Como las bases de datos permiten el acceso concurrente a los datos. La manera en la que el software de base de datos evita que las operaciones que están haciendo los usuarios modifiquen datos que son procesados por otro usuario es dividiendo el trabajo que cada usuario realiza en unidades lógicas de trabajo llamadas **TRANSACCIONES**. La parte básica de la recuperación es la recuperación de transacciones.

Una transacción es definida por el estándar SQL ANSI / ISO (el cual es observado por el Oracle7 Server y todas las versiones posteriores) como una o más sentencias DML ejecutadas por un usuario, la cual inicia con la ejecución de la primer sentencia y termina hasta que el usuario confirma o deshace los cambios hechos. [BOCH93]

Dentro de una transacción es muy posible ver que se hacen cambios a más de un registro en una tabla. Las reglas de integridad y relaciones vigilan si es necesario el actualizar mas datos en todas las tablas que estén relacionadas. Un cambio no siempre es una operación sobre una tabla, en ocasiones es una secuencia de operaciones que permiten mantener consistente la información de la base de datos.

Por estas secuencias de operaciones no es permitido que una de las citadas operaciones se realice y la otra no porque eso podría dejar la base de datos en un estado inconsistente. Lo ideal sería que se pudiera garantizar que todas las operaciones pudieran realizarse, sin embargo, es imposible ya que siempre existe la posibilidad de una falla. Por ejemplo, una caída del sistema puede ocurrir mientras se ejecutan las operaciones de una transacción, o un error en alguna operación aritmética en parte de las operaciones. Un RDBMS debe proveer la manera de garantizar que, si una transacción ejecuta algunos cambios y una falla ocurre antes que la transacción sea terminada, los cambios hechos serán deshechos. De esta manera las transacciones ejecutan todos los cambios o son canceladas.

El componente básico que permite manejar las operaciones de una transacción de manera granular, a pesar de ser en la mayoría de las ocasiones secuencias de operaciones, es conocido como el Administrador de Transacciones, sus operaciones básicas son:

COMMIT TRANSACTION; esta operación (conocida como **COMMIT**), envía una señal al administrador de transacciones de fin-de-transacción exitosa, equivale a que la unidad lógica de trabajo fue completada y la base de datos se encuentra en un estado consistente, en este momento todos los cambios realizados por la unidad de trabajo pueden hacerse permanentes.

ROLLBACK TRANSACTION; esta operación (conocida como **ROLLBACK**), envía la señal de fin-de-transacción no exitosa, indicando al administrador de transacciones que algo salió mal, la base de datos puede estar en un estado inconsistente y todos los cambios hechos por la unidad de trabajo deben ser cancelados.

Para poder deshacer los cambios hechos por una transacción el sistema mantiene un registro o bitácora en disco del detalle de todos los cambios hechos por las operaciones de la transacción. De esta manera, si se requiere deshacer un cambio hecho, el sistema usa la entrada hecha en registro de cambios para restaurar el objeto actualizado a su valor previo.

Una transacción inicia con la ejecución de la sentencia **BEGIN TRANSACTION**, (que se definió como la primer sentencia DML ejecutada por el usuario), y termina con la ejecución de cualquier operación **COMMIT** o **ROLLBACK**.

El COMMIT, indicado por el usuario, establece un punto de sincronización, que marca el final de la transacción y alcanzando un momento en el que la base de datos esta en un estado consistente. De manera opuesta, el ROLLBACK, que puede ser indicado por el usuario o automático en caso de presentarse un problema, regresará la base de datos a un estado previo igual al que la base de datos tenia cuando se inicio la transacción, que es el punto de sincronización anterior.

Es importante hacer notar que un COMMIT o ROLLBACK, marcan el fin de una transacción no del programa. Por lo general la ejecución de un programa consiste en una secuencia de muchas transacciones ejecutándose una después de la otra.

Las transacciones tienen en resumen cuatro importantes propiedades, atómicas, consistentes, independientes y durables (referidas de manera coloquial como propiedades 'ACID') [DATE95]

- **ATÓMICAS.** Las transacciones son atómicas, todo o nada.
- **CONSISTENTES.** Los cambios realizados preservan la consistencia, toda transacción lleva la base de datos de un estado consistente a otro.
- **INDEPENDIENTES.** Las transacciones están aisladas entre ellas. Esto es, a pesar que existen muchas transacciones que se ejecutan de manera concurrente, todos los cambios de una transacción no son visibles para el resto de transacciones, hasta que esta hace COMMIT.
- **DURABLES.** Una vez que la transacción hace permanentes los cambios a través de un COMMIT, los cambios seguirán aunque el sistema caiga momentos después.

1.2.2 Protección de la base de datos.

Además de la recuperación de transacciones ésta la recuperación de la base de datos, no sólo de fallas locales, como desbordamientos de memoria de una transacción particular; también están las fallas generales como lo son las caídas de energía.

Una falla local, por definición solamente afecta la transacción donde el problema ocurrió, de manera opuesta, la falla global afecta todas las transacciones activas al tiempo de la falla. Las fallas globales pueden caer en dos categorías:

- **Fallas de sistema** (una falla de energía), afectan todas las transacciones activas pero no causan daño físico a la base de datos. Una falla del sistema es generalmente recuperable.
- **Fallas físicas** (que pueden ser fallas de discos o CPUs), causan daños a la base de datos, o a una porción de ella, y afectan al menos todas las transacciones que usaban la parte afectada. La recuperación de este tipo de fallas requiere el restaurar los datos de una copia no dañada que se tenga para recuperar lo perdido.

El punto crítico de las fallas del sistema es que el contenido de la memoria principal se pierde (de manera particular, las secciones donde reside la base de datos) El estado de cualquier transacción que estaba ejecutándose y se encontraba en memoria es desconocido, por lo que debe ser cancelada cuando el sistema sea iniciado nuevamente. Sin embargo, aquellas transacciones que fueron completadas de manera exitosa antes de la caída, pero que no lograron hacer los cambios registrados en las secciones en memoria a los archivos físicos, deben ser también restauradas.

El mecanismo para deshacer y rehacer dichos cambios es conocido como **CHECKPOINT** [DATE95], en determinados intervalos de tiempo, particularmente cuando una cierta cantidad de entradas son hechas al registro o bitácora de cambios el sistema realiza un checkpoint. Esto quiere decir:

- (a) Escribir físicamente los contenidos de los cambios hechos en memoria en los archivos físicos de la base de datos; y
- (b) Actualizar un registro especial del checkpoint que contiene una lista de todas las transacciones que se encontraban activas al momento que el checkpoint fue realizado.

Con esta información el sistema es capaz de reconstruir y hacer permanentes los cambios que no le fue posible completar por causa de la falla del sistema. Al iniciar nuevamente, el RDBMS cuenta con un par de listas con información de transacciones que hay que deshacer y aquellas que hay que volver a realizar.

El RDBMS recorre el archivo con el registro de cambios hechos de atrás hacia adelante, deshaciendo los cambios que se registraron en la lista de transacciones no confirmadas; después trabaja de adelante hacia atrás para completar los cambios que debieron ser permanentes que se encuentran registrados en la lista de lo que ya se había confirmado por los usuarios. Esto dejará la base de datos nuevamente en un estado consistente y listo para procesar nuevas transacciones.

De manera similar una falla física, como lo es la falla de un disco o de uno de sus controladores, deja una parte de la base de datos dañada o destruida. La recuperación de dichas fallas incluye restaurar la base de datos de una copia de respaldo que se tenga, y entonces usar los archivos de registro de cambios para rehacer todas las transacciones que se completaron desde que el respaldo fue generado.

1.2.3 Proceso transaccional. Transacciones concurrentes

La concurrencia entre transacciones es garantizada usando diferentes mecanismos que aseguran la correcta ejecución de las transacciones. Estos mecanismos varían dependiendo si es una base de datos centralizada o distribuida, entre los mecanismos para los ambientes distribuidos destaca el **TWO-PHASE COMMIT**, también conocido como *protocolo de compromiso de dos fases*.

Para seguir con la revisión del control de concurrencia de transacciones es necesario definir la siguiente notación: [DATE95]

- T_i = cualquier transacción, i es numero de transacción.
- R_i = operación de lectura perteneciente a la transacción i .
- W_i = operación de escritura de la transacción i .
- x, y, z = objetos de la base de datos que son manipulados por la transacción.
- $R_i(x)$ = operación de lectura en el objeto x .
- $W_i(x)$ = operación de escritura en x .
- $WL_i(x)$ = la transacción i solicitando un candado de escritura sobre el objeto x .
- $RL_i(x)$ = la transacción i solicitando un candado de lectura sobre x .

Seguindo la notación anterior, una transacción esta compuesta por la **UNIÓN** de las operaciones de lectura y escritura:

$$\begin{aligned}T1 &= R1(x), W1(x), R1(y), W1(z) \\T2 &= R2(x), R2(y), R2(z), W2(x)\end{aligned}$$

Una estrategia de ejecución para ambas transacciones tendr a que incluir las operaciones de T1 y T2, esta es llamado serial:

$$\begin{aligned}E &= \{ R1(x), W1(x), R1(y), W1(z), R2(x), R2(y), R2(z), W2(x) \} \\SERIAL(E) &= T1, T2\end{aligned}$$

Es una **ESTRATEGIA SERIAL** ya que las transacciones se ejecutaran una despu es de la otra, las operaciones de T1 se completan antes que las de T2 se inicien, de esta manera no existe ninguna posibilidad de conflicto entre las operaciones de la transacci n. Sin embargo, este tipo de estrategias tiene un pobre desempe o de ejecuci n si el RDBMS recibe m ultiples transacciones a la vez. [DATE95]

Los **MECANISMOS DE CONTROL DE CONCURRENCIA** tratan de ejecutar paralelamente operaciones de escritura y lectura de diferentes transacciones, esta ejecuci n generar  resultados similares que haga parecer que su ejecuci n se hace siguiendo una estrategia serial.

Las operaciones concurrentes que pueden causar conflictos de ejecuci n son:

1. Aquellas que operan sobre el mismo objeto
2. Una de ellas, o ambas, es son operaciones de escritura
3. Cada una de las operaciones pertenece a diferentes transacciones

Del ejemplo con la estrategia serial antes definido, tenemos:

$$E = \{ R1(x), W1(x), R1(y), W1(z), R2(x), R2(y), R2(z), W2(x) \}$$

Las operaciones con conflicto son:

$$\begin{aligned}R1(x), W2(x) \\W1(x), R2(x) \\W1(x), W2(x) \\W1(z), R2(z)\end{aligned}$$

Si asumimos que las transacciones realizan las siguientes operaciones:

Transacci n 1

$$\begin{aligned}R1(x); \\x = x + 5; \\W1(x); \\R1(y); \\z = x + y;\end{aligned}$$

Transacci n 2

$$\begin{aligned}R2(x); \\R2(y); \\R2(z); \\x = x + y + z; \\W2(x);\end{aligned}$$

$W1(z);$

Como ya se reviso, en el caso de la estrategia serial, las operaciones conflictivas no son problema ya que la transacción T2 se ejecutara antes de la T1. El objetivo de los mecanismos de control de concurrencia, por el contrario, es generar estrategias de ejecución que sean equivalentes a las estrategias seriales. Una estrategia que explota la concurrencia de las transacciones sería:

$$E = \{R1(x), W1(x), R1(y), R2(x), W1(z), R2(y), R2(z), W2(x)\}$$

Las operaciones que tienen el asterisco son aquellas que pueden ejecutarse paralelamente.

$$\begin{aligned} T1 &= R1(x), W1(x), R1(y)*, W1(z)* \\ T2 &= R2(x)*, R2(y)*, R2(z), W2(x) \end{aligned}$$

La manera de determinar si el orden de ejecución de las operaciones que pueden tener conflictos es correcto es verificando la siguiente formula:

$$Rm(x) < Wn(x)$$

Es decir que la operación de lectura de la transacción m debe preceder la operación de escritura de la transacción n en el mismo objeto x . En el ejemplo analizado, $R1(x) < W2(x)$; Además $W1(x)$ precede a $W2(x)$, y $W1(z)$ precede a $R2(z)$ Es posible realizar varias operaciones de manera concurrente al ejecutar $R1(y)$ y $R2(x)$, además de $W1(z)$ y $R2(y)$ El ejemplo anterior sirvió para probar que el orden de las operaciones que podrían causar conflicto en la nueva estrategia siguen el orden de las mismas operaciones en una estrategia serial. Ya que esta condición es cumplida, la estrategia y no-serial son equivalente; Ambas estrategias producirán los mismos resultados en la base de datos.

Por definición el Oracle7 Server permite la ejecución concurrente de transacciones para modificar, insertar o borrar registros en la misma tabla y sobre los mismos registros. Los cambios hechos por una transacción no son vistos por otra transacción concurrente hasta que la transacción que está cambiando los datos hace un COMMIT TRANSACTION. [BOCH93]

En Oracle, si una transacción A intenta borrar o actualizar un registro que a sido bloqueado por una transacción B (por una sentencia DML) entonces la sentencia DML de la transacción A es bloqueada hasta que B hace COMMIT o ROLLBACK a la transacción. En este momento A puede ver los cambios hechos por B y puede proseguir con su transacción.

El método de control de concurrencia del Oracle7 Server (mantenido con mínimos cambios en versiones Oracle8, Oracle8i y Oracle9i) se basa en diferentes niveles de bloqueo o **CANDADOS** que prevén la interacción destructiva entre usuarios. El Oracle7 Server automáticamente bloquea un recurso a favor de una transacción para evitar que otras lo modifiquen al mismo tiempo. El candado se libera también automáticamente cuando la transacción no necesita mas el recurso [BOCH93]

Los candados que existen son de diferentes tipos dependiendo del recurso que se va a modificar y de la operación. El estándar SQL92 de ANSI / ISO define tres posibles tipos de interacciones con transacciones concurrentes en las bases de datos. Los tipos de candados que existen en el Oracle7 Server y cumplen con el estándar son:

- **Candados DML (candados de datos)**
Protegen exclusivamente datos. Bloqueo a tablas (toda la tabla, por registro)
- **Candados DDL (candados del diccionario de datos)**
Protegen la estructura del objeto particularmente la definición de las tablas.
- **Candados internos.**
Candados automáticos que protegen las estructuras internas de la base de datos. Son controlados en su totalidad por el RDBMS.

Dependiendo de las operaciones realizadas por una transacción, existen dos tipos de bloqueo o candado DML: los candados de lectura y los candados de escritura. Algunos candados son compatibles entre ellos, es decir que más de una transacción puede colocar un candado en el mismo objeto.

Un candado de lectura es compatible con otro candado de lectura. Si la transacción T1 solicita un candado de lectura en el objeto X, y la transacción T2 ya tiene un candado del mismo tipo en X, entonces T1 obtiene el candado de cualquier forma. Por el contrario, si cualquiera de las transacciones solicita un candado de escritura y una de ellas ya tiene el candado de escritura otorgado, entonces el candado es negado a la última transacción en solicitarlo hasta que la primera libere el objeto.

Este modelo de concurrencia es apropiado para casi todas las aplicaciones. Sin embargo, en ocasiones es necesario permitir que las transacciones sean seriales. El Oracle7 Server puede forzar que las transacciones se ejecuten de manera serial y no concurrentemente como ocurre normalmente. Los cambios que se hagan a la base de datos podrán completarse programando las transacciones para ejecutarse una después de la otra con un orden definido.

1.2.4 Transacciones distribuidas. Protocolos de compromiso de dos y tres fases.

Una **TRANSACCIÓN DISTRIBUIDA** cambiara información en mas de un RDBMS que puede o no residir en el mismo servidor. El servidor de base de datos que participa en un ambiente distribuido es conocido como **NODO**.

En el caso de transacciones distribuidas, existe el termino **ORDEN TOTAL (OT)**, que es simplemente el listado de orden de todas las transacciones distribuidas ejecutadas en los diferentes nodos. [DATE95]

Si por ejemplo tenemos cuatro transacciones distribuidas y cada una de ellas tiene operaciones que se ejecutan en tres de los nodos, el orden total de estas transacciones es:

$$OT = T1, T2, T3, T4$$

Cada uno de los nodos tiene una estrategia que ejecuta operaciones concurrentes de las transacciones T1, T2, T3 y T4. Haremos referencia de estas estrategias como S1, S2 y S3. De las definiciones anteriores, tenemos que para que una estrategia concurrente sea correcta, esta debe

ser equivalente a una estrategia en serie. En nuestro caso, definimos tres estrategias en serie $S1'$, $S2'$, $S3'$. Y la condición siguiente debe ser verdadera.

$S1' = \text{SERIE}(S1)$
 $S2' = \text{SERIE}(S2)$
 $S3' = \text{SERIE}(S3)$

La manera de asegurar la ejecución en serie de transacciones distribuidas es dando el mismo orden a las operaciones conflictivas en las estrategias $S1'$, $S2'$ y $S3'$ que las que fueron definidas anteriormente en el orden total OT. Esto significa, si la operación $OP1(x)$ precede a $OP2(x)$ en OT, entonces este mismo orden debe reflejarse en las estrategias en serie en los nodos que ejecutan las transacciones T1 y T2.

El protocolo de dos fases **TWO-PHASE COMMIT**, es una estrategia de coordinación para completar o abortar transacciones distribuidas entre múltiples nodos. Los participantes en una transacción distribuida se clasifican como "participantes" y "coordinador." La función de coordinador es la de coleccionar los estados de terminación de todas las transacciones en los nodos remotos participantes. [BOBA93]

La Figura 1.2 enseguida, muestra el diagrama de transiciones del protocolo de compromiso de dos fases, el nodo coordinador entra en un estado de "espera" mientras recibe y analiza los resultados de los nodos participantes. Una vez recibidos, el coordinador pasa a una fase de terminación global o commit global de la transacción. A su vez, los participantes entran a un estado de "preparado" después de haber informado al coordinador su decisión. En esta etapa, los participantes esperan la decisión final del coordinador para la transacción. Al recibirla, los participantes pasan a la fase de commit local o terminación local dependiendo del voto recibido.

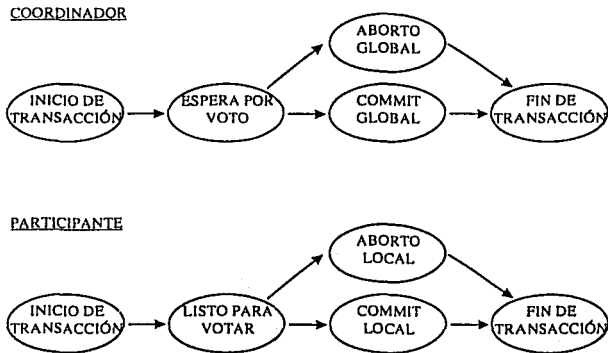


Figura 1.2 Diagrama de transiciones del two-phase commit.

TESIS CON
FALLA DE ORIGEN

Los protocolos de two-phase commit y three-phase commit funcionan a través de una arquitectura de comunicación entre los nodos para determinar cual es el resultado de una transacción. Las arquitecturas de comunicación son:

1. Centralizada
2. Lineal
3. Distribuida

Arquitectura centralizada.

Involucra la comunicación entre dos agentes llamados coordinador y participante. El papel del coordinador es coleccionar las condiciones de terminación de las transacciones que se ejecutaron en cada nodo. Cada participante vota entre hacer commit a la transacción o abortarla. Una vez recibidas todas las decisiones de los participantes, el coordinador actualiza su propio archivo de log (registro de cambios a la base de datos) y entonces informa a todos los participantes que pueden hacer commit a su parte de la transacción distribuida. Si al menos uno de los nodos decide abortarla, todos los nodos deben hacer lo mismo.

La arquitectura centralizada dará problemas si el nodo coordinador falla antes de emitir su voto de confirmar o deshacer la transacción. Si esto pasa los participantes se quedan en estado de espera. La manera de aliviar este problema es utilizando la estrategia de tres fases llamado **THREE-PHASE COMMIT**.

Por su funcionamiento el protocolo de two-phase commit es considerado un protocolo de bloqueo ya que si el coordinador falla, los participantes quedan en un estado de espera, mientras que el protocolo de three-phase commit es catalogado como de no bloqueo.

Arquitectura lineal.

No tiene un coordinador centralizado. Los participantes pasan sus decisiones a los nodos adyacentes en un orden predeterminado. Conforme cada nodo toma la decisión de abortar o confirma la transacción, este lo comunica al siguiente hasta que el último nodo involucrado en la transacción distribuida es alcanzado. Es entonces cuando el último nodo manda su decisión siguiendo la cadena que se formo para alcanzarlo. Esta última decisión es la que el resto de los nodos toman.

Si alguno de los nodos en la parte media de la cadena aborta la transacción, los siguientes nodos cambian entonces su decisión de commit y abortan la transacción. El último nodo al votar por abortar la transacción manda su voto de regreso y todos los nodos que habfan decidido hacer commit originalmente cambian su decisión y la abortan hasta alcanzar el primer nodo.

Arquitectura distribuida.

Permite a los nodos en el ambiente distribuido comunicarse entre ellos. Aquí cada nodo recibe las decisiones del resto de los nodos participantes en la transacción. Cada nodo toma su decisión después de analizar los resultados de los otros nodos. Con esta estrategia cada nodo actúa como un coordinador.

La **Figura 1.3** muestra a continuación el funcionamiento del protocolo two-phase commit con una arquitectura de comunicación lineal en una transacción distribuida en la que participan 3 nodos.

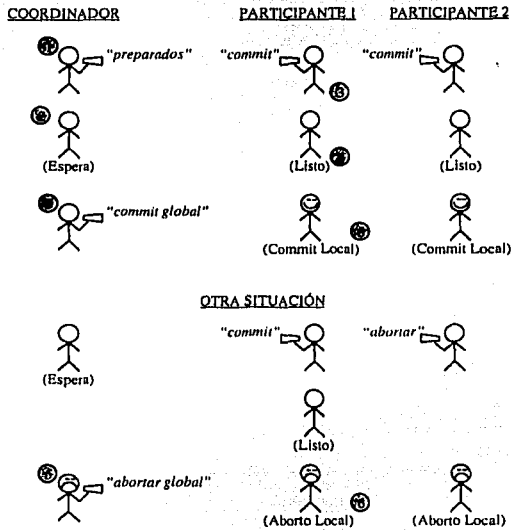


Figura 1.3 Two-phase commit con comunicación lineal.

El coordinador entra en un estado de INICIO al mandar el mensaje de PREPARADOS a los participantes de la transacción. El coordinador entra inmediatamente en un estado de ESPERA (*Pasos 1 y 2*)

Los participantes al recibir el mensaje del coordinador, envían una respuesta en forma de "aborto local" o "commit local" (*paso 3*) Si el nodo local aborta pasa al estado de terminación local y espera por la respuesta del coordinador. Si el participante decide hacer commit, entra en un estado de LISTO (*paso 4*) Si ambos participantes votaron "commit," el coordinador deja el estado de ESPERA y transmite el comando de COMMIT GLOBAL (*paso 5*) Al recibir el mensaje, los participantes dejan el estado de LISTO y ambos nodos concluyen la transacción realizando commit localmente (*paso 6*)

De igual manera, si uno de los participantes voto por abortar la transacción, el coordinador revisa los votos y manda la instrucción de ABORTO GLOBAL a los participantes (*pasos 5 y 6*)

TESIS CON
FALLA DE ORIGEN

El algoritmo funciona bien si todos los nodos permanecen comunicados durante el tiempo de vida de la transacción y no existe alguna falla en la red de comunicación. Si se presentan problemas en uno de los nodos, los restantes podrían quedar esperando en un estado de ESPERA o LISTO, por una respuesta del coordinador que obviamente nunca llegara, al menos hasta que el problema de comunicación sea detectado y resuelto.

El diagrama de transición del protocolo three-phase commit o de tres fases, es idéntico al two-phase commit con la excepción de la inclusión de un estado llamado PRE_COMMIT y la transición de este estado hacia el estado de aborto de la transacción.

La Figura 1.4 enseguida muestra el diagrama de transición para el protocolo three-phase commit cuya lógica es la misma que el protocolo de two-phase commit, cambiando solamente en que los participantes entran en el estado de PRE_COMMIT y esperan por instrucciones del coordinador. En este estado, los participantes esperan por un determinado período de tiempo. Si el intervalo de tiempo expira, el participante asume que el coordinador falló cambiando entonces al estado de ABORTO_LOCAL terminando la transacción. Al momento de hacer recuperación, el coordinador se da cuenta que los participantes abortaron la transacción y actualiza sus archivos de log para reflejarlo. Por el contrario, si el coordinador no falla, manda la decisión de COMMIT_GLOBAL a los participantes que dejan el estado de PRE_COMMIT para terminar con la transacción.

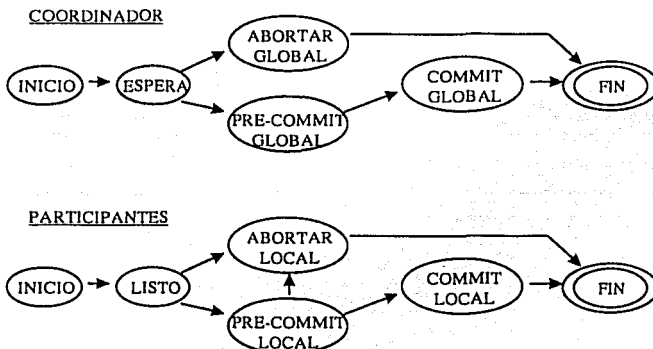


Figura 1.4 Diagrama de transición de protocolo three-phase commit

Para evitar problemas por posibles fallas del coordinador o participantes durante la transacción es incluido un periodo de espera para cada estado donde tanto el coordinador como los participantes son vulnerables a las citadas fallas. Con esta inclusión, si no es recibida una respuesta en un cierto período de tiempo, los participantes o el coordinador pueden asumir que el otro nodo tuvo problemas y puede comenzar a abortar la transacción localmente.

El coordinador puede implementar un medidor de tiempo en el estado de ESPERA. Si el período de tiempo expira y todos los participantes no enviaron sus votos, el coordinador decide terminar con la transacción mandando la instrucción de ABORTO GLOBAL a todos los participantes.

Enseguida, el coordinador puede colocar otro medidor de tiempo ya sea en el estado de COMMIT GLOBAL o en el ABORTO GLOBAL. Si todos los participantes no recibieron el mensaje después que el coordinador espero por un lapso de tiempo, el coordinador sigue enviando el mensaje a los nodos con problemas hasta que reciba mensaje de ellos.

De igual manera, los estado donde se podrían colocar medidores de tiempo en los participantes. Los estados son INICIO y LISTO. Si el participante no recibe respuesta en el estado de LISTO después de esperar un período de tiempo, aborta la transacción localmente y termina. La expiración del medidor de tiempo en este estado significa que el coordinador fallo. Cuando el coordinador se recupera, estará esperando por la respuesta del participante. Como el nodo participante ya termina la transacción la respuesta nunca llega. El coordinador expira su medidor de tiempo y enviara la instrucción de ABORTO GLOBAL.

Si el participante coloca el medidor en el estado de LISTO, se encontrara en un dilema. Para pasar a este estado, antes tuvo que haber votado por hacer commit a la transacción y necesita la decisión que tome el coordinador (que probablemente no esté funcionando) para moverse al siguiente estado. El participante no puede decidir por si mismo y no tiene manera de saber la decisión del resto de los participantes. El protocolo de tres fases toma en cuenta esta consideración que se revisará enseguida.

En los participantes el estado de PRE-COMMIT tiene conexión con el estado de ABORTAR LOCAL. Si el participante agota el tiempo de espera en el estado de PRE-COMMIT LOCAL puede proceder a abortar localmente la transacción, esto a pesar de haber votado por hacer commit a su parte de la transacción distribuida.

Si recordamos, en el protocolo de dos fases, una vez que los participantes deciden hacer commit a la transacción, estos no podían cambiar de opinión. Por lo tanto, el protocolo de tres fases no es un protocolo de bloqueo ya que permite a los participantes de la transacción distribuida una ruta de escape del estado PRE-COMMIT LOCAL en caso de que el coordinador tenga alguna falla o exista algún problema de comunicación.

El Oracle7 Server (al igual que versiones Oracle8, Oracle8i y Oracle9i) maneja las transacciones distribuidas siguiendo el protocolo two-phase commit [DUFE95]. Sin embargo, el Oracle Server implementa algunas peculiaridades que son el **Árbol de sesiones**, **coordinador local y global**, y el **Nodo de commit**.

Un **árbol de sesiones** es un modelo jerárquico de una transacción distribuida donde se representan las funciones de los nodos participantes. Un nodo puede ser:

- **Coordinador local.** Es un nodo en la transacción distribuida que modifica datos en otros nodos con los que está directamente relacionado. Además se encarga de pasar el estatus de la transacción de los estos nodos al coordinador global donde se originó la transacción.

TESIS CON
FALLA DE ORIGEN

- **Coordinador global.** Nodo donde la transacción distribuida se inicia. El nodo se convierte en la raíz del árbol de sesiones. Es propiamente el coordinador en el two-phase commit. Comunica al nodo de commit el voto de todos los nodos y / o coordinadores locales para hacer commit o rollback.
- **Nodo de commit.** Inicia la fase de commit o rollback de la transacción que le comunica el coordinador global (que podría tratarse del mismo nodo coordinador global) El nodo de commit debe ser por definición el que tiene la información mas crítica dentro de la base de datos distribuida y se define con un valor numérico asignado por el administrador del ambiente. Si se tienen tres nodos con un valor de:

Nodo1 - *commit_point_strength* = 75
Nodo2 - *commit_point_strength* = 100
Nodo3 - *commit_point_strength* = 50

El nodo de commit sería siempre el Nodo2, por tener los datos más críticos debe ser el nodo menos propenso a fallas.

Si se identifica como un nodo de commit nunca estará en estado de preparado para votar como el resto de los nodos que tendrán que retener candados sobre los datos afectados en caso de presentarse una falla en uno de los nodos. Hace commit a su parte de la transacción antes que el resto de los nodos.

1.2.5. Transacciones distribuidas en duda.

Por el protocolo two-phase commit una transacción distribuida puede estar en duda para Oracle cuando una falla ocurre en alguna de las fases por [DUFE95]:

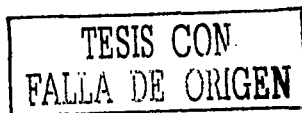
1. Uno de los nodos corriendo una base de datos Oracle falla.
2. Error de conexión entre los nodos participantes en la transacción distribuida.
3. Algún error de software ocurre.

El Oracle Server cuenta con un proceso automático, llamado **RECO** que resuelve este tipo de fallas. Mientras la transacción es resuelta por el proceso los datos son bloqueados para lectura o escritura porque no hay manera de determinar que versión de los datos (antes o después de ser modificados) debe ser desplegada en caso de que se realice una lectura en otra transacción.

En la mayoría de los casos estas transacciones son resueltas automáticamente. Si ocurre durante la fase de "Preparados":

- a. Un usuario inicia una transacción distribuida en el *Nodo A*.
- b. El *Nodo A* se convierte en el coordinador global y solicita el voto de todos los nodos involucrados (menos del nodo de commit)
- c. El *Nodo B* tiene una falla antes de enviar su voto al nodo coordinador.
- d. El proceso **RECO** hace rollback a la transacción en el *Nodo A* y en *Nodo B* cuando la base de datos es reestablecida.

En una transacción con dos nodos, si la falla ocurre en la fase de "commit global":



- e. Se inicia la transacción en el *Nodo A* y Oracle lo define como coordinador global
- f. El *Nodo A* solicita el voto de los nodos participantes
- g. El *Nodo B* comunica voto de commit al *Nodo A*, las transacciones hace commit en el nodo de commit.
- h. El nodo de commit envía el voto de commit al *Nodo B*.
- i. El *Nodo B* recibe el voto de commit pero no puede responder por una falla de conexión con el *Nodo A*.
- j. La transacción es completada en el *Nodo B* por el proceso RECO cuando la comunicación es restablecida con el *Nodo A*.

En ocasiones es necesario resolver estas transacciones manualmente pero solamente debe hacerse cuando los candados colocados por el proceso RECO no permiten usar datos críticos de la base de datos que impidan su operación. De igual manera si la falla con algún nodo remoto, la comunicación o la base de datos misma no puede ser corregida rápidamente. Para esto se debe:

- Identificar que transacción es la que sigue como "en duda"
- Consultar el diccionario de datos para determinar si las bases de datos involucradas hicieron commit o no.
- Forzar como DBA un commit o rollback de la transacción.

1.3 Características y terminología de las bases de datos distribuidas.

1.3.1 Definición de base de datos distribuida.

En un sistema de base de datos distribuido los datos se almacenan en varias computadoras o servidores. Donde cada una de ellas se comunican a través de diversos medios. Cada servidor, también conocido como LOCALIDAD o NODO, puede participar en la ejecución de transacciones teniendo acceso a datos de varios otros nodos.

Las bases de datos distribuidas se rigen por el principio de que cualquier aplicación debe funcionar transparentemente sin importar que sus datos se encuentren dispersos en bases de datos diferentes, manejadas por sistemas RDBMS distintos, corriendo en diferentes nodos, sobre diferentes sistemas operativos, y conectados entre ellos por diferentes medios de comunicación; donde transparentemente significa que la aplicación funciona como si los datos estuvieran en una sola base de datos manejada por un sólo RDBMS en un único servidor. [DUFE95]

De esta manera la base de datos distribuida es un objeto lógico, cuya información se encuentra físicamente repartidas en diferentes bases de datos reales. Cada nodo es un sistema de base de datos por sí mismo, con usuarios propios, corriendo un sistema manejador de base de datos local que puede ser relacional o no.

El principio fundamental de una base de datos distribuida es el de aparecer ante el usuario como un sistema no distribuido. [DATE95]

Cualquier operación se podrá realizar sobre los datos como si la base misma no formara parte de un sistema distribuido. La propiedad de trabajar en conjunto con otros nodos es una extensión del

TESIS CON
FALLA DE ORIGEN

manejador de base de datos que permite ahora nombrarlo Sistema Manejador de Bases de Datos Distribuidas (**DDBMS** por sus siglas en Inglés)

La **Figura 1.5**, enseguida ilustra los componentes de una base de datos distribuida. Esta formada por una colección de nodos, conectados entre sí a través de una red de comunicaciones, donde:

1. Cada nodo es un sistema de base de datos por sí mismo, pero,
2. Estos nodos han acordado trabajar juntos para que cualquier usuario en cualquier nodo pueda tener acceso a los datos en cualquier punto de la red como si estuviesen almacenados en su propio nodo.

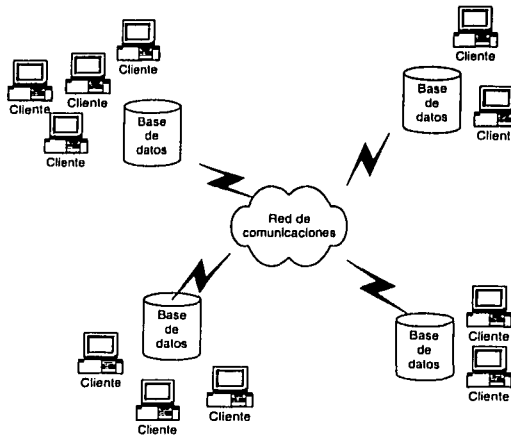


Figura 1.5 Componentes de una base de datos distribuida

Las bases de datos distribuidas son necesarias porque las organizaciones en la actualidad son distribuidas en forma de divisiones, departamentos, grupos de trabajo. Esta distribución además de funcional puede ser física en la forma de plantas de producción, laboratorios, almacenes, etc.

La información perteneciente a cada una de estas áreas se encuentra cerca de la localidad donde es necesaria, generada y es relevante para la operación de cada una de ellas. Un sistema distribuido refleja a través de la base de datos la estructura de la organización: Datos locales son mantenidos de manera local, mientras que los mismos datos pueden ser usados remotamente cuando son necesarios.

TESIS CON
FALLA DE ORIGEN

El mantener los datos de manera distribuida combina la eficiencia de procesamiento (ya que los datos están almacenados cerca del punto donde son frecuentemente usados), con alta disponibilidad (es posible disponer de datos pertenecientes a otros nodos dentro de la organización)

1.3.2 Las 12 reglas de las bases de datos distribuidas.

En 1987, C. J. Date definió las **12 reglas de un sistema manejador de bases de datos distribuidas**. Estos buscan asegurar que los problemas de distribución sean solucionados internamente. Enseguida se revisan las 12 reglas propuestas por Date, comentando si el Oracle7 Server cumple o no con cada una de ellas.

1. Autonomía Local

Los nodos en un ambiente distribuido deben ser autónomos, independientes uno de otro.

Para el Oracle7 Server (y todas las versiones posteriores) todos los datos locales y las operaciones hechas sobre estos no dependen de la disponibilidad de otro nodo para seguir funcionando.

2. No-dependencia de un nodo central

Una base de datos distribuida no deberá depender de uno nodo central porque esto crearía un punto de falla para todo el ambiente o ser un cuello de botella afectando el desempeño.

Las RDBMS Oracle en un ambiente distribuido son tratados como iguales, cada nodo con una base de datos Oracle tiene un diccionario de datos propio y en caso de falla de otro nodo siguen en funcionamiento.

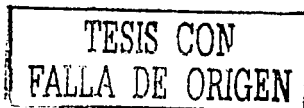
3. Operación continua

Una cualidad deseada en toda base de datos distribuida es que no debería tener bajas y operar de manera continua. La confiabilidad y disponibilidad deben garantizarse en un ambiente distribuido.

Para los nodos corriendo Oracle como RDBMS es transparente y no afecta su funcionamiento el agregar mas nodos al ambiente distribuido o que otros queden inaccesibles por tareas de mantenimiento o administración..

4. Independencia de ubicación

La idea básica de independencia local (también conocida como transparencia de ubicación) es simple: Los usuarios no tienen necesidad de saber donde los datos están almacenados físicamente, pero estos deben comportarse como si se encontraran en su propio nodo local.



En las bases de datos Oracle en ambientes distribuidos, los usuarios pueden recuperar datos de cualquier nodo independientemente del nodo en el que se encuentre.

5. Independencia de fragmentación

La **FRAGMENTACIÓN** se define como la habilidad del RDBMS para dividir, manejar y almacenar fragmentos de una o más tablas en diferentes nodos de un ambiente distribuido. La fragmentación puede ser *horizontal* o *vertical*. Esto debe ser transparente para el usuario y la aplicación.

Se llama *fragmentación horizontal*, cuando los fragmentos de las tablas dependen de los valores almacenados en una o varias columnas.

La *fragmentación vertical*, sucede cuando las columnas de las tablas fragmentadas son almacenadas en diferentes nodos.

En bases de datos Oracle ambos tipos de fragmentación pueden ser implementados [PRAT95], la fragmentación horizontal se hace usando LIGAS DE BASE DE DATOS (Database Link)

Una liga de base de datos es una conexión entre dos nodos en un ambiente distribuido que permite al usuario usar datos en el nodo remoto como si fuera una sola base de datos lógica.

Físicamente la liga de base de datos es un apuntador que define una ruta de comunicación de un sentido de un RDBMS Oracle a otro. Para hacer uso de la liga de base de datos el usuario debe estar conectado a la base de datos local donde esta creada la liga.

Si se quisiera crear una liga de base de datos del nodo A hacia el nodo B, el DBA del nodo A lo hace con la siguiente instrucción:

```
Create public database link Liga_B connect to VENTAS identified by NORTE
Using 'B';
```

Esto crea la liga de base de datos Liga_B en el nodo A y los usuarios de este nodo consultan y actualizan los objetos en el esquema CLAVE (que es un usuario en el nodo B que tiene como clave de acceso NORTE) de la base remota B, pero los usuarios de B no podrán ver los objetos en el nodo A.

Si el esquema VENTAS en el nodo B tiene una tabla llamada clientes, el DBA en A puede crear un SINÓNIMO de la manera siguiente:

```
Create public synonym CLIENTES_NODO_B for CLIENTES@Liga_B;
```

Los usuarios del nodo A, perciben el sinónimo CLIENTES_NODO_B como una tabla local sin saber que están leyendo y cambiando datos de la tabla remota CLIENTES.

La fragmentación vertical en Oracle restringe la habilidad de modificar o seleccionar solamente determinadas columnas de los registros de una tabla. La propiedad de cada una de estas columnas es de diferentes nodos. Esta característica se implementa en forma de GRUPOS DE COLUMNAS, que ligan una colección de columnas en una tabla a una columna lógica única.

Un grupo de columnas puede estar formado por una, varias o todas las columnas de una tabla, pero cada columna puede pertenecer solamente a un grupo.

A cada grupo de columnas se le asigna un método de solución de conflictos provistos por Oracle para resolver problemas de unicidad, borrado y actualización de datos en transacciones distribuidas que afectan grupos de columnas.

Cualquier columna que no es asignada a un grupo de columnas automáticamente es asignada a un grupo SHADOW para manejar la solución de conflictos, un grupo SHADOW no es visible para el usuario y no se le puede asignar ningún método de solución de conflictos.

6. Independencia de replicación

La **REPLICACIÓN** en ambientes distribuidos permite mantener copias de los datos que cada nodo necesita para operar. Varios nodos podrían tener copias actualizables de los mismos datos lo que requiere mayores recursos pero es recomendable en ambientes donde los datos se utilizan de manera frecuente desde diferentes nodos.

La independencia de replicación es cuando estas copias a los diferentes nodos se hacen de manera transparente y sin afectar a los usuarios.

En Oracle la replicación de datos es posible a partir de las versiones Oracle7 y posteriores. Existen dos modelos principales de replicación que son:

***Básica.** Usando snapshots que son solamente de lectura. Los cambios se hacen en un nodo primario y replicados en ciertos intervalos de tiempo hacia los nodos*

***Avanzada o simétrica.** Usando snapshots que son actualizables cualquier nodo puede modificar los datos en el snapshot, estos cambios son automáticamente propagados al resto de los nodos del ambiente distribuido.*

*Una **SNAPSHOT** es la forma básica de replicación implementada por Oracle. Y es una copia de una tabla maestra localizada en un nodo remoto, este objeto de base de datos puede consultarse como una tabla normal, se actualizan de manera periódica para reflejar los cambios de la tabla maestra.*

*Si tenemos dos nodos llamados AlmacenNorte y Ventas, donde el nodo primario es Ventas y tiene una tabla de los clientes de la compañía llamada **CLIENTES**; de los cuales para el nodo AlmacenNorte son relevantes en su operación los clientes de la región "Noreste."*

*La replicación de datos ocurre cuando el DBA del nodo AlmacenNorte crea un snapshot de la tabla **CLIENTES** del nodo primario Ventas pero con los clientes que pertenecen a la región "Noreste" de la manera siguiente:*

```
CREATE SNAPSHOT clientes FOR UPDATE AS  
SELECT * FROM clientes@Ventas WHERE region = 'Noreste';
```

TESIS CON
FALLA DE ORIGEN

El snapshot clientes se crea en el diccionario de datos del nodo AlmacenNorte y puede ser usado por los usuarios de ese nodo como cualquier otra tabla.

7. Procesamiento distribuido de consultas.

El desempeño para la ejecución de cualquier consulta debería ser independiente a la localidad desde la cual esta es realizada.

Las consultas realizadas en un nodo con una base de datos Oracle son procesadas en el nodo donde los datos están almacenados.

8. Manejo distribuido de transacciones.

Un sistema distribuido debe cumplir con las propiedades ACID de las transacciones, ya definidas, permitiendo las consultas remotas y las transacciones distribuidas.

Una transacción distribuida en Oracle puede modificar, insertar o borrar datos de múltiples nodos. Para esto usa el mecanismo two-phase commit y bloqueos a nivel registro para garantizar la integridad de los datos en sistemas que además son concurrentes.

9. Independencia de hardware

Una base de datos distribuida debería operar y tener acceso a datos en una gran variedad de plataformas de hardware.

La base de datos Oracle (versiones Oracle7 y posteriores) corren en todas las plataformas de hardware usadas.

10. Independencia de sistemas operativos.

Con la independencia de hardware, la independencia del sistema operativo es consecuencia. Cualquier base de datos distribuida debe correr en diferentes sistemas operativos.

Para Oracle no es necesario un sistema operativo específico ya que corre en una gran variedad de sistemas operativos.

11. Independencia de la red de comunicación

Es la habilidad en un ambiente distribuido para operar sin importar los protocolos de comunicación y/o la topología de red usadas para comunicar los nodos.

*El Oracle7 (y versiones posteriores) cuenta con una capa adicional de software de comunicación llamado SQL*Net [NEVF92] que es responsable de enviar las consultas de datos desde un cliente (o desde otro servidor en caso de ser transacciones distribuidas) hacia el*

*RDBMS Oracle. Una vez procesada en la base de datos, SQL*Net recibe los resultados de la consulta y los para al cliente. Esto es transparente para el usuario.*

*SQL*Net es independiente de protocolo o topología de red.*

12. Independencia del DBMS

Un DDBMS ideal debe contar con la habilidad de manejar y operar con otras bases de datos corriendo en nodos remotos que pueden ser o no relacionales. Estos ambientes son llamados heterogéneos.

Oracle define un ambiente distribuido homogéneo, cuando cada base de datos que es parte del ambiente distribuido corre sobre el RDBMS Oracle, si al menos uno de ellos no lo es para Oracle es llamado un ambiente distribuido heterogéneo. [DUF95]

*Oracle cuenta con una tecnología llamada Open Gateways para manejar datos de otras bases de datos que pueden ser también relacionales, de red o jerárquicas. Para esto usa un servidor llamado Gateway para conectarse al nodo remoto usando SQL*Net.*

En el nodo remoto esta corriendo otra parte del Gateway y su función es emular una base de datos relacional Oracle, esto es transparente para la aplicación y usuarios que perciben al nodo remoto como otro RDBMS Oracle.

En una base de datos distribuida con nodos corriendo RDBMS Oracle, pueden incluirse nodos corriendo diferentes versiones Oracle7, Oracle8, Oracle8i y Oracle9i; sin embargo, la aplicación que usa la base de datos distribuida debe tomar en cuenta la funcionalidad disponible en cada nodo en el que ésta trabajando, de esta manera una aplicación no puede esperar que un nodo corriendo una base de datos Oracle7 entienda la definición de algunos objetos que sólo están presentes en una versión Oracle8i.

1.3.3 Prototipos y primeros DDBMS.

Con las reglas para las bases de datos distribuidas establecidas, por la importancia que esta tecnología fue ganando y al volverse norma para muchas organizaciones, mas grupos se interesaron en desarrollar prototipos que extendieran la habilidad de los RDBMS para trabajar en ambientes distribuidos.

[BURE97] Los prototipos mas destacados y algunas implementaciones comerciales de los sistemas distribuidos son:

- a) **SDD-1**, construido por la división de investigación de "Computer Corporation of America", a finales de los 70's y principios de los 80's. Proveía localización completa de nodos, fragmentación, e independencia de replicación.
- b) **R*** (pronunciado "R Star"), es la versión distribuida del prototipo System R de "IBM research" a principios de los 80's. El prototipo proveía independencia de nodos, pero no soportaba fragmentación o replicación. El control de concurrencia estaba basado en un

TESIS CON
FALLA DE ORIGEN

sistema de bloqueos o candados y la recuperación de transacciones estaba basada en el two-phase commit.

- c) **Distributed Ingres**, versión distribuida del prototipo Ingres de principios de los 80's desarrollado por "University of California Berkeley". Permite la independencia de los nodos, fragmentación de datos y replicación de fragmentos. La propagación de los cambios en los datos funciona actualizando la copia primaria y regresando el control de la transacción, después procesos esclavos realizan el resto de los cambios paralelamente en el resto de los nodos. El control de concurrencia esta basado en un sistema de bloqueos o candados.

Los productos comerciales más conocidos son:

- d) **INGRES/STAR**, desarrollado por "The ASK Group Inc.'s Ingres Division"
- e) **Distributed Database Option of Oracle7**, de "Oracle Corporation" (disponible también en las versiones Oracle8, Oracle8i y Oracle9i)
- f) **Distributed Data Facility of DB2**, de IBM

Se debe mencionar que los sistemas descritos, prototipos y comerciales, son relacionales. El presente trabajo esta enfocado sobre bases de datos distribuidas relacionales, es decir un sistema distribuido homogéneo, con Oracle7 Server como RDBMS; corriendo sobre la misma plataforma aun cuando el sistema operativo no es un impedimento para ninguna de las versiones del RDBMS Oracle.

1.4 Particionamiento de la información.

1.4.1 Alternativas para la distribución de datos.

Por su naturaleza, las bases de datos distribuidas comparten datos y pueden tener procesos repartidos en los diferentes nodos. El objetivo es separar los datos de tal manera que los nodos tengan los mas cerca posible los datos que son mas utilizados. La distribución de la información puede hacerse con almacenamiento **redundante** y **no redundante**.

Cuando se opta por una distribución no redundante se incluyen los modelos fragmentados conocidos como **Fragmentación** o **Particionamiento Horizontal** y **Fragmentación** o **Particionamiento Vertical**.

La distribución con redundancia de datos, que se revisará en el Capítulo IV "*Replicación de datos*", incluye los modelos llamados **Replicación Maestro / Esclavo** y **Replicación con Actualización en todas partes** (también conocida como **Replicación Simétrica**) Este modelo es considerado costoso en términos de almacenamiento y sincronización de las copias pero requerido cuando se tiene acceso frecuente a datos de diferentes nodos.

Modelos Fragmentados.

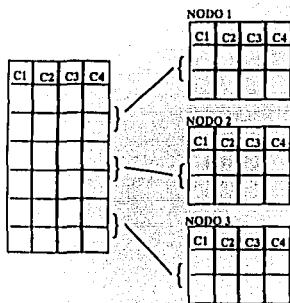
En estos modelos la distribución se hace usando la fragmentación horizontal y / o vertical. No existen datos redundantes en los nodos de la base de datos distribuida con la excepción de las llaves primarias en caso de usarse la fragmentación vertical. En bases de datos relacionales, la **FRAGMENTACIÓN HORIZONTAL** se hace al dividir una tabla basándose en los valores en los registros de la misma.

De igual manera la **FRAGMENTACIÓN VERTICAL** separa la tabla por atributos o columnas, replicando en todos los nodos la columna que es la llave primaria de la tabla.

Un tercer modelo fragmentado es llamado **FRAGMENTACIÓN HÍBRIDA**, que es una combinación de los modelos horizontal y vertical.

La **Figura 1.6** muestra los dos tipos de fragmentación. Cuando son fragmentos horizontales todas las columnas de la tabla son replicadas en tres nodos usando el valor en la columna C4. Por otra parte los fragmentos verticales separan la tabla original (Tabla1) por columnas en los tres diferentes nodos. La columna C1 que es la llave primaria de la tabla se replica a los 3 nodos.

Fragmentación horizontal.



Fragmentación vertical.

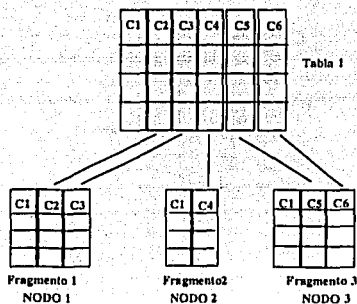


Figura 1.6 Modelos con Fragmentación Horizontal y Vertical

[BURE97] Las características generales del modelo fragmentado son:

- Existe una sola copia de los datos, pero está es fragmentada (particionada) entre los diferentes nodos.
- La propiedad de los datos y la posibilidad de actualizar la fuente de datos distribuida es posible desde todos los nodos.

TESIS CON
 FALLA DE ORIGEN

Existen diferentes metodologías para la fragmentación de los datos [BOBA93]. Para la fragmentación horizontal se tiene el método de los **MÍNIMOS Y COMPLETOS** cuyo objetivo es ayudar a identificar cuales son las condiciones que determinan que parte de los datos será repartida en cada nodo.

Dichas condiciones son llamadas **PREDICADOS**; así un predicado tiene la forma:

Predicado 1 = Donde el Nodo es 'Londres'

Predicado 2 = Donde las ganancias al año son mayores a 1,000,000

A todos los predicados identificados se les aplica la regla de **MÍNIMOS Y COMPLETOS** cuyo resultado será las condiciones que los datos deben cumplir para pertenecer a un nodo específico.

La regla de los **MÍNIMOS Y COMPLETOS** es la siguiente:

*Un **MÍNIMO** es un fragmento de tabla que será usado en el nodo de manera única. De lo contrario, la existencia del citado fragmento es redundante.*

*Un **COMPLETO** es cuando la posibilidad de seleccionar dos o más registros del mismo fragmento, debe existir y ser la misma para cada registro.*

De igual manera, existe una metodología para la formación de los fragmentos verticales [BOBA93], llamada de **MATRIZ DE AFINIDAD DE GRUPO**, que tiene por objetivo el minimizar la cantidad de operaciones de JUNTA (del álgebra relacional) que tenga que realizar cualquier proceso mientras trabaja sobre todos los fragmentos distribuidos de la tabla.

Los pasos que sigue son:

Construcción de matriz de definición de uso de atributos. Las columnas de la matriz son cada uno de los atributos de la tabla a fragmentar; y los renglones se representan con cada uno de los nodos participantes.

Definición de matriz de afinidad de grupo. Se construye basándose en la posibilidad de que dos o más atributos (columnas) de la matriz de uso de atributos sean usados juntos por un mismo proceso en uno de los nodos.

Fragmentación de matriz de afinidad y asignación de fragmentos los diferentes nodos. Se revisa el peso (importancia) y la frecuencia de uso que tiene cada proceso en cada nodo, se agrupan y reorganizan los atributos, colocando juntos los que tienen un mayor valor. Se hacen iteraciones fragmentando la matriz por columnas y registros y revisando los procesos que quedan en la mitad superior e inferior de la matriz dividida y el peso de los citados procesos. Se repite hasta que la combinación optima de "proceso / nodo" se alcanza.

La implementación que hace el Oracle7 Server (y mantenida con mínimos cambios en las versiones Oracle8, Oracle8i y Oracle9i) de los llamados modelos fragmentados, y como se explica ya en las 12 reglas de las bases de datos distribuidas (Regla 12, Independencia de

fragmentación) es usando las *ligas de base de datos* para la fragmentación horizontal y los *grupos de columnas*, para la horizontal.

Por ser parte de la opción de replicación avanzada de Oracle, además de ser hasta cierto punto un modelo replicado pues las llaves primarias deben replicarse a cada nodo para fragmentar la tabla por columnas, los *grupos de columnas* se abordaran en el capítulo IV.

El trabajo típico en una base de datos sin replicación de datos son transacciones distribuidas que usan datos locales y remotos que se modifican en tiempo real. Para estos ambientes distribuidos Oracle cuenta con las ligas de base de datos que fueron definidas como un apuntador de una sola dirección de un nodo local hacia un nodo remoto.

La Figura 1.7, a continuación, muestra una liga de base de datos llamada *cu.unam.edu.mx* de un *Nodo Local* hacia un *Nodo Remoto*. El usuario *profesor* puede usar los objetos en el nodo remoto, en el ejemplo la tabla *estudiantes*, como si se tratase de una tabla en su propio nodo de base de datos.

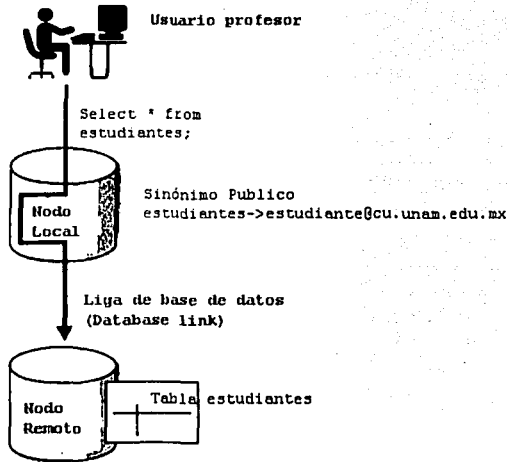


Figura 1.7 Liga de base de datos

La conexión a la base de datos remota se hace con los permisos de acceso que tiene el usuario en el nodo remoto usado para crear la liga de base de datos. Así todos los usuarios en un nodo local que usan una misma liga de base de datos se conectan usando el mismo usuario remoto.

Existen variaciones a la manera en la que se hace la conexión remota con las ligas de base de datos, que puede ser de tres distintas maneras:

- **Usuario conectado.** En este escenario, al momento de creación de la liga de base de datos, no se especifica un usuario o clave de acceso existe en el nodo remoto. La conexión se hace con el mismo usuario que esta conectado en el nodo local.
- **Usuario actual.** Es un usuario global que tiene una capa adicional de seguridad y debe existir en los dos nodos.
- **Usuario fijo.** El usuario y clave se especifican al crear la liga de base de datos y no pueden cambiarse. Cuando el usuario local usa la liga, la conexión remota se hace como el usuario en el nodo remoto definido en la liga.

Las ligas pueden ser además **PUBLICAS** o **PRIVADAS**. Si son privadas sólo el usuario que crea la liga puede usarla; Si es publica esta disponible para todos los usuarios del nodo local.

Para que la conexión en la base de datos distribuida funcione cada base de datos en la base distribuida debe tener **NOMBRE GLOBAL** único que lo identifica en el sistema distribuido. Oracle forma el nombre global con el nombre de la base de datos y el dominio de red del nodo.

Si tenemos dos nodos en un ambiente distribuido con dominios:

Nodo 1.	<i>acatlan.unam.edu.mx</i>
Nodo 2.	<i>cu.unam.edu.mx</i>

Y en ambos nodos existe una base de datos llamada *alumnado*, Oracle formaría el nombre global de la manera siguiente:

Nodo 1.	<i>alumnado.acatlan.unam.edu.mx</i>
Nodo 2.	<i>alumnado.cu.unam.edu.mx</i>

Así los nombres de las ligas de base de datos serán los mismos al nombre global del nodo remoto. Oracle garantiza esto si tiene definido el **PARÁMETRO DE BASE DE DATOS GLOBAL_NAMES** en **TRUE**.

Es recomendado colocar el parámetro **GLOBAL_NAMES** en **TRUE**, aunque su verdadero uso es aprovechar muchas características incluidas en la opción de Replicación Avanzada del RDBMS de Oracle. [PRAT95]

Capítulo II.

Cliente-Servidor y sistemas distribuidos.

2.1 Arquitectura Cliente-Servidor

2.1.1 Diferencia entre procesamiento y ambiente distribuido. Antecedentes del Cliente-Servidor.

Es necesario entender la diferencia entre base de datos distribuida y **procesamiento distribuido**, si bien están relacionados, son diferentes. Comparándolo con la definición antes hecha de ambiente distribuido, el procesamiento distribuido ocurre cuando la aplicación reparte sus tareas entre diferentes computadoras en una red.

Hablando de bases de datos, un cliente conectado al RDBMS, es encargado de presentar la información al usuario y enviar los requerimientos al servidor; mientras que la función del RDBMS es acceder de manera compartida a los datos, verificar permisos de acceso, ordenamiento y otras tareas intermedias.

Con esta separación de tareas, en una base de datos distribuida los posibles métodos de acceso de un cliente a un servidor pueden ser en forma de: *petición remota*, *transacción remota* y *distribuida*, y *petición distribuida*. Donde cada uno se caracteriza por:

Petición remota.

El cliente puede leer y / o actualizar datos en un RDBMS usando una sola sentencia SQL. El RDBMS puede estar localizado en cualquier parte, y la sentencia SQL constituye una unidad de trabajo simple.

Transacción remota.

Una transacción remota ocurre cuando una aplicación de usuario pueden leer o modificar un RDBMS usando múltiples sentencias SQL. Una transacción remota soporta las propiedades ACID de las transacciones.

Transacción distribuida.

En una transacción distribuida la aplicación de usuario corriendo en la máquina cliente pueden leer o modificar datos en múltiples RDBMS usando varias sentencias SQL, que son parte de la misma transacción. La única restricción es que cada sentencia SQL puede tener acceso sólo a un RDBMS a la vez. Esto permite a las transacciones distribuidas contener varias peticiones de datos residentes en múltiples bases de datos. Al igual que las transacciones remotas, las transacciones distribuidas cumplen con las propiedades ACID.

Petición distribuida.

En este método de acceso el cliente pueden leer o modificar datos en diferentes tablas localizadas en distintos RDBMS usando una sola sentencia SQL. Es el método de acceso distribuido más complejo ya que cada petición puede hacer referencia a datos que residen en diferentes nodos, todo dentro de la misma operación SQL, cumpliendo además con las propiedades ACID de las transacciones, el método de bloqueo usado es el *two-phase commit*.

En un ambiente de bases de datos distribuidas con los nodos corriendo el RDBMS de Oracle la repartición de tareas y tipos de peticiones ocurre de manera similar. Cuando uno de ellos requiere

datos localizados en otro nodo, el primero actuara como cliente cuando los datos están en un nodo diferente y convirtiéndose el último en el servidor de la petición. [DUF95]

Es un **servidor** el nodo donde reside el RDBMS, y un **cliente** es la aplicación de usuario que solicita información de un servidor. Cada computadora en la red puede ser un nodo si en ella puede existir una base de datos. Además, cada nodo puede: ser un cliente, si los datos que requiere están en otro nodo; un servidor, si procesa información para un cliente; o ambos.

La **Figura 2.1** muestra el diagrama básico del procesamiento distribuido con un cliente corriendo una aplicación de usuario conectado a un servidor, en este caso de base de datos. El nodo servidor de base de datos puede aceptar mas conexiones de otros clientes o inclusive de otros nodos en el caso de los ambientes distribuidos.

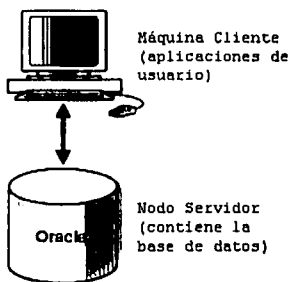


Figura 2.1 Diagrama básico del procesamiento distribuido

El término procesamiento distribuido nació como parte de algo llamado arquitectura Cliente-Servidor. Que a finales de los 80's hacia referencia a computadoras personales conectadas a una red. El software que se desarrollo posteriormente tenia gran versatilidad, flexibilidad y era escalable comparado contra lo que se tenía disponible con el procesamiento centralizado usado la tecnología mainframe. [EDEL94]

En sus inicios el concepto Cliente-Servidor, que fue ganando aceptación, se usaba para definir un grupo de computadoras personales trabajando en red. Las primeras redes de computadoras se enfocaban en compartir archivos con otras localidades remotas, donde los servidores simplemente bajaban archivos o los transmitían hacia los clientes y en estos últimos residía la lógica y el trabajo mismo para procesarlos. Con el tiempo, la evolución de los clientes y servidores generaron dos vertientes que impulsaron a la tecnología Cliente-Servidor:

1. *Con mas usuarios compartiendo archivos, la capacidad para compartirlos en un ambiente multiusuario se mejoro pero no fue posible hacerlo para un gran número de ellos por las razones ya explicadas en el primer capítulo.*

2. *La evolución de las computadoras personales y la introducción de la interfaz gráfica marcaron el estándar para la presentación de información en los clientes.*

La tecnología que resulto para afrontar estas dos necesidades fue el Cliente-Servidor siguiendo un enfoque de dos niveles. Se reemplazó el servidor de archivos por un verdadero servidor de datos que permitió responder a las peticiones del cliente en forma de registros de una tabla, en el caso de un RDBMS, y no con un archivo completo de información como ocurría en los sistemas de manejo de archivos revisados anteriormente. Este enfoque redujo el tráfico en la red e hizo posible el ambiente multiusuario donde los datos son actualizados y consultados desde las máquinas cliente.

2.1.2 Definición de Cliente-Servidor.

La arquitectura Cliente-Servidor no percibe al cliente o al servidor como entidades físicas. Los dos son elementos que ejecutan programas de aplicación. Así, es posible que cualquier computadora capaz de manejar el programa de aplicación puede actuar como cliente o servidor, inclusive ser ambos al mismo tiempo.

En su forma más pura Cliente-Servidor involucra a una entidad de software (cliente) haciendo una petición específica a otra entidad de software (servidor) que la atenderá. [BEHM93]

El cliente procesa y envía los requerimientos al servidor, el servidor interpreta los mensajes e intenta atenderlo. Los requerimientos son en forma de petición de información (en el caso de una base de datos), procesamiento (hacer un cálculo), control de un periférico, o hacer un requerimiento a otro servidor. De esta manera el cliente puede hacer peticiones a diferentes servidores y éste último atender a varios clientes.

La relación entre cliente y servidor es de forma comando-control. En cualquier intercambio de mensajes, el cliente hace la petición y el servidor la atiende. El servidor no iniciara el dialogo con un cliente.

La arquitectura Cliente-Servidor puede ser clasificada basándose en tres factores principales y como van a ser divididos en las entidades de software y distribuidos en una la red. Los factores son:

- **Presentación de información**
- **Procesamiento**
- **Datos**

La repartición de estos elementos origina dos tipos principales de arquitectura Cliente-Servidor, la llamada de dos-capas (también conocida como *two-tier*) y de tres-capas (conocida como *three-tier* o *multi-tier*) [EDEL94]

Arquitectura de dos-capas.

Es la implementación más común del Cliente-Servidor, donde los tres componentes de la aplicación (*presentación, procesamiento y datos*) se dividen en dos entidades o capas de software

que son la aplicación de cliente y el servidor de datos. En general estas aplicaciones cliente son muy robustas y el intercambio de peticiones con el servidor es muy versátil y variado. Su tarea es solamente la presentación de datos, el procesamiento se divide entre el cliente y el servidor y los datos son almacenados y accedidos exclusivamente por el servidor.

En este tipo de configuración, el cliente contiene la lógica (funcionalidad) de la aplicación esto con respecto al procesamiento; mientras que el servidor cuida la integridad de datos y procesa las ordenes enviadas desde el cliente.

Traduciendo esto a un ambiente cliente-servidor de Oracle, las peticiones de usuario se pasan en forma de comandos SQL. Para hacerlo, el cliente debe entender la misma sintaxis en las instrucciones del RDBMS. Esto es transparente para él pues se hace a través de un API (Application Program Interface), también llamado **programa de interfaz de usuario**. El cliente además debe definir la ubicación del servidor y saber cómo y qué existe en cada tabla de datos.

Arquitectura de tres capas.

Surgió por las limitantes que puede tener el esquema de dos-capas en redes de computo que están repartidas en una zona geográfica demasiado extensas. Los componentes de la aplicación fueron separados en tres entidades (capas) de software. El mismo tipo de API puede usarse en esta configuración pero ahora se centra exclusivamente en la presentación. En caso de requerir procesamiento o acceso a datos el cliente hace llamadas a una capa intermedia (también conocida como *middle-tier*) llamada **servidor de aplicaciones**. Este último puede hacer algunos cálculos por el usuario o dirigir el requerimiento al servidor correcto.

Otra diferencia con la configuración de dos capas donde se tiene un intercambio considerable de mensajes entre el cliente y el servidor; al usar tres capas de software y dejar en el cliente sólo la presentación de información, la comunicación se da en forma de **RPC** (Remote Procedure Call) conocidos como **llamadas a procedimientos remotos**. El cliente pasa en forma de parámetros de procedimientos las peticiones de datos que requiere y define a su vez una estructura de datos para recibir información en caso de necesitarla, esto disminuye la cantidad de mensajes y el tráfico en la red además de permitir a la maquina cliente presentar los resultados en cualquier otro formato que podría inclusive ser no relacional.

2.1.3 Estándares Cliente-Servidor y programación de aplicaciones.

[DATE95] Existen infinidad de estándares que pueden ser aplicados a la arquitectura Cliente-Servidor. Algunos de estos estándares han sido adaptados inclusive por los definidos para el SQL (estándar SQL/92)

Uno de estos estándares define las operaciones de **conexión y desconexión**. Sin estar conectado a un servidor, el cliente es incapaz de hacer un requerimiento al servidor y comenzar a intercambiar mensajes. Una vez que la conexión es hecha el API del cliente forma los comandos SQL para que sean atendidos por el RDBMS.

Se definió la posibilidad de tener conexión a mas de un servidor, el estándar Cliente-Servidor permite al cliente, con una conexión activa, establecer una segunda conexión a otro servidor. Esta segunda conexión coloca en estado latente la primera mientras que el procesamiento de

datos se lleva a cabo en el segundo servidor. Una vez concluidas las sentencias SQL, el cliente puede: (a) volver a la sesión original, dejando latente la segunda o terminándola; (b) establecer una tercera conexión a otro servidor dejando las dos sesiones anteriores en estado de espera.

Un cliente SQL puede tener una conexión activa y un número cualquiera de conexiones inactivas o latentes, donde todos los requerimientos de datos son dirigidos y procesados por el servidor donde se tiene la sesión activa. De la misma manera en la que se establecen las conexiones, con el uso de la operación de conexión, eventualmente debe ser terminada de igual manera con la operación de desconexión.

Un segundo estándar es el de Acceso Remoto de Datos (Remote Data Access) definido por ISO (International Organization for Standardization), que ha sido ya implementado por el SAG (SQL Access Group), que es un consorcio de proveedores de software de base de datos, incluyendo a Oracle, dedicados a asegurar la operabilidad de sistemas abiertos.

El estándar define formatos y protocolos para la comunicación Cliente-Servidor. El cual asume:

- (a) El cliente hace sus requerimientos al servidor en forma de sentencias SQL (básicamente un subconjunto del estándar SQL/92), además;
- (b) El servidor reconoce un diccionario de datos base (también definido por SQL/92) Con esto se asegura el intercambio claro de mensajes (peticiones SQL, datos, resultados e información de diagnóstico) entre el cliente y el servidor.

Además se definió que todas las bases de datos relacionales son sistemas donde los resultados son de nivel conjunto, no de nivel registro. Esto quiere decir que la programación de aplicaciones Cliente-Servidor debe considerar al servidor de base de datos no solo como un método de acceso donde se ejecute código de nivel registro. Contrariamente, para evitar problemas de desempeño, se debe tener claro el manejo de peticiones de nivel de conjunto (operar sobre tablas)

En términos de peticiones SQL significa que se debe de evitar al máximo que operaciones DML sean anidadas. Cada una de estas operaciones implica el intercambio de mensajes entre el cliente y el servidor, a mayor número de operaciones mayor intercambio de mensajes y mayor tráfico de red.

Un método que define el estándar para disminuir el número de mensajes intercambiados es el uso de los **Procedimientos Almacenados**, son unidades compiladas de programa que se encuentran almacenadas en el RDBMS. Estas son llamadas por el cliente a través de RPC. Las ventajas identificadas son: {DUFE95}

1. El procesamiento de datos manejados por el procedimiento almacenado se lleva a cabo en el servidor no en el cliente. Esto reduce los mensajes intercambiados.
2. Los procedimientos pueden ser compartidos por muchos clientes.
3. Mejora el desempeño pues son unidades de programación ya compiladas, no se compila al momento de ejecución como en el caso de procedimientos ejecutados desde los clientes.

Estos estándares del SQL/92 son observados por el Oracle7 Server (y las versiones 8, 8i y 9i)

2.2 Diseño de un ambiente distribuido.

2.2.1 Consideraciones en el diseño.

La construcción del ambiente distribuido debe balancear las necesidades del negocio con la infraestructura física de la empresa y la tecnología disponible para la implementación. Es necesario plantear compromisos entre los niveles de servicio que son requeridos con los costos asociados para construir o adaptar la infraestructura que se necesitara.

Se requiere la participación tanto de desarrolladores como de administradores de sistemas para hacer la recolección de todos los factores que puedan afectar el diseño, siguiendo las premisas:

- *La distribución física de la información esta dada por la lógica de la aplicación.*
- *Reducir tráfico en la red de comunicación, acercando los datos a los procesos que hacen uso de ellos.*

Idealmente debería partirse con una aplicación que tenga visibilidad en todas las áreas de la organización pero a la vez no sea de alto riesgo, al decir alto riesgo se habla en términos de complejidad.

Otro factor importante es el contar con el soporte de las diferentes partes involucradas y definir claramente cual es el papel de cada una de ellas.

Se debe revisar cuidadosamente cual es la naturaleza distribuida de la aplicación, si por el tipo de recursos con los que se cuentan y por los resultados esperados; va a ser necesaria la distribución de datos con un modelo fragmentado, o utilizando replicación entre los nodos.

En complemento se debe revisar que tipo de RDBMS se tiene en cada nodo participante. De esto se determinara si será un *ambiente distribuido homogéneo*, donde todos los RDBMS son del mismo tipo, o un *ambiente distribuido heterogéneo* en el caso contrario. Esto obviamente puede representar esfuerzos adicionales de configuración entre los nodos que permitan su integración.

Al elegirse un enfoque distribuido de datos, la principal ventaja es que no existe redundancia pues es una copia única de datos distribuida entre los nodos, están siempre actualizados y son consistentes. Además de no representar un esfuerzo considerable en administración para el DBA. La desventaja es que existe un punto único de falla. Si un problema ocurre y los datos en uno de los nodos no están disponibles, la sección de la aplicación que los usa y sus usuarios dejan de trabajar.

Si se decide usar el modelo con redundancia de datos, usando replicación. Desde el punto de vista de los usuarios el ambiente tendrá alta disponibilidad y desempeño; contrario a la administración que será un poco más compleja.

Una decisión importante es el balancear el hecho de que entre menos nodos replicados se tengan, la dificultad del trabajo para el DBA es menor. De igual manera, a mayor redundancia o nodos replicados, un mejor desempeño y disponibilidad de aplicaciones existirá para los usuarios.

Antes de distribuir los datos se debe tomar en cuenta que el modelo distribuido no es siempre conveniente para todos los negocios. Si por ejemplo, la naturaleza de una empresa no tiende a la descentralización, entonces no debería usar un modelo distribuido. En estos casos puede considerarse el mejorar la infraestructura de comunicaciones para mejorar el tiempo de acceso a datos de manera remota en lugar de hacer copias inútiles de datos dentro de la misma.

La **Tabla 2.A.** enseguida muestra una **tabla comparativa entre ambientes centralizados y distribuidos**, desglosa las ventajas y desventajas de cada ambiente y puede servir para soportar la toma de decisiones al diseñar la aplicación y ambiente a implementar:

	<i>Ambiente Centralizado</i>	<i>Ambiente distribuido</i>
Costos de acceso de Datos	Los costos pueden ser altos. Ambas aplicaciones, remotas y locales requieren acceso al nodo central. Si hay consultas pesadas, los costos pueden ser considerables.	Son generalmente más bajos. Las aplicaciones tienen acceso a datos almacenados localmente. El método de replicación empleado para mantener la consistencia de datos es factor.
Disponibilidad del sistema y datos	Solamente un punto de falla existe, si el nodo maestro falla el sistema completo queda inhabilitado.	Una falla en un nodo raramente afecta el funcionar de los otros.
Consistencia de datos con la fuente primaria	Los datos son siempre consistentes pues no hay redundancia en el nodo maestro.	Para las replicas, la consistencia queda latente y depende del método empleado para mantener la consistencia.
Desempeño de la aplicación	Depende de los recursos de la red disponibles.	Debido a que el acceso se realiza de manera local el desempeño es mejor que en el ambiente centralizado.
Balance de trabajo	Existen pocas posibilidades para balancear la carga de trabajo pues es realizado en un sólo servidor.	Ya que se cuenta con muchos CPUs en el ambiente distribuido hay mas opciones de balancear la carga de trabajo.
Control de versiones de software, código de aplicación y seguridad	El control de software, de la seguridad y de las versiones de la aplicación es simple.	El control es complejo por las copias múltiples existentes en el ambiente y la seguridad debe tomar en cuenta todos los componentes involucrados.
Costos de operación	Bajos. Los esfuerzos se ven consolidados en un nodo central.	Altos. El soportar los nodos existentes en el ambiente va a generar costos para cada uno.
Costos de almacenamiento	Bajos. Los datos no se almacenan de manera redundante.	Altos. Depende de cuantas copias redundantes de los datos son mantenidas.

Tabla 2.A. **Tabla comparativa entre ambientes centralizados y distribuidos.**

2.2.2 Plan de trabajo de los nueve pasos.

Marie Buretta sugiere el siguiente plan de trabajo compuesto por 9 pasos para lograr la distribución de datos. Cada paso tiene su objetivo, entrada de información, sugerencias y produce un resultado.

[BURE97] Las tareas necesarias para crear un esquema distribuido son las siguientes:

1. Preparación del perfil distribuido para la aplicación.
2. Preparación del perfil para la infraestructura física.
3. Creación de grupos de recuperación de datos referenciales.
4. Preparación del perfil de proceso / datos.
5. Integración de datos de uso general con el perfil proceso / datos.
6. Creación del esquema de colocación de datos.
7. Verificar que el esquema de colocación de datos cumpla con las reglas existentes del negocio y tecnologías disponibles.
8. Validación del esquema de colocación de datos, debe cumplir con los requerimientos al nivel de servicio.
9. Implementación del esquema distribuido.

1. Preparación del perfil distribuido para la aplicación.

Los objetivos de este paso son los siguientes:

- Descripción de la naturaleza distribuida de la aplicación.
- Tomar en consideración los posibles requerimientos de la red de comunicaciones.

La entrada recibida en esta fase es:

1. Conocimiento de los requerimientos de la aplicación, así como futuras localidades de uso.
2. Diagramas de descomposición funcional. Que son esquemas estructurados donde las funciones del negocio son separadas en procesos de aplicación. La separación o descomposición representará tareas al nivel de procesos manuales o módulos de aplicación.

Es necesario identificar todas las categorías lógicas de desarrollo como lo son nodos, usuarios y procesos. Una vez identificados es necesario asignar valores cuantitativos, esto se puede lograr de la siguiente manera:

Identificar localidades lógicas para la aplicación. Estas representarán diferentes categorías para nodos por desarrollar. En un banco, por ejemplo, se pueden incluir como localidades lógicas las sucursales.

Identificar usuarios finales lógicos para la aplicación. Representan los papeles más representativos jugados por los usuarios de la aplicación. Siguiendo con el ejemplo del banco, los usuarios pueden ser analistas financieros y comerciantes.

Identificar las transacciones del negocio y / o mayores procesos que podrían ser ejecutados por los usuarios finales lógicos. Es decir, las funciones de negocio más importantes según fueron identificadas en los diagramas de descomposición.

Asignar valores cuantitativos a los tipos de localidades. Se trata de anticipar valores al número de usuarios y procesos que cada localidad tendrá.

La **Tabla 2.B** enseguida, muestra una **matriz del perfil distribuido de la aplicación**, como resultado del primer paso. Representa la información recolectada en cada nodo, los usuarios y proceso que se realizara.

Nombre de la aplicación:

<i>Tipo de Localidad</i>	<i>Localidad</i>	<i>Tipos de usuario y numero esperado de usuarios</i>	<i>Transacciones de negocios con velocidades anticipadas de ejecución.</i>
--------------------------	------------------	---	--

Tabla 2.B Matriz de perfil distribuido de la aplicación.

2. Preparar el perfil de infraestructura física.

Los objetivos de esta etapa son:

- Documentar los recursos físicos disponibles en cada posible nodo a desarrollar.
- Identificar los posibles problemas de distribución.

Para determinar la viabilidad de la distribución de datos y procesos para un determinado nodo, es importante tomar en cuenta los recursos disponibles y cuales son las limitaciones.

Los datos de entrada en esta etapa incluyen:

1. Conocimiento de la infraestructura física existente en cada localidad potencial.
2. Conciencia de cualquier limitación que exista en el ambiente.

El enfoque que se puede seguir es encuestando al personal responsable de la infraestructura y consultando la documentación existente, esta investigación debe incluir:

- Conectividad de la red - ambas LAN y WAN.
- Recursos de hardware y la capacidad disponible.
- Recursos de software y su capacidad.
- Niveles de seguridad en función.
- Requerimientos de disponibilidad del sistema actualmente definidos.
- Niveles de conocimiento y habilidad del personal que administra los recursos.
- Cualquier limitación potencial que pueda influir el desarrollo de un nodo como parte del ambiente distribuido.

El resultado del segundo paso se muestra en la **Tabla 2.C**, abajo. Muestra la **matriz del perfil de infraestructura física** y representa la actual infraestructura disponible para el desarrollo del ambiente distribuido, es importante identificar en esta etapa las limitaciones en infraestructura para corregirlas en las primeras etapas de la implementación.

Nombre de la aplicación:							
Tipo de nodo	Red tipo y velocidad	Hardware Recursos	Software Recursos	Nivel de seguridad	Nivel de Disponibilidad	Nivel técnico y soporte de administradores	Otras limitantes

Tabla 2.C Matriz de perfil de infraestructura física.

3. Creación de grupos de recuperación de datos referenciales.

Los objetivos de esta etapa son:

- Identificar grupos de recuperación de datos. Son entidades de datos que por estar relacionados a través de reglas de integridad, o reglas de negocio; deben ser recuperadas juntas para mantener la consistencia de datos.
- Identificar posibles oportunidades para implementar particionamiento de datos en diferentes localidades físicas. Se deben identificar de igual manera las llaves de datos que serán usadas para el particionamiento.

Un componente importante de todo ambiente distribuido es la integridad de datos la cual debe ser preservada no sólo al momento de hacer actualizaciones, también durante las tareas administrativas como lo es la recuperación de fallas. Es vital el identificar las fuentes primarias de datos.

Las entradas en esta etapa son:

1. Diagramas Entidad-Relación que representan la vista lógica de los datos requeridos para soportar la aplicación.
2. Conocimiento de que tan importantes son las reglas de integridad definidas entre las entidades de datos.

En la etapa del modelado lógico de datos del ambiente distribuido, es de gran ayuda considerar características adicionales de las entidades, relaciones y atributos. Las citadas características incluyen:

Entidades

- *Nombre de la entidad y columnas de llave primaria;*
- *Breve descripción;*
- *Guardián de negocios / propietario, quien define reglas de negocio;*
- *Volumen anticipado, incluir tamaño de registros y número;*

- *Crecimiento anticipado;*
- *Lista de atributos que definen las llaves foráneas;*
- *Identificador de confidencialidad, que es el nivel de acceso para usuarios;*
- *Identificador crítico, grado de importancia de la entidad para el negocio;*
- *Disponibilidad, indicando el uso que se dará a la entidad y por quien;*
- *Periodicidad, representa el posible uso cíclico que tendrá la entidad;*
- *Retención, período de tiempo que los datos deberían ser preservados;*
- *Criterio de limpieza, reglas de negocio para el borrado de datos.*

Relaciones

- *Nombre de la relación;*
- *Nombre de entidad 1;*
- *Nombre de entidad 2;*
- *Cardinalidad de las entidades, incluye el máximo número de ocurrencias en el lado "muchos" de la relación.*

Atributos

- *Nombre estándar;*
- *Nombre en el negocio;*
- *Descripción;*
- *Formato;*
- *Identificador de confidencialidad, indica el nivel de acceso a los usuarios debido a la sensibilidad del atributo;*
- *Lista global "usado donde", listado de los procesos que hacen uso del atributo;*
- *Lista de "contenido en", lista de entidades que contienen el atributo.*

La metodología a seguir separará el diagrama Entidad-Relación en subconjuntos, de tal manera que cada grupo de entidades no contenga referencias de llaves foráneas hacia otro grupo de entidades contenidas en otro de los subconjuntos. El proceso es iniciado agrupando lógicamente las entidades, una vez agrupadas, listarlas en una matriz de tal manera que las entidades relacionadas (o grupos de entidades) estén contenidas solamente una vez en la matriz.

El registro de las entidades en la matriz debe considerar lo siguiente:

- Las entidades fuertemente relacionadas deben ser listadas dentro del mismo grupo en la matriz, esto es, deben ser almacenadas y recuperadas juntas.
- Aquellas entidades relacionadas en menor grado que son altamente volátiles (con una alta frecuencia de cambios a los datos contenidos), deben pertenecer al mismo grupo dentro de la matriz. Las relaciones de menor grado ocurren cuando existen relaciones hacia otros grupos de entidades.
- Las relacionadas en menor grado, pero que no son altamente volátiles, deben permanecer juntas. Estas entidades son candidatas a ser replicadas o para ser **DESNORMALIZADAS**.
- Entidades con relaciones débiles son claras candidatas para replicación en los nodos donde son usadas. Son generalmente catálogos de la aplicación, listas de valores y tablas de equivalencias.

El resultado final contendrá grupos de entidades altamente relacionadas que deben ser almacenadas y recuperadas como un conjunto. Un segundo conjunto de entidades relacionadas débilmente que serán replicadas a todas las localidades que hagan uso de ellas. Y un pequeño grupo de entidades medianamente relacionadas con otros grupos que han sido clasificados como altamente relacionados.

Para los grupos medianamente relacionados, donde las llaves foráneas son no volátiles, se puede considerar la desnormalización de datos o la replicación a los nodos donde los datos son requeridos.

La **DESNORAMALIZACIÓN** de datos es el proceso donde de manera deliberada se violan las técnicas de normalización. Un ejemplo es, almacenar de manera redundante el nombre de un cliente en la entidad de ORDENES. El nombre del cliente se encuentra correctamente almacenado en la entidad de CLIENTES, pero por problemas de desempeño puede ser requerido que se almacene en la entidad de ORDENES. Almacenarlo de manera redundante evita la lectura de dos tablas al momento de desplegar la información de pedidos.

Cuando se usa la técnica de desnormalización se debe considerar que es posible la existencia de anomalías al borrar o actualizar datos en una pero no en todas las entidades donde los datos son almacenados. El resultado es inconsistencia en los datos consultados por los usuarios. Es necesario entonces incorporar procesos que realicen estos cambios en todas las entidades.

Si bien el uso de la replicación de datos o la desnormalización traerán mejoras de desempeño, se debe tener en cuenta que la complejidad para el mantenimiento se ve incrementada.

4. Preparación del perfil de uso de proceso / datos.

El objetivo de esta tarea es:

- Identificar los datos usados entre los procesos.
- Remarcar cuales son los procesos mas utilizados.

Esta parte del proceso es también conocida como el modelo Proceso / Entidad. Es una matriz que relaciona procesos de la aplicación a las entidades que estos crean, leen, actualizan o borran. La entrada en esta etapa del proceso es:

1. Diagrama Entidad-Relación (usado en el paso anterior)
2. Diagrama de descomposición funcional (generado en el primer paso)
3. Conocimiento de las características de cada proceso de la aplicación.

Una vez que se inicia la construcción de la matriz es conveniente marcar aquellos procesos que son mas utilizados. El proceso comienza al listar todas las entidades en la parte superior de la matriz, y todos los procesos en la parte izquierda. En la intersección Proceso / Entidad marcar cuales son las actividades que cada proceso realiza.

[BURE97] En la mayoría de las aplicaciones de sistemas, existe una regla de 80/20. Generalmente un 20 por ciento de los procesos realizan un 80 por ciento del trabajo en la

aplicación. Como consecuencia de ello es más importante el hacer y tomar decisiones en favor de los procesos importantes; la manera de identificar este 20 por ciento es considerando:

- Procesos con alta frecuencia de ejecución.
- Que tengan alta visibilidad o importantes para el funcionamiento del negocio.
- Procesos con niveles críticos de servicio.
- Con requerimientos de grandes cantidades de datos y / o procesamiento complejo.

La **Tabla 2.D** muestra una **matriz de proceso / entidad** que incluye los procesos: Captura de ordenes, Facturación de orden y Aprobación de crédito. Y las columnas están representadas por las entidades que generar los procesos. En el ejemplo son: Cliente, Orden, Detalle orden y factura. En la intersección de ambos se muestra el tipo de operación que se realiza de cada proceso en las entidades.

Procesos	Entidades =>	Cliente	Orden	Detalle_Orden	Factura
Captura de ordenes *		L		LEAB	
Facturación de orden *		L		LA	LEA
Aprobación de crédito		L	L	LA	

Tabla 2.D Matriz de Proceso / Entidad.

Donde:

- L - lectura
- E - entrada de datos
- A - actualización
- B - borrado
- * - procesos altamente usados

5. Integración de datos de uso general con el perfil de Proceso / Datos.

El objetivo de esta etapa es completar lo siguiente:

- Integración de datos de uso general con los perfiles de la aplicación proceso / datos.
- Negociación de acuerdos de nivel de servicio.

Esta etapa es importante ya que, en un ambiente distribuido, los datos son un recurso compartido que fluye a través de todas las líneas de negocios de la empresa. En otras palabras, los datos son de uso general. La entrada recibida en esta etapa es:

1. Los perfiles de uso proceso / datos creados en la etapa anterior
2. Conocimiento de requerimientos de datos externos hacia y desde la aplicación.

Esta tarea se completa realizando reuniones con la gente de desarrollo de la aplicación y aquellas personas identificadas como responsables o propietarios de los datos que son requeridos por la aplicación, o que de igual manera necesitan tener acceso a los datos que la aplicación crea o modifica.

Es necesario tomar en cuenta la opinión de los propietarios de los datos pues ellos son responsables de la coordinación de las reglas de negocio asociados a cada entidad, además de

coordinar la administración de las entidades y ser responsables directos de la seguridad de los datos.

Los acuerdos del nivel de servicio requeridos se clasifican en requerimientos de entrada de datos de la aplicación y requerimientos de resultados o salida que necesitan de la aplicación. En la parte de entrada de datos, es necesario tomar en cuenta:

- Identificar cuales datos son requeridos.
- Cuales son las fuentes de cada pieza de datos.
- El rango de cambio típico para cada pieza de datos. Esto en caso de ser necesario el alertar a los nodos de la aplicación en caso de que ocurra un incremento no normal en la cantidad de modificaciones hechas a parte de los datos.
- Acordar requerimientos de nivel de servicio, como se espera que los componentes funcionen y la calidad del servicio que es esperada, confiabilidad del proceso y seguridad.

Con respecto a los datos de salida que la aplicación generará, es necesario decidir lo siguiente:

- Identificar datos requeridos.
- Identificar cuales son los procesos y / o nodos que necesitan algún tipo de notificación de la aplicación.
- Identificar los rangos típicos en los cambios que sufre cada pieza de datos.
- Acuerdos para los niveles de servicio esperados para todas las aplicaciones involucradas.

6. Creación de esquema para la colocación de los datos.

El objetivo es integrar toda información recolectada durante las etapas anteriores y formular un esquema para la colocación de los datos en el ambiente distribuido.

El esquema debe considerar no sólo el flujo de datos de la aplicación hacia la base de datos, además debe considerar como fluyen los datos, por ejemplo, entre las fuentes primarias de datos y las replicas. La entrada de información requerida para completar esta etapa incluye:

1. Perfil para distribución de la aplicación (resultado del primer paso)
2. Perfil de la infraestructura física (generada en la segunda etapa)
3. Listado de grupos de recuperación de datos (generado en tercera etapa)
4. Perfil de proceso / datos (generado en etapa cuatro)
5. Acuerdos de niveles de servicio (resultado de la etapa cinco)
6. Conocimiento de la dirección estratégica de la empresa en relación con preferencias en el uso de una determinada tecnología.

El esquema de distribución debe sugerir:

- Nodos viables para la distribución de datos.
- Consolidar el modelo de distribución elegido
- Distribuir los grupos de datos identificados en el paso tres, recordando que se debe revisar:

- ✓ Colocar los datos cerca de los puntos donde son mas utilizados.

- ✓ Si las mismas instancias de datos serán actualizadas en múltiples nodos, es conveniente centralizarlos.
- ✓ Si el mismo tipo de datos será actualizado desde diferentes nodos (no instancias de ellos), se puede considerar el particionamiento vertical.
- ✓ Si la entidad de datos ya existe en un nodo, es conveniente compartir la copia existente.

Entre mas necesaria es la consistencia de los datos, menor el numero de nodos que pueden existir en el ambiente distribuido.

Una vez decidido donde estarán localizados los datos modificables, el siguiente paso es decidir donde se colocara la información de sólo lectura usada por la aplicación. Si existe un requerimiento inmediato para leer datos de sólo lectura, esto quiere decir que la aplicación necesita acceder datos en tiempo real por lo que tendrá que tener acceso a la fuente primaria de datos a través de peticiones remotas. De la misma manera, si por necesidad de la aplicación se puede considerar un acceso con retraso de unos cuantos minutos, hasta un par de horas por ejemplo, se implementaría la replicación de datos a los nodos que los requieren.

Enseguida se deben definir los nodos que estarán recibiendo los datos generados por la aplicación. Es conveniente definir los anchos de banda existentes entre cada uno de los nodos. Con esta información se puede hacer un análisis previo de los volúmenes esperados de flujo de datos entre los nodos, incluyendo:

- **Flujo de datos de transacciones.** Estimando los picos en el flujo de los procesos mas usados que puede calcularse al determinar la cantidad de datos desde y hacia el proceso por cada ejecución. Multiplicar este valor por el numero máximo de ejecuciones que se esperan en los picos de trabajo.
- **Flujo de replicación de datos.** Que son todas aquellas tareas que replicarán datos de las fuentes primarias al resto de los nodos.

7. Verificar que el esquema de colocación de datos cumpla con las reglas del negocio y tecnologías disponibles.

El objetivo de este paso es asegurarse que la infraestructura física soportará el esquema de distribución de datos definido en la etapa anterior. Si la infraestructura lo permite se puede entonces validar el desempeño; de lo contrario, es necesario modificar el esquema de distribución, o corregir el problema con la infraestructura.

La entrada usada en esta etapa, incluye:

7. Esquema de colocación de datos con la representación de los flujos de datos (salida del paso anterior)
8. El perfil de la infraestructura física (generado en el paso dos)
9. Listado de los grupos de recuperación de datos (salida del paso tres)
10. Perfil proceso / datos (salida del paso 4)
11. Conocimiento de la funcionalidad de la tecnología actualmente usada.

Se hace necesaria una revisión metódica de las posibles limitaciones cada nodo a desarrollar, una vez identificadas se debe preparar un análisis de costo beneficio de corregir los problemas o replantear el esquema de distribución.

Al hacer la revisión de los posibles problemas que un nodo puede tener, se debe usar el perfil de infraestructura física y esquema de colocación de datos, tomando en cuenta:

- Asegurar por anticipado la disponibilidad y confiabilidad de las plataformas usadas.
- Verificar los niveles de seguridad de la aplicación son posibles de implementar.
- Asegurarse que los niveles de administración y soporte de la aplicación esté disponible.
- Verificar que cualquier error detectado en el perfil de la infraestructura física puede ser resuelto.

Al evaluar posibles limitaciones de capacidad, usar el perfil de la infraestructura física con los estimados del tráfico de la red del esquema de colocación de datos, revisando:

- No existen limitaciones por las plataformas usadas. Basado en la cantidad de datos que ha sido estimada y el numero de usuarios concurrentes de la aplicación.
- Revisión de posibles limitaciones del RDBMS usado.
- Validar que los anchos de banda entre los nodos son suficientes para soportar el tráfico estimado de datos.

Para la revisión de las tecnologías disponibles, usar el perfil de infraestructura, el perfil de proceso / datos, listado de grupos de recuperación, y el esquema de colocación de datos. Esta sección debe incluir:

- Soporte para las unidades de trabajo distribuidas.
- Soporte de las alternativas para replicación de datos que se requieren.

8. Validar que el esquema de colocación de datos cumpla con los requerimientos al nivel de servicio.

En este paso se debe asegurar que el esquema de distribución de datos propuesto y la infraestructura física podrá cubrir los requerimientos de desempeño definidos en los acuerdos del nivel de servicio. Si cumple, la implementación del sistema distribuido puede iniciar, de lo contrario se debe modificar el esquema de distribución o hace las mejoras necesarias a la infraestructura existente, también es posible negociar nuevamente los niveles de servicio pedidos.

La entrada para esta etapa incluye:

12. Esquema de colocación de datos, mostrando una representación de los flujos de datos entre los nodos (generada en el paso seis)
13. Perfil de infraestructura física (salida del paso dos)
14. Perfil de proceso / datos (salida del paso cuatro)
15. Acuerdos de niveles de servicio (salida del paso cinco)
16. Conocimiento de la capacidad de desempeño usando las tecnologías existentes en la empresa.

TESIS CON
FALLA DE ORIGEN

Una vez que se identifican todas las violaciones de desempeño y / o de nivel de servicio, un análisis costo beneficio ayudara a definir entre, mejorar la infraestructura física para alcanzar los niveles requeridos, negociar niveles de servicio mas relajados, o un diferente esquema de distribución de datos.

Use como herramienta los procesos identificados como de alto uso, los perfiles de proceso / datos y los componentes identificados en el perfil de infraestructura para definir si los requerimientos de desempeño son realmente posibles.

9. Implementación del esquema distribuido.

Una vez completado el análisis se puede seguir con el desarrollo del ambiente distribuido. Los pasos anteriores permitirán a los grupos de desarrollo y técnico, plantear una correcta distribución de datos y procesos. Además, la salida producida en cada uno de los pasos servirá como un repositorio de información de los datos, procesos, hardware, software y recursos de red que podrían servir de base a futuras implementaciones o para anticipar incrementos en el flujo de datos y posibles mejoras que la infraestructura necesite.

2.3 Consideraciones de infraestructura y administración del sistema distribuido.

2.3.1 Para la administración del sistema distribuido.

Al igual que la etapa de diseño, la administración de un ambiente distribuido se vuelve más compleja pues se deben cuidar factores como recursos de red, esquemas de base de datos que contienen los datos particionados, y los que tienen datos replicados (datos y otras estructuras de control) La falta de vigilancia de estos factores pueden ocasionar problemas de desempeño, uso ineficiente de la red y otros recursos; además de problemas de consistencia de datos. Es responsabilidad de los DBA, que apoyados para ciertas tareas por los administradores de red el vigilar correctamente los recursos y así maximizar el tiempo de operación del ambiente distribuido.

Una vez que se ha decidido el enfoque en la distribución de datos que se va a seguir los administradores tendrán que prepararse para resolver algunos inconvenientes propios del enfoque, principalmente con la replicación de datos. Algunos de estos factores son:

Semántica de transacciones.

En un ambiente distribuido con replicación donde existen varias copias de los datos en los diferentes nodos. Se llama **semántica de transacciones** al cuidar que las transacciones distribuidas propaguen los cambios hechos a los datos como se haría en una base de datos central única. Esto es necesario para mantener la integridad referencial definida en cada nodo.

Para prevenir los estados inconsistentes de los datos se debe determinar el orden correcto de actualización el cual cambia si se trata de operaciones DML de inserción y modificación, comparado con el usado para el borrado de datos.

En Oracle la semántica de las transacciones esta garantizada ya que basándose en la información contenida en el diccionario de datos Oracle internamente ordenara el orden de ejecución de las transacciones dependientes para asegurar la integridad de la base de datos[PRAT95]

Replicación de datos usando procedimientos almacenados.

La replicación de procedimientos almacenados se puede definir también como replicación funcional. La característica de la replicación por procedimientos es que permite encapsular muchos cambios a los datos dentro de una sola función replicada. Contrario a lo que pasa en la replicación de tablas donde el cambio a nivel registro es propagado a los nodos replicados.

Oracle permite la replicación de datos por procedimientos como parte de la opción de Replicación Avanzada para propagar cambios al nivel de tabla. Genera los procedimientos necesarios para simular la misma transacción en el nodo remoto y más tarde lo ejecuta usando RPC. [PRAT95]

(En Oracle 8i y posteriores se introdujo la propagación paralela para reducir el tiempo de espera en la propagación de actualizaciones masivas)

Triggers de base de datos y llaves primarias.

Si el ambiente distribuido incluye varias copias primarias (nodos dedicados a la captura de información) que requieren de identificadores globales únicos o llave primaria, la manera en la que se deben generar es a través del esquema de la base de datos o de la aplicación. Las opciones existentes son: [BURE97]

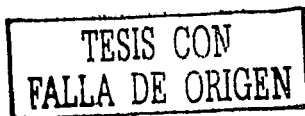
- ✓ *Agregando columnas a un identificador local único que permitan designar quien es propietario o donde se están originando los datos.*

Con esta solución el identificador primario es una llave primaria compuesta, cada base de datos primaria asigna una secuencia única de números y / o letras que la hace única entre todas las bases de datos del ambiente distribuido.

- ✓ *Asignando rangos de valores o combinaciones de números y letras que son únicas en cada base de datos.*

Se define un esquema para asignación de un rango de valores a cada uno de los nodos. Este esquema de asignación de valores tiene como desventaja que algunos de los nodos podría quedarse sin números en el rango que tiene asignados. Además, el identificador de la columna no es atómico (formado de una columna), ya que esta formado por mas de una columna.

- ✓ *Uso de un mecanismo externo para la generación de valores únicos para todas las bases de datos.*



El uso de esta alternativa implica que todos los nodos deben ir al nodo designado para generar sus identificadores únicos, esto puede traer como consecuencia cuellos de botella que afectaran el desempeño.

Otro factor importante a considerar son los triggers de base de datos. Un trigger es un bloque de código asociado a una tabla que esta almacenado en la base de datos. Las acciones programadas en el trigger se ejecutan de manera automática por el RDBMS cuando se ejecuta una sentencia SQL especifica en la tabla donde esta definido.

Al usar replicación se debe tener cuidado si se tienen definidos triggers en las bases de datos primarias y las replicadas. Cambios a los datos hechos por la ejecución de triggers son también marcados para ser actualizados al resto de las replicas como si se tratase de cambios hechos por la aplicación de usuario. En la base de datos replicada, estos triggers serán ejecutados cuando los cambios sean hechos por el proceso de replicación de datos. Por lo tanto, si los mismos triggers se definen en todas las bases de datos primarias y replicas, los cambios hechos por los triggers en la base de datos primaria se ejecutaran dos veces en las bases de datos replicadas, primero por el trigger después por el proceso de replicación.

En bases de datos Oracle, la manera de asegurar que el trigger solamente se ejecuta una vez por transacción es proveyendo un procedimiento llamado `DBMS_REP_UTIL.FROM_REMOTE` que coloca una bandera de verdadero o falso dependiendo si el cambio es hecho por el usuario o el proceso de replicación de datos. Esto se coloca como parte del código del trigger que será ejecutado si el valor es falso.

Un segundo método habilita y deshabilita la replicación de datos usando otro procedimiento llamado `DBMS_SNAPSHOT.I_AM_A_REFRESH`, que con valores de falso o verdadero que se pueden definir también al inicio y final del trigger para solamente ejecutarse cuando los cambios son hechos por el usuario.

2.3.2 Para la infraestructura del ambiente distribuido.

La implantación del ambiente distribuido depende en gran parte de la infraestructura con la que se cuenta. La infraestructura no incluye solamente el ambiente físico y tecnológico, es además política; y procedimientos necesarios para soportar un ambiente distribuido de manera eficiente y efectiva.

Con el objeto de determinar si la infraestructura existente cubrirá los requerimientos del ambiente distribuido se debe definir:

1. *Definición del un enfoque de análisis.* Ya sea descendente, comenzando con la aprobación del proyecto de implementación de la organización como tal; o ascendente, partiendo del trabajo con grupos de usuarios del ambiente distribuido.
2. *Determinar cual sería el entrenamiento requerido por administradores y usuarios.*
3. *Definición de estándares y procedimientos entre los nodos.*

Todo ambiente distribuido requiere un esfuerzo extra para la definición de infraestructura necesaria y el soporte que esta necesita, esto permitirá que el sistema distribuido permanezca abierto, escalable y aproveche al máximo todos los recursos (red, hardware, software y habilidad de administradores)

El **análisis descendente** obtiene un compromiso de la gerencia. Partiendo de la importancia de implementar el ambiente distribuido y detallando la complejidad que esto acarrea. Este compromiso permite asignar recursos para las tareas de implementación sobre una infraestructura robusta. Esta es la situación ideal porque:

- Provee el compromiso necesario para destinar recursos.
- Crea una nueva cultura dispuesta a modificar las políticas existentes.
- Crea un foro para señalar fallas en la infraestructura existente.

Por su parte el **análisis ascendente** integrara grupos de administradores de aplicación, desarrollo y usuarios. Los grupos conocen los problemas con inconsistencia en los datos, desempeño y problemas para obtener datos de varias localidades; además de ser consientes de la complejidad requerida para corregir esta situación. Los objetivos primarios son adaptar al máximo o mejorar el uso de los recursos actuales. El resultado de estas reuniones de trabajo será una guía inicial de la infraestructura necesaria para la implantación del ambiente distribuido.

Enseguida se debe definir el entrenamiento necesario para los usuarios y administradores. Esta es una tarea que no se puede dejar de lado pues es más compleja la tecnología involucrada en una base de datos distribuida. Los individuos que reciban el entrenamiento estarán desempeñando las siguientes tareas:

- **Administradores del sistema**, su papel será el dar soporte a todo el software del sistema.
- **Administradores de base de datos**, encargados de vigilar y dar soporte a la base de datos, sus objetos y estructuras.
- **Administradores de datos**, cuya labor es trabajar con los usuarios para definir datos y reglas necesarias para que el negocio funcione.
- **Administradores de red**, su papel incluye dar soporte a la infraestructura para comunicación.

El tercer paso señalado es definir estándares y procedimientos, estos tendrán que formar la base de todos los ambientes técnicos dentro de la organización. Se definen responsables para tareas específicas, tipo de maquinas cliente y los servidores de base de datos, formato de pantallas de captura de usuario, definición de ambientes de desarrollo, el control de calidad y ambiente de producción.

Las personas que deben participar en esta tercera etapa son:

Administradores de datos.

Definiendo formatos de entrada de datos al ambiente distribuido y reglas del negocio que se deben cumplir. Además definen los niveles de seguridad y acceso.

Administradores de base de datos.

Al definir el formato para las estructuras de base de datos que están involucradas en el proceso de distribuir o replicar los datos. Establecen niveles de seguridad.

Administradores del sistema.

Definiendo el desarrollo de aplicación, sistemas operativos y cualquier API usado en el ambiente distribuido.

Administradores de red.

Define los estándares para asegurar una infraestructura que permita el tráfico generado en el ambiente distribuido. Incluye herramientas para administración y monitoreo.

En el proceso de definición de la infraestructura para la implantación varias cosas podrían pasar inadvertidas aun contando con elementos de gran experiencia. Las siguientes recomendaciones pueden servir como base para la definición de los procedimientos validos para todo el ambiente distribuido.

En la definición del ambiente distribuido es importante considerar: [BURE97]

Recomendaciones de seguridad.

Los programas encargados de la distribución y replicación de datos, nombres de usuario y sistemas operativos deben adherirse a las políticas de seguridad seguidas por la organización. Se debe incluir lo siguiente:

- ✓ Claves de acceso almacenados o desplegados al momento de ingresar al sistema deben estar encriptados.
- ✓ Las cuentas requeridas o creadas por el servicio de replicación de datos deben cumplir con los estándares de la organización.
- ✓ Todos los sistemas de archivos de sistema operativo, dispositivos de almacenaje que contienen estructuras de la base de datos deben ser debidamente asegurados.

En el Oracle7 Server (y las versiones posteriores) las claves de acceso son encriptadas por el RDBMS y no están disponibles ni siquiera para el DBA.

Recomendaciones para el intercambio de datos y control de calidad.

Es muy posible usar un nivel extra de software que establezca el intercambio de información, principalmente en *ambientes heterogéneos*, entre todos componentes del ambiente distribuido. Debe considerar posibles migraciones de las versiones de RDBMS que se estén usando en el momento. El software para el intercambio debe ser capaz de funcionar con varias versiones de RDBMS en el mismo ambiente.

El servicio encargado de intercambiar información tendría que incluir métodos para:

- ✓ Cambios en el número de elementos de datos involucrados inicialmente. Esto es cambios en las estructuras de datos replicadas, creación de nuevas y manejo de estructuras eliminadas.
- ✓ Cambios en las características de los elementos de datos involucrados actualmente. Como lo son cambios en el tipo de dato.

En el Oracle7 Server la capa encargada para la replicación de los datos esta integrada en el RDBMS mismo. Además, el RDBMS Oracle pueden funcionar en ambientes distribuidos donde los nodos tiene diferentes versiones, Oracle7 y posteriores.

En el área de control de calidad, existen dos campos que deben vigilarse:

- ✓ Desempeño que tiene el software de replicación de datos. Debe incluir pruebas en ambientes controlados para definir la base en la medición del tráfico en la red generado. Registrando el tráfico antes y después.
- ✓ Diseño eficiente de las aplicaciones que usan los datos distribuidos. Probar antes en ambientes de desarrollo para maximizar su desempeño y eficiente acceso a los datos distribuidos. En caso de detectarse problemas en el ambiente de desarrollo es posible para los desarrolladores hacer correcciones antes de colocarla en producción.

Recomendaciones para recuperación.

Se deben definir o identificar los métodos disponibles para la recuperación de transacciones en el ambiente distribuido. Sin son manejados por el RDBMS o es necesaria la intervención del DBA.

Las operaciones de recuperación pueden clasificarse en tres secciones. La primera debe indicar cómo es el monitoreo normal del ambiente distribuido, la segunda trata los procedimientos de recuperación definidos para fallas; y la tercera es solución a problemas comunes.

Los procedimientos de recuperación deben tomar en cuenta el que hacer en caso de:

- ✓ Falla de algún componente del ambiente distribuido. Incluyendo mecanismos para la propagación a los nodos los cambios a datos.
- ✓ Fallas de hardware. ¿Qué hacer en caso de una falla física en uno de los nodos?
- ✓ Fallas de la red de comunicaciones.
- ✓ Errores internos en los RDBMS.

Recomendaciones para reconciliación de datos y resolución de errores.

Los procedimientos de reconciliación de datos sirven para validar la consistencia de los datos entre todos los nodos del ambiente distribuido y para sincronizar los datos una vez que los datos quedaron inconsistentes. Las inconsistencias pueden ser originadas por la falla de algún componente, errores en la codificación o en propagación de los cambios hacia los nodos.

Al momento de implementar el servicio de distribución, estos procedimientos sirven para validar que la propagación de datos se esta haciendo de manera correcta. Los procedimientos no son otra cosa que utilitarios de comparación de datos, generalmente son provistos por el software empleado en forma de paquetes en la base de datos. Los utilitarios comparan los datos que se encuentran en alguno de los nodos con los que se han sido capturados o actualizados en otro reportando las diferencias encontradas.

En cuanto a la resolución de errores, la meta es asegurar que después de un error, la solución del mismo existe. El software de distribución de datos puede ser configurado para enviar los datos que no pueden ser propagados, y hacen que una transacción falle, a un archivo de log y permitir que el resto de la replicación de datos continúe. Sin embargo, los encargados de la administración deben ser alertados de cierta manera de estas transacciones que no pudieron completarse. Los procedimientos de manejo de errores sirven para capturar posibles errores de diseño o codificación que no fueron detectados durante las pruebas del sistema.

Oracle7 y las versiones posteriores proveen mecanismos automáticos para la solución de conflictos por inconsistencia de datos causada por fallas en alguna transacción distribuida. Es posible asignar estos mecanismos para en caso de errores Oracle corrija los errores con intervención mínima o nula del DBA.

Además el DBA cuenta con vistas especiales del diccionario de datos para detección de transacciones distribuidas que pudieron causar datos inconsistentes. Y puede regresar los datos a su estado consistente forzando el completar o deshacer los cambios a los datos hechos por la transacción.

Recomendaciones para herramientas de administración.

Una vez elegido el software para la distribución de datos, el grupo de administradores podría elegir desarrollar herramientas y procedimientos automáticos para el mantenimiento y corrección de errores en el ambiente distribuido. Algunas sugerencias para a considerar son:

- ✓ Uso de hojas de cálculo para ayudar en la planeación de capacidad. Su uso permitiría anticipar el volumen de replicación de datos.
- ✓ Utilizar herramientas, generalmente del proveedor de software, que ayuden a la creación y modificación de objetos distribuidos.
- ✓ Agregar el servicio de distribución de datos al monitoreo hecho en la organización a otras aplicaciones para medir el uso y desempeño dentro de la red de comunicaciones.

Oracle provee con un grupo de programas para la administración de la base de datos, desde la versión Oracle7 y posteriores, se tiene un módulo llamado Administrador de Replicación (o Replication Manager) que con el uso de una interfaz gráfica permite ver el estado de cada nodo de replicación, ver como se están replicando los datos, la comunicación entre los nodos, que transacciones están en duda y hacer cambios en tiempo real a la configuración y frecuencia con la que se replican los datos. Por estar integrado con el resto de las herramientas administrativas permite ver el tamaño físico que tienen las estructuras de control de replicación y el espacio que tienen disponible para crecer en caso de requerirlo.

TESIS CON
FALLA DE ORIGEN

Capítulo III.

Arquitectura del Oracle7 Server y SQL*Net.

3.1 Introducción y estructuras de datos del Oracle7 Server.

3.1.1 Introducción al RDBMS Oracle.

Un servidor de base de datos relacional conocido como RDBMS, es la pieza de software que implementa una base de datos siguiendo los conceptos del álgebra relacional. Es capaz de mantener la integridad de datos y seguridad en ambientes multiusuario. Cuenta con un lenguaje para la manipulación de datos existentes y para la creación de nuevas estructuras propias de la base; el lenguaje es conocido como SQL.

El Oracle7 Server es un RDBMS que además cuenta con mecanismos para implementar bases de datos relacionales distribuidas usando cualquiera de los métodos de distribución ya revisados (replicación y / o fragmentación de datos) Sin embargo, y el RDBMS de Oracle no esta exento, la evolución de dichos programas y nuevas tecnologías han resultado en versiones posteriores que son Oracle8, Oracle8i y Oracle9i. Los cambios más importantes entre las versiones serán señalados según sea necesario y relevante al objetivo del trabajo.

El concepto básico en el Oracle Server es llamado **Instancia de Oracle**, se define como la combinación de procesos internos de Oracle y un área en memoria reservada exclusivamente para la base de datos que es compartida por usuarios y procesos Oracle, está es llamada System Global Area (SGA)

Físicamente, la base de datos son archivos de sistema operativo que contiene los datos de los usuarios y otras estructuras de control. Son manejados en su totalidad por el RDBMS. Cada base de datos es operada por una Instancia de Oracle.

La Figura 3.1 enseguida muestra una Instancia de Oracle compuesta por el área de memoria SGA y los procesos Oracle encargados de hacer todas las funciones del RDBMS.

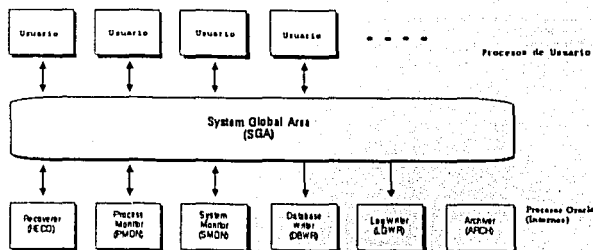
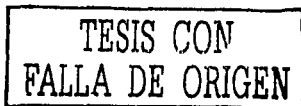


Figura 3.1 Instancia de Oracle

Para que la base de datos este disponible a los usuarios, la instancia es creada por el DBA. Esto reserva la memoria (SGA) y levanta los procesos de Oracle necesarios, los cuales revisan la integridad de los archivos físicos, y al terminar esto se dice que la base de datos esta abierta.



El siguiente concepto son los procesos de Oracle y procesos de usuario. Son los dos tipos de proceso que pueden existir en una instancia Oracle, se definen como:

- **Proceso usuario.** Como su nombre lo indica es el proceso creado cuando un usuario se conecta a la base de datos. Ejecuta peticiones SQL que el usuario indica usando una herramienta de interfaz (API) para manipular los datos en la base
- **Proceso Oracle.** Proceso interno levantado y mantenido por el RDBMS encargado de realizar tareas para los procesos de usuario o de mantenimiento para el Oracle Server.

3.1.2 Principales estructuras lógicas y físicas.

Para el RDBMS Oracle, la base de datos es una unidad cuyo simple propósito es el de almacenar y recuperar información que se encuentra relacionada. La manera de facilitar el control de esta llamada unidad es separando las estructuras internas en **lógicas** y **físicas**. Mientras se realizan tareas de mantenimiento a las estructuras físicas el acceso a algunas estructuras lógicas puede ser permitido. [BOCH93]

Las estructuras lógicas son llamadas: *tablespaces*, *schema* (*esquemas de usuario*), *data block* (*bloques de datos*), *extent*, y *segment* (*o segmentos*) Se definen como:

TABLESPACE.

Es la división lógica que agrupa objetos relacionados para facilitar la administración. Cada base de datos se divide en uno o más tablespaces y cada tablespaces esta formado de uno o más archivos físicos de datos.

El tablespace puede estar **en línea** (accesible para los usuarios) o **fuera de línea** (no accesible) Los objetos contenidos en el tablespace pueden ser utilizados solamente si el tablespace está en línea. El DBA tiene los privilegios para colocar un tablespace en línea o fuera de línea.

SCHEMA (ESQUEMA DE USUARIO)

Es una colección de objetos (estructuras lógicas) los cuales contienen los datos mismos de los usuarios. No existe relación alguna entre el tablespace y el esquema pues un esquema puede tener objetos en varios tablespaces y un tablespace puede tener objetos de varios esquemas. El DBA al agregar un nuevo usuario a la base de datos hace que internamente se designe un esquema lógico con el mismo nombre, el cual contendrá todos los objetos que el usuario va creando.

Las estructuras lógicas que puede contener un esquema de usuario son: *tablas*, *vistas*, *procedimientos almacenados*, *indices*, y *ligas de base de datos*.

DATA BLOCK (BLOQUE DE DATOS)

Es la unidad lógica que define el grado de almacenamiento de datos mas granular con el que cuenta Oracle. Un bloque de datos corresponde a un número determinado de bytes de espacio en disco y se especifica la primera vez cuando la base de datos es creada.

TESIS CON
FALLA DE ORIGEN

Oracle reserva y usa el espacio libre en los archivos de datos físicos en forma de bloques de datos.

EXTENTS.

Un extent es el siguiente nivel lógico para el manejo de espacio. Se define como un conjunto contiguo de bloques de datos que son reservados por Oracle para almacenar datos específicos.

SEGMENTS (SEGMENTOS)

Un segmento es un conjunto de extents (no necesariamente contiguos) reservados para almacenar una determinada estructura, los tipos de segmento que existen son:

- ✓ Segmento de datos. Usados para almacenar **tablas** de la base de datos. Todos los datos de la tabla se almacenan en los extents del segmento que le corresponde.
- ✓ Segmento de índice. Usados para almacenar **índices** de base de datos.
- ✓ Segmentos de rollback. Segmentos especiales de la base de datos, creados por el DBA y usados por Oracle almacenando información necesaria para hacer 'rollback' a las transacciones no confirmadas.
- ✓ Segmentos temporales. Segmentos creados por Oracle usados cuando una sentencia SQL requiere de un espacio temporal de trabajo para ser completada (ordenamiento de datos, entre otras) Al concluirse los segmentos son liberados para ser reutilizados.

Revisando los objetos lógicos contenidos en el esquema de usuario tenemos:

TABLES (TABLAS)

Una tabla es la unidad básica de almacenamiento de datos en una base de datos Oracle. Contiene toda la información que es accesible para el usuario.

Los datos son almacenados en la tabla en renglones (rows) y columnas (columns) donde cada tabla esta definida con un nombre y un conjunto de una o más columnas. Cada columna tiene un nombre y puede contener un **tipo de dato**.

VIEWS (VISTAS)

Presentación personalizada de los datos contenidos en una ó más tablas. NO almacenan datos solamente muestran la información de sus tablas base. Pueden realizarse las mismas operaciones DML que están permitidas sobre las tablas con algunas restricciones.

PROGRAM UNITS (UNIDADES DE PROGRAMACIÓN)

Son bloques de código PL/SQL que se almacenan en forma de **procedimientos almacenados (stored procedures), funciones (functions), paquetes (packages) y triggers.**

SEQUENCES (SECUENCIAS)

Lista única de números controlados por Oracle que es asignada a columnas tipo numérico.

Son usadas para simplificar la programación de aplicaciones al generar valores únicos para una o más columnas que lo necesitan.

SYNONYMS (SINÓNIMOS)

Es un alias o nombre corto para otro objeto de usuario como lo son *tablas*, *vistas*, *unidades de programación*. No es un objeto sino una referencia a uno. El sinónimo puede ser público (para ser usado por todos los usuarios) o privado (para el usuario que lo creó)

INDEXES (ÍNDICES)

Son estructuras opcionales que están asociadas a tablas, su objetivo es disminuir el tiempo de acceso en la recuperación de datos. El índice es una ruta de acceso directo a los datos de una tabla. Los índices se crean sobre una o varias columnas de la tabla. Una vez creado el índice es administrado y usado por Oracle.

DATABASE LINKS (LIGAS DE BASE DE DATOS)

Un database link es un nombre que se da a una ruta de acceso a objetos en una base de datos remota. Como ya se definió en el primer capítulo los database links permiten hacer operaciones DML sobre los objetos remotos, que si se combina con el uso de sinónimos lo hace transparente para el usuario.

DATA DICTIONARY (DICCIONARIO DE DATOS)

Conjunto de tablas y vistas de sólo lectura que contienen información de referencia de la base de datos. Es usada con propósitos de administración y contiene información de todas las estructuras lógicas y físicas de la base de datos.

El diccionario de datos es creado junto con la base y es manejado en su totalidad por el RDBMS.

Las estructuras físicas de la base de datos Oracle son: *datafiles (archivos de datos)*, *control files (archivos de control)* y *redo log files (archivos de redo)*

DATAFILES (ARCHIVOS DE DATOS)

Cada base de datos tiene uno o más archivos de datos. Físicamente contiene todos los datos de las estructuras lógicas. Puede estar asociado a un solo tablespace, y un tablespace estar formado a su vez por varios archivos de datos.

La primera vez que los datos del archivo son leídos, Oracle los almacena en un área del SGA para que sean procesados por los usuarios lo que reduce el tiempo de acceso a los discos.

Lo mismo ocurre cuando las transacciones son completadas, para mejorar el desempeño en lugar de hacer estos cambios cada vez que el usuario lo modifica, las transacciones son colocadas en otra área del SGA y actualizadas después de un determinado tiempo a cada archivo de datos por uno de los procesos de Oracle llamado DBW (*DB-Writer*)

CONTROL FILES (ARCHIVOS DE CONTROL)

Cada base de datos tiene un archivo de control el cual contiene el detalle de la estructura física de toda la base de datos.

Pueden existir varios por seguridad que son actualizados automáticamente por el RDBMS. La información en el archivo de control es leída cada vez que la base de datos va a ser abierta, indica que archivos de datos y archivos de log deben ser abiertos para la base de datos. Otro uso que tiene es en la recuperación en caso de fallas del sistema.

REDO LOG FILES (ARCHIVOS DE REDO)

Cada base de datos tiene un grupo de dos o más archivos de redo. En conjunto son conocidos como *redo log* de la base de datos. El redo log esta formado por entradas de redo que contienen vectores de cambios hechos por cada transacción.

Cuando es necesario hacer un *rollback* a la transacción, la información original es recuperada del redo log. Su principal uso es para la protección de la base de datos en caso de fallas.

3.2 Arquitectura del Oracle7 Server.

3.2.1 Estructuras de memoria y procesos Oracle.

Una vez que la instancia de Oracle ha sido levantada, una base de datos puede ser abierta por la citada instancia. Es posible levantar múltiples instancias en el mismo servidor y cada una de ellas abrir una base de datos diferente. [BOCH93]

Ya se definieron los diferente tipos de proceso en la instancia Oracle que son de usuario y de Oracle. Un **PROCESO** se define como un flujo de control el cual puede ejecutar una serie de tareas en el sistema operativo donde reside la base de datos.

Cuando un proceso usuario se conecta a la base de datos, Oracle crea para este un llamado **Proceso Servidor** cuya función es atender los requerimientos específicos del proceso usuario en la base de datos. El proceso servidor es considerado un Proceso Oracle pero es diferente a los procesos internos.

Los procesos servidor llevan a cabo tareas como lo son el parseo y ejecución de las sentencias SQL que el usuario hace a través de la aplicación; lectura de los bloques de datos de los archivos de datos, donde los coloca en una zona compartida del SGA, y el regreso de los resultados al proceso usuario para que la aplicación procese la información.

Los procesos internos llevan a cabo tareas de control para el Oracle7 Server, los principales procesos de background que se ejecutan en una instancia son los siguientes:

- Database Writer (DBWR)
- Log Writer (LGWR)
- Checkpoint (CKPT)
- System Monitor (SMON)
- Process Monitor (PMON)
- Archiver (ARCH)

- Recoverer (RECO)
- Lock (LCKn)
- Snapshot Refresh (SNPn)
- Dispatcher (Dnnn)
- Server (Snnn)

La Figura 3.2 enseguida muestra las Estructuras de memoria y procesos Oracle y su interacción dentro de la una base de datos Oracle.

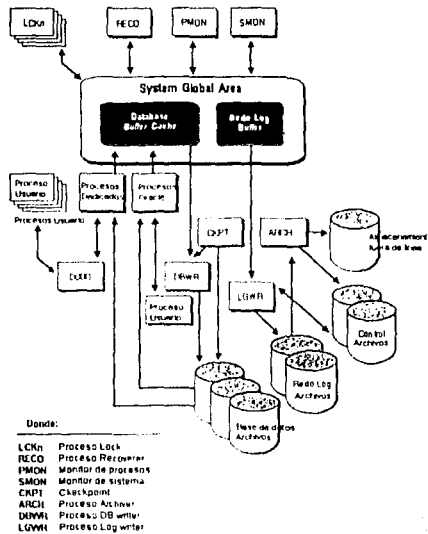


Figura 3.2 Estructuras de memoria y procesos Oracle

Las principales áreas en las que se divide el SGA son como mínimo:

Área de datos (Database Buffer Cache)

Almacena los datos que fueron más recientemente leídos de los archivos de datos, son bloques de datos modificados por usuarios y sin modificar. Se mantienen en memoria para mejorar el tiempo de lectura y reducir el I/O a los archivos de datos.

Área de Redo Log (Redo Log Buffer)

TESIS CON
FALLA DE ORIGEN

Almacena entradas de redo, es la bitácora de todos los cambios en la base de datos. Al llenarse el área en memoria se escribe a un archivo de redo log el cual es usado para recuperación en caso de falla.

Área compartida (Shared Pool)

Es una zona para construir sentencias SQL, almacena las sentencias con las rutas de ejecución que se seguirán. El SQL puede ser compartido por varios usuarios si ellos ejecutan exactamente el mismo SQL.

Por su parte los procesos internos de Oracle se definen de la manera siguiente:

Proceso DATABASE WRITER (DBWR)

Encargado de escribir el contenido en memoria del SGA a los archivos de datos (datafiles) Los datos al ser actualizados por una transacción y esta hacer *commit*, los bloques se marcan como "sucios" y es la marca para que el DBWR los escriba al archivo de datos.

Proceso LOG WRITER (LGWR)

El área en memoria que almacena entradas del redo log son escritas a disco (en los archivos de redo log) por el LGWR. La información de redo log es una copia del dato original que se encuentra en el área de datos del SGA y es modificado por un usuario. Mientras la modificación no sea confirmada, la copia del dato modificado se mantiene en el área de redo log del SGA para garantizar lecturas consistentes para otros usuarios leyendo los mismos datos.

Proceso CHECKPOINT (CPKT)

Actualiza los encabezados internos de los archivos de datos con un número de secuencia para asegurar que estén sincronizados con la última escritura del LGWR y DBWR. Actualiza el archivo de control con información usada para la recuperación de la base de datos.

Proceso SYSTEM MONITOR (SMON)

Realiza recuperación de la instancia Oracle al ocurrir una falla. Libera los segmentos temporales no usados por transacciones y hace reorganización de espacio de los extents libres.

Proceso MONITOR (PMON)

Hace la recuperación cuando el proceso usuario sufre una falla. Libera el área en memoria y otros recursos usados por el proceso que fallo.

Proceso RECOVERER (RECO)

Usado en la distribución de datos, resuelve automática fallas en transacciones distribuidas que están en duda y no se sabe con certeza si fueron completadas o abortadas. Verifica remotamente el estado de la transacción y sincroniza el resultado de la transacción en ambos nodos.

Proceso ARCHIVER (ARCH)

Hace copias de los archivos de redo log a otros dispositivos de almacenamiento secundario como unidades de cinta. Mantiene un histórico de cada cambio hecho a la

base y se usa en bases de datos que no tienen tolerancia a fallas. El proceso ARCH permite respaldar la base de datos sin tener que cerrarla.

Proceso DISPATCHER (Dnn)

Permite a varios procesos usuario compartir un mismo proceso servidor.

Cuando un proceso cliente tiene un proceso servidor para atender solamente sus peticiones, se dice que la conexión usa un proceso de **SERVIDOR DEDICADO**, si hay procesos Dispatcher asignando varios procesos cliente a un mismo proceso servidor se dice que la conexión usa un proceso **SERVIDOR COMPARTIDO**. Este es usado cuando recursos como la memoria es limitada, o existe una cantidad importante de usuarios concurrentes.

Proceso SNAPSHOT REFRESH (SNPn)

Proceso que es parte de la opción distribuida, periódicamente propaga los cambios hacia los diferentes nodos en una base de datos distribuida que usa replicación de datos. Se pueden tener hasta 10 procesos SNP en una instancia Oracle7.

En las versiones 8i y posteriores, este proceso es conocido como Job Queue, su función es la misma pero permite levantar hasta 36 procesos por instancia para la propagación de cambios.

3.2.2 Servicio de replicación de datos del Oracle7 Server.

El RDBMS de Oracle es capaz de participar en una base de datos distribuida, cuando se trata de distribución de datos lo hace de manera transparente usando ligas de base de datos, funcionalidad que esta integrada en el RDBMS.

En el caso de replicación de datos, la opción de replicación es otra tecnología que se encuentra perfectamente integrada al RDBMS y su objetivo es el de replicar y mantener copias de datos en diferentes bases de datos.

La primera implementación de replicación de datos fue desde la versión 6.0 de Oracle en la forma de snapshots de sólo lectura. El siguiente paso, la posibilidad de actualización de los snapshots se implementó en Oracle 7.0.

El concepto snapshot siguen existiendo en versiones 8i y posteriores en la forma de Vistas materializadas (Materialized Views)

El uso de los recién definidos procesos internos llamados *Snapshot Refresh* (SNPn, donde n es el numero de procesos encargados de la propagación de cambios) para automatizar las operaciones de actualización no se dio sino hasta la versión 7.0.13. En la versión 7.0.12 se contaba con una herramienta externa llamada *refsnap* que leía el usuario, clave de acceso y la frecuencia de las actualizaciones de un archivo externo.

El proceso SNP cambio de nombre a partir de las versiones 8i por Job Queue tiene la misma función pero permite levantar mas de 9 procesos por instancia.

TESIS CON
FALLA DE ORIGEN

La replicación avanzada, que incluye replicación *multimaster* (donde varios nodos pueden actualizar datos) y snapshots modificables apareció en la versión 7.1.6. La versión 7.3 agregó los grupos de replicación y la replicación en tiempo real.

Un grupo de replicación es la agrupación lógica de varios objetos replicados cuyas transacciones deben resolverse al mismo tiempo para garantizar la consistencia de datos.

Con Oracle8 apareció la replicación en paralelo que permite replicar simultáneamente a varios nodos, los triggers de control quedaron internos en el código del RDBMS lo que mejoró su desempeño. Últimas versiones como Oracle8i incluye plantillas de implementación para hacer rápida puesta en producción de nodos de replicación de datos.

Los principales componentes de la maquina de replicación son los siguientes:

- Diccionario de replicación (Replication Catalog)
- RPC's diferidos (Deferred RPC - DRPC)
- Generador de código de replicación (Replication Code Generator RepGen)
 - o Triggers y paquetes
- Definiciones de interfaz de lenguaje C
- Grupos de replicación.

La Figura 3.3 a continuación muestra los Componentes del servicio de replicación del Oracle7 Server y como están relacionados entre ellos.

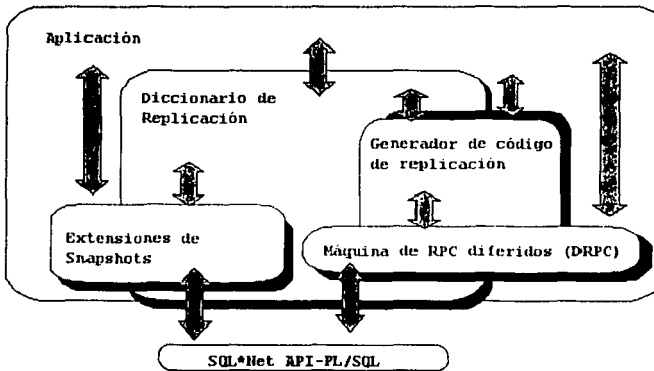


Figura 3.3 Componentes del servicio de replicación del Oracle7 Server

Estos componentes se encuentran presentes en cada nodo Oracle. La interacción entre los componentes del servicio de replicación se inicia en el nodo donde se origina la transacción y es necesario propagar la información al resto de los nodos.

Los datos a ser replicados, junto con otra información de control, son colocados en el SGA; el siguiente paso son llamadas que hace el RDBMS Oracle a otros servicios de SQL*Net para establecer la comunicación con los otros nodos e iniciar la transferencia de los paquetes de datos.

El primer componente es el **Diccionario de replicación**, es un conjunto de tablas del diccionario de datos que son usadas y mantenidas por Oracle para administrar la replicación de datos. Incluye definiciones de todos los objetos replicados, el estado de otras estructuras de control, de las transacciones distribuidas, información de auditoría. El **Apéndice B** muestra las vistas del diccionario de datos que controlan la replicación de datos.

La interacción del DBA con el diccionario de replicación es a través de unos procedimientos de interfaz. Debido a que solamente Oracle puede actualizar la información contenida en las tablas del diccionario, la manera en la que se pueden hacer modificaciones para resolver problemas es con el uso de paquetes especialmente diseñados para ello.

Siendo los más utilizados el **DBMS_REPCAT**, que contiene procedimientos para la administración; el **DBMS_REPCAT_ADMIN**, para el manejo de privilegios y usuarios administradores del ambiente distribuido; **DBMS_REPCAT_INTERNAL**, para administrar y actualizar directamente el diccionario de replicación.

Los siguientes componentes son los **RPC's diferidos (DRPC)** Son usados en una de las configuraciones típicas de replicación. Que está basada en la propagación de datos al nivel de registro y usan replicación diferida de datos.

La replicación diferida ocurre cuando una aplicación actualiza una tabla local, estos cambios se almacenan en una cola, y entonces son enviados al resto de los nodos momentos más tarde. Por esta razón este tipo de replicación es conocido como *store-and-forward*. Para hacer esto Oracle internamente utiliza triggers, transacciones diferidas, colas de transacciones diferidas y los procesos internos **SNP** de base de datos para la propagación. Los triggers son usados para capturar y almacenar lo modificado por la transacción. Otro tipo de triggers construye las llamadas a procedimientos remotos o RPCs para reproducir remotamente los cambios hechos en el nodo local. Estos triggers se encuentran en forma de componentes dentro del mismo ejecutable del RDBMS Oracle.

Los RPCs son almacenados en una cola de transacciones diferidas en los nodos remotos para ejecutarlas posteriormente. La propagación de los cambios se realiza usando **colas de jobs**. Cada nodo participante en un sistema de replicación avanzada tiene una cola local de jobs. La cola de jobs es una tabla que tiene información de las llamadas de código PL/SQL y el momento en el que debería ser ejecutado. Los jobs típicos en un ambiente de replicación pueden ser para envío de transacciones a nodos remotos, para eliminar las transacciones que ya fueron aplicadas, y para la actualización de snapshots.

El siguiente componente es el **Generador de código de replicación (Replication Code Generator)** El objetivo es el construir objetos requeridos para soportar la replicación de datos, nombrar los triggers para la captura de cambios locales y los paquetes para aplicar las transacciones distribuidas.

En Oracle8 y Oracle8i el código de generación también está interno en el ejecutable del RDBMS de Oracle.

Los ICD o definiciones de interfaz C (Interface C definitions), son una interfaz entre PL/SQL y C. Casi todo el trabajo de replicación es realizado por funciones internas del ejecutable Oracle escrito en lenguaje C. La interfaz PL/SQL con la que el usuario puede administrar el ambiente de replicación es solamente una envoltura y un punto de entrada a estas funciones para no tener que integrar comandos de SQL nuevos.

3.3 Arquitectura del SQL*Net.

3.3.1 Papel de SQL*Net en Cliente-Servidor y Servidor-Servidor.

La arquitectura Cliente-Servidor se basa en el intercambio de peticiones y resultados entre cliente y servidor en forma de mensajes. En el ámbito de las bases de datos distribuidas, el intercambio de peticiones SQL puede darse también entre nodos servidores de base de datos.

El SQL*Net es un software basado en la tecnología de Oracle llamada Transparent Network Substrate (TNS), su función es establecer la conexión entre: clientes y servidores; y de servidor a servidor, la última conocida como Servidor-Servidor.

Los pasos para iniciar el intercambio de mensajes en la arquitectura Cliente-Servidor, son:

- A) El cliente envía un SQL al servidor.
- B) Un mensaje es enviado al servidor y el SQL pasado como parámetro.
- C) El servidor recibe la petición y la ejecuta, modifica, crea datos o los recupera.
- D) El resultado en forma de mensaje de éxito o error es enviado a la máquina cliente.
- E) Los resultados son presentados al usuario.

En la comunicación Servidor-Servidor el proceso es muy similar pero el intercambio de las sentencias SQL ocurre entre dos RDBMS y no de forma Aplicación-RDBMS. Cuando la transacción de usuario tiene la necesidad de datos localizados en varios nodos la comunicación Servidor-Servidor se hace internamente entre los nodos involucrados.

El nodo donde se origina la transacción es llamado **SERVIDOR COORDINADOR** el cual inicia el diálogo SQL con los otros nodos donde están los datos. Además de solicitar datos, recibe los resultados; comunica a los nodos participantes el éxito o fracaso de la transacción los que remotamente completan sus transacciones. Enseguida el servidor coordinador presenta el resultado al usuario y completa así la transacción.

El intercambio de mensajes Cliente-Servidor y Servidor-Servidor permiten distribuir el procesamiento en la entidad donde los datos se encuentran. En ambos casos el responsable de la comunicación entre procesos es SQL*Net, que además de establecer esta comunicación envía y recibe mensajes mientras la transacción dura. Al terminar, SQL*Net desconecta ambos procesos.

SQL*Net funciona sobre el protocolo nativo en el que se encuentran el cliente y el servidor, o los servidores según sea el caso. Cuando una acción es requerida por un cliente o servidor, SQL*Net

recibe la petición, y si más de una máquina está involucrada envía la petición al TNS vía el protocolo usado en la red.

En el otro extremo, SQL*Net es responsable de recibir la petición enviada por el TNS y la pasa a la base de datos como un mensaje de la red con los parámetros necesarios.

Con la arquitectura TNS, se tiene que cualquier aplicación desarrollada para trabajar con una base de datos local Oracle puede ser ejecutada sobre otra base de datos Oracle que este usando otro protocolo nativo de comunicación sin tener que hacer ninguna modificación a la aplicación dado que la comunicación es responsabilidad de SQL*Net.

Desde la perspectiva del usuario o de del equipo de desarrollo de aplicaciones la interacción con SQL*Net y el protocolo usado para comunicarse con la base de datos son transparentes.

3.3.2 El nivel del TNS (Transparent Network Substrate)

La tarea de SQL*Net es asegurar que las diferencias entre clientes y servidores en términos de representación de los diversos tipos de datos y parámetros del ambiente de red en el que se encuentran sean resueltas de manera transparente.

La manera en la que se establece la comunicación entre un proceso cliente y un proceso servidor está dada en forma de pila con varios niveles de comunicación relacionados. [NEVF92]

El intercambio entre cada uno de estos niveles incluye:

1. Intercambio de sentencias SQL y datos entre los procesos.
2. Intercambio entre UPI/OPI, que son las siglas para **Interfaz de programación de usuario** (User Programmatic Interface) e **Interfaz de programación Oracle** (Oracle Programmatic Interface) Su tarea es traducir las sentencias SQL y datos en rutinas programadas para validar la información de conexión a la base de datos, ejecución de las sentencias y la recuperación de datos.
3. En el nivel inferior se encuentra el SQL*Net que se encarga de recibir y enviar la información producida por las rutinas en el nivel UPI/OPI; esto lo hace en forma de paquetes de información que serán enviados a través de la red.
4. Por último encontramos el nivel del protocolo de red y es particular para cada una de las plataformas.

El intercambio anterior, en el caso de la comunicación Cliente-Servidor el primer paso se lleva a cabo en el proceso cliente, donde encontramos en el nivel más externo la aplicación del usuario. La aplicación del usuario como su nombre lo dice es el medio, generalmente en forma de interfaz gráfica, por el que se pueden realizar operaciones en la base de datos. La aplicación de usuario identifica cualquier sentencia SQL para ser enviada al RDBMS, esto es llamado UPI.

La **interfaz de programación de usuario (UPI)**, es una pequeña capa de código capaz de dar inicio a la comunicación entre el cliente y el servidor; y sus tareas incluyen:

- Se hace validación de sintaxis a las sentencias SQL.
- Definición de variables de la aplicación del cliente en la memoria compartida de la base de datos.
- Describe el contenido de campos que son regresados por el proceso servidor, basado en las definiciones existentes en el diccionario de datos
- Despliega uno o más registros en la aplicación del usuario.
- Cierra el área en memoria usada para la ejecución de la sentencia SQL.

Algunas de estas actividades requieren actividad en el servidor, y es en este momento cuando el control es pasado al nivel siguiente. Es SQL*Net quien establece y mantiene la conexión o transmite la información al servidor.

De igual manera, pero en el servidor, SQL*Net recibe conexiones a la base de datos provenientes de un proceso llamado **TNS Listener** y pasa entonces el control al servidor de base de datos.

En el nivel inferior se encuentra el **TNS** que recibe las peticiones de SQL*Net y básicamente define la manera de conectarse a nivel máquina, esto incluye el manejo de interrupciones entre cliente y servidor; y la localización del servidor, llamada **TNS Destination**.

Las funciones genéricas realizadas por el **TNS** están basadas en un adaptador de protocolo quien realiza las llamadas específicas al protocolo de red. El adaptador de protocolo realiza una conversión de las funciones del **TNS** a un protocolo estándar de la industria que es usado para la conexión Cliente-Servidor.

En el lado del servidor se llevan a cabo las mismas tareas, se inician en el extremo inferior de la pila con el protocolo y llega hasta la comunicación con el proceso servidor.

La única operación que difiere ambos procesos es la de recibir la conexión inicial. El servidor tiene un proceso llamado "*listener*" (**TNS Listener**) que revisa si hay nuevas conexiones y define cual es su destino. Un mismo listener puede dirigir las conexiones a varias bases de datos en el mismo servidor.

El **TNS Listener** es un proceso que recibe peticiones de conexión de cualquier aplicación que use la arquitectura **TNS**, sin importar el protocolo usado donde la conexión se origina. Las conexiones de clientes SQL*Net son identificadas por el **TNS Listener** por tener definido un puerto particular de entrada.

Del lado del servidor, existe la **Interfaz de programación Oracle (OPI)** que funciona como el **UPI** pero esta orientado a responder a las peticiones hechas por los clientes.

En el nivel superior en la pila del servidor (**OPI**) se tiene al **RDBMS** quien ejecuta el código recibido del cliente. Al completar la transacción el **RDBMS** los envía al **OPI** para que sean preparados y sean enviados a la aplicación del cliente.

En el caso de comunicación Servidor-Servidor, para hacer posible una transacción distribuida se sigue el mismo proceso seguido en el Cliente-Servidor, con la excepción que no se tiene aplicación de usuario.

El servidor tiene una versión propia de UPI llamada **Interfaz de programación de red (NPI)** (Network Program Interface) La interfaz NPI lleva a cabo todas las tareas que hace el UPI para un proceso cliente, convirtiendo al servidor que la ejecuta en coordinador de la transacción distribuida lo que implica que construye la sentencia SQL y envía la petición al servidor o servidores que estén involucrados.

La **Figura 3.4** a continuación, muestra las **Pilas de TNS usadas para el intercambio de mensajes entre procesos cliente y servidor**. Muestra las dos pilas formadas y la interfaz UPI del lado del cliente y su correspondiente OPI del lado del servidor y como se comunican los diferentes niveles de ambas pilas.

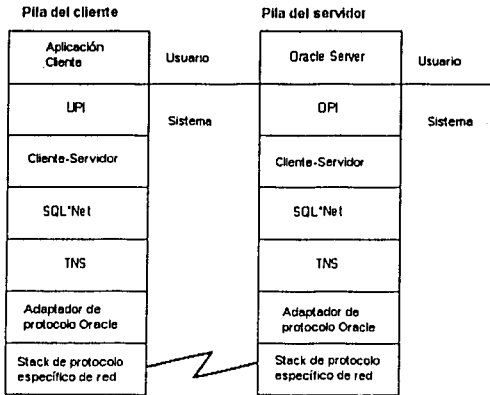


Figura 4.3 Pilas de TNS de intercambio de mensajes entre procesos cliente y servidor

En un ambiente de base de datos distribuida, la importancia del SQL*Net es vital pues permite la comunicación entre las aplicaciones del cliente y el nodo donde la base de datos reside. Asegurando:

Transparencia en la red de comunicación, pues permite la conexión entre diferentes protocolos de red independientemente de la aplicación.

Independiente del protocolo usado, pues cualquier aplicación puede ejecutarse en cualquier protocolo de red.

Independiente de la topología de red, puede usar cualquier método de transmisión de datos.

TESIS CON
FALLA DE ORIGEN

Permite la transparencia de ubicación, los objetos lógicos de la base de datos parecen ser locales aún cuando físicamente residen en un nodo remoto. SQL*Net resuelve la localización de los objetos.

*Con la introducción de las versiones Oracle8 y posteriores, SQL*Net se convirtió en Net8. Que tiene las mismas características pero agrega mas servicios de red pues permite la implementación del procesamiento distribuido en ambientes heterogéneos entre diferentes nodos sin importar el proveedor, sistema operativo o la arquitectura de hardware. Soporta además otras tecnologías Internet como el IIOP, la conectividad con aplicaciones Java y soporte para la encriptación de datos usando algoritmos estándar de la industria.*

Capítulo IV.

Replicación de datos.

4.1 Introducción a la replicación de datos.

4.1.1 Definición de replicación de datos.

Por el tamaño de muchas organizaciones, la cantidad de usuarios y procesos, ha sido necesario distribuir los repositorios de datos usando diversas técnicas para la implementación de una base de datos distribuida.

La replicación es el proceso que permite la existencia de copias redundantes de datos dispersas en diversos nodos y asegura que las copias son totalmente consistentes entre ellas; pero de igual manera puede permitir un cierto nivel de inconsistencia por cortos periodos de tiempo, siempre que la aplicación lo permita. [BURE97]

La replicación incluye también procesos y otros objetos de la base de datos; debe considerar la implantación y el mantenimiento del proceso de copiado.

En esencia, replicación es un servicio de administración de copiado de datos en un ambiente distribuido, el servicio de replicación de datos debe tener la siguiente funcionalidad:

- *Escalable*, que significa ser capaz de replicar pequeños y grandes volúmenes de datos entre diferentes manejadores de base de datos.
- *Transformación de datos y servicios de conversión*, al permitir que el esquema origen y el de destino sean diferentes, conteniendo la misma información pero existiendo en diferentes plataformas y posiblemente con distintos tipos de datos.
- *Permitir la replicación además de datos de otros objetos*, como los son procedimientos almacenados o triggers de base de datos.
- *Soportar la replicación sincrónica (tiempo real) y asincrónica (replicación diferida)*
- *Proveer un mecanismo de registro de errores (log de actividad) que registre cualquier intento fallido de replicación*, lo que permitiría contar con un método de recuperación.

Un servicio de replicación es necesario cuando las organizaciones requieren que copias de datos en un ambiente distribuido se mantengan consistentes. Las razones para mantener copias redundantes de datos son:

- (1) Las organizaciones utilizan diferentes plataformas de hardware y software;
- (2) Las organizaciones por naturaleza son distribuidas;
- (3) Las organizaciones requieren alternativas de recuperación de desastres.

Un beneficio directo al implementar replicación de datos es la mejora en el desempeño de la aplicación pues los datos están mas cerca del lugar donde se necesita por lo que explotarlos de manera remota no es necesario, esto se hace más evidente cuando la distribución geográfica de los nodos es amplia y se encuentran en una red WAN. Esto trae un segundo beneficio, los costos de red disminuyen. Sin embargo, se debe considerar que con la replicación se requiere de mayor administración y mantenimiento.

La replicación de datos se clasifica de la siguiente manera. **Replicación sincrónica o en tiempo real y Replicación asincrónica o diferida.**

Replicación sincrónica.

Provee alta consistencia de datos entre los diferentes nodos. Lo que significa que el tiempo de latencia antes de alcanzar la consistencia de los mismos es prácticamente cero. Los datos en los nodos replicados son siempre los mismos, sin importar en donde fue originado el cambio. Para esto el RDBMS debe ser capaz de manejar la transacción distribuida como si se tratara de una transacción local. Todo el proceso de actualización ocurre dentro de la misma unidad de trabajo, las transacciones mantienen las propiedades ACID usando el mecanismo de *two-phase commit*.

Por su naturaleza, algunos factores a considerar en el diseño e infraestructura necesaria para implementar la replicación en tiempo real son:

- *Asegurar que el diseño de las transacciones refleje una unidad básica de recuperación.* Que es la manera en la que un proceso del negocio se transforma en su correspondiente transacción de base de datos ya que así se delimita la unidad básica de recuperación en caso de fallas.
- *Tener un buen soporte de infraestructura.* La actualización de transacciones en múltiples nodos depende de la confiabilidad, disponibilidad y eficiencia de los componentes que forman el ambiente distribuido.
- *Mantener el número de manejadores de base de datos razonablemente limitado.* El número y ubicación de las bases de datos no deben exceder la capacidad de los componentes de infraestructura.

Replicación asincrónica.

Genera consistencia de datos no tan estricta. Lo que significa que el tiempo de latencia antes de alcanzar la consistencia en todos los nodos es siempre mayor a cero. El proceso de replicación a los diferentes nodos siempre ocurre después de que la transacción es completada en el nodo donde se origina, es decir, siempre hay un retraso antes de poder ver los cambios hechos por la transacción en el resto de las replicas. Además es posible controlar la manera en la que son replicados.

La propagación de los datos puede hacerse:

- *Actualización completa o por incrementos.* La actualización completa de datos toma secciones de la fuente primaria de datos que son replicados a los nodos. Estas actualizaciones se hacen por ciertos periodos de tiempo o basándose en cambios hechos a la fuente primaria. En un refresco por incrementos, una primera carga de los datos de la fuente primaria se realiza en el resto de los nodos, para mantenerlas actualizadas solamente se replican los cambios posteriores a la primera carga.
- *Propagación delta de eventos.* Puede alcanzar el procesamiento de las transacciones casi en tiempo real. Se implementa usando de comunicación Servidor-Servidor. La propagación delta requiere la iniciación de las replicas para dejarlas consistentes con

la fuente primaria. Una vez realizada la carga, solamente los eventos que actualizan datos son reenviados a las replicas generalmente usando paquetes integrados en el RDBMS. El proceso de replicación diferida de forma Servidor-Servidor esta formado por cuatro componentes:

- (1) Recolección de datos de la fuente primaria de datos;
- (2) El proceso de distribución de cambios a las replicas;
- (3) Proceso de actualización hacia la replica; y
- (4) Proceso de monitoreo de los tres procesos anteriores. La replicación diferida Servidor-Servidor es implementado como parte de la aplicación y se basa en los mensajes que se intercambian entre los API's utilizados.

La replicación diferida debe considerar varios aspectos para su uso, destacando:

- *Requerimientos de semántica e integridad de las transacciones.* Debido a que en este tipo de replicación las transacciones no se replican en tiempo real, el orden en el que se realizan las transacciones debe ser vigilado pues podría ser causa de errores de integridad al intentar actualizar las replicas.
- *Utilización de mecanismos para la detección de conflictos y errores.* En caso de existir errores en los datos replicados y como las transacciones ya fueron completadas en los nodos donde se origina la transacción. Es necesario identificar las herramientas (propias o provistas por el RDBMS) para detectar y resolver conflictos en las transacciones.

4.1.2 Modelos de replicación asincrónica o diferida.

El uso de la replicación diferida representa un menor costo de administración y uso de recursos más relajado. Es conveniente siempre que los requerimientos del negocio no demandan tener consistencia de datos en tiempo real. Los diferentes modelos de replicación asincrónica de datos pueden clasificarse en:

Replicación simple en un sentido.

Este modelo ilustra la replicación en un sentido usando el modelo maestro/esclavo. Aquí, la replicación de una fuente primaria de datos (donde se capturan y actualiza la información) es replicada a varias copias que son de sólo lectura. Esto se puede dar también en sentido inverso donde varios nodos primarios replican la información a una sola base de datos consolidada que es de sólo lectura.

El modelo maestro/esclavo más sencillo es aquel usado para la distribución de datos, donde la fuente primaria de datos se encuentra en un nodo único y la replicación a diferentes nodos se hace en un sentido. Las actividades de actualización de datos solamente ocurren en el nodo maestro (fuente primaria); los cambios son entonces propagados de manera diferida a varias replicas. Los nodos esclavo pueden estar localizados en el mismo nodo servidor de base de datos o en uno remoto.

Otra variante del maestro/esclavo es usada para consolidación de datos. Diferentes fuentes primarias de datos replican las actualizaciones hacia una base de datos centralizada de sólo lectura. La fragmentación de las fuentes primarias usualmente se hace de manera vertical y los cambios son replicados de manera diferida a la base de datos consolidada cada vez que estos ocurren en las fuentes primarias.

Replicación compleja.

Es conocido como **actualización en todas partes**, no existe un nodo maestro o fuente primaria designada. Cualquier nodo replicado puede ser usado como la fuente de datos en cualquier momento. En la replicación compleja los cambios podrían propagarse en tiempo real o de manera diferida. Si se usa de en tiempo real el protocolo *two-phase commit* vuelve a usarse y los datos son siempre consistentes. Oracle7 define esto como **Replicación Simétrica**.

Por el contrario, si los cambios se propagan de manera diferida puede traer por un lado muy buen desempeño pero a costo de sacrificar y poner en riesgo la consistencia de los datos. Los problemas para mantener la integridad de datos se multiplican pues pueden generarse conflictos entre tablas, y con los datos en la tabla misma. Es imposible detectar problemas de inconsistencia de datos cuando los mismos son almacenados de manera redundante y esas copias pueden ser consideradas fuentes primarias. Esta variante se recomienda para aplicaciones que tienen un volumen muy bajo de transacciones y el total de nodos a replicar es poco.

Una variante de replicación compleja con propagación diferida es llamada **maestro/esclavo con fragmentos primarios distribuidos**. Aquí se realiza replicación en dos sentidos donde cada nodo actúa como emisor (fuente primaria) y receptor (réplica) de datos. Puede actualizar datos de un fragmento particular (vertical) para que sean replicados al resto de los nodos, además de recibir los cambios de otros fragmentos actualizados en otros nodos solamente para lectura.

Este modelo es utilizado cuando las actualizaciones son permitidas en todos los nodos del ambiente distribuido pero que están claramente restringidas las secciones de información que son responsabilidad del nodo local, al momento de hacer consultas o generar reportes requieren la inclusión total de los datos del resto de los nodos. La complejidad de este modelo no es tanto su implementación pero sí en su mantenimiento. La recuperación y consolidación de los datos después de una falla son complejas, la recuperación ocurre al nivel relacional de la tabla pero la consolidación de los datos replicados se debe realizar al nivel de registro entre todas las fuentes primarias de datos.

Otra variante es el tipo *pass-the-book*, utiliza replicación asincrónica, múltiples replicas existen, el nodo donde se realizan los cambios es migrado entre las diferentes replicas. El criterio de migración es frecuentemente un período de tiempo durante el día cuando un nodo en particular está en operación. Las características del modelo incluyen:

- Cada nodo tiene un período de tiempo en el que asume la responsabilidad de ser la fuente primaria de datos.
- Cada nodo debe tener los privilegios necesarios para actualizar información en el resto de los nodos mientras actúa como fuente primaria.
- El diseño de la aplicación debe asegurar que solamente existe una fuente primaria en un determinado momento.

- Si existe la posibilidad de definir un lapso para consolidar los cambios antes de hacer la migración de la fuente primaria, todos los cambios que están pendientes deben ser aplicados.

Implica además incorporar una tabla "nodo_info" con información del nodo que es el dueño de la información. La tabla debe existir en todas las replicas y debe ser replicada. Los cambios a la columna "dueño" son replicados a todos los nodos.

4.1.3 Tecnologías de replicación de datos.

Antes de entrar a la revisión detallada de la replicación de datos en Oracle, se revisaran de manera breve los enfoques usados por otros RDBMS, que tienen diferentes arquitecturas para implementar la replicación de datos.

Las diferencias pueden caer en tres diferentes áreas: el papel del RDBMS, soporte de replicación basada en transacciones o basada en tablas, y cuales son los tipos de replicación soportados.

Se describen las implementaciones llamadas *Replication Server* (Sybase) y *DataPropagator Relational DPropR* (IBM)

DataPropagator Relational (IBM)

DPropR soporta replicación de datos en ambientes heterogéneos de bases de datos relacionales. Utiliza el modelo maestro/esclavo y define que cada pieza de información tiene solamente una fuente primaria. Todo el proceso de replicación es realizado de manera asíncrona. Soporta dos tipos de replicación completa y por incrementos.

DPropR esta formado por dos componentes principales, uno de captura y otro de aplicación de cambios, las funciones de cada uno de ellos son las siguientes:

- Captura, componente que toma los cambios en el nodo primario y los salva en tablas especiales. Este producto es independiente del RDBMS y son una interfaz a los archivos de log que registran los cambios en DB2. DPropR tiene otras interfaces para soportar la replicación con otras fuentes de datos distintas a IBM.
- Aplicación, la función de este componente es propagar los cambios a las replicas. Soporta todas las bases de datos DB2 que permiten la captura de cambios.

DPropR esta compuesto por tres servidores lógicos, incluyendo Data Server, Copy Server y Control Server. El Data Server esta localizado en el RDBMS marcado como fuente primaria, contiene las tablas con la información a replicar, y contiene varias tablas de control. El Copy Server se localiza en los nodos marcados como replicas. Contiene las tablas destino, tablas de cambios que almacenan datos recibidos del componente de captura antes de ser actualizados en la base de datos replicada, además de tablas de control. El Control Server puede estar en tanto en la fuente primaria como en los nodos replicados y básicamente maneja las tablas de control encargadas de los procesos de captura y aplicación de cambios.

Replication Server (Sybase)

Soporta la replicación de datos y procedimientos almacenados en un ambiente de base de datos heterogéneo. Usa el modelo maestro/esclavo donde cada pieza de datos tendrá una fuente primaria de datos en un momento dado. Mantiene la semántica de las transacciones al usar tres procesos que son de captura, distribución y aplicación.

La arquitectura de los procesos internos del Replication Server, funciona de la manera siguiente:

- Replication Server (RS) Provee el servicio de distribución y coordina las actividades de replicación. Recibe los cambios hechos de otros nodos usando algo llamado Log Transfer Manager. De igual manera los distribuye a otros Replication Servers. Recibe y distribuye cambios de fuentes primarias, y recibe y aplica hacia nodos replicados.
- Replication Server System Database (RSSD) Es el servidor SQL que contiene las tablas internas del RS. Cada RS tiene su propio RSSD.
- Replication Agent/Log Transfer Manager (LTM) Su función es notificar al RS de cualquier acción que debe ser replicada a otros RDBMS.

4.2 Snapshots de sólo lectura y actualizables.

4.2.1 Definición de snapshots de sólo lectura.

El mecanismo básico del Oracle Server para la implantación de bases de datos distribuidas que usan replicación es el snapshot.

La replicación maestro/esclavo en un sentido es el método más sencillo. Oracle implementa este tipo de replicación en la forma de los **snapshots de sólo lectura** (o read-only) que se definen de la siguiente manera:

Snapshots de sólo lectura.

Estructura que implementa la replicación básica de un sentido. Se define como una copia en un momento en el tiempo de una tabla maestro, o de parte de la misma. *En las versiones posteriores a Oracle8 se incluye el concepto de vista materializada que es sinónimo de snapshot.*

El snapshot está basado en una sentencia SQL de SELECT que hace referencia a una o más tablas, vistas u otros snapshots. Las tablas usadas son llamadas tablas base y la base de datos donde se encuentran se convierte en el **nodo maestro** (o master site) El nodo donde el snapshot se crea es llamado **nodo de snapshot** (o snapshot site)

La Figura 4.1 muestra la **replicación de datos usando snapshots**. Describe como la tabla en el nodo maestro sirve de base al snapshot que es creado en el nodo remoto. El snapshot es entonces usado como si se tratase de una tabla local y se crea basándose en un DML ejecutado sobre la tabla o tablas del nodo maestro.

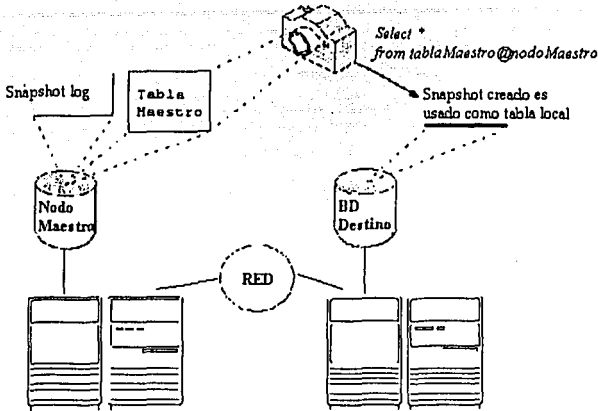


Figura 4.1 Replicación de datos usando snapshots.

Un snapshot de sólo lectura por definición no puede tener conflictos de propagación de datos por problemas en el nodo maestro pues la integridad es responsabilidad del RDBMS remoto. La manera de crear un snapshot es la siguiente:

```
CREATE SNAPSHOT CLIENTES AS Select * from clientes@nodoRemoto;
```

El DBA del nodo de snapshot ejecuta la sentencia para crear un snapshot sobre la tabla base *clientes*, para ello debe tener un dblink al nodo maestro, que en el ejemplo se llama *nodoRemoto*. El snapshot *CLIENTES* se crea en el mismo nodo de snapshot y puede ser usado como una tabla común por el resto de los usuarios.

Internamente Oracle crea otros objetos de control en el esquema donde el snapshot es creado, que no deben ser alterados o modificados por el usuario. En el nodo de snapshots crea una tabla base llamada *SNAPS_nombre-del-snapshot* para almacenar los registros que fueron propagados de la tabla base. Además crea un índice usando el ROWID de la tabla base llamado *I_SNAPS_nombre-del-snapshot*.

En el ejemplo anterior, snapshot *CLIENTES*, el objeto que los usuarios pueden usar es en realidad una vista. Una segunda vista se crea en el nodo maestro llamada *MVIEWS_nombre-de-snapshot* que es usada para propagar los cambios de la tabla base al nodo de snapshots.

Si el snapshot se forma de datos de una sola tabla base y no tiene funciones de grupo en el DML que lo define se dice que es un *snapshot simple*. De igual manera, si los datos se toman de más de una tabla y tiene funciones de grupo es llamado *snapshot complejo*.

La principal desventaja de los snapshots de sólo lectura es la no-posibilidad de actualizar los datos desde el nodo de snapshots. Pero las ventajas en una base de datos distribuida son:

- Consultas sobre los snapshots locales. Buen tiempo de acceso y carga mínima de la red pues los datos no tienen que ser transportados a través de la red.
- Si el nodo maestro no está disponible, por ejemplo por una falla de red, los datos en las copias de sólo lectura pueden seguir siendo utilizadas.
- Puede servir para proteger información confidencial pues permite limitar los datos que contendrá el snapshot.

Una estructura adicional relacionada a los snapshots es el llamado **SNAPSHOT LOG** que como su nombre lo indica registra todos los cambios hechos a la tabla base y es usado para la propagación de cambios hacia el snapshot y se crea en el mismo nodo maestro. Retomando el ejemplo anterior, se crearía de la manera siguiente:

```
CREATE SNAPSHOT LOG on CLIENTES;
```

No es posible asignar nombre al snapshot, es asignado por Oracle. Internamente se crean las estructuras: tabla llamada `MLOG$_nombre-de-tabla-maestra` que almacena el ROWID y el momento en que fueron actualizados. El contenido en el log es eliminado después de que son propagados al nodo de snapshots. Además de la tabla, se crea un trigger llamado `TLOG$_nombre-de-la-tabla-maestro` que se ejecuta después de que un registro es borrado, actualizado o creado en la tabla base para registrarlos en el snapshot log. Al igual que las otras estructuras creadas con el snapshot, el usuario no debe alterar o borrar registros de ellos.

Las vistas del diccionario de datos donde puede revisarse información de los snapshots y snapshot logs son `DBA_SNAPSHOTS` y `DBA_SNAPSHOTLOGS`.

Oracle cuenta con dos unidades lógicas para administrar y organizar los snapshots en un nodo de snapshot. Estas son:

Grupos de snapshots (*Snapshot group*) Permite propagar los cambios hacia todos los snapshots de un mismo grupo y forman el mismo grupo de tablas base en el nodo maestro. Con esto se mantiene la consistencia de las transacciones entre las copias de varias tablas base que tienen reglas de integridad referencial entre ellas.

Grupos de actualización (*Refresh groups*) Permite propagar cambios a snapshots que pertenecen a otro grupo de snapshots. Así, selectos snapshots pueden ser actualizados dentro del proceso de propagación de su grupo; además, por reglas de integridad, si son parte de un grupo de actualización.

Un grupo de actualización es una estructura lógica más completa, es creada con el API provisto por Oracle llamado `DBMS_REFRESH` que es uno de varios paquetes para la administración del ambiente replicado. El grupo de actualización permite propagar cambios de manera automática y son creados en los nodos donde se encuentran los snapshots.

Para que la propagación automática pueda realizarse uno o varios procesos internos `SNPn` deben ser levantados con la instancia pues son los encargados de revisar los cambios hechos a las tablas base y propagarlos hacia los nodos de snapshots. [BOCH93]

Los procedimientos contenidos en el paquete DBMS_REFRESH son los siguientes:

- **MAKE**, para crear grupos de actualización. Recibe como parámetros el nombre, los miembros (snapshots), cuando se hará la primer actualización e intervalos de propagación.
- **ADD**, permite agregar nuevos miembros a un grupo ya existente.
- **SUBSTRACT**, si es que se necesita remover un miembro del grupo.
- **CHANGE**, si se quiere cambiar frecuencia de actualización de los snapshots del grupo.
- **DESTROY**, si se quieren eliminar todos los snapshots del grupo. Al destruir un grupo sus miembros ya no serán actualizados de manera automática. Si se hace esto los snapshots deben actualizarse manualmente o agregarlos a otro grupo.

Frecuencia de propagación de cambios.

Como se definió, la propagación de cambios hacia los nodos replicados puede hacerse de manera total o por incrementos. Oracle puede hacerlo de ambas formas. Es llamado **REFRESH COMPLETE** para la propagación total y **FAST REFRESH** para la propagación por incrementos.

La manera en la que se actualizan los datos es definida al momento de la creación del snapshot. El ejemplo anterior del snapshot CLIENTES podría quedar:

```
CREATE SNAPSHOT CLIENTES AS Select * from clientes@nodoRemoto  
REFRESH COMPLETE;
```

Lo que significa que el snapshot será actualizado en su totalidad, primero borrando todos los registros y después actualizando el contenido en la tabla maestro. De igual manera una actualización por incrementos se especifica como:

```
CREATE SNAPSHOT CLIENTES AS Select * from clientes@nodoRemoto  
REFRESH FAST;
```

Con esto se indica que después de la primer actualización (que siempre es completa) del snapshot, las siguientes actualizaciones se harán tomando la información registrada en el snapshot log creado para la tabla base. Para esto debe existir obviamente el log.

De no especificarse el método de refrescamiento Oracle intentara primero hacerlo por incrementos, de no lograrse realizaría el completo.

El borrar todos los datos de la tabla base hará que el snapshot sea actualizado de manera completa la siguiente ocasión. De igual manera, si la tabla base es eliminada, usando un comando DDL, el snapshot prevalece y puede seguir siendo utilizado. En el nodo maestro al borrarse la tabla también se elimina el snapshot log. Oracle generará un error solamente si se intenta actualizar un snapshot basado en una tabla no existente.

Si la tabla base es recreada después del snapshot, este puede ser nuevamente actualizado siempre que el SELECT que lo define pueda ejecutarse contra la nueva tabla. La actualización por incrementos no será posible hasta que no se reconstruya el snapshot log. Aún con esto es posible que el snapshot no pueda ser actualizado. De pasar esto se tendrá que recrear el snapshot.

El snapshot log es una tabla en la misma base de datos donde reside la tabla base. Los registros en el snapshot log contienen los cambios que han sido hechos a la tabla base, e información de que snapshots han sido actualizados y cuales no con estos cambios.

Una diferencia entre las versiones Oracle7 y Oracle8 es que el volumen de datos generado por la propagación de cambios es menor lo que significa un mejor desempeño. En Oracle7 el registro entero con los valores del antes y después era propagado. Con Oracle8 solamente los valores cambiados son incluidos en los registros del snapshot log.

4.2.2 Snapshots actualizables.

Los snapshots actualizables son parte de la opción de Replicación Avanzada que es usada para implementar ambientes de replicación de *actualización en todas partes* o *simétrica*.

La opción de replicación avanzada es un componente adicional que se instala además del RDBMS, y una vez instalado trabaja en coordinación con el código de Oracle. La replicación avanzada de Oracle permite la replicación de tablas, vistas, índices, triggers y paquetes.

La replicación simétrica puede implementar dos variantes:

Replicación de nodos de snapshots.

(Ambiente de replicación con snapshots actualizables)

Cuando solamente un grupo de objetos se replica de un nodo maestro a un nodo de snapshot. Pero los objetos replicados pueden ser actualizados.

Replicación de nodos maestro o Multi-Master.

(Ambiente de replicación con Múltiples Nodos Maestro)

Cuando todos los nodos maestro participantes en el ambiente distribuido tienen los mismos objetos replicados y todos pueden actualizarlos. Cada nodo maestro propaga los cambios al resto de los nodos en su grupo de replicación.

Snapshots actualizables.

Como su nombre lo dice son snapshots que pueden ser actualizados y estos cambios propagados hacia el nodo maestro, esto es la única diferencia con los snapshots de sólo lectura.

Además de los objetos creados con los de sólo lectura, en el nodo de snapshots se crea una tabla de log llamada `USLOG$_nombre-de-snapshot` que almacena el ROWID y momento en que los registros en el snapshot son actualizados. El segundo objeto es un trigger llamado `USTRG$_nombre-de-snapshot` sobre la tabla creada con el snapshot se dispara después de hacer cambios con DML y actualiza la tabla de log del snapshot.

En un ambiente de replicación avanzada, un **OBJETO REPLICADO** es un objeto de la base de datos que es copiado a múltiples nodos en el sistema distribuido. Los grupos de replicación son la unidad básica de control de la replicación avanzada, y son creados por los administradores del servicio de replicación para todos aquellos objetos que están asociados a una sola aplicación. Los objetos replicados pueden formar parte solamente de un grupo de replicación.

Cuando el snapshot actualizable es creado como un *objeto replicado*, Oracle creará un trigger adicional y un paquete sobre la tabla creada con el snapshot que harán las llamadas RPC en el nodo maestro para propagar cambios. Para hacer esto, se debe usar igualmente un API incluido en Oracle llamado **DBMS_REPCAT.CREATE_SNAPSHOT_REOBJECT**.

De no usarse el API para crear el snapshot actualizable, al propagarse los cambios del nodo maestro, los cambios hechos en el snapshot serán perdidos. [PRAT95]

De esta manera si los cambios se propagan en tiempo real, el trigger ejecutara los procedimientos en el nodo maestro para completar la transacción usando *two-phase commit*. Si se elige hacerlo de manera diferida, el trigger registrara los cambios en una cola de transacciones especial en el nodo de snapshots para aplicar los cambios después hacia el nodo maestro.

Al usarse propagación diferida, los cambios del snapshot hacia la tabla maestro se hacen comparando las columnas de tiempo de actualización de cada renglón en las tablas de log creadas para la tabla base y para el snapshot.

Sí encuentra diferencias porque la tabla base se actualizo antes de propagar los cambios hechos en el snapshot, los **métodos de solución de conflictos** incluidos en Oracle pueden resolver el problema siempre que se definiera alguno en el nodo maestro quien debe definir como se resolverá un conflicto en caso de presentarse. Los criterios que podrían ser definidos son:

1. Manteniendo el valor anterior en el snapshot (versión anterior en el snapshot)
2. Cambiando por el nuevo valor del registro en el snapshot (valor posterior a la última actualización)
3. Tomando los valores actuales de la tabla base (valor de ultima actualización hecha en el nodo maestro)

Estos criterios son definidos por el administrador de replicación y los conflictos no ocurren cuando los cambios se propagan desde el nodo maestro al snapshot.

La manera de crear el snapshot es con el comando **CREATE SNAPSHOT** pero incluye la cláusula **FOR UPDATE**. Siguiendo con el ejemplo del snapshot *CLIENTES*:

```
CREATE SNAPSHOT CLIENTES FOR UPDATE  
AS Select * from clientes@nodoRemoto;
```

Los nodos de snapshots deben tener un nodo maestro asociado (que puede cambiar en cualquier momento), y a diferencia de un nodo maestro, los nodos de snapshots solamente pueden recibir cambios de su maestro asociado. Antes de crear un nodo de snapshot debe existir el nodo maestro.

4.2.3 Mecanismo para propagación de datos.

El proceso de propagación es la esencia de la replicación de Oracle pues es el mecanismo usado para enviar y distribuir cualquier cambio al resto de los nodos.

Los triggers internos capturan todos los cambios hechos con DML's sobre el objeto replicado y estos pueden ser propagados al nodo destino.

Los cambios pueden realizarse de manera diferida que es llamada **replicación store-and-forward** o **asincrónica**, donde el proceso de replicación almacena los cambios en una cola de transacciones y posteriormente envía estas **transacciones diferidas** a los diferentes nodos. Esto pasa como sigue en el nodo maestro:

- (1) Al hacer un cambio local, el trigger genera una llamada a un procedimiento remoto para ser ejecutado vía (RPC)
- (2) El RPC es almacenado en una cola de transacciones diferidas (vista del diccionario de datos *deftran*)
- (3) En una segunda tabla (vista *repsites*) se almacenan la ubicación de los nodos de replicación hacia donde se tienen que propagar los cambios.
- (4) Los registros en la tabla de transacciones diferidas son removidos hasta que la transacción se ha propagado a todos los nodos de replicación.

La propagación posteriormente se hace en ciertos intervalos de tiempo o manualmente por el administrador usando uno de los APT's DBMS_REFRESH o DBMS_SNAPSHOT.

De igual manera, los cambios pueden propagarse en tiempo real, que recibe el nombre de **propagación sincrónica**. Las transacciones que son de tiempo real funcionan de manera similar pero no existe la cola de transacciones diferidas. Al usar el *two-phase commit*, el cambio debe ser exitoso en todos los nodos y son los triggers internos los encargados de colocar los candados en los objetos replicados, mismos que son removidos al terminar la transacción.

La **Figura 4.2** muestra el proceso de **propagación sincrónica** entre dos nodos replicados. Los mismos triggers usados para capturar los cambios de transacciones asincrónicas hacen llamadas RPC a los procedimientos en el nodo destino y hace propagación en la misma transacción.

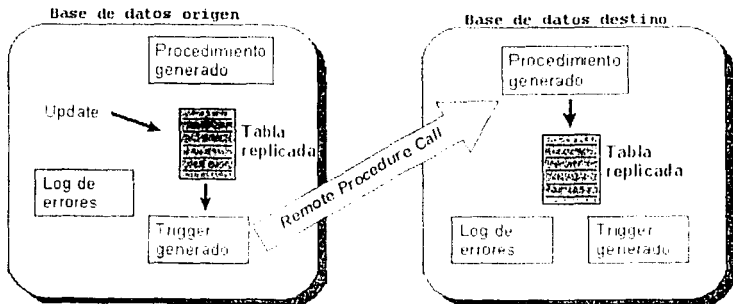


Figura 4.2 Proceso de propagación sincrónica.

La diferencia entonces entre los dos mecanismos es que en replicación *store-and-forward* el trigger de captura registra los cambios en la base de datos origen en una cola de transacciones y después propaga lo que existe en esa cola a la base destino con los procesos *SNPn* que están corriendo en misma instancia.

4.3 Replicación simétrica y solución de conflictos.

4.3.1 Creación de nodos maestro para replicación Multi-master.

La replicación de nodos maestro permite copiar tablas, vistas, sinónimos, triggers y paquetes. Todos los nodos maestro que están en un ambiente de replicación simétrica deben tener los mismos objetos replicados.

Durante la etapa de diseño del ambiente distribuido se define si se replicaran datos o distribuirán. En caso de replicarse, se debe elegir si se replicaran todos los objetos de la aplicación a los diferentes nodos y además si será permitido hacer actualizaciones desde cada uno de ellos.

Ya con la función de cada nodo definida e identificados los objetos que se van a replicar se deben tener privilegios de creación de estos objetos en los nodos remotos. Verificar que la comunicación entre los nodos es posible para la creación de los dlinks.

Existen tres categorías de usuarios en un ambiente de replicación.

1. **Administrador de replicación** (*Replication administrator*) Usuario que es responsable de la configuración y mantenimiento del ambiente de replicación.
2. **Opción de replicación simétrica.** Algunas de las actividades realizadas al replicar datos son ejecutadas como usuario **SYS** en los nodos remotos.
3. **Usuarios finales.** Usuarios que consultan y modifican los objetos replicados.

Un administrador de replicación se encarga de configurar el ambiente de replicación. Es posible tener un solo administrador para todos los objetos en el nodo, pero también es posible tener un administrador por esquema. Todas las tareas administrativas deben ejecutarse como usuario administrador de replicación.

La manera en la que se crea el usuario administrador es con el usuario **SYS** y se hace utilizando el paquete **DBMS_REPCAT_ADMIN**. Para crear un administrador para los objetos de un esquema se hace de la siguiente manera:

```
CREATE_REPCAT_ADMIN.GRANT_ADMIN_REPGROUP(userid => 'mac');
```

Esto dará los privilegios necesarios para que el usuario *mac* administre todos los objetos de replicación que se encuentran en su propio esquema. Para crear un administrador global de replicación para todos los grupos en el nodo se usa el siguiente comando:

```
DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_REPGROUP(userid => 'repadmin');
```

Con esto ahora el usuario *repadim* es capaz de administrar todos los objetos de replicación existentes en el nodo. Para quitar los privilegios a estos usuarios existen otros procedimientos llamados *REVOKE_ADMIN_REPGROUP* y *REVOKE_ADMIN_ANY_REPGROUP*.

La comunicación entre los nodos es con *dblinks*, son definidos por el usuario administrador (deben ser privados, es decir en su propio esquema) hacia el resto de los nodos, usando el nombre de usuario y clave de acceso del administrador de replicación en el nodo remoto.

Todas las tareas administrativas del ambiente de replicación deben hacerse con el paquete *DBMS_REPCAT* y solamente puede ser ejecutado por aquellos usuarios con privilegios de administración de replicación.

Algunas de estas tareas son realizadas por la opción de Replicación Simétrica de Oracle y deben ser ejecutadas como el usuario *SYS* del nodo remoto, por cuestiones de seguridad no se recomienda crear *dblinks* usando el usuario *SYS*, para ello se crea un administrador de replicación sustituto (*surrogate replication administrator*) que lleva a cabo estas tareas en el nodo remoto. Los permisos para el administrador sustituto se otorgan usando el paquete *DBMS_REPCAT_ADMIN* de la siguiente manera:

```
DBMS_REPCAT_ADMIN.GRANT_SURROGATE_REPCAT(userid => 'mac');
```

Al crear los *dblinks* para la opción de replicación simétrica se debe utilizar el usuario con privilegios de administrador sustituto.

Después de haber creado los usuarios administradores y *dblinks* para comunicar los nodos se puede comenzar la creación del ambiente de replicación multi-master.

Usando el API *DBMS_REPCAT*, las siguientes tareas que el administrador de replicación realiza son:

1. Seleccionar el nodo que será el **nodo maestro de definición** (master definition site) desde donde se harán todas las tareas administrativas al resto de los nodos, idealmente debe ser el nodo con menos probabilidad de quedar fuera de línea.
2. Crear un grupo de replicación vacío en el nodo maestro de definición:
DBMS_REPCAT.CREATE_MASTER_REPGROUP
3. Por cada objeto, tabla o procedimiento, que se quiera incluir en un grupo de replicación se debe realizar:
 - ✓ Agregar el objeto al grupo de replicación usando el procedimiento
CREATE_MASTER_REPOBJECT
 - ✓ Si el objeto es una tabla que requiere de alguna rutina de solución de conflictos incluirlo con el procedimiento:
ADD_conflicttype_RESOLUTION
 - ✓ Para cada objeto que requiere ser replicado se debe generar algo llamado soporte de replicación, esto se hace ejecutando el procedimiento:
GENERATE_REPLICATION_SUPPORT

TESIS CON
FALLA DE ORIGEN

El último paso crea los triggers, paquetes y procedimientos que permiten replicar los datos entre los nodos maestro.

1. Para cada nodo maestro que se necesite agregar al ambiente de replicación utilizar:
ADD_MASTER_DATABASE
2. Para los nodos del ambiente de replicación se propagaran de manera asincrónica la propagación debe programarse de la siguiente manera:
DBMS_DEFER_SYS.SCHEDULE_EXECUTION
3. Debido a que toda la actividad de la base de datos distribuida se suspende al ejecutar el procedimiento **CREATE_MASTER_REPGROUP**, es necesario resumir la actividad con el procedimiento **RESUME_MASTER_ACTIVITY**.

[PRAT95] Debido a que los triggers que se generan podrían incluir llamadas a procedimientos generados en esta misma etapa (para permitir la replicación asincrónica), el proceso de soporte de replicación se genera en dos fases:

1ª Etapa.

En la primera fase Oracle transmite a todos los nodos las instrucciones para la creación de paquetes necesarios, que es hecho de manera asincrónica.

2ª Etapa.

La segunda fase no se inicia hasta que todos los nodos comunican al nodo maestro de definición que la etapa anterior fue completada. En ese momento y de manera sincrónica Oracle inicia la creación de triggers necesarios y otros paquetes faltantes en cada nodo.

Como definimos en la replicación sincrónica los cambios son propagados en el mismo instante en que se están modificando los datos localmente. Oracle se encarga de que las transacciones distribuidas sean completadas o se les haga rollback en caso de falla en uno de los nodos. La manera de asegurar la consistencia de las actualizaciones es usando un método bloques que consiste en colocar un candado en el registro que esta siendo modificado, posterior a esto un trigger AFTER ROW intenta colocar el mismo candado en el registro remoto. Estos candados son liberados cuando la transacción es completada en cada uno de los nodos.

Al propagar cambios de manera sincrónica no es posible que se presenten conflictos de actualización. Sin embargo, si uno de los nodos lo hace de manera asincrónica es posible que estos conflictos se presenten, para esto Oracle cuenta con técnicas de solución de conflictos que se revisarán enseguida.

4.3.2 Propagación asincrónica en multi-master

Los conflictos en la replicación de datos suceden en los ambientes de replicación avanzada que permiten actualizaciones concurrentes a los mismos datos desde diferentes nodos, especialmente en ambientes multi-master. Esta situación solamente se presenta cuando la propagación de datos se hace de manera asincrónica.

Una manera de evitar estos conflictos y garantizar la consistencia en las transacciones diferidas es asegurar que el orden en que las transacciones son ejecutadas en cada nodo sea el mismo orden una vez que son propagadas.

Cada vez que se agrega un nodo al ambiente distribuido se puede elegir el método de propagación. Debido a que la manera en la que se hace la propagación determina cómo el nodo envía y recibe transacciones del resto de los nodos, el orden en que los nodos son agregados es también importante.

La Figura 4.3a muestra a continuación el orden de integración de nodos maestro. El ambiente tiene los nodos A, B y C. El nodo A es el *nodo maestro de definición*; se agrega el nodo B con un método de propagación sincrónico, la comunicación entre A y B se dará de esta manera sin transacciones diferidas. Si se agrega el nodo C con un método de propagación asincrónica, la replicación se da de la siguiente manera:

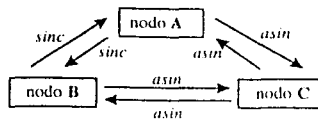


Figura 4.3a Orden de integración de nodos maestro.

Como el nodo C va a generar y recibir transacciones diferidas, se debe programar su propagación de los nodos A y B al C, así como del nodo C a los nodos A y B.

Supongamos que el nodo A sigue siendo el *nodo maestro de definición* y se agrega el nodo C con propagación asincrónica, agregando en tercer lugar el nodo B con propagación sincrónica, la replicación entre los nodos será Como se muestra en la Figura 4.3b a continuación.

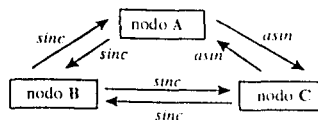


Figura 4.3b Orden de integración de nodos maestro.

Cada vez que se agregan nodos al ambiente distribuido se deben considerar los efectos que se tendrán en la recepción y emisión de transacciones, especialmente si son diferidas. En caso de que la propagación no se este dando en la manera que se requiere es posible cambiarla usando el API `DBMS_REPCAT.ALTER_MASTER_PROPAGATION`

TESIS CON
FALLA DE ORIGEN

Si el cambio en la propagación entre los nodos no resuelve los conflictos de actualización Oracle puede detectar los problemas usando rutinas definidas para resolverlos.

4.3.3 Tipos de conflictos y grupos de columnas.

El objetivo de la solución de conflictos es el asegurar la convergencia de datos. La convergencia garantiza que todos los nodos en el ambiente de replicación tienen los mismos datos. [PRAT95]

La manera en la que se detectan los conflictos es durante la propagación de las actualizaciones. Al empujar los cambios que se encuentran en la cola de transacciones diferidas, la replicación simétrica de Oracle hace llamadas a procedimientos remotos localizados en el nodo que recibe la transacción. Es el procedimiento remoto es el que detecta si existe un conflicto con los datos.

Por cada registro cambiado, Oracle envía:

- El valor anterior por cada columna del registro (antes de ser modificados)
- El nuevo valor para cada columna. Si se trata de INSERT no existen valores anteriores para el registro, si es DELETE no hay valor nuevo.

Si el procedimiento en el nodo receptor de los cambios no detecta conflicto, el RDBMS remoto aplica los cambios a los datos locales. Si se detecta un conflicto se aplica una de las rutinas definidas. Cualquier conflicto no resuelto es registrado en una vista en el nodo receptor, la vista se llama *DefError*.

Existen tres tipos de conflictos que son detectados por la replicación simétrica:

Unicidad.

Ocurre cuando la replicación de un registro intenta violar una regla de integridad de datos (llave primaria) Esto puede presentarse en operaciones de INSERT y UPDATE de un registro replicado.

Actualización.

Se presenta cuando la replicación de transacciones a un registro tienen conflicto con otra sentencia de UPDATE al mismo registro pero originada en un no lo diferente.

La manera de detectar este tipo de conflictos es comparando los valores anteriores con los existentes en cada registro replicado

Borrado.

Cuando dos transacciones que vienen de nodos diferentes, y una de ellas borra un registro que esta siendo actualizado o borrado por la otra.

Se detectan cuando una sentencia de UPDATE o DELETE no puede encontrar la llave primaria del registro.

Para poder detectar de manera precisa los conflictos es necesario identificarlos de manera única para todos los nodos. Una manera típica es usando la llave primaria de la tabla. Si la tabla no tiene una llave primaria es posible asignar una llave alterna (puede ser una columna o grupo de columnas) usada para identificar los registros durante la replicación de datos.

Es importante que cuando se asigne una llave alterna en lugar de la llave primaria, la aplicación debe ser la responsable de garantizar unicidad; además la aplicación no debe permitir al usuario actualizar las columnas de llave primaria para garantizar que Oracle pueda identificar los registros como únicos.

La manera de evitar conflictos de unicidad de datos es usando **dueños compartidos de información**, que puede conseguirse con *secuencias de base de datos* en cada nodo, donde los valores generados son excluyentes. Esto tiene un inconveniente, como se señaló antes, cuando el número de nodos o el número de registros replicados aumenta. Otra alternativa es usar las mismas secuencias agregando también el identificador del nodo como parte de una llave primaria compuesta.

Para evitar problemas de actualización, hay que entender donde son posibles estos errores en la aplicación y evitarlos, si es posible, en la etapa de diseño. Siempre que la aplicación lo permita se debe definir un nodo primario responsable de las actualizaciones. El mismo escenario aplica para los conflictos por borrado de datos. Adicionalmente, aquellas aplicaciones que propagan datos asincrónicamente y tienen dueños compartidos de datos no deben borrar registros con la instrucción DELETE, lo que se recomienda hacer es marcar los registros para ser borrados y configurar el sistema para que borre estos registros marcados de manera periódica.

Como ya vimos los conflictos de actualización se detectan mediante la comparación de valores de los registros replicados anteriormente con los nuevos del nodo emisor con los del nodo receptor. Dependiendo de las versiones de Oracle es posible minimizar la cantidad de datos que se replican para detectar este tipo de conflictos.

A partir de la versión Oracle8, es posible minimizar la propagación de datos, como resultado el desempeño general del ambiente se ve mejorado. Sin embargo, si el ambiente de replicación usa ambas versiones Oracle7 y Oracle8, no es posible minimizar la comunicación de datos del registro para resolver conflictos de actualización. En la versión Oracle7, el RDBMS debe propagar las versiones nuevas y existentes de cada registro.

Grupos de columnas.

Estos son usados para detectar y resolver conflictos durante la replicación asincrónica de datos.

Un grupo de columnas es una agrupación lógica de una o más columnas en una tabla. Cada columna en una tabla replicada es parte solo un grupo de columnas. Es posible crear grupos de columnas y asignar para cada grupo varias rutinas de solución de conflictos. Si se indican varios métodos de solución de conflictos Oracle es capaz de resolver este tipo de problemas de diferentes maneras.

No se recomienda que las columnas de llave primaria pertenezcan a un grupo de columnas. Otra **recomendación** es colocar aquellas columnas que deben tener valores consistentes en el mismo **grupo de columnas** para asegurar la integridad de datos.

La manera en la que se crean los grupos de columnas es con el paquete **DBMS_REPCAT**, el cual contiene los siguientes procedimientos:

- ✓ **MAKE_COLUMN_GROUP**. Crea un nuevo grupo de columnas con uno o varios miembros.
- ✓ **DROP_COLUMN_GROUP**. Elimina un grupo de columnas.
- ✓ **ADD_GROUPED_COLUMN**. Agrega una columna a un grupo existente.
- ✓ **DROP_GROUPED_COLUMN**. Elimina una columna de un grupo.
- ✓ **DEFINE_COLUMN_GROUP**. Crea un grupo de columnas vacío.

Además de los grupos definidos, existe un grupo llamado **SHADOW**, y cada tabla replicada tiene uno por definición. Las columnas que no pertenecen a ningún grupo específico forman parte del grupo *shadow*. Una característica de estos grupos es que no se les puede asignar una rutina de solución de conflictos.

El mecanismo de solución de conflictos está basado en paquetes de código PL/SQL llamados \$RR. Estos son creados automáticamente al generar el soporte para la replicación de tablas, sin embargo, no resuelven ningún conflicto hasta que no se asocia alguna rutina de solución a uno de sus grupos de columnas.

Las rutinas de solución de dependen del número de nodos maestro que tenga el ambiente de replicación, si tenemos solamente un nodo maestro las rutinas disponibles son:

- Valor mínimo, valor máximo.
- Tiempo de actualización (primero y último)
- Grupos de prioridad.
- Nodos de prioridad.
- Reemplazo, descartar, promedio y aditivo.

Para aquellos ambientes con dos nodos maestros se tienen:

- Valor mínimo y máximo.
- Tiempo de actualización (primero y último)
- Grupo de prioridad.
- Nodos de prioridad.
- Aditivo

Si el ambiente tiene más de dos nodos maestros las rutinas de solución disponibles se reducen a valores mínimos (decremento de columna), valores máximos (incremento de columnas), tiempo de actualización (último), y aditivo.

No se recomienda tener más de dos nodos maestro en un ambiente distribuido con propagación diferida ya que las rutinas de solución no garantizan la convergencia de los datos. El riesgo de datos no convergentes se ve aumentado por posibles problemas de red y por intervalos poco frecuentes de propagación de los RPCs diferidos.

Así es como funcionan las rutinas de solución de conflictos:

Valores mínimos. La rutina compara el nuevo valor del nodo donde se originan los cambios con el valor existente en el nodo destino, esto para una columna específica dentro de un grupo de columnas. Si el valor designado es menor que el existente los valores del grupo de columnas son copiados en el nodo destino. Si el valor es mayor, el conflicto es resuelto dejando los valores existentes sin cambio.

Valores máximos. El valor máximo funciona en la misma manera que la rutina anterior, con la excepción de que los cambios serán aplicados solamente si los valores del nodo origen son mayores a los existentes en el nodo destino.

Actualización más temprana. Aplica los cambios solamente de aquellos que fueron primeramente modificados.

Última actualización. Los cambios se aplican si es más reciente el momento en el que fueron modificados. En ambos casos la hora de la actualización debe ser actualizada en la tabla. En las dos últimas rutinas es importante el considerar si los nodos se encuentran en diferente horario. De ser este el caso puede ser necesario código extra (en forma de triggers) que realicen la conversión automática de la diferencia de horario.

Aditiva. Esta rutina funciona para una columna tipo numérico dentro del grupo de columnas. Funciona agregando la diferencia entre el valor viejo y nuevo en el nodo de origen al valor existente en el nodo destino

$$valor_existente = valor_existente + (nuevo_valor - valor_existente)$$

La rutina aditiva da convergencia con cualquier número de nodos maestro.

Promedio. Esta rutina saca el promedio de los nuevos valores en los nodos origen con el valor existente en el nodo destino.

$$valor_existente = (valor_existente + nuevo_valor) / 2$$

La rutina de promedios no puede garantizar la convergencia de datos si el ambiente de replicación tiene más de dos nodos maestro. Se recomienda para ambientes con un nodo maestro y varios nodos de snapshots actualizables.

Grupos y nodos de prioridad. Los grupos de prioridad permiten asignar un nivel de importancia a cada posible valor de una columna. Si un conflicto se detecta, Oracle actualiza la tabla con una prioridad más baja con el valor de aquella columna que tiene la prioridad alta. El nodo de prioridad, por su parte, utiliza el nombre de la base de datos para determinar la importancia de los datos. La vista `DBA_REPPRIORITY` despliega la prioridad asignada a cada nodo del ambiente replicado.

Al igual que los grupos de prioridad, la información del nodo con una prioridad mayor es usada para resolver conflictos de actualización. La restricción de esta rutina es que si el ambiente de replicación tiene más de dos nodos maestro la convergencia de datos no está garantizada.

La **Tabla 4.4** describe un posible conflicto de actualización en un ambiente con tres nodos maestro. Si el nodo A tiene una prioridad de 30, el nodo B tiene una prioridad de 25, y el nodo

C tiene prioridad de 10. X es una columna de un registro en particular en un grupo de columnas donde se asigno el método de solución de nodos de prioridad.

Tempo	Acción	Nodo A	Nodo B	Nodo C
1	Todos los nodos funcionan y saben que X=2	2	2	2
2	Nodo A actualiza x=5	5	2	2
3	El nodo C no es accesible	5	2	Cerrada
4	El nodo A propaga los cambios al nodo B. Los nodos A y B saben que X=5 El nodo C no esta disponible y la transacción queda pendiente en el nodo A	5	5	Cerrada
5	El nodo C es funcional nuevamente con X=2 Nodos A y B tienen X=5	5	5	2
6	El nodo B actualiza X=7	5	7	2
7	Nodo B propaga la transacción al nodo A Nodos A y B saben que X=7 Nodo C sigue con valor X=2	7	7	7
8	El nodo B propaga la transacción al nodo C. El nodo C informa que X=2 y el nodo B informa que X tenía un valor 5. En este momento Oracle detecta el conflicto y resuelve aplicando la actualización del nodo B, que tiene una prioridad mas alta (25) que el nodo C (10) Todos los nodos tienen un valor de X=7	7	7	7
9	El nodo A propaga el cambio de X=5 al nodo C. Oracle detecta el conflicto dado que el valor en C (X=5) es diferente al valor registrado en A (X=2) El nodo A tiene mayor prioridad (30) que el nodo C (10) Oracle resuelve aplicando la transacción del nodo A (X=5)	7	7	5

Tabla 4.4 Posible conflicto de actualización en un ambiente con tres nodos maestro

Como se vio en este escenario con conflicto de actualización, los datos en los nodos no son convergentes si se usan nodos de prioridad. La manera de resolver estos problemas es asignando orden al flujo de las actualizaciones. Esto es, no permitiendo que todos los datos cambien ciertos valores si es que estos ya fueron actualizados por otro nodo con una prioridad mayor.

Los grupos de prioridad se manejan con el paquete `DBMS_REPCAT`, los procedimientos existentes son:

- ✓ `DEFINE_PRIORITY_GROUP`. Para crear un nuevo grupo de prioridad para un grupo de objetos replicados.
- ✓ `DROP_PRIORITY_GROUP`. Para eliminar un grupo de prioridad.
- ✓ `ADD_PRIORITY`. Para agregar un miembro al grupo de prioridad.
- ✓ `ALTER_PRIORITY`. Para agregar un nuevo nodo al grupo de prioridad, y para cambiar el nivel de prioridad de un miembro del grupo de prioridad.
- ✓ `DROP_PRIORITY`. Para eliminar un miembro de un grupo de prioridad por nivel de prioridad, o por nombre.

TESIS CON
 FALLA DE ORIGEN

El manejo de los nodos de prioridad usa el mismo paquete, los procedimientos son:

- ✓ **DEFINE_SITE_PRIORITY.** Crear un nuevo nodo de prioridad para un grupo de objetos replicados.
- ✓ **DROP_SITE_PRIORITY.** Elimina el nodo de prioridad para el grupo replicado.
- ✓ **ADD_SITE_PRIORITY_SITE.** Agrega un nodo a un nodo de prioridad existente.
- ✓ **ALTER_SITE_PRIORITY_SITE.** Cambia el nodo asociado a un nivel de prioridad.
- ✓ **ALTER_SITE_PRIORITY.** Cambia la prioridad de un nodo.
- ✓ **DROP_SITE_PRIORITY_SITE.** Elimina un nodo por nombre del grupo de nodos de prioridad.

Reemplazo y Descartar. La rutina de reemplazo cambia el valor existente en el nodo local con el valor del nodo remoto. Mientras que la de descartar no toma en cuenta nunca el valor del nodo remoto. Como se puede notar estas rutinas ignoran los valores de ambos nodos origen y destino por lo que no pueden garantizar la convergencia de datos para ambientes con mas de un nodo maestro. Están diseñadas para ser usadas por un nodo primario y múltiples nodos de snapshots.

La Tabla 4.5 muestra un resumen de las rutinas de solución de conflictos previstas por Oracle y en que escenarios se garantiza la convergencia de datos.

Operación	Modo de solución	Convergencia	Convergencia en nodos maestros
UPDATE	Mínimos	Si	No, a menos que sean siempre decrezcan
	Máximos	Si	No, a menos que siempre se incrementen
	Actualización temprana	Si	No
	Última actualización	Si	Si, con un método de respaldo
	Grupo de prioridad	Si	No, a menos que siempre se incremente
	Nodo de prioridad	Si	No
	Reemplazo	Si, un nodo maestro	No
	Descarte	Si, un nodo maestro	No
	Promedio (solamente numérico)	Si, un nodo maestro	Si
Aditivo (solamente numérico)	Si	Si	
UNICIDAD (UPDATE - INSERT)	Agregar nombre nodo	No	No
	Ignorar y eliminar	No	No
BORRADO	Ninguno		

Tabla 4.5 Rutinas de solución de conflictos previstas por Oracle.

TESIS CON
 FALLA DE ORIGEN

Capítulo IV. Replicación de datos.

Capítulo V.

Implementación de un esquema de replicación.

5.1 Definición del caso de estudio.

La organización de Soporte Técnico de Oracle para Latinoamérica esta formada por 10 subsidiarias y un Centro de Soporte al Producto (PSC) localizado en Orlando. Las subsidiarias son:

- México y Brasil
- Venezuela y Uruguay
- Ecuador y Perú
- Costa Rica y Puerto Rico
- Chile y Colombia

El personal localizado en la subsidiaria es de tipo administrativo, de ventas de soporte y de soporte técnico que trabaja en las instalaciones del cliente. Mientras que en el PSC hay personal técnico y de atención a clientes.

El cliente puede reportar problemas técnicos y solicitar actualizaciones de productos de manera telefónica (haciendo una llamada local que es automáticamente dirigida al PSC en Orlando), o de manera electrónica. Para validar si el cliente tiene derecho a recibir el servicio se utiliza la información capturada en el sistema Voyager/2000. Cuya pantalla principal se muestra en la **Figura 5.1** a continuación:

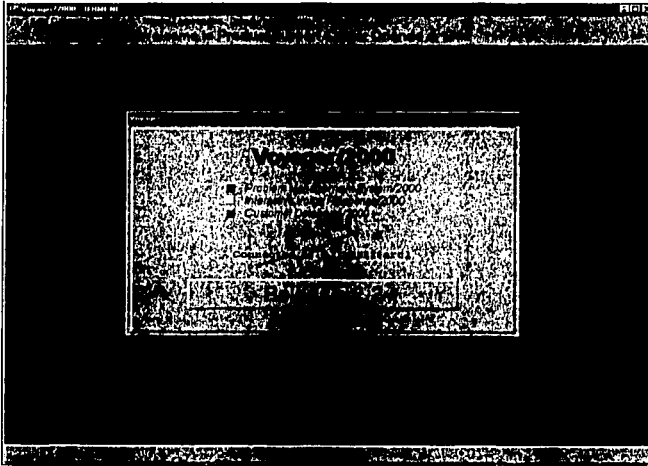


Figura 5.1 Pantalla principal de aplicación Voyager/2000.

Voyager/2000 es una aplicación Cliente-Servidor usada para administrar la información de los clientes de soporte técnico, incluye: datos del cliente, contactos técnicos, fechas de expiración de contratos, tipos de servicio y productos para los que el cliente puede recibir servicio.

Es una aplicación corporativa que es usada en otras Divisiones además de Latinoamérica. Está desarrollada en Developer/2000 que es un API de Oracle para construir aplicaciones Cliente-Servidor, los clientes pueden ser estaciones Windows95, 98 o NT/2000 y la base de datos puede ser de Oracle7 hasta Oracle9i en plataforma UNIX/Linux o Windows NT. La aplicación es formada por dos módulos principales:

- **CD2000 (Customer Database)** En este modulo se agrupan las formas y tablas necesarias para el registro y manejo del perfil del cliente; contratos de soporte; renovaciones de servicio. Cuyas formas se describen en la **Tabla 5.2**

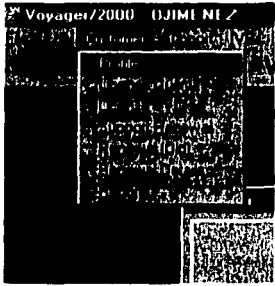
<i>Profile</i>	Para registrar y modificar el perfil de los clientes. Se incluye nombre de compañía, contactos, dirección.	
<i>License Query</i>	Para la consulta rápida de clientes registrados por numero de identificación (CSI)	
<i>License</i>	Para el registro de contratos, vigencia, monto, productos en el contrato, plataforma y generación de ID (CSI)	
<i>Support Renewal y renewal (old)</i>	Para renovar contratos de soporte, cambios en el nivel de servicio y productos.	
<i>Renewal Details</i>	Para uso administrativo es información de control de los montos de la renovación de contratos.	
<i>Reports</i>	Reportes por cliente, sistema operativo, plataforma, numero de identificación (CSI) y del detalle de contratos.	

Tabla 5.2 Formas del modulo CD2000.

- **PMS2000 (Problem Management System)** En este modulo se registran las solicitudes de actualización de productos y es repositorio de solicitudes técnicas hechas por los clientes y se agrupa en:
 1. *Soporte (Support)* Para dar seguimiento a registros creados para dar seguimiento a las actualizaciones solicitados por el cliente.
 2. *Envíos (Shipping)* Para registrar las actualizaciones de versiones de software que el cliente hace.

TESIS CON
 FALLA DE ORIGEN

La Tabla 5.3 enseguida muestra las principales formas del modulo PMS2000.

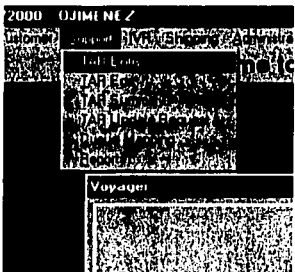
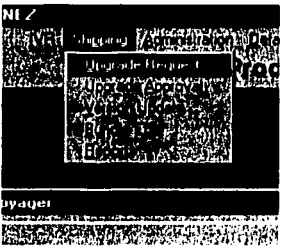
<i>TAR Entry</i>	Para registrar solicitudes de atención técnica (TAR) Para hacerlo el cliente necesita tener un CSI valido.	
<i>TAR Edit</i>	Para editar TARs existentes.	
<i>TAR Summary</i>	Para ver un resumen de los TARs asignados al analista de relaciones con clientes.	
<i>TAR Update Request</i>	Para dar seguimiento a los TARs creados para las solicitudes de actualización de productos.	
<i>Queue Mapping</i>	Forma para ordenar los analistas por especialidad o familia de productos.	
<i>Reports</i>	Reportes de TARs abiertos por cliente, por productos por numero de CSI, por plataforma.	
<i>Upgrade request</i>	Registro de solicitud de actualizaciones de producto. Esta solicitud genera un numero de TAR. (Que es no técnico)	
<i>Upgrade approval</i>	Forma para la aprobación de las solicitudes una vez que se revisa la vigencia del contrato, versión solicitada, plataforma.	
<i>Version update</i>	Forma para cerrar el ciclo de renovación de productos. Esta forma permite reflejar las nuevas versiones enviadas al cliente en el modulo CD2000.	
<i>Bundle edit</i>	Permite agrupar productos que se venden juntos de manera que se agilice su registro.	
<i>Reports</i>	Reporte de detalle de actualización por numero de TAR generado.	

Tabla 5.3 Formas de envíos y soporte del modulo CD2000.

La vigencia de los contratos y otra información de control se transfiere a bases de datos que son usadas por diversos centros de soporte, entre ellos el PSC en Orlando, donde se resuelven los problemas técnicos de los clientes.

Sin embargo, hay otros procedimientos que requieren la atención de analistas no técnicos del grupo de relaciones con clientes en el PSC. Estos utilizan la aplicación Voyager/2000 para solicitar actualizaciones de productos de clientes así como resolver otros problemas que no

permiten al cliente usar el servicio de soporte, por lo que tienen que hacer conexiones Cliente-Servidor del PSC a las diferentes subsidiarias en América Latina.

Una premisa del negocio es que la subsidiaria es responsable de la información de clientes, contactos y contratos por lo que toda entrada y modificación debe hacerse utilizando Voyager/2000.

Estos procesos nos ha permitido dividir la utilización del sistema Voyager2000 entre el PSC en Orlando y cada una de las subsidiarias de la siguiente manera:

Procesos hechos en las subsidiarias:

1. Registro de información del cliente y sus contactos técnicos, direcciones para envío de actualizaciones de software. (Forma *Profile* del modulo CD2000)
2. Captura de nuevos contratos de soporte (Forma *License* y *License Query* del modulo CD2000) además de cotizaciones y renovación de contratos existentes.
3. Autorización de solicitudes de actualización de productos. Solamente personal autorizado (Forma: *Upgrade Approval* del modulo PMS2000)
4. Actualización de versiones de productos enviados al cliente. (Forma: *Version Update* del modulo PMS2000)
5. Seguimiento de estado TARs no técnicos creados para la solicitud de productos (Formas: *TAR Edit*, *TAR Summary* y *TAR Update Request* del modulo PMS2000)

Procesos en el PSC en Orlando:

1. Verificación de datos de contactos técnicos. (Lectura en forma *Contact Registration* del modulo CD2000)
2. Revisión de vigencias de soporte técnico (Lectura en forma *License query* y *License* del modulo CD2000)
3. Registro de solicitudes de actualización de productos (Actualización en forma *Upgrade Request* del modulo PMS2000)

Problema

El problema es un tiempo de respuesta lento al usar la aplicación, provocado por la lejanía entre el PSC y las subsidiarias, unas incluso con un pobre ancho de banda (que determina la velocidad de envío y recepción de paquetes de datos en la red)

Esto se traduce en frustración del cliente que llama al PCS pues el tiempo para que procesen sus solicitudes es muy alto, además de inconformidad de los analistas que usan la aplicación.

La **Tabla 5.4** Muestra el promedio del tiempo de respuesta de 10 operaciones (lectura y actualización) en un lapso de una hora desde el PSC a las diferentes subsidiarias.

<i>Pais</i>	<i>Tiempo de respuesta por transacción (seg.)</i>
Uruguay	15
Brasil	10
Costa Rica y Puerto Rico	20
Colombia, Venezuela y Chile	17
Perú	35
México	7
Ecuador	45

Tabla 5.4 Tiempo promedio por transacción en una hora.

Objetivo

El objetivo del siguiente análisis busca mejorar el tiempo de respuesta en el uso de la aplicación Voyager/2000 desde el PSC.

Un primer análisis fue el afinar la base de datos y seguir con la base de datos centralizada en la subsidiaria con las conexiones Cliente-Servidor desde el PSC. Sin embargo, no se percibió mejora significativa por lo que se considera una base de datos distribuida. Lo que nos lleva al objetivo de la tesis:

"Analizar las propiedades que debe cumplir una base de datos distribuida e implementar un esquema de replicación utilizando uno de los métodos disponibles en Oracle7 como alternativa al uso de las bases de datos centralizadas."

Restricciones para la implementación.

Voyager/2000 es un sistema corporativo, lo que quiere decir que no se permite hacer modificaciones a la estructura de las tablas y formas de despliegue ya que en caso de presentarse errores originados por cambios no se recibiría apoyo del grupo de desarrollo de la aplicación. Esta es una restricción importante en caso de que se deseen plantear otras alternativas para la distribución de los datos.

5.2 Construcción del esquema de replicación.

Siguiendo la metodología de los 9 pasos para la distribución de datos de Marie Burette revisada en el capítulo cuarto, tenemos:

El **primer paso** es la preparación del perfil distribuido de la aplicación. Se identificaron dos tipos lógicos de nodo y la naturaleza distribuida de sus procesos comunes:

- PSC.
- Subsidiaria (para cada uno de los países de Latinoamérica)

Los usuarios de la aplicación son básicamente de tres tipos:

- Analista de relaciones con clientes (ubicados en el PSC)
- Gerentes de venta de soporte (localizados en la subsidiaria)
- Personal administrativo (también trabajando desde la subsidiaria)

Los procesos y / o actividades lógicas más importantes:

- Registro de nuevos clientes y contactos.
- Actualización de información existente
- Captura de nuevos contratos y renovación de contratos existentes.
- Captura de solicitud de productos.
- Procesamiento de solicitudes de actualizaciones de productos.

Las actividades y usuarios anteriores son comunes para la operación diaria entre el PSC y los países. A partir de este momento se seguirá analizando el caso del ambiente de replicación para el PSC y Oracle México. Las cantidades expresadas en las siguientes matrices pueden variar dependiendo de la subsidiaria que se revise.

Las Tablas (5.5 a 5.9) de perfil distribuido para cada proceso son las siguientes:

Nombre de la aplicación: Registro de nuevos clientes y contactos.			
<i>Tipo de Localidad</i>	<i>Localidad</i>	<i>Tipos de usuario y numero esperado de usuarios</i>	<i>Transacciones de negocios con velocidades anticipadas de ejecución. (promXdia)</i>
PSC	Orlando		0-no hay registro desde Orlando
Subsidiaria	México	Personal administrativo (3 analistas)	6 clientes cada uno con 5 contactos por cliente

Tabla 5.5 Perfil distribuido para registro de nuevos clientes y contactos.

Nombre de la aplicación: Actualización de información de clientes.			
<i>Tipo de Localidad</i>	<i>Localidad</i>	<i>Tipos de usuario y numero esperado de usuarios</i>	<i>Transacciones de negocios con velocidades anticipadas de ejecución. (promXdia)</i>
PSC	Orlando	Analistas de relaciones con clientes (4 analistas)	25 cambios de información de contactos
Subsidiaria	México	Personal administrativo (3 analistas)	5 actualizaciones y/o nuevos contactos

Tabla 5.6 Perfil distribuido para actualización de información de clientes.

**TESIS CON
FALLA DE ORIGEN**

Nombre de la aplicación: Captura de nuevos contratos.

<i>Tipo de Localidad</i>	<i>Localidad</i>	<i>Tipos de usuario y numero esperado de usuarios</i>	<i>Transacciones de negocios con velocidades anticipadas de ejecución. (promX día)</i>
PSC	Orlando		0-contratos sólo desde subsidiaria
Subsidiaria	México	Personal administrativo (3 analistas), Gerentes de cuenta de ventas de soporte	6 nuevos contratos, 5 renovaciones diarias.

Tabla 5.7 Perfil distribuido para captura de nuevos contratos.

Nombre de la aplicación: Solicitud de productos.

<i>Tipo de Localidad</i>	<i>Localidad</i>	<i>Tipos de usuario y numero esperado de usuarios</i>	<i>Transacciones de negocios con velocidades anticipadas de ejecución. (promX día)</i>
PSC	Orlando	Analistas de relaciones con clientes (4 analistas)	14 solicitudes de actualización
Subsidiaria	México		0- no hay solicitudes locales

Tabla 5.8 Perfil distribuido para solicitud de productos.

Nombre de la aplicación: Procesamiento de solicitud de actualizaciones de productos.

<i>Tipo de Localidad</i>	<i>Localidad</i>	<i>Tipos de usuario y numero esperado de usuarios</i>	<i>Transacciones de negocios con velocidades anticipadas de ejecución. (prom. Diario)</i>
PSC	Orlando		0 lectura de estatus de solicitudes
Subsidiaria	México	Personal administrativo (3 analistas)	13 procesamientos entre PSC y subsidiaria

Tabla 5.9 Perfil distribuido para registro de nuevos clientes y contactos.

El resultado esperado es que cada uno de los procesos ocupe de 3 a 5 segundos para procesar las transacciones.

El paso número dos es definir el perfil distribuido de infraestructura física. La Tabla 5.10 muestra los recursos con los que se cuenta en cada nodo y es la base sobre la que será construido el ambiente de replicación.

**TESIS CON
FALLA DE ORIGEN**

Nombre de la aplicación: Voyager2000 Oracle México - PSC Orlando						
Nodo/ tipo de nodo	Red tipo y velocidad	Hardware recursos	Software recursos	Nivel de seguridad	Nivel de disponibilidad	Nivel técnico y soporte de administradores
PSC	Enlace privado México- Orlando	Dell Power Edge	Server NT4.0	Intranet privada	Servidores de base de datos	2 DBAs, 1 administrad or OS, área de MIS
		Dell Opti- Plex	Worksta tion NT4.0	Cientes protegid os a nivel SO.	Estaciones por UPS. con UPS individuales	
México	Red local T-1 LAN	HP9000 Dell Opti- Plex	HP-UX 10.20 Worksta tion NT4.0	Acceso a DB por cuentas individu ales.		2 DBAs, 1 administrad or de OS. Soporte MIS

Tabla 5.10 Perfil distribuido de infraestructura.

Para los pasos tres y cuatro, que son la definición de grupos de recuperación, y los perfiles proceso / dato respectivamente. Se identifican cuales son las entidades involucradas en cada uno de los procesos entre el PSC y la subsidiaria.

De ser necesario, y por razones ya explicadas, no sería posible hacer *desnormalización* de datos (técnica usada en implementación de bases distribuidas) o considerar cualquier otro cambio que implique modificar las entidades ya definidas en CD2000 y PMS200.

Al terminar esta etapa se habrán presentado las pantallas usadas por cada proceso, el tipo de operación desde cada nodo y las entidades involucradas.

Los modos de operación que cada entidad puede tener los definimos como: (L) lectura; (E) entrada de datos; (A) actualización; y (B) borrado.

Enseguida se muestra los grupos de recuperación por cada proceso y el perfil proceso / dato incluyendo la pantalla de captura:

1. Registro de nuevos clientes y contactos. Actualización de información de contactos.

a) Entidades de base de datos y modo de operación (Tabla 5.11)

Esquema	Tabla	Modo de operación	
		PSC Orlando	México
CD2000	COMPANY	L	LEAB
	PERSON	LEAB	LEAB
	ADDRESS	L	LEAB
	CD2000_SEQ	-	LA
PMS2000	-	-	-

Tabla 5.11 Grupo recuperación

TESIS CON
 FALLA DE ORIGEN

- b) Entidades de base de datos y modo de operación
 Nombre de la forma: Company Profile

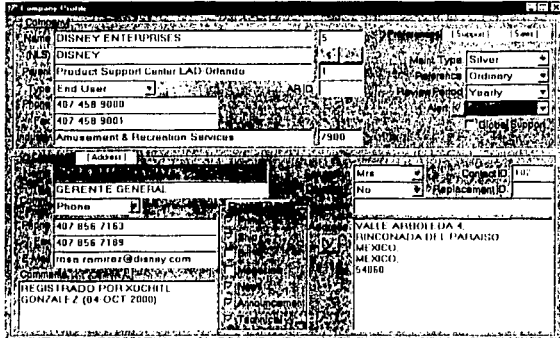


Figura 5.11a Pantalla de Company Profile

- c) Relaciones entre las entidades.

Esquema	Tabla	Relacionado con	Tipo de relación	Cardinalidad
CD2000	COMPANY	PERSON	Debe tener ...	1:M
		ADDRESS	Puede tener ...	1:M
	PERSON	ADDRESS	Puede tener ...	1:M
	CD2000_SEQ	COMPANY	Califica a ...	1:M
		PERSON	Califica a ...	1:M
		ADDRESS	Califica a ..	1:M

Tabla 5.11b Perfil proceso / dato.

2. Captura de nuevos contratos

- a) Entidades de base de datos y modo de operación (Tabla 5.12)

Esquema	Tabla	Modo de operación	
		PSC Orlando	México
CD2000	COMPANY	L	L
	MAC_TYPES	L	L
	MACHINE	L	LEAB
	PROD_LIC	L	LEAB
	PRODUCTS	L	L
	VER_HISTORY	L	L
	MACHINE_LOG	-	EA
	PRODUCT_LOG	-	LA
	CD2000_SEQ	L	LA

Tabla 5.12 Grupo recuperación.

b) Pantalla de captura (Figura 5.12a)

Nombre de la forma: **Company Maintenance License**

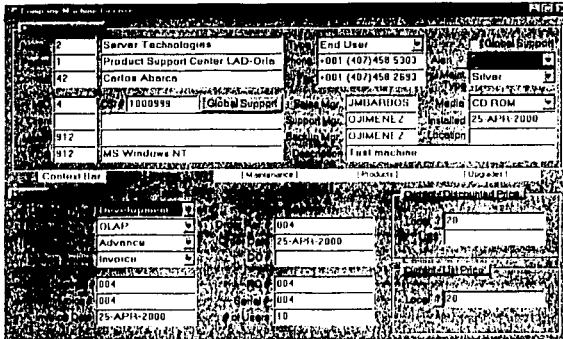


Figura 5.12a Pantalla de captura de nuevos contratos.

c) Relaciones entre las entidades.

Esquema	Tabla	Relacionado con	Tipo de relación	Cardinalidad
CD2000	COMPANY	MACHINE	Puede tener ...	1:M
	MACHINE	MAC_TYPES	Es de tipo ...	M:1
		PROD_LIC	Tiene varios ...	1:M
		MACHINE_LOG	Registra cambio en ...	1:M
PROD_LIC	PRODUCTS	Que describe ...	M:1	
	VER_HISTORY	Tiene cambios en ..	1:M	
CD2000_SEQ	PRODUCT_LOG	Registra cambio en .	1:M	
	MACHINE	Que califica ..	1:M	
PMS2000	-	-	-	-

Tabla 5.12b Perfil proceso / dato.

3. Captura de solicitudes de productos

b) Entidades de base de datos y modo de operación

Esquema	Tabla	Modo de operación	
		PSC Orlando	México
CD2000	UPG_REQUEST	LEA	LA
	UPG_STATUS	L	L
	UPG_DETAILS	LEA	LEA
PMS2000	CALLS	LEA	LEA
	ACTIONS	LEA	LEA

Tabla 5.13 Grupo de recuperación.

TESIS CON
 FALLA DE ORIGEN

b) Pantalla de captura (Figura 5.13 a)

Nombre de la forma: Upgrade Request

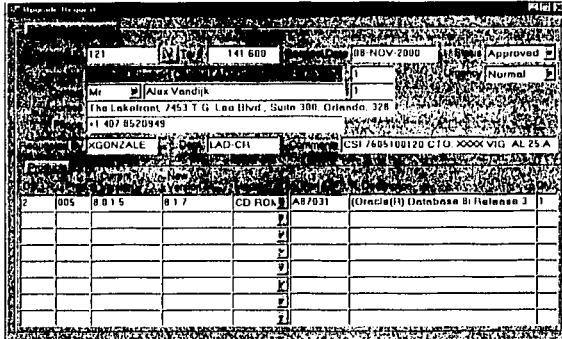


Figura 5.13a Pantalla solicitud de productos.

c) Relaciones entre las entidades.

Esquema	Tabla	Relacionado con	Tipo de relación	Cardinalidad
CD2000	UPG_STATUS	UPG_REQUEST	Que describe ...	1:M
	UPG_REQUEST	UPG_DETAILS CALLS	Que detalla ...	1:M
			Descrito en ...	1:1
PMS2000	UPG_DETAILS	PRODUCTS	Contenido en ...	M:1
	CALLS	ACTIONS	Que detalla	1:M

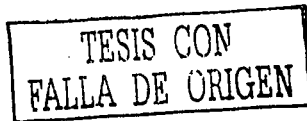
Tabla 5.14b Perfil proceso / dato.

4. Procesamiento de solicitudes de actualizaciones de productos

a) Entidades de base de datos y modo de operación

Esquema	Tabla	Modo de operación	
		PSC Orlando	México
CD2000	UPG_REQUEST	L	LA
	UPG_STATUS	L	L
	UPG_DETAILS	L	LEA
PMS2000	CALLS	L	LEA
	ACTIONS	L	LEA

Tabla 5.14 Grupo de recuperación.



a) Pantallas de captura

Nombre de la pantalla: Upgrade Approval

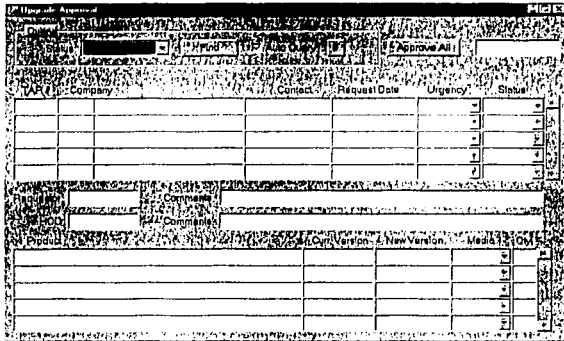


Figura 5.14a Pantalla para solicitar actualización de productos.

c) Relaciones entre las entidades.

Esquema	Tabla	Relacionado con	Tipo de relación	Cardinalidad
CD2000	UPG_STATUS	UPG_REQUEST	Que describe ...	1:M
	UPG_REQUEST	UPG_DETAILS CALLS	Que detalla ...	1:M
			Descrito por ..	1:1
PMS2000	UPG_DETAILS	PRODUCTS	Que contiene ...	M:1
	CALLS	ACTIONS	Detallada por	1:M

Tabla 5.14b Perfil proceso / dato.

El **paso número cinco** es identificar los datos que son requeridos por la aplicación pero que son de uso general.

La aplicación permite modificar las listas de valores usadas por la aplicación y valores comunes para el registro de la información de contactos, teléfonos, período de gracia para contratos expirados, nombres de usuarios administradores, tiempo horario, etc.

Estos parámetros son almacenados en forma de tablas de base de datos y controlan el comportamiento de Voyager/2000. Estas tareas son realizadas por un usuario administrador con privilegio de VOYAGER_DBA. Dado que estos valores son requeridos por la aplicación es necesario que sean replicados en ambos nodos del ambiente distribuido. Quedando la responsabilidad de su actualización en la subsidiaria y dejando acceso de solo lectura en el PSC.

En la siguiente hoja se muestran las entidades que contienen información de control para la aplicación, se incluye un comentario breve de cual es su uso. (Tabla 5.15)

TESIS CON
 FALLA DE ORIGEN

Esquema	Entidad	Comentarios de uso
CD2000	<p>BUNDLE</p> <p>BUNDLE_PRODUCTS COMM_PREFERENCE CSL_REFERENCE</p> <p>INDUSTRY LIC_TYPES MAC_TYPES MAINT_TYPES MEDIA_CODES PRODUCTS PRODUCTS_GROUP PRODUCT_FAMILY</p>	<p>Agrupar productos que son vendidos en grupo para facilitar el registro de nuevos contratos.</p> <p>Detalla cada BUNDLE de productos.</p> <p>Califica la preferencia de comunicación que tiene el cliente.</p> <p>Tabla de control para la generación de números de identificación del cliente (CSI)</p> <p>Define el ramo del cliente.</p> <p>Tipo de licencia comprada por el cliente.</p> <p>Plataforma y sistema operativo registrado.</p> <p>Tipo de servicio de soporte comprado.</p> <p>Tipos de medios de distribución de software.</p> <p>Tabla de productos que vende Oracle</p> <p>Tabla por grupos de productos TOOLS, DB ...</p> <p>Familia de producto Oracle, OLAP, etc.</p>
PMS2000	<p>ACTION_TYPE_CODES APPL_PREFERENCES</p> <p>APPL_PRIVS BUG_PRODUCTS BUG_PROD_LINES CONTACT_STATUS</p> <p>SEVERITY_TYPES</p>	<p>Tipos de acciones de los TARs abiertos por el cliente.</p> <p>Tabla con valores de configuración y despliegue de datos de Voyager2000.</p> <p>Tabla con roles de privilegios de acceso a la aplicación.</p> <p>Tabla de productos de la corporación.</p> <p>Plataformas para las que se venden productos.</p> <p>Lista de valores del estado que tiene un contacto al ser registrado.</p> <p>Tipos de severidad que un TAR puede tener.</p>

Tabla 5.15 Entidades con información de control.

El paso número seis es el esquema de colocación de datos, y su validación en referencia a los procesos del negocio. Las tablas identificadas en la etapa anterior existirán en ambos nodos. Actualizadas en la subsidiaria y replicadas como entidades de sólo lectura hacia el PSC.

Estas tablas serán actualizadas a intervalos regulares de tiempo para reflejar los últimos cambios hechos. Como ya definimos es necesario que todas las tablas sean replicadas en ambos nodos para asegurar el correcto funcionamiento de la aplicación.

Ya en la etapa proceso / dato se identificaron las entidades que serán replicadas y el tipo de operación que se realizara sobre estas desde ambos nodos (paso cuatro) Con esto definido se procede a implementar el esquema distribuido.

El paso siete, es la verificación que la distribución de datos puede ser implementada con la tecnología e infraestructura existente; y el paso ocho que es la validación de los procesos distribuidos o replicados y que cumplan con las reglas de negocio descritas. Se abordan enseguida.

5.3 Implementación de una base de datos distribuida usando replicación.

Oracle provee un API gráfico para la implementación y administración de una base que usa replicación de datos llamada *Replication Manager* que facilita y acelera la implementación. La herramienta funciona con versiones Oracle7 y posteriores.

La replicación de la aplicación *Voyager/2000* requiere de un ambiente de replicación avanzada con actualización en todas partes. Uno de los nodos será un nodo maestro de definición y el otro un nodo de maestro.

El tipo de propagación será en tiempo real, con excepción de las tablas que tienen información de control de la aplicación que serán replicadas manualmente.

El primer paso es hacer la configuración de las instancias para que soporten la replicación de datos. Enseguida proseguir con *Replication Manager* para definir que objetos vamos a replicar y de que manera.

Los siguientes pasos deben seguirse en todos los nodos (PSC y Subsidiaria) que fueron identificados anteriormente:

1. *Parámetros del archivo de init.ora*

El archivo de *init.ora* contiene parámetros de configuración de cada base de datos. El tamaño de SGA de la instancia, número de conexiones, etc. Para la replicación de datos se necesitan tener los siguientes definidos:

<u>Nombre del parámetro</u>	<u>Valor inicial recomendado</u>
COMPATIBLE	7.3.0.0.0
SHARED_POOL_SIZE	10000000
PROCESSES	Agregar 9 al valor que se tiene
GLOBAL_NAMES	TRUE
OPEN_LINKS	4 (Agregar 2 por cada nodo maestro adicional)
DISTRIBUTED_LOCK_TIMEOUT	300 segundos
DISTRIBUTED_TRANSACTIONS	5 (Agregar 2 por cada nodo maestro adicional)
JOB_QUEUE_INTERVAL	10 segundos
JOB_QUEUE_PROCESSES	2 (Agregar 1 por cada nodo maestro adicional)

2. *Requerimientos de TABLESPACES.* (Recomendados)

<u>TABLESPACE</u>	<u>Valor inicial recomendado</u>
SYSTEM	Al menos 20MB de espacio libre
ROLLBACK SEGMENTS	Al menos 5MB de espacio libre
TEMPORARY	Al menos 10MB de espacio libre

TESIS CON
FALLA DE ORIGEN

3. Instalación del catálogo de replicación.

El catálogo de replicación son las tablas especializadas del diccionario de datos que registran la información de control de que objetos y datos están siendo replicados entre los nodos. Para crearlo hay que utilizar una interfaz llamada Server Manager y ejecutar un script de comandos SQL. Los pasos son:

- a. Ejecutar el API Server Manager con el comando SVRMGR.
- b. Conectarse como usuario administrador INTERNAL
- c. Ejecutar el script CATREP.SQL una vez que la base de datos está levantada. El script se localiza en el directorio

```
%ORACLE_HOME%\RDBMS73\ADMIN
```

- d. Verificar que el script se ejecuto correctamente haciendo un select de la vista del diccionario de datos ALL_OBJECTS que tengan un status de 'INVALID' de la siguiente manera

```
SELECT * FROM ALL_OBJECTS  
WHERE STATUS = 'INVALID';
```

- e. Si se encuentran paquetes inválidos, estos tendrán que compilarse manualmente con el comando ALTER PACKAGE *nombre_paquete* COMPILE BODY

Si el script CATREP.SQL se ejecuta de manera exitosa, varias tablas de replicación serán creadas en el tablespace de SYSTEM. En este punto la base de datos esta configurada para la replicación avanzada.

4. Configuración de SQL*Net.

Un proceso listener de SQL*NET debe estar ejecutándose en cada uno de los servidores del ambiente de replicación y el archivo listener.ora debe tener una entrada para la instancia que será usada. Además en cada servidor se debe tener un archivo tnsnames.ora con la información del resto de los nodos y las bases de datos del ambiente distribuido. Después de esto es necesario probar la conexión vía SQL*Net desde cada servidor al resto de los nodos.

6. Creación de usuarios de replicación.

Para que la configuración sea exitosa, los usuarios administradores del ambiente de replicación deben tener los mismos nombres de usuario ya que estos son pasados entre los nodos como argumentos de las llamadas a procedimientos remotos (RPCs que revisamos anteriormente)

Los usuarios que deben tener los mismos nombres en todos los nodos son:

- a. **Administrador de replicación sustituto (Surrogate Administrator)** En este caso llamaremos al usuario REPSYS. Como ya se definió, este usuario permite realizar actividades como

usuario administrador SYS, sin tener todos sus privilegios. La manera de crearlo es con los siguientes comandos:

```
CREATE USER repsys IDENTIFIED BY repsys;  
ALTER USER repsys DEFAULT TABLESPACE users;  
ALTER USER repsys TEMPORARY TABLESPACE temp;  
GRANT connect, resource TO repsys;  
EXECUTE dbms_repcat_auth.grant_surrogate_repcat('repsys');
```

- b. El otro usuario es el administrador de replicación. Le daremos el nombre de REPADMIN. Este usuario configura el ambiente de replicación de todos los esquemas replicados. Los comandos para su creación son:

```
CREATE USER repadmin IDENTIFIED BY repadmin;  
ALTER USER repadmin DEFAULT TABLESPACE users;  
ALTER USER repadmin TEMPORARY TABLESPACE temp;  
GRANT connect, resource TO repadmin;  
EXECUTE dbms_repcat_admin.grant_admin_any_repgroup('repadmin');
```

Si se trata de un nodo de tipo Snapshots actualizables, el usuario REPADMIN requiere el siguiente privilegio:

```
GRANT lock any table TO repadmin;
```

- c. El tercer usuario es el dueño del esquema. Lo definiremos como REPDDBA. Este usuario es responsable de la administración diaria del esquema y puede ser el mismo o diferente al usuario REPADMIN, sin embargo por razones de seguridad usaremos un usuario diferente:

```
CREATE USER repdba IDENTIFIED BY repdba;  
ALTER USER repdba DEFAULT TABLESPACE users;  
ALTER USER repdba TEMPORARY TABLESPACE temp;  
GRANT connect, resource TO repdba;  
GRANT execute ON dbms_defer TO repdba;
```

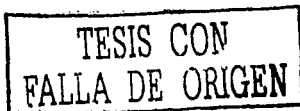
Si el usuario REPDDBA es diferente de REPADMIN y se quiere que los usuarios administren su propio esquema, estos usuarios necesitan un privilegio adicional:

```
EXECUTE dbms_repcat_admin.grant_admin_repgroup('repdba');
```

7. Database links.

Antes de definir los links de base de datos es necesario asegurarse que todos los nodos del ambiente distribuido tengan un nombre global (GLOBAL_NAME) único. El nombre global por definición es el mismo que el nombre de la base de datos, que puede cambiarse solamente recreando el archivo de control. Para cambiar el nombre global hay que ejecutar el siguiente comando:

```
ALTER DATABASE RENAME GLOBAL_NAME TO mexico.us.oracle.com;  
SELECT * FROM GLOBAL_NAME;
```



Antes de proceder, es necesario asegurarse que cada usuario involucrado en la replicación simétrica se conecte a cada uno de los nodos usando los links creados. La configuración de los links para REPSYS, REPADMIN y REPDBA es como sigue:

- a. Crear un link público para que las transacciones iniciadas por los usuarios finales sean replicadas.

```
CREATE PUBLIC DATABASE LINK orlando.us.oracle.com
USING 'orlando.us.oracle.com';
```

El alias de SQL*Net debe ser el mismo al definido para la base de datos remota en el archivo tnsnames.ora del servidor.

- b. Un link privado de SYS hacia el usuario administrador sustituto (que llamamos REPSYS)

```
CONNECT INTERNAL
CREATE DATABASE LINK orlando.us.oracle.com
CONNECT TO repsys IDENTIFIED BY repsys
USING 'orlando.us.oracle.com';
```

- c. Para el usuario administrador (que ya fue creado como REPADMIN)

```
CONNECT repadmin/repadmin
CREATE DATABASE LINK orlando.us.oracle.com;
CONNECT TO repadmin IDENTIFIED BY repadmin
USING 'orlando.us.oracle.com';
```

Antes de proceder todos los links deben ser probados manualmente usando:

```
SELECT * FROM DUAL@orlando.us.oracle.com;
```

Resumiendo, hay que verificar que los links funcionen en ambas direcciones, por ejemplo para sistemas con dos nodos de replicación los links básicos son:

Nodo maestro de definición
mexico.us.oracle.com

Segundo Nodo Maestro
orlando.us.oracle.com

SYS	=>	REPSYS
REPSYS	<=	SYS
REPADMIN	=>	REPADMIN
REPADMIN	<=	REPADMIN
REPDBA	=>	REPDBA
REPDBA	<=	REPDBA

El siguiente paso es usar Replication Manager para configurar los objetos que se quieren replicar. Estas tareas deben completarse siempre como usuario REPADMIN en la base de datos que se designe como nodo maestro de definición (*Master Definition Site*)

El **Administrador de Replicación** (*Replication Manager*) es un API provisto por Oracle para la configuración y administración de una base de datos distribuida que implementa replicación.

Puede configurar ambos tipos de replicación Multi-master, de nodos de snapshots actualizables y de sólo lectura. Y con el uso de asistentes paso a paso configura los diferentes objetos requeridos, que van desde los nodos maestros y de snapshot, la manera de propagar los datos entre los nodos, grupos de replicación, snapshots y snapshots logs; además de definir rutinas de solución de conflictos.

La **Figura 5.16** muestra la pantalla del **Administrador de Replicación**. La pantalla esta compuesta por un árbol jerárquico en la parte izquierda para recorrer los diferentes componentes. La parte derecha mostrara el detalle de la hoja seleccionada en el árbol. La imagen muestra la sección de administración del nodo maestro de definición *mexico.us.oracle.com* y en la parte derecha se puede ver la topología formada por los dos nodos replicados y la dirección de la propagación que es en ambos sentidos.

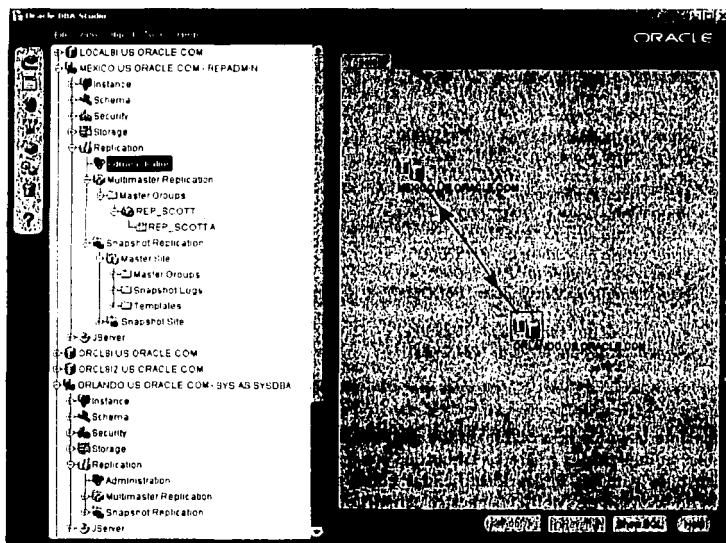


Figura 5.16 Pantalla del administrador de replicación (*Replication Manager*)

Se deben registrar los dos usuarios dueños de la aplicación CD2000 y PMS2000 para incluir los objetos en los grupos de replicación, esto se hace desde el Administrador de Replicación, en la opción de Configuración de nodos maestro.

Las sentencias generadas una vez que el asistente es completado se encuentran en el Listado 1. y contiene lo siguiente:

Listado 1. Creación de usuarios dueños de la aplicación en ambos nodos.

```
/*Conectándose al nodo ORLANDO.US.ORACLE.COM y MEXICO.US.ORACLE.COM como
usuario administrador SYSTEM...*/
```

```
grant alter session to "CD2000"
grant create cluster to "CD2000"
grant create database link to "CD2000"
grant create sequence to "CD2000"
grant create session to "CD2000"
grant create synonym to "CD2000"
grant create table to "CD2000"
grant create view to "CD2000"
grant create procedure to "CD2000"
grant create trigger to "CD2000"
grant unlimited tablespace to "CD2000"
grant create type to "CD2000"
grant create any snapshot to "CD2000"
grant alter any snapshot to "CD2000"
grant alter session to "PMS2000"
grant create cluster to "PMS2000"
grant create database link to "PMS2000"
grant create sequence to "PMS2000"
grant create session to "PMS2000"
grant create synonym to "PMS2000"
grant create table to "PMS2000"
grant create view to "PMS2000"
grant create procedure to "PMS2000"
grant create trigger to "PMS2000"
grant unlimited tablespace to "PMS2000"
grant create type to "PMS2000"
grant create any snapshot to "PMS2000"
grant alter any snapshot to "PMS2000"
```

La siguiente etapa es la creación de los **grupos de replicación**. Se crearán dos grupos uno con las tablas **actualizables** y para las tablas que son de **lectura** en el nodo orlando.us.oracle.com. Se dividen pues para las tablas identificadas como solamente de lectura no es necesario definir rutinas de solución de conflictos pues se actualizan en el nodo mexico.us.oracle.com y se pueden propagar cuando estas son cambiadas.

El Listado 2, en la hoja siguiente contiene las sentencias generadas para la creación del grupo PMS2000_LECTURA que incluye las tablas del esquema PMS2000 identificadas antes en la Tabla 5.17.

Además de la creación del grupo e inclusión de las tablas, se incluyen las llamadas al API DBMS_REPCAT usado para generar los triggers internos para permitir la propagación entre los nodos maestros. Un listado similar será generado para el grupo CD2000_LECTURA.

Listado 2. Sentencias generadas para creación de grupo PMS2000_lectura.

```
BEGIN
  DBMS_REPCAT.CREATE_MASTER_REPGROUP(
    gname => "PMS2000_LECTURA",
    qualifier => "",
    group_comment => 'Objetos de solo lectura del esquema PMS2000');
END;
/
BEGIN
  DBMS_REPCAT.CREATE_MASTER_REOBJECT(
    gname => "PMS2000_LECTURA",
    type => TABLE,
    oname => "ACTION_TYPE_CODES",
    sname => "PMS2000",
    copy_rows => TRUE,
    use_existing_object => TRUE);
END;
/
BEGIN
  DBMS_REPCAT.CREATE_MASTER_REOBJECT(
    gname => "PMS2000_LECTURA",
    type => TABLE,
    oname => "APPLICATION_PREFERENCES",
    sname => "PMS2000",
    copy_rows => TRUE,
    use_existing_object => TRUE);
END;
/
.....

BEGIN
  DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT(
    sname => "PMS2000",
    oname => "SEVERITY_TYPES",
    type => TABLE,
    min_communication => TRUE,
    generate_80_compatible => FALSE);
END;
/
BEGIN
  DBMS_REPCAT.RESUME_MASTER_ACTIVITY(
    gname => "PMS2000_LECTURA");
END;
/
```


Una vez creados los grupos, estos aparecen en el Administrador de replicación en el nodo maestro `mexico.us.oracle.com`. Se debe revisar que el estatus de la propagación de datos este activa, ya que cada grupo puede estar activo (propagando) o inactivo (para hacer tareas administrativas)

La **Figura 5.19** Muestra el grupo `PMS2000_LECTURA` en el Administrador de replicación, incluye las tablas que son parte del grupo, y en la parte derecha se muestran el estatus del grupo que es `RUNNING (Normal)` que significa que los cambios en el nodo México se están propagando al nodo `orlando.us.oracle.com`

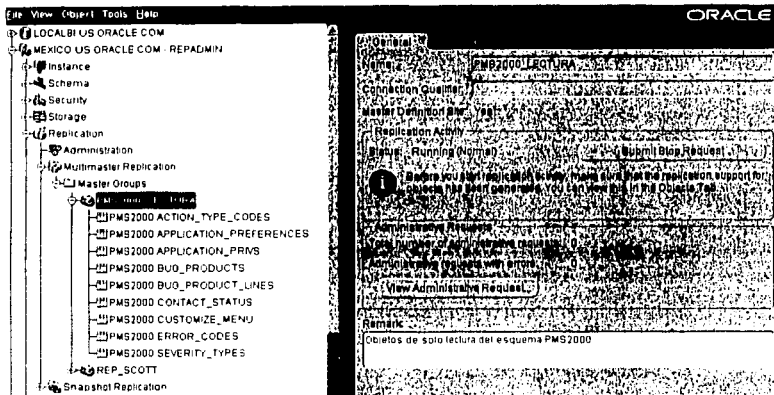


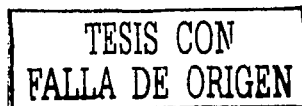
Figura 5.17 Grupo de replicación `PMS2000_LECTURA`.

La siguiente etapa es la creación de los grupos `CD2000` Y `PMS2000` que incluirán las tablas y otros objetos relacionados que serán actualizables en ambos nodos. Las tablas son la base de las formas de captura identificadas en los procesos a replicar.

El primer proceso es el de **Registro de nuevos clientes y actualización de información de contactos** donde la forma de la aplicación esta basada en las tablas del esquema `CD2000` llamadas: `COMPANY`, `PERSON`, `ADDRESS` y `CD2000_SEQ`.

La actualización de estas tablas se realiza solamente en el nodo `MÉXICO`, con excepción de la tabla `PERSON` que registra los contactos del cliente y esta información se definió como actualizable en los dos nodos, por lo que sólo es necesario agregar el método de solución de conflictos para la tabla `PERSON`.

Ya incluidas las tablas y sus llaves primarias, se deben modificar los triggers de la aplicación para que no propaguen cambios hechos por el proceso de replicación, solamente los que son



generados por el usuario a través de la aplicación. El Listado 3. muestra los triggers que serán cambiados y creados externamente en el nodo ORLANDO.

Los triggers a modificar son:

<u>Tabla</u>	<u>Nombre del trigger</u>	<u>Evento que lo dispara</u>
ADDRESS	LOG_ADDRESS	INSERT, UPDATE
COMPANY	LOG_COMPANY	INSERT, UPDATE
PERSON	LOG_PERSON	INSERT, UPDATE

Listado 3. Cambios a triggers en ambos nodos, inclusión de paquete dbms_repadmin.

```

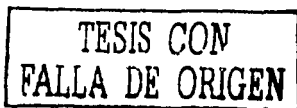
CREATE OR REPLACE TRIGGER "CD2000".Log_address
Before Insert Or Update On address
For Each Row
Begin
If dbms_reputil.from_remote=false then
:new.ad_last_changed := sysdate;
:new.ad_who_changed := user;
end if;
End;
/
CREATE OR REPLACE TRIGGER "CD2000".Log_PERSON
Before Insert Or Update On PERSON
For Each Row
Begin
If dbms_reputil.from_remote=false then
:new.pe_last_changed := sysdate;
:new.pe_who_changed := user;
end if;
End;
/
CREATE OR REPLACE TRIGGER "CD2000".Log_COMPANY
Before Insert Or Update On COMPANY
For Each Row
Begin
If dbms_reputil.from_remote=false then
:new.co_last_changed := sysdate;
:new.co_who_changed := user;
end if;
End;
/
    
```

El Listado 4. muestra las sentencias ejecutadas por el Administrador de replicación para el grupo de replicación CD2000_ACTUALIZACION que contiene las tablas y procedimientos necesarios para el primer proceso.

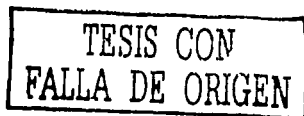
Listado 4. Generación del grupo de replicación CD2000_ACTUALIZACION.

```

BEGIN
DBMS_REPCAT.CREATE_MASTER_REPGROUP(
gname => "CD2000_ACTUALIZACION";
    
```



```
    qualifier => ",
    group_comment => 'Contiene tablas, procedimientos de los procesos replicados';
END;
/
BEGIN
  DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
    gname => "CD2000_ACTUALIZACION";
    type => 'TABLE';
    oname => "CD2000_SEQ";
    sname => "CD2000";
    copy_rows => TRUE,
    use_existing_object => FALSE);
END;
/
BEGIN
  DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
    gname => "CD2000_ACTUALIZACION";
    type => 'TABLE';
    oname => "COMPANY";
    sname => "CD2000";
    copy_rows => TRUE,
    use_existing_object => FALSE);
END;
/
BEGIN
  DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
    gname => "CD2000_ACTUALIZACION";
    type => 'TABLE';
    oname => "ADDRESS";
    sname => "CD2000";
    copy_rows => TRUE,
    use_existing_object => FALSE);
END;
/
BEGIN
  DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
    gname => "CD2000_ACTUALIZACION";
    type => 'TABLE';
    oname => "PERSON";
    sname => "CD2000";
    copy_rows => TRUE,
    use_existing_object => FALSE);
END;
/
BEGIN
  DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
    gname => "CD2000_ACTUALIZACION";
    type => 'INDEX';
    oname => "PERSON_IX_AR_ID";
    sname => "CD2000";
    copy_rows => TRUE,
    use_existing_object => FALSE);
END;
/
BEGIN
```



```
DBMS_REPCAT.CREATE_MASTER_REPOBJECT(  
  gname => "CD2000_ACTUALIZACION",  
  type => 'INDEX',  
  oname => "PERSON_IX_AR_ORIG_SYSTEM_REF",  
  sname => "CD2000",  
  copy_rows => TRUE,  
  use_existing_object => FALSE);  
END;  
/  
BEGIN  
  DBMS_REPCAT.CREATE_MASTER_REPOBJECT(  
    gname => "CD2000_ACTUALIZACION",  
    type => 'INDEX',  
    oname => "PERSON_IX_COID_PEID",  
    sname => "CD2000",  
    copy_rows => TRUE,  
    use_existing_object => FALSE);  
END;  
/  
BEGIN  
  DBMS_REPCAT.CREATE_MASTER_REPOBJECT(  
    gname => "CD2000_ACTUALIZACION",  
    type => 'INDEX',  
    oname => "PE_PK",  
    sname => "CD2000",  
    copy_rows => TRUE,  
    use_existing_object => FALSE);  
END;  
/  
BEGIN  
  DBMS_REPCAT.CREATE_MASTER_REPOBJECT(  
    gname => "CD2000_ACTUALIZACION",  
    type => 'INDEX',  
    oname => "AD_PK",  
    sname => "CD2000",  
    copy_rows => TRUE,  
    use_existing_object => FALSE);  
END;  
/  
BEGIN  
  DBMS_REPCAT.CREATE_MASTER_REPOBJECT(  
    gname => "CD2000_ACTUALIZACION",  
    type => 'INDEX',  
    oname => "SE_PK",  
    sname => "CD2000",  
    copy_rows => TRUE,  
    use_existing_object => FALSE);  
END;  
/  
BEGIN  
  DBMS_REPCAT.CREATE_MASTER_REPOBJECT(  
    gname => "CD2000_ACTUALIZACION",  
    type => 'INDEX',  
    oname => "COMPANY_NI",  
    sname => "CD2000",
```

TESIS CON
FALLA DE ORIGEN

```

        copy_rows => TRUE,
        use_existing_object => FALSE);
END;
/
BEGIN
DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
  gname => "CD2000_ACTUALIZACION";
  type => 'INDEX';
  oname => "CO_IX_AC_REP";
  sname => "CD2000";
  copy_rows => TRUE,
  use_existing_object => FALSE);
END;
/
BEGIN
DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
  gname => "CD2000_ACTUALIZACION";
  type => 'INDEX';
  oname => "CO_NAME_IX";
  sname => "CD2000";
  copy_rows => TRUE,
  use_existing_object => FALSE);
END;
/
BEGIN
DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
  gname => "CD2000_ACTUALIZACION";
  type => 'INDEX';
  oname => "CO_PK";
  sname => "CD2000";
  copy_rows => TRUE,
  use_existing_object => FALSE);
END;
/
BEGIN
DBMS_REPCAT.ADD_MASTER_DATABASE(
  gname => "CD2000_ACTUALIZACION";
  master => 'ORLANDO.US.ORACLE.COM';
  use_existing_objects => FALSE,
  copy_rows => TRUE,
  propagation_mode => 'SYNCHRONOUS');
END;
/
BEGIN
DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT(
  sname => "CD2000";
  oname => "CD2000_SEQ";
  type => 'TABLE';
  min_communication => TRUE,
  generate_80_compatible => FALSE);
END;
/
BEGIN
DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT(
  sname => "CD2000";

```

```

    oname => "COMPANY";
    type => TABLE;
    min_communication => TRUE;
    generate_80_compatible => FALSE);
END;
/
BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT(
        sname => "CD2000";
        oname => "ADDRESS";
        type => TABLE;
        min_communication => TRUE;
        generate_80_compatible => FALSE);
END;
/
BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT(
        sname => "CD2000";
        oname => "PERSON";
        type => TABLE;
        min_communication => TRUE;
        generate_80_compatible => FALSE);
END;
/
BEGIN
    DBMS_REPCAT.RESUME_MASTER_ACTIVITY(
        gname => "CD2000_ACTUALIZACION");
END;
/

```

Como se definió, los contactos pueden ser creados y modificados en ambos nodos, para evitar los problemas de inserción de nuevos contactos y debido a que el ID del contacto es generado usando una *secuencia de base de datos* es necesario crear la misma secuencia en el nodo ORLANDO.

Como no es permitido replicar secuencias porque el mismo valor podría ser generado en los dos nodos. Un mecanismo muy simple para solucionarlo es alterar la secuencia original para que los siguientes valores sean sólo pares y crear una nueva secuencia en el otro nodo maestro que inicie en el siguiente valor par y posteriormente generando otros pares. De esta manera se evita cualquier conflicto por posibles ID's duplicados.

El Listado 5, muestra los cambios a la secuencia existente en el nodo MÉXICO y la manera de crear la misma secuencia en el otro nodo para que genere solamente valores pares.

Listado 5. Alteración y creación de secuencia CONTACT_SEQ de la tabla PERSON.

```

/*En el nodo México.us.oracle.com la secuencia tiene un último valor
generado de 864983*/
alter sequence contact_seq
increment by 2;

```

```
/* En el nodo orlando.us.oracle.com crear la misma secuencia comenzando en
864984 e incrementándola en 2 */
create sequence contact_seq
increment by 2 start with 864984
nomaxvalue ;
```

El siguiente paso es agregar el mecanismo de solución de conflictos en la tabla PERSON. El método usado está basado en la columna de última actualización de tipo DATE (*pe_last_changed*) prevalectiendo el valor del nodo que actualizo el registro más recientemente.

Usando el **Administrador de replicación** como se muestra en la **Figura 5.20** se crea el grupo de columnas *person_conflicts* y se asigna el método *LATEST_TIMESTAMP* sobre la columna *PE_LAST_CHANGED*

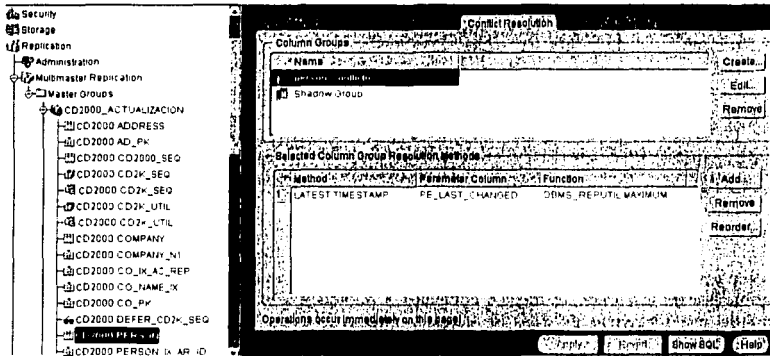


Figura 5.18 Creación de solución de conflictos para tabla PERSON

Cada vez que una tabla es alterada en el **Administrador de replicación** es necesario crear nuevamente el soporte para replicación para la tabla, esto se hace con el paquete *DBMS_REPCAT*, en este caso, se propagará el método de solución de conflicto al resto de los nodos maestro.

Conectado como usuario *REPADMIN* en el nodo *mexico.us.oracle.com* se ejecuta la sentencia descrita en el **Listado 6**, enseguida.

Listado 6. Regeneración del soporte a replicación para la tabla PERSON

```
/* Como usuario REPADMIN en nodo México */
BEGIN
  DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT(
    sname => "CD2000";
```



```
oname => "PERSON";  
type => TABLE;  
min_communication => TRUE,  
generate_80_compatible => FALSE);  
END;
```

El siguiente proceso a replicar es el de **Captura de nuevos contratos**. Parte de los objetos identificados en la **Tabla 5.12** pertenecientes al este proceso ya han sido incluidos en el grupo **CD2000_ACTUALIZACION**.

Las tablas faltantes son del esquema **CD2000**, el tipo de operación sobre ellas es de sólo lectura en el nodo **ORLANDO** y pueden ser modificadas en el nodo maestro de definición por lo que no es necesaria definir alguna rutina de resolución de conflictos. Las tablas por agregar son: **MACHINE**, **PROD_LIC** y **VER_HISTORY**.

El **Listado 7**, generado con el Administrador de replicación agrega las tablas al grupo **CD2000_ACTUALIZACION** y genera el soporte de replicación para estos objetos.

Listado 7. Inclusión de tablas *machine*, *prod_lic* y *ver_history* en grupo de replicación.

```
BEGIN  
DBMS_REPCAT.CREATE_MASTER_REPOBJECT(  
  gname => "CD2000_ACTUALIZACION",  
  type => TABLE,  
  oname => "MACHINE",  
  sname => "CD2000",  
  copy_rows => TRUE,  
  use_existing_object => TRUE);  
END;  
/  
BEGIN  
DBMS_REPCAT.CREATE_MASTER_REPOBJECT(  
  gname => "CD2000_ACTUALIZACION",  
  type => TABLE,  
  oname => "PROD_LIC",  
  sname => "CD2000",  
  copy_rows => TRUE,  
  use_existing_object => TRUE);  
END;  
/  
BEGIN  
DBMS_REPCAT.CREATE_MASTER_REPOBJECT(  
  gname => "CD2000_ACTUALIZACION",  
  type => INDEX,  
  oname => "MACHINE_IX_CID_MID",  
  sname => "CD2000",  
  copy_rows => TRUE,  
  use_existing_object => TRUE);  
END;  
/  
BEGIN
```



```
DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
  gname => "CD2000_ACTUALIZACION",
  type => 'INDEX',
  oname => "MACHINE_IX_CID_OSID",
  sname => "CD2000",
  copy_rows => TRUE,
  use_existing_object => TRUE);
END;
/
BEGIN
DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
  gname => "CD2000_ACTUALIZACION",
  type => 'INDEX',
  oname => "MACHINE_IX_CID_RENEWAL",
  sname => "CD2000",
  copy_rows => TRUE,
  use_existing_object => TRUE);
END;
/
BEGIN
DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
  gname => "CD2000_ACTUALIZACION",
  type => 'INDEX',
  oname => "MACHINE_IX_CSI#";
  sname => "CD2000",
  copy_rows => TRUE,
  use_existing_object => TRUE);
END;
/
BEGIN
DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
  gname => "CD2000_ACTUALIZACION",
  type => 'INDEX',
  oname => "PL_PK";
  sname => "CD2000",
  copy_rows => TRUE,
  use_existing_object => TRUE);
END;
/
BEGIN
DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
  gname => "CD2000_ACTUALIZACION",
  type => 'INDEX',
  oname => "PRODLIC_IX_CID_PRID";
  sname => "CD2000",
  copy_rows => TRUE,
  use_existing_object => TRUE);
END;
/
BEGIN
DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
  gname => "CD2000_ACTUALIZACION",
  type => 'TABLE',
  oname => "VER_HISTORY";
  sname => "CD2000";
```

```
copy_rows => TRUE,  
use_existing_object => TRUE);  
END;  
/  
BEGIN  
DBMS_REPCAT.CREATE_MASTER_REPOBJECT(  
  gname => "CD2000_ACTUALIZACION",  
  type => 'INDEX',  
  oname => "VERHIST_IX_COID_MID_PRID",  
  sname => "CD2000",  
  copy_rows => TRUE,  
  use_existing_object => TRUE);  
END;  
/  
BEGIN  
DBMS_REPCAT.SET_COLUMNS(  
  sname => "CD2000",  
  oname => "VER_HISTORY",  
  column_list => "VH_C_ID","VH_M_ID","VH_PR_ID");  
END;  
/  
/
```

Cada vez que se hace un cambio a los objetos replicados o a los grupos de replicación la transferencia de transacciones entre los grupos es detenida, el grupo es colocado en un estado suspendido o "quiesced" en todos los nodos; al completarse el Administrador de replicación terminara los comandos generados con el comando mostrado en el Listado 8. que retorna el grupo al estatus normal o "running" que significa que esta listo para recibir y propagar transacciones.

Listado 8. Sentencia generada para resumir la replicación de transacciones.

```
BEGIN  
DBMS_REPCAT.RESUME_MASTER_ACTIVITY(  
  gname => "CD2000_ACTUALIZACION");  
END;  
/  
/
```

5.4 Revisión de resultados.

Hasta ahora se replicaron dos procesos y se resolvieron los posibles escenarios que pueden surgir en la implementación del ambiente de replicación para el sistema Voyager/2000. Los dos procesos restantes que deben ser replicados, son repeticiones (con diferentes tablas) de los pasos seguidos anteriormente.

Fueron definidos grupos de replicación con tablas de información de referencia y validación de la aplicación, propagados en demanda y no en tiempo real.

Se crearon además grupos con tablas de la aplicación que son modificados en ambos nodos y que requirieron de la asignación de un mecanismo para solución de conflictos que pueda garantizar la convergencia de datos.

TESIS CON
FALLA DE ORIGEN

Se alteraron en cada nodo secuencias que no pueden ser replicadas y se modificaron triggers de tablas de manera que se ejecuten solamente cuando un DML ocurre a través de la aplicación y no por la propagación de cambios desde otro nodo.

La aplicación Voyager/2000 esta formada por una cantidad importante de paquetes y procedimientos que realizan diversas validaciones, esto ocurre también con otras tablas que son usadas por procesos que no tuvieron que ser replicados pues son parte de las operaciones hechas en la subsidiaria.

Aún cuando es posible incluir estos objetos en los grupos de replicación no fue necesario por dos razones. Primero, se hubiera agregado complejidad para la administración del ambiente distribuido; y segundo, el objetivo de incluirlos en los grupos es el de permitir que cualquier cambio, no solo en los datos sino en la estructura, sea propagado de manera automática al resto de los nodos, pero al ser paquetes internos no se pueden modificar.

El paso final que debe ser parte de la verificación de cada proceso y que permitirá evitar conflictos u operaciones prohibidas desde un nodo es la definición de **ROLES DE BASE DE DATOS** (*database role*), un rol es un grupo de privilegios sobre objetos y las operaciones que se pueden ejecutar sobre ellos. El rol es asignado a los usuarios y ellos heredan los permisos de acceso otorgados al rol. De esta manera tenemos un nivel adicional para protegernos contra conflictos por actualizaciones. Se tienen los mecanismos de la replicación avanzada como rutinas de solución de conflictos y las transacciones distribuidas que usan el protocolo de dos fases; y con el uso de roles se tiene la posibilidad de limitar el tipo operaciones que cada usuario puede realizar desde cada nodo.

Una vez concluida la construcción del ambiente distribuido, se debe revisar cada proceso en relación con las hipótesis de la tesis.

La primer hipótesis establece.

"Al acercar la fuente de datos a las aplicaciones de usuario, usando una base de datos distribuida se cumplen las propiedades que las definen y se mejora el tiempo de procesamiento de datos en comparación con el que se tiene con una base de datos centralizada."

Las doce reglas de C. J. Date son cumplidas por las bases de datos Oracle que participan en un ambiente distribuido, esto fue demostrado en el primer capítulo. Inclusive para las reglas 2 y 3 que definen que *"no se debe depender de un nodo central"* y *"permitir operación continua"* respectivamente.

Aun teniendo un nodo maestro de definición, no se depende de un nodo central, pues en caso de necesitarse es posible cambiarlo dinámicamente usando el Administrador de replicación y asignarlo a otro nodo maestro.

La operación continua es deseada inclusive en bases de datos centralizadas, esto implica el realizar cualquier tarea de mantenimiento sin usuarios o procesos programados cambiando datos.

En caso de requerirse, estas actividades se planean para hacerse fuera de horas de operación. Si se tuviese una emergencia en los esquemas replicados, los grupos deben ser suspendidos para permitir que los cambios se propaguen al resto de los nodos. Si bien se pueden leer datos, no es

posible cambiarlos hasta que se reestablezca el estatus normal de los grupos. Por fortuna la interrupción en la operación es minimizada pues las opciones de replicación de Oracle permiten al administrador modificar las estructuras en un nodo y estas acciones son automáticamente aplicadas en el resto de las localidades.

La segunda parte de la hipótesis dice que al acercar la fuente de datos al nodo que los requiere se mejora el tiempo de procesamiento en los datos en comparación con una base de datos centralizada.

La Tabla 5.19 muestra una tabla comparativa del tiempo de respuesta promedio de 10 transacciones conectándose desde el PSC a la subsidiaria y a la instancia en el nodo ORLANDO.

Revisión de tiempos promedio de respuesta / 10 transacciones (en segundos)			
Proceso	Nodo	Base de datos Centralizada	NA - No Aplica
			Base de datos distribuida
<u>Creación de Clientes y contactos</u> (Tiempo para completar el commit)	PSC México	NA <= 1	NA <= 1
<u>Actualización de contactos</u> (Tiempo para completar el commit)	PSC México	5 <= 1	
<u>Captura y renovación de contratos</u> (Tiempo para completar el commit)	PSC México	NA <= 1	NA <= 1
<u>Actualización de versiones</u> (Tiempo para completar el commit)	PSC México	NA <= 1	NA <= 1
<u>Revisión de contratos de soporte</u>	PSC México	7 <= 1	<= 1 <= 1
<u>Registro de actualización de productos</u> (Tiempo para completar el commit)	PSC México	6 NA	<=1 NA

Tabla 5.19 Tabla comparativa de tiempos de respuesta entre la base de datos centralizada y distribuida.

Se puede ver que los tiempos de respuesta se mejoraron al acercar la fuente de datos a los usuarios. Se prueba con esto que la primer hipótesis es verdadera pues la replicación de datos es una alternativa válida del uso de bases de datos centralizadas, especialmente cuando los clientes y servidores están dispersos en redes WAN. Un tiempo de espera mínimo o casi inexistente terminara con la frustración de los analistas al usar la aplicación.

TESIS CON
 FALLA DE ORIGEN

La segunda hipótesis establece:

“La construcción de un esquema de replicación, con mecanismos existentes a partir de la versión Oracle7, se facilita usando un API gráfico de administración e implementación de bases de datos distribuidas.”

Como se reviso, todo el proceso de implementación y mantenimiento esta basado y debe hacerse usando múltiples paquetes provistos por Oracle para el manejo de las estructuras internas que permiten replicar datos.

Además de los paquetes se cuenta con un diccionario de replicación que es una herramienta adicional, en forma de vistas de diccionario de datos con la que cuenta el DBA y el usuario REPADMIN. Las vistas para el manejo de la replicación se incluyen en el Apéndice B y son una herramienta de consulta para resolver problemas o determinar el estatus del ambiente distribuido.

Las bases de datos distribuidas agregan complejidad a la administración e implementación. Y pueden variar dependiendo del tipo de ambiente que se va a crear. El llevar control de los componentes involucrados sería casi imposible. El uso del API “Administrador de replicación” usado para implementar el caso de estudio permitió encapsular parte de esta complejidad en asistentes gráficos.

Para crear y modificar el ambiente, el API combina la información existente en el diccionario de replicación con guías gráficas, las que generan las llamadas necesarias a los paquetes de replicación. De esta manera en minutos se pueden producir las sentencias necesarias para ir construyendo cada estructura del esquema. El API además puede usarse para administrar un ambiente ya creado, no se limita a la fase de construcción.

Las sentencias que genera pueden ser listadas para su revisión, como las incluidas antes en el presente capítulo. Por salvar complejidad, integrar los componentes del ambiente, permitir determinar el estatus, y por la facilidad de uso se comprueba entonces la segunda hipótesis como verdadera pues el API “Administrador de replicación” facilita el construir una base de datos distribuida.

Conclusiones.

Hoy en día las organizaciones tienden a separar sus procesos en diferentes localidades, que además de requerir información la generan, no sólo para ellas sino para el resto de la organización. La necesidad de distribuir datos comunes puede ir más allá de los límites físicos de una red de área local y abarcar diferentes zonas geográficas.

Las tecnologías para distribución de datos han probado ser una solución confiable que permiten compartir información de manera controlada. Abarcan además de la problemática para la distribución, las diferencias entre los sistemas operativos, tipos de red de comunicación y hasta diferencias de los sistemas de manejo de base de datos que pueden ser no-relacionales.

A la par de la implementación, existen metodologías para repartir óptimamente datos y procesos, comenzando desde el modelado y análisis de la naturaleza distribuida de la aplicación, hasta el desarrollo del ambiente distribuido. El resultado final de una base de datos distribuida es aparecer ante los usuarios como una base de datos única y centralizada.

Como fue revisado, estas tecnologías se desarrollaron a partir de las propiedades enunciadas por C. J. Date en 1987, que en la forma de 12 Reglas Fundamentales, definía las características ideales que debía tener una base de datos distribuida con el objetivo común de mantener interna la complejidad en la distribución de datos.

La distribución de datos puede agruparse en dos tipos, con redundancia en la forma de replicación de datos, y sin redundancia con la fuente única de datos fragmentada horizontal o verticalmente en los diferentes nodos. Son los requerimientos de la aplicación y los recursos de infraestructura los que dictan cual es el mecanismo más conveniente siendo una directriz el intentar colocar los datos cerca de los procesos que los requieren. Ha sido tal el impacto en la industria que múltiples aplicaciones se pueden ver beneficiadas por el uso de las bases de datos distribuidas. Esto impulsó a que diferentes proveedores de software de base de datos hayan implementado diferentes mecanismos para distribuir datos garantizando la consistencia en ambientes multiusuario y además con múltiples nodos.

Como se reviso en los capítulos anteriores, para las bases de datos distribuidas y centralizadas el mantener la integridad de datos en ambientes multiusuario es una premisa importante, para ello se valen de unidades de trabajo llamadas transacciones y bloqueos internos de datos a diferentes niveles que dependiendo de la operación realizada, pueden ir desde un registro, un grupo de ellos o una o más tablas. Este método de bloqueo para las transacciones distribuidas se da en forma del protocolo de bloqueo llamado two-phase commit que coordina con los nodos participantes en la transacción los bloqueos a los registros afectados.

Al usar el Oracle Server como RDBMS nos permitió demostrar que en ambientes distribuidos cumple con las 12 reglas enunciadas por Date que le permite entre otras cosas el interactuar con otros nodos sin importar el sistema operativo, red o enfoque de datos que estén usando. Que como vimos son características deseadas en un ambiente distribuido.

Oracle además integra en el mismo RDBMS la funcionalidad que le permiten fragmentar o replicar datos, y al igual que en una base de datos centralizada permite garantizar la consistencia

Conclusiones

de datos, con el two-phase commit, cuando la propagación de transacciones es de manera sincrónica. E incluir mecanismos para la solución de conflictos que garantizan la convergencia de datos cuando nos encontramos con sistemas con transacciones diferidas.

Las características que tienen las versiones recientes de Oracle están presentes desde la versión de Oracle7, y con el cambio de nuevas versiones han sufrido pocos cambios que principalmente la han mejorado el desempeño de Oracle para propagar cambios con menor tráfico de red.

Se demostró además la importancia que tiene una interfaz gráfica que ayude a la administración e implementación de un ambiente distribuido pues se quiera o no, agrega complejidad a la administración y es muy difícil controlar todos los componentes involucrados sin una herramienta que los organice.

Otra cosa que se demostró es la mejora en desempeño que se gana al usar replicación al acercar las fuentes de datos a los procesos y usuarios que las utilizan, especialmente cuando es considerable la lejanía. Con esto se redujo el tráfico en la red, se dejó la comunicación de transacciones a nivel servidor, y lo más importante esto fue transparente para los usuarios que siguieron percibiendo la base de datos como una base de datos centralizada.

Apéndice A.

Modelo Entidad-Relación

El modelo Entidad-Relación es una técnica para la definición de las necesidades de manejo de información de toda organización. Es una base firme para proveer sistemas de alta calidad y apropiados a las necesidades de negocio.

En su forma más simple, el modelo Entidad-Relación involucra la definición de todas las cosas de importancia dentro de una organización (**entidades**), las propiedades de estas cosas (**atributos**) y la manera como se relacionan unas con otras (**relaciones**)

Los objetivos primarios son: el generar un modelo de información preciso que sirva de base para el desarrollo de un nuevo o mejorado sistema. Además de ser un modelo de datos independiente a cualquier método de almacenamiento y acceso, lo cual permitirá la toma de decisiones objetivas en la implementación y su coexistencia con sistemas existentes.

Existen 8 aspectos importantes en el uso del modelo Entidad-Relación para lograr una correcta definición de las necesidades de información del negocio, estas son:

1. Los datos son un recurso clave. Actualmente se ha reconocido que los datos de la organización tan importantes como las finanzas y los recursos humanos y materiales. Por lo tanto debe tratarse a la información como algo vital
2. Los requerimientos de información deben ser confirmados con la gerencia del negocio, sin importar que tan experimentado se puede ser construyendo modelos de datos.
3. Se deben usar convenciones en los datos, incluso en la fase de definición de conceptos y normalización de datos.
4. Cualquier concepto de datos debe ser definido de manera única, de igual manera, se deben identificar todos los objetos relacionados al nuevo concepto.
5. Identificar patrones de datos dentro de las diversas áreas de la organización que permitan el uso de estructuras comunes de procesamiento y almacenamiento.
6. Se debe cuidar la comunicación con los usuarios finales y evitar con ellos el uso de términos técnicos. Aclararles en todo momento las cosas que son de significado, como lo es, que parte de la información necesita ser conocida o almacenada.
7. La representación de los requerimientos de la información es valiosa solamente si soporta las necesidades funcionales de la organización, cumpliendo los objetivos y las metas de negocio.
8. Todo requerimiento de información debe definirse de tal manera que sea independiente a cualquier método de almacenamiento o acceso, lo cual garantizara una decisión objetiva del negocio y diseño subsecuente.

TESIS CON
FALLA DE ORIGEN

La Figura A.1, ilustra la flexibilidad del diagrama Entidad-Relación como base para la implementación de una base de datos RELACIONAL, DE RED y JERARQUICA.

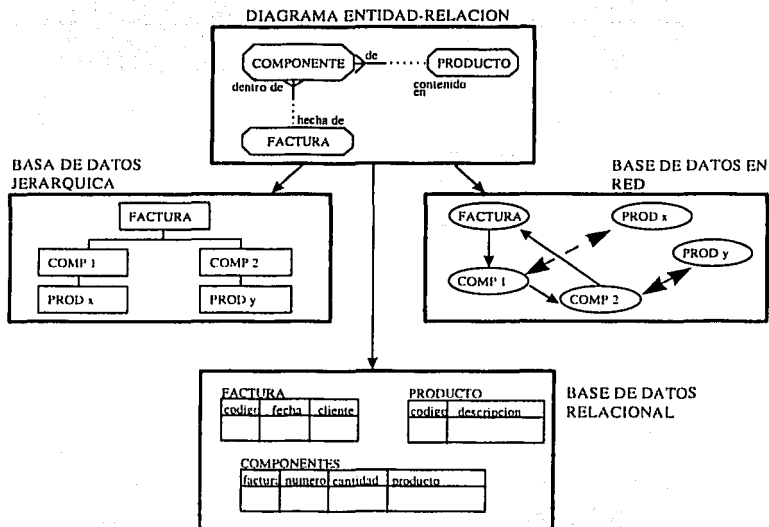


Figura A. 1 Construcción de base de datos a partir del diagrama Entidad-Relación

Conceptos básicos y definiciones.

Definición de Entidad. Una entidad (Figura A.2) es una cosa u objeto de significado, ya sea real o imaginario, acerca de las necesidades de información que necesita conocerse o almacenarse.

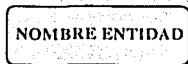


Figura A. 2 Entidad

Gráficamente se representa por un rectángulo con las esquinas redondeadas. El nombre de la entidad es singular (sin abreviaciones) y se despliega en mayúsculas.

El nombre de la entidad debe representar el tipo o clase de objeto que representa, se debe tener cuidado para no usar ocurrencias como nombres de entidades. Si por ejemplo se tiene la entidad ESTADO, Monterrey y Toluca no pueden ser nombres de entidades, son ocurrencias de la entidad ESTADO.

Relación de negocio. Una relación de negocio (Figura A.3) es una asociación o relación significativa entre dos entidades. Las relaciones de negocio son binarias ya que solamente se da entre dos entidades de manera única, o entre una entidad y ella misma.

Cada relación tiene dos terminales, y cada una de ellas tiene las propiedades:

- nombre
- grado / cardinalidad (cuántos)
- existencia (opcional o obligatoria)

Las relaciones están representadas por una línea que une dos entidades, o de manera recursiva a una entidad consigo misma. La relación más común es aquella que tiene un grado de muchos a uno, es obligatoria en el muchos y opcional en el extremo uno como se muestra a continuación.

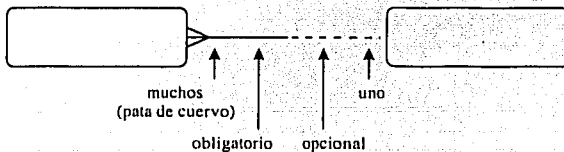


Figura A. 3 Relación

Para el grado muchos, la línea de relación se une a la entidad en tres puntos, llamados 'pata de cuervo'. Para el grado uno se une en un sólo punto. Cuando la relación es obligatoria la línea es sólida. Si es una relación opcional, la línea será punteada. Es útil frecuentemente pensar en una relación uno a muchos como una relación de padre a hijo, con la existencia del hijo de cierta manera dependiente de sus padres.

Una relación recursiva (Figura A.4) con las mismas propiedades mostradas en el ejemplo anterior se muestra a continuación.

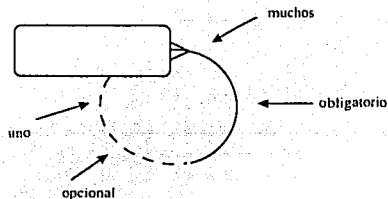


Figura A. 4 Relación recursiva

Los nombres de las relaciones se colocan en los extremos de cada relación y con letras minúsculas (Figura A.5) Cuando el extremo de la relación indica que es obligatoria (línea continua) se debe colocar la frase 'debe de' antes del nombre de la relación; para relaciones opcionales (línea punteada), se usa la frase 'puede ser/estar'.

Consideremos el siguiente ejemplo:

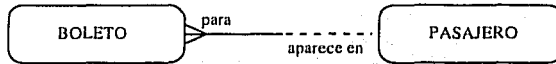


Figura A. 5 Colocación de nombres de relaciones

Leyendo la relación anterior de izquierda a derecha tenemos:

Cada BOLETO debe ser para uno y solo un PASAJERO

y de derecha a izquierda:

Cada PASAJERO puede aparecer en uno o más BOLETO(s)

Es posible usar el plural del nombre de la entidad cuando el grado de la relación es muchos. Y el grado de muchos de la misma se le 'uno o más', mientras que el grado uno es leído como 'uno y sólo uno'. Al dibujar diagramas Entidad-Relación el grado mayor de la relación (el extremo con la para de cuervo), se recomienda colocarlo el extremo izquierdo o superior de la entidad.

Definición de atributos. Un atributo es cualquier detalle que sirve para describir la calidad, identidad, clasificación, cantidad; de igual manera para expresar el estado de una entidad (Figura A.6)

Es la descripción de una propiedad con significado dentro de la entidad. La manera de representarlos es escribiendo su nombre en singular y letras minúsculas, de manera opcional se puede incluir un ejemplo de su valor.

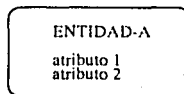


Figura A. 6 Definición de atributos

La lectura de los atributos se puede realizar de una de las siguientes maneras:

nombre de la entidad nombre del atributo

O:

nombre del atributo 'de' nombre de la entidad

Algunas recomendaciones básicas para la definición correcta de los atributos son:

- Todo atributo debe describir la entidad en la que aparece
- No se debe usar como parte del nombre del atributo el nombre de la entidad.
- No deben existir atributos repetidos en una entidad. Si múltiples valores del atributo son requeridos se debe crear una nueva entidad que los contenga con una relación muchos a uno hacia la entidad original. Esta propiedad es conocida como la **primera forma normal (1FN)**
- El nombre del atributo debe ser singular
- Cada entidad debe ser identificable de manera única por un atributo o combinación de atributos, no deben existir atributos repetidos en la entidad que son parte de otro atributo o combinación de estos. Esta característica es conocida como **segunda forma normal (2FN)**
- No debe existir dependencia de ningún tipo entre el valor de un atributo con el valor que puede tener otro atributo de la misma instancia. Esto es conocido como la **tercera forma normal (3FN)**

Los atributos pueden ser opcionales dentro de la entidad u obligatorios (Figura A.7) Cuando el valor del atributo no este presente en todas las ocurrencias de la entidad, la manera de representarlo es con una pequeña 'o' del lado izquierdo del mismo. De igual manera si el atributo es obligatorio, es decir que debe ser siempre conocido, se representará con un '*' al frente del nombre.

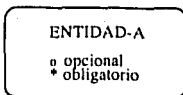


Figura A. 7. Calificación de tipo de atributo

Definición de identificador único. Se trata cuando para cada ocurrencia de la instancia existe un atributo o combinación de atributos que la identifican de manera única de cualquier otra ocurrencia de la instancia. Como ya se mencionó, el identificador puede ser, además de un atributo o combinación de ellos; u a combinación de relaciones, o combinación de relaciones y atributos.

La manera de representar los identificadores únicos en los diagramas Entidad-Relación es colocando un '#' antes de los atributos que conforman el identificador, y colocando una barra atravesando la línea de relación (Figura A.8)

TESIS CON
FALLA DE ORIGEN

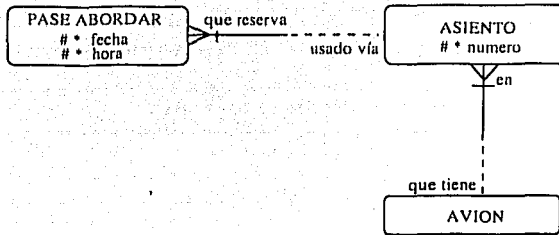


Figura A. 8 identificadores únicos en las entidades

Reglas en el trazado de diagramas Entidad-Relación.

Conjuntos de diagramas. Mientras se discute una área funcional con un usuario o cualquier asunto de diseño con los analistas del sistema es recomendable crear subconjuntos de diagramas y dibujarlos las veces que sean necesarias de manera que estos actúen como un medio efectivo de comunicación. Esto garantizará una rápida y fácil corrección de errores u omisiones.

Limpio y ordenado. Ordenar el diagrama de tal manera que las cajas de entidades se encuentren alineadas, y las líneas de relaciones se orienten principalmente de manera recta vertical u horizontal. Reduzca el cruce de líneas.

Cuando exista la necesidad de cruzar líneas de relaciones, incline una de ellas en un rango de 30 a 60 grados, esto permitirá a la vista seguirlas fácilmente.

Sea consiente que la construcción de un diagrama con un gran número de líneas paralelas es difícil de leer. Es recomendable utilizar bastante espacio para evitar la sensación de congestión.

Uso de nombres. Incluir el título, fecha y nombre del autor(es) en cada diagrama.

Textos. Elimine toda ambigüedad del diagrama, no se deben usar abreviaciones. La mayor parte del texto debe orientarse de manera horizontal para facilitar su lectura. Use mas de una línea para aquellos nombres que provocan problemas de trazado de entidades o relaciones.

Grado de las relaciones. Coloque las relaciones en el extremo muchos (pata de cuervo) en la parte izquierda o superior de la línea de relación.

Esta simple regla a demostrado que aumenta la precisión del modelo al obligar mayor consideración para las relaciones con entidades de mayor ocurrencia sobre aquellas con una menor. Esto se debe a que la mayoría de la gente lee los diagramas de derecha a izquierda y de arriba a abajo, eso sigue la ruta natural. Si se sigue la regla, las entidades que contienen menor cantidad de, aparecerán en el extremo superior derecho del

diagrama y son en general entidades de gran importancia en el modelo; por ejemplo, empresa, producto, aeropuerto. Por lo que leer a través de ellas nos ayudara a entender el papel de otras entidades.

Tamaño de entidades y forma. El tamaño o forma de la entidad no tiene ningún significado especial por lo que las cajas pueden ser estirados, alargados o reducidos para ayudar el trazado del diagrama.

Normalización de datos.

La normalización de datos es un procedimiento que asegura que un modelo de datos cumple con algunos estándares. Para los modelos de datos y Entidad-Relación, estos estándares han sido definidos para minimizar la duplicación de datos, proveer la flexibilidad necesaria para cubrir diversos requerimientos funcionales, y lo más importante que permita que el modelo pueda ser trasladado a un diseño de base de datos.

El modelo Entidad-Relación generalmente produce entidades están por naturaleza ya normalizadas. Esto porque al modelar se siguieron los siguientes pasos:

- Separar de las cosas con significado para el modelo que información se necesita conocer o almacenar. Las entidades son mutuamente excluyentes, y son representadas en un diagrama por una caja o rectángulo con un nombre singular de entidad y usando letras mayúsculas.
- A las entidades se agregaron las relaciones de negocio, que son asociaciones nombradas significativas entre las entidades. Son representadas como una línea entre dos rectángulos; y cada extremo tiene un grado (un triángulo o 'pata de cuervo' significa mucho, no triángulo significa uno) y puede ser obligatoria (una línea punteada quiere decir opcional, una línea sólida obligatoria)
- Para cada entidad se incluyeron que tipo de información necesita conocerse. Estos atributos se representan con nombres en minúsculas dentro de la entidad.
- Finalmente, se garantiza que es posible identificar de manera única cada ocurrencia de la entidad. Esto será a través de la combinación de atributos y/o relaciones. Si el atributo es parte del identificador único se precede de un '#', si se trata de la relación por una línea que atraviesa la línea de relación.

Siguendo el proceso anterior de manera rigurosa concluirá con un modelo normalizado, pero estará basado en como se realizo el análisis de lo que en realidad es un atributo, una relación y una entidad.

Para garantizar que un modelo Entidad-Relación tiene definidas de manera única todas sus entidades y está completamente normalizado, lo que también se conoce como Tercer Forma Normal del modelo, es aplicando las siguientes reglas:

Primera Forma Normal (1FN)

Basados en la premisa de que todas las entidades se identifican de manera única por una combinación de atributos y/o relaciones. La Primera Forma Normal (1FN) del modelo se obtiene *removiendo todos los atributos repetidos*. Si un atributo puede tener más de un

TESIS CON
FALLA DE ORIGEN

valor único a la vez o existe más de un atributo con el mismo nombre, es necesario definir una nueva entidad, que será definida por ese atributo. El identificador único de esta nueva entidad incluirá uno de los atributos que fueron sacados de la entidad original y una relación (muchos a uno) hacia la entidad original.

La primera forma normal es entonces el mecanismo para identificar entidades y relaciones faltantes.

Segunda Forma Normal (2FN)

Eliminar los atributos dependientes del valor que toma un atributo parte del identificador único de cada entidad. Si existe una entidad con un identificador único formado por varios atributos, y si otro atributo de la misma entidad depende del valor que uno de los componentes del identificador compuesto, entonces el atributo dependiente y el que es parte del identificador único deben ser la base para definir una nueva entidad, teniendo una relación uno a muchos hacia la entidad original. Al igual que la primera forma normal, la segunda forma normal nos permitirá identificar entidades faltantes.

Tercera Forma Normal (3FN)

Remover atributos dependientes de otros atributos que no son parte del identificador único de la entidad. Si el valor de un atributo depende del valor de otro dentro de la misma entidad, sin que este sea parte del identificador único, entonces formarán una nueva entidad con una relación muchos a uno a la entidad original.

Apéndice B.

Vistas para el manejo de replicación del diccionario de datos de Oracle7.

Las vistas del diccionario de datos necesarias para el manejo de replicación simétrica pueden ser organizadas en las siguientes categorías:

- Vistas del catalogo de replicación.
- Vistas de transacciones diferidas.
- Vistas de la cola de jobs.
- Vistas de snapshots y de snapshots groups.

Vistas del catalogo de replicación.

En cualquier base de datos donde se instale la opción de replicación simétrica, Oracle instalará de manera automática el catalogo de replicación, que esta formado por tablas y vistas. Estas son usadas por nodos definidos como *master site* y *snapshot site* para determinar información como que objetos están siendo replicados, hacia donde se están replicando, y si algún error se ha presentado durante la replicación.

Ningún usuario bajo ninguna circunstancia debe modificar directamente las tablas del catalogo de replicación; de ser necesaria alguna modificación se deben usar los procedimientos incluidos en el paquete DBMS_REPCAT.

Cada una de las vistas tiene tres versiones, que tienen diferentes prefijos: USER_*, ALL_* y SYS.DBA_*. Para ser leídas por el usuario propietario, los que son de dominio publico y por el DBA respectivamente.

Vista RepGroup.

La vista RepGroup lista todos los grupos de objetos que están siendo replicados. Los miembros de cada grupo de objetos se encuentran en una vista diferente llamada RepObject. La Tabla B.1 contiene las columnas de la vista Repgroup.

Columna	Descripción
Sname	Nombre del esquema replicado. Obsoleto con la versión 7.3 y posteriores.
Gname	Nombre del grupo de objetos replicados.
Master	'Y' indica que es un master site. 'N' se trata de un snapshot site.
Status	Usado sólo en master sites. Los valores pueden ser: normal, quiescing o quiesced.
Schema_comment	Cualquier comentario de usuario.

Tabla B.1 Columnas de la vista RepGroup.

Vista RepCatLog.

La vista RepCatLog en cada master site contiene el estado de cualquier petición y otros mensajes de error generados. Todos los mensajes generados al ejecutar una petición son transferidos al master site y pueden ser consultados en RepCatLog. Si todas las actividades de replicación son completadas exitosamente, al final, toda la información de estatus es removida de la vista RepCatLog. La Tabla B.2 Columnas de vista RepCatLog describe el contenido de las columnas en la vista.

Columna	Descripción
id	Numero de secuencia, juntos ID y SOURCE, identifican todos los registros de log de todos los master sites que pertenecen a la misma petición.
Source	Nodo donde se origino la petición.
userid	Usuario que realizó la petición.
Timestamp	Cuando fue la petición realizada.
role	Indica si el nodo es el 'masterdef' o un 'master'.
Master	Si el role es 'masterdef' y la tarea es remota, indica que nodo master esta realizando la tarea.
sname	El nombre del esquema del objeto replicado, si aplica.
Request	Nombre del procedimiento DBMS_REPCAT que fue ejecutado.
Oname	El nombre del objeto replicado, si aplica.
type	Tipo del objeto replicado.
status	El estado de la petición: ready, do_callback, await_callback, o error.
Message	Cualquier mensaje de error que fue retornado.
Errnum	El numero de error Oracle retornado.
Gname	Nombre del objeto del grupo replicado.

Tabla B.2 Columnas de la vista RepCatLog.

Vista RepColumn_Group.

La vista RepColumn_Group contiene todos los grupos definidos para cada tabla replicada. La Tabla B.3 enseguida lista las columnas de la vista

Columna	Descripción
Sname	El nombre del esquema que contiene la tabla replicada.
Oname	Nombre de la tabla replicada.
group_name	Nombre del grupo de columnas.
group_comment	Cualquier comentario de usuario.

Tabla B.3 Columnas de la vista Repcolumn_group.

Vista RepConflict.

La vista RepConflict despliega el nombre de la tabla para la que se definió un método de solución de conflictos y el método de solución usado. La Tabla B.4 describe el detalle de la vista.

Columna	Descripción
Sname	Nombre del esquema que contiene la tabla replicada.

Oname	Nombre de la tabla a la que se le definió el método de solución de conflictos.
conflict_type	El tipo de conflicto que el método de solución resuelve: delete, uniqueness, o update.
reference_name	El objeto sobre el que la rutina se aplica. Para conflictos delete, es el nombre de una tabla. Para conflictos de unicidad, nombre de la regla de integridad. Para conflictos update, nombre de un grupo de columnas.

Tabla B.4 Columnas de la vista RepConflict.

Vista RepDDL.

Es usada para desplegar cualquier DDL, aplicado a objetos replicados. La vista RepDDL es descrita en la Tabla B.5 a continuación.

Columna	Descripción
log_id	Indica e número de registro de log en RepCat.
source	Nombre de la base de datos donde se origino la instrucción.
role	'Y' si la base de datos es masterdef; 'N' si la base de datos es master.
master	Nombre de la base de datos que proceso la petición.
line	Ordenamiento de registros dentro de la misma petición. (numero de línea)
text	Parte de argumentos o texto del DDL.

Tabla B.5 Columnas de la vista RepDDL.

Vista RepGenerated.

La vista RepGenerated contiene información de objetos de sistema generados. El detalle de las columnas de la vista se detalla en la Tabla B.6 enseguida.

Columna	Descripción
sname	Dueño del objeto.
oname	Nombre del objeto.
type	Tipo de objeto.
base_sname	Dueño del objeto base.
base_oname	Nombre del objeto base.
base_type	Tipo del objeto base.
package_prefix	Prefijo del nombre del paquete que contiene al objeto.
procedure_prefix	Prefijo para los procedimientos en el paquete que contiene al objeto.
distributed	'Y' si es generado de manera distribuida; 'N' si los objetos generados fueron clonados. Debe ser siempre 'Y' para ambientes Rep3.
reason	Razón por la cual el objeto fue generado.

Tabla B.6 Columnas de la vista RepGenerated.

Vista RepGrouped_Column.

Lista todas las columnas que forman los grupos de columnas para cada tabla. Esta detallada en la siguiente tabla (Tabla B.7)

Columna	Descripción
sname	Nombre del esquema que contiene la tabla replicada.

Oname	Nombre de la tabla.
group_name	Nombre del grupo de columnas.
Column_name	Nombre de la columna en el grupo.

Tabla B.7 Columnas de la vista RepGrouped_column.

Vista RepKey_Column.

Tiene información relacionada a todas las columnas que son llave primaria. El detalle de las columnas en la **Tabla B.8**

Columna	Descripción
sname	Dueño de la tabla replicada.
oname	Nombre de la tabla.
col	Columna de llave primaria en la tabla.

Tabla B.8 Columnas de la vista Repkey_column.

Vista RepSite.

La vista contiene los miembros de cada grupo replicado. El detalle en la **Tabla B.9** abajo

Columna	Descripción
Gname	El nombre del grupo de objetos replicados
Dblink	El database link al master site para el grupo de objetos.
Masterdef	Indica cual de los dblink es el master definition site.
snapmaster	Usado por snapshot sites para indicar que dblink usar al refrescar.
master_comment	Comentarios de usuario.

Tabla B.9 Columnas de la vista Repsite.

Vista RepObject.

La vista provee información de los objetos en cada grupo replicado. Un objeto puede pertenecer solamente a un grupo de objetos. Un grupo de objetos puede abarcar múltiples esquemas. La **Tabla B.10** detalla la vista.

Columna	Descripción
Sname	Nombre del esquema que contiene el objeto replicado.
Oname	Nombre del objeto replicado.
Type	Tipo de objeto: table, view, package, package body, procedure, function, index, synonym, trigger, o snapshot.
Status	CREATE Indica que Oracle esta aplicado un DDL de usuario o generado por Oracle mismo en la base de datos local. Si la replica local existe, Oracle compara COMPARE la replica con el master definition para asegurar consistencia. Cuando la comparación o creación son exitosas, Oracle cambia la columna a VALID; de lo contrario, lo cambia a ERROR. Si el objeto es borrado, Oracle actualiza su valor a DROPPED antes de borrar el registro de la vista RepObject.
Id	Identificador del objeto local, si existe.

object_comment	Cualquier comentario de usuario.
Gname	El nombre del grupo de objetos replicado al cual el objeto pertenece.

Tabla B.10 Columnas de la vista RepObject.

Vista RepParameter_Column.

Además a la información en la vista RepResolution, la vista RepParameter_Column contiene información provista por el usuario que debería ser usada para la solución de conflictos.

Estas son los valores de las columnas que son pasadas como argumentos LIST_OF_COLUMN_NAMES al procedimiento ADD*_RESOLUTION del paquete DBMS_REPCAT. La vista *reparameter_column* se detalla en la Tabla B.11 abajo

Columna	Descripción
Sname	El nombre del esquema que contiene la tabla replicada.
oname	Nombre de la tabla replicada.
conflict_type	Tipo de conflicto que la rutina resuelve: delete, uniqueness, o update.
reference_name	El objeto al cual la rutina se aplica. Para conflictos delete, es el nombre de la tabla. Para conflictos uniqueness, nombre del constraint. Para conflictos update, es el nombre de grupo de columnas.
sequence_no	El orden en que los métodos de solución son aplicados, siendo 1 el que se aplica primero.
method_name	Nombre del método de solución provisto por Oracle. Para métodos de usuario, el valor es 'user function'.
function_name	Para métodos 'user function', el nombre de la rutina de solución.
priority_group	Para métodos 'priority group', el nombre del grupo de prioridad.
parameter_table_name	Tabla de PL/SQL con el mismo nombre del objeto. Contiene las columnas pasadas a la función de solución de conflictos.
parameter_column_name	Nombre de la columna usada en como parámetro IN para la rutina de solución.
parameter_sequence_no	El orden de las columnas en la entrada de la rutina.

Tabla B.11 Columnas de la vista RepParameter_column.

Vista RepPriority.

Despliega el valor y nivel de prioridad de cada miembro del grupo de prioridades. Los nombres del grupo de prioridad deben ser únicos en el grupo de objetos replicados. Los niveles de prioridad deben ser únicos en el grupo de prioridad. La Tabla B.12 contiene el detalle de las columnas de la vista.

Columna	Descripción
sname	Nombre del esquema replicado. Obsoleto para versión 7.3 y posteriores.
gname	Nombre del grupo de objetos replicados.
priority_group	Nombre del grupo de prioridad o grupo de nodos de prioridad.
priority	Nivel de prioridad del miembro. El valor más alto es la mas alta prioridad.

data_type	Tipo de datos de los valores en el grupo de prioridad.
fixed_data_length	Longitud máxima de valores tipo CHAR.
char_value	Valor del miembro del grupo de prioridad, si data_type = char.
varchar2_value	Valor del miembro del grupo de prioridad, si data_type = varchar2.
number_value	Valor del miembro del grupo de prioridad, si data_type = number.
date_value	Valor del miembro del grupo de prioridad, si data_type = date.
raw_value	Valor del miembro del grupo de prioridad, si data_type = raw.

Tabla B.12 Columnas de la vista Reppriority.

Vista RepPriority_Group.

La vista RepPriority_Group contiene listas de prioridades y grupos de nodos de prioridades que se han definido para un grupo de objetos replicados. La vista es detallada en la Tabla B.14

Columna	Descripción
sname	Nombre de esquema replicado. Obsoleto en 7.3 y posteriores. No mostrado en vistas USER.
gname	Nombre de grupo de objetos replicado. No mostrado en vistas USER.
priority_group	Nombre del grupo de prioridad o grupo de nodos de prioridad.
data_type	Tipo de datos en el grupo de prioridad.
fixed_data_length	Máxima longitud para tipo de dato CHAR.
priority_comment	Cualquier comentario de usuario

Tabla B.13 Columnas de la vista Reppriority_group.

Vista RepProp.

Indica la técnica usada para propagar operaciones sobre un objeto al mismo objeto en otro master site. Las operaciones pueden ser resultado de una llamada a un procedimiento almacenado o paquete, o haber sido ejecutados directamente en la tabla. Se detalla en la Tabla B.14

Columna	Descripción
sname	Nombre de esquema que contiene objeto replicado.
oname	Nombre de objeto.
type	Tipo de objeto siendo replicado.
dblink	Calificador completo del nombre de la base de datos del master site hacia donde los cambios están siendo propagados.
how	Como se esta haciendo la propagación. Los valores permitidos son 'none' para el master site; 'local', y 'synchronous' o 'asynchronous' para el resto.
propagate_comment	Cualquier comentario de usuario.

Tabla B.14 Columnas de la vista Repprop.

Vista RepResolution.

Muestra las rutinas usadas para resolver conflictos de update, unique o delete para cada tabla replicada usando replicación de nivel row-level en un esquema dado.

La descripción de las columnas de la vista se incluyen en la Tabla B.15

Columna	Descripción
sname	Nombre de esquema replicado.
oname	Nombre de tabla replicada.
conflict_type	Tipo de conflicto a resolver: delete, uniqueness, o update.
reference_name	El objeto sobre el que la rutina es aplicada. Para conflictos de delete, es el nombre de la tabla. Para uniqueness, nombre del constraint. Para conflictos por update, nombre del grupo de columnas.
sequence_no	Orden en el que los métodos de resolución son aplicados, siendo 1 el que se aplica primero.
method_name	Nombre del método de resolución provisto por Oracle. Para métodos creados por el usuario el valor es 'user function'.
function_name	Para métodos 'user function', es el nombre de la rutina.
priority_group	Para métodos de tipo 'priority group', el nombre del grupo de prioridad.
resolution_comment	Comentarios de usuario.

Tabla B.15 Columnas de la vista Represol.

Vista RepResol_Stats_control.

La vista RepResol_Stats_Control contiene información de estadísticas recolectadas de todas las soluciones de conflictos para todas las tablas replicadas en la base de datos. Detalle de la vista en la Tabla B.16

Columna	Descripción
Sname	Dueño de la tabla.
Oname	Nombre de la tabla.
Created	Hora de inicio en las que las estadísticas fueron primeramente colectadas.
Status	Estado de las estadísticas: ACTIVE, CANCELLED
status_update_date	Hora en la que fueron actualizadas por última vez.
purged_date	Hora en la que fueron eliminadas.
last_purged_start_date	La última fecha de inicio en la que las estadísticas fueron eliminadas.
statistics_purged_end_date	La última fecha de terminación (rango) en el que las estadísticas fueron eliminadas.

Tabla B.16 Columnas de la vista Represol_stats_control.

Vista RepResolution_Method.

Contiene las rutinas de resolución disponibles en la base de datos. En un inicio, solamente las rutinas estándar provistas con la opción de replicación simétrica están disponibles. Conforme se crean mas funciones y se agregan como métodos de solución, estos son desplegados en esta vista. La Tabla B.17 contiene el detalle de las columnas.

Columna	Descripción
conflict_type	El tipo de conflicto que esta rutina soluciona: update, uniqueness, o delete.
method_name	Nombre del método, ya sea provisto por Oracle o definido por el usuario.

Tabla B.17 Columnas de la vista Represolution_method.

Vista RepResolution_Statistics.

La vista RepResolution_Statistics contiene información de aquellos conflictos de update, uniqueness, y delete que fueron manejados exitosamente. Las estadísticas para la tabla son coleccionadas únicamente si se invocó el paquete DBMS_REPCAT.REGISTER_STATISTICS. El detalle se muestra en la **Tabla B.18**

Columna	Descripción
Sname	Nombre del esquema replicado.
Oname	Nombre de tabla replicada.
conflict_type	Tipo de conflicto que fue exitosamente resuelto: delete, uniqueness, o update.
reference_name	El objeto al cual la rutina de solución fue aplicada. Para conflictos delete, es el nombre de la tabla. Para uniqueness, nombre de constraint. Para update, es el nombre del grupo de columnas.
method_name	Nombre del método de solución de Oracle. Para métodos de usuario el valor de la columna es 'user function'.
function_name	Para métodos 'user function', el nombre de la rutina de usuario empleada.
priority_group	Para métodos de tipo 'priority group', nombre del grupo de prioridad.
primary_key_value	Concatenación de la llave primaria del registro.
resolved_date	Fecha en la que el conflicto fue resuelto.

Tabla B.18 Columnas de la vista Represolution_statistics.

Vista RepSchema.

La vista es usada por compatibilidad con versiones anteriores de la opción de replicación simétrica (antes de 7.3) Después de esta versión se usa la vista RepSite. La **Tabla B.19** describe las columnas de la vista

Columna	Descripción
Sname	Nombre del esquema replicado. Obsoleto en 7.3 y posteriores.
gname	El nombre del grupo de replicación.
dblink	El database link hacia donde los cambios de replicación deben ser encolados en cada master site.
masterdef	Indica cual de los dblinks es el master definition site.
snapmaster	Usado por los snapshot sites para indicar que dblinks usar para refrescar.
master_comment	Cualquier comentario de usuario.

Tabla B.19 Columnas de la vista RepSchema.

Vistas para el manejo de transacciones diferidas.

Oracle proporciona varias vistas para administrar transacciones diferidas. Estas vistas contienen información sobre cada transacción diferida como lo es destino de la transacción, número de llamadas diferidas que la transacción realiza, y cualquier error encontrado durante la ejecución de la transacción.

Estas tablas no deben ser modificadas directamente, hay que usar los paquetes provistos DBMS_DEFER y DBMS_DEFER_SYS.

Vista DefCall.

Registra todas las llamadas remotas diferidas a procedimientos. Se detalla enseguida en la Tabla B.20

Columna	Descripción
Callno	ID único de la llamada en deferred_tran_db.
deferred_tran_db	La base de datos que origino la llamada.
deferred_tran_id	El ID único de la transacción asociada.
Schemaname	Nombre del esquema.
Packagename	Nombre del paquete.
Procname	Nombre del procedimiento de la llamada diferida.
Argcount	Numero de argumentos al procedimiento.

Tabla B.20 Columnas de la vista defcall.

Vista DefCallDest.

Lista los destinos para cada llamada remota diferida a procedimientos. La Tabla B.21 describe las columnas de la vista

Columna	Descripción
Callno	ID único de la llamada en deferred_tran_db.
deferred_tran_id	Corresponde al deferred_tran_id en la vista DefTran. Cada transacción diferida esta formada de una o más llamadas.
deferred_tran_db	Base de datos donde se inició la transacción diferida. El 'callno' y deferred_tran_db identifican de manera única a cada llamada.
Dblink	Identificador único de la base de datos destino.

Tabla B.21 Columnas de la vista defcallDest.

Vista DefDefaultDest.

Sí no se está usando la opción de replicación simétrica y no se proporciona un destino para la transacción diferida o para las llamadas dentro de la misma transacción, Oracle utiliza la vista DefDefaultDest para determinar la base de datos remota a la que se desean hacer las llamadas. Enseguida, la Tabla B.22, describe el contenido de la Vista

Columna	Descripción
dblink	Nombre único de la base de datos a donde es replicada la transacción.

Tabla B.22 Columnas de la vista defDefaultDest.

Vista DefError.

Proporciona el ID de cada transacción que no pudo ser aplicada. Se puede usar este ID para localizar las llamadas encoladas asociadas a esta transacción.

Estas llamadas son almacenadas en la vista DefCall. Se pueden usar los procedimientos en el paquete DBMS_DEFER_QUERY para determinar los argumentos en los procedimientos listados en la vista DefCall. Las columnas de la vista se detallan en la Tabla B.23

Columna	Descripción
deferred_tran_db	Calificador único de la base de datos que genero o copio las llamadas a procedimientos remotos.
deferred_tran_id	ID de la transacción que generó o copió las llamadas a procedimientos remotos que esta causando el error.
callno	ID único de la llamada en deferred_tran_db.
destination	El Database link usado para conectarse al destino.
error_time	Tiempo en el que ocurrió el error.
error_number	Numero de error Oracle.
error_msg	Texto del mensaje de error.

Tabla B.23 Columnas de la vista defError.

La vista DefSchedule.

Despliega información de cuando un job esta programado para su ejecución. Columnas en la Tabla B.24 enseguida.

Columna	Descripción
Dblink	Nombre de la ruta al master site para el que se han programado ejecuciones periódicas llamadas diferidas a procedimientos remotos.
Job	Número interno asignado al job cuando es creado usando el procedimiento DBMS_DEFER_SYS.SCHEDULE_EXECUTION. Consultar la columna WHAT de la vista USER_JOBS que se ejecuta cuando el job esta corriendo.
Interval	Función usada para calcular el siguiente tiempo de ejecución de aplicación de cambios.
next_date	Siguiente fecha en la que el job será ejecutado.
last_date	La última vez que DBMS_DEFER_SYS.EXECUTE envió (o intento enviar) llamadas a procedimientos remotos a esta base de datos.
Disabled	¿Está la propagación al destino habilitada?
last_txn_count	Numero de transacciones enviadas durante el último intento.
last_error	Numero de error de Oracle del último envío.
last_msg	Mensaje de error del último envío.

Tabla B.24 Columnas de la vista defSchedule.

Vista DefTran.

La vista DefTran registra todas las transacciones diferidas. Detalle en Tabla B.25

Columna	Descripción
deferred_tran_id	ID de transacción que origina o copia las llamadas diferidas a procedimientos remotos.
deferred_tran_db	Calificador único del nombre de la base de datos que origina o copia las llamadas diferidas.
origin_tran_id	ID de transacción que originó las llamadas diferidas a procedimientos

	remotos.
origin_tran_db	Nombre de la base de datos donde se originan las llamadas diferidas.
origin_user	El usuario del usuario que originó las llamadas.
delivery_order	Identificador que determina el orden de las transacciones en la cola. Es derivado del 1 orden de las transacciones en la cola. Es derivado del <i>system commit number</i> de la transacción que origino las llamadas.
destination_list	'R' o 'D'. 'R' Indica que los destinos son determinados por la vista RepSchema. 'D' que los destinos fueron definidos por la vista DefDefaultDest o el argumento NODE_LIST a los procedimientos TRANSACTION, CALL, o COPY.
start_time	Tiempo de inicio de la transacción.
commit_comment	Cualquier comentario de usuario.

Tabla B.25 Columnas de la vista defTran.

Vista DefTranDest.

La vista contiene los destinos de las transacciones diferidas. Detalle de columnas abajo en **Tabla B.26**

Columna	Descripción
deferred_tran_id	La transacción a replicar al database link indicado.
deferred_tran_db	La base de datos donde se origina la transacción diferida. El deferred_tran_id y el deferred_tran_db identifican de manera única a la transacción.
dblink	El nombre de la base de datos destino.

Tabla B.26 Columnas de la vista defTranDest.

Consulta de la cola de Jobs.

Las vistas del diccionario de datos que despliegan información de los jobs encolados son:

- DBA_JOBS
- USER_JOBS
- DBA_JOBS_RUNNING

Vistas DBA_JOBS y USER_JOBS.

La vista DBA_JOBS proporciona información de todos los jobs encolados en la base de datos. La vista USER_JOBS limita la información a aquellos jobs que pertenecen al usuario; esto es, jobs para los cuales uno es el PRIV_USER. La **Tabla B.27** detalla la vista *_JOBS

Columna	Descripción
JOB	Identificador del job.
LOG_USER	Usuario conectado cuando el job fue sometido. Por ejemplo, si el usuario SCOTT llama un paquete que es ejecutado en el esquema de SYS, y el paquete inicia un job, el LOG_USER sería SCOTT.
PRIV_USER	Usuario cuyo privilegio por definición es aplicado para el job. Por ejemplo, si SCOTT llama a un paquete que es ejecutado

**TESIS CON
FALLA DE ORIGEN**

	en el esquema de SYS, y el paquete inicia un job, el PRIV_USER sería SYS.
SCHEMA_USER	Esquema por definición que hace parseo del job. Sí por ejemplo, el SCHEMA_USER es SCOTT y se somete el procedimiento HIRE_EMP como un job, Oracle buscará SCOTT.HIRE_EMP
LAST_DATE	Última vez que el job se ejecuto de manera exitosa, redondeado al día mas cercano.
LAST_SEC	Última vez que el job se ejecuto de manera exitosa, redondeado al segundo día mas cercano..
THIS_DATE	Fecha de inicio de ejecución actual, redondeado al día mas cercano. Usualmente el valor no existe si el job no esta en ejecución.
THIS_SEC	Fecha de inicio de ejecución actual, redondeada al segundo mas cercano. Sin valor so el job no se encuentra en ejecución.
NEXT_DATE	Siguiente fecha de ejecución programadas. En días.
NEXT_SEC	Siguiente fecha de ejecución programadas. En segundos.
TOTAL_TIME	Tiempo total transcurrido empleado por el sistema en el job, en segundos.
BROKEN	"N" indica que no esta roto. "Y" considera al job roto y no será ejecutado.
INTERVAL	Función que calcula el siguiente tiempo de ejecución para el job.
FAILURES	Numero de ocasiones que el job fue sometido y fallo desde la última vez que fue completado exitosamente. Después de 16 fallas, el job es marcado broken.
GAT	Definición del job.
CURRENT_SESSION_LABEL	Etiqueta de Trusted Oracle7 de la sesión, como está siendo vista por el job.
CLEARANCE_HI	Más alto nivel de liquidación disponible para el job. Aplica sólo para Trusted Oracle7.
CLEARANCE_LO	El más bajo nivel de liquidación disponible para el job. Solamente para Trusted Oracle7.
NLS_ENV	Parámetros de ALTER SESSION que describen el ambiente para el job.
MISC_ENV	Otros parámetros de sesión que aplican para el job.

Tabla B.27 Columnas de la vista dba_jobs y user_jobs.

Vista DBA_JOBS_RUNNING.

La vista DBA_JOBS_RUNNING tiene la información de cualquier job que se esté ejecutando. El contenido de las columnas se detalla en la **Tabla B.28**

Columna	Descripción
SID	ID de la sesión que ejecuta el job.
JOB	ID del job que está siendo ejecutado.
FAILURES	Numero de veces que el job fue sometido y fallo desde la última vez que termino exitosamente.

LAST_DATE	Última ejecución exitosa, al día más cercano.
LAST_SEC	Última ejecución exitosa, al segundo más cercano.
THIS_DATE	Fecha de inicio de ejecución actual, medida al día más cercano.
THIS_SEC	Fecha de inicio de ejecución actual, medida al segundo más cercano.

Tabla B.28 Columnas de la vista dba_jobs_running.

Vistas de Snapshot y Snapshot Refresh Group.

Oracle proporciona las siguientes vistas con información de los snapshots y los snapshot refresh groups.

- DBA_SNAPSHOTS
- USER_REFRESH
- USER_REFRESH_CHILDREN

Vista DBA_SNAPSHOTS.

Esta vista del diccionario de datos contiene información de todos los snapshots en la base de datos.

En la **Tabla B.29** se describe el contenido de las columnas de la vista DBA_SNAPSHOTS.

Columna	Descripción
OWNER	Dueño del snapshot.
NAME	Nombre de la vista usada por los usuarios y aplicaciones para consultar y actualizar el snapshot.
TABLE_NAME	Tabla en la cual el snapshot es almacenado (es una columna extra en el master rowid)
MASTER_VIEW	Vista del master table, que pertenece al dueño del snapshot, usado para refrescar la información.
MASTER_OWNER	Dueño del master table.
MASTER	Master table de donde el snapshot copia los datos.
MASTER_LINK	Nombre del database link al master site
CAN_USE_LOG	'YES' si el snapshot puede usar un snapshot log, 'NO' si el snapshot es muy complejo para usar un log.
UPDATABLE	'YES' indica que el snapshot es actualizable. 'NO' que es solamente de lectura.
LAST_REFRESH	Fecha y hora de la última actualización de master site.
ERROR	Último error encontrado al intentar un refresco automático, o número de intentos fallidos desde la última ejecución exitosa.
TYPE	Tipo de actualización automática: COMPLETE, FAST, FORCE
NEXT	Función usada para calcular la siguiente fecha de refresco
START_UTI	Función usada para calcular las siguientes fechas.
REFRESH_GROUP	Identificador de grupos para actualizaciones consistentes.
UPDATE_TRIG	Nombre del trigger que actualiza el UPDATE_LOG para un snapshot modificable.
UPDATE_LOG	Nombre de la tabla que almacena los logs de cambios de un snapshot

TESIS CON
 FALLA DE ORIGEN

	modificable.
QUERY	Sentencia usada para crear el snapshot.

Tabla B.29 Columnas de la vista dba_snapshots.

Vista USER_REFRESH.

Tiene cada refresh group en la base de datos, incluye información de los intervalos de refresco para cada grupo.

Columna	Descripción
ROWNER	Dueño del grupo de actualización.
RNAME	Nombre del grupo.
REFGROUP	Numero de ID del grupo.
IMPLICIT_DESTROY	Bandera implícita de borrado. Si es 'Y', Oracle borra el grupo de actualización después de haber sustraído el último miembro del grupo.
JOB	ID del job usado para hacer refresco automático del snapshot group. SE puede usar esta información para consultar la vista USER_JOBS para obtener mas información del job.
NEXT_DATE	Fecha en la que los miembros del grupo serán refrescados.
INTERVAL	Función usada para calcular el intervalo entre refrescos.
BROKEN	Bandera que indica problemas con el refresh group. Si la bandera está en 'Y', Oracle no refrescará el grupo, aún si esta programado para ser refrescado.

Tabla B.30 Columnas de la vista user_refresh.

Vista USER_REFRESH_CHILDREN.

La vista USER_REFRESH_CHILDREN tiene los miembros de cada refresh group que pertenecen al usuario. Incluye información de los intervalos de refresco para cada grupo. Detalle en Tabla B.31 enseguida.

Columna	Descripción
OWNER	Dueño del miembro del refresh group.
NAME	Nombre del miembro del refresh group.
TYPE	Tipo del miembro del refresh group, por ejemplo, SNAPSHOT.
ROWNER	Dueño del refresh group.
RNAME	Nombre del refresh group.
REFGROUP	ID del refresh group.
IMPLICIT_DESTROY	Bandera implícita de borrado. Si el valor es Y, Oracle borra el refresh group una vez que se sustrajo el último miembro del grupo.
JOB	ID del job usado para ejecutar el refresco automático del snapshot refresh group. Se puede usar la información para consultar detalles del job en la vista USER_JOBS.
NEXT_DATE	Fecha en la que los miembros del grupo serán refrescados.
INTERVAL	Función usada para calcular el intervalo entre actualizaciones.
BROKEN	Bandera que indica si hay problemas con el refresh group. Si el valor es 'Y' Oracle no refrescará el grupo aún si está programado para serlo.

Tabla B.31 Columnas de la vista user_refresh_children.

Glosario.

3.

3GL. (3rd Generation Language) Los lenguajes de tercera generación son estructurados y están basados en procedimientos. Ejemplos de este tipo de son el lenguaje C y lenguaje Cobol.

A.

ACID, propiedades. Propiedades de las transacciones de base de datos que dicen que deben ser Atómicas, Consistentes, Independientes y Durables

Análisis descendente. Metodología para el análisis de requerimientos de sistemas que parte de la revisión de lo general a lo particular. Comienza por revisar el todo para pasar a los componentes del sistema.

API (Application Program Interface) Programa de interfaz de usuario. Es la pieza de software o aplicación que tiene el usuario para interactuar con un servidor de datos.

Árbol de sesiones. (Session tree) Modelo jerárquico de una transacción distribuida usado por el Oracle Server para representar las relaciones entre los nodos participantes

Atributos. En el modelo Entidad-Relacion son las propiedades que califican a las diferentes entidades representadas en el diagrama

C.

Candados. (Locks) Mecanismo usado por el Oracle Server para controlar el acceso concurrente a la base de datos, esto garantiza que se mantenga la consistencia e integridad de los datos.

Concurrente, acceso. Uso de la misma pieza de datos en un mismo momento en sistemas RDBMS que son multusuario

Cerradura, propiedad de. En el álgebra relacional, es la propiedad que tienen las operaciones básicas de generar nuevas tablas a partir de las tablas de entrada.

Commit, operación de. Operación que indica que la transacción realizada fue exitosa y los datos que fueron cambiados dejaron la base de datos en un estado consistente. Los cambios se hacen permanentes

D.

Database Link. (Ver liga de base de datos)

TESIS CON
FALLA DE ORIGEN

Glosario.

DBA. (Database Administrator) Persona encargada de la administración de la base de datos, que incluye usuarios y como se organizan los objetos que la forman. Maneja y define la seguridad y permisos a usuarios, el respaldo de los datos, y el buen desempeño de la misma.

DBMS. (Database Management System) Sistema manejador de base de datos. Programa encargado de manejar de manera homogénea y no redundante los datos comunes a una organización. Mantiene las propiedades definidas para las bases de datos definidas en la arquitectura ANSI / SPARC.

DDL. (Data Definiton Language) La categoría de sentencias SQL que crea o elimina objetos de la base de datos como lo son tablas o vistas de base de datos.

Desnormalización de datos. Proceso que viola de manera intencional las reglas de normalización de datos. Usado para efectos de desempeño.

Diccionario de datos. También llamado catalogo de datos es un conjunto de objetos contenidos en la misma base de datos (en el caso de una base de datos relacional en forma de tablas y vistas) que son directamente actualizados por el DBMS y contiene información administrativa de todos los objetos de la base de datos.

DML. (Data Manipulation Language) Categoría de sentencias SQL usadas para recuperar o modificar datos dentro de la base de datos. Las sentencias comunes son las de SELECT, UPDATE, INSERT y DELETE.

Das fases, protocolo de compromiso. Ver two-phase commit.

E.

Entidad. En el modelo Entidad-Relación, es la definición de cualquier cosa que es importante para la organización. Que genera o recibe información.

Entidad-Relación, diagrama. Representación grafica los requerimientos de información de una organización. Es generado en la etapa de modelado de la base de datos y se basa en los modelos de Peter Chen y E.F. Codd.

F.

Fragmentación. La habilidad de un RDBMS en un ambiente distribuido para manejar fragmentos de tablas localizadas en diferentes nodos de manera transparente para el usuario y la aplicación. Existe la fragmentación horizontal (cuando los fragmentos son divididos basándose en los valores en las columnas de las tablas); y la fragmentación vertical (cuando los fragmentos se forman dividiendo las columnas de la tabla)

Función de base de datos. Procedimiento almacenado en la base de datos que retorna un valor a la instrucción que la invoca.

G.

Grupos de columnas. Implementación de fragmentación vertical de Oracle al ligar varias columnas a una sola columna lógica. Un grupo puede estar formado por una, varias o todas las columnas de una tabla; sin embargo, cada columna sólo puede pertenecer solamente a un grupo.

Grupo de columnas Shadow. Grupo formado automáticamente con las columnas que no pertenecen a ningún grupo de columnas.

H.

Heterogéneo, ambiente distribuido. Definición que tiene Oracle para un ambiente distribuido donde al menos uno de los nodos corre una base de datos no-Oracle que puede ser inclusive de red o jerárquica.

Híbrida, fragmentación. Es la combinación, en un ambiente distribuido, del particionamiento o fragmentación horizontal y vertical.

Homogéneo, ambiente distribuido. Definición de Oracle para un ambiente distribuido donde todos los nodos tienen bases de datos relacionales Oracle.

Horizontal, fragmentación. Enfoque para la división no redundante de datos entre nodos de un ambiente distribuido basándose en el valor que una determinada columna tiene.

I.

Índice de base de datos. Estructuras de la base de datos Oracle creadas para disminuir el tiempo de acceso a datos almacenados en tablas. Un índice contiene la localización precisa en los bloques de datos de información específica. Una vez creadas son administradas internamente por el RDBMS.

Instancia Oracle. Combinación de un área reservada de memoria (SGA) y procesos Oracle para la manipulación de una base de datos relacional.

J.

Job de base de datos. Tarea programada en una base de datos Oracle para ser ejecutada en un determinado momento o en un periodo definido. Ejecuta cualquier programa de PL/SQ.

L.

Liga de base de datos. Es un apuntador a una base de datos remota en un ambiente distribuido que permite utilizar al usuario los datos de ambos nodos como si se tratase de una sola base de datos lógica.

Ll.

Llave primaria. Valor único usado en bases de datos relacionales para calificar a cada registro existente en una tabla. La llave primaria es usada además para que los valores en otras tablas sean relacionados a ella en la forma de una llave foránea.

Llave foránea. Es el valor asignado en la columna de una tabla en bases de datos relacionales. Usado para asociar sus valores con los existentes en la llave primaria de otra tabla.

N.

Nodo. Servidor de base de datos que forma parte de una base de datos distribuida, es una base de datos por sí misma pero se comunica con otros nodos por diferentes medios de comunicación.

Nombre Global. Es un identificador único de cada base de datos Oracle en un ambiente distribuido. Es usado para la creación de las ligas de base de datos entre los diferentes nodos.

O.

Open Gateway. Software Oracle usado en ambientes distribuidos para conectar clientes y bases de datos Oracle con bases de datos en otros nodos que no son relacionales o corren otro tipo de RDBMS. Su función es emular una base de datos Oracle en el nodo remoto y es transparente para usuarios y aplicaciones que la perciben como cualquier RDBMS Oracle.

Orden Total (OT) Listado de orden de todas las transacciones distribuidas ejecutadas en los diferentes nodos. Es usado para construir la estrategia concurrente.

P.

Parámetro de base de datos. Parámetros de iniciación de bases de datos Oracle que determinan el comportamiento de ciertas características del RDBMS.

Parseo. (Parsing) Es la revisión de la sintaxis de una sentencia SQL creada por el usuario.

Paquete de base de datos. Procedimiento de base de datos Oracle usado para encapsular otros procedimientos o funciones almacenados.

PL/SQL. (Procedural Language/SQL) Lenguaje del Oracle Server que combina un lenguaje por procedimientos con los comandos SQL de la base de datos esto permite controlar el flujo de los comandos y usar variables, entre otras cosas.

Procedimiento almacenado. Bloque de código PL/SQL almacenado en la base de datos Oracle y esta programado para ejecutar tareas específicas.

R.

RDBMS. (Relational Database Management System) Sistema Manejador de bases de datos relacionales. Programa para la manipulación de bases usando las operaciones definidas en el álgebra relacional sobre un elemento base llamado tabla.

RECO. Proceso automático en Oracle encargado de resolver las transacciones distribuidas que se encuentran en un estado "en duda" por alguna falla de los nodos participantes.

Recuperación. Mecanismo del RDBMS que permite traer la base de datos a un estado que se sabe "correcto" después de una falla del sistema.

Relaciones. En el modelo Entidad-Relación califica la manera en la que las diferentes entidades de la organización interactúan unas con otras. Una relación es única entre dos entidades.

Remote Procedure Call (RPC) *Llamada a procedimiento remoto*, usadas en la tecnología Cliente-Servidor donde las peticiones del cliente a un servidor intermedio se hacen en forma de procedimientos con parámetros específicos.

Replicación de datos. Característica de un ambiente distribuido donde se mantienen copias de los mismos datos en los diferentes nodos. Datos requeridos por cada nodo para su operación. Esto requiere de mayores recursos pero es necesario en ambientes donde el acceso a datos desde diferentes locaciones es frecuente.

Replicación funcional. Método para la replicación de datos usando procedimientos almacenados. Esta encapsula varios cambios hechos a los datos en forma de un bloque de código que más tarde es ejecutado en el nodo replicado.

Rol de base de datos. (role) Grupo de privilegios que pueden ser dado a usuarios u otros roles. Son usados para ayudar a administrar los permisos de acceso a la base de datos. Se otorgan permisos sobre los objetos y la operación permitida en cada uno.

Rollback, operación de. Operación que deshace los cambios hechos por sentencias DML pertenecientes a la misma transacción. Regresa la base de datos al último estado consistente registrado.

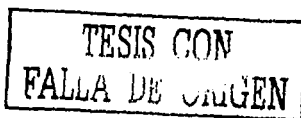
Rollback, segmentos de. Unidades lógicas de una base de datos Oracle usados para almacenar la información necesaria para completar la *operación de rollback* de las transacciones no confirmadas por el usuario. Además de ser usados para permitir lecturas consistentes, y al momento de recuperación de la base de datos después de una falla.

ROWID, columna. Columna interna de toda tabla en una base de datos Oracle que esta formada de una secuencia de caracteres que identifican de manera única un registro y representa la dirección física en el archivo de datos del registro mismo. Pueden usarse en sentencias de SELECT usando la palabra reservada ROWID.

S.

Sinónimo. (Synonym) Objeto de diccionario de datos, se define como un nombre lógico en una base de datos Oracle que apunta a otro objeto en la misma base. Es usado principalmente para simplificar el acceso que los usuarios tienen a ciertos objetos.

Snapshot. Mecanismo básico para la implementación de la replicación de datos usado por Oracle. Un snapshot es una copia de una tabla maestra localizada en un nodo remoto. Un snapshot puede ser consultado como cualquier tabla y sus datos son periódicamente actualizados para reflejar los cambios hechos a la tabla maestra.



SQL. (Structured Query Language) Lenguaje estándar para el manejo de bases de datos relacionales considerado de cuarta generación. Conjunto de instrucciones para la manipulación de objetos y datos.

SQL*Net. Capa de software Oracle usada para la establecer la comunicación entre el cliente y la base de datos Oracle, o en el caso de ser un ambiente distribuido la comunicación es de servidor a servidor para determinadas transacciones. Responsable de envío y recepción de peticiones.

SYS, SYSTEM, usuarios. Usuarios administradores de una base de datos Oracle. Son creados junto con la base de datos. SYS es el esquema propietario de las tablas del diccionario de datos. SYSTEM es un usuario con permisos de sólo lectura sobre estas tablas, es usado para hacer tareas de mantenimiento en la base de datos.

T.

Tabla. En una base de datos relacional, es la manera en la que son percibidos los datos. Esta formado por atributos que forman las columnas de la tabla y califican a la entidad y contiene registros que son los datos que cumplen con estos atributos. Una tabla mas que ser un objeto físico es una estructura lógica.

Tipos de datos. Los tipos de datos que puede tener una columna que existen en Oracle y se basan en el ANSI SQL son: Char (caracteres), Number (Numéricos enteros), Float (reales de doble precisión), VARCHAR (cadenas de caracteres)

Transacción. Unidad lógica de trabajo que esta formada de una o más sentencias DML ejecutadas por un solo usuario. El estándar SQL ANSI / ISO indica que una transacción con la primer sentencia DML ejecutada y termina cuando el mismo usuario confirma o deshace los cambios hechos.

Transacción distribuida. Es una transacción que modifica datos en mas de una base de datos. Cambia información residente en dos o más RDBMS llamados nodos.

Trigger de base de datos. Definido como un bloque de código asociado a una tabla en una base de datos relacional. Dicho código es ejecutado automáticamente por el RDBMS dependiendo de una acción específica sobre los registros de la tabla.

Two-phase commit. Mecanismo usado durante el proceso de COMMIT TRANSACTION de una transacción distribuida. Es usado para garantizar la integridad de datos en todos los nodos participantes en la transacción. El two-phase commit asegura que todos los cambios realizados sean completados o ninguno.

V.

Vertical, fragmentación. En un ambiente distribuido es la separación por columnas de una tabla en diferentes nodos.

Bibliografía.

- [BOCH93] Steven Bobrowski, Cynthia Chin-Lee
Oracle7 Server Concepts Manual
Copyright © 1993, 1996 Oracle Corporation
- [CHEE94] Kevin Cheek
Oracle7 Reference Manual
Copyright © Oracle Corporation 1994, 1996
- [NEVF92] Larry Neumann, Sandy Venning, Laura Ferrer
Understanding SQL*Net
Copyright © 1992, 1993, 1994, 1995, 1996 Oracle Corporation
- [DUFE95] Jason Durbin, Laura Ferrer
Oracle7 Server Distributed Systems Manual, Vol.1
Copyright © 1995, 1996 Oracle Corporation
- [PRAT95] Maria Pratt
Oracle7 Server Distributed Systems Manual, Vol.2
Copyright © 1995 Oracle Corporation
- [DATE95] C.J. Date
An Introduction to Database Systems
Sixth Edition
Adison-Wesley Publishing Company, 1995
- [BOBA93] Angelo R. Bobak
Distributed and Multi-Database Systems
Artech House, 1993
- [KOSI93] Henry F. Kort, Abraham Siberschatz
Fundamentos de Bases de Datos
Segunda Edición
McGraw Hill, 1993
- [BURE97] Marie Buretta
Data Replication
John Wiley and Sons Inc., 1997
- [EDEL94] Herb Edelstein
Unraveling Client/Server Architecture.
DBMS 7., 1994
- [BEHM93] Bever, M, Geihs, K., Heuser, L., Muhlhauser, M., Schil, A.
Distributed Systems, OSF DCE and Beyond.
Springer-Verlag, 1993