2 ejs.

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES ACATLAN

TECNOLOGIA CLIENTE / SERVIDOR
ELEMENTOS QUE LO CONFORMAN

T E S I N A

QUE PARA OBTENER EL TITULO DE

LICENCIADO EN MATEMATICAS APLICADAS Y COMPUTACION

P R E S E N T A

OSCAR IGNACIO VALLEJO MORALES

ASESOR. ING RUBEN ROMERO RUIZ

MEXICO, D. F. MAYO 1998

INDICE

INTRODUCCION

El presente trabajo está orientado a lideres de proyecto (o desarrolladores) que tengan experiencia en el desarrollo de aplicaciones pequeñas; esto es aplicaciones que pueden ser desarrolladas por una persona, aplicaciones departamentales, aplicaciones que utilizan DBMSs (Data Base Management System) tales como Access, Paradox, Dbase, etcetera.

Este trabajo surgió de la falta de conocimientos con que se enfrentan los lideres de proyecto (o desarrolladores) cuando son asignados a participar en proyectos grandes y/o complejos donde las bases de datos pueden ser de cientos de megabytes, gigabytes o terabytes, donde la información será accesada por cientos de usuarios pertenecientes a distintos departamentos dentro de una red LAN o WAN, donde diferentes usuarios requieran funcionalidades distintas, donde la información debe estár disponible las 24 horas del dia. El objetivo del trabajo es difundir conceptos e ideas que pueden ser de utilidad en éste tipo de desarrollos, sin llegar a plantear un caso práctico, que resultaría de gran complejidad, ya que se requiere de una gran cantidad de conocimientos, recursos computacionales, experiencia y trabajo en equipo ya que un desarrollo de éste tipo no puede ser desarrollado por una sola persona. El capítulo 1 muestra superficialmente lo que es la tecnología cliente/servidor sin entrar en detalles. El capítulo 2 hace referencia a la arquitectura de tres hileras para el desarrollo de aplicaciones cliente/servidor, los puntos tocados en este capítulo probablemente ya sean conocidos por los desarrolladores que generalmente ocupan el puesto de analista programador. El capítulo 3 trata sobre la red sin profundizar en el hardware y enfocandose sobre todo en el software intermedio que nos permite comunicar una aplicación con la base de datos, a ésta parte no se le da la importancia debida hasta que una aplicación requiere de la mezcla de diferentes tecnologías de hardware y software. El capítulo 4 presenta información sobre lo que brindan los RDBMSs (Relational Data Base Management System) de alto desempeño, esto es con la finalidad de saber el porque de la elección de RDBMSs como Oracle y Sybase sobre RDBMSs como Access y Paradox sin hacer un estudio profundo sobre lo que pueden brindar, ni sobre los trabajos de un DBA (Data Base Administrator), más bien puede ser considerado como una introducción. Los capítulos se pueden considerar en forma independiente cada uno Este trabajo puede ser considerado como una introducción al desarrollo de aplicaciones cliente/servidor de gran complejidad.

CAPITULO 1

INTRODUCCION A CLIENTE/SERVIDOR

1.1 Introducción

Lo que hace útil a un sistema de cómputo son las aplicaciones. De nada sirven procesadores cada vez más rápidos concentrados dentro de un gabinete, sin un conjunto de instrucciones y datos que manejar. En el pasado, debido a las características de las aplicaciones, los desarrollos se efectuaban en el lenguaje de programación más apropiado en función de su naturaleza, y en un equipo con un sistema operativo específico, probablemente propietario. Esto ocasionaba que las aplicaciones estuvieran corriendo en una gran variedad de computadoras de características muy distintas, con diferencias en el sistema operativo, arquitectura de cómputo y conectividad. Al existir incompatibilidades en el subsistema de comunicación del equipo, las posibilidades de interconexión con otros ambientes era mínima.

La proliferación de microcomputadoras y el auge en la instalación de redes empresariales, trajo consigo el desarrollo de dispositivos de conexión hacia los equipos principales. Fue así como aparecieron los gateways, una solución de hardware/software para la comunicación de las PCs con sistemas mayores, como minis y mainframes. Una estrategia para la interconexión de ambientes de cómputo heterogéneos, es la implantación de un protocolo de transporte común, y es en esta labor que TCP/IP se a convertido en el estándar.

En la terminología de cliente/servidor el término programa se substituye por aplicación dado que hablar de un programa en cliente/servidor es ambiguo, ya que puede ser un programa en el cliente, un programa en el servidor, o bien un manejador de bases de datos. En cambio la aplicación, es en esencia, el conjunto de programas necesarios para que el usuario final maneje de manera transparente la información.

La arquitectura Cliente/servidor es un modelo para el desarrollo de aplicaciones. Las que se hacen con este esquema pueden estar en equipos de cómputo diferentes, involucrando un ambiente distribuido para el acceso y proceso de la información, lo que hace la diferencia con la arquitectura centralizada en la que la mayor parte, si no es que el total, de los procesos ocurren sobre la computadora central mainframe, en esta arquitectura la terminal con la que el usuario interactúa simplemente manda las teclas oprimidas a la mainframe, que devuelve pantallas de información basadas en carácter a la terminal.

1.2 Arquitectura

"Una arquitectura técnica es un conjunto de definiciones y protocolos para construir un producto"[1] "La arquitectura de la aplicación es un extenso plan para el completo desarrollo de un sistema"[2]. Una arquitectura cliente/servidor abarca ambas, la arquitectura técnica y la arquitectura de la aplicación, las cuales se esbozan brevemente más adelante.

1.2.1 Arquitectura Servidor de Archivos

Los DBMSs tales como Microsoft Access y Borland Paradox están basados sobre una arquitectura de servidor de archivos, esto significa que la base de datos consiste de un archivo y que no hay un motor de base de datos corriendo en el servidor. Estas bases de datos corren exactamente igual sobre un servidor de archivos en red como ellos lo hacen sobre el disco duro de una PC local. Sin embargo, cuando se piensa en una arquitectura cliente/servidor más compleja generalmente se tiene en mente un activo proceso servidor de base de datos, el cual puede ser Oracle, DB2, Informix, Watcom, Sybase, o algún otro DBMS corriendo en el servidor, en ésta arquitectura el cliente pasa peticiones hacia el servidor el cual procesa las peticiones y retorna solamente los resultados hacia el cliente, los procesos del front-end son desarrollados en 4GLs tales como VB o PowerBuilder, y proporcionan acceso a back-ends tales como Oracle o Infotmix.

[1] Neil Jenkins. Client Server Unleashed.
[2] Idem

1.2.2 Arquitectura Cliente/Servidor

Cliente/servidor es una *arquitectura en la cual la funcionalidad* del sistema y su procesamiento es dividido entre el cliente que por lo general es una PC donde se coloca el front-end y el servidor en el que estará instalado un RDBMS el cual será el back-end. La funcionalidad del sistema como la lógica de programación, las reglas de negocios, y la administración de datos, son separadas entre las máquinas cliente y servidor. En esta definición, el término cliente/servidor describe una aplicación en la cual dos o más procesos lógicos separados trabajan juntos para completar una tarea, esto es llamado procesamiento cooperativo, a causa de que dos procesos separados están cooperando para completar una tarea. La separación apropiada de la funcionalidad del sistema entre el front-end y el back-end difiere con cada sistema y depende de la importancia que el diseñador de sistemas pone sobre factores como: rápido desempeño, ejecución de las reglas de negocio, mantenimiento de la aplicación y seguridad.

El usuario final usa la aplicación front-end para solicitar información de un servidor de bases de datos. El servidor de bases de datos recibe estas peticiones, las procesa, y manda los resultados de regreso hacia el cliente para ser desplegados. Los procesamientos del sistema para manejar estas tareas de acceso a la base de datos y presentación de la información también está dividida entre el cliente y el servidor. Los procesadores en ambas máquinas son utilizados en el mismo momento (in tandem) para satisfacer rápidamente las peticiones del usuario.

Si bien existe una separación lógica y física entre el cliente y el servidor, un sistema cliente/servidor debe coordinar el trabajo de ambos componentes y ocupar eficientemente cada uno de los recursos disponibles para completar las tareas asignadas. Las máquinas cliente se comunican con el servidor a través de una red de área local (LAN) o de área amplia (WAN). Además, un sistema cliente/servidor puede incluir más de un servidor en donde cada servidor disponible sobre el sistema manejará una función específica.

1.2.3 Arquitectura Técnica

La arquitectura técnica es la infraestructura sobre la cual la aplicación es construida, esta es una lista del hardware, software y elementos de red que residen sobre los componentes cliente y servidor para una aplicación.

- Sistemas Operativos
- DBMS
- Memoria
- Esquemas de Red (tipo y disposición física de la red)
- Esquema de Base de Datos (disposición física de la base de datos incluyendo Data Definition Languages)
- Asignación de Espacio
- Definición de Indices
- Seguridad y Candados
- Protocolos de Red
- Dispositivos de hardware (workstations, servers, bridges, routers)
- Sistemas Operativos de Red
- Monitoreo de Red y Afinación de software
- Gateways y software de conectividad (APIs, ODBC, messaging, ORBs, RPCs, etc.)
- Herramientas de Desarrollo de Aplicaciones
- Software de Seguridad
- Software de Administración de Configuración

1.2.4 Arquitectura de la Aplicación

La tecnología cliente/servidor permite al diseñador y al usuario final una gran flexibilidad para asignar funciones y datos entre el cliente y el servidor. El alcance y la complejidad de la arquitectura de la aplicación esta determinada por el número de clientes y servidores, tamaño de las bases de datos, volumen de datos, y cantidad de accesos concurrentes a los datos.

La arquitectura de la aplicación determina la distribución de los componentes de la aplicación entre el cliente y el servidor. Estos componentes representan capas conceptuales que son asignadas a diferentes componentes físicos o de hardware. Esencialmente, tres capas necesitan ser distribuidas: presentación, reglas/lógica de negocio, y administración de datos (ver tabla 1.1)

Capa	Contenido
Presentación	Formatos de pantalla Navegación pantalla/menú Manejo de ratón y teclado *Edición de entradas*
Reglas/Lógica de Negocio	Lógica booleana Validación de dominio y rango Validación de dependencia de dato Chequeo del estado de transición de precedencia Procesamiento de reglas/seguridad Manejo de mensajes
Administración de datos	Acceso a datos/Data Manipulation Language Manejo de concurrencias Análisis del SQL Seguridad del dato Seguros

Tabla 1.1

La capa de presentación suministra la interfaz al usuario para accesar el sistema. Esta permite a los usuarios finales introducir y manipular datos, analizar información y navegar a través del sistema. Esta también podrá interactuar con aplicaciones existentes en el escritorio. Esta capa deberá cubrir la complejidad de la totalidad del sistema y proporcionar una interfaz fácil de usar. La lógica de validación y verificación es frecuentemente necesaria en esta capa para editar valores de entrada y selecciones.

La capa de reglas/lógica de negocios procesa las políticas y restricciones que la aplicación es responsable por hacer cumplir que incluye: decisiones, ejecución de políticas, y decisión de administración de recursos. Estas son reglas de negocio atribuidas a los datos, estado y modelos evento-proceso.

La capa de administración de datos mantiene consistencia y protege los datos por medio de la ejecución de integridad referencial y seguridad de dato. Las transacciones y consultas contra la base de datos son manejadas en esta capa e incluyen lo siguiente:

- Accesos
- Validación
- Actualización
- Inserción
- Boqueo
- Recuperación
- Seguridad
- Registros
- Respaldo
- Restauración

Cliente/servidor permite implementar estas capas en una forma modular para la apropiada localización física. Esto permite la flexibilidad para cambiar ubicaciones si las necesidades de negocio o problemas de desempeño así lo requieren. Las capas de presentación, reglas de negocio y administración de datos de las aplicaciones pueden ser particionadas en una variedad de modos. Cada capa, o combinación de capas, residen en los componentes cliente y servidor. En algunos casos, porciones de una sola capa, tales como reglas de negocio, podrían ser divididas entre múltiples capas físicas. Por ejemplo, ambas, la aplicación en la estación de trabajo y la aplicación en la máquina servidora, podrían contener reglas de negocio.

1.3 Componentes

1.3.1 Cliente

La parte de un sistema cliente/servidor que la mayoría de los usuarios siempre ven es el lado cliente. El lado cliente consiste de la aplicación front-end y la computadora que un usuario final usa para accesar información de la base de datos. La mayoría de las aplicaciones cliente/servidor tienen una interfaz gráfica de usuario (GUI). Un front-end con GUI proporciona un ambiente amigable al usuario, una interfaz point-and-click que es más intuitiva para usarse que un sistema character-based. Un GUI contiene controles visuales, tales como botones y listas, y puede también desplegar imágenes gráficas. Las aplicaciones GUI son usualmente diseñadas de acuerdo a ciertos estándares de la industria. Para seguir estos estándares, la aplicación cliente/servidor tendrá una presentación y percepción estandar que se traduce en una curva de aprendizaje reducida.

La utilización del CPU del lado cliente para la realización de una parte de las tareas del sistema libera al servidor de la realización de procesos que no le corresponden. La utilización de una interfaz familiar y amigable al usuario así como el equipo que ya se está acostumbrado a usar y en el entorno de trabajo habitual evita el entrenamiento de los usuarios finales en un entorno operativo específico, ya que se aprende solamente el uso de la aplicación y no el de todo un sistema operativo, que en ocasiones es muy distinto a lo que el usuario acostumbra usar.

Detrás de la interfaz gráfica está la compleja programación de manejo de eventos que determinan que es lo que la aplicación hará en respuesta a las acciones del usuario. Detrás de cada evento, tal como un click sobre un botón, hay alguna programación que invoca una acción. Esta acción puede ser mandar una petición a la base de datos, validar la introducción de datos, desplegar información, realizar cálculos, o abrir otra pantalla. Para asegurar que la aplicación front-end pueda realizar todas estas diferentes tareas eficientemente, la máquina cliente deberá satisfacer ciertos requerimientos de hardware respecto a rapidez del procesador y memoria.

1.3.2 Servidor

El servidor es el administrador de los recursos. Estos pueden ser: periféricos como faxes o impresoras, sistemas de telecomunicación, archivos y bases de datos. Es en este último recurso en donde se aplica el modelo cliente/servidor con mayor fuerza. Los clientes utilizan la Interfaz Gráfica de Usuario (GUI), la cual proporciona un agradable ambiente de trabajo y oculta todas las rutinas y procesos que se realizan en el servidor para generar peticiones de acceso a los datos y de procesamiento de los mismos.

El servidor (proceso o procesos), o back-end, consiste de un sistema administrador de base de datos (DBMS) y la computadora sobre la que éste corre. El DBMS almacena información en bases de datos y habilita a múltiples clientes para accesarla simultáneamente. Las aplicaciones front-end pueden leer, actualizar, insertar y borrar datos de la base de datos a través de un lenguaje de consulta estructurado (SQL), un estandar para accesar bases de datos relacionales. Para asegurar un rápido desempeño, un DBMS deberá soportar características avanzadas tales como procedimientos almacenados y triggers. El DBMS también realiza numerosas funciones para mantenimiento de la base de datos, incluyendo el aseguramiento de la integridad de los datos, administración de seguros, manipulación de procesamiento de transacciones, hacer cumplir reglas de negocio, y soportar indexaciones. Para manejar eficientemente el total de éstas tareas y responsabilidades eficientemente, la computadora que opera como servidor es usualmente una máquina dedicada de alto rendimiento. Éste también deberá ser escalable de modo que pueda soportar el incremento en el número de usuarios así como volúmenes más grandes de datos en el futuro.

1.3.3 Red

El tercer componente de un sistema cliente/servidor es la red. Las computadoras cliente se comunican con el servidor a través de la red. La red juega un papel importante en la determinación del desempeño de un sistema cliente/servidor a causa de que éste controla

que tan rápido la petición del cliente es trasladada a el servidor y que tan rápido el resultado es llevado de regreso hacia el cliente. En el desarrollo de un sistema cliente/servidor, uno de los objetivos del desarrollador es separar eficientemente el procesamiento entre el cliente y el servidor con el objetivo de minimizar el trafico en la red.

En la implantación, una vez que la aplicación ya ha sido desarrollada, se procede a distribuirla en los diferentes clientes y servidores. Este es un paso delicado, ya que implica problemas de distribución y de desempeño que no siempre es posible detectar en las fases de diseño y desarrollo. Por ejemplo, en el caso de una plataforma cliente homogénea, puede ser una buena idea centralizar todos los archivos (por ejemplo, los ejecutables, run-times, librerías y hasta los bitmaps) en un servidor adicional que compartan todas las computadoras ésto facilitará que si se desea una corrección al programa se haga en un solo lugar, y esto puede ser justificable pero dado que las herramientas de desarrollo gráficas son un estándar y estas requieren de un ancho de banda que favorezca el rápido acceso a la información y evite cuellos de botella, esto puede afectar el desempeño de la red en los momentos pico de carga de trabajo o que se presenten conflictos con otras aplicaciones de mayor prioridad en el sistema, que pueden adueñarse del CPU servidor y detener o hacer notoriamente más lento el lado cliente. Cuando un sistema cliente/servidor es implementado, la arquitectura de la red deberá ser optimizada para asegurar que ésta pueda soportar apropiadamente el incremento de tráfico que será generado.

1.4 Ventajas

La tecnología cliente/servidor capacita a las compañías para desarrollar sistemas de información robustos y flexibles para resolver necesidades de negocios específicas. Se puede desarrollar un sistema cliente/servidor que soporte un pequeño departamento con solo tres usuarios o una aplicación de mision critica: sistema de nivel empresarial que soporte miles de usuarios a través del país, incluso del mundo. Los sistemas de procesamiento de transacciones en línea, los sistemas de información ejecutiva, así como los sistemas de soporte de decisiones pueden ser diseñados con éste tipo de arquitectura.

Un sistema cliente/servidor puede proporcionar capacidades de acceso remoto, lo cual capacita a personas trabajando fuera de la oficina a accesar la información como si ellos estuvieran físicamente en la oficina. Los usuarios pueden transparentemente accesar información de una variedad de diferentes fuentes de datos (DBMSs) sin considerar la localización física de la fuente. El usuario no necesita avanzados conocimientos de como y desde donde llegan los datos requeridos. Además, los sistemas cliente/servidor pueden compartir información con otras aplicaciones, tales como procesadores de palabras y hojas de cálculo, y también ser integrados con tecnologías más avanzadas tales como document imaging. Las compañías están desarrollando sistemas cliente/servidor para resolver una variedad de diferentes necesidades de negocios.

Un sistema cliente/servidor diseñado apropiadamente provee a una compañía y sus empleados con numerosos beneficios. El sistema capacita a las personas para hacer mejor su trabajo al permitirles enfocar su tiempo y energía en adquirir nuevas cuentas, cerrar tratos, y trabajar con clientes, en lugar de trabajar en tareas administrativas; proporciona acceso instantáneo a información para tomar decisiones, facilita la comunicación, y reduce tiempo, esfuerzo, y costo para realizar tareas.

Los negocios están cambiando rápidamente. El mercado es ahora más competitivo que nunca y continuara siendo así cada vez más. Las compañías se están enfrentando con el desafío de la preservación de sus negocios al día, y para ello deberán hacer negocios eficientemente a fin de permanecer en el mercado.

Los sistemas computacionales que fueron desarrollados en los 80s tendieron a ser basados al rededor de un sistema computador centralizado. Las LANs fueron conectadas a este sistema, no obstante, una cantidad pequeña de los procesamientos de negocios fue hecho en las LANs o las Pcs. Cualquier cambio a los negocios se realizaba sobre el sistema centralizado. Si un nuevo producto estaba listo para ser vendido o un nuevo sistema de contabilidad estaba listo para ser implementado, éste era normalmente colocado sobre la computadora central. Con el tiempo cada vez más sistemas fueron colocados sobre el computador centralizado, el costo de correr esta máquina se elevó. El

tiempo para cambiar este sistema, si una nueva función de negocios se requería, también se incremento. Al paso del tiempo, esta situación se convirtio en algo tan inconveniente que es comun oír de sistemas que tomaron un tiempo excesivo de tres años para su desarrollo e implementación cuando el producto necesitaba estar listo para ser trabajado en seis meses. Es claro que las compañías no pueden continuar consumiendo tiempos tan largos para obtener sistemas en el medio ambiente de negocios de hoy en día.

Los sistemas cliente/servidor dividen una gran área de negocios de una compañía en varias unidades distintas, las cuales pueden ser consideradas independientes aunque esten integradas. Si estos sistemas son correctamente desarrollados, pueden proporcionar mejoras para los negocios más rápido de lo que puede un sistema centralizado. Algunas de las razones son:

- El área de negocios es más pequeña y por lo tanto puede ser cambiada más rápidamente.
- Los cambios en el sistema cliente servidor normalmente no afectan a tantos usuarios como lo hacen los cambios a un sistema centralizado.
- Se pueden hacer prototipos rápidamente sobre sistemas más pequeños lo cual resulta en un tiempo de desarrollo más rápido.
- Un sistema operativo de red puede soportar muchos diferentes tipos de servidores. Implementar una nueva función sobre un servidor específico puede ser más rápido que intentar implementar un nuevo producto sobre un mainframe.

1.4.1 Acceso a la información

Un sistema cliente/servidor bien diseñado provee a los usuarios un fácil acceso a toda la información que necesitan para hacer sus trabajos. El usuario despliega la información requerida mediante la aplicación front-end con solo algunos clicks. Esta información podría residir en diferentes bases de datos o incluso sobre servidores separados físicamente, sin importar lo complicado del acceso a los datos, para el usuario este proceso es transparente. El sistema cliente/servidor también contiene características que

capacitan al usuario para más adelante analizar la información recuperada. A causa de que el acceso a la información y la funcionalidad es proporcionado desde un solo sistema, los usuarios no necesitan muchas claves de entrada (login) dentro de diferentes sistemas o depender de otras personas para obtener sus respuestas.

1.4.2 Incremento en la productividad

Un sistema cliente servidor incrementa la productividad de sus usuarios al proveerlos con las herramientas para completar sus tareas más rapido y más facilmente. Por ejemplo, una pantalla de introducción de datos con controles gráficos y lógica de programación para soportar reglas de negocios habilita a los usuarios para introducir información más rápidamente y con menos errores y omisiones. El sistema automáticamente valida la información, realiza los cálculos, y reduce la intoducción de datos duplicados. Un sistema cliente/servidor puede estar integrado con otras tecnologías tales como correo electrónico, y groupware para proporcionar mejoras adicionales en la productividad.

1.4.3 Automatización de procesos de negocios

Un sistema cliente/servidor puede automatizar los procesos de negocios de una compañia y ser una solución de flujo de trabajo eliminando una gran cantidad de labores manuales y ejecutando los procesos requeridos más rápido y con menos errores. Por ejemplo, un proceso de negocios común, de una compañía, es la realización de una orden de compra manualmente, ésta actividad involucra búsquedas a través de un gabinete para encontrar una forma para una orden de compra, llenarla, realizar todos los cálculos con una calculadora, determinar como deberá aprobarse, y entonces mandarlo a la persona apropiada a través del mensajero. Un sistema cliente/servidor puede automatizar este proceso y terminarlo en un lapso menor de tiempo. Una versión electrónica de la orden de compra puede ser diseñada en forma de una aplicación front-end y estar disponible en linea. Usando una interfaz gráfica un usuario rápidamente introduce la información, y el sistema automáticamente ejecuta todos los cálculos, entonces la forma es automáticamente despachada a través de la red hacia la persona apropiada (basada en

una regla de negocio) para su aprobación. La persona que la debe aprobar inmediatamente recibe la orden de compra en su buzón electrónico para revisión y no tiene que esperar hasta que este llegue a través del mensajero.

1.4.4 Generación de reportes

La información en un sistema cliente/servidor se almacena en una base de datos relacional, lo cual implica que debe ser consultada fácilmente para propósito de generación de reportes. Los programadores pueden crear nuevos reportes rápidamente usando SQL. No obstante, un sistema cliente/servidor puede proporcionar características que capacitan a usuarios finales para crear sus propios reportes y adecuar algunos ya existentes sin tener que aprender SQL. Con estas capacidades, los usuarios pueden generar sus propios reportes mucho más rápido que en el pasado y no son completamente dependientes de los sistemas de información para obtenerlos. Aquellas personas que toman una copia del reporte y reescriben toda la información en una hoja de cálculo para poder generar reportes pueden ahorrarse una gran cantidad de tiempo usando un sistema cliente/servidor.

1.4.5 Servicio a clientes

Una compañía puede mejorar su servicio al cliente al proporcionar respuestas más rápidas y minimizar el número de veces que un cliente tiene que contactar a la compañía. Un sistema cliente/servidor habilita a los representantes de servicios al cliente a servir mejor, y una razón clave es su capacidad para proporcionar información de diferentes fuentes de datos. Un banco, por ejemplo, puede tener diversas bases de datos separadas físicamente. Cada una de ellas almacena un tipo específico de información de la cuenta del cliente tales como ahorros, hipoteca y estudio de préstamo. Actualmente, un cliente que tiene los tres tipos de cuentas con este banco y necesita información sobre todas ellas tiene que ocupar tres diferentes números, lo cual es muy inconveniente. Un sistema cliente/servidor puede ser diseñado para proporcionar un servicio al cliente con acceso a la información de las tres bases de datos. Por tanto, el cliente únicamente necesita ocupar

un número. Los clientes buscan los servicios con este tipo de conveniencia.

1.4.6 Desarrollo de aplicaciones

La mayoría de las herramientas para desarrollo cliente/servidor capacitan a los programadores para crear aplicaciones tomando ventaja de las técnicas de programación orientada a objetos y desarrollando aplicaciones modulares. Reusando objetos y código que ya a sido escrito para sistemas existentes, nuevos sistemas cliente/servidor pueden ser desarrollados mucho más rápidamente. Las herramientas de diseño gráficas proporcionan facilidades drag-and-drop que permiten a los programadores crear rápidamente pantallas visuales sin tener que programar el código para la generación de las mismas. Las aplicaciones cliente/servidor también pueden ser fácilmente modificadas en caso de que un cambio, tal como una nueva regla de negocios, sea necesario. Además, las herramientas para cliente/servidor pueden ser usadas para crear prototipos de sistemas rápidamente, que capacitan al desarrollador para mostrar el sistema a los usuarios y obtener retroalimentación inmediata.

1.4.7 Reducción de Costos

Un sistema cliente/servidor reduce costos en varios modos, algunos de los cuales son más fáciles de cuantificar que otros. Muchas compañías han reemplazado sus sistemas mainframe con cliente/servidor y se han ahorrado millones de dólares en costos de mantenimiento anual. Otros se han beneficiado del acceso de información en línea y reducido significativamente sus costos por el uso de papel incluyendo su compra, almacenamiento, y distribución. La información en línea también capacita a las personas para identificar más rápido el mal funcionamiento de algún punto del negocio y así tomar las medidas correspondientes. A causa de que las personas pueden terminar sus tareas más rápido, ellos ahorran tiempo y esfuerzo, lo cual también se traduce en un ahorro financiero.

CAPITULO 2

MODELO CLIENTE/SERVIDOR DE TRES HILERAS

2.1 Hilera

Las Hileras (Tiers) son las plataformas físicas sobre las cuales residen las capas de las aplicaciones cliente y servidor. Una primera generación de aplicaciones cliente/servidor utiliza dos plataformas computacionales en colaboración, la aplicación front-end que reside sobre la estación de trabajo del usuario final proporcionando la interfaz a la aplicación y la mayor parte de las reglas de negocios y el componente back-end reside sobre uno o más servidores proporcionando el servicio de bases de datos incluyendo el acceso a los datos, administración y servicios de integridad, y algún procesamiento de reglas de negocio en la forma de procedimientos almacenados y triggers.

En aplicaciones Two-Tier (dos hileras), las aplicaciones cliente accesan al servidor de base de datos usando una interfaz, como puede ser, ODBC. Los procesos clientes son construidos usando herramientas de desarrollo de aplicaciones como Powersoft's PowerBuilder, Centura's SQLWindows o Microsoft Visual Basic. A causa de que las reglas de negocio pueden ser ejecutadas sobre el cliente como lógica de aplicación, ellos son conocidos como *fat clients* o *client-centric*. Para muchas aplicaciones, esta es una arquitectura relativamente simple.

Una arquitectura física más escalable pero también más compleja usa tres o más hileras, en la cual en lugar de emplear lógica de aplicación basada en servidor que es parte de los servicios de base de datos, la lógica de negocios es aislada de los servicios de base de datos al colocarla sobre su propia capa (ver figura 2.1), de este modo el dato es separado de la capa de la lógica de negocios, en esta arquitectura la lógica de negocios operará *sobre su propia plataforma corriendo* bajo el control de su propio sistema operativo, o alguna forma de middleware.

Las ultimas versiones de PowerBuilder y Visual Basic se están desarrollando para habilitar tres (o *n*) particionamientos de capas usando OLE (object linking and embedding) en conjunción con tecnologías RPCs (remote procedure calls).

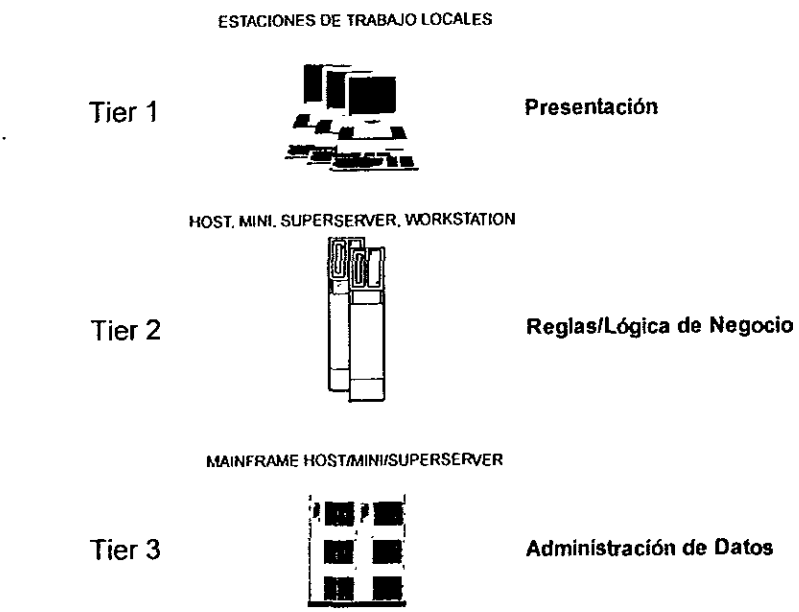

Figura 2.1

En la arquitectura three-tier, una aplicación servidora esta en la nueva capa intermedia y es usada para accesar multiples y heterogéneas bases de datos, esto evita la necesidad de usar un lenguaje propietario (como procedimientos almacenados SQL) para la ejecución de las reglas de negocios en forma centralizada, además facilita la reutilización de la lógica de aplicaciones y permite al código de la aplicación residir sobre una variedad de plataformas. El mantenimiento de las reglas de negocio es más fácil; en lugar de redesplegar la lógica de las aplicaciones en múltiples estaciones de trabajo de usuarios finales como en la arquitectura two-tier, la lógica es actualizada únicamente sobre el o los servidores en los cuales esta reside.

La arquitectura de una aplicación de tres o más capas es un medio ambiente más complejo que involucra hardware y software adicional. Sin embargo el desarrollo de alguna aplicación puede requerir ésta segunda generación de arquitectura cliente/servidor.

2.2 Modelo de Servicios

En el modelo de servicios una capa tiene comunicación interna con sus padres, hijos, o hermanos - éste puede hacer peticiones y retornar respuestas a un proceso dentro de su propia capa. Una capa puede comunicarse inmediatamente arriba o abajo de su capa. En general la única comunicación que no puede ocurrir es una aplicación cliente comunicándose directamente con un servicio de datos, como ocurre en la arquitectura tradicional de estratos (Two-Tier).

En una arquitectura tradicional de estratos una capa puede comunicarse únicamente con otra capa directamente arriba o abajo. Aunque los servicios de usuario, servicios de negocios, y servicios de datos que se muestran en la tabla 2.1 pueden comunicarse entre ellos como en el modelo de estratos, este modelo es llamado el modelo de servicios a causa de que, a diferencia de un modelo estratificado, cualquier servicio puede invocar a otro servicio dentro de su propia hilera.

Un ejemplo de esto podría ser un servidor de negocios de procesamiento de comisiones usado para interactuar con un servidor de negocios de personal y un servidor de negocios de países para colaborar en la realización de una comisión, sin que la aplicación cliente tenga interacción visible con el servidor de negocios de países (ver figura 2.2). Una petición de una aplicación cliente para actualizar una comisión, podría ser solicitada directamente desde un servidor de negocios de procesamiento de comisiones, el cual a su vez solicita a un servidor de negocios de países para programar el transporte. Ambos servidores usan un servicio de bases de datos. En este caso, una sola petición de una aplicación cliente involucra dos servidores de negocios y un servidor de datos.

Los componentes de cada servicio de usuario, servicio de negocio o servicio de dato, se encuentran dentro del contexto de una sola hilera y un solo servicio, y cada hilera alberga muchos servicios. Mientras un servicio es un concepto lógico, un componente describe un paquete físico de funcionalidad. Así, un servicio puede ser descrito como una agrupación lógica de componentes físicos.

Hilera	Tipo de servicio	Carácterísticas	Funciones	Herramientas
Servicios de usuario	Aplicaciones cliente	Interfaz de usuario	Presentación, navegación	4GLs; aplicaciones de escritorio
Servicios de negocios	Servidores de negocios	Políticas de negocio, objetos, propiedades, y métodos	Ejecución de política de negocios, seguridad, integridad de negocios	Algunos 4GLs,COBOL,C
Servicios de datos	Servidores de datos	Administradores de datos	Integridad y seguridad de datos, recuperación y almacenamiento de datos	Bases de datos, sistemas de imágenes, otros

Tabla 2.1[1]

Los componentes son piezas de software que son desplegadas en las computadoras:

- La capa de la aplicación cliente esta compuesta de aplicaciones cliente, tales como captura de comisiones y consulta de las mismas.
- La capa de servidor de negocios esta compuesta de servidores de negocios, tales

[1] Three-Tier Client Server Arrives. Visual Basic Programmer's Journal. November 1995.

como procesamiento de comisiones y programación del transporte.

- La capa de servicios de datos esta compuesta de servidores de datos, tales como comisiones y personal.

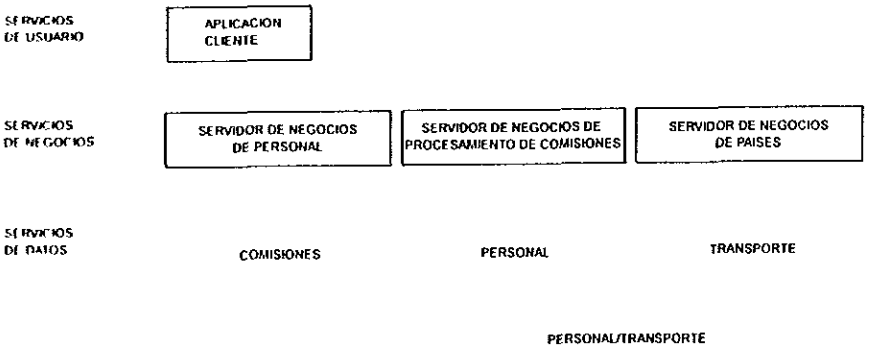

Figura 2.2

Esta es una construcción jerárquica simple. Un solo servidor es frecuentemente formado de un solo componente. Algunos servidores proporcionan servicios complejos, sin embargo, de esta manera en lugar de ser colocados en grandes servidores monolíticos, ellos son divididos en múltiples componentes. Una aplicación cliente, por ejemplo, es un servicio de usuario. Una aplicación cliente de procesamiento de comisiones puede estar compuesta de varios componentes: múltiples VB EXEs, OCXs y VBXs. El mismo concepto se aplica a servidores de negocios y servidores de datos.

Los componentes son el denominador común más bajo: ellos no pueden ser divididos en ningún otro. Ellos también pueden ser considerados cajas negras: que es lo que entra y que es lo que sale esta bien definido, pero se desconoce que ocurre dentro del componente.

2.3 Aplicaciones Objeto

Los programas en la capa de servicios de usuario, servicios de negocios, y servicios de

datos que se comunican entre sí son llamados aplicaciones objeto. El diseño de aplicaciones objeto difiere del tipo de aplicaciones con las cuales se está más familiarizado. Cuando se diseñan programas, se tienen conceptos tales como código reusable, object interfaces, y un método de comunicación (el método de comunicación utilizado en Windows es OLE Automation). Las técnicas de programación orientadas a objetos empleadas difieren de las técnicas tradicionales que ven el mundo como un programa individual que trabaja sobre datos aislados.

El concepto de un programa cambia en este medio ambiente: una solución de negocios por lo general no está compuesta de una sola aplicación monolítica individual, sino por el contrario de una colección de servicios distribuidos. En donde cada uno de esos servicios es un programa ejecutable.

Para que dos servicios se comuniquen, ambos tienen que hablar el mismo lenguaje o protocolo. La comunicación puede ocurrir localmente (sobre las mismas máquinas) o remotamente (sobre máquinas separadas), sin embargo esto deberá ser transparente al proceso.

En VB y en el medio ambiente de Windows, el mecanismo de comunicación es OLE. Para comunicar dos servicios, éstos deberán estar de acuerdo en dos cosas: una bien definida, interfaz pública y un método para transportar su dialogo. La interfaz es OLE object interface, y el método de transporte es OLE Automation.

La combinación de las propiedades, métodos y colección de objetos es llamada su interfaz. Una interfaz de objeto puede permanecer igual aun cuando cambie el encapsulado de la lógica de negocios.

Cuando ocurre una comunicación OLE en el contexto de la misma máquina, esta es llamada automatización local o solamente automatización. Cuando se maneja sobre una red a ésta se le llama automatización remota.

2.4 Servidor de Negocios

Los servidores de negocios contienen el total de la lógica de negocios del sistema. Los servidores de negocios controlan la forma en que los datos empresariales son administrados por los servicios de datos. Los servidores de negocios son generalmente grandes objetos que administran una específica entidad de negocios. En el contexto de una partición de aplicación, ellos son frecuentemente llamados objetos de servicio o line-of-business objects (LOBjects).

Las reglas de los negocios son almacenadas en la capa de servicios de negocio, pero los datos elementales que estas reglas utilizan para su desempeño están en la capa de servicio de datos. La mayoría de los servicios de datos que se usan son DBMSs tales como Oracle, Sybase, Informix, y DB2. El modo como esos servicios de datos almacenan los datos en el modelo de servicios three-tier no difiere mucho del modo como ellos almacenan datos en el modelo two-tier. Lo que significa, que la teoría de las bases de datos relaciónales y un buen diseño también son aplicables en la arquitectura three-tier.

El cambio de como se utilizan los servidores de bases de datos en la arquitectura Three-Tier radica en la colocación de las reglas de negocio. A causa de que en el mundo Two-Tier las reglas de negocio se colocan en un lugar centralizado (los servidores de bases de datos), los vendedores de bases de datos empiezan a expandir la funcionalidad de su sintaxis de procedimiento de almacenado. Dado que la sintaxis empezó a parecerse más a un lenguaje de programación, los desarrolladores son capases de agregar lógica de negocios más compleja dentro de procedimientos almacenados. El mundo three-tier para esta tendencia.

Las reglas de negocios se manifiestan en el código como declaraciones Case o If complejas, que necesitan tratar con seguridad, y múltiples y heterogéneos servicios de datos; por lo que deben ser desarrolladas en un medio ambiente de equipo y deben proporcionar una interfaz lógica, fácil de usar.

Sin embargo, en el mundo Three-Tier, los procedimientos almacenados no desaparecen, éstos son útiles para encapsulamiento de funciones de datos elementales, tales como agregar un número de orden válido e insertar un valor por omisión. Una o más de esas funciones de datos elementales usualmente componen un método de servicio de negocios, tal como crear comisión.

Cualquier función de dato elemental es una función que deja el dato de la empresa en un estado valido. Por ejemplo, una función para agregar una comisión asegura que pertenezca a un empleado existente, pero no necesariamente ajustaría el rango de viáticos del empleado (una regla de negocio). Algunas veces la línea entre las reglas de negocio y las funciones de datos elementales no están muy claras. Una forma de determinar si una regla es de negocio o de dato es preguntarse a si mismo esta cuestión, "¿Anticipar el manejo es siempre una decisión que cambia la validez de esta regla? Si la respuesta es sí, esta es probablemente una regla de negocios. Si la respuesta es no, entonces probablemente sea una regla de dato"[1].

La desventaja más grande de usar procedimientos almacenados en lugar de declaraciones SQL incrustadas en algún programa para funciones de datos elementales es que los procedimientos almacenados no son portables. Aun cuando, por ejemplo, declaraciones ANSI SQL son portables de SQL Server a Oracle, los procedimientos almacenados de SQLServer Transact-SQL no son fácilmente traducidos a procedimientos de PL/SQL de Oracle.

Sin embargo, hay ventajas al usar procedimientos almacenados para funciones de datos elementales. Por ejemplo, son más rápidos, mas fáciles de administrar y más fáciles de afinar para su desempeño que SQLs dinámicos diseminados por todos los servidores de negocios. Además, los procedimientos almacenados permiten al equipo de desarrollo sacar ventaja de las habilidades: los analistas pueden diseñar código de servidor de negocios, mientras los DBAs pueden diseñar código DBMS SQL.

[1] Three-Tier Client/Server Arrives. Visual Basic Programmer's Journal, November 1995.

Los servidores de negocios generalmente deberán residir sobre una máquina servidor centralizada. A causa de que ellos ejecutan la mayoría de la lógica de procesos de negocios, localizarlos sobre un servidor central proporciona grandes ventajas: el desempeño del servidor puede ser medido y su capacidad incrementada, el uso del sistema es fácilmente monitoreado, la funcionalidad de negocio esta encerrada en un medio ambiente seguro y los cambios en la lógica de negocios, los cuales son frecuentemente sensibles al tiempo, solamente necesitan ocurrir en el servidor.

Los servidores de negocios son críticos para las operaciones diarias del sistema - ellos son accesados por muchas aplicaciones cliente. Por tal razón, los servidores de negocios deberán correr sobre máquinas de alto rendimiento, con tolerancia a fallas.

Los servicios de datos son usados por varios servidores de negocios clientes. Ellos soportan las aplicaciones de la línea de negocios de la organización, son parecidos a los servidores de negocios, y deberán residir en máquinas centralizadas, con tolerancia a fallas.

La mayoría de los servicios de datos son servidores DBMS que tienen alguna flexibilidad en el tipo de sistema operativo y hardware en el cual son desplegados; tales como Oracle, Sybase, y DB2 que corren sobre una variedad de sistemas operativos, incluyendo Unix y OS/2, y sobre diferentes plataformas de hardware, tales como IBM, Hewlett-Packard, Data General y Digital Equipment.

2.5 Servicios de Usuario

Dentro de un sistema three-tier un servicio individual de usuario usualmente puede ser identificado como la aplicación cliente. La aplicación cliente coordina el restante de los servicios dentro del sistema proporciona la interfaz primaria al usuario y administra la comunicación entre los servicios de usuario y negocios.

La aplicación cliente es usualmente una aplicación estándar, que no necesita ser un

servidor a causa de que generalmente no necesita ser controlada por ningún otro servicio. A causa de esto la aplicación cliente contiene la mayor parte de el código de presentación, esto se parece a la tradicional aplicación cliente realizada en two-tier. La diferencia es que esta no contiene SQL, acceso a la base de datos o lógica de negocios; solo código de presentación.

Las herramientas de escritorio, tales como Excel, pueden usarse para crear aplicaciones cliente por usuarios finales, prototipos, y aplicaciones con un corto alcance de vida. Pero para aplicaciones que requieren múltiples desarrolladores, compleja interacción de interfaces gráficas y una extensión de vida medida en años se debe elegir un 4GL con un buen control de versión que soporte herramientas de desarrollo en equipo y que no permita que el código fuente esté disponible para el usuario.

Desde el punto de vista presentación, en la arquitectura Three-Tier la aplicación cliente no es una imagen espejo del diseño lógico de la base de datos como ocurre en la arquitectura Two-Tier. No hay razón de que los usuarios sean presentados con un mapeo, uno a uno, del diseño lógico de la base de datos; una aplicación cliente debe ser una representación del proceso de trabajo del usuario. Diferentes usuarios podrían requerir interfaces muy diferentes a el mismo servidor de negocios.

2.6 Beneficios de Tres-Hileras

Tres-hileras probablemente parece involucrar una mayor cantidad de trabajo que una arquitectura tradicional two-tier cliente/servidor. Aquí se tocara brevemente algunas de las ventajas más significativas de una arquitectura three-tier.

En el medio ambiente three-tier, puesto que los servicios individuales pueden comunicarse sobre una red, cualquier servicio puede ser distribuido a cualquier máquina. Diferentes servicios pueden correr bajo diferentes sistemas operativos. También es posible tener las tres hileras corriendo sobre la misma máquina.

El cambio de administración y mantenimiento es mucho más manejable. Si los cambios ocurren en el funcionamiento interno de un servicio y no en su interfaz, entonces ningún otro servicio necesita ser modificado. Por ejemplo, si el servidor de negocios de procesamiento de comisiones necesita cambiar el modo como los viáticos son ajustados, los cambios pueden ser realizados en el servidor sin cambiar la aplicación cliente.

En la arquitectura three-tier, la lógica de negocios no esta limitada ni entrelazada con la aplicación de escritorio. Si tres diferentes aplicaciones cliente necesitan funcionalidad de introducción de comisiones, lo complicado del proceso de negocios no necesita ser redundantemente codificado y mantenido en tres diferentes lugares. Por el contrario, un solo servidor de negocios de procesamiento de comisiones atiende a las tres aplicaciones cliente. Igualmente, el desarrollador de la aplicación cliente no tiene que tratar con el código fuente del servicio de negocios o datos.

Si esto suena como código reusable, lo es - pero no meramente de objetos tales como botones OK o list boxes. Se reusan objetos que tienen un valor de negocio como pueden ser: un servidor de facturación, un servidor de embarque, o un servidor de administración de almacén.

Uno de los aspectos más importantes del modelo de servicios es que, al menos en el modelo, no importa donde esta colocado físicamente el software sobre la red. Desafortunadamente, el mundo real no es tan flexible. Sistemas operativo, redes, compiladores y lenguajes limitan el medio en el que se puede desplegar componentes de servicio.

CAPITULO 3

CONECTIVIDAD ENTRE EL CLIENTE Y EL SERVIDOR

3.1 Introducción

El concepto de red como es conocido hoy en día comenzó a principios de los 80's. Con el surgimiento de las PCs y las herramientas de escritorio de principios hasta mediados de los 80's, los usuarios finales dependían de sus PCs para soporte de decisiones. Pero usualmente una aplicación de misión crítica de una compañía así como los datos eran almacenados sobre mainframes o minicomputadoras. La conectividad fue establecida por emulación de terminal y métodos de transferencia de archivo. Como las PCs y el software para las PC comenzó a ser más sofisticado, las LANs proliferaron de mediados a finales de los 80s por sus capacidades de servicio de archivos y compartición de recursos. Sin embargo, los administradores de sistemas de información (MIS) (Management Information System) veían una gran diferencia entre las LANs y los sistemas centralizados (mainframes y minicomputadoras). Con la aceptación de la computación cliente/servidor como una plataforma tecnológica para aplicaciones de misión crítica, la red ya sea de área local o área amplia asumió una gran importancia al proporcionar la comunicación para aplicaciones cliente/servidor.

3.2 Topología de red

Las redes locales son frecuentemente caracterizadas en términos de su topología. Tres son las topologías comunes: estrella, anillo (ring), y bus o árbol.

En una topología **estrella**, un central switching element es usado para conectar todos los nodos en la red como lo muestra la figura 3.1. El elemento central usa circuit switching para establecer una ruta dedicada entre dos estaciones deseando comunicarse.

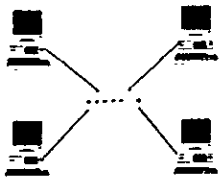

Figura 3.1 Estrella

La topología **anillo** (figura 3.2) consiste de un círculo cerrado, con cada nodo unido a un elemento repetidor. Los datos circulan al rededor de el anillo sobre una serie de ligas de datos punto a punto entre repetidores. Una estación deseando transmitir espera por su siguiente turno y entonces manda datos por el anillo en la forma de un paquete. El paquete contiene campos de direcciones de origen y destino así como datos. Como el paquete circula, la estación destino copia los datos dentro de un buffer local. El paquete continua circulando hasta que éste retorna a la estación origen, proporcionando una forma de aceptación. Un protocolo de control distribuido es usado para determinar la secuencia en la cual los nodos transmiten.

Figura 3.2 Ring

La topología **bus o árbol** (figura 3.3) es caracterizada por el uso de un medio multipunto. El bus es simplemente un caso especial de árbol, en el cual hay solamente un tronco, sin ramas. A causa de que todos los dispositivos comparten un medio de comunicación común, solo un par de dispositivos sobre un bus o árbol pueden comunicarse en un momento dado. Un protocolo de acceso al medio es utilizado para determinar cual estación transmitirá a continuación. Como con el anillo, la transmisión emplea un paquete conteniendo campos de direcciones origen y destino Cada estación monitorea el medio y

copia paquetes direccionados a el mismo.

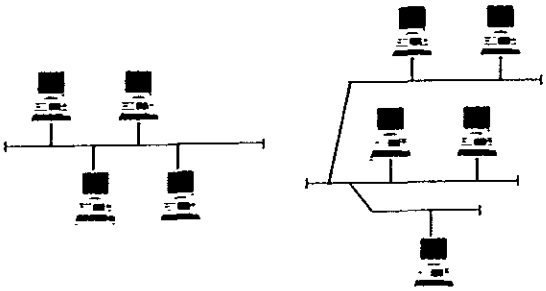

Figura 3.3

3.3 Modelo OSI

El modelo OSI proporciona una estructura conceptual para diseñar redes. Los protocolos son un conjunto de especificaciones y software de control que administra todos o algunos de los siete niveles de el modelo OSI. Muchos de los protocolos extensamente usados hoy en día, TCP/IP por ejemplo, son actualmente conjuntos de protocolos. TCP/IP esta hecho de cuatro protocolos componentes: IP (Internet Protocol), TCP (Transmission Control Protocol), UDP (User Datagram Protocol), y ARP (Address Resolution Protocol) Desde una perspectiva puramente técnica, estos tipos de protocolos son protocol suites Generalmente, protocols y protocol suites son usados indistintamente, en ocasiones se refiere al conjunto de protocolos como un stack de protocolos, donde cada capa está diseñada para un propósito particular y es independiente de lo que pase en las demás capas.

Algunos protocolos, no siguen la estructura del modelo OSI estrictamente, pero el resultado final es el mismo (por ejemplo, comunicación dispositivo a dispositivo).

El modelo OSI describe la actividad de la red con una estructura de 7 capas, cada una con uno o más protocolos asociados.

Las capas de este modelo de referencia son:

7) Aplicación

Conjunto de servicios de comunicación standard y aplicaciones de uso común.

6) Presentación

Se asegura que la información sea recibida de forma que la máquina pueda entenderla.

5) Sesión

Maneja el inicio y fin de conexión entre computadoras.

4) Transporte

Maneja la transferencia de datos y se asegura de que la información sea idéntica.

3) Red

Maneja el direccionamiento y repartición de datos en la red.

2) Enlace

Maneja la transferencia de datos a través del medio de transmisión.

1) Física

Define las características del medio de transmisión.

Las funciones definidas por el modelo OSI son conceptuales y no únicas de un conjunto de protocolos de red en particular.

3.3.1 Protocolo de comunicación TCP/IP

Podemos decir que un protocolo de comunicaciones de red es un conjunto formal de reglas que describen cómo el software y el hardware interaccionan con la red. Para que la red funcione adecuadamente es necesario que la información sea enviada de forma adecuada, debido a que diferentes programas de comunicación por red necesitan interactuar para desarrollar una función de red, se hace necesario entonces una forma homogénea de comunicación a través de la red.

Las principales características de TCP/IP son las siguientes:

- Totalmente independiente del hardware o software de una computadora en particular. Debido a que es ampliamente soportado, TCP/IP es el ideal para comunicar computadoras de diferentes plataformas (tanto de software como de hardware).
- Independiente de la tecnología de red que se utilice. Esto permite integrar redes diferentes, tales como Ethernet, Token Ring, y X.25.
- Un esquema de direccionamiento común. Permite identificar inequívocamente un host dentro de la red, no importando si esta red es de cobertura mundial.

El modelo OSI idealiza el conjunto de protocolos para la comunicación por red, en tanto que, el modelo TCP/IP combina las diferentes capas del modelo OSI para hacer una implementación real. En la tabla 3.1 se presenta este modelo de referencia, en un cuadro comparativo con el modelo OSI.

Capa OSI	Nombre OSI	Capa TCP/IP	Protocolos
5,6,7	Aplicación, sesión, presentación	Aplicación	NFS, telnet, FTP
4	Transporte	Transporte	TCP, UDP
3	Network	Internet	IP, ARP, ICMP
2	Enlace	Enlace	PPP, IEEE802.2
1	Física	Red Física	Ethernet (IEEE 802.3), Token Ring, RS-232, otros.

Tabla 3.1[1]

Red física

Especifica las características del hardware que será utilizado en la red; especifica el standard IEEE 802.3 para la red y el RS-232 para conectores.

[1] Herrera Hidalgo. TCP IP en diferentes plataformas.

Enlace

Identifica el tipo de protocolo de red del paquete, que en este caso será TCP/IP. Provee control de errores.

Internet

Acepta y distribuye paquetes a través de la red. Incluye los protocolos IP, ARP, e ICMP.

Protocolo **IP**

El protocolo IP es responsable de:

- Direccionamiento IP
- Comunicaciones Host a Host. Decide la ruta que el paquete tomará en base a la dirección del host que ha recibido.
- Formato de paquete. IP ensambla paquetes en unidades llamadas datagramas IP.
- Fragmentación. Si un paquete es demasiado largo para su transmisión por la red, IP los divide en paquetes más pequeños. IP en el host receptor se encarga de reconstruir el paquete original.

Protocolo **ARP**

El protocolo de resolución de dirección ARP (Address Resolution Protocol) conceptualmente existe entre la capa de enlace y la capa de internet. Asiste a IP en dirigir datagramas apropiadamente hacia el host receptor cotejando la dirección Ethernet (48 bits) con su dirección IP conocida (32 bits).

Protocolo **ICMP**

El protocolo internet de control de mensajes ICMP (Internet Control Message Protocol) es responsable de detectar errores en la red y reportarlos. Los estados de error que reporta son los siguientes:

- Cuando los paquetes llegan demasiado rápido para ser procesados (Dropped Packets)
- Fallas de conectividad; cuando un host no puede ser alcanzado.

- Redireccionamiento. Indica cuándo utilizar una ruta alternativa para enviar la información.

Capa de transporte

Esta capa se asegura que los paquetes sean recibidos en secuencia y sin errores. Los protocolos involucrados en esta capa son TCP (Transmission Control Protocol) y UDP (User Datagram Protocol).

Protocolo **TCP**

Hace posible que las aplicaciones se comuniquen como si estuvieran físicamente conectados. TCP envía datos de tal forma que parece una transmisión carácter por carácter. Esta transmisión consiste en un inicio de conexión, la transmisión de datos, y un fin de conexión.

Integra un encabezado en los datos transmitidos. Contiene parámetros que ayudan en el proceso de envío y recepción de información.

Por cada paquete enviado, TCP espera una confirmación de recepción para poder continuar la transmisión, es decir, por cada paquete debe recibir un acknowledge que le permita saber que el receptor recibió el paquete sin errores. A este tipo de comunicación se le llama end-to-end connection. Es por esto que TCP se considera un protocolo seguro, orientado a conexión.

Protocolo **UDP**

Provee un servicio de repartición de datagramas. No hace ninguna verificación de que el datagrama fue recibido o enviado, sin errores. Debido a que elimina los procesos de establecer y verificar las conexiones, las aplicaciones que envían pequeñas porciones de información, lo utilizan en lugar de TCP.

Capa de aplicación

La capa de aplicación define los servicios internet standard y las aplicaciones de red que pueden ser utilizadas. Estos servicios trabajan junto con la capa de transporte para enviar y recibir datos. Existen muchos protocolos en esta capa como:

Servicios TCP/IP standard: comandos ftp y telnet.

3.4 LAN Protocols

Aunque no son protocolos en el sentido estricto, ARCnet, Ethernet, y Token Ring[1] son frecuentemente referidos como protocolos de red. La mayoría, si no es que todas, las LANs existentes usan uno de ellos.

ARCnet

ARCnet fue diseñado en los 70's, tiene una base instalada bastante grande. Este sigue una topología estrella. A causa de su bajo volumen de transferencia de 2.5Mbps, los vendedores ya no proporcionan más productos siguiendo éste protocolo.

Ethernet

Ethernet continúa siendo el protocolo LAN más popular. Este puede ser instalado usando cualquier tipo de cable, en una topología estrella o bus. Ethernet tiene un máximo volumen de transferencia de 100 Mbps. Hay una gran cantidad de vendedores de productos y de soporte para esta tecnología. Básicamente, Ethernet maneja los problemas de trafico en red usando un protocolo llamado Carrier Sense Multiple Access with Collision Detection (CSMA/CD). Usando este protocolo, cada nodo sobre una red Ethernet checa para asegurarse de que la red no está transportando tráfico de otro nodo antes de mandar cualquier mensaje. En el evento de colisión (o mensajes de nodos siendo mandados simultáneamente), los adaptadores de Ethernet detectan las colisiones y retrasan las transmisiones en un esfuerzo por reducir las colisiones.

Token Ring

El más costoso de los protocolos de red es Token Ring. Inventado por IBM, este opera a un máximo de 16 Mbps. Las redes Token Ring evitan colisiones al hacer circular un token o un paquete de datos al rededor de la LAN. Este token deberá establecer contacto con un nodo antes de que este pueda mandar un mensaje. Después de que el token regresa a

su punto de emisión, este es purgado, y un nuevo token es generado y circulado. Esto es llamado token passing.

3.5 Dispositivos de Intercomunicación

Los dispositivos de intercomunicación son una solución de hardware-software para pasar mensajes (datos) de una LAN a otra basándose en una cierta cabecera de la información y en criterios de ruteo. A continuación se tratarán en forma superficial los siguientes dispositivos:

- Repeater (Ethernet)
- Bridge
- Router
- Gateway

3.5.1 Repeaters

El repetidor (figura 3.4) proporciona el modo más barato de conectar LANs. Un repetidor toma la señal (data/message) sobre una extensión de cable y la regenera, previniendo perdida de señal (atenuación). Los repetidores son una forma de incrementar la distancia de transmisión máxima al utilizar un tipo de cable. Los repetidores se utilizan en la unión de LANs en el mismo edificio, entre dos departamentos o grupos de trabajo etc..

- El nivel más bajo de union de una red
- Convierte dos segmentos Ethernet en un segmento lógico

[1] Tipos de targetas de red

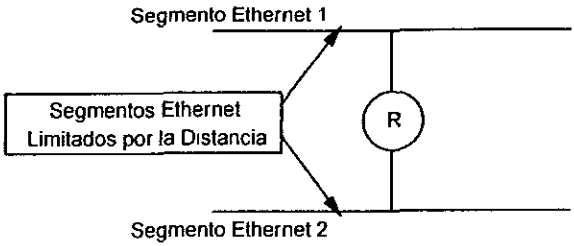

Figura 3.4

3.5.2 Bridges

Usar bridges (figura 3.5) es un método más eficiente de conectar redes que usan repetidores. La figura 3.6 ilustra una WAN que conecta dos sitios con britdges.

- Independiente del protocolo
- Une dos redes separadas
- Reconoce paquetes y direcciones

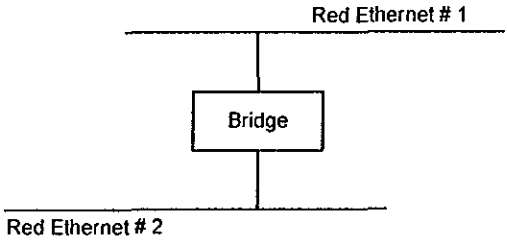

Figura 3.5

Cuando un nodo en una LAN corporativa manda mensajes (datos), eventualmente los paquetes de datos llegan al bridge A. El bridge A mira en la información del que manda o fuente y del recibidor o destino. El entonces observa una tabla (que éste mantiene y constantemente actualiza). Si el recibidor no es un miembro de ésta tabla (en este caso, un nodo sobre la rama Office LAN), el Bridge A envía estos paquetes por cualquier

método de transmisión de una red WAN (línea rentada, dial-up, Frame Relay, ISDN, etc.) a donde se encuentra la LAN remota. En la rama Office LAN, el Bridge B checa si el que recibe es un miembro de su tabla y entonces lo envía. Los Bridges proporcionan intercomunicación segura y rápida entre LANs.

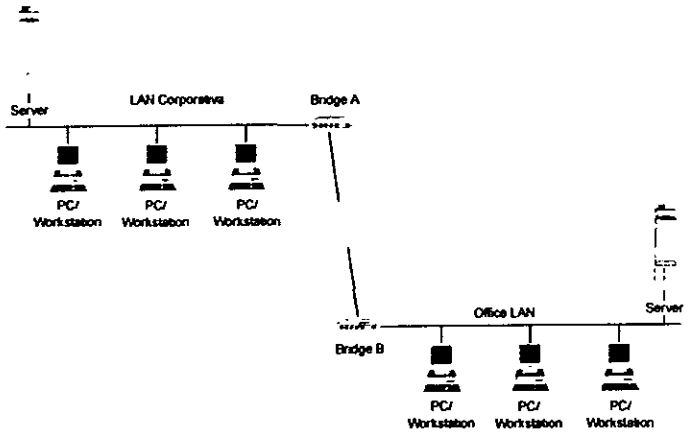

Figura 3.6

3.5.3 Routers

Los Routers (figura 3.7) proporcionan servicios de conexión inteligente entre LANs. Esencialmente, los routers son similares a los bridges en que cada LAN en una WAN necesita al menos un router. Sin embargo, a diferencia de los bridges, los dispositivos de red estan conscientes de los routers en la red. Un router puede leer paquetes de datos direccionados a el y determinar el nivel de prioridad con la cual el necesita pasarlos por la red. Un router puede fragmentar grandes paquetes de datos en unos más pequeños haciendo la transmisión más eficiente. Los routers mantienen sofisticadas tablas de ruteo (routing tables) y las actualizan dinámicamente a través de una WAN. Los routers son también lo suficientemente capaces para encontrar la mejor ruta (por ejemplo, la menos costosa) entre la fuente y el destino en una WAN, de este modo decrecen las cargas de telecomunicación. A causa de la capacidad involucrada, los routers generan más carga

sobre la red que los bridges.

- Dependiente del protocolo
- Une múltiples redes separadas

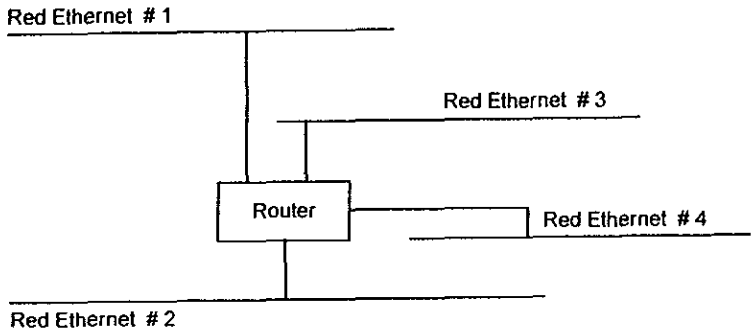

Figura 3.7

3.5.4 Gateways

Los gateway (figura 3.8) proporcionan el grado más alto de inteligencia en dispositivos de conectividad, éstos usualmente operan a nivel de software. Si se supone que se necesita una conexión entre una base de datos en SQL Server residente sobre Windows NT Server y una base de datos DB/2 residiendo sobre un mainframe, se usa un gateway para conseguir esto. Por lo tanto, un gateway es esencialmente un traductor entre diferentes protocolos. En este ejemplo, la traducción sería entre IPX y SNA usando un SNA gateway. Los gateways proporcionan más que una mera conexión; proporcionan un método de comunicación entre dispositivos. Los gateways son invaluables cuando se tienen plataformas heterogéneas de aplicación/dato. Ellos permiten a la aplicación comunicarse con cualquier otra. Los gateways típicamente son difíciles de configurar y mantener.

- Dependiente del protocolo
- Une redes diferentes

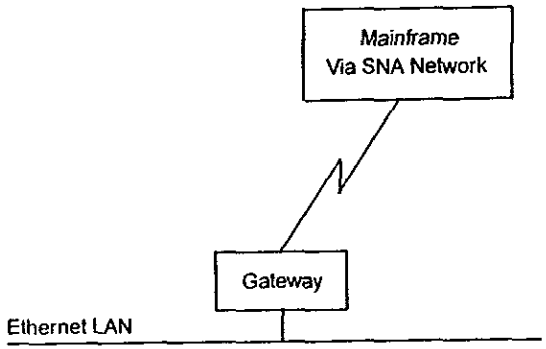

Figura 3.8

3.6 Middleware de conexión a base de datos

Durante el desarrollo de sistemas cliente/servidor, existe la necesidad de ocultar la complejidad de la interacción entre la máquina cliente y la máquina servidora. Esta necesidad ha conducido al desarrollo de un conjunto de productos, nombrados middleware, que proporcionen ésta funcionalidad. El middleware se ha convertido en un termino que abarca todo un rango de diferentes tecnologías de software diseñadas para proporcionar conectividad en un medio ambiente de computadoras distintas. Desafortunadamente, debido al gran número de modos diferentes por medio de los cuales cliente/servidor pueden ser construidos, el middleware es ahora usado para describir todo, desde una conexión básica de SQL (structured query language) usada en un medio ambiente de base de datos hasta un sistema de procesamiento de transacciones como IBM's CICS.

Este capítulo usa la siguiente definición de middleware: software de conectividad que soporta procesamiento distribuido en tiempo de corrida y es usado por desarrolladores para construir software distribuido. Este software proporciona protocolos de comunicación de alto nivel para permitir a aplicaciones completas o procesos dentro de aplicaciones comunicarse a través de una red.

Con la llegada de las redes de área local y la interconexión de las PCs, cada vez más compañías están enfocadas sobre la interconexión de sus bases de datos y sistemas host. En los primeros días, esta interconexión fue manejada por simples gateways. Sin embargo éstos no proporcionan la interacción requerida en el nivel cliente. Las compañías se dieron cuenta de que el software era requerido para situarse entre los sistemas PCs y hosts. Este software necesitó contener las características y funcionalidad que lo hicieran tan seguro y tan robusto como los sistemas host. Esta área es aún la que tiene mayor enfoque de desarrollo en el middleware hoy en día. Aunque al software cliente (particularmente el middleware) no se le ha dado la importancia debida hasta ahora, este hecho está cambiando. La característica del software cliente siempre ha sido, fácil de usar y fácil de instalar. Como un resultado, el middleware se ha convertido en un requerimiento importante en sistemas cliente/servidor proporcionando trabajos de procesamiento distribuido.

Como los sistemas de información van aumentando en complejidad, la necesidad de soluciones middleware genéricas para habilitar interoperabilidad de sistemas y portabilidad de aplicaciones se hace más importante. Los sistemas de información están evolucionando de un mundo donde la simplicidad en los servicios de comunicación era suficiente a uno donde la transparencia de la conexión, facilidad de administración, portabilidad de funciones y lógica de negocios, rápido desarrollo y desplegado de función distribuida para habilitar aplicaciones modulares son los requerimientos para soportar los negocios.

El desafío de los sistemas cliente/servidor es la conectividad de la base de datos, la cual involucra la habilidad para accesar múltiple fuentes de datos heterogéneos desde de una sola aplicación corriendo sobre el cliente. Un segundo desafío es la flexibilidad; la aplicación deberá ser capaz de accesar datos de una variedad de fuentes de datos (DBMSs) sin modificación en la aplicación. Por ejemplo, en un medio ambiente de red, una aplicación puede accesar datos de Foxpro en un pequeño medio ambiente de oficina y de Sybase u Oracle en un medio ambiente más grande (empresarial). Estos desafíos son

acontecimientos diarios para programadores de aplicaciones y para desarrolladores corporativos que intentan proporcionar soluciones a usuarios finales o para migrar datos a nuevas plataformas. Estos desafíos crecen exponencialmente para desarrolladores y personal de soporte conforme el número de fuentes de datos crece y la complejidad de la aplicación final se incrementa.

Los problemas de las base de datos comienzan evidentemente en las diferencias entre la interfaz de programación, los protocolos del DBMS, los lenguajes del DBMS, y los protocolos de red de las fuentes de datos. Aún cuando las fuentes de datos son restringidas a DBMSs relacionales que usan SQL, hay diferencias significativas en la sintaxis y semántica del SQL que deberán de ser resueltas.

Las principales diferencias en la implementación de cada uno de esos componentes son los siguientes:

- Interfaz de programación. Cada DBMS suministra su propia interfaz de programación propietaria. El método de acceso a un DBMS relacional puede ser a través de embedded SQL o un API.
- Protocolo del DBMS. Cada DBMS suministra formatos de datos de uso propietario y métodos de comunicación entre la aplicación y el DBMS. Por ejemplo, hay muchos diferentes modos de delinear el final de un renglón de dato y comienzo del siguiente.
- El lenguaje del DBMS. SQL se ha convertido en el lenguaje de elección para DBMSs relacionales, no obstante aún existen muchas diferencias entre implementaciones SQL. Por ejemplo, una diferencia es el uso de DECODE en Oracle.
- Protocolos de red. Muchos diversos protocolos de LAN y WAN existen en redes hoy en día. Los DBMSs y las aplicaciones deberán coexistir en estos diversos medio ambientes. Por ejemplo, SQL Server usaría DECnet sobre una VAX, TCP/IP sobre UNIX, y Netbeui o SPX/IPX sobre una PC.

Para accesar varios ambientes de bases de datos, un desarrollador de aplicaciones tendrá que aprender a usar cada interfaz de programación suplida por el DBMS, usar

cada SQL proporcionado por el DBMS y asegurarse de que la apropiada interfaz de programación, red y software del DBMS este instalada sobre el sistema cliente. Esta complejidad hace una amplia conectividad de base de datos inaccesible para la mayoría de los desarrolladores y usuarios de hoy en día. Si el número de DBMSs soportado se incrementa, la complejidad se incrementa severamente.

3.6.1 Vias para la conectividad de las bases de datos

Los proveedores de DBMSs y compañias de terceros han intentado abordar el problema de la conectividad de las bases de datos en diferentes modos. Las principales vías han incluido el uso de gateways, una interfaz de programación común, y un protocolo común.

3.6.1.1 Gateway

En una vía gateway, los programadores de aplicaciones usan una interfaz de programación del proveedor, gramática SQL, y el protocolo del DBMS. Un gateway provoca que un DBMS destino se muestre a la aplicación como una copia de el DBMS seleccionado. El gateway traduce y envía peticiones al DBMS y recibe resultados de él. Por ejemplo, aplicaciones que accesan SQL Server pueden también accesar datos DB2 a través de el Micro Decisionware DB2 Gateway. Este producto permite a DBMS DB2 aparecer ante aplicaciones basadas en Windows como un SQL Server DBMS. Cualquier aplicación usando un gateway necesitará un diferente gateway para cada tipo de DBMS que este necesite accesar, tal como DEC Rdb, Informix, Ingres y Oracle.

La via gateway esta limitada por diferencias de estructuras y arquitecturas entre DBMSs, tales como diferencias en catálogos e implementaciones SQL, y la necesidad usual de un gateway para cada DBMS destino.

3.6.1.2 Interfaz Común

En la vía de la interfaz común, una sola interfaz de programación es proporcionada al programador. Esto hace posible proporcionar alguna estandarización en el medio ambiente de desarrollo de aplicaciones de base de datos o interfaz de usuario aún cuando la interfaz fundamental sea diferente para cada DBMS. Esta estandarización es el resultado de la creación de un API standard, un macro lenguaje, o un conjunto de herramientas de usuario para accesar datos y trasladar peticiones y resultados para, y desde, cada DBMS destino. Una interfaz común es usualmente implementada escribiendo un manejador para cada DBMS. El ODBC de Microsoft permite esta vía. La figura 3.9 muestra como esta interfaz ajusta arquitecturalmente dentro del sistema cliente.

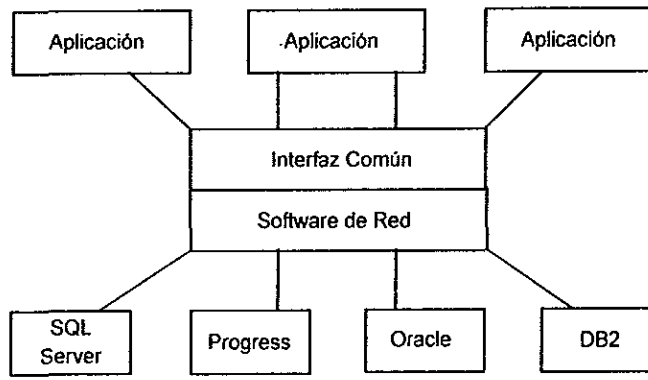

Figura 3.9

3.6.1.3 Protocolo Común

El protocolo del DBMS, la gramática de SQL y los protocolos de red son comunes a todos los DBMSs, así la aplicación puede usar el mismo protocolo y gramática SQL para comunicarse con todos los DBMSs. Ejemplos de este tipo son remote data access (RDA) y distributed relational database architecture (DRDA). RDA es un estándar salido de SGA. DRDA es un DBMS protocol alternativo de IBM.

Interfaces, protocolos y gateways comunes pueden ser combinados. Un protocolo e interfaz común proporcionan un API estándar para desarrolladores así como un solo protocolo para comunicación con todas las bases de datos. Un gateway e interfaz común proporcionan un API estándar para desarrolladores y permite al gateway proporcionar funcionalidad, tal como traducción y conectividad a redes de área amplia (WAN), que de lo contrario necesitarían ser implementados sobre cada estación cliente, no obstante un gateway o protocolo común aún requieren una interfaz común para ocultar complejidades de desarrolladores.

3.6.2 Panorama del middleware

La figura 3.10[1] muestra una vista básica de como una estación de trabajo cliente interactúa con el servidor de base de datos a través de una red.

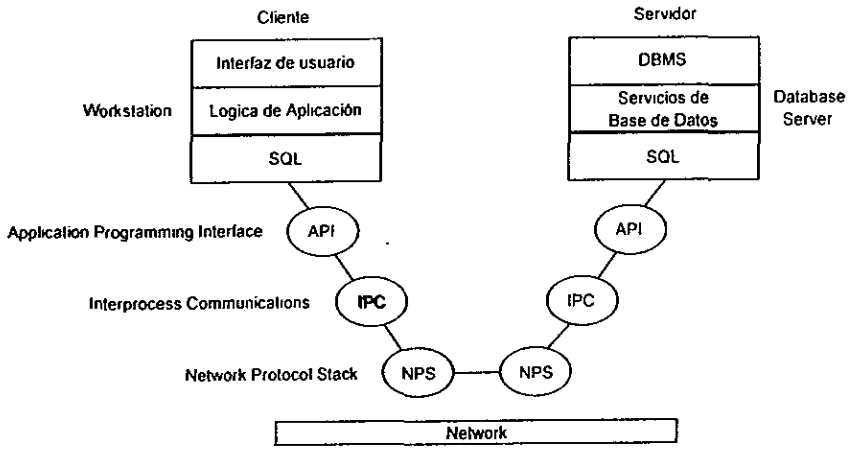

Figura 3.10

Cuando una aplicación en el cliente final requiere datos del servidor, una transacción es mandada desde la lógica de aplicación vía SQL a la red. Esta transacción es pasada a

través de una API (*application programming* interface), un IPC (interprocess communications protocol), y un NPS (network protocol stack) hacia el servidor. La parte del API y el IPC del proceso pueden ser parte del middleware.

La figura 3.11 muestra *donde el middleware está colocado* en el sistema cliente/servidor. Sobre el lado izquierdo del diagrama, una aplicación de negocios se está comunicando con el middleware que a su ves se está comunicándo con un servidor de base de datos sobre el lado derecho del diagrama. Estos sistemas están físicamente separados y pueden ser localizados en cualquier parte.

En algunos casos, el middleware también puede proporcionar el vocabulario del lenguaje de base de datos. Ejemplos de este tipo de middleware incluyen Rumba Database Access y productos ODBC de Microsoft.

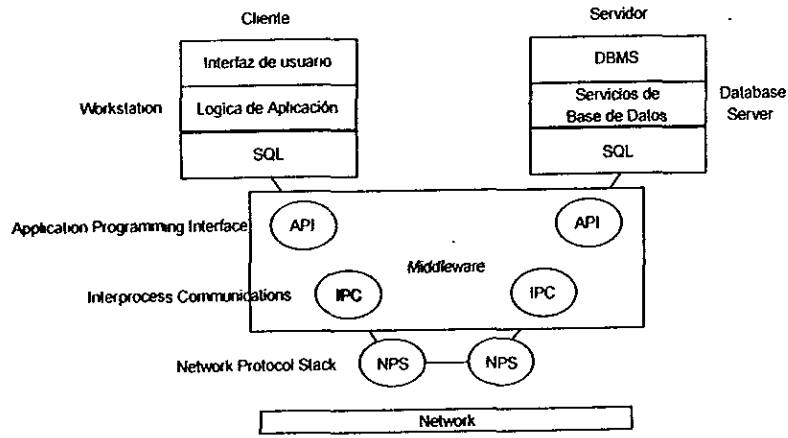

Figura 3.11

3.6.3 Tipo de comunicación del middleware

Los productos middleware, en un alto nivel, usan uno de los tres tipos de comunicación

[1] Neil Jenkins. Client/Server Unleashed.

- Comunicación sincrona orientada a transacciones.
- Comunicación asincrona orientada a lotes. o middleware orientado a mensajes (message-oriented middleware) (MOM)
- Comunicaciones aplicación-servidor.

3.6.3.1 Comunicación Sincrona Orientada a Transacciones

El middleware que usa comunicación sincrona orientada a transacción involucra intercambio de información hacia atrás y hacia adelante entre dos o más programas. Por ejemplo, una PC corriendo ODBC recupera información de la aplicación servidora residente en el host, solicitada por la aplicación. Los aspectos sincronizados de este estilo de comunicación demandan que cada programa ejecute su tarea correctamente; de otro modo, la transacción no será completada.

Productos de este tipo incluyen lo siguiente:

- Productos que proporcionan APIs que permiten a los programas de PC comunicarse con un AS/400 usando APPC se ajustan a este tipo, a causa de que APPC esta orientado a transacciones en modo sincrono.
- Productos que soportan TCP/IP sockets[1] permiten a los programas de PC comunicarse con otros sistemas basados en sockets. Este acceso es similar al de la vía APPC para el AS/400. Las aplicaciones que soportan Winsocks para Microsoft Windows trabajan en este modo.
- Productos que proporcionan APIs, high-level language APIs (HLLAPIs), o extended HLLAPIs (EHLLAPIs) que permiten a los programas de PC comunicación con mainframes a través de un programa intermediario de emulación de terminal son incluidos en el grupo de comunicación sincrona orientada a transacciones. Usando

[1] Socket : mecanismo en el kernel que actúa como una interfaz entre procesos sobre diferentes computadoras. Hay tres tipos de sockets disponibles con TCP/IP para sistemas AViiON : stream sockets, datagram sockets, y raw sockets.

esta tecnología, el programa aplicación se comunica con un programa de emulación de terminal a través de APIs para entrar en la computadora host, y entonces interactuar con la aplicación host como si el programa de la PC fuera una sesión desplegada de usuario. Ejemplos de estos paquetes incluyen Rumba Office de Walldat, productos de Netsoft tales como Netsoft Elite, y Attachmate series.

- Los productos de comunicación orientados a la plataforma Windows de Microsoft que soportan las facilidades de Dynamic Data Exchange (DDE) o el Object Linking and Embedding (OLE) para crear ligas entre información residente en el host (típicamente accesada a través de una sesión de emulación de terminal) y programas nativos de Windows, son también productos de comunicación sincrona orientada a transacciones. Con DDE y OLE, se puede crear una liga entre la información presentada en la pantalla por una aplicación corriendo en el host (host application screen) (la cual es vista a través de un software de emulación de terminal) y un manejador de hoja de cálculo nativo de Windows (tal como Excel o Lotus 1-2-3). Note que ambas aplicaciones involucradas en una conversación DDE u OLE deberán soportar los formatos DDE u OLE. Ambos Rumba y Netsoft soportan estos formatos para el AS/400.

3.6.3.2 Comunicación Asincrona Orientada a Lotes

En el tipo de comunicación asincrona orientada a lotes los mensajes son mandados ya sea uno a la vez o en lotes, sin la espera de una respuesta inmediata (y algunas veces de ninguna respuesta). Por ejemplo, un programa de actualización de la base de datos del servidor usa una cola de datos para mandar subconjuntos de registros actualizados a los programas de la PC que entonces actualizan la base de datos local del cliente. Un programa de PC usa un API de transferencia de archivo para mandar registros de ordenes de comisiones a un programa AS/400 a medida que ellos son introducidos. Este método es comúnmente llamado *message-oriented middleware*.

3.6.3.3 Comunicación Aplicación-Servidor

El middleware puede también ligar una aplicación de negocios con un programa servidor generalizado, que típicamente reside sobre otro sistema. Un servidor de base de datos, un servidor de imágenes, un servidor de vídeo, y otros servidores de propósito general pueden comunicarse con un programa de aplicación a través de una solución middleware.

Los productos en el rango de middleware orientado a servidor incluyen lo siguiente:

- Productos que conforman la especificación de ODBC basada en Windows son middleware orientados a servidor. Bajo esta especificación, los vendedores proporcionan un ODBC driver que, por un lado, proporciona un conjunto consistente de rutinas de acceso orientadas a SQL para usarse por programas que se ejecutan en Windows y, por el otro lado, administra el acceso a la base de datos remota del vendedor.
- Productos de acceso remoto de vendedores específicos y un grupo de soluciones genéricas de acceso remoto basadas en SQL, las cuales ofrecen alternativas para acceso a bases de datos remotas a través de ODBC, son también categorizadas bajo server-oriented middleware. Oracle's Oracle Transparent Gateway es un ejemplo de producto de acceso remoto basado en SQL.
- En el margen del mercado de middleware orientado a servidor es el CICS (Customer Information Control System) el mejor ejemplo de un sistema de procesamiento de transacciones.

3.6.4 ODBC

Open database connectivity (ODBC) es la interfaz estratégica de Microsoft para accesar datos en un medio ambiente distribuido hecho de DBMSs relacionales y no relacionales. ODBC proporciona un modo abierto de accesar datos almacenados en una variedad de bases de datos propietarias de computadoras personales, minicomputadoras y

mainframes, al ser proporcionado por un vendedor neutral. ODBC alivia o aligera la necesidad de los vendedores de software independientes y de los desarrolladores corporativos de tener que aprender múltiples APIs (Application Programming Interface). ODBC proporciona ahora una interfaz de acceso a datos universal. Con ODBC, los desarrolladores de aplicaciones pueden permitir a una aplicación concurrentemente accesar, ver, y modificar datos de múltiples, diversas bases de datos. ODBC ha emergido como la industria estándar para accesar bases de datos en aplicaciones tanto basadas en Windows como en Macintosh.

Los puntos sobresalientes de ODBC en el medio ambiente de desarrollo cliente/servidor son :

- ODBC puede ser proporcionado por un vendedor neutral, permitiendo acceso a DBMSs de múltiples vendedores.
- ODBC es abierto. Trabaja con estándares ANSI, el SQL Access Group (SAG), X/ Open, y numerosos vendedores de software independientes, y se ha ganado un muy amplio consenso sobre su implementación, y es ahora el estándar dominante.
- ODBC es poderoso; éste ofrece capacidades críticas a aplicaciones cliente/servidor de procesamiento de transacciones en línea (OLTP) y a aplicaciones de sistemas de soporte de decisiones (DSS), incluyendo transparencia de tablas de sistema, completo soporte de transacciones, llamadas asíncronas, arreglo de consulta y actualización, un modelo de conexión flexible, y procedimientos almacenados para desempeño estable del SQL.

Los beneficios claves de ODBC son :

- Permite a los usuarios accesar datos en más de una localización de almacenamiento de datos (por ejemplo, más de un servidor) desde dentro de una sola aplicación.
- Permite que los usuarios accesen datos en más de un tipo de DBMS (tal como DB2, Oracle, DEC Rdb, y Progress) desde dentro de una sola aplicación.
- Simplifica el desarrollo de aplicaciones. Es más fácil ahora para los desarrolladores

proporcionar acceso a datos en múltiples, concurrentes DBMSs.

- Es una interfaz de programación para aplicaciones (API) portable, que habilita la misma interfaz y tecnología de acceso para poder accesar otra plataforma.
- Aisla las aplicaciones de cambios a la red y versiones de DBMS. Modificaciones al modo de transporte en la red, servidores, y DBMSs no afectarán a las aplicaciones ODBC actuales.
- Promueve el uso de SQL, el lenguaje estándar para DBMSs, como está definido en el estándar ANSI 1989.
- Permite a las corporaciones proteger sus inversiones en DBMSs existentes y proteger las habilidades adquiridas en el DBMS por los desarrolladores. ODBC permite a las corporaciones continuar usando manejadores existentes de DBMSs mientras se mueven a sistemas basados en cliente/servidor.

La solución ODBC

ODBC maneja los problemas de conectividad usando la vía de interfaz común delineada previamente. Los desarrolladores de aplicaciones pueden usar un API para accesar todas las fuentes de datos. ODBC está basado en una especificación CLI (Call Level Interface), la cual fue desarrollada por un consorcio de más de 40 compañías (miembros de el SQL Access Group y otros) y tiene amplio soporte de proveedores de aplicaciones y bases de datos. El resultado es un único API que proporciona toda la funcionalidad que los desarrolladores de aplicaciones necesitan y una arquitectura que desarrolladores de bases de datos requieren para asegurar interoperabilidad. Como resultado, un grupo muy grande de aplicaciones usa ODBC.

3.6.4.1 Como trabaja ODBC

ODBC describe un API. Cada aplicación usa el mismo código como está definido en la especificación del API, para poder hacer peticiones a muchos tipos de fuentes de datos a través de drivers de DBMSs específicos. Un administrador del driver permanece entre la aplicación y el driver. En Windows, el Driver Manager y los drivers están implementados como dynamic-link libraries (DLL). La figura 3.12 muestra como trabaja el ODBC driver

para Windows 3.1 y Windows para trabajo en grupos, Windows 95 y Windows NT trabajan en forma similar, pero como éstos son para sistemas operativos de 32-bit, usan una versión de 32-bit de ODBC.

ARQUITECTURA ODBC

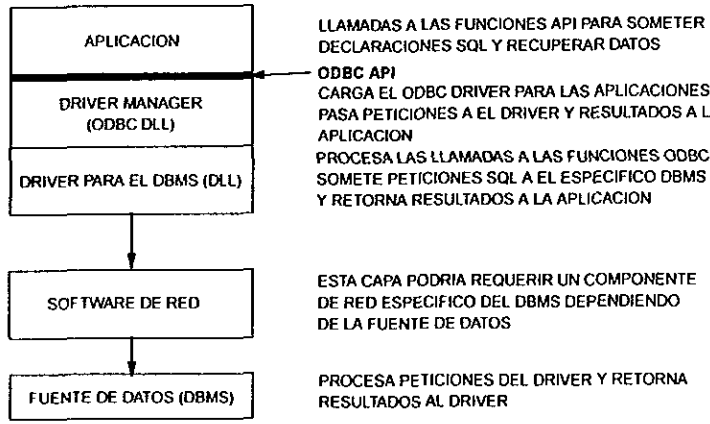

Figura 3.12

La aplicación llama a las funciones ODBC para conectarse a las fuentes de datos, ya sea local o remotamente, para mandar y recibir datos, y para desconectar. El Driver Manager proporciona información para una aplicación, tal como una lista de fuentes de datos disponibles, carga drivers dinámicamente conforme ellos son necesitados y, proporciona argumentos y chequeo de transición de estados. El driver, desarrollado separadamente de la aplicación, permanece entre la aplicación y la red. El driver procesa llamadas a funciones ODBC, administra el total del intercambio entre una aplicación y un específico DBMS, y realiza la traducción de la sintaxis estándar de SQL al SQL nativo de la fuente de datos destino. Todas las transacciones SQL son responsabilidad del desarrollador del driver.

Las aplicaciones no están limitadas a comunicarse a través de un driver. Una sola aplicación puede hacer múltiples conexiones, cada una a través de un diferente driver, o

múltiples conexiones a fuentes similares a través de un solo driver. Para accesar un nuevo DBMS, un usuario o un administrador instala un driver para el DBMS. Este es un gran beneficio para usuarios finales y proporciona un significativo ahorro para organizaciones IS (Information System) en soporte y costos de desarrollo.

3.6.4.2 Como beneficia ODBC a usuarios finales

Los usuarios finales no trabajan directamente con el ODBC API; su configuración y setup es manejada durante la instalación. Los usuarios se benefician en diferentes modos cuando usan aplicaciones escritas con ODBC :

- Los usuarios pueden seleccionar una fuente de datos de una lista de nombres de las fuentes de datos disponibles o proporcionar el nombre de una fuente de datos, en un modo consistente, a través de las aplicaciones.
- Los usuarios pueden someter peticiones de acceso a datos en la gramática estándar SQL sin importar el DBMS destino.
- Los usuarios pueden accesar diferentes DBMSs usando aplicaciones de escritorio que le son familiares. Cuando los usuarios necesitan accesar datos sobre una nueva plataforma, ellos tendrán un nivel común, de capacidad funcional, mientras ellos accesen los nuevos datos con una herramienta que les sea familiar. Similarmente, si los datos se mueven a una plataforma diferente, solo la definición ODBC necesita cambiar; la aplicación puede permanecer igual.

La figura 3.13 muestra como un usuario puede estar corriendo dos aplicaciones accesando tres diferentes fuentes de datos a través de ODBC. Las tres fuentes de datos podrían estar sobre tres sistemas completamente diferentes en otra parte de la red, no obstante ellos son ligados dentro de la aplicación en el escritorio del cliente.

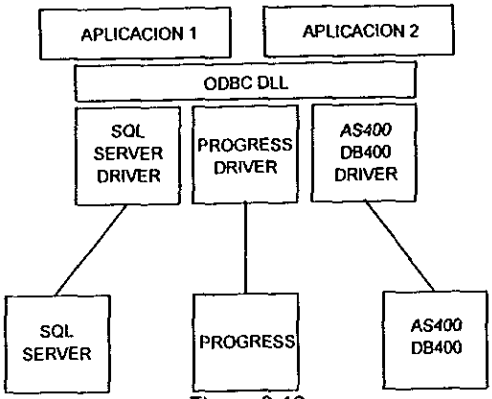

Figura 3.13

3.6.4.3 Que significa ODBC para desarrolladores de aplicaciones

ODBC define una gramática estándar de SQL y un conjunto de llamadas a funciones que son conocidas como core grammar y core functions, respectivamente. Si un desarrollador de una aplicación elige usar solamente el core functions, el no necesita escribir ningún código adicional para verificar capacidades especificas de un driver.

Usando el core functions, una aplicación puede hacer lo siguiente :

- Establecer una conexión con una fuente de datos, ejecutar declaraciones SQL, y recuperar resultados.
- Recibir mensajes estándar de errores.
- Proporcionar una interfaz estándar de registro (logon) a el usuario final para accesar la fuente de datos.
- Usar un conjunto de tipos de datos estándar definidos por ODBC.
- Usar una gramática estándar de SQL definida por ODBC.

ODBC también define una gramática SQL extendida y un conjunto de funciones extendidas para proporcionar a los desarrolladores un modo estándar para explotar capacidades avanzadas de un DBMS. En adición a las características precedentes, ODBC tiene extensiones que proporcionan mejoras en el desempeño e incrementan el poder a

través de lo siguiente :

- Tipos de datos tales como date, time, timestamp, y binary.
- Una gramática SQL estándar para outer joins, y procedures.
- Ejecución asíncrona.
- Un modo estándar para que desarrolladores de aplicaciones averigüen que capacidades proporciona un driver y una fuente de datos.

Finalmente, ODBC soporta el uso de gramáticas SQL de DBMSs específicos, permitiendo a las aplicaciones explotar las capacidades de un particular DBMS. ODBC tiene disponible un vasto número de bases de datos soportadas incluyendo IBM DB2/6000, IBM DB2/400, ORACLE, SQL Server, dBASE, Interbase, DEC Rdb, Microsoft Access, IBM DB2, y Progress.

ODBC es una herramienta para proporcionar a usuarios finales el acceso a datos almacenados en una amplia variedad de bases de datos sin requerimientos de SQL o programación. Esto proporciona una excelente interfaz para aplicaciones cliente/servidor que son portables a través de múltiples bases de datos y aún a través de múltiples plataformas cliente. La extensa aceptación de ODBC en la industria (prácticamente cada vendedor de base de datos tiene un ODBC driver para su producto) ha resultado en muchos drivers, programas, y herramientas que ofrecen un prominente arreglo de características y capacidades.

CAPITULO 4

EL PAPEL DE UNA BASE DE DATOS Y EL SOPORTE DE UNA HERRAMIENTA ADECUADA

4.1 Introducción.

Los RDBMS deben manejar eficientemente los requerimientos demandantes de las aplicaciones en línea ya que hoy en día éstas automatizan las operaciones de misión crítica, que hacen a las organizaciones más competitivas. Estas aplicaciones afectan el flujo de utilidades diario, control operacional, y la eficiencia de costos de una empresa.

Los RDBMS ayudan a manejar las aplicaciones en línea al poder implementar una arquitectura cliente/servidor en la cual las funciones de interfaz del usuario son claramente separadas de las funciones de administración de los datos y las funciones de transacción. Los *clientes* y *servidores* pueden correr sobre la misma máquina o transparentemente a través de una red.

Los RDBMS proporcionan un poder y funcionalidad para manejar los requerimientos demandantes de una aplicación en línea de gran escala, en un medio ambiente muti-usuario por el suministro de:

- Alto desempeño escalable : El RDBMS proporciona un alto numero de transacciones por segundo y bajo tiempo de respuesta, y permite el escalamiento de aplicaciones a el más eficaz costo de hardware.

- Integridad y seguridad llevada a cabo en el servidor: El RDBMS da fuerza a la integridad y seguridad de los datos al mantenerla centralizada, en el servidor, independiente de la aplicación.

- Alta disponibilidad de la aplicación : El RDBMS mantiene aplicaciones corriendo

durante las actividades de mantenimiento del sistema, y soporta recuperación rápida y tolerancia a fallas basada en software.

- DBMS abierto y distribuido : El RDBMS soporta la distribución de aplicaciones y bases de datos sobre redes de estaciones de trabajo y procesadores heterogéneos.

4.2 Características

4.2.1 Alto Desempeño Escalable

Las aplicaciones en línea requieren de un alto número de transacciones por segundo, tiempo de respuesta rápido, y la habilidad para mantener niveles de desempeño a medida que los sistemas crecen - docenas de cientos de usuarios accesando grandes bases de datos de multi-gigabyte sobre sistemas single- o multi-processor.

Este tipo de desempeño consigue tiempos de respuesta de subsegundos en un medio ambiente de grandes aplicaciones, y esto reduce costos de hardware y software al manejar más usuarios sobre una misma máquina.

4.2.2 Estándares de desempeño

Los estándares de desempeño están basados en:

- multi-threaded server.
- Lenguaje procedural/SQL y procedimientos almacenados.

Multi-Threaded Server. Servidor de proceso único. Esto es, no importa cuantos usuarios estén accesando el RDBMS, solo un proceso es consumido. En esencia, éste es un sistema operativo de base de datos.

Algunos RDBMS como SYBASE y ORACLE manejan funciones multi-usuario

tradicionalmente asociadas con el sistema operativo, tales como scheduling, task switching, disk caching, indexing, transaction processing, and locking. Esto elimina casi el total de los ineficientes manejos del sistema operativo que plaga (molesta) otros sistemas manejadores de bases de datos relacionales.

Procedimientos almacenados y lenguaje procedural/SQL. Los roles de negocios, lógica de transacción e integridad referencial son definidos como procedimientos almacenados. Estos conjuntos precompilados de declaraciones escritas en lenguaje procedural, extensión de SQL, proporciona comandos para iniciar/perpetuar/deshacer (begin/commit/rollback) transacciones, control de flujo, declaración de variables, emisión de mensajes, e iniciar una acción después de un tiempo definido de espera.

Una vez definidos, los procedimientos almacenados son compilados y almacenados en el diccionario de datos. En tiempo de corrida, la aplicación hace una llamada a un procedimiento y pasa parámetros. El RDBMS entonces ejecuta el procedimiento almacenado sin recompilarlo. Si cualquiera de los objetos referenciados por un procedimiento almacenado es modificado, el procedimiento es automáticamente recompilado para incorporar la nueva información. Los procedimientos almacenados son rehusables y compartibles.

Los procedimientos almacenados ofrecen dos beneficios claves de desempeño. Primero, la comunicación a través de la red es significativamente reducida a causa de que una llamada a un procedimiento reemplaza muchas declaraciones SQL. Segundo, puesto que los procedimientos almacenados están ya compilados, ellos son procesados cinco veces más rápido que un comando individual SQL.

4.2.3 Integridad y Seguridad Realizada en el Servidor

La integridad y la seguridad son requerimientos críticos de aplicaciones en línea que implican accesos de datos por docenas de cientos de usuarios. La integridad se refiere a la precisión y consistencia de los datos. La seguridad significa autorización y control de

acceso a datos. El RDBMS atiende estos requerimientos al hacer cumplir integridad y seguridad centralmente, en lugar de ser llevada a cabo en cada aplicación que accesa los datos.

Beneficios:

- Alta seguridad. Al centralizar el control de los datos, éste no puede ser esquivado por ninguna aplicación, usuario o herramienta.
- Costo de dasarrollo reducido. El ciclo de desarrollo es significativamente acortado a causa de que la lógica de integridad y seguridad, y las reglas de negocios son codificadas solamente una vez y almacenadas en la base de datos, en vez de ser escritas una y otra vez para cada aplicación.
- Costo de mantenimiento reducido. El mantenimiento también se simplifica, a causa de que las aplicaciones están aisladas de cambios de integridad. Cuando es necesario cambiar una regla de negocios, esta tiene que ser modificada solo en el Servidor.
- Incremento de la seguridad en la base de datos. Roles y transacciones, así como tablas, vistas, y campos, están protegidos de acceso no autorizado

Los RDBMSs usan una variedad de mecanismos para hacer cumplir integridad y seguridad:

- La integridad en un campo individual se cumple por tipos de datos suplidos por el sistema y definidos por el usuario, valores nulos, defaults y reglas definidas en el RDBMS.
- La integridad referencial, que es la capacidad de sincronizar los contenidos de múltiples tablas basadas en valores llave, es ejecutado con un tipo especial de procedimiento almacenado llamado trigger.

4.2.4 Alta disponibilidad

El RDBMS debe proporcionar una alta disponibilidad - las 24 horas del dia, los siete días

de la semana.

Beneficios:

- Protege a las organizaciones contra costosas interrupciones de negocios
- Reduce costos de horas extraordinarias a causa de que las operaciones de mantenimiento pueden ser realizadas durante horas de trabajo normales.
- Protege a las organizaciones contra fallas de medios de almacenamiento y CPU.

Los RDBMS dan alta disponibilidad de la aplicación al proporcionar mantenimiento en línea, recuperación rápida, y tolerancia a fallas basada en software.

- Mantenimiento en línea. La mayoría de los sistemas manajadores de bases de datos requieren que los usuarios estén fuera de línea para actividades de mantenimiento tales como respaldos (backups), diagnósticos, cambios en la base de datos, y cambios en la integridad. En un RDBMS, estas operaciones de mantenimiento son manejadas en línea, mientras las aplicaciones continúan corriendo.
- Recuperación rápida. Los métodos de registro (logging) físico del RDBMS y el registro de transacciones escritas hacia adelante (write-ahead) garantizan una rápida y correcta recuperación. El RDBMS también permite un tiempo máximo de recuperación a ser especificado. Sobre las bases de esta figuran las especificaciones del usuario, los RDBMS dinámicamente calculan el intervalo apropiado del punto de chequeo (checkpoint), dependiendo de la carga de trabajo sobre el sistema. En cada punto de chequeo, el RDBMS escribe todas las páginas con polvo (páginas que han sido cambiadas) a el disco.
- Tolerancia a fallas basada en software. Los RDBMS soportan disco espejo para el registro de transacciones - protegiendo contra perdida de cualquier transacción perpetrada (commited transaction) - o la base de datos misma - garantizando operación continua a pesar de fallas de disco. Algunos sistemas manejadores de bases de datos confían en hardware tolerante a fallas para soportar estas capacidades.

4.2.5 DBMS Abierto y Distribuido

Los RDBMS deben proporcionar una arquitectura abierta y las capacidades de distribución para integrar información, eficazmente, a través de un medio ambiente multi-vendedor distribuido.

Arquitectura abierta. La arquitectura cliente/servidor de los RDBMS proporciona las bases para ambas, independencia de hardware e independencia de software.

Los RDBMS proporcionan productos para ligar datos que residen sobre diferentes tipos de máquinas, cubriendo un amplio rango de hardware y soportando medio ambientes heterogéneos de administradores de datos distribuidos.

Las interfaces para Client/Server permiten comunicación basada en estándares entre software heterogéneo y medio ambientes de datos heterogéneo. Un lenguaje de programación interfaz (programming language interface) para usar por aplicaciones cliente, proporciona la flexibilidad de utilizar una variedad de paquetes front-end o aplicaciones para accesar y actualizar datos. Un lenguaje de programación interfaz para aplicaciones Open Server, proporciona la flexibilidad para accesar y actualizar una variedad de datos fuente.

Administración de base de datos distribuida. Dos tipos de configuraciones distribuidas son posibles: acceso distribuido y base de datos distribuida. El acceso distribuido se caracteriza por un Servidor (RDBMS) central soportando aplicaciones corriendo sobre diferentes máquinas.

La base de datos distribuida se caracteriza por una distribución de los datos mismos sobre diversas máquinas en una red. En ésta configuración una aplicación deberá ser capaz de accesar datos de múltiples RDBMS en la misma transacción

Los RDBMS proporcionan recuperación de datos distribuida y actualización distribuida.

Las actualizaciones distribuidas son más complejas a causa de que la consistencia de los datos deberá ser mantenida a través de múltiples máquinas, aún si un error ocurre durante la ejecución de una transacción.

Un RDBMS puede soporta dos características para asegurar la consistencia de datos para actualizaciones distribuidas: perpetuar de dos fases (two-phase commit), el cual garantiza la consistencia de datos en caso de falla, y comunicación servidor-servidor, lo que permite a los RDBMS comunicarse mutuamente a través de stored procedures. La comunicación servidor-servidor permite a las aplicaciones tener una interfaz con sólo un RDBMS, aún cuando la aplicación requiere datos de más de un servidor. Las aplicaciones no muy grandes tienen que saber donde está cada dato.

En un medio ambiente distribuido, es crítico mantener el control mientras la bases de datos y aplicaciones están siendo actualizadas. Un RDBMS proporciona control distribuido para mantener la integridad de las bases de datos distribuidas mientras preserva autonomía de sitio.

4.3 Transacciones en línea

Desempeño y Manipulación de datos para Transacciones En Linea

El RDBMS acopla el desempeño con la manipulación de datos y la administración de datos a través de su lenguaje de base de datos (lenguaje procedural/SQL). El lenguaje procedural/SQL es una extensión del SQL que ofrece muchas mejoras sobre éste.

4.3.1 Desempeño de las transacciones en línea

Para aplicaciones en línea, la clave de su desempeño es throughput (el número de transacciones por segundo) y tiempo de respuesta (la espera experimentada por el usuario) en transacciones corriendo por un gran número de usuarios contra grandes bases de datos. Una típica aplicación en línea de producción se caracteriza por ejecuciones repetidas de transacciones predefinidas por un gran número de usuarios

simultáneos.

Los sistemas en línea requieren actualizaciones - típicamente inserciones, borrados, actualizaciones, o cambios a múltiples tablas. Puesto que la integridad y disponibilidad de los datos en un sistema en línea es obligatorio, el sistema deberá ser capas de recuperar el dato en un estado consistente en caso de falla. Esto requiere que una transacción sea tratada como una unidad y ser completamente grabada (commited) a la base de datos o, en caso de falla, completamente eliminada de la base de datos.

La mayoría de los sistemas administradores de bases de datos usan alguna forma de locking y logging para proporcionar esta funcionalidad. Mientras una transacción se ejecuta, el sistema manejador de la base de datos deberá asegurar el registro que está en uso, para asegurar que el dato no sea cambiado por otro usuario mientras la transacción está en progreso.

4.3.2 Throughput, Tiempo de Respuesta, y Numero de Usuarios

En la mayoría de los sistemas de administración de base de datos relacionales, el número de transacciones por segundo (throughput) alcanzan un máximo con un modesto número de usuarios y entonces desciende rápidamente cuando se añaden más usuarios. Un RDBMS debe permanecer constante el número de transacciones por segundo en el máximo nivel aún cuando se incorpore una gran cantidad de usuarios adicionales.

En sistemas convencionales, el tiempo de respuesta aumenta en una razón más grande que la lineal, creciendo excesivamente cuando el número de usuarios aumenta. En ORACE y SYBASE el aumento en el tiempo de respuesta es lineal con el número de usuarios.
La figura 4.1 ilustra la diferencia entre SQL Server de SYBASE y otros sistemas:

Arquitectura Cliente/Servidor Multi-threaded. Una de las razones de que el desempeño de muchos sistemas convencionales de administración de bases de datos se

degrade cuando el número de usuarios se incrementa es su excesiva confianza sobre el sistema operativo, empeorando con factores como:

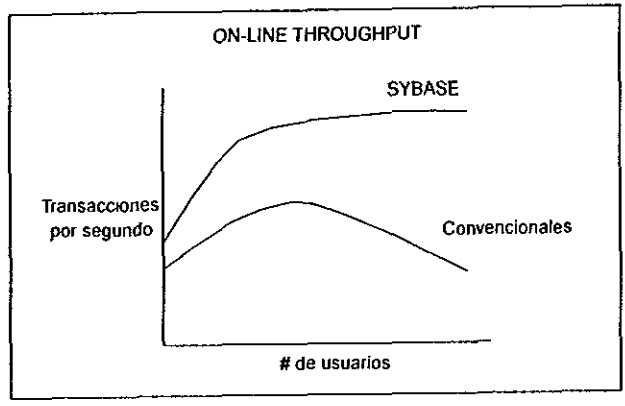

Figura 4.1: Desempeño Multi-Usuario[1]

- incremento en el número de procesos
- incremento en requerimientos de memoria
- incremento en el número de seguros (locks)

Para los sistemas convencionales de administración de bases de datos, el gasto involucrado en el procesamiento de una transacción para un usuario adicional puede crecer exponencialmente. Esto es particularmente verdadero para sistemas con una arquitectura de proceso por usuario (un proceso por cada usuario), donde procesos sombra son requeridos para cada usuario. En tales sistemas, el gasto de paging, process swapping, system level locking, y otras funciones del sistema operativo pueden virtualmente llevar al sistema a un alto en términos del número de transacciones por segundo.

Hay RDBMSs que, por otro lado, usan una arquitectura multi-threaded - RDBMS corre como un proceso individual y maneja agendas (scheduling), intercambio de tareas (task

switching), cacheo de disco (disk caching), locking (seguros), y transaction processing (procesamiento de transacciones) sin ningún proceso adicional del sistema operativo.

En SYBASE, cada usuario adicional de SQL Server requiere únicamente una memoria adicional de 34k, liberando más memoria para disk caching (cacheo de disco) y procedure caching (cacheo de procedimiento). A causa de que SQL Server no necesita crear procesos adicionales de sistema operativo para cada usuario, la impredesible fluctuación del tiempo de respuesta de aplicaciones de requerimientos de computo variable u otros trabajos de recursos intensivos son eliminados.

La arquitectura cliente/servidor también significa que las aplicaciones podrán residir sobre estaciones de trabajo cliente separadas, descargando de sus tareas al procesador de base de datos. Esto permite al procesamiento cooperativo tomar lugar entre la aplicación cliente sobre una computadora y el servidor sobre otra, y esta es una de las razones por las que un RDBMS puede manejar la carga de trabajo de cientos de usuarios.

La congestión de entrada/salida. Otro congestionamiento potencial para sistemas de administración de bases de datos relacionales corriendo aplicaciones en línea es entrada/salida (input/output). El sistema administrador de la base de datos deberá minimizar el número de lectura y escritura física actual (o real), y amortizar el costo de I/O sobre muchos usuarios.

Por ejemplo, SQL Server usa buffers compartidos para datos así como para procedimientos, permitiendo a un usuario beneficiarse del uso de otro del mismo dato Las páginas son escritas a disco sólo cuando se necesita espacio en el buffer para nuevos registros o cuando un checkpoint es tomado. Antes que escribir cada registro a disco cuando este cambia, la consistencia de datos esta garantizada por write-ahead logging a un registro de transacción compartido. Aún estas escrituras son compartidas, así que si varios usuarios perpetran (commit) en aproximadamente el mismo tiempo, una sola, escritura compartida ocurre para todos los usuarios. De hecho, en algunas situaciones la

---

[1] SYBASE SQL Server Technical Overview

cantidad de I/O por usuario en realidad decrece cuando el número de usuarios aumenta.

Los sistemas convencionales podrían escribir paginas a disco cada vez que un registro cambia sobre una de esas páginas. Algunos sistemas manejan registros (logging) por usuario. Otros sistemas requieren hacer copias de páginas para manejar locking.

Locking. Locking (seguro) es otra área que típicamente reduce eficiencia en un medio ambiente en linea. Mientras la mayoría de los sistemas convencionales se fían en el operating system's lock manager, el cual puede ser ineficiente, Hay DBMSs que manejan su propio locking por medio de una tabla residente en memoria. Obteniendo un lock tomando solo unos pocos microsegundos y no requiere I/O o mensajes de tráfico

El nivel de locking es también un factor. Algunos sistemas usan row level locking (aseguramiento a nivel renglón). Sin embargo, si cualquier cambio a un renglón es realizado de modo que cambia su espacio o localización, el lock debe ser subido a nivel de página. Otros sistemas suben a un table level lock durante el tiempo en que la actualización esta siendo realizada. Cuando este es el caso, todas las transacciones están en efecto single-threaded, lo cual es claramente una situación no aceptable en un medio ambiente en linea.

SQL Server de SYBASE usa page-level locking, y solo sube a table locks cuando éste detecta que la mayoria de la tabla será afectada, como en el caso de una carga voluminosa.

Stored Procedures. Los procedimientos almacenados son un factor clave en la eficiencia de un RDBMS. Estos procedimientos precompilados reducen trafico en red y eliminan los costosos gastos de compilar transacciones cada vez que ellas son invocadas. Típicamente un procedimiento almacenado puede ser procesado en una quinta parte del tiempo que toma procesar un sòlo comando embedded SQL.

El DBMS, además, optimiza el uso de procedimientos con un cache de procedimiento

compartido, así que múltiples usuarios pueden hacer uso del mismo código compartido.

4.3.3 Mejoras al SQL (lenguaje procedural/SQL)

El lenguaje de base de datos ha sido diseñado para mejorar el poder de SQL y para minimizar - si no eliminar - las ocasiones en las cuales el usuario deberá recurrir a un lenguaje de programación afín de poder realizar la tarea deseada. El lenguaje de base de datos va más allá que el SQL ANSI (American National Standards Institute) estándar y la muy comercial versión de SQL.

Las mejoras implementadas a SQL varían en los diferentes RDBMSs, aquí se presentan las mejoras más importantes implementadas en Transact-SQL de SYBASE

La cláusula COMPUTE
La cláusula COMPUTE es una importante extensión de el lenguaje de base de datos que es usada con la función de agregado de renglón (SUM, MAX, MIN, AVG, y COUNT) para calcular valores sumarios.

El resultado de una consulta que incluye una cláusula COMPUTE es desplegado con renglones de detalles y sumario, y se parece a un reporte que la mayoría de los sistemas de administración de bases de datos pueden producir solamente con un generador de reportes. COMPUTE despliega valores sumarios como renglones adicionales en los resultados, en vez de como nuevas columnas.

Ejemplo: un query que calcula el precio promedio de diferentes tipos de libros de cocina, y sus resultados:

```
select title_id, type, price
from titles
where type like "%cook"
order by type, price
```

```
compute avg(price) by type

title_id    type          price
---------   ----------    ---------------------------
MC3021      mod_cook                              2.99
MC2222      mod_cook                             19.99
                          avg
                          ---------------------------
                                                11.49

title_id    type          price
---------   ----------    ---------------------------
TC4203      trad_cook                            11.95
TC7777      trad_cook                            14.99
TC3218      trad_cook                            20.95
                          avg
                          ---------------------------
                                                15.96
```

Lenguaje de control de flujo.

El lenguaje de base de datos proporciona un lenguaje de control de flujo que puede ser usado como parte de cualquier declaración SQL o bloque.

Estas construcciones son disponibles:

BEGIN-END
BREAK
CONTINUE
DECLARE
GOTO label
IF-ELSE

PRINT
RAISERROR
RETURN
WAITFOR
WHILE

Se pueden definir variables locales con DECLARE y asignarles valores. Además diversas variables globales predefinidas son suplidas por el sistema.

Stored Procedures

Una de las más importantes extensiones del lenguaje de base de datos al estándar SQL es la habilidad para crear procedimientos almacenados. En un procedimiento almacenado se puede mezclar declaraciones SQL con construcciones procedurales.

Un procedimiento almacenado puede contener declaraciones SQL como:

- Declaraciones de Lenguaje de Manipulación de Datos (DML) (INSERT, UPDATE y DELETE)
- Declaraciones de procesamiento de transacciones (COMMIT, ROLLBACK)

Así como construcciones procedurales como pueden ser:

- Declaraciones de control de flujo como:
  - IF-ELSE
  - GOTO
- Declaraciones de repetición como:
  - WHILE
  - FOR
- Declaraciones de Asignación como:
  - X:=Y+Z

Triggers

Un trigger es un clase especial de stored procedure que es usado para proteger integridad referencial. Los triggers ejecutan reglas a cerca de las relaciones entre datos en diferentes tablas.

Rules and Defaults

Rules y defaults definen restricciones de integridad que entran en juego durante la introducción y modificación de datos.

4.4 Características de Integridad, Consistencia y Restauración

El RDBMS proporciona una variedad de herramientas y mecanismos que garantizan la integridad, consistencia, y restauración de los datos. En éste subcapítulo se estudiarán:

- defaults y rules que proporcionan restricciones de dominio.
- triggers que protegen la integridad referencial.
- administración de transacciones en el RDBMS que aseguran consistencia de datos.
- administración de transacciones y mecanismos de recuperación automática del RDBMS.
- backup dinámicos y mecanismos de carga.

4.4.1 Integridad de Datos y Restricciones de Dominio

En general la integridad de datos se refiere a la precisión y consistencia de los datos en la base de datos. En un RDBMS, las restricciones de dominio (domain constraints) se refieren a la precisión y consistencia de datos en columnas particulares o en todas las columnas de un tipo de dato definido por el usuario. Un dominio es el conjunto de valores conexos lógicamente de los cuales el valor en una columna particular puede ser obtenido (o derivado). El RDBMS proporciona mecanismos que garantizan que cualquier valor introducido es un miembro de este conjunto.

En el nivel más básico, el RDBMS mantiene integridad de datos al asegurarse de que un valor que está siendo introducido es del tipo de dato correcto. Para definir otros dominios específicos, el lenguaje de base de datos permite crear **rules**.

Otra característica del lenguaje de base de datos es que te permite crear **dafaults**, valores presentes que son automáticamente introducidos por el RDBMS cuando el usuario no proporciona valores explícitos. Los defaults son convenientes ya que pueden identificar la razón particular de que no hay valor explícito en una columna. Los defaults también ayudan a mantener consistente una base de datos, ya que todas las columnas en las cuales el valor es desconocido o será determinado igual a los demás.

4.4.2 Rules (reglas)

En el mundo de la administración de las bases de datos, una regla especifica que es lo que se está o no autorizado a introducir en una columna particular o en cualquier columna con un tipo de dato determinado, definido por el usuario. Las reglas (Rules) son restricciones de integridad que van más allá de aquellas implícitas por un tipo de dato de una columna. Ellas pueden ser atribuidas a una columna específica, a varias columnas especificas, o a un específico tipo de dato definido por el usuario.

Cada vez que un usuario introduce un valor (con una declaración INSERT o un UPDATE) el RDBMS lo checa contra la regla más reciente que a sido ligada a la columna especificada. El dato introducido anterior a la creación y ligado de una regla no es checado.

En SYBASE las reglas son creadas con el comando CREATE RULE, y entonces ligada a una columna o tipo de dato definido por el usuario con el procedimiento de sistema *sp_bindrule*.

4.4.3 Defaults

Un default es un valor que el RDBMS inserta dentro de una columna si el usuario no introduce explícitamente uno.

En un sistema de administración de base de datos relacional, cada data element (que es, una particular columna en un particular renglón) deberá contener algún valor - aun cuando el valor sea NULL. Algunas columnas no aceptan el valor nulo; por lo cual algún otro valor deberá ser introducido explícitamente por el usuario, o el RDBMS deberá introducir un default definido por el usuario.

El RDBMS permite a los usuarios definir valores default y ligar un default a una particular columna, a un número de columnas, o a todas las columnas de un especificado tipo de dato definido por el usuario. Por ejemplo, se puede crear un default que tenga el valor "???" o el valor "se llenara más tarde", y entonces instruir al RDBMS para introducirlo en una columna particular si un usuario no hace una introducción.

Defaults y Valores Null. Si se especifica NOT NULL cuando se crea una columna y no se crea un default para ella, el RDBMS producirá un mensaje de error siempre que alguien inserte un registro y no especifique un valor en la columna.

La siguiente tabla ilustra la relación entre la existencia de un default y la definición de una columna como NULL o NOT NULL.

	NO ENTRY, NO DEFAULT	NO ENTRY, DEFAULT	ENTER NULL, NO DEFAULT	ENTER NULL, DEFAULT
NULL	null	default	null	null
NOT NULL	error	default	error	error

4.4.4 Integridad Referencial y Triggers

Un trigger es una clase especial de procedimiento almacenado que entra en efecto cuando se inserta, borra o actualiza un dato en una tabla especificada. Los triggers pueden ayudar a mantener la integridad referencial del dato manteniendo consistencia entre datos relacionados lógicamente en diferentes tablas.

La principal ventaja de los triggers es que ellos son automáticos - ellos trabajan sin importar que causó la modificación del dato; la introducción de un dato por un oficinista o la acción de una aplicación. Un trigger es especifico para una o más de las operaciones de modificación de datos UPDATE, INSERT, o DELETE. El trigger es ejecutado una vez por declaración SQL - éste descarga (o dispara) inmediatamente después de que la declaración de modificación del dato es completada. El trigger y la declaración, la cual lo dispara, son tratadas como una sola transacción que puede ser deshecha (rolled back) desde dentro del trigger. Si un error es detectado, la transacción entera será deshecha (rolled back).

Los triggers son más útiles en situaciones como:

- Los triggers pueden hacer cambios en cascada a través de tablas relacionadas en la base de datos. Por ejemplo, un delete trigger en la columna *title_id* de la tabla *titles* puede causar un correspondiente borrado de renglones iguales en otras tablas, usando la columna *title_id* como una llave única para localizar renglones en las tablas *titleauthor*, *sales*, y *roysched*.
- Los triggers pueden desaprobar o deshacer (roll back) cambios que violen la integridad referencial, cancelando la transacción de modificación de datos intentada.
  Un trigger podría entrar en efecto cuando se intenta insertar una llave foránea (foreign key) que no es igual a su llave primaria (primary key). Por ejemplo, se puede crear un insert trigger sobre *titleauthor.title_id* que rolled back un insert si el nuevo valor no es igual a algún valor en *titles.title_id*.
- Los triggers pueden hacer cumplir restricciones mucho más complejas que aquellas

definidas con rules. A diferencia de los rules, los triggers pueden referenciar columnas u objetos de base de datos. Por ejemplo, un trigger puede roll back updates que intentan incrementar un precio de un libro por más de el 1% del anticipo.

- Los triggers también pueden ejecutar simples análisis what-if: un trigger puede comparar el estado de una tabla antes y después de una modificación de dato (data modification), y tomar acción(es) basada(s) en la comparación.

4.4.5 Definiendo un Trigger

Se crea un trigger especificando la tabla y los comandos de modificación de dato que deberán disparar o activar el trigger. Entonces se especifica la acción o acciones que el trigger podrá tomar.

Ejemplo: Este trigger imprime un mensaje cada vez que alguien intenta insertar, borrar, o actualizar un dato en la tabla *titles*:

```
create trigger t1
on titles
for insert, update, delete
as
print "Ahora modifica la tabla titleauthor del mismo modo."
```

4.4.6 Usando Triggers para Mantener Integridad Referencial

La integridad referencial está coordinada a través de el uso de llaves primarias y foráneas.

La llave primaria es la columna o combinación de columnas que identifican un renglón de forma única. Éste deberá siempre ser no nulo y tener un índice único. Una tabla con una llave primaria es elegible para ligarse con llaves foráneas (foreign keys) en otras tablas. La llave primaria de la tabla puede ser considerada como la tabla maestra (master table) en

una relación maestro - detalle.

Por ejemplo, la columna *title_id* es la llave primaria (primary key) de la tabla *titles*. Ésta identifica de forma única los libros en *titles*, y puede ser asociada con *title_id* en *titleauthor*, *sales*, y *roysched*. La tabla *titles* es la tabla maestra en relación a *titleauthor*, *sales*, y *roysched*.

La foreign key es una columna o combinación de columnas cuyos valores deben ser iguales a la primary key. Una foreign key no tiene que ser única (foreign key son frecuentes en una relación muchos a uno contra una primary key). Los valores de foreign key deberán ser copias de los valores primary key - ningún valor en una foreign key deberá existir a menos que el mismo valor exista en la primary key. Las tablas con foreign keys son frecuentemente llamadas detalle (detail) o tablas dependientes de la tabla maestra.

La columna title_id en titleauthor, sales, y roysched son foreign keys; estas tablas son llamadas tablas detalles (detail tables).

4.4.7 Como Trabajan los Triggers

Los triggers de integridad referencial observan (o cuidan) los valores de foreign keys en línea con aquellos en primary keys. Cuando una modificación de datos afecta una columna llave, los triggers comparan los nuevos valores de la columna para relacionar llaves usando tablas de trabajo temporales llamadas **trigger test tables**. Cuando los triggers se ejecutan, basan sus comparaciones en los datos que están temporalmente almacenados en una o ambas de ellas. También se usan trigger test tables para establecer condiciones para trigger actions. No se puede alterar directamente el dato en las trigger test tables, pero se pueden usar las tablas en declaraciones SELECT para detectar los efectos de un INSERT, UPDATE o DELETE.

Las dos trigger test tables son la tabla *deleted* y la tabla *inserted*.

- La tabla *deleted* almacena copias de los renglones afectados durante las declaraciones DELETE y UPDATE. Durante la ejecución de una declaración DELETE o UPDATE, los renglones son removidos de la trigger table y transferidos a la *deleted* table. La tabla *deleted* y la trigger table ordinariamente no tiene renglones en común.
- La tabla *inserted* almacena copias de los renglones afectados durante declaraciones INSERT y UPDATE. Durante un INSERT o un UPDATE, nuevos renglones son añadidos a *inserted* y al trigger table al mismo tiempo. Los renglones en *inserted* son copias de los nuevos renglones en la trigger table.
- Un UPDATE es un delete seguido por un insert. Los renglones viejos son copiados primero a la tabla *deleted*, entonces los nuevos renglones son copiados a la trigger table y a la *inserted* table.

La figura 4.2 muestra la condición de las trigger test tables durante un INSERT, un DELETE, y un UPDATE.

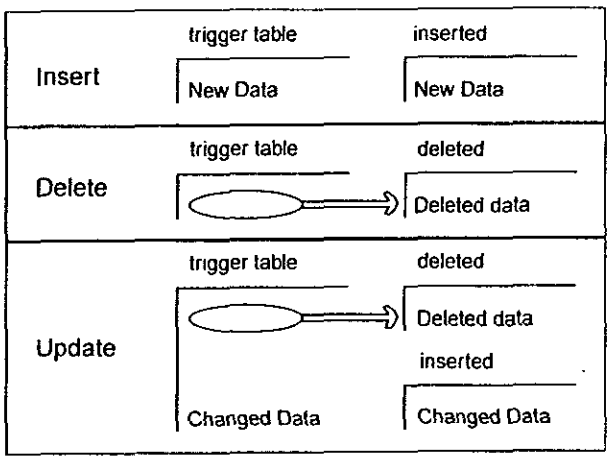

Figura 4.2 Trigger Test Tables durante INSERT, DELETE, o UPDATE[1]

Ejemplo de un Delete Trigger. Cuando se borra un renglón primary key, se deberá

[1] SYBASE SQL Server Technical Overview.

borrar los correspondientes renglones foreign key en las tablas dependientes. Esto preserva integridad referencial al asegurar que los renglones detalle (detail rows) son removidos cuando su renglón maestro (master row) es borrado. Si esto no se hace, se podría finalizar con una base de datos que tiene detail rows que pueden no ser recuperados o identificados. El trigger deberá ejecutar un borrado en cascada (cascading delete).

Ejemplo. Cuando una declaración DELETE se ejecuta sobre la tabla *titles*, uno o más renglones dejan la tabla *titles* y son añadidos a la tabla *deleted*. Un trigger puede revisar las tablas dependientes - *titleauthor*, *sales* y *roysched* - para ver si ellas tienen renglones con un *title_id* que sean iguales a los *title_id*(s) que fueron removidos de *titles* y almacenados en el la tabla *deleted*. Si el trigger encuentra cualquier número de renglones, este los remueve.

```
Create trigger delcascadetrig
on titles
for delete
as

delete titleauthor
from titleauthor, deleted
where titleauthor.title_id = deleted.title_id
    /* Remueve el renglón de titleauthor
    ** que es igual a los renglones borrados de titles. */

delete sales
from sales, deleted
where sales.title_id = deleted.title_id
    /* Remueve los renglones de sales
    ** que son iguales a los renglones borrados de titles. */
```

```
delete roysched
from roysched, deleted
where roysched.title_id = deleted.title_id
    /* Remueve los renglones de roysched
    ** que son iguales a los renglones borrados de titles. */
```

En la practica, se podrá encontrar que se necesita mantener algunos de los renglones detalles, para propósitos históricos (para revisar cuantas ventas fueron realizadas sobre titulos descontinuados mientras estos eran activos) o a causa de transacciones sobre los renglones detalles (detail rows) que aún no son terminadas. Un trigger bien escrito deberá tomar en consideración estos factores.

Ejemplo de un Update Trigger. A causa de que una primary key es el único identificador para su renglón y para foreign key rows en otras tablas, un intento de actualizar una primary key deberá ser tomado con mucho cuidado. En este caso, se necesita proteger la integridad referencial por rolling back el update a menos que las condiciones especificadas sean cumplidas.

En general, es sabido que para prohibir cualquier cambio de edición a un primary key, por ejemplo, se revocan todos los permisos sobre la columna. Pero si se necesita prohibir updates solo bajo ciertas sircunstancias, se usa un trigger.

Este trigger previene actualizaciones para titles.title_id en el fin de semana. La cláusula IF UPDATE en stopupdatetrig permite enfocarse sobre una particular columna, titles.title_id. Modificaciones al dato en la columna causan que el trigger entre en acción, pero cambios a los datos en otras columnas no lo hacen. Cuando éste trigger detecta una actualización que viola las condiciones del trigger, este cancela el update e imprime un mensaje.

```
Create trigger stopupdatetrig
on titles
for update
```

```
as
if update (title_id)
    and datename (dw, getdate())
    in ("Sábado", "Domingo")
begin
    rollback transaction
    print "No se permite jugar con"
    print "primary keys el fin de semana!"
end
    /* Si se realiza un intento por cambiar
    titles.title_id el Sábado o Domingo,
    se cancela el update. */
```

4.4.8 Consideraciones Multi-Row

Cada trigger dispara sólo una vez por query. Si la acción del trigger depende del número de renglones que una modificación de dato afecta, el trigger deberá ser evaluado para modificaciones de dato multirenglón y tomar la acción apropiada. Uno de los modos más comunes para verificar milti-row insert, delete, o update es con la variable @@rowcount (@@rowcount es proporcionada en SQL Server de SYBASE), la cual almacena el número de renglones afectados por la última operación de modificación de datos.

Consideraciones multi-row son particularmente importantes cuando la función de un trigger es calcular automáticamente valores sumarios.

4.4.9 Triggers Condicionales

Los triggers examinados hasta este momento se han dado en cada declaración de modificación de dato en conjunto. Si un registro de un insert de cuatro registros es inaceptable el insert completo es inaceptable y la transacción es rolled back.

Este es el modo como los triggers trabajan. Un trigger descarta o acepta cada transacción de modificación de datos en conjunto.

Sin embargo, no se tiene que roll back el total de las modificaciones de datos simplemente a causa de que alguno de ellos es inaceptable. Usando un subquery correlacionado en un trigger se puede forzar al trigger para examinar los renglones modificados uno por uno. El trigger puede entonces tomar diferentes acciones sobre diferentes renglones.

4.4.10 Triggers Anidados

Los triggers se pueden anidar - en otras palabras, un trigger puede llamar a otro trigger. Un trigger puede hacer cambios a una tabla sobre la cual hay otro trigger que puede disparar el segundo trigger. Este a su vez puede disparar un tercer trigger, etcétera.

Los triggers pueden anidar hasta una profundidad de 16 niveles (en SQL Server de SYBASE). Si ningún trigger en la cadena termina la transacción se presenta un ciclo infinito, el nivel de anidación es excedido y el trigger aborta.

Se pueden usar triggers anidados para ejecutar funciones útiles tales como almacenar una copia backup de renglones afectados por un previo trigger.

Un trigger no puede llamarse a si mismo recursivamente - esto es, un update trigger no puede llamarse a si mismo en respuesta a una segunda actualización a la misma tabla dentro del trigger. Por ejemplo, si un update trigger sobre una columna de una tabla resulta en una actualización a otra columna, el update trigger dispara solo una vez, y no repetidamente.

4.4.11 Transacciones y Recuperación

Una transacción es un mecanismo para asegurar que un conjunto de una o más

declaraciones SQL es tratado como una única unidad de trabajo. El RDBMS maneja automáticamente todos los comandos de modificación de datos, incluyendo peticiones de cambio de un solo paso, como transacciones.

Además, los usuarios pueden agrupar un conjunto de declaraciones SQL dentro de una transacción definida por el usuario con los comandos de comienzo de la transacción y perpetración de la transacción (BEGIN TRANsaction y COMMIT TRANsaction en SYBASE).

Las transacciones permiten al RDBMS garantizar:

- **Consistency** (consistencia). Consultas y peticiones de cambio simultáneos no pueden chocar entre sí y los usuarios nunca ven u operan sobre datos que son parte de un posible cambio (aún no se verifica que los datos no alteren la integridad de la base de datos)
- **Recovery** (restauración). En caso de fallar el sistema, la base de datos es recuperada completa y automáticamente.

4.4.12 Transacciones y Consistencia

En un medio ambiente multi usuario, El RDBMS debe evitar que queries simultáneos y peticiones de modificación de datos interfieran con algún otro. Esto es importante a causa de que si el dato siendo procesado por un query puede ser cambiado por una actualización de otro usuario mientras el query esta corriendo, los resultados podrían ser ambiguos.

El RDBMS automáticamente establece el nivel apropiado de locking para cada transacción.

Los comandos de inicio de transacción y perpetración de transacción permiten a los usuarios instruir al RDBMS para procesar cualquier número de declaraciones de lenguaje

de base de datos como una única unidad. El comando para deshacer la transacción (ROLLBACK TRANsaction en SYBASE) permite a los usuarios deshacer transacciones hacia atrás hasta su comienzo o hacia tras hasta un savepoint, dentro de la transacción, que ha sido definido con el comando correspondiente (por ejemplo SAVE TRANsaction de SYBASE).

Los comandos de begin/commit/rollback transaction dan al usuario control sobre el manejo de la transacción. Las transacciones definidas por el usuario también mejoran el desempeño, a causa de que el system overhead (los gastos generales del sistema) es incurrido solo una vez por un conjunto de transacciones, en lugar de una vez por cada comando individual.

4.4.13 Transacciones y Restauración Automática

Una transacción es una unidad de trabajo y una unidad de recuperación. El hecho de que el RDBMS maneje peticiones de cambio de un solo paso como transacciones significa que la base de datos puede ser recuperada completamente en caso de una falla de sistema o medio.

Las fallas de sistema son manejadas por la recuperación automática del RDBMS. Los mecanismos de recuperación automática son ejecutados cada ves que el RDBMS es reiniciado. Recuperación Automática asegura que todas las transacciones completadas antes de que el sistema dejara de funcionar sean escritas completamente al dispositivo de base de datos, y que todas las transacciones no completadas antes de un crash sean removidas (rolled back).

El mecanismo de recuperación no automático, el cual protege de fallas de medio, implica respaldar las bases de datos y los registros de transacciones. A causa de que éstos mecanismos no son automáticos, el DBA es el responsable de hacer backups y de cargarlos en caso de falla de medio.

El tiempo de recuperación automática del RDBMS es medido en segundos y minutos, y los usuarios pueden especificar el tiempo de recuperación máximo aceptable.

Registro de Transacciones. Cada cambio a la base de datos, ya sea que ésta sea el resultado de un solo SQL update statement (una transacción definida por el sistema) o un conjunto agrupado de declaraciones SQL (una transacción definida por el usuario), ésta es automáticamente registrada en una tabla de sistema (*syslogs en SQL Server de SYBASE).* Esta tabla es llamada transaction log.

Cuando una transacción comienza, un evento de inicio de la transacción (BEGIN TRANsaction) es registrado en el log. Cada ves que una declaración de modificación de dato es recibida, esta es grabada en el log.

El cambio es siempre registrado en el log antes de que cualquier cambio sea realizado en la base de datos misma. Este tipo de log, llamado write-ahead log, asegura que la base de datos pueda ser recuperada completamente en caso de una falla.

Para recuperarse de una falla, las transacciones que estaban en progreso pero todavía no perpetuadas (committed) en el momento de la falla deberán ser desechas (a causa de que una transacción parcial no es un cambio correcto). Las transacciones completadas se deberán rehacer si no hay garantía de que hayan sido escritas al dispositivo de base de datos.

4.4.14 Respaldo y Restauración

El dump dinámico del RDBMS permite a la base de datos y al transaction log ser respaldados mientras se continúa usando la base de datos. El realizar los procesos de respaldo de forma rápida y fácil alienta el habito de hacer frecuentes respaldos (backups) - una consideración importante, puesto que su frecuencia determina la cantidad de trabajo que puede perderse si una falla ocurre.

En caso de falla de medio, se pueden recuperar las bases de datos si - y solo si - se han estado haciendo regulares backups de las bases de datos y sus transaction logs. Estos mecanismos de recuperación no automática dependen completamente del uso regular de los comandos DUMP.

El propietario de cada base de datos o el Administrador del Sistema es el responsable de hacer los respaldos de la base de datos y su transaction log con los comandos DUMP, y de cargarlos en caso de falla con los comandos LOAD. Una vez que los comandos apropiados LOAD son ejecutados. El RDBMS maneja todos los aspectos de los procesos de recuperación.

El RDBMS también controla automáticamente el checkpoint interval (el punto en el cual se garantiza que todas las paginas de datos que han sido cambiadas se han escrito al dispositivo de base de datos). Los usuarios pueden forzar un checkpoint si es necesario con el comando CHECKPOINT.

CONCLUSIONES

Las aplicaciones Cliente/Servidor varian dependiendo de su complejidad. La situación más simple puede ser una aplicación en donde la presentación, reglas de negocio y reglas de dato son codificadas dentro de un solo programa y la información se almacena en un archivo de base de datos localizado en un servidor. Dicha aplicación puede ser desarrollada por una persona aplicando programación estructurada. Una aplicación complicada que involucre múltiples entornos de operación y desarrollo, en donde varios sistemas operativos conviven en una misma red compartiendo o distribuyendo amplios volúmenes de información entre cientos de equipos, necesita ser desarrollada entre un equipo de personas en donde la tarea que realizará cada una de ellas dependerá del tipo y nivel de conocimientos de coordinación, análisis, programación y comunicaciones. En este entorno una sola persona no puede realizarlo todo.

La ventaja de la programación orientada a objetos y basada en objetos, es que permite la reutilización de código, ya sea de botones dentro de un mismo programa o de servicios que fueron desarrollados para otras aplicaciones, con lo cual se reducen los tiempos y costos de una aplicación. Como lo indica el modelo de servicios expuesto en el capítulo 2 ; pero en el mundo real la implementación no es tan fácil, por ejemplo las aplicaciones objeto que fungen como servidores de negocio no corren en sistemas UNIX, a pesar de ser utilizado ampliamente en aplicaciones de misión crítica.

La arquitectura three-tier no es apropiada en todas las situaciones porque podría ser excesiva para un departamento o un grupo de trabajo. Igualmente, ésta tampoco podría ser apropiada para aplicaciones de soporte de decisiones caracterizadas por recuperación de datos y consulta, sin operaciones transaccionales.

La elección del RDBMS dependerá del tipo de aplicación, ya que un DBMS de gran capacidad podria ser demasiado para las funciones de un departamento y viceversa. Los DBMS como Oracle proporcionan capacidades para establecer reglas de dato y reglas de negocio, lo cual incrementa el desempeño de la aplicación y reduce su tiempo de desarrollo al poder utilizar reglas ya existentes. Por otro lado, los servicios de datos no son únicamente servidores DBMS. Hay organizaciones que accesan otros servicios de datos tales como el legado de las aplicaciones transaccionales del mainframe, Internet, servidores de imágenes, servidores de bases de datos de objetos. Estos servicios de

datos podrían proveer sus propios APIs, los cuales podrían no ser una interfaz OLE (por ejemplo WinSock para Internet, o APPC para aplicaciones mainframe).

Al realizar este trabajo se observo que los fundamentos computacionales y sobre todo los de análisis aprendidos durante la carrera son útiles independientemente de si la aplicación es o no compleja, además como la computación es algo que está en constante evolución al abordar los temas hay que estar dispuestos al cambio ya que lo que hoy consideramos como lo mejor mañana puede ser obsoleto o ineficiente y esto se debe a que las aplicaciones se van haciendo cada vez más complejas, otro aspecto que se observo al realizar el trabajo fue que a veces los programadores analistas subestiman el trabajo de los administradores de sistemas (servidores) y de los administradores de base de datos, en una ocasión un líder de proyecto iba a solicitar privilegios sobre un producto instalado en un servidor institucional, cuando se le observo que ese tipo de privilegios no se le podían dar por la forma en que dicho producto estaba instalado y por razones de seguridad ella contesto que a ella tenían que darle ese tipo de servicio, fue a solicitar dichos privilegios y no se los dieron. Considero que el trabajo puede ser útil para los estudiantes de Matemáticas Aplicadas y Computación y sobre todo para los de Sistemas Computacionales que deseen realizar sistemas de información.

BIBILIOGRAFIA

Articulos

- A Client Server Overview
  Deborah Hess Senior Analyst Datapro
  UNIX Open '95 México, July 11, 1995

- ¿Qué es cliente-servidor?
  Gerardo Marín
  Personal Computing México, Noviembre 1995

- Three-Tier Client/Server Arrives
  Craig Goren
  Visual Basic Programmer's Journal, November 1995

Libros

- Client/Server Unleashed
  Neil Jenkins, et al
  SAMS, 1996

- TCP/IP en diferentes plataformas
  Herrera Hidalgo
  Fideicomiso SEP UNAM, Mayo 1995

Manuales

- Local Area Network
  Bronson Purdy
  Technology Training S.A. de C.V., October 1991

- Managing TCP/IP on the DG/UX System
  AViiON Product Line
  Data General Corporation, 1993

- Oracle7 Server for UNIX Administrator's
  Reference Guide
  Oracle Corporation, May 23, 1994

- SYBASE SQL Server
  Technical Overview
  Sybase, Inc.

GLOSARIO

API

Application Programming Interface. Interfaz de programación para aplicaciones.

BACK-END

Aplicación o procesos servidores con los cuales interactúa la aplicación front-end, la aplicación back-end corre independientemente de si la aplicación front-end está o no corriendo.

BRIDGE

Un dispositivo que copia paquetes de una red a otra del mismo tipo. Usualmente los bridges operan en el nivel físico de la red. Los bridges difieren de los repeaters a causa de que los bridges almacenan y reenvían paquetes completos , mientras que los repeaters reenvían señales eléctricas.

CUI

Character User Interface. Interfaz en modo caracter con la que el usuario interactúa.

DBA

DataBase Administrator. Persona encargada de administrar el RDBMS.

DBMS

Database Management System, fue creado para proporcionar un mecanismo de almacenamiento de datos compartido.

DDE

Dynamic Data Exchange. Un mecanismo que automatiza los procesos de copiar y pegar datos de una aplicación a otra en Microsoft Windows.

DDL

Data Definition Language. Una declaración DDL es una declaración SQL como

create table, que crea, actualiza o elimina objetos de la base de datos (como tablas, vistas, etc.).

DML

Data Manipulation Language. Una declaración DML es una declaración SQL como INSERT, UPDATE y DELETE que manipulan los datos.

FRONT-END

Aplicación con la que el usuario interactúa directamente para solicitar servicios, datos o procesamiento de otra aplicación.

GATEWAY

Una computadora (o, en muchos casos un equipo de propósito especial) que convierte de un protocolo a otro.

GUI

Graphical User Interface. Interfaz gráfica con la que el usuario interactúa.

HOST

Una computadora que está configurada para compartir recursos con otras computadoras en una red.

LAN

Local Area Network. Una red en una área pequeña o medio ambiente común, como puede ser dentro de un edificio.

MIDDLEWARE de conexión a base de datos

Software de conectividad que soporta procesamiento distribuido en tiempo de corrida.

OCX

Ole Control eXtensions La siguiente generación de los VBX. Es un control personalizado de 32 bits basado en OLE.

ODBC

Open DataBase Conectivity. Interfaz estratégica de Microsoft para accesar datos en un medio ambiente heterogéneo de DBMSs relacionales y no relacionales.

OLE

Object Linking and Embeding. Un método para proporcionar interoperabilidad entre aplicaciones en Microsoft Windows. Crea un medio ambiente en el cual las aplicaciones pueden compartir datos, donde los datos pueden ser considerados como objetos. OLE Automation permite controlar una aplicación desde otra aplicación a través de código.

RDBMS

Relational Database Management System (Sistema Administrador de Bases de Datos Relacionales). Sistema para almacenamiento de datos que separa la representación interna de la forma en que el dato es accesado y representado lógicamente.

REPEATER

Un dispositivo que copia cada bit de un paquete desde un segmento de una red a otro. Un repeater es como un amplificador. Este incrementa la longitud física de la red.

ROUTER

Un dispositivo (una computadora o equipo especial) que reenvía paquetes de un tipo de protocolo particular (por ejemplo IP) de una red a otra. Es posible usar una computadora como un router siempre que ésta tenga más de una interfaz de red y su software sea capas de reenviar datagrams.

SQL

Structured Query Language. Interfaz estándar para almacenar y consultar información en una base de datos relacional.

STORED PROCEDURES

Lenguaje procedural/SQL que permite mezclar declaraciones SQL con construcciones procedurales.

TRIGGERS

Es una clase especial de procedimiento almacenado que entra en operación cuando se inserta, borra o actualiza un dato.

VBX

Visual Basic eXtensions. Controles visuales que proporcionan retro alimentación visual durante el proceso de diseño.

WAN

Una red que se extiende a lo largo de una área extensa.