24021
24

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
"A C A T L A N"

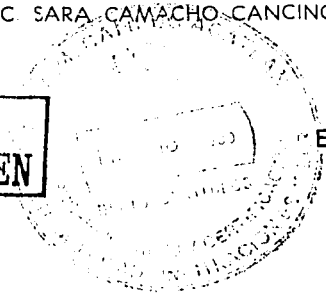
DESARROLLO DE UN PROTOTIPO DE REPLICACION DE BASES DE DATOS DISTRIBUIDAS, UN ENFOQUE TECNICO.

T E S I S

QUE PARA OBTENER EL TITULO DE:
LICENCIADO EN MATEMATICAS APLICADAS Y COMPUTACION
P R E S E N T A :
DANIEL LEAL LARIS

Acompañada de un disco de 3½

ASESOR: LIC. SARA CAMACHO CANCINO



ENERO 2003

A

A mi Madre:

Martha Laris Peralta

Gracias por estar siempre a mi lado, cuando estuve solo
Por quererme y alentarme, cuando quise claudicar
Por tu amor y cariño, soy el hombre que ves ahora
Este triunfo considéralo tuyo mamá.

A mis Hermanas:

Patricia:
Ejemplo eres de la determinación.

Ruth:
El mejor regalo fue tu bondad y comprensión.

Alicia:
Tu alma noble me enseño a ser humilde.

Graciela:
Aprecio mucho que creas en mí.

Marisela:
Gracias por enseñarme con tu ejemplo a no claudicar nunca.

A mis sobrinas:

Casandra: Gracias por ser feliz y disfrutar lo máximo de tu vida, valóralo y sé así siempre.

Alison: Gracias por tu sonrisa e inocencia, deseo de corazón que sea infinita.

Un agradecimiento muy especial:

Mauricio Méndez Carrión: Gracias por creer en mí y darme una oportunidad, cuando nadie más lo hizo.

Francisco Hernández Vázquez: Gracias por darme confianza y la oportunidad de demostrar lo mejor de mi.

A mis amigos:

Yanet Alejandra Hernández Tabiel : Por escucharme cuando necesite tu consejo.

Genaro Abisaí Becerra Pedraza : Por ser mi razón e inteligencia cuando estuve perdido.

Daniel Braulio De La Riva Medina : Por tu confianza y tus sabios consejos.

Miguel Padilla Gamero : Por tu confianza y sincera amistad.

A mi asesora: M. Sara Camacho Cancino

Por sus valiosas aportaciones.

A toda la gente que esta dispuesta a cumplir sus sueños ante cualquier adversidad.

A mi Universidad.

C

ÍNDICE

Introducción

Durante los días de Huelga en la UNAM, en el año 2000, la consultoría en la cual trabajaba implantó un sistema de administración para los aeropuertos de la zona Norte del país entre los que se incluían los Aeropuertos de Mazatlán, Acapulco, Chihuahua, Culiacán y Monterrey, en una primera etapa, el sistema se conectaba a una Base de Datos local (ubicada físicamente en cada aeropuerto) y se pretendía que replicara información hacia una Base de Datos central (Corporativo) y esta a su vez que replicara básicamente catálogos para el control del sistema; de esta manera se tenía un control prácticamente en tiempo real de la situación de los aeropuertos: contable, de recursos humanos, programación de vuelos, líneas aéreas, control de combustibles. Técnicamente hablando se usaba una Base de Datos de Sybase comercialmente conocida como ASE por sus siglas en inglés Adaptive Server Enterprise y un sistema de replicación también de Sybase conocido como Replication Server, al ser parte del staff técnico de la empresa consultora en la cual trabajaba se me asigno la tarea de especializarme en el software de replicación para posteriormente inicializar los sistemas.

En el transcurso del proyecto nacieron inquietudes respecto a la inversión gigantesca que sé estaba realizando en el proyecto, y es entonces cuando surge la idea de crear un software que replique información, creado en un lenguaje de interfaz gráfica a un bajo costo.

Esta es una de las razones por la cual se propone una alternativa de Replicación de Datos abierta, el concepto abierto se refiere a poder aplicarse a cualquier tipo de manejador de Base de Datos que soporte la interfaz ODBC, y la razón principal de esta tesis es la de demostrar que se puede realizar software de calidad orientado a resolver problemas de alto nivel minimizando los costos del software empleado.

Se decidió darle un enfoque técnico a este trabajo, ya que se va a desarrollar un sistema que sea utilizado como una herramienta para poder replicar información

entre Bases de Datos. Antes de implementar un sistema de replicación es necesario realizar un estudio de factibilidad en el cual se justifique que se requiere trabajar con replicación de datos de manera distribuida.

Es necesario tener presente que un sistema de replicación puede ser implementado en cualquier aplicación ODBC, con ciertas limitaciones las cuales se estudiarán en el transcurso de la tesis.

Para realizar el prototipo y probarlo se usaron tres máquinas con plataformas distintas, Windows 95, Windows 2000 Server y LINUX conectadas en red. También se requirió de una gran variedad de software en cuestión de manejadores de bases de datos como SQL Anywhere de Sybase, MySQL y Access 97. El papel que desempeña la carrera de Lic. En Matemáticas Aplicadas y Computación es muy importante ya que el nivel de conocimientos proporcionados al alumno durante ésta le permiten tener una visión amplia respecto a los sistemas que se utilizan y ser capaz de crear sus propias herramientas para resolver problemas específicos ya sean estadísticos o de toma de decisiones, entre muchos otros, según se presenten en la vida profesional.

El presente trabajo esta dirigido a analistas, desarrolladores, administradores de Bases de Datos y a todas aquellas personas interesadas en construir software, a quienes están concientes del auge de los sistemas de replicación y su utilidad en la actualidad, a los que desean implementar sistemas desarrollados por si mismos; en general a toda la gente creativa en el ámbito de la computación e ingeniería del software que tengan conocimientos de Bases de Datos y programación a nivel avanzado.

El primer capítulo de esta tesis tiene como objetivo identificar brevemente los antecedentes históricos de las Bases de Datos, su definición, como funcionan, como han evolucionado desde sus orígenes, que es un sistema de replicación, para que sirve, se realizan algunos ejemplos de los sistemas distribuidos y su

relación con los sistemas de replicación, además de una breve reseña de este tipo de sistemas.

En el segundo capitulo de la tesis, se desarrolla el modelo y se define la estructura del sistema de replicación que se propone, se explica su relación con los manejadores de Bases de Datos así como la forma de conexión que se usa, los procesos de información que se llevan a cabo, explicándolos detalladamente a través de diagramas de flujo y de procesos, es aquí donde se describe paso a paso el proceso que se realiza al replicar datos de una Base de Datos a otra.

Para concluir en el tercer y último capitulo se hace un diagnóstico de las ventajas del sistema de replicación y realiza una evaluación respecto a la minimización de costos y tiempos de envío de información. Y finalmente las desventajas como la dependencia de la eficiencia en los medios de comunicación y otros. A pesar de que las comunicaciones actualmente cada vez son más confiables sin un enlace dedicado no existe la replicación y en gran parte la calidad del software de replicación depende de un sistema de comunicación aceptable; para propósitos de esta tesis es oportuno aclarar que solo contamos para simular la situación con un hub[1] que comunica tres máquinas, sin embargo su funcionalidad es la misma a la de un sistema que se comunica a grandes distancias, es decir la funcionalidad del sistema en una LAN[2] es la misma que la de una LAN que se conecta y forma parte de una WAN[3].

Tal parece que lo negativo de algunas situaciones son áreas de oportunidad que nos ayudan y en algunos casos nos obligan a perfeccionarnos y ese es el caso de esta tesis, es necesario aprovechar lo aprendido de situaciones difíciles y hasta adversas para solucionar un problema; al cual probablemente nos enfrentemos de nuevo en algún episodio de nuestra vida laboral.

[1] Dispositivo electrónico que sirve para comunicar varias computadoras en red
[2] LAN es un acrónimo de Local Area Network que quiere decir red de área local
[3] WAN es un acrónimo de Wide Area Network que quiere decir red de área amplia

Objetivo:

Diseñar un prototipo de software abierto de Replicación de Bases de Datos Distribuidas, que minimice costos de software y tiempos de respuesta, además de interconectarse con múltiples plataformas, para facilitar la interacción e intercambio de datos sin necesidad de una plataforma homologada que pueda implicar costos adicionales.

Alcance:

El presente prototipo fue diseñado con la finalidad de demostrar que con pocos recursos, es posible crear un software capaz de resolver el problema de replicar información entre bases de datos; sin embargo el sistema esta adaptado para funcionar adecuadamente bajo ciertas restricciones como la plataforma, que debe ser Windows, el protocolo de comunicación TCP IP y utilizar la Arquitectura ODBC, entre otras que se analizan detalladamente en la presente tesis. El prototipo, por su naturaleza tiene errores que pueden dar pauta a perfeccionar el sistema en un futuro o adecuarlo a ciertos tipos de replicación como computo móvil o comercio electrónico.

CAPÍTULO I BASE DE DATOS

1.1 DEFINICIÓN

Es un hecho que los humanos en el transcurso de su existencia han necesitado comunicarse y también han conocido, a partir de su necesidad de subsistencia, diversas formas de dar a conocer sus ideas a sus semejantes contemporáneos.

Al principio de la historia social los humanos se comunicaban con gestos y sonidos guturales, después se perfecciono el método de comunicación y se pudo estructurar un lenguaje hablado. Posteriormente necesitaron perfeccionar la forma en que contabilizaban sus cosechas, sus animales, la manera en que llevaban la cuenta del tiempo, y muchas otras actividades que fueron causa para registrar "datos" de alguna manera o comunicar ideas, un ejemplo de esto puede ser desde pinturas hasta complejos jeroglíficos.

El conocimiento ha sido la recopilación de todas esas experiencias que se han almacenado en alguna parte del cerebro humano transmitidas, en algunos casos, verbalmente de generación en generación y otras veces a través de textos escritos.

Para conocer el significado de que es una Base de Datos se analizará primeramente la palabra datos: "la palabra datos (del latín data, plural de datum) significa simplemente hechos, entidades independientes sin evaluar"[1], a partir de esto se puede definir la palabra dato como un hecho o conjunto de hechos con cierto sentido o la unidad mínima de información que puede registrarse manteniendo un significado, algunos ejemplos de datos pueden ser la cantidad de consumibles específicos en un inventario, el número total de libros de una

[1] Tsai Alice y H. SISTEMAS DE BASES DE DATOS, ADMINISTRACIÓN Y USO, pp. 3, Ed. Prentice Hall Hispanoamericana, México, 1990

biblioteca etcétera. Un conjunto de datos que mantiene una relación entre ellos pueden denominarse como información.

Los datos, en el contexto de computación y generalizando el concepto, se pueden definir como "hechos conocidos que pueden registrarse y que tienen un significado implícito"[2]. Se puede definir una Base de Datos en primera instancia como "Un conjunto de datos relacionados entre sí", es un hecho que los datos deben cumplir con ciertas características para poder conformar una Base de Datos de lo contrario esta misma tesis sería una Base de Datos, una Base de Datos tiene ciertas propiedades, Elmasri y Navathe[3] proponen las siguientes:

Una Base de Datos representa algún aspecto del mundo real, en ocasiones llamado minimundo o universo de discurso. Las modificaciones del minimundo se reflejan en la Base de Datos.

Una Base de Datos es un conjunto de datos lógicamente coherente, con cierto significado inherente. Una colección aleatoria de datos no puede considerarse propiamente una Base de Datos.

Toda Base de Datos se diseña, construye y pobla con datos para un propósito específico. Está dirigida a un grupo de usuarios y tiene ciertas aplicaciones preconcebidas que interesan a dichos usuarios.

Actualmente, además de las propiedades anteriormente mencionadas, el contexto de Bases de Datos se resume de la siguiente manera: "En un entorno de Bases de Datos tratamos con datos comunes típicamente compartidos con muchos

---

[2] Elmasri y Navathe, SISTEMAS DE BASES DE DATOS CONCEPTOS FUNDAMENTALES, pp. 2, Adison Wesley Iberoamericana S.A, Wilmington, Delaware. E.U.A.

[3] Elmasri y Navathe, SISTEMAS DE BASES DE DATOS CONCEPTOS FUNDAMENTALES, Opp cit. Adison Wesley Iberoamericana S.A, Wilmington, Delaware. E.U.A.

usuarios. Estos datos se refieren a "cosas y hechos" externos, y esas "cosas y hechos" constituyen un Universo de Discurso (UoD) (ISO, 1981).

Podemos considerar también un universo del discurso como una parte seleccionada del mundo en el que estamos interesados. Este mundo de interés puede ser real, como inventarios, abstracto, como estructuras de organización de una empresa, o imaginario, como Alicia en el País de las Maravillas."[4]

De forma lógica, los datos se almacenan en estructuras de almacenamiento de datos; donde definimos este tipo de estructuras como el repositorio lógico que establece donde y de que forma se almacenarán los datos, también aquí se especifica que tipos de datos serán almacenados, es decir que tipo de información será almacenada. "Los datos pueden organizarse en muchas formas diferentes; el modelo matemático o lógico de una organización particular de datos recibe el nombre de estructura de datos".

La elección de un modelo de datos es particular, depende de dos cuestiones:

Primero, debe ser lo suficientemente complejo para mostrarnos la relación entre los datos y lo que representan.

Por el contrario, la estructura debe ser lo suficientemente simple para que los datos puedan ser procesados de forma eficiente cuando sea necesario."[5]

Las estructuras básicas de un manejador de datos son las siguientes:

[4] Deen, S.M. FUNDAMENTOS DE LOS SISTEMAS DE BASES DE DATOS, p.p. 303, Editorial Gustavo Gili, S.A., Barcelona, 1987

[5] Seymour, Lipschutz, ESTRUCTURA DE DATOS, p.p. 2, Editorial Mc Graw-Hill, México, 1989

Elemento	Descripción
Relación bidimensional (Tablas):	Una organización de datos en una tabla (también conocida como Tabla).
Renglones (Tuplas):	Representa entidades básicas ó hechos de algún orden, es la estructura de datos elemental, sus operaciones elementales son insertar, recuperar, borrar y consultar. (También conocidos como Registros).
Columnas (Atributos):	Representan propiedades de dichas entidades (también conocidos como Campos).

Tabla 1-1 Estructuras Básicas de un Manejador de Base de Datos

MODELOS BÁSICOS DE ALMACENAMIENTO

A continuación se muestra algunos modelos básicos de almacenamiento, según Lester:[6]

Modelo de Archivo Sencillo

Este Modelo lleva funcionando muchos años. Nació como una extensión natural de los archivadores que se solían utilizar. Es un modelo muy simple, todos los datos se almacenan en una tabla única. Sin embargo, el uso de este modelo está decayendo, excepto para los programadores de COBOL; y desafortunadamente, como no se ajusta bien en Internet, el rendimiento cae drásticamente cuando se introducen muchos registros en este tipo de formato.

[6] Christopher S. Lester, BASES DE DATOS CON VISUAL J++, p.p. 20, Prentice Hall, Madrid, 1998

Frecuentemente, cambiar un pequeño grupo de datos requiere cambiar varios cientos, o miles, de registros.

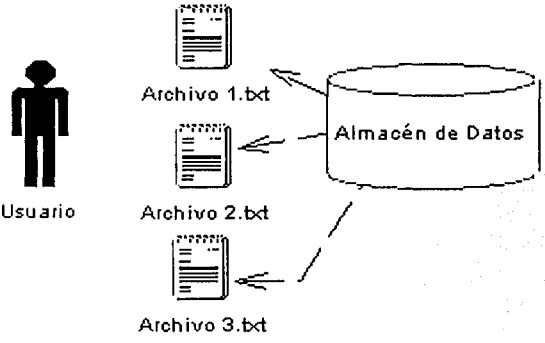


Figura 1-1 Modelo de Archivo Sencillo

Modelo Relacional

Tenemos que agradecer a E.F. Codd la existencia de estos modelos. Su teoría sobre Bases de Datos Relaciónales ha sido el fundamento para casi todos los modelos. La mayor parte de la tecnología es relacional y se ha perfeccionado hasta tal punto que el rendimiento bajo la mayoría de las cargas es bueno.

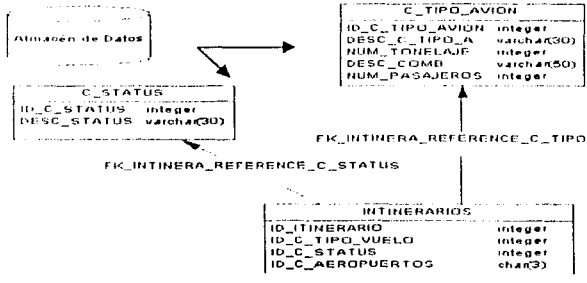


Figura 1-2 Modelo Relacional

¿Qué hace que una Base de Datos sea relacional? Bueno, las relaciones. En realidad, aunque relacionar grupos individuales de datos es la esencia de este modelo, tiene otras ventajas por ejemplo es más sencillo pensar en un modelo relacional como una serie de uniones entre objetos únicos.

Modelo Orientados a Objetos

Este modelo nació de la fascinación por los objetos en la década pasada. El concepto de una Base de Datos Orientada al Objeto es muy parecido al de Java[7]: defina el objeto y encuentre aquellas propiedades que pertenezcan únicamente a ese objeto y nada más; si encuentra propiedades que pueden relacionarse con algo diferente, deben pertenecer a un padre. Esto es parecido a tener varios filtros, uno encima de otro; lo que no se filtra en un nivel pasa al siguiente. En este modelo existe un problema, porque los objetos hijo heredan los datos que contienen los objetos padre.

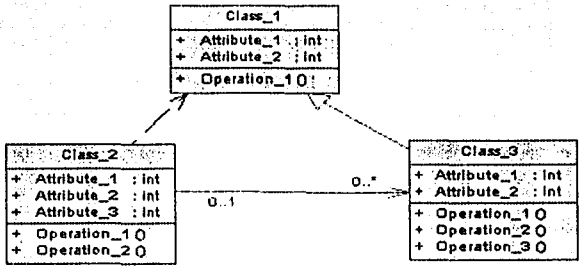


Figura 1-3 Modelo Orientado a Objetos

[7] Java es un lenguaje de programación, que permite crear aplicaciones que interactúen con usuarios de Internet y trabaja bajo la metodología orientada a objetos, Nota del Autor.

Modelo Multidimensional

Este modelo es relativamente nuevo y se desarrolló a partir de las teorías matemáticas de matrices. Aunque la mayoría de las Bases de Datos se basan en matrices, este modelo se estructura para imitar una matriz. Encaja mejor en un entorno en el que se almacenan cantidades masivas de datos resumidos. Las vistas multidimensionales se hacen más fácilmente cuando el motor de la Base de Datos mantiene el concepto de dimensiones directamente, a diferencia de los modelos relaciónales. Los tiempos de consulta también se reducen dramáticamente debido a su estructura de matriz. Ya que una Base de Datos normal repite cada registro durante una consulta, el tiempo adicional llega a ser considerablemente más alto cuando el cursor se mueve por grandes cantidades de registros. Un método que los distribuidores de Bases de Datos han utilizado para minimizar los tiempos de consulta es el uso de los índices en los datos. Comúnmente llamados índices, estas estructuras son simplemente desplazamientos del puntero en la Base de Datos para buscar elementos determinados en los datos. El MDBMS (modelo multidimensional de Base de Datos) almacena los datos en un formato que favorece las interrogaciones porque el motor conoce la localización general de los datos que está buscando. Como se almacena realmente una estructura de matriz, el sistema de Gestión de Bases de Datos Multidimensionales (MDBMS) no tiene que repetir cada línea. En vez de eso, escoge selectivamente las columnas de la matriz que necesita, acortando a veces el tiempo de búsqueda exponencialmente. También ha desaparecido la necesidad de las claves, ya que el propósito principal de estas era facilitar un medio sencillo de buscar los datos deseados. La manera de organizar la información ha variado desde tiempos inmemoriales; se busca que con la organización de los datos se pueda tener información concisa y oportuna para poder aprovecharla de muy diversas maneras y cumplir un objetivo en específico, por ejemplo: actualmente teniendo información organizada podemos saber el número de libros de matemáticas en una biblioteca, la cantidad de un determinado producto en un almacén.

Tal vez el ejemplo más antiguo y no solo de organización de datos sino además de procesamiento de éstos (donde el proceso de un dato implica aplicar en ellos una serie de pasos que usen al dato propio como parámetro ó referencia para producir un resultado), sea el siguiente: "La contabilidad o teneduría de libros por partida doble se originó en los centros comerciales de Italia del siglo XIV; el ejemplo más antiguo conocido proviene de un mercader en Génova y corresponde al año 1340. Su empleo se difundió en forma gradual, pero fue hasta 1494, en Venecia, que un monje franciscano, de nombre Luca Pacioli, publicó su obra Suma de Aritmética, Geométrica, Proportioni Poportionalita, la cual tuvo un importante efecto al difundir el empleo de la contabilidad por partida doble"[8], se puede considerar este ejemplo como un primer intento por procesar y organizar datos que en este caso produce un control para los centros comerciales de aquella época. Con el paso del tiempo el procesar y almacenar los datos se convirtió en una necesidad primordial para mantener un control sobre los procesos[9] que interactúan en una empresa. El procesamiento de datos ha dado pauta al desarrollo de la industria computacional. Un ejemplo de esto puede reflejarse antes de las computadoras electrónicas cuando el Gobierno de los Estados Unidos de América, allá por 1880, requería de un censo de datos y el buró de censos contrato para este fin al Doctor e inventor Herman Hollerit para construir un sistema de procesamiento de datos, el sistema a grandes rasgos consistió en grabar datos sobre una tarjeta de papel del tamaño de un billete perforando la tarjeta, esas tarjetas eran enviadas a una oficina de censos donde máquinas mecánicas lectoras de perforaciones iban procesando los datos. "La necesidad fue la madre del invento: a partir del conocimiento del uso de las tarjetas perforadas en telares Jacquard, el Dr. Hollerit inventó un método de almacenamiento de información basado en ellas. Y así comenzó la era de los ficheros de tarjetas

[8] Cisneros González, José Luis. PANORAMA SOBRE BASES DE DATOS (UN ENFOQUE PRACTICO), p.p. 19-31, Universidad Nacional Autónoma de Baja California, 1998

[9] Al referirse aquí a la palabra "proceso" nos referimos a los procesos que se llevan a acabo en una organización o empresa y que influyen en el flujo de información, Nota del Autor.

mecanizadas que subsistieron como medio para el almacenamiento de la información durante los siguientes sesenta años."[10]

El siguiente es un bosquejo histórico de la Evolución de Sistemas de Procesamiento y Almacenamiento de Datos según Cisneros[11]:

La primera computadora grande, funcional, totalmente electrónica, fue producto de las necesidades balísticas de la milicia durante la guerra; llamada Electronic Numerical Integrator and Calculator (ENIAC), se comenzó a construir en 1943 por John Mauchly y J. Presper Eckert y se terminó tres años después.

A fines de la década de los cincuenta y principios de los sesenta, aparecieron las computadoras basadas en transistores, en vez de tubos de vacío (bulbos); estas máquinas fueron las computadoras de segunda generación: eran mucho más confiables, pequeñas y veloces que las de primera generación. Además de incorporarse en computadoras grandes, esta nueva tecnología, combinada con mejoras en la memoria de núcleos magnéticos, creó las primeras computadoras operativas de menor escala, que tuvieron mucho éxito y proliferaron con rapidez. Dichas máquinas fueron las primeras computadoras para muchos miles de empresas. En los inicios de la década de 1970, se presenta comercialmente la tecnología de disco flexible (floppies), que es uno de los principales dispositivos de almacenamiento secundario en las computadoras actuales.

---

[10] Deen, S.M. FUNDAMENTOS DE LOS SISTEMAS DE BASES DE DATOS, p.p. 17, Editorial Gustavo Gili, S.A., Barcelona, 1987

[11] Cisneros González, José Luis. PANORAMA SOBRE BASES DE DATOS (UN ENFOQUE PRACTICO), p.p. 19-31, Universidad Nacional Autónoma de Baja California, 1998

DESARROLLO DEL MÉTODO DE BASE DE DATOS

En las primeras épocas del procesamiento automatizado de datos, la mayor parte del tiempo y la atención en el desarrollo de la aplicación se invertía en los programas, en vez de dedicarlos a los datos y las estructuras de éstos. El hardware era costoso y limitado en cuanto a la velocidad de la recuperación de los datos. La programación era una nueva disciplina y había mucho que hacer a fin de lograr un procesamiento eficiente. La mecanización parcial del proceso de programación, o aun la estandarización del estilo, se desconocían. En este medio ambiente, el tratamiento de los datos era la preocupación de mayor prioridad. Se volvió más claro que la forma en la que se manejaban (o mal manejaban) los datos en el pasado, era uno de los principales factores en el problema de mantenimiento de programas al que se enfrentaban los programadores. De la síntesis de estos problemas se concluyó que:

1.- Los datos se almacenaban en formatos diferentes y en archivos distintos.
2.- Con frecuencia, los datos no podían compartirse entre programas diferentes, lo que hacía que se requirieran archivos redundantes o programación extra para recuperar los datos de un formato para convertirlos al formato adecuado.
3.- A menudo, los datos no eran recuperables ni estaban seguros.
4.- Por lo general, los programas se escribían de manera tal, que si se modificaba la forma en que los datos se almacenaban, era necesario modificar el programa para continuar trabajando.

Así, insatisfechos por las técnicas aplicadas para las soluciones de los problemas planteados, ocasionados por el método de archivos orientados a las necesidades de cada departamento, algunos diseñadores de sistemas empezaron a buscar, a finales de la década de los sesenta, diversas formas de consolidar las actividades utilizando un método de Bases de Datos.

Una vez analizado el contexto de Bases de Datos, las razones por las cuales se han creado, su evolución y algunas de sus propiedades; se da a continuación una definición explicita de Bases de Datos: **Una Base de Datos es un conjunto de datos que mantiene cierta relación entre sí, estos datos deben ser coherentes, con cierto sentido y deben de estar organizados, además una Base de Datos debe estar orientada a satisfacer necesidades de ciertos usuarios mediante algunos procesos como consulta, actualización, eliminación e inserción de registros. Con el nacimiento del modelo relacional se inicio la era del almacenamiento de datos, fue este acontecimiento el que originó la creación de los manejadores de Bases de Datos.**

1.2 SISTEMA MANEJADOR DE BASES DE DATOS

Un manejador de Bases de Datos o DBMS (de sus siglas en inglés Database Management System) es un sistema administrador de datos, que tiene la capacidad de administrar gran cantidad de datos de forma organizada y coherente; además de proporcionar acceso compartido a múltiples usuarios. "Un sistema manejador de Base de Datos, o DBMS, es un software diseñado para ayudar en el mantenimiento y utilización de grandes colecciones de datos" [12].

Un breve resumen histórico acerca de la evolución de los manejadores de Bases de Datos se presenta a continuación[13]:

[12] Ramakrishnan, Raghu; Gehrke Johannes, DATABASE MANAGEMENT SYSTEMS, p.p. 1-20, Mc Graw Hill, 2000

[13] Extraído de Ramakrishnan, Raghu; Gehrke Johannes, DATABASE MANAGEMENT SYSTEMS, Opp. Cit., Mc Graw Hill, 2000

Desde los primeros días de la computación, almacenar y manipular los datos ha sido el principal punto de aplicación. El primer Manejador de propósito general (DBMS) fue diseñado por Charles Bachman en General Electric en los comienzos de los años 60s y fue llamado el Almacén de Datos Integrado (Integrated Data Store). Este formo las bases para el modelo de datos de red, el cuál fue estandarizado por la Conferencia de Datos, Sistemas y Lenguajes (CODASYL Conference on Data Systems Languages) y fuertemente influenciado por los sistemas de Bases de Datos de los años 60s. Bachman fue el primero en recibir el premio ACM's Turing Award (en la ciencia de la computación equivalente al premio Nóbel) por su trabajo en el área de las Bases de Datos, él recibió el premio en 1973. A finales de los años 60s, IBM desarrollo el sistema manejador de información (Information Management System IMS) DBMS, usado aún hoy en día en muchas instalaciones. IMS formó las bases para un esquema alternativo de representación de datos llamado el modelo de datos jerárquico. El sistema SABRE para hacer reservaciones en las aerolíneas fue desarrollado conjuntamente por American Airlines e IBM al mismo tiempo, y permitió que mucha gente accediera a los mismos datos a través de una computadora en red. Interesantemente, hoy en día el mismo sistema SABRE es usado en servicios de viaje basados en WEB con una supervelocidad.

En 1970, Edwar Codd, del laboratorio de investigación IBM's San José, propuso un nuevo esquema de representación de datos llamado el modelo relacional de datos. Esto comprobó ser una línea divisoria en el desarrollo de los sistemas de Bases de Datos y como consecuencia desencadenó un rápido desarrollo de muchos DBMSs basados en el modelo relacional, junto con un rico organismo de resultados teóricos que impulsaron el campo en una fundación de firma. Codd ganó en 1981 el premio Turing por su trabajo seminal. Los sistemas de Bases de Datos maduraron como una disciplina académica y la popularidad de los DBMSs relacionales cambio al panorama comercial. Sus beneficios fueron ampliamente reconocidos, y el uso de los DBMSs por manejar datos empresariales se convirtió en una práctica estándar. En 1980, el modelo relacional consolido su posición

como el paradigma dominante de los DBMSs, y sistemas de Bases de Datos continuaron ganando amplio uso. El lenguaje de consulta SQL para Bases de Datos Relaciónales, desarrollado como parte del proyecto sistema R, es ahora el lenguaje de consulta estándar. SQL fue estructurado a finales de los 80's, y el actual estándar, SQL-92, fue adoptado por el Instituto de Estándar Nacional Americano (ANSI American National Standart Institute) y la Organización Internacional de Estándares. Dudosamente, el más amplio uso establecido de programación concurrente es la ejecución de los programas de Bases de Datos (llamados transacciones). Los usuarios escriben programas y ellos son quienes los corren por si mismos, y la responsabilidad de correrlos concurrentemente es del DBMSs. James Gray ganó en 1999 el premio Turing por sus contribuciones en el campo del manejo de transacciones en un DBMS. A finales de los años 80's y los 90's se hicieron muchos avances en muchas áreas de los sistemas de Bases de Datos. Investigaciones considerables se han llevado a cabo en cuestión de más poderosos lenguajes de consulta y modelos de datos más enriquecidos, y ha habido un gran énfasis en el soporte de análisis complejo de todas las partes de una empresa. Muchos vendedores (Ej. IBM DB2, Oracle 8, Informix UDS) han extendido sus sistemas con la habilidad de almacenar nuevos tipos de datos como imágenes y texto, y con la habilidad de solicitar consultas más complejas. Se han desarrollado sistemas especializados por numerosos vendedores para crear almacenes de datos (data warehouses), consolidando datos de muchas Bases de Datos, y para llevar a cabo análisis especializados. Un interesante fenómeno es el surgimiento de paquetes de planeación de recursos empresariales (Enterprise resource planning ERP) y manejadores de planeación de recursos (MRP), los cuales añaden una capa substancial de aplicación orientada a presentarse sobre un DBMSs. Los sistemas usados ampliamente, incluyen sistemas de Baan, Oracle, PeopleSoft, SAP, y Siebel. Esos paquetes identifican un conjunto de tareas comunes (e.j. manejamiento de inventarios, planeación de recursos humanos, análisis financiero) realizadas por un gran número de organizaciones y proporcionando una aplicación general orientada a llevar a cabo esas tareas. Los datos son almacenados en DBMSs, relaciones y la capa de aplicación puede

construirse a la medida de diferentes compañías, encaminado a descender los costos globales de las compañías, comparado con el costo de construir la capa de aplicación inconscientemente. Más significativamente, quizá, los DBMSs han entrado a la era de Internet. Mientras la primera generación de sitios Web almacenaban sus datos en sistemas de archivos del sistema operativo exclusivamente, el uso de un DBMSs para almacenar los datos que son accedidos a través de un explorador WEB ha llegado a usarse ampliamente. Las consultas son generadas a través de aplicaciones WEB accesibles y las respuestas son formateadas usando un lenguaje de marcación como el HTML, con la intención de ser fácilmente visualizada en un explorador. Todos los vendedores de Bases de Datos están añadiendo características a sus DBMSs buscando hacer más adecuado para el desarrollo sobre Internet. Un sistema de Bases de Datos es el recipiente donde es almacenada la información. Este recipiente tiene las características de ser[14] :

a) Compartido (varios usuarios pueden accesarlo al mismo tiempo).
b) Integrado (es visto como una unidad, aun cuando esté formado por varios archivos de diferentes tipos de datos).
c) Proporciona independencia de datos y de programas. Por este concepto se entiende que la modificación de la distribución y la organización física de los datos que no afectan ni la estructura lógica ni los programas de aplicación.

Razones para almacenar la información en una Base de Datos

Un sistema de Base de Datos provee de un control centralizado de sus datos, algunas razones para almacenar la información en Bases de Datos son:

[14] Cisneros González, José Luis. PANORAMA SOBRE BASES DE DATOS (UN ENFOQUE PRACTICO), p.p. 19-31, Universidad Nacional Autónoma de Baja California, 1998

1.- La redundancia puede ser reducida y controlada. En un sistema tradicional (archivos orientados a las necesidades), cada aplicación cuenta con sus propios archivos; esto puede ocasionar una redundancia (repetición) considerable en el almacenamiento de datos, con el resultado de un gasto excesivo de espacio para dicho almacenamiento.
2.- La inconsistencia puede ser eliminada. Facilita que los datos guarden en todo momento las similitudes al mundo real.
3.- Los datos pueden ser compartidos.
4.- Seguridad más amplia. El acceso a los datos puede ser reducido hasta el nivel de un solo dato.
5.- Mayor facilidad para mantener la integridad.

Existe una gran diversidad de opiniones de muchos autores acerca de la estructura de un DBMSs, y hay gran variedad de esquemas que describen detalladamente los componentes de un DBMSs, sin embargo el estándar de la arquitectura básica de un DBMSs , donde se describen sus niveles de abstracción, se conoce como la arquitectura de tres esquemas (three-schema architecture también conocida como arquitectura ANSI/SPARC según Tsichritzis y Klug 1978).

El propósito de esta arquitectura es separar las aplicaciones de usuario de la Base de Datos física, y se pueden separar en tres niveles, como se muestra en la figura 1.1, principalmente[15]:

[15] Elmasri y Navathe, SISTEMAS DE BASES DE DATOS CONCEPTOS FUNDAMENTALES, pp. 27, Adison Wesley Iberoamericana S.A, Wilmington, Delaware. E.U.A.

1. El nivel interno tiene un esquema que describe la estructura de almacenamiento físico de la Base de Datos. El esquema interno usa un modelo físico de datos y describe todos los detalles de almacenamiento de datos y rutas de acceso para la Base de Datos.
2. El nivel conceptual tiene un esquema que describe la estructura de la Base de Datos completa para una comunidad de usuarios. El esquema conceptual esconde los detalles de las estructuras de almacenamiento físico y se concentra en describir las entidades, tipos de datos, relaciones, operaciones de usuario, y restricciones. Un modelo de datos de alto nivel o una implementación de un modelo de datos pude usarse en este nivel.
3. El nivel externo o de vista incluye un gran número de esquemas externos o vistas de usuario. Cada esquema externo describe la parte de la Base de Datos que un grupo de usuarios en particular esta interesado y esconde el resto de la Base de Datos de ese grupo de usuarios. Un modelo de datos de alto nivel o una implementación de un modelo de datos pude usarse en este nivel.

La arquitectura de tres esquemas es una herramienta para el usuario para visualizar los niveles esquemáticos de la Base de Datos.

Muchos DBMSs no separan completamente los tres niveles, pero soportan la arquitectura de tres esquemas como una extensión. Algunos DBMSs pueden incluir los detalles del nivel físico en el esquema conceptual.

En muchos DBMSs que soportan niveles de usuario, los esquemas externos son especificados en el mismo modelo de datos que describe la información del nivel conceptual.

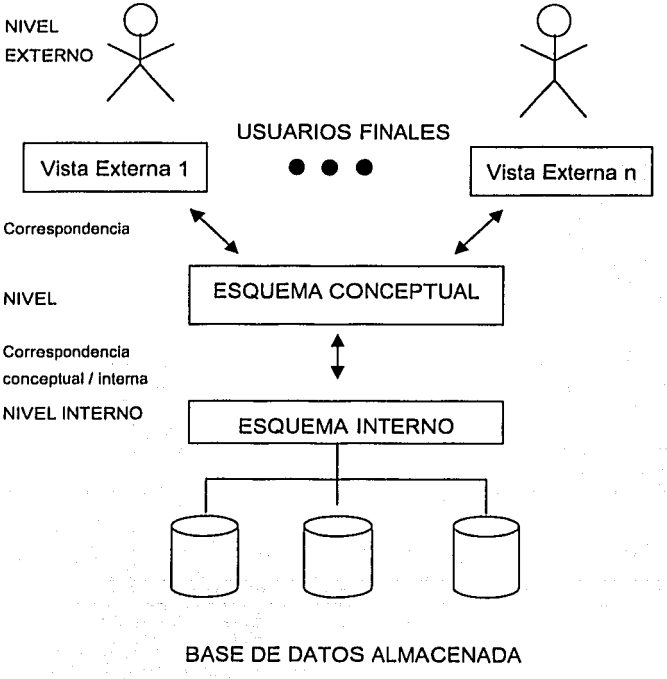


Figura 1-4 Arquitectura de tres esquemas

El primer esquema se presenta a continuación[16] :

[16] tanto el esquema como la descripción del esquema fueron extraídos de: García Molina, Héctor; D. Ullman Jeffrey, Widom; DATABASE SYSTEM IMPLEMENTATION, p.p. 1-20, Prentice Hall, 2000

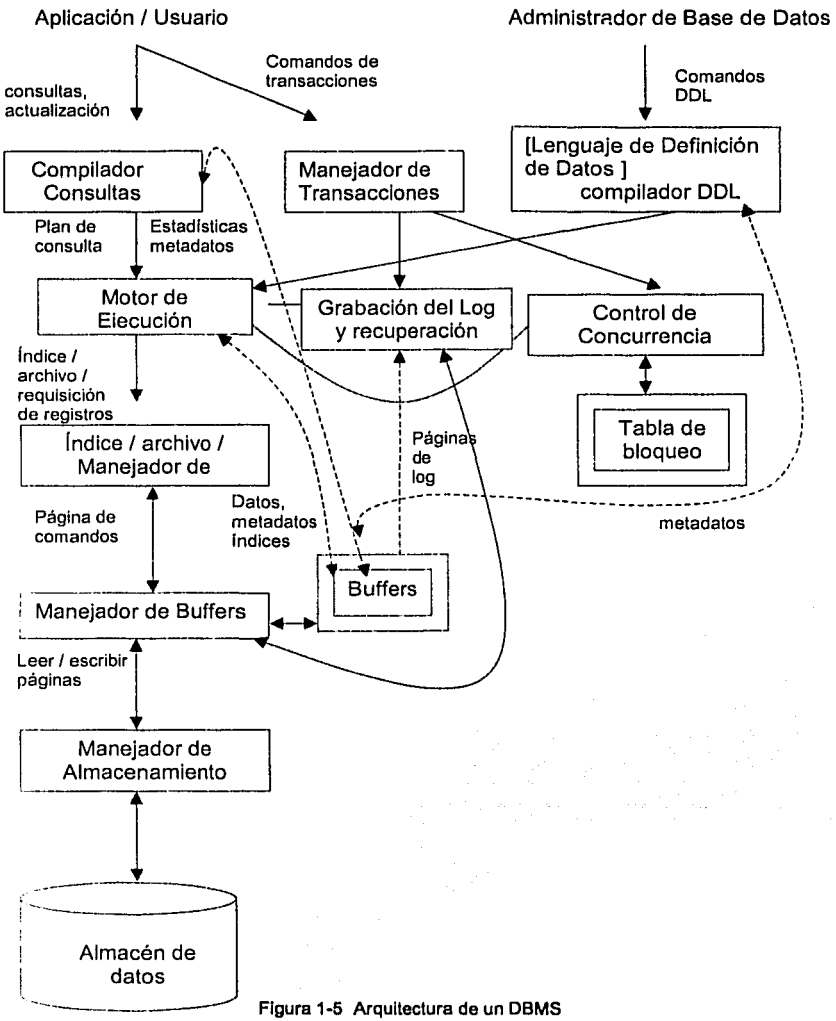


Figura 1-5 Arquitectura de un DBMS

Algunos DBMSs permiten diferentes modelos de datos para ser usados en los niveles conceptual y externo. El esquema describe los componentes que forman un DBMSs, varia en las opiniones de varios autores, para fines de esta tesis se analizarán dos esquemas según Garcia Molina, Ullman, Widom y Ramakrishnan, Gehrke.

"Una importante ventaja del uso de un DBMSs es el que ofrece la independencia de los datos. Esto es, los programas de aplicación son aislados de los cambios de tal manera que los datos son estructurados y almacenados. La independencia de los datos se lleva a cabo a través del uso de los tres niveles de abstracción de datos; en particular, el esquema conceptual y el esquema externo proporcionan distintos beneficios en esta área[17]."

Cajas simples representan componentes del sistema, mientras las cajas dobles representan estructuras de datos en memoria. Las líneas continuas indican control y flujo de datos, mientras que las líneas discontinuas indican solamente flujo de datos. Puesto que el diagrama es complicado, deberíamos considerar los detalles en muchas etapas. Primero, en la cima, sugerimos que hay dos fuentes distintas de comandos en el DBMS:

a) Usuarios convencionales y programas de aplicación que piden datos o modifican datos.
b) Un administrador de Base de Datos: una persona o persona responsable para la estructura o esquema de la Base de Datos.

[17] Ramakrishnan, Raghu; Gehrke Johannes, DATABASE MANAGEMENT SYSTEMS, p.p. 1-20, Mc Graw Hill, 2000

Comandos del lenguaje de definición de datos

El segundo tipo de comando es simple de procesar, y seguimos su camino comenzando por la esquina superior derecha de la figura 1.5. Por ejemplo, el administrador de Base de Datos, o DBA, para registros de una Base de Datos de una universidad debe decidir que debe haber una tabla ó relación con columnas para un estudiante, el curso que el estudiante ha tomado, y una calificación para el estudiante en ese curso. El DBA debe también decidir que las únicas calificaciones permitidas son A, B, C, D y F. Esta información de estructura y restricción es toda una parte del esquema de Base de Datos.

Esto se muestra en la figura 1.5. cuando el DBA introduce datos, quien necesita una autorización especial para ejecutar los comandos de autorización de esquema, puesto que estos pueden tener efectos profundos en la Base de Datos. La alteración del esquema mediante comandos DDL (de sus siglas en inglés Data Definition Language, Definición de Lenguaje de Datos) son analizadas gramaticalmente por el compilador DDL y pasadas al motor de ejecución, el cual envía a través del manejador de índice/ archivo y registro a alterar los metadatos, esto es, el esquema de información para la Base de Datos.

Visión General del procesamiento de consultas

La gran mayoría de interacciones con el DBMS siguen el patrón del lado izquierdo de la figura anexa. Un usuario o un programa de aplicación inicia alguna acción que no afecta el esquema de la Base de Datos, pero puede afectar el contenido de la Base de Datos (si la acción es un comando de modificación) o extraer los datos de la Base de Datos (si la acción es una consulta). Existen dos patrones por los cuales las acciones del usuario afectaran la Base de Datos.

1.- Contestando la consulta. La consulta es analizada gramaticalmente y optimizada por el compilador de consultas. El plan de consulta resultante, o secuencia de acciones que se llevarán a cabo para contestar la consulta, es pasada al motor de ejecución. El motor de ejecución emite una serie de requerimientos para pequeñas piezas de datos, comúnmente registros o tuplas de una relación, a un manejador de recursos que conoce acerca de los archivos de datos (relaciones de contención), el formato y tamaño de los registros en esos archivos, y archivos de índices, los cuales ayudaran a encontrar elementos de archivos de datos rápidamente. El requerimiento de los datos es transformado en páginas y ese requerimiento es pasado al manejador de buffer, cuya tarea básicamente es traer las porciones apropiadas de los datos del almacenamiento secundario (disco, normalmente) donde es guardada permanentemente, a los buffers de memoria principal. Normalmente la Página o bloque de disco es la unidad de transferencia entre buffers y disco. El manejador de buffer se comunica con el manejador de almacenamiento para obtener los datos del disco. El manejador de almacenamiento debe involucrar comandos del sistema operativo, pero más comúnmente, el DBMS emite comandos directamente al controlador de disco. Antes de describir el proceso de transacciones del esquema, se da a continuación una serie de conceptos (propiedades de las transacciones de datos) con las cuales el autor (Garcia Molina, Ullman, Widom y Ramakrishnan, Gehrke) se auxilia durante la descripción del proceso de transacción.

Las propiedades ACID de las transacciones

Propiedad	Descripción
A	Una clave para Atomicidad, la ejecución de todas o ninguna transacción.
I	Una clave para Aislamiento (Isolation en inglés), el hecho de cada transacción debe aparecer ejecutada si es que otra transacción no sé esta ejecutando al mismo tiempo.
D	Una clave para Durabilidad, la condición del efecto en una Base de Datos de que una transacción no debe perderse, una vez que la transacción haya sido completada.

Tabla 1-2 Propiedades ACID de las Transacciones

La restante letra C, es una clave para Consistencia. Esto es, todas las Bases de Datos tienen restricciones, o expectativas acerca de las relaciones entre los elementos de datos (e.j. un cierto atributo es una llave, los estudiantes no pueden tomar más de 8 cursos al mismo tiempo, y así por el estilo). Las Transacciones se usan para mantener las consistencia de la Base de Datos.

2.- Procesamiento de Transacciones: Consultas y otras acciones son agrupadas en transacciones, las cuales son unidades que deben ser ejecutadas atómicamente y en aislamiento, a menudo cada consulta ó acción de modificación es una transacción por si misma. En suma, la ejecución de las transacciones debe ser durable, esto quiere decir que el efecto que tiene cualquier transacción completada debe preservarse aun si el sistema falla de alguna manera justamente después que la transacción se haya completado. Se divide el procesador de transacciones en dos partes principales:

(a) Un manejador del control de concurrencia, (o programador), responsable de asegurar la atomicidad y aislamiento de las transacciones.

(b) Un manejador de registro de log y recuperación responsable de la durabilidad de las transacciones

Buffers de Memoria Principal y el Manejador de Buffer

Los datos de una Base de Datos normalmente residen en almacenamiento secundario; en los sistemas de computación actuales <<almacenamiento secundario>> generalmente significa disco magnético. Sin embargo para optimizar cualquier operación útil en los datos, esos datos deben estar en memoria principal.

Así de esta manera, un componente del DBMS llamado el manejador de buffer es el responsable de particionar la memoria principal disponible en buffers, los cuales son regiones de páginas-dimensionadas en las cuales los bloques de disco pueden ser transferidos. Así, todos los componentes del DBMS que necesiten

información del disco interactuarán con el buffer y el manejador de buffer, ambos directamente o a través del motor de ejecución. Los tipos de información que muchos componentes deben incluir son:

Componente	Descripción
1. Datos:	El contenido de la Base de Datos por si misma.
2. Metados:	El esquema de la Base de Datos que describe la estructura, y restricciones de la Base de Datos.
3. Estadísticas:	Información recolectada y almacenada por el DBMS acerca de las propiedades de los datos como el tamaño, valores, varias relaciones u otros componentes de la Base de Datos.
4. Índices:	Estructuras de datos que soportan acceso eficiente a los datos.

Tabla 1-3 Tipos de Información que muchos componentes incluyen

PROCESAMIENTO DE TRANSACCIONES

Como se habia mencionado, es normal agrupar una o más operaciones de Base de Datos en una transacción, la cual es la unidad de trabajo que debe ejecutarse atómicamente y en aparente aislamiento de otras transacciones. En suma, un DBMS ofrece la garantía de la durabilidad: que el trabajo de una transacción completada no puede perderse. Por consiguiente el manejador de transacciones acepta comandos de transacciones desde una aplicación, la cual le dirá al manejador de transacciones cuando comienzan y terminan las transacciones, al igual que la información acerca de las expectativas de la aplicación (algunas veces no se pide que se requiera atomicidad, por ejemplo). El procesador de transacciones optimiza las siguientes tareas:

1.- Registro de log [logging]: En orden de asegurar durabilidad, cada cambio en la base de datos es registrado en el log separadamente en el disco. El manejador de log sigue una de muchas políticas destinadas a asegurar que no importe si una falla del sistema o <<accidente>> ocurra, un manejador de recuperación estará habilitado para examinar los cambios registrados en el log y restaurar la Base de Datos a algún estado consistente. El manejador de log inicialmente escribe el log en buffers y trata con el manejador de buffer para asegurar que se escribieron a disco (a fin de que los datos puedan sobrevivir a un accidente) en los momentos apropiados.

2. Control de Concurrencia: Las Transacciones deben ejecutarse en aislamiento. Pero en muchos sistemas, las transacciones se ejecutan al mismo tiempo. El programador (manejador de control de concurrencia ó scheduler) debe asegurar que las acciones individuales de transacciones múltiples son ejecutadas en tal orden que el efecto de la red sea el mismo que si las transacciones hayan sido ejecutadas en su totalidad, una a la vez. Un programador clásico funciona manteniendo bloqueos (candados) en ciertas piezas de la Base de Datos. Esos bloqueos previenen que dos transacciones acceden a la misma pieza de datos de tal forma que interactúen de forma errónea. Los bloqueos generalmente están almacenados en una tabla en la memoria principal, como se sugiere en la figura anexa. El programador afecta la ejecución de las consultas y otras operaciones de la Base de Datos prohibiendo al motor de ejecución el acceso a las partes bloqueadas de la Base de Datos.

3. Resolución de punto muerto (checkpoint) : Como las transacciones compiten por recursos a través de los bloqueos que el programador otorga, estas pueden encontrarse en una situación donde ninguna puede proceder

cada una necesita que alguna otra transacción haya terminado. El manejador de transacciones tiene la responsabilidad de intervenir y cancelar (abortar) una o más transacciones para dejar a otras proceder.

El DBMSs acepta comandos SQL generados por una gran variedad de interfases de usuario, produce planes de evaluación de consulta, ejecuta esos planes contra la Base de Datos, y regresa los resultados. (Esto es una simplificación: los comandos SQL pueden ir incrustados en un lenguaje de programación para aplicaciones anfitrión, Ej. Programas como Java o Cobol).

Cuando un usuario emite una consulta, la consulta es verificada por el analizador gramatical y presentada al optimizador de consultas, el cual usa información acerca de cómo los datos están almacenados para producir un plan de ejecución eficiente para la evaluación de la consulta. Un plan de ejecución es un anteproyecto para la evaluación de una consulta, y es usualmente representada como un árbol de operadores relacionales (con comentarios que contienen información detallada adicional acerca de que métodos de acceso usar, etc.).

Los operadores relacionales sirven como los ladrillos de construcción para la evaluación de las consultas expuestas contra los datos.

El esquema de Ramakrishnan se muestra en la figura 1.6:

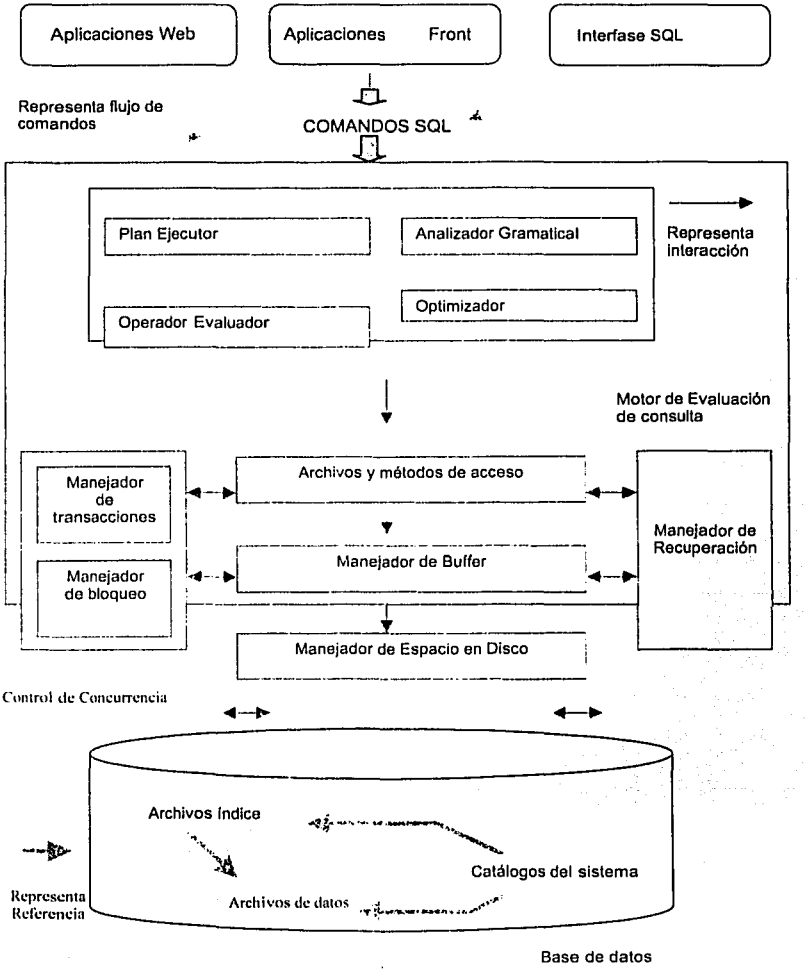


Figura 1-6 Arquitectura de un DBMS según Ramakrishnan

El código que implementan operadores relacionales ubicados en la parte superior de la capa de archivos y métodos de acceso. Esta capa incluye una variedad de software para soportar el concepto de un archivo, el cual, en un DBMSs, es una colección de páginas o una colección de registros. Esta capa comúnmente soporta un archivo de acumulamiento (heap file), o archivo de páginas desordenadas, mejor conocido como indices. En suma para mantener un camino de las páginas en un archivo, esta capa organiza la información dentro de las páginas.

El código de la capa de archivos y métodos de acceso situada sobre el manejador de buffer, trae paginas del disco a la memoria principal como se necesite en respuesta a los requerimientos de lectura.

La última capa del software del DBMSs trata con el **manejo de espacio en disco,** donde los datos son almacenados. Los niveles superiores asignan espacio, quitan espacio, leen y escriben páginas a través de esta capa, llamada manejador de espacio en disco.

El DBMSs soporta concurrencia y recuperación de fallas del sistema por medio de una cuidadosa programación (scheduler) de los requerimientos del usuario y manteniendo en un log todos los cambios de la Base de Datos. Los componentes del DBMSs asociados con un control de concurrencia y recuperación involucran al manejador de transacciones, el cual asegura que los bloqueos a las transacciones requeridas o canceladas en acuerdo con un adecuado protocolo de bloqueo y programación de las transacciones ejecutables; el manejador de bloqueo, el cual mantendrá los requerimientos de bloqueo y el otorgamiento de bloqueos a los objetos de la Base de Datos cuando estos estén disponibles; y cuando el manejador de recuperación, el cual es responsable de mantener el log, y restaurar el sistema a un estado consistente después de una falla en el sistema. El manejador de espacio en disco, manejador de buffer, y la capa de archivos y métodos de acceso deberán interactuar con esos componentes.

Como puede observarse ambos diagramas Figura 1-5 y 1-6 describen prácticamente el mismo flujo de datos para un sistema administrador de Base de Datos, y de esta forma queda explicado su funcionamiento; en el siguiente capítulo se hará un estudio más profundo del manejo del log de transacciones, puesto que es de ahí de donde se obtendrán los datos necesarios para efectuar la replicación.

Actualmente existen una gran cantidad de manejadores de Bases de Datos, todos con características similares, a continuación se presenta un bosquejo de algunos manejadores de Bases de Datos y sus características principales[18]:

Manejador de Base de Datos	Características Principales
Microsoft SQL Server	Sistema basado en el modelo relacional Multiusuario Compatibilidad con XML Análisis habilitado para web Alta disponibilidad Seguridad
Oracle	Sistema basado en el modelo relacional Multiusuario Ambientes Cliente Servidor (procesamiento distribuido) Manejador de grandes cantidades de espacio en la Base de Datos (potencialmente terabytes) Alto rendimiento en el procesamiento de transacciones Seguridad

[18] Aquí es importante señalar que para propósitos de esta tesis se tomaron las características de las últimas versiones en general (últimos 5 años), la información fue extraída principalmente de manuales de configuración e instalación de los diferentes manejadores; las plataformas (sistemas operativos) varían dependiendo de cada sistema, para la presente investigación se tomaron características para Windows NT y Windows 2000 Nota del Autor.

Manejador de Base de Datos	Características Principales
Sybase Adaptive Server Enterprise	Sistema basado en el modelo relacional Multiusuario Soporta gran demanda de requerimientos de transacción intensiva Manejador avanzado de información XML Alto desempeño de ejecución de JavaBeans empresariales Seguridad
Informix	Sistema basado en el modelo relacional Particionamiento de datos Consultas de datos paralelas Respaldo y restauración paralelas Carga y descarga paralela
My SQL	Sistema basado en el modelo relacional Multiusuario Conectividad Rapidez Seguridad Software libre

Tabla 1-4 Características principales de un manejador de Base de Datos

VISIÓN GENERAL DE LA ARQUITECTURA DE UN MANEJADOR DE MICROSOFT SQL SERVER

En este sistema manejador de Bases de Datos los datos están organizados en componentes lógicos visibles para los usuarios. Una Base de Datos es físicamente implementada como dos o más archivos. Cuando se usa una Base de Datos, se trabaja principalmente con componentes lógicos como tablas, vistas,

procedimientos, y usuarios. Comúnmente, solo el administrador de la Base de Datos es el que necesita conocer los detalles de la implementación física[19].

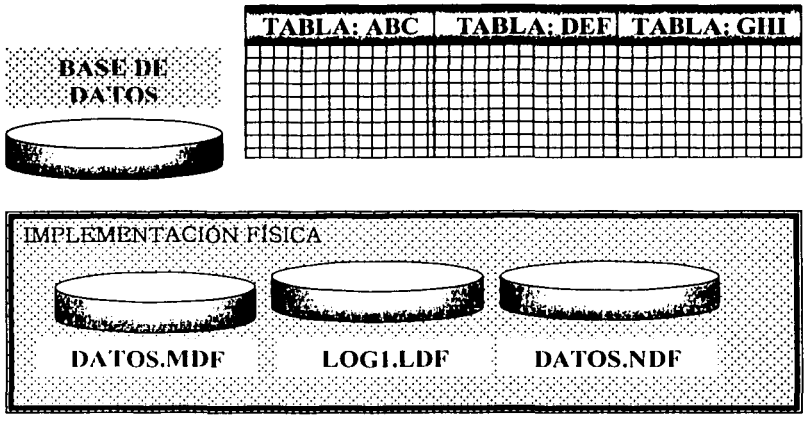


Figura 1-7 Implementación física de una Base de Datos de Microsoft SQL Server

Cada instancia de SQL Server tiene cuatro Bases de Datos de sistema (master, model, tempdb, y msdb) y una o más Bases de Datos de usuario. Algunas organizaciones tienen solo una Base de Datos de usuario, la cual contiene todos los datos de la organización. Algunas organizaciones tienen diferentes Bases de Datos para cada grupo en su organización, y algunas veces una Base de Datos se usa por una aplicación simple. Por ejemplo, una organización puede tener una Base de Datos para clientes, una para nómina, una para una aplicación manejadora de documentos, y así por el estilo.

[19]Una nota importante es que en sistemas anteriores a Microsoft SQL 2000, las extensiones para los dispositivos (el espacio físico o archivos donde se almacenan los datos en disco) eran archivos con extensiones "dat", en Microsoft SQL Server 2000 esto ha cambiado y se les da una extensión "mdf" si son datos y "ldf" de log

Algunas veces una aplicación usa solo una Base de Datos, otras aplicaciones deben acceder a muchas Bases de Datos.

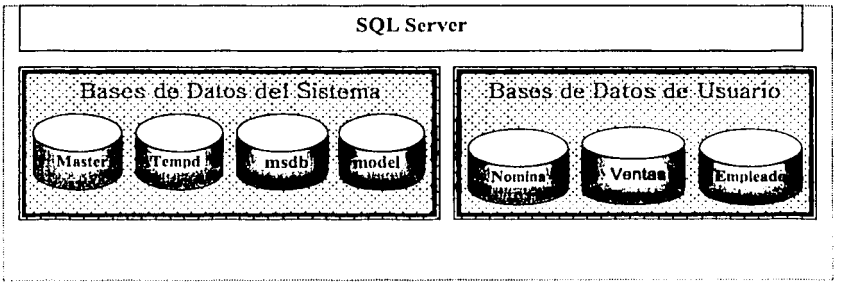


Figura 1-8 Implementación física de una Base de Datos de Microsoft SQL Server

A continuación se describen algunas características técnicas de este manejador en las últimas versiones[20]:

Los usuarios crean una Base de Datos en SQL Server con la intención de almacenar sus datos. Esas Bases de Datos son llamadas Bases de Datos de usuario. SQL Server también usa algunas Bases de Datos para optimizar sus propias operaciones y mantener su propia información acerca de los usuarios de Bases de Datos. Esas Bases de Datos son creadas durante la instalación de SQL Server y son conocidas como Bases de Datos del Sistema.

Toda la información a nivel sistema es almacenada en la Base de Datos Master, la cual guarda cuentas de usuario y configuración de ambiente del sistema así como información acerca de los usuarios de la Base de Datos. La Base de Datos Master es muy crucial para SQL Server.

[20] Extraído de Divya Chaturvedi, Paritosh Pathak, Administering SQL Server 7, p.p. 14-21, Mc Graw Hill, 1999

La Base de Datos Temdb almacena todos los objetos temporales y todas las conexiones de usuario son almacenadas en ella. Es también usada por SQL Server para crear tablas de trabajo intermediarias cuando se ejecuta una consulta compleja.

La Base de Datos Model proporciona una plantilla para todos los usuarios de Bases de Datos. Cada Base de Datos de usuario tiene algunas tablas, las cuales son usadas por SQL Server para almacenar metadatos. Esas tablas son conocidas como tablas del sistema. En cualquier momento que una nueva Base de Datos es creada, todos los objetos del sistema son copiados de la Base de Datos model. Los usuarios crean las restantes tablas para esa Base de Datos.

La Base de Datos MSdb es usada para almacenar información acerca de las tareas programadas. SQL Server contiene un servicio de construcción de tareas (Agente del SQL Server) para correr tareas de una agenda básica. Esto proporciona la facilidad de emitir alertas para reconocer sucesos o fallas de las tareas. El Agente de SQL Server usa la Base de Datos MSdb para administrar las tareas y alertas.

La Base de Datos de Distribución es usada para los procesos de replicación. No es creada durante la instalación sino en cuando la funcionalidad de replicación se instala.

La Base de Datos pubs no es una Base de Datos del sistema, pero se crea durante la instalación; es una Base de Datos de ejemplo.

Los requerimientos mínimos de Hardware para Instalar SQL Server son los siguientes:

Procesador: Pentium 133 Mhz
RAM: Memoria en Ram 32 Mb
Unidad de CD: CD-ROM

Requerimientos de Disco Duro:

Tipo de Instalación	Máximo	Mínimo	Típico
Espacio Necesario	190 Mb	73 Mb	75 Mb

MODOS DE LICENCIA DE SQL SERVER

Modo Licencias por Servidor

El modo licencia de SQL Server por Servidor se basa en el número de conexiones simultaneas. La persona selecciona el máximo número de usuarios que accederán simultáneamente a SQL Server. El número de licencias de acceso de cliente compradas deben ser igual al número máximo de conexiones especificadas por el modo servidor. Este modo es adaptable para organizaciones donde no todos los usuarios necesiten acceder al servidor al mismo tiempo- por ejemplo, una compañía de renta de autos donde puede haber 50 empleados accesando al Servidor SQL, sin embargo, solo 20 usuarios acceden simultáneamente al servidor SQL. Los requerimientos de licencia de la compañía de renta podrían ser solo 20 licencias de conexión, y por consiguiente este método de licencia es menos costoso que el de licencias por sede.

Modo Licencias por Sede

El modo licencia por sede requiere una licencia por cada computadora que acceda al Servidor SQL. Si se selecciona este modo durante la instalación, la licencia no puede cambiarse después. Es el más cómodo para instalaciones donde todos los usuarios necesitan acceder al servidor al mismo tiempo y tener conexiones simultaneas al servidor.

VISIÓN GENERAL DE LA ARQUITECTURA DE UN MANEJADOR DE ORACLE

Una base de Oracle tiene una estructura lógica y una física separadas, el almacenamiento físico de los datos puede manejarse sin afectar el acceso a las estructuras lógicas de almacenamiento.

Estructura Física de una Base de Datos

La estructura física de una Base de Datos de Oracle es determinada por los archivos del sistema operativo que constituyen la Base de Datos. Cada Base de Datos de Oracle esta hecha de tres tipos de archivos: Uno o más archivos de datos, dos o más archivos de reconstitución de log de datos y uno o más archivos de control. Los archivos de una Base de Datos de Oracle proporcionan el almacenamiento físico actual para la información de la Base de Datos.

Estructura Lógica de Una Base de Datos

La Estructura lógica de una Base de Datos Oracle esta determinada por:

- Uno o más espacios de tablas (tablespaces): Un espacio de tabla (tablespace) es una área lógica de almacenamiento
- Los objetos esquema de la Base de Datos: Los *objetos esquema* son las estructuras lógicas que directamente se refieren a los datos de la Base de Datos. Los objetos esquema incluye estructuras tales como tablas, vistas, secuencias, procedimientos almacenados, sinónimos, índices, clusters, y ligas de la Base de Datos.

Las estructuras de almacenamiento lógico, incluyendo espacios de tablas, segmentos, y extensiones estipulan como se usa el espacio físico de la Base de Datos. Los objetos esquema y las relaciones forman el diseño relacional de la Base de Datos.

Una Instancia de Oracle

Cada vez que una Base de Datos se inicia, el sistema de área global (system global area SGA) se aloja y se inician procesos alternos de Oracle (background). El sistema de área global es un área de memoria usada por la información compartida por los usuarios de la misma. A la combinación de los procesos alternos y los buffers de memoria se le llama una instancia de Oracle. Una instancia tiene dos tipos de procesadores: procesos de usuario y procesos de Oracle. Un proceso de usuario ejecuta el código de una aplicación (como las mismas aplicaciones de Oracle) o una herramienta de Oracle (como Oracle Enterprise Manager <administrado empresarial de Oracle>). Procesos de Oracle son procesos del servidor que ejecutan los procesos de usuario y los procesos alternos que ejecutan el trabajo de mantenimiento para el servidor Oracle.

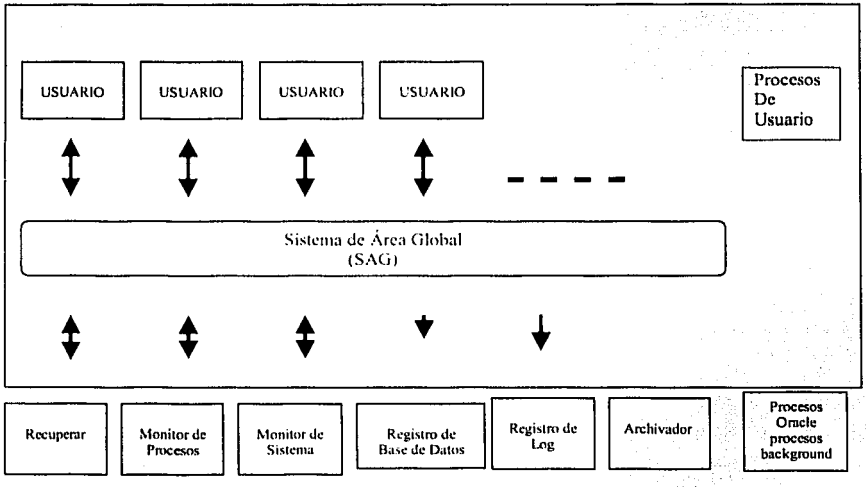


Figura 1-9 muestra una instancia de multiprocesos de Oracle

VISIÓN GENERAL DE LA ARQUITECTURA DE UN MANEJADOR DE SYBASE SQL SERVER (ADAPTIVE SERVER ENTERPRISE)

El servidor SQL (en versiones anteriores a la 11.5 se conoce como servidor SQL) organiza los datos, basándose en el modelo relacional de datos. La vista lógica de los datos este modelo involucra relaciones entre grupos de tablas. Cada tabla contiene atributos que describen aspectos de una entidad u objetos. SQL Server soporta muchas Bases de Datos. Algunas de ellas se llaman Bases de Datos del sistema y contienen información relevante acerca del control del Servidor SQL. El otro tipo de Bases de Datos soportada por el Servidor SQL se llaman de aplicación o usuario.

SQL Server puede instalarse con un número variable de Bases de Datos. La Base de Datos Master contiene información para la administración del sistema y sobretodo el control del SQL Server y otras Bases de Datos a través de las tablas del sistema.

La Base de Datos Model sirve como una plantilla para crear otras Bases de Datos en SQL Server. Cuando el comando create database es emitido, el contenido de la Base de Datos Model es copiado a la nueva Base de Datos

La Base de Datos Sybsystemprocs fue añadida al liberar la versión del sistema 10 para almacenar los procedimientos almacenados que solían residir en la Base de Datos master en versiones anteriores.

La Base de Datos sybsecurity también fue añadida al sistema 10. Esta Base de Datos contiene dos tablas de sistema específicamente (sysaudits, sysaditoptions) que mantienen una auditoria de los principales eventos que ocurren en SQL Server.

Como se han mencionado anteriormente la Base de Datos master contiene información acerca del ambiente de SQL Server en su totalidad. Ejemplos significantes de esta información: El número de Bases de Datos en SQL Server, parámetros de configuración, información de los dispositivos usados para almacenar Bases de Datos y logs de transacciones, dispositivos de respaldo, información relacionada con los dispositivos físicos, bloqueos activos, roles de usuarios, información de cuentas de usuario, mensajes de error y de sistema, procesos activos y servidores remotos.

A continuación se muestra un listado de las tablas de sistema que conforman la Base de Datos master y model; y una breve descripción de las funciones que realizan:

Nombre de Tabla de Sistema	Descripción
Syslogins	Contiene información acerca de las cuentas de usuario
Sysservers	Contiene los ids (identificadores) de otros servidores con los cuales el servidor puede comunicarse
Sysprocesses	Información acerca de los procesos activos actuales
Sysconfigures	Lista de variables que los usuarios pueden definir y usar en sus programas SQL.
Sysmessages	Lista de mensajes de error y advertencias usadas por el sistema
Sysdatabases	Información acerca de todas las Bases de Datos en este servidor
Sysusages	Asignación de espacio físico en disco para cada Base de Datos

Nombre de Tabla de Sistema	Descripción
Syslocks	Lista de bloqueos activos en el sistema por la Base de Datos y las tablas
Syscharsets	Soporta conjunto de caracteres o modos de ordenación
Syscurconfigs	Parámetros de configuración
Sysengines	Número de motores activos
Syslanguages	Información acerca de los lenguajes soportados
Sysloginroles	Información acerca de los roles definidos por el sistema
Sysremotelogins	Información acerca de los usuarios remotos
Syssrvroles	Información acerca de los roles del servidor
Sysdevices	Información acerca de los dispositivos de respaldo y de Base de Datos
Sysmonitors	Información de interés para propósitos de monitoreo
Systestlog	Información acerca de las iteraciones
Syslogshold	Información del log en el nivel dbid
Syslisteners	Información de la dirección

Tabla 1-5 Tablas de sistema de la Base de Datos Master

Las siguientes tablas de sistema existen en la Base de Datos model, y por consecuencia, son copiadas a todas las Bases de Datos de usuario cuando son creadas.

Nombre de Tabla de Sistema	Descripción
Sysalternates	Información de las cuentas de usuario
Syscolums	Información acerca de todas las columnas en tablas y vistas, y parámetros en procedimientos

Syscomments	Información acerca de vistas, reglas, defaults y triggers
Sysdepends	Referencia cruzada de procedimientos, vistas, tablas que son invocadas por otros procedimientos, vistas y triggers
Sysindexes	Información: índices no agrupados y agrupados, tablas sin índices, registros que contienen texto o imágenes
Syskeys	Información de llaves primarias, foráneas y comunes
Syslogs	El transaction log (LOG DE TRANSACCIONES) de la Base de Datos
Sysobjects	Información de todas las tablas, vistas, procedimientos, reglas y triggers
Sysprocedures	Información de cada vista, regla, default, trigger y procedimiento
Sysprotects	Información de los permisos de usuario
Syssegments	Información de segmentos
Systypes	Información de los tipos de datos proporcionados por el sistema y los definidos por el usuario
Sysusermessages	Información de los mensajes definidos por el usuario
Syscontraints	Información de la columna / tabla marcada con restricción (constraint)
Syslabels	Información de etiquetas
Sysreferences	Información de las restricciones columna / tabla de integridad referencial (constraint)
Sysroles	Información de roles

Nombre de Tabla de Sistema	Descripción
Systhresholds	Información de los umbrales (usados para triggering transaction log housekeeping)
Sysusers	Información de los usuarios de Bases de Datos
Syspartitions	Información de particiones para tablas
Sysattributes	Información de clases y atributos

Tabla 1-6 Tablas de sistema de la Base de Datos Model

SQL Server tiene un proceso simple, de arquitectura de multihilos. Todos los clientes conectados son vistos como tareas con un proceso simple, como lo son muchas otras actividades (como el respaldo y recuperación automática), lo cual quiere decir que cada cliente adicional que se conecta al servidor requiere 48K de memoria adicional (un subconjunto de los requerimientos de memoria de un proceso) en lugar del total de memoria (el cual puede ser de hasta 2 MB).

Esto permite a SQL soportar un número de usuarios mayor a los que el núcleo del servidor de Bases de Datos esta diseñado alrededor de "un proceso simple por arquitectura cliente".

Desde 1991, SQL Server ha soportado SMP (symmetric Multiprocessing, Multiprocesamiento Simétrico), el cual permite tomar ventaja de las computadoras que tienen plataformas que soportan múltiples CPUs (Ej. Computadora central status multiplexado tolerante a fallas). Esto termina esparciendo el trabajo que normalmente hace un solo CPU a través de múltiples CPUs.

Respecto a la última versión de Sql Server Sybase

Plataformas soportadas:

- Compaq Tru64	- HP/UX (32 &64 bits)	- IBM AIX (32 &64 bits)
- Linux	- Microsoft Windows NT/2000	- Microsoft windows 98
- SGI IRIX (32 &64 bits)	- Sun Solaris (32 &64 bits)	

Especificaciones del Servidor	Tamaño
Bases de Datos por Adaptive Server Enterprise	32767
Tamaño de la Base de Datos	4 terabytes
Extensión de Base de Datos por un Update	16
Tablas en una consulta	50
Usuario por Base de Datos	2,146,484,223
Columnas por tabla	1024
Tamaño del servidor	8 terabytes
Tamaño de página	2K, 4K, 8K, 16K
Bases de Datos abiertas por una Transacción	16
Logins por servidor	2,147,516,416
Grupos por Base de Datos	1,032,193
Argumentos para un Procedimiento almacenado	2048

UNA VISIÓN GENERAL DE UN MANEJADOR INFORMIX (INFORMIX DYNAMIC SERVER).

Informix Dynamic Server es un servidor de Bases de Datos Relaciónales multitareas que explota arquitecturas de multiprocesadores simétricos (SMP) y uniprocesadores.

Dynamic Server tiene las siguientes características:

- Arquitectura Cliente-Servidor: Esta diseñado para trabajar en entornos cliente – servidor.
- Escalabilidad: El sistema podrá ser actualizado a futuras versiones.
- Alto rendimiento: Soporta alta intensidad de concurrencia.
- Tolerancia a fallas y alta disponibilidad: Contiene un log de transacciones para llevar un seguimiento de las transacciones que se realizan.
- Sistema dinámico de administración: La administración de las Bases de Datos es rápida y confiable.
- Consultas distribuidas de datos: permite consulta Bases de Datos Distribuidas.
- Seguridad del servidor de Base de Datos: Alta seguridad a usuarios conectados.

Dynamic Server es un servidor de Base de Datos que procesa requerimientos de datos de las aplicaciones del cliente.

Manejador dinámico de memoria compartida

Todas las aplicaciones que usan una instancia de un servidor de Base de Datos comparten espacio de memoria del servidor de Base de Datos.

Después de que una aplicación lee datos de una tabla, otra aplicación puede acceder cualquier dato que ya se encuentra en memoria.

Algunos requerimientos para la instalación de informix, en sistemas Windows:

- Windows NT, Versión 4.0 o anterior
- Service pack 3
- TCP/IP
- 16 Megas en RAM
- 140 Megabytes de espacio en disco[21]

Se dice que una Base de Datos tiene o usa un registro de log de transacciones, cuando la manipulación de los datos por medio de expresiones SQL en una Base de Datos generan registros lógicos de log.

Actividades que registran log en una Base de Datos con el Log de Transacciones

Si una Base de Datos utiliza un log de transacciones, todas las expresiones SQL de manipulación SQL que se ejecutan en la Base de Datos generan uno o más registros de log. Las expresiones son las siguientes:

INSERT : Inserta registros de una tabla en una Base de Datos
LOAD : Carga registros de una tabla en una Base de Datos desde un archivo
DELETE : Sirve para eliminar registros de una tabla en una Base de Datos
UPDATE : Actualiza Registros de una tabla en una Base de Datos
UNLOAD : Descarga registros de una tabla de una Base de Datos en un archivo

[21] incluyendo un mínimo de 30 Mb para la raiz de espacio en base de datos y 20 Mb de espacio en base de datos adicionales

Una Visión General de un manejador MySQL

MySQL es un muy rápido, multitareas, multiusuario y robusto servidor de Base de Datos (SQL). MySQL pretende como misión critica, cargar sistemas 'pesados' de producción, igualmente incrustarse en el conjunto de software destacado, es el más popular software abierto de Bases de Datos.

El software de Base de Datos MySQL es un sistema cliente/servidor que consiste en un servidor SQL multitareas que soporta diferentes back ends, diferentes programas clientes y librerías, herramientas administrativas, y un amplio rango de interfases de programación (APIS).

Sintaxis y utilización del SQL de MySQL

Creación, vaciado y selección de Bases de Datos

CREATE DATABASE:	Crea la Base de Datos
DROP DATABASE:	Elimina la Base de Datos
USE:	Selecciona la Base de Datos a usar

Creación, alteración y vaciado de Tablas e Índices

ALTER TABLE	Altera la estructura de la tabla
CREATE INDEX	Crea un índice
CREATE TABLE	Crea una tabla
DROP INDEX	Borra un índice
DROP TABLE	Borra una tabla

Obtención de Información sobre Bases de Datos, tablas y consultas

DESCRIBE	Obtiene información de la estructura de las tablas
EXPLAIN	Obtiene información del contenido de las tablas
SHOW	Muestra las Tablas

Selección de Información de tablas

SELECT	Selecciona datos de las tablas y los muestra

Modificación de Información en tablas

DELETE	Elimina registros de una tabla en una Base de Datos
INSERT	Inserta registros de una tabla en una Base de Datos
LOAD DATA	Carga registros de una tabla en una Base de Datos desde un archivo
OPTIMIZE TABLE	Optimiza una tabla para agilizar la manipulación de registros
REPLACE	Reemplaza el contenido de una tabla
UPDATE	Actualiza registros de una tabla en una Base de Datos

Sentencias de Administración

FLUSH	Liberar memoria
GRANT	Otorgar permisos
KILL	Matar procesos
REVOKE	Quitar Permisos

Una desventaja de MySQL es la carencia de un rollback automático para cancelar todas las sentencias, si alguna de ellas falla. Si se tienen aplicaciones donde se requiera realizar transacciones financieras complejas, que impliquen el uso de varias sentencias interrelacionadas que deben ejecutarse obligatoriamente como un grupo o que sino no lo harán, entonces se debería considerar otro tipo de Base de Datos.

1.3 SISTEMAS DISTRIBUIDOS

Un sistema distribuido es un conjunto de Bases de Datos ubicadas físicamente en sitios distintos, esto puede implicar que se encuentren geográficamente en lugares diferentes; "Una colección de datos, los cuales se encuentran distribuidos en diferentes localidades interconectadas por una red de comunicaciones. Es necesario aclarar que cada localidad constituye un sistema de Bases de Datos; es decir, cada localidad tiene sus Bases de Datos locales, cuenta con un sistema administrador de Base de Datos, un administrador local de comunicación de datos y los programas necesarios para la administración de transacciones como son programas de bloqueo, bitácora, recuperación, etcétera. Cada localidad tiene autonomía propia y puede ejecutar aplicaciones locales; también participa en la ejecución de al menos una aplicación global, por lo cual se requiere acceder datos en diferentes localidades usando un sistema de comunicaciones.

En una Base de Datos distribuida, el usuario puede tener acceso tanto a la información de la localidad donde se encuentra, como a la información que se encuentra en el resto de las localidades. También es conveniente aclarar que las localidades de una Base de Datos distribuida pueden estar dispersas de forma física en una área geográfica extensa (red de larga distancia) o en una área reducida (red de área local). El sistema de comunicación utilizado depende en alto grado del tipo de red; por ejemplo, en una red de área local, las conexiones más comunes son cables dobles trenzados, coaxiales de banda base, coaxiales de banda ancha y fibra óptica; en una red de larga distancia, los medios más comunes son las líneas telefónicas, conexiones de microondas y canales de satélites"[22].

---

[22] Cisneros González, José Luis. PANORAMA SOBRE BASES DE DATOS (UN ENFOQUE PRACTICO), p.p. 147-186, Universidad Nacional Autónoma de Baja California, 1998

Los datos en un sistema de Bases de Datos distribuidos son almacenados a través de diferentes sitios, y cada sitio es comúnmente administrado por un DBMSs que puede correr independientemente de otros sitios; algunas propiedades de las Bases de Datos distribuidas son:[23]

Independencia de los datos distribuidos: Los usuarios deben ser capaces de pedir consultas sin especificar de donde las relaciones son referenciadas.

Atomicidad de transacciones distribuidas: Los usuarios deben ser capaces de escribir transacciones que acceden y actualicen datos en muchos sitios simplemente como si ellos escribieran transacciones sobre datos puramente locales. En particular, los efectos de una transacción a través de diferentes sitios deben continuar siendo atómicas, esto es, todos los cambios persisten si la transacción se compromete (commit), y no persisten si aborta (rollback).

Cisneros propone 12 principios de las Bases de Datos distribuidas[24] :

El principio fundamental de las Bases de Datos distribuidas consiste en que los usuarios deberán comportarse exactamente igual como si el sistema no estuviera distribuido. A este principio también se le conoce como "regla cero de los sistemas distribuidos", el cual conduce a las 12 reglas secundarias, las cuales son las siguientes:

[23] Ramakrishnan, Raghu; Gehrke Johannes, DATABASE MANAGEMENT SYSTEMS, p.p. 607, Mc Graw Hill, 2000

[24] Cisneros González, José Luis. PANORAMA SOBRE BASES DE DATOS (UN ENFOQUE PRACTICO), p.p. 147-186, Universidad Nacional Autónoma de Baja California, 1998

1. **Autonomía local.** Las localidades deben ser autónomas; es decir, que todas las operaciones realizadas en una localidad sean controladas en esta localidad. Es necesario señalar que una localidad no puede ser totalmente autónoma, ya que existen varias situaciones en las cuales una de ellas debe ceder el control a otra, perdiendo así, parte de su autonomía.

2. **No dependencia de una localidad central.** Este principio es un corolario del primero. Es indeseable la dependencia de una localidad central por dos razones: primero, el sistema sería vulnerable si la localidad central sufriera un desperfecto, y segundo, la localidad central llegaría a ser un cuello de botella.

3. **Operación continua.** Se refiere a que el sistema nunca deberá suspenderse para realizar alguna función, como puede ser añadir una nueva localidad o instalar la versión mejorada del DBMS. Es decir, el sistema debe mantener su funcionamiento de manera constante.

4. **Independencia con respecto a la localización (también conocida como transparencia de localización).** Para los usuarios, el sistema deberá comportarse como si todos los datos estuvieran almacenados en su propia localidad, esto simplifica la realización del trabajo a los usuarios, ya que no requieren conocer dónde se encuentran almacenados físicamente los datos.

5. **Independencia con respecto a la fragmentación.** La fragmentación se refiere a la división física de los datos de una tabla, la cual es deseable por razones de desempeño, de tal forma que los datos pueden almacenarse en la localidad donde sean más utilizados, con el fin de reducir el tráfico de la red y proporcionar un mejor servicio de acceso a la información.

6. **Independencia de réplica (también llamada transparencia de réplica).** Consiste en que, desde el punto de vista de usuario, la información se encuentra en una sola Base de Datos sin estar consciente del posible uso de una réplica de la misma. Esto es importante, ya que las aplicaciones pueden operar sobre copias locales en vez de tener que comunicarse con localidades remotas, mejorando la disponibilidad de la información. La desventaja principal de las réplicas consiste en que al realizar una transacción en una de ellas, ésta deberá actualizarse en todas las demás réplicas.

7. **Procesamiento distribuido de consultas.** La realización de consultas en una Base de Datos distribuida implica la transmisión de mensajes entre las localidades. La optimización del procesamiento distribuido de consultas se ve afectada por la manera como los datos son transmitidos desde una localidad inicial a una localidad final.

8. **Manejo distribuido de transacciones.** El manejo de transacciones cuenta con dos aspectos principales: el control de recuperación y el control de concurrencia. En el control de recuperación, para asegurarse de la atomicidad de una transacción (se refiere a que todas o ninguna de las operaciones de una transacción se realicen. La atomicidad requiere que si una transacción se interrumpe por una falla, el resultado parcial se deshaga), el sistema deberá verificar que todas las operaciones correspondientes a esa transacción se comprometan (realicen commit), o bien retrocedan (realicen rollback) a un mismo tiempo; esto se lleva a cabo mediante el protocolo de commit a dos fases. El control de concurrencia se realiza mediante la utilización de bloqueos.

9. **Independencia con respecto al equipo.** Debido a la gran variedad de equipos que existen, es conveniente poder integrar los datos en

diferentes equipos, y lograr que el usuario vea la información como si fuera un solo conjunto de datos almacenado en la misma localidad. Esto se logra al ejecutar el mismo sistema administrador de Base de Datos en diferentes equipos.

10. **Independencia con respecto al sistema operativo.** Así como existe una gran variedad de equipos, también existen en el mercado diversos sistemas operativos, por lo cual este principio señala la importancia de ejecutar el mismo sistema administrador de Bases de Datos en diferentes sistemas operativos.

11. **Independencia con respecto a la red.** Si es posible que el sistema funcione con equipos diferentes, sistemas operativos diversos y múltiples localidades, también es conveniente poder manejar varias redes de comunicación distintas. La red de comunicación utilizada influirá en la velocidad en que los datos sean transmitidos, por lo tanto, resulta importante elegir el medio de comunicación óptimo.

12. **Independencia con respecto al sistema administrador de la base de datos.** En un sistema de Base de Datos distribuida, no es necesario contar los mismos sistemas administradores de Bases de Datos, sólo es necesario contar con la misma interfaz. Por ejemplo, si tanto Ingres como Oracle manejan la norma oficial de SQL, podría ser posible lograr una comunicación entre las dos localidades; es decir, el sistema distribuido podrá ser heterogéneo hasta cierto grado. Un sistema distribuido ideal deberá cumplir este principio.

El doceavo principio de los sistemas distribuidos es la base de esta tesis; pues como se mencionó anteriormente es lo que se busca con este prototipo lograr una conexión heterogénea entre diferentes manejadores de Bases de Datos, invirtiendo así lo mínimo en recursos.

Un aspecto muy importante es la diferenciación entre Bases de Datos distribuidas y Bases de Datos centralizadas; una Base de Datos distribuida tiene toda la información ubicada en distintos lugares físicos, las bases centralizadas almacenan toda su información en un solo sitio por lo cual el tráfico de red es mayor, además que en este tipo de Base de Datos se busca evitar la redundancia, por el almacenamiento en disco, y la inconsistencia de datos; a su vez, las Bases de Datos Distribuidas muestran una gran ventaja que es deseable que se tenga en este tipo de arquitectura: la replicación de datos.

Las razones por las cuales es necesario distribuir los datos, así como las ventajas y las desventajas de esta arquitectura, se muestran a continuación[25]; existen tres razones por las cuales es conveniente distribuir los datos:

1. **Razones organizacionales y económicas.** La mayoría de las organizaciones se encuentran distribuidas, por lo menos desde el punto de vista lógico (departamentos proyectos), y probablemente en el sentido físico (plantas, talleres, laboratorios), lo cual trae como consecuencia que la información también se encuentre distribuida, ya que cada unidad de trabajo dentro de la organización contará con los datos necesarios para su propio funcionamiento. Es por ello que las Bases de Datos Distribuidas permiten que la estructura de la Base de Datos refleje la estructura de la organización (los datos locales se mantienen en la misma ubicación, donde por lógica deben estar, pero al mismo tiempo, es posible obtener acceso a datos remotos en caso necesario).

[25] Cisneros González, José Luis. PANORAMA SOBRE BASES DE DATOS (UN ENFOQUE PRACTICO), p.p. 147-186, Universidad Nacional Autónoma de Baja California, 1998

2. **Interconexión de las Bases de Datos ya existentes.** Las Bases de Datos distribuidas son la solución natural cuando muchas Bases de Datos ya existen en la organización y surge la necesidad de ejecutar aplicaciones globales. Esto requiere cierto proceso de reestructuración; sin embargo, el esfuerzo que se necesita para la reestructuración es menor del requerido para la creación de una nueva Base de Datos centralizada.
3. **Crecimiento continuo.** Las Bases de Datos distribuidas soportan un continuo crecimiento con un mínimo impacto sobre las unidades ya existentes, de tal manera que éstas resultan ideales si una organización crece agregando nuevas unidades organizacionales autónomas.

El principal apoyo que proporcionan las Bases de Datos distribuidas a una organización, consiste en que cada aplicación deberá ser capaz de trabajar en forma transparente con datos dispersos en varias localidades, las cuales cuentan con sus propias Bases de Datos locales, usuarios, DBMS, administrador local de comunicación de datos y equipo de cómputo, apoyadas por diversos sistemas operativos, donde cada una de las localidades se encuentran conectadas entre sí mediante diversas redes de comunicación.

VENTAJAS DE LAS BASES DE DATOS DISTRIBUIDAS

a) La principal ventaja de las Bases de Datos distribuidas consiste en que permiten que la estructura de la Base de Datos refleje la estructura de la organización; es decir, cada localidad se encontrará donde por lógica le corresponde y al mismo tiempo, podrá obtener acceso a datos ubicados en localidades remotas en caso de ser requeridos.
b) La asignación de datos en diferentes localidades permite disminuir el tiempo de transferencia de datos entre una localidad y otra, dando como resultado la reducción de costos. Por otra parte, el acceso local

de datos mejora el tiempo de respuesta si la transmisión remota se elimina.

c) Debido a la independencia y autonomía que existe entre cada una de las localidades, proporcionadas por la réplica de datos en caso de falla de uno de ellos, las demás localidades podrán continuar sus actividades normales sin que se vean afectadas en su funcionamiento, lo cual proporciona a los usuarios un mayor control sobre los datos.

d) Si una consulta requiere datos de varias localidades, es posible dividir la consulta en varias sub consultas que se ejecuten en paralelo en distintas localidades.

DESVENTAJAS DE LAS BASES DE DATOS DISTRIBUIDAS

Al llevar a cabo la distribución de datos, es necesario tomar en cuenta que esto traerá consigo ciertas desventajas; la principal de ellas es la complejidad que se requiere para garantizar una coordinación adecuada entre las localidades.

La complejidad de la distribución de datos se ve reflejada en las siguientes desventajas:

a) **Costo de desarrollo de software.** El costo para desarrollo de software para Bases de Datos distribuidas es elevado por la complejidad de su estructura.
b) **Mayor posibilidad de errores.** Puesto que las localidades del sistema distribuido operan en paralelo, resulta difícil garantizar que los algoritmos funcionen de manera correcta, por lo cual existe la posibilidad de pequeños errores.

c) **Mayor tiempo extra de procesamiento.** El intercambio de mensajes y los cálculos adicionales necesarios para coordinar la comunicación entre localidades requiere la inversión de tiempo extra, lo cual no ocurre en los sistemas centralizados.

1.4 REPLICACIÓN

Replicación significa duplicar información entre una o más Bases de Datos; esto es transferir información que se requiere a algún otro sitio en un sistema distribuido; esto con el fin de satisfacer las necesidades mismas del sistema; "una de las razones de esto es que si un sitio falla, puede haber otros sitios que proporcionen los mismos datos que se perdieron en el sitio de falla. un segundo uso de esto es una mejora en la rapidez de respuesta de una consulta haciendo una copia de los datos necesarios disponibles en los sitios donde las consultas son iniciadas"[26]:

A continuación se presentan dos razones para la replicación, y algunos métodos de replicación actuales[27]:

[26] García Molina, Héctor; D. Ullman Jeffrey, Widom; DATABASE SYSTEM IMPLEMENTATION, p.p. 568-571, Prentice Hall, 2000

[27] Ramakrishna, Raghu; Gehrke Johannes, DATABASE MANAGEMENT SYSTEMS, p.p. 596-611, Mc Graw Hill, 2000

Incrementar la disponibilidad de los datos: Si un sitio que contiene una réplica se cae, podemos encontrar los mismos datos en otros sitios. De manera similar, si las copias locales de las relaciones remotas están disponibles, somos menos vulnerables a una falla de ligas de comunicación.

Evaluación Rápida de consultas: Las consultas pueden ejecutarse de forma más rápida por medio de una copia local de la relación en lugar de ir hasta el sitio remoto.

Existen dos tipos de replicación, llamados replicación síncrona y replicación asíncrona las cuales difieren principalmente en como se mantienen las replicas concurrentes cuando la relación es modificada.

REPLICACIÓN SÍNCRONA

Existen dos técnicas básicas para asegurar que las transacciones ven el mismo valor, sin considerar en cual copia de un objeto accedan. En la primera técnica, llamada voting, una transacción debe escribir en una mayoría de copias en orden para modificar un objeto y leer al menos las suficientes copias para asegurarse que una de las copias es actual. Por ejemplo, si hay 10 copias y 7 copias son escritas por transacciones de actualización, entonces al menos 4 copias deben leerse. Cada copia tiene un número de versión, y la copia con el número de versión más alto es la actual. Esta técnica no es tan atractiva en muchas situaciones porque leer un objeto requiere leer múltiples copias, en muchas aplicaciones los objetos son leídos más frecuentemente de lo que son actualizados, y el rendimiento eficiente en la lectura es muy importante. En la segunda técnica, llamada "lee cualquiera escribe todos", para leer un objeto, una transacción puede leer cualquier copia, pero para escribir en un objeto, debe escribir en todas las copias. La lectura es rápida especialmente si tenemos una copia local, pero la escritura es lenta, relativamente en comparación con la primera

técnica. Esta técnica es atractiva cuando leer es mucho más frecuente que escribir, y esto es usualmente adoptado para implementar replicaciones síncronas.

Una alternativa aprovechable de replicación, llamada replicación asíncrona, ha llegado a ser ampliamente usada por DBMSs comerciales. Copias de una relación modificadas son actualizadas solo periódicamente en este proceso, y una transacción que lee diferentes copias de la misma relación debe ver diferentes valores. Así, la replicación asíncrona compromete la independencia de los datos distribuidos, pero esto puede ser más eficientemente implementado que la replicación síncrona.

REPLICACIÓN ASÍNCRONA

La replicación síncrona trae consigo un costo considerable. Antes que una transacción de actualización pueda comprometerse, debe obtener bloqueos exclusivos en todas las copias - Asumiendo que la técnica "leo cualquiera escribo todos" sea usada- de los datos modificados. La transacción debe enviar requerimientos de bloqueo a todos los sitios remotos, y esperar a que los bloqueos sean ejecutados, y durante este período potencialmente largo, está continua realizando otros bloqueos. Si el sitio o la comunicación falla, la transacción no puede comprometerse hasta que todos los sitios en los cuales se hayan modificado los datos se recuperen y sean alcanzables. Finalmente, aun si los bloqueos se obtienen leyendo y no hay fallas, comprometer una transacción requiere muchos mensajes adicionales para que sean enviados como parte de un protocolo de compromiso. Por esas razones, la replicación sincronía es indeseable o aún inalcanzable en muchas situaciones. La Replicación asíncrona esta ganando popularidad, aun cuando esto permita diferentes copias del mismo objeto para tener diferentes valores por cortos períodos de tiempo.

Esta situación viola el principio de la independencia de los datos distribuidos; los usuarios deben conocer que copia están accesando, dándose que las copias son inactivas solo periódicamente, y vivir con este reducido nivel de consistencia en los datos.

1.5 RELACIÓN ENTRE REPLICACIÓN Y SISTEMAS DISTRIBUIDOS

La replicación necesariamente se aplica a un sistema distribuido, incluso la replicación es una forma de mejorar el rendimiento en un sistema distribuido independientemente de la forma en que se replique; para aclarar más el concepto de replicación y describir su funcionamiento se exponen a continuación algunos problemas[28]:

Problemas:

1. Un banco tiene muchas sucursales. Cada sucursal (o el grupo de sucursales de una determinada ciudad) contendrá una Base de Datos de la contabilidad que se lleva a cabo en esa sucursal (o ciudad). Los clientes pueden escoger a un banco en cualquier sucursal, pero normalmente el banco en "su" propia sucursal, contendrá su propia información contable almacenada. El banco tendrá también datos que serán almacenados en la oficina central, como los registros de empleados y pólizas así como las tasas de interés actuales.

2. Una cadena de tiendas departamentales tiene muchas tiendas individuales. Cada tienda (o un grupo de tiendas en una ciudad) tiene una Base de Datos de ventas de esa tienda y un inventario de esa tienda. Debe haber también

[28] los ejemplos fueron extraidos de: García Molina, Héctor; D. Ullman Jeffrey, Widom; DATABASE SYSTEM IMPLEMENTATION, p.p. 568-571, Prentice Hall, 2000

una oficina central con datos acerca de los empleados, el inventario total de la cadena, las tarjetas de crédito de los clientes, e información acerca de suministros como ordenes sin llenar (pedidos vacíos), y lo que se adeuda.

En suma, debe haber una copia de toda los datos almacenados en las tiendas en un <<almacén de datos>> <data warehouse>, el cuál es usado para analizar y predecir la ventas a través de las consultas realizadas por analistas.

3. Una biblioteca digital puede estar constituida por un consorcio de universidades donde cada una contenga una Base de Datos de libros en línea y otros documentos. Buscando en cualquier sitio se puede examinar el catálogo de documentos disponibles para todos los sitios y entregar una copia electrónica del documento a cualquier usuario si es que el sitio la tiene.

Soluciones mediante replicación:

1. Un banco hace copias de las pólizas de tasas de interés actuales disponibles para cada sucursal, así que la consulta acerca de las tasas no tiene que ser enviada hasta la oficina central.

2. Una cadena de tiendas puede hacer copias de información acerca de los suministros de cada tienda, así que los requerimientos locales para información acerca de los suministros (ejemplo: el manejador necesita el número de teléfono de un suministro para checar lo referente a un embarque) pueden ser manipulados sin enviar mensajes a la oficina central.

3. Una biblioteca digital puede almacenar temporalmente una copia de un documento popular de una escuela donde a los estudiantes se les ha asignado la tarea de leer ese documento.

Conclusiones del Capítulo 1.

Muchos de los conceptos que se describen en este capítulo, son la base para la correcta comprensión de los capítulos siguientes.

Hasta aquí se ha explicado la evolución de las Bases de Datos, y de manera muy general algunos de los manejadores de Bases de Datos contemporáneos, además se dio una visión general de lo que es un sistema de replicación.

Las Bases de Datos han evolucionado y se han convertido de archivos planos a estructuras completas y relacionadas, que soportan distintos tipos de datos. Las principales características de un Sistema Manejador de Base de Datos son: la capacidad de administrar gran cantidad de datos, organizar datos de forma coherente, acceso compartido a múltiples usuarios entre otras.

Un sistema distribuido es un conjunto de Bases de Datos ubicadas físicamente en sitios distintos, esto puede implicar que se encuentren geográficamente en lugares diferentes; cada localidad se encontrará donde por lógica le corresponde y al mismo tiempo, podrá obtener acceso a datos ubicados en localidades remotas en caso de ser requeridos también este sistema permitirá disminuir el tiempo de transferencia de datos entre una localidad y otra, dando como resultado la reducción de costos.

Replicación significa duplicar información entre una o más Bases de Datos; esto es transferir información que se requiere a algún otro sitio en un sistema distribuido; esto con el fin de satisfacer las necesidades mismas del sistema.

CAPÍTULO II ELABORACIÓN DEL PROTOTIPO

2.1 BASES DEL SISTEMA DE REPLICACIÓN PROPUESTO

DEFINICIÓN DE ODBC

Hace quince años, la idea de usar un software de aplicación para acceder a una base de datos y, sin programar, usar la misma aplicación para acceder a otros tipos de Bases de Datos no era tomada en cuenta.

ODBC es un acrónimo de Conectividad Abierta de Bases de Datos, es una Interfaz de programación de aplicaciones[1] estándar para acceder datos en Sistemas Administradores de Bases de Datos relacionados y no relacionados. Usando el API de ODBC, las aplicaciones pueden acceder datos en una gran variedad de computadoras personales, mini computadoras, y mainframe.

Existen muchas equivocaciones acerca de ODBC en el mundo de la computación. Para el usuario final, puede ser simplemente otro acrónimo de computación. Para el desarrollador de aplicaciones, es una librería para acceder a rutinas de una API.

Primeramente, ODBC es una especificación para un conjunto de APIs de interfaz de nivel de llamada de acceso a datos. Este conjunto de APIs mantiene la misma independencia del origen de datos, sistema operativo, o lenguaje de programación de alto nivel usado (aunque muchas funciones trabajan con "tipos de datos de lenguaje C", las APIs de ODBC son un lenguaje independiente).

---

[1] API (Application programming interface): un conjunto de funciones relacionadas que el programador de computadoras usa para obtener algún tipo de servicio desde otra pieza de software. Se desconoce el funcionamiento interno de las funciones pero se sabe bien que resultados producirán y se proporciona un punto de entrada para mandar a llamar a la función que es propiamente la función API.

ODBC es basado en la especificación de interfaz de nivel de llamada del grupo "X/Open SQL Access", algo así como un estándar ISO ó ANSI[2].

Aunque las funciones API de ODBC son llamadas desde dentro de una aplicación, las funciones por si mismas son actualmente implementadas por controladores (drivers) específicos del origen de datos. Esto permite a las aplicaciones acceder a datos en dicho origen de una manera independiente. El uso de controladores también proporciona la habilidad de acceder a múltiples orígenes de datos simultáneamente, sin embargo el código requerido para hacer esto puede ser bastante complejo.

Se debe tomar en cuenta la ventaja que representa ODBC al permitir a desarrolladores de aplicaciones distribuidas acceder a múltiples DBMSs, en las cuales el cliente local y servidor de almacenamiento son usados en el sistema de producción final. Esta adaptación esta llegando a ser más común desde que los datos son almacenados en laptops y PDA (personal digital assistant, o más conocidas como palm top) y los datos son resincronizados en servidores corporativos por la noche. La situación es también un problema en la amplia área de ambientes en redes en los cuales los cambios a los datos deben propagarse a diferentes DBMSs en diferentes plataformas. Mientras la solución adecuada es la que brinda la replicación, y así mismo presente prototipo busca proporcionar una solución, una API común a múltiples orígenes de datos puede hacer este problema técnico menos atemorizante cuando múltiples DBMSs están involucrados[3].

---

[2] X/ Open SQL Access : Un consorcio industrial de vendedores de DBMSs, cuyo objetivo es habilitar productos basados en SQL de muchos vendedores para que trabajen juntos.

[3] Kyle Geiger; ISIDE ODBC, p.p. 20-21, Microsoft Press, 1995

ARQUITECTURA Y ESTRUCTURA BÁSICA DE ODBC

El Modelo Cliente Servidor

La arquitectura cliente servidor se enfoca en dividir una aplicación en dos partes poniendo esas partes en dos máquinas diferentes, por lo menos el servidor debe tener comunicación con cada aplicación que se ejecute. Esto permite al DBMS residir en una computadora o mainframe servidor donde el poder de computo y control centralizado puede ser usado para proporcionar rápidamente, acceso coordinado a datos mientras la aplicación lógica reside en una o más PCs (cliente) esto puede hacer efectivo el uso de todos los recursos que la PC pueda ofrecer sin causar un cuello de botella en el servidor. Aparentemente este modelo solo tiene dos componentes: el cliente y el servidor, sin embargo existe otro componente muy importante el protocolo de datos que proporciona la capa de comunicación entre el cliente y el servidor.

El rol del cliente

En un sistema cliente-servidor, la aplicación lógica reside y corre en el cliente. La aplicación lógica es responsable de administrar la información que se muestra en la pantalla, procesamiento de entrada del usuario, e interactuar con el servidor.

En comparación con otras aplicaciones, una aplicación cliente servidor no usa entradas y salidas primitivas (I/O, input/output) a archivos para comunicarse con los orígenes de datos. En lugar de eso utiliza un mecanismo totalmente diferente de programación- el API de la Base de Datos. Aunque existen alternativas, el API de la base de datos es básicamente una combinación de SQL y algún tipo de interfase de programación (como CLI[4]) que envía sentencias SQL a un origen de

[4] CLI (call level interface): Interfaz de nivel de llamada, un conjunto de funciones (como API definida previamente). Sin embargo, CLI es usada en los estándares SQL mundiales para describir una interfase que no sea SQL incrustado.

datos para su procesamiento. El API de la Base de Datos proporciona un nivel de abstracción superior que las entradas y salidas primitivas (I/O, input/output) a archivos.

Debajo del API de la base de datos, está la interfaz de comunicación de red. Este subcomponente trabaja con la capa de transporte de red para transmitir y recibir datos de una estación de trabajo o servidor. La interfaz de comunicación de red es otra capa de abstracción que esconde de la aplicación lógica cliente-servidor los detalles de un software de red en particular que se este usando (por ejemplo, Novell NetWare, TCP/IP, o SNA)[5].

El rol del Servidor

En un sistema cliente-servidor, el DBMS reside y corre en un servidor. El DBMS interactúa con uno o más clientes mediante una interfaz de comunicación de red similar a la interfaz de comunicación de red usada en el cliente. En suma a la interfaz de comunicación de red, el DBMS también contiene muchos otros subcomponentes como un programador de tareas, un analizador gramatical, previamente analizados en el capítulo anterior, que permiten procesar peticiones de la aplicación cliente-servidor en la estación de trabajo cliente.

El DBMS controla todo el acceso a datos.

[5] Sanders Roger E.; ODBC 3.5 DEVELOPER'S GUIDE, p.p. 20-21, McGraw Hill, 1998

El siguiente diagrama, según Sanders, muestra la arquitectura básica del modelo cliente- servidor:

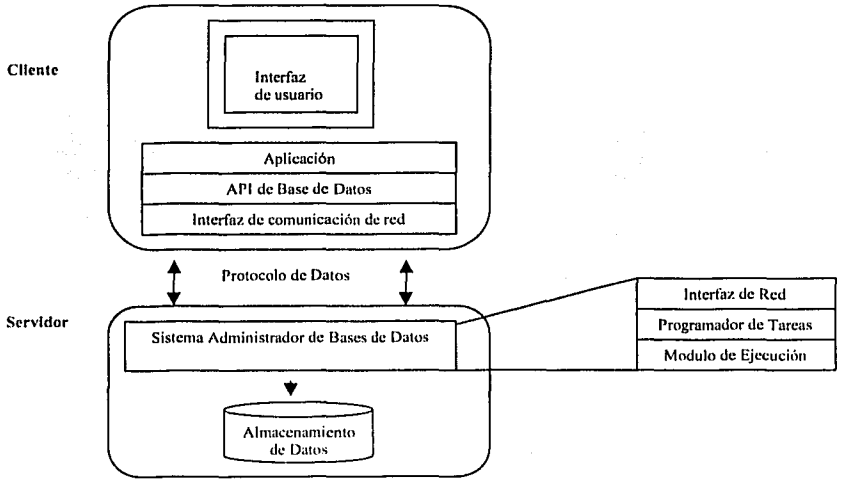


Figura 2-1 Los tres principales componentes de la Arquitectura Cliente Servidor, junto con algunos de sus subcomponentes.

Antes de iniciar con el análisis de los componentes de la Arquitectura ODBC, es importante mencionar la información acerca de las librerías dinámicas de Windows.

El primer paso en el diseño de programas es la reunión en librerías de aquellas clases, funciones o datos susceptibles de ser reutilizados. Las ventajas de disponer de librerías son claras: disminuyen el riesgo de alterar los archivos fuente y aceleran el proceso de compilación, entre otras. Cuando se habla de librerías a secas se entiende que son librerías estáticas. Las librerías estáticas no son más que una colección de archivos objeto (.obj), reunidos en un único archivo (.lib) que facilitan su manejo. Así, en el proceso de enlazado del programa se puede (linker)

buscar referencias a símbolos tanto en archivos de código objeto (.obj) como en las librerías estáticas. Únicamente cuentan con un inconveniente y es que cuando varios programas se enlazan con la misma librería estática, se incluyen copias de dicha librería en cada uno de ellos con el consecuente espacio en disco. Además, cuando se ejecutan todos los programas habrá un consumo innecesario de memoria, puesto que existen réplicas del código de la librería en cada ejecutable[6].

Una de las finalidades de ODBC no es simplemente proporcionar acceso a múltiples orígenes de datos sino a múltiples orígenes de datos al mismo tiempo. Esto inmediatamente crea una dificultad técnica: ¿cómo puede una aplicación llamar a la misma función que se debe ejecutar en diferente código? Cualquier desarrollador sabe que si se define una función dos veces, el analizador de código mandara un error diciendo algo así como, "Símbolo duplicado definido". Para resolver este problema, podemos acudir a la arquitectura de Windows que permite el enlace dinámico de librerías (de aquí el nombre, enlace dinámico de librerías del inglés dynamic link library o DLL), es decir estas librerías son librerías de objetos compartidos.

Un programa que usa enlace dinámico puede cargar y usar otro programa. Windows por si mismo confía con exceso en el enlace dinámico. Las aplicaciones basadas en Windows llaman funciones para optimizar el pintado de la pantalla, entrada y salida en disco (I/O), procesamiento de mensajes, y todos los demás servicios proporcionados por el sistema. Pero, por supuesto, las aplicaciones basadas en Windows no incluyen el conjunto completo de librerías de tiempo de ejecución en Windows. Mas bien, cuando una función es llamada por una aplicación, Windows determina cual librería dinámicamente enlazada contiene la llamada, carga la librería del disco en memoria, y llama la función en la librería. En

---

[6] Pascual Jorge, Charte Francisco, Segarra Miguel, Ángel de Antonio, Clavijo José, 684-685,Programación Avanzada en Windows 2000, 2000, McGraw Hill, España

Windows existen actualmente dos maneras por las cuales el enlazador (linker) puede enlazar una DLL a una aplicación. El primer método (carga implícita) es enlazar una aplicación a una librería importada que defina todos los puntos de entrada en la DLL. Cuando la aplicación es enlazada, la librería de entrada define todos los puntos de entrada para la DLL, pero ningún código de la DLL es incluido actualmente en el programa ejecutable (.EXE). Cuando la aplicación es ejecutada, Windows automáticamente busca a la DLL en disco y la carga en memoria. Si la DLL no esta presente, la aplicación por si misma no correrá. Desde el punto de vista de la aplicación, este tipo de enlace para DLL es el mismo que el enlazar una librería estática a la aplicación, excepto que el enlace no sucede hasta el tiempo de ejecución. La segunda manera de enlazar una DLL a una aplicación es la de cargar explícitamente la DLL en tiempo de ejecución. En este caso la aplicación tiene que manejar todo por sí misma. Tiene que cargar explícitamente la DLL por nombre usando la función de Windows "LoadLibrary". Una vez que la librería es cargada, la aplicación debe determinar cada punto de entrada en la DLL llamando otra función de Windows, "GetProcAddress". Cada punto de entrada puede ser cargado por el nombre de la función (de forma lenta) o por el número ordinal de la función (la forma rápida). Cada punto de entrada en la DLL debe ser almacenado en memoria así que puede ser llamado después a través de una llamada indirecta. Esto es, mientras que una llamada a una función normal de C es hecha incluyendo simplemente el nombre de la función en el programa de aplicación, una llamada a una función indirecta es hecha con un puntero referenciado a una variable que contenga la dirección que se quiera llamar[7].

Por ejemplo, una aplicación usando una llamada directa a una función, llamada "ObtenPuerto" sin argumentos, se llamaría así:

```
Porto = ObtenPuerto();
```

[7] Kyle Geiger; ISIDE ODBC, p.p. 104-105, Microsoft Press, 1995

A través de una función en una DLL el código sería así:

```
int  (*Porto) (void)  =  (int (*) (void)) GetProcAddress(hinst,"ObtenPuerto") ;
```

Componentes De Una Arquitectura ODBC[8]

ODBC fue diseñado basado en el modelo de arquitectura cliente-servidor. ODBC también fue diseñado con la suposición de una interfaz de programación que pueda enviar y recibir el protocolo de datos de un origen de datos que soporte SQL que funcionara y optimizara el rendimiento como si fuera un API nativo (precompilador SQL Generado) del origen de datos.

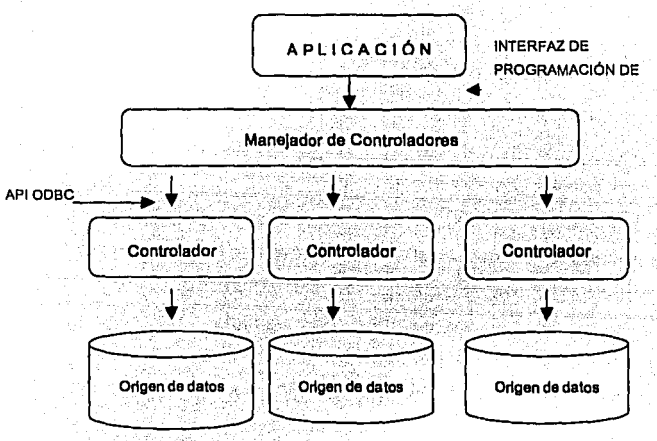


Figura 2-2 Arquitectura ODBC

[8] El resumen general de la arquitectura de ODBC, así como sus componentes, fue extraído de Sanders Roger E.; ODBC 3.5 DEVELOPER'S GUIDE, p.p. 22-115, McGraw Hill, 1998

Aplicación

Ejecuta y procesa llamadas a las funciones ODBC para someter sentencias SQL (lenguaje estructurado de consulta) y recuperar resultados.

Las aplicaciones son programas ejecutables que llaman a las funciones API de ODBC para acceder a los datos en uno o más orígenes de datos. Porque la mayoría de los datos trabajan con SQL, las aplicaciones usualmente hacen llamadas a las funciones API de ODBC para someter sentencias SQL a orígenes de datos y recuperar los resultados (si hay) generados cuando esas sentencias son ejecutadas. En suma para llamar a las funciones API de ODBC, las aplicaciones optimizan todos los trabajos externos a la interfaz ODBC.

Controlador

Procesa llamadas a las funciones ODBC, somete peticiones SQL a un origen de datos específico, y regresa los resultados a la aplicación. Si es necesario, el controlador modifica una petición de la aplicación de tal forma que se adapte a la sintaxis soportada por el DBMSs (manejador de Bases de Datos) asociado.

Los controladores son librerías que implementan funciones en el API de ODBC para un origen de datos especifico, por consiguiente no son intercambiables (esto es un controlador de ORACLE no puede ser usado para acceder datos de una base de datos de Dbase). Porque los controladores son hechos a la medida para un origen de datos simple, estos típicamente limitan la implementación del API de ODBC a las capacidades de su subcapa el origen de datos para el cual están escritos.

Los controladores proporcionan a las aplicaciones la habilidad de acceder a una variedad de orígenes de datos. Sin embargo, dado que las funciones del API de ODBC soportadas por cada controlador pueden variar, el desarrollador de una aplicación necesita considerar las funciones que son soportadas por el controlador (y cuales no) con el que esta trabajando. Afortunadamente, cada controlador debe

soportar dos APIs de ODBC que permitan a una aplicación determinar en tiempo de ejecución, las capacidades del API de ODBC y la gramática de construcción de SQL (y consecuentemente la subcapa de origen de datos) que se soporte.

En resumen los controladores son responsables de las siguientes tareas:

1. Conectarse y desconectarse del origen de datos.
2. Buscar funciones de error no buscadas por el administrador de controladores.
3. Inicializar transacciones.
4. Someter sentencias SQL al origen de datos para su ejecución. Si es necesario, el controlador convertirá el SQL ODBC a un SQL especifico del origen de datos antes de someterlo al origen de datos para su procesamiento. A menudo, esta conversión esta limitada a reemplazar secuencias de escape con el SQL apropiado y especifico del origen de datos.
5. Enviar datos y recuperarlos desde el origen de datos, incluyendo la conversión de tipos de datos como se especifique en la aplicación.
6. Mapear errores específicos del origen de datos a SQLSTATEs (estados SQL) de Odbc.

Niveles De Conformación De Controladores

Cuando el grupo ANSI creó el estándar SQL-92, entendieron que cada DBMS soporta un conjunto diferente de funcionalidad y sintaxis SQL. Por esta razón, el estándar SQL-92 define tres niveles distintos de funcionalidad SQL (entrada, intermedia y llena). El estándar ODBC va un paso más allá definiendo los niveles de conformación para controladores en dos áreas: El API de ODBC y la gramática SQL de ODBC (incluyendo los tipos de datos SQL ODBC). Puesto que el presente trabajo aprovecha la funcionalidad de los controladores ya creados no se ahondara en este tema.

Manejador De Controladores

Carga y descarga controladores (drivers) en nombre de la aplicación, procesa llamadas a las funciones ODBC o las pasa al controlador.

El manejador de controladores es una librería especial que administra la comunicación entre las aplicaciones y los controladores. Las Aplicaciones llaman a las funciones API ODBC en el manejador de controladores, y el manejador de controladores es responsable de rutear las llamadas al controlador apropiado. Cuando una aplicación trata de conectarse a un origen de datos, el manejador de controladores determina que controlador es requerido, lo carga (asumiendo que no este cargado previamente), almacena la dirección de cada llamada a la función API del controlador en memoria, y llama a la función API de conexión en el controlador (lo cual ocasiona que el controlador se inicie a sí mismo y que establezca una conexión a su subcapa el origen de datos). A partir de ese punto, el manejador de controladores simplemente examina cada llamada a las funciones hechas por la aplicación y, usando las direcciones almacenadas en memoria de las funciones API del controlador, llama a la función API correspondiente en el controlador (a menos que sea una llamada a una función que el manejador de controladores procese por sí mismo, como es el caso cuando la aplicación pregunta por el nombre del controlador). Cuando una aplicación se desconecta de un origen de datos, el manejador de controladores llama a la función API de desconexión en el controlador (lo cual causa que el controlador termine la conexión a su subcapa el origen de datos) y descargue el controlador correspondiente de memoria hasta que la última aplicación usada no la necesite.

En suma al descargar y cargar controladores, el manejador de controladores también optimizan algunos chequeos rudimentarios de error para asegurar que las funciones del API de ODBC se están llamando en el orden correcto y que los argumentos de las funciones API de ODBC contienen valores válidos.

Origen De Datos

Consiste en los datos a los que el usuario quiere acceder y su sistema operativo, DBMSs, y plataforma de red (si hay) usada para acceder al DBMSs asociados.

Un origen de datos puede ser una base de datos en un DBMS en particular, un archivo, una hoja de cálculo. Por ejemplo un origen de datos puede ser una Base de Datos ORACLE corriendo en Windows NT, accesada por Novel Netware, una colección de archivos Dbase en un directorio de un servidor, una Base de Datos de Microsoft Access.

COMPONENTES ESPECIALES DE UNA ARQUITECTURA ODBC

ODBC usa un conjunto especial de componentes para ayudar a la aplicación a comunicarse con uno o más orígenes de datos.

Handle[9]

En Windows, un handle es una simple variable puntero de aplicación que se refiere a un objeto de datos en el cual ODBC (o en Windows en general) puede almacenar información de contexto. ODBC usa cuatro tipos de handles.

- Handles de ambiente
- Handles de conexión
- Handles de sentencia
- Handles de descripción

Handle de ambiente

Es un puntero a un área de almacenamiento que contiene información ODBC específica que es global por naturaleza. Cada programa de aplicación que usa

[9] A pesar de que existe una traducción de handle, se decidió usar la propia palabra en inglés ya que implica la definición de un nuevo concepto Nota del Autor.

ODBC debe comenzar asignando espacio en un handle de ambiente y solo un handle de ambiente se le puede asignar espacio por aplicación. A un handle de ambiente se le debe de asignar espacio antes de que se le asigne a cualquier otro tipo de handle.

Handle de conexión

Es un puntero a una estructura de datos que contiene información acerca de una conexión a un origen de datos que ha comenzado ha administrarse por el manejador de controladores ODBC. Desde la perspectiva del controlador del origen de datos, un handle de conexión es usado para mantener una ruta de conexión de red a un servidor o de forma alterna mantener una ruta a un directorio que contenga información de archivos de datos locales. Desde la perspectiva del administrador de controladores, un handle de conexión es usado para identificar que controlador (driver) se usará y que origen de datos se usará con ese controlador.

Handle de sentencia

Es un puntero a una estructura de datos que contiene información acerca de una sentencia SQL única. El handle de sentencia es el verdadero caballo de fuerza de ODBC. Es usado para procesar todas las sentencias SQL contenidas en una aplicación. Cada sentencia SQL debe tener sus propio handle de sentencia y cada handle de sentencia usado puede ser asociado únicamente con un handle de conexión.

Handle Descriptor

Es un puntero a un área de almacenamiento de datos que contiene una colección de meta datos describiendo los parámetros de una sentencia SQL o las columnas de datos de un conjunto de resultados, esto es visto ya sea por la aplicación o el controlador.

PARTES DE UN PROGRAMA DE APLICACIÓN ODBC

Los programas de aplicación escritos por ODBC desempeñan tres distintas tareas:

1.- Inicialización

2.- Procesamiento de Transacciones

3.- Terminación

La tarea de inicialización

En la tarea de inicialización, el manejador de controladores es cargado y los recursos necesarios para la tarea de procesamiento de transacciones se les asigna espacio y se inicializan. Como mínimo, una aplicación ODBC debe asignar espacio para un área de almacenamiento de datos de ambiente y al menos un área de almacenamiento de datos de una conexión. Una vez que esas áreas de almacenamiento han sido asignadas y sus handles correspondientes inicializados, uno o ambos handle(s) son pasados al manejador de controladores (como argumentos) cuandoquiera que es hecha una llamada a una función API de ODBC. Durante la tarea de inicialización, una aplicación también le dice al manejador de controladores cuál especificación ODBC se planea seguir.

La tarea de procesamiento de transacciones

La tarea de procesamiento de transacciones constituye el volumen de una aplicación ODBC. Aquí es donde las sentencias SQL que consultan y/o modifican datos son pasadas al manejador de controladores (el cual las envía a el origen de datos vía el controlador del origen de datos) mediante muchas llamadas a las funciones API de ODBC. En la tarea de procesamiento de transacciones, una aplicación desempeña lo siguientes pasos, en el orden mostrado:

1. Reserva espacio para handles de sentencia
2. Prepara y ejecuta sentencias SQL

3. Procesa los resultados
4. Compromete o Neutraliza las transacciones (commit o roll back)
5. Libera los handles de sentencia

En el diagrama mostrado en la figura 2.3 se muestra la tarea de procesamiento.

Reservar espacio para handles de sentencia, como se había mencionado anteriormente un handle de sentencia se refiere a un objeto de datos que contiene información acerca de una sentencia SQL única. Esta información incluye el texto de la sentencia SQL, cualquier argumento dinámico de la sentencia SQL, información del cursor, parámetros que marcan argumentos obligatorios para sentencias SQL dinámicas, y columnas de conjuntos de resultados de datos, valores de resultado, e información de estatus. Los handles de sentencia reservan espacio haciendo llamada a la función SQLAllocHandle(). Un handle de sentencia debe asignar espacio para una sentencia SQL antes de que la sentencia sea ejecutada. También, cada handle de sentencia debe estar asociado con un handle de conexión específico de un origen de datos. Una vez que a un handle de sentencia le ha sido asignado espacio, existen dos métodos que pueden ser usados para especificar y ejecutar una sentencia SQL actual:

Sec.	Descripción
1.-	Preparar y Ejecutar: este método separa la preparación de las sentencias SQL de su actual ejecución y típicamente es usada cuando una sentencia SQL se ejecuta repetidamente (usualmente con valores de parámetros diferentes) o cuando la aplicación necesita información acerca de las columnas en el conjunto de resultados de datos producidos antes de que la sentencia SQL pueda ser ejecutada.

2.-	Ejecución Directa: Este método combina el paso de preparación y el paso de ejecución en un solo paso y es usado cuando una sentencia SQL es ejecutada solo una vez o cuando la aplicación no necesita información adicional acerca de las columnas en el conjunto de resultado de datos producidos antes de que la sentencia SQL pueda ser ejecutada.

Tabla 1-1 la tarea de inicialización

Unas de las tareas adicionales del Manejador de Controladores es la de realizar un seguimiento, opcional, de las transacciones que ejecuta el cliente; principalmente de las sentencias SQL que se ejecutan en el servidor, el seguimiento es el registro en un archivo de las transacciones que se ejecuten (incluso errores).

El seguimiento se lleva a cabo específicamente por una DLL de seguimiento que se encuentra instalada junto con el programa ODBC en cualquier sistema operativo de Windows, dicha DLL puede activar el seguimiento de manera sencilla, los pasos a seguir para la activación son los siguientes:

a) Entrar al panel de control.
b) Iniciar el programa de ODBC de 32 bits.
c) En la pestaña de traza (tracing en inglés) aparecerá el nombre y ruta del archivo de seguimiento en el cual quedarán registradas las transacciones (el nombre por defecto es SQL.LOG), en la misma pestaña se mostrará el nombre de la DLL de seguimiento (C:\WINNT\System32\odbctrac.dll ó C:\WINDOWS\SYSTEM\odbctrac.dll).
d) Dar clic en iniciar traza (start tracing en inglés).
e) Inicializar (comenzar a trabajar) con la aplicación.

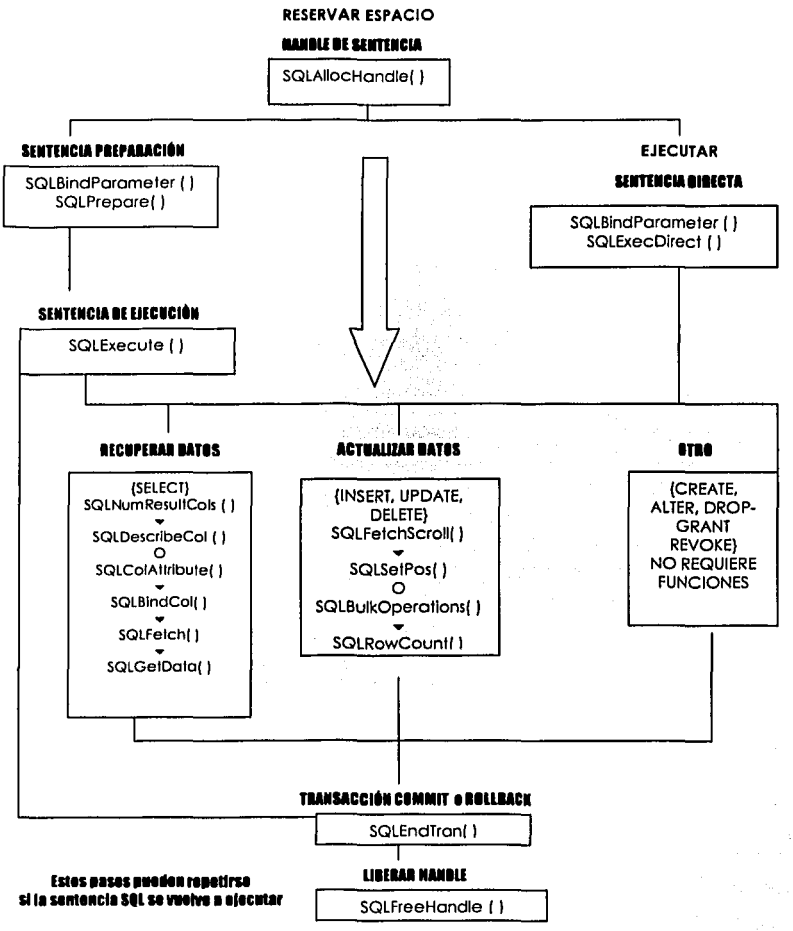


Figura 2-3 Orden Típico De Las Llamadas a las Funciones API De ODBC en una Tarea De Procesamiento ODBC (lo que se muestra en lo cuadros con las tres primeras letras SQL son las funciones de la API de ODBC, más básicas en una aplicación ODBC)

Al terminar de usar la aplicación se debe dar clic en detener seguimiento (stop tracing en inglés) e inmediatamente se puede consultar el archivo SQL.LOG generado.

FUNCIÓN DEL ARCHIVO DE SEGUIMIENTO EN UNA BASE DE DATOS CONECTADA MEDIANTE ODBC.

La DLL que desempeña el seguimiento de transacciones[10] es uno de los componentes principales de ODBC. La DLL de seguimiento debe ser instalada en el directorio de sistema, o fallará al cargarse.

La DLL hace el seguimiento de argumentos de entrada, de salida, códigos de retorno, y estados SQLSTATES. Cuando el seguimiento esta activado, el manejador de controladores llama a la DLL de seguimiento en dos puntos: uno en una función de entrada (antes de la validación de argumentos) y justamente después del retorno de las funciones.

Cuando una aplicación llama a una función, el manejador de controladores llama a una función en la DLL de seguimiento antes de llamar a la función en el controlador o procesar la llamada por sí mismo. Cada función ODBC tiene su función correspondiente[11] (con el prefijo Trace) que es idéntica a la función ODBC excepto por el nombre. Cuando se llama a la función de seguimiento, la DLL captura los argumentos de entrada y regresa un código de retorno. Dado que la DLL de seguimiento se llama antes de que el manejador de controladores valide los argumentos, las llamadas invalidas se registran, así que los estados de errores de transición y argumentos inválidos son registrados en el archivo.

---

[10] Extraído de Microsoft Data Access Components (MDAC) SDK, Nota del Autor

[11] Esto es dentro del código de la DLL obtenida mediante el Microsoft Data Access Components (MDAC) SDK en http://msdn.microsoft.com/downloads/, Nota del Autor

Después de llamar a la función de seguimiento en la DLL, el manejador de controladores llama a la función ODBC en el controlador. Esto entonces llama a la función TraceReturn en la DLL de seguimiento. Esta función toma dos argumentos: el valor retornado por la DLL de seguimiento para la función de seguimiento, y el código de retorno regresado por el controlador al manejador de controladores para la función ODBC (o el valor regresado por el mismo manejador de controladores si este es quien la procesa). La función usa el valor regresado para la función de seguimiento para manipular valores capturados de argumentos de entrada. Esta escribe el código retornado para la función ODBC en el archivo de seguimiento.

ARQUITECTURA BÁSICA DE COMUNICACIÓN EN SISTEMAS OPERATIVOS WINDOWS (WINSOCKETS).

A continuación se muestra un breve resumen de los principales componentes red en Windows[12].

DIRECCIONES IP

Los nodos de una red IP se conocen como hosts y pueden ser tanto origen como destino de datos. Todos los nodos de una red tienen una dirección IP que los identifican dentro de la red de forma única y que, a su vez, los clasifica como pertenecientes a una determinada zona dentro de la red.

El objetivo de la clase de red (o red lógica) es permitir considerar, de forma distinta, a los hosts situados en la misma red lógica que a los que están en otras. Esto permite, por ejemplo, optimizar el tráfico de la red o filtrar paquetes para que no entren o salgan de una determinada red.

---

[12] Pascual Jorge, Charte Francisco, Segarra Miguel, Ángel de Antonio, Clavijo José, 638-655,Programación Avanzada en Windows 2000, 2000, McGraw Hill, España

Las direcciones IP se definen como enteros de 32 bits. IP define cuatro clases de direcciones IP que se identifican de la siguiente forma:

Dirección IP: bits de Prefijo | bits de identificación | bits de hosts.

El prefijo identifica la clase de direcciones. Las direcciones IP cuentan con distinto número de bits para identificar redes y hosts en cada una de las clases.

El conjunto de direcciones que falta (prefijo 1111) se encuentra reservado para desarrollos futuros o redes experimentales.

Para facilitar su lectura, se suele representar por separado los bytes de la dirección agrupados en valores separados por puntos. La más conocida es la que agrupa los bits en cuatro grupos de 8:

10.0.0.1 ó 130.0.1.1

Clases de direcciones IP

Clase	Prefijo	bits red	bits host	Descripción
A	0	7	24	Permite configurar 2^7(128) redes de 2^{14} (16777216) computadoras
B	10	14	16	Permite configurar 2^{14}(16384) redes de 2^{16} (65536) computadoras
C	110	21	8	Permite configurar 2^{21}(2097152) redes de 2^8 (256) computadoras
D	1110	28	0	Permite configurar 2^{28}(268435456) direcciones de Multidestino

Tabla 1-2 clases de direcciones IP

La dirección 127.0.0.1 (loopback) está reservada para permitir el funcionamiento del protocolo dentro de la misma computadora. Esto permite que podamos desarrollar programas basados en red (como compartir archivos del explorador de Windows o navegadores), aunque no dispongamos de tarjeta de red en la computadora.

Las direcciones de red o de hosts con todos sus bits a uno indican todas las redes o todos las computadoras de una red. Este esquema se emplea en el envío de mensajes de localización de recursos (broadcast).

Subredes IP

Para especificar los bits de red y de host se emplean las máscaras de red que son palabras de 32 bits con los bits que corresponden al prefijo y a la red <<1>>, manteniendo en cero los bits del host. Las clases A, B y C tienen valores por defecto de mascara 255.0.0.0, 255.255.0.0 y 255.255.255.0 respectivamente.

Cuando se configura una nueva red de computadoras, se puede cambiar este esquema y hacer que se tengan más bits para la identificación de la red, independientemente de su clase. Así por ejemplo, una red tipo B se puede dividir en 512 subredes de hasta 128 computadoras cada una, haciendo que la máscara de red sea 255.255.255.128

Sistema de Nombres de Dominio (DNS)

DNS (Domain Name System) surgió como consecuencia de dos factores fundamentales: la necesidad de identificar hosts de forma amigable y la expansión del uso de Internet.

En un principio, y aún hoy para redes no publicas pequeñas, era corriente emplear archivos que almacenan una tabla (normalmente archivos hosts) donde se establece una correspondencia entre direcciones IP y nombres de máquinas

(hostname), puesto que es más sencillo recordar un nombre, por ejemplo hal9000, que su dirección IP, por ejemplo 10.231.253.104.

El problema surge con la necesidad de identificar muchas computadoras. DNS es un servicio distribuido en toda Internet que establece dominios de nombres que mantienen una jerarquía. La identificación de la computadora se hace no sólo por su nombre de host, sino indicando también los nombres de los dominios a los que pertenece. Funciona de forma similar a un árbol y sus archivos. Los archivos serían el equivalente a los hosts y los directorios a los nombre de dominio.

Para separar los nombres de dominio se emplea el punto ('.'), de modo que un nombre de computadora con su dominio será:

hal9000.scilabs.es

Datagramas

Los datagramas constituyen la unidad de información básica que maneja IP para transmitir información. Contienen una información de cabecera y los datos de las tramas de niveles superiores. La longitud máxima de un datagrama es de 65,535 bytes, de los cuales, como mínimo, 20 constituyen la información de cabecera.

Los datagramas se pueden a su vez fragmentar para transmitirlos sobre la red, en función de la unidad de transmisión máxima (MTU) que soporte el hardware de red. Para redes Ethernet este valor suele ser fijado en 1500 bytes.

Protocolos de nivel de transporte IP

IP proporciona en la actualidad dos protocolos de transporte, TCP/IP y UDP/IP, y la ampliación de UDP para admitir IP Multicast. En general, todos ellos proporcionan servicios que las aplicaciones necesitan para enviar datos de un extremo de la red a otro (u otros).

TCP/IP es orientado a la conexión y requiere, al igual que sucede con la telefonía, de un proceso de llamada y del establecimiento de un canal de comunicación para la transmisión de los datos. Es fiable y comprueba que los datagramas han llegado sanos y salvos a su destino.

Sockets

Son el par de números que designan una dirección IP y un puerto. En definitiva, son una identificación de un punto de comunicación al que puede acceder un proceso.

Para el sistema operativo, son algo más que la dirección IP y el puerto. Constituyen un auténtico dispositivo de entrada y salida que al igual que, por ejemplo los archivos, necesita ser mantenido mediante buffers de entrada y ser supervisado por el sistema operativo.

Transporte Tcp/Ip

Proporciona los servicios necesarios para la realización de la comunicación punto a punto bidireccional, garantizando el orden de los datos recibidos (byte-stream). Para ello emplea un socket (dirección IP + Puerto) que ofrece garantías de datos ordenados (STREAM). Para cada interfaz de red (con tarjetas de red o vía PPP telefónico) que tenga su propia dirección IP, el protocolo proporciona un conjunto completo de puertos que pueden ser solicitados por los procesos para establecer canales de comunicación.

Los sockets TCP se crean con la función socket() indicando el tipo SOCK_STREAM y la familia de direcciones AF_INET. Los sockets TCP pueden ser activos o pasivos. Los activos inician el proceso de establecimiento de la conexión. Los sockets pasivos, en cambio, reciben solicitudes de conexión. Cuando se crea un socket TCP, por defecto se crea como activo. Para convertirlo a pasivo se debe asociar con una dirección IP y un puerto libre mediante la

función bind() y emplear la función listen(). La dirección deberá ser una de las posibles redes a las que se encuentre conectada la computadora (puede tener varias tarjetas de red y algunas conexiones telefónicas como posibles direcciones de escucha). Sólo los sockets pasivos pueden emplear la función accept() para aceptar peticiones de conexión que realizan sólo los sockets activos llamando a connect(). Es en connect() donde indican los sockets activos la dirección y el puerto de socket pasivo con el que se desean conectar. Establecida la conexión se emplean send() y recv() para la emisión y recepción de datos. Los sockets pasivos pueden emplear direcciones especiales (wildcard addresses) para admitir conexiones provenientes desde varias redes. De este modo, un proceso puede atender peticiones de servicio de hosts que están en múltiples redes. Si se desea que un socket atienda peticiones procedentes de cualquier dirección de red se le asocia a la dirección INADDRR_ANY, a la vez que se especifica su puerto de escucha. Cuando se acepta y establece una conexión con accept(), se crea un nuevo socket para realizar sobre él la comunicación.
A esté se le asigna la dirección de la interfaz de red por la que le llegan los datos y se le asigna un puerto libre.

A continuación se muestra un resumen de las funciones utilizadas en Visual C++ para manipular sockets:

Función	Descripción
Socket()	Con Socket( ) se crean todos los tipos posibles de sockets.
CloseSocket()	Cuando no sean necesarios se deben cerrar con CloseSocket() para liberar recursos del sistema.
Listen( )	Especifica el número (máximo 5) de peticiones de conexiones que pueden recogerse y dejar como pendientes, hasta que sean aceptadas con accept( ), por un socket de tipo SOCK_STREAM que se deja en <<escucha>> (hace las veces de operadora de teléfono).

Función	Descripción
Accept( )	Acepta una conexión de SOCK_STREAM cliente, cuando hay conexiones pendientes, creando un nuevo socket para la comunicación select( ) junto con la ayuda de las macros FD_ZERO( ) Y FD_SET, permite inspeccionar el estado de los buffers de entrada y de salida de uno o varios sockets y esperar mientras no se den las condiciones de lectura o escritura (la función suspende la ejecución si los buffers no contienen datos para lectura o se encuentran llenos ante escritura).Se puede especificar un tiempo de espera límite para evitar esperas indefinidas ocasionadas por errores o para permitir hacer tareas alternativas.
shutdown( )	Los sockets permiten comunicación bidireccional. No obstante, si se desea impedir la comunicación en algún sentido, o en los dos (para evitar pérdida de datos mientras se cierra el socket), se puede emplear shutdown( ).
recv( )	Permiten la recepción de datos desde un socket. Normalmente, bloquean el socket hasta que se encuentren datos disponibles, pero se puede cambiar este comportamiento con la ayuda de setsockopt( ), recv( ) solo se puede emplear con sockets conectados (SOCK_STREAM) mientras que recvfrom( ) admite los dos tipos.
recvfrom( )	
Send( )	Solo permite enviar datos en sockets conectados, sendto( ), y sendto( ) en los dos. Cuando se envían datos con sendto( ) a sockets no conectados (SOCK_DGRAM) se debe tener cuidado en no rebasar el tamaño del paquete de todas las redes que atraviese para evitar truncamientos. Este tamaño se puede consultar con ayuda de las getsockopt( ), empleando como parámetro de socket SO_MAX_MSG_SIZE.
sendto( )	

Tabla 1-3 Resumen de funciones utilizadas en Visual C++

2.2 ESTRUCTURA DEL SISTEMA DE REPLICACIÓN PROPUESTO (SRP)

DEFINICIÓN DEL PROBLEMA

En la actualidad los sistemas de información son importantes instrumentos de control para las empresas privadas y aun para algunas instituciones gubernamentales, es un hecho que disponer de información en tiempo real puede ahorrar grandes cantidades de trabajo extra, como la recolección de este tipo de información de forma documental y posteriormente de forma electrónica para almacenarla en Bases de Datos, y puede ayudar a predecir con mayor certeza el futuro de la empresa.

Por el dinamismo que envuelve el mundo de la computación tanto en aspectos de software como de hardware, es muy difícil que exista una homologación respecto a los sistemas, por ejemplo: pueden existir cierta cantidad de equipos con características medias como equipos Pentium con Windows 95, y otra cantidad de Pentium 3 ó Pentium 4 con plataformas como Windows 98, Windows NT Workstation ó 2000, además de servidores con distintas plataformas como Windows NT, Windows NT 5 (o Windows NT Server), Linux y Unix , y muchos otros. Hasta el momento se ha comprendido que una aplicación en Windows puede conectarse mediante ODBC a cualquier plataforma y por ende cualquier manejador de base de datos que disponga de los controladores para ODBC del manejador de base de datos en el cliente; además de la posibilidad de realizar un seguimiento de la aplicación a nivel cliente mediante una herramienta de ODBC controlada por el manejador de controladores llamada traza, físicamente controlada por una DLL llamada "odbctrac.dll", esto se analizó profundamente en el punto anterior, y a pesar de que existen múltiples sistemas de replicación como Replication Server de Sybase, SQL Remote de Sybase, el sistema snapshot de Microsoft SQL Server, algunos Gateway de Sybase - Oracle, etc . No existe un sistema lo bastante amigable que permita establecer una replicación entre diferentes plataformas y menos aun entre diferentes sistemas manejadores de Bases de Datos, esta es la esencia del problema que se pretende resolver con el

presente trabajo, realizar un sistema lo suficientemente sencillo de implementar, que funcione basándose en el envió de sentencias SQL al servidor al igual que muchos otros sistemas de replicación, sea bastante rápido en enviar dichas sentencias y que al igual que la aplicación pueda correrse en cualquier aplicación con Windows (mínimo Windows 95), además de que el sistema de replicación sea invisible al usuario; es decir el usuario usé la aplicación sin necesidad de saber que la información se esta replicando.

El sistema de replicación propuesto, es desarrollado en Visual C++ de Microsoft Versión 6.0. y consta básicamente de tres componentes un servidor de escucha (ubicado opcionalmente en el servidor a replicar), una DLL de seguimiento (ubicada en los clientes) la cual replica (envía los mensajes de sentencia) a dicho servidor y una DLL que contiene las llamadas a las funciones para obtener la dirección y el puerto del servidor destino de un archivo ".ini". Se aprovecha al máximo la interoperabilidad de ODBC y de su sistema de seguimiento, ya que se adecuo el código fuente de la librería de seguimiento "odbctrac.dll" extraído del Microsoft Data Access SDK2.6 (software developer Kit - Kit de Desarrollo de Software); para crear otra DLL que permitiera mandar mensajes (mediante una DLL intermedia que contiene las llamadas a las funciones para obtener la dirección IP y Puerto) al servidor que se conecta al sistema administrador de base de datos que contiene la base de datos a replicar.

En el esquema de la figura 2.4 queda explicado de forma general el proceso de sistema de replicación propuesto.

Características Generales Del Sistema De Replicación Propuesto SRP

- El SRP esta diseñado para replicar información en tiempo real
- Solo corre bajo sistemas Windows
- Requiere la arquitectura ODBC para su optimo funcionamiento
- Utiliza el protocolo TCP IP

El alcance del sistema de replicación propuesto se describe detalladamente en la última sección de este capitulo.

FLUJO DE PROCESOS DE INFORMACIÓN EN EL SRP

Los procesos del sistema propuesto de replicación, se dividen principalmente en procesos del servidor y procesos de cliente, a continuación se muestra detalladamente cada proceso que se relaciona con el sistema, la descripción de dichos procesos, al igual que el código fuente realizado en Visual C++ V. 6.0. también se incluyen en el archivo leeme.txt del disco anexo a esta tesis[13].

[13] El código fuente de la aplicación, en relación a los procesos del servidor, esta en el Apéndice A de la presente Tesis; Nota del Autor.

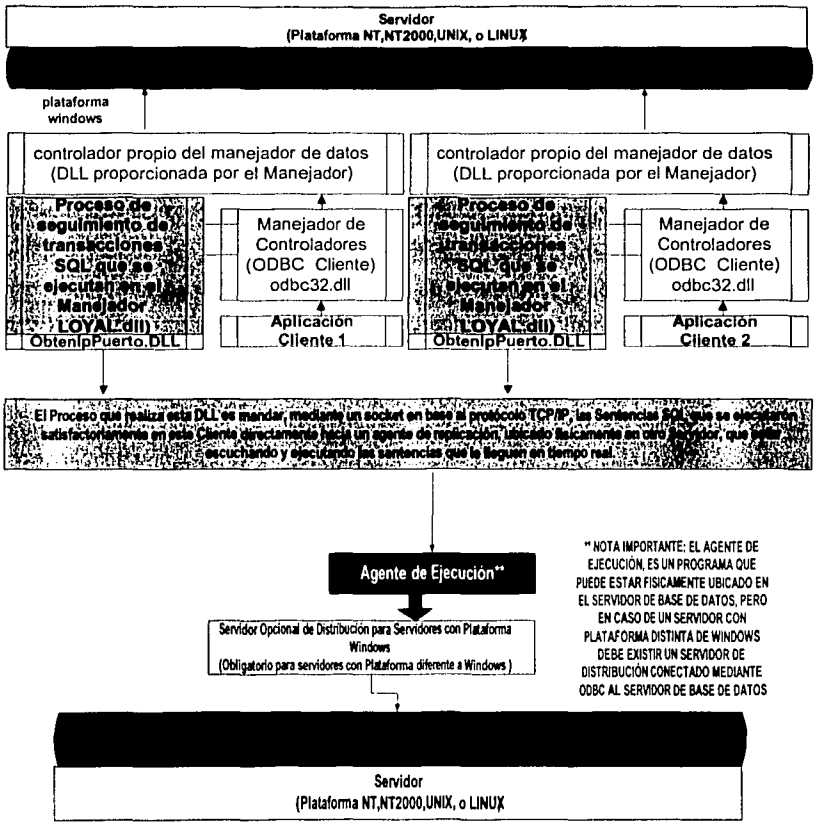


Figura 2-4 Proceso general del Sistema Replicación Propuesto.

Procesos del Servidor de Replicación (Agente de Ejecución)

El Agente de ejecución o servidor de replicación es un archivo ejecutable llamado LoyalServer.exe cuya función es escuchar peticiones de sentencias SQL y ejecutarlas, también cuenta con un archivo por lotes llamado LoyalServer.bat, en el cual se establecen los parámetros del programa ejecutable que son los siguientes: la dirección IP, el puerto de escucha del servidor, el nombre del dsn (data source name) nombre del origen de datos especificado en el ODBC del panel de control previamente, un usuario y una contraseña (el usuario debe tener permisos para ejecutar sentencias SQL). El archivo ejecutable esta creado con tecnología de sockets y el proceso que realiza específicamente, mostrado en el diagrama de actividades contenido en la figura 2.5, se muestra a continuación:

Secuencia de pasos del Servidor de Replicación

1. Crear un objeto llamado Servidor.
2. Crear un objeto para conectarse a una base de datos (previamente especificada en el nombre del origen de datos ODBC).
3. Iniciar conexión, a la base de datos y continuar, si no, abortar el programa.
4. Si la conexión es satisfactoria, verificar que se pueda ejecutar sentencias SQL y seguir con el paso 5, si no, abortar el programa.
5. Realizar un ciclo, que no terminará hasta que la aplicación sea interrumpida, es decir hasta que el programa se cierre.
6. Verificar en el método Servicio si ha llegado un mensaje y continuar con el paso 7 , si no es así regresar al paso 5.
7. Si llego algún mensaje (esto significa que llego una sentencia SQL), ejecutarlo y regresar al paso 5, si falla reportar el mensaje en pantalla y regresar al paso 5.

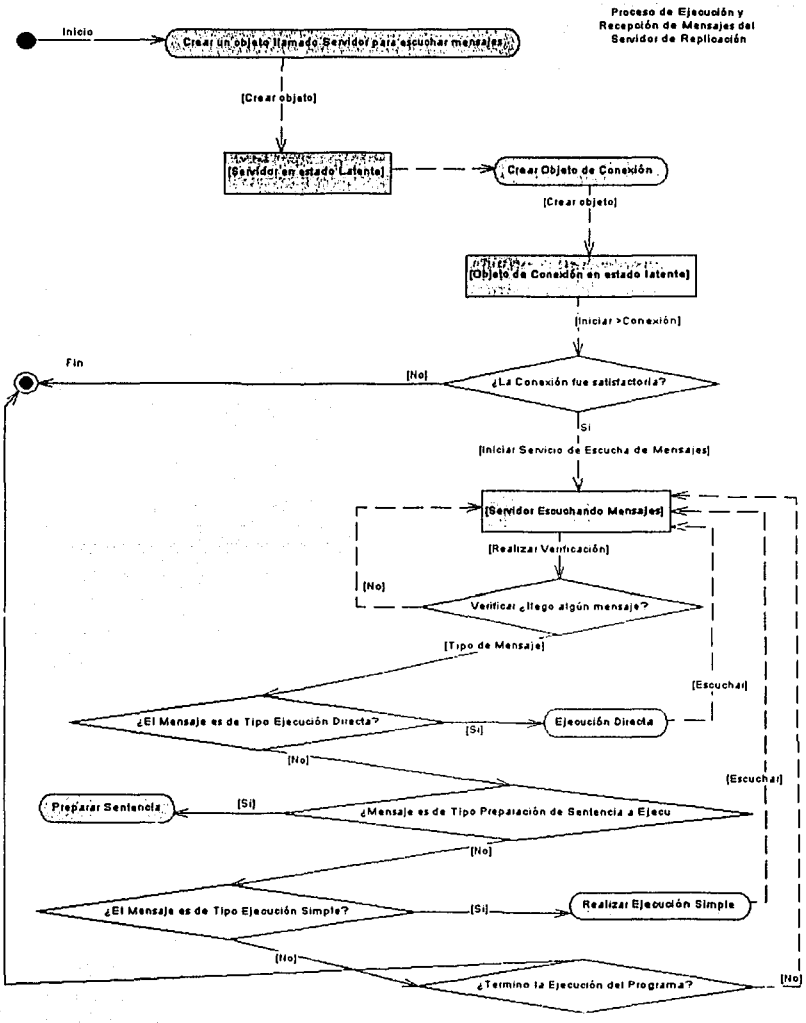


Figura 2-5 Diagrama de actividades del Servidor de Replicación

Proceso del Cliente

Obviamente el cliente debe tener un DNS (data source name) nombre del origen de datos ODBC especificado para su conexión con alguna base de datos ubicada en algún servidor opuesto al de replicación; el cual, por supuesto, debe soportar una conexión ODBC, independientemente de la plataforma y el manejador de base de datos.

El proceso del cliente, hace una llamada a un archivo[14] que contiene las funciones para obtener la dirección Ip y el puerto, de un archivo de configuración llamado Loyal.ini, el segundo[15] esta basado en el seguimiento realizado por el manejador de controladores de ODBC, el proceso consiste en realizar el seguimiento de la aplicación en el cliente (o clientes), tal como se muestra en la figura 2.3 en el punto anterior (prácticamente la parte medular del sistema), y esto es, el registro en un archivo .log de las sentencias SQL que se realicen por dicho cliente y además mandarlas, usando la tecnología de sockets, vía red apuntando al servidor de replicación o agente de ejecución, con la dirección Ip y el puerto que han sido previamente obtenidos en el archivo Loyal.ini.

Proceso especifico de Obtención de Ip y Puerto

En realidad está solo es la definición de la funciones y mientras no se llame, no realiza ningún proceso, pero si se manda a llamar realiza las siguientes tareas:

[14] El código fuente de la aplicación, en relación al proceso de lectura del archivo de configuración Loyal.ini, esta en el Apéndice C de la presente Tesis, y en el Disco Adjunto a la Tesis; Nota del Autor.

[15] El código fuente de la aplicación, en relación al proceso de seguimiento de transacciones en el cliente, esta en el Apéndice B de la presente Tesis, y en el Disco Adjunto a la Tesis; Nota del Autor.

Secuencia de pasos para la función ObtenIP

1. Obtiene una variable entera, que es la localidad del carácter en un buffer auxiliar
2. Abre el archivo de configuración Loyal.ini.
3. Salta los primeros nueve caracteres.
4. Lee 16 caracteres en un buffer (la dirección Ip) y cierra el archivo.
5. Inicializa una variable en 0 y almacena un carácter en un buffer auxiliar, si es menor ó igual repite el paso 3, en caso contrario sigue con el paso 4.
6. Retorna el carácter de la localidad especificada en el paso 1.

Para la función ObtenPuerto

1. Abre el archivo de configuración Loyal.ini
2. Salta los primeros cuarenta y cinco caracteres.
3. Lee 5 caracteres en el buffer y cierra el archivo.
4. Convierte a número el buffer y lo retorna.

A continuación se muestran los diagramas de actividades de los procesos anteriores:

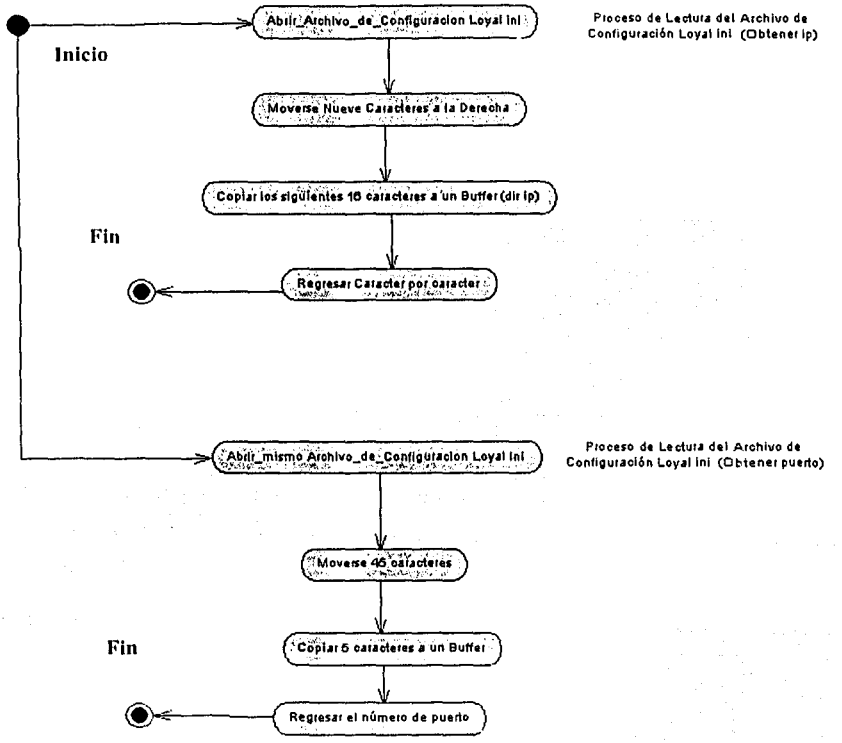


Figura 2-6 Diagrama de actividades del Proceso específico de Obtención de Ip y Puerto

Proceso específico de seguimiento

1. Inicializar el programa y obtener Dirección Ip y Puerto (llamando al proceso ObtenPuerto).

2. Dependiendo de la función que llama la aplicación; llamar a la función con prefijo Trace de seguimiento, esta a su vez llamara un subproceso descrito en el paso tres.

3. Si no está abierto abre el archivo de seguimiento predeterminado.

4. Si es alguna de las funciones principales descritas en la tarea de Procesamiento de Transacciones de ODBC ("SQLPrepare", "SQLExecDirect") y además sus argumentos son cadenas, pasa al paso siguiente.

5. Si la sentencia no lleva la palabra "SELECT" en la cadena pasa al siguiente paso.

6. Escribe la sentencia en un archivo de seguimiento y manda mediante un socket, la sentencia, al servidor que este replicando.

7. Regresa al Paso 2.

El proceso representado como un diagrama de actividades quedaría de la siguiente manera:

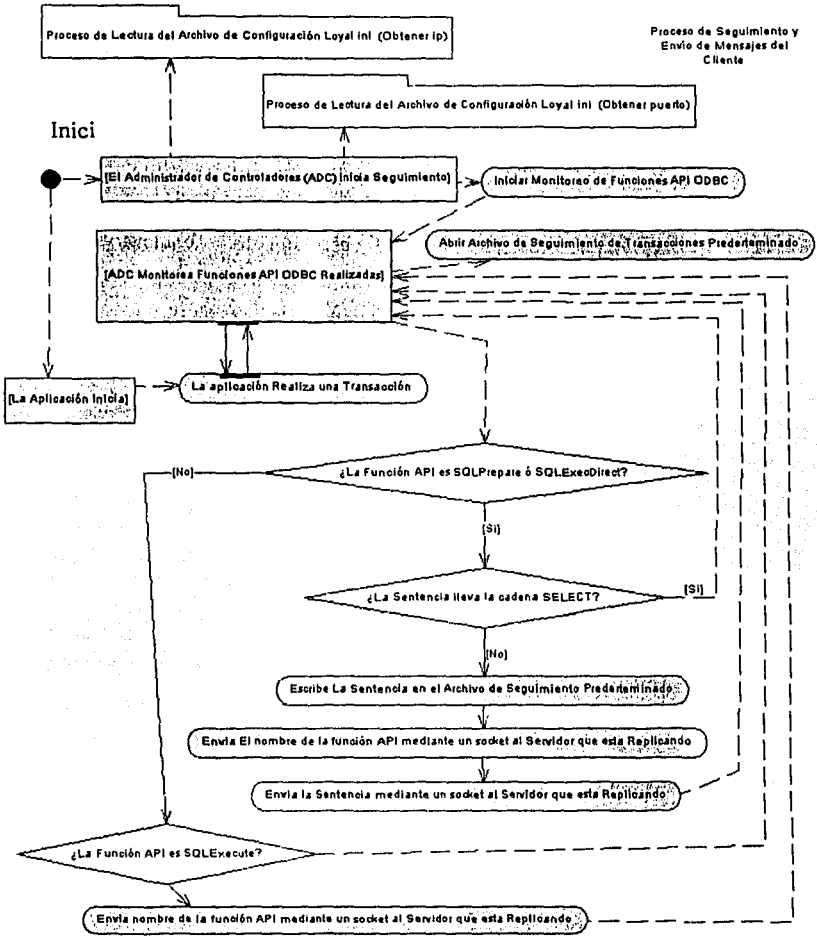


Figura 2-7 Diagrama de actividades del Proceso de Seguimiento

2.3 APLICACIONES DEL SISTEMA DE REPLICACIÓN PROPUESTO

A continuación se mencionan algunas aplicaciones reales donde podría usarse el sistema desarrollado:

Comercio Electrónico

El comercio electrónico es cualquier actividad de intercambio comercial en la que las órdenes de compra, venta y pagos se realizan a través de un medio telemático, los cuales incluyen servicios financieros y bancarios suministrados por Internet.

El comercio electrónico es la venta a distancia aprovechando las grandes ventajas que proporcionan las nuevas tecnologías de la información, como la ampliación de la oferta, la interactividad y la inmediatez de la compra, con la particularidad que se puede comprar y vender a quién se quiera, dónde y cuándo se quiera.

Las necesidades del comercio electrónico conllevan una serie de características técnicas acerca del manejo de la información, se han creado algunos dispositivos electrónicos que ayudan a la captura de datos de forma rápida en el momento preciso, la rapidez en el procesamiento de datos ha llegado a ser una necesidad primordial y es aquí donde esta propuesta de replicación puede jugar un papel muy importante en el desarrollo de algún sistema.

Existen algunas empresas que requieren saber información de sus ventas casi a tiempo real; obtener esta información puede ser muy difícil si se tiene una red de ventas amplia. Algunas empresas piden a sus vendedores registrar información de las ventas realizadas en dispositivos móviles que descargan dicha información en Bases de Datos alternas que a su vez replicaran información a una base de datos global que contendrá los datos a ser consultados en tiempo real. Esto trae consigo grandes ventajas como son las siguientes:

- Los parámetros para tomar decisiones se obtienen casi en tiempo real.
- La información que se obtiene es constante y concisa.
- Se eliminan procesos intermedios manuales que retrasarían el envío de información.

Las desventajas son las siguientes:

- El envío de datos depende de las comunicaciones y si estas son interrumpidas los datos no pueden ser consultados.
- Si no existe un buen diseño de las estructuras de las Bases de Datos; puede crearse inconsistencia en los mismos.

El flujo de la aplicación se muestra gráficamente en la figura 2.8.

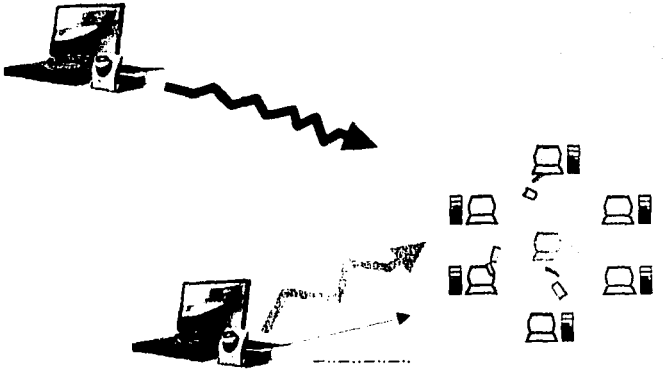


Figura 2-8 Flujo de la aplicación de comercio electrónico para el Sistema de Replicación Propuesto

Transacciones Bancarias

La necesidad de que la información acerca de las transacciones que ha realizado un cliente en un banco mediante cualquier medio este debidamente actualizada en el menor tiempo posible es de suma importancia; así como es indispensable que dicha información este disponible en cualquier momento con la menor posibilidad de falla aun ante una catástrofe de perdida de información.

Una vez más, el sistema de replicación propuesto puede ser una alternativa para satisfacer estas necesidades proporcionando un ambiente de Bases de Datos Distribuidas que se conectaran a un servidor central y duplicaran cualquier transacción que se realice en cualquiera de esas Bases de Datos alternas mediante el sistema de replicación.

Ventajas:

- En cualquier catástrofe de perdida de datos, se tiene un servidor alterno que contiene la misma información respaldada en tiempo real.
- La información puede ser replicada a más de un servidor en tiempo real.

Desventajas:

- El envío de datos depende de las comunicaciones y si estas son interrumpidas los datos no pueden ser respaldados.
-

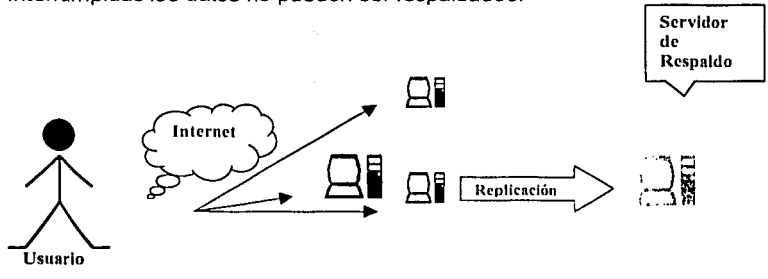


Figura 2-9 Transacciones Bancarias mediante el Sistema de Replicación Propuesto

Reservación de Vuelos en líneas Aéreas

Es muy común en la actualidad reservar ya sea vía telefónica, Internet o incluso personalmente un vuelo para dirigirse a algún sitio; es importante, en este caso, que se tenga un control centralizado de las reservaciones que se han realizado ya que esas reservaciones pueden hacerse desde cualquier parte del país, incluso del mundo. El sistema de replicación mantendría la información de los vuelos en un servidor central en tiempo real; así cada reservación efectuada se visualizaría en una computadora central. Así de esta manera el sistema de replicación puede ser una alternativa de solución permitiendo conocer la disponibilidad de vuelos en cualquier parte del país y/o del mundo.

Ventajas:

- El acceso a la información es en casi tiempo real.
- Las reservaciones pueden ser efectuadas al momento; en caso de disponibilidad.

Desventajas:

- El envío de datos depende de las comunicaciones y si estas son interrumpidas los datos no pueden ser consultados.

Figura 2-10 Reservación de Vuelos mediante el Sistema de Replicación Propuesto

Limitaciones y Alcance del Sistema de Replicación Propuesto

A pesar de que el sistema de replicación desarrollado es funcional, hay que recordar que es un prototipo y como todo sistema funciona dentro de un entorno delimitado; por tal razón a continuación se muestran una serie de aspectos que especifican bajo que estándares el prototipo funciona adecuadamente.

Entorno Cliente Servidor

El sistema de replicación esta diseñado para replicar transacciones en un entorno clásico cliente – servidor, en donde el control de las transacciones en su mayoría puede estar controlado por el cliente ignorando las nuevas tendencias de diseño de sistemas donde los manejadores de Bases de Datos (servidores) llevan el control de las transacciones importantes, por medio de algunas técnicas que proporciona el manejador como stored procedures, o incluso puede existir un servidor intermedio de transacciones entre el cliente y el servidor (arquitectura de tres capas). Los stored procedures son código DML (Data Manipulation Language) que interpreta el compilador del manejador de Base de Datos y que sirve para que el servidor administre las transacciones que llegan del cliente, mediante una simple petición del mismo, sin necesidad de que dicho cliente almacene el código.

Protocolo TCP/IP

El protocolo de comunicación utilizado para enviar y recibir mensajes entre dos computadoras es el protocolo TCP-IP, la comunicación utilizando este protocolo es esencial, el enlace debe ser directo; para fines de esta tesis no se indagara entre los diferentes métodos de comunicación y protocolos que existen solo se insistirá en que la especificación, en cuestión de comunicación, para el funcionamiento correcto del sistema propuesto es el protocolo TCP/IP.

Plataforma

El sistema de replicación propuesto solo corre bajo sistemas operativos Windows, pero puede comunicarse a través ODBC con otros sistemas operativos como UNIX o LINUX.

Arquitectura ODBC

El sistema de replicación soporta casi toda la arquitectura ODBC, excepto la ejecución de transacciones por medio de SQLBindParameter. En lenguajes de programación visuales que permitan generar aplicaciones que utilicen el sistema ODBC y que utilicen lenguaje SQL incrustado para ejecutar las transacciones básicas (insertar, modificar y eliminar), el sistema de replicación propuesto funcionara sin ningún problema; siempre y cuando no se utilicen algunos objetos de datos que usen el comando SQLBindParameter de ODBC. Al principio de este capitulo se describen los pasos básicos de una aplicación ODBC, donde se menciona el uso de los diferentes comandos para ejecutar una transacción, uno de ellos es el SQLBindParameter el cual primeramente prepara la transacción (SQLPrepare) y después agrupa los parámetros ya sea en variables o en arreglos de datos, para su posterior ejecución (SQLExecute) ya que el proceso de mandar arreglos completos de datos por medio de la red es un proceso complejo (incluso podría diseñarse un nuevo algoritmo para mandar datos) se decidió omitir la funcionalidad de dicha función.

Soporte de tipos de datos básicos

El sistema de replicación propuesto replica transacciones mandando mensajes, da por hecho que las Bases de Datos han sido diseñadas, a pesar de estar ubicadas en diferentes plataformas y administradas por diferentes manejadores, para poder replicar y por ende soportar los mismos tipos de datos; si un cliente replica una transacción el mensaje se envía, se ejecuta y puede no realizarse

satisfactoriamente si el tipo de dato por ejemplo no es soportado por el servidor destino que ejecuta el mensaje.

El sistema de replicación propuesto soporta los tipos básicos de datos: carácter, fecha y número; no esta diseñado para replicar tipos de datos complejos como blobs (imágenes) o binary.

Hacia donde ir

El siguiente paso de este proyecto tiene muchas variantes que van desde rediseñar el prototipo hasta generar nuevas técnicas de replicación, sin embargo, todas llevan al mismo camino: mejorar el sistema de replicación propuesto, puede ser una buena alternativa diseñar el sistema de replicación de tal forma que sea un servidor quien envíe mensajes de datos y sea otro servidor quien los reciba y los procese; además de que los clientes conectados a ese servidor puedan visualizar los datos replicados, o sea que la replicación la realice el servidor. Otro aspecto importante seria el diseño de un programa administrador de replica en forma gráfica para el usuario y la replicación en ambos sentidos; es decir que una terminal pudiera ser servidor y cliente al mismo tiempo, esta situación nos llevaría al desarrollo de un nuevo administrador de orígenes de datos a nivel servidor que tendría la capacidad de generar nuevos orígenes de datos apuntando hacia algún cliente en específico, así cuantos clientes fueran necesarios, y también debe de hacer un seguimiento de las transacciones que se realicen para mandar dichas transacciones al servidor con el cual se desee replicar datos.

El desarrollo de esta propiedad está fuera del alcance de este trabajo; sin embargo el prototipo de replicación propuesto puede adaptarse a sistemas donde el número de terminales conectadas a un servidor sea mínimo; a pesar de que el sistema no ha sido probado en una WAN, porque una infraestructura de tal magnitud estaba fuera del alcance de este proyecto, la teoría nos dice que su funcionamiento debe ser el mismo que el comportamiento en la infraestructura simulada.

Conclusiones del Capítulo 2.

En el presente capitulo se muestra la estructura del sistema de replicación propuesto, se muestran los procesos que conforman el sistema, previamente se hace un análisis de las principales tecnologías que se utilizan para desarrollar el prototipo, como el APIs de ODBC y la tecnología de comunicación en Windows (WinSockets). Cada proceso descrito es parte de un programa codificado y se encuentra impreso en los anexos y en un disco, adjunto a la presente tesis. Un aspecto importante, es tener en cuenta que el sistema solo funciona en aplicaciones ODBC (que permitan una conexión ODBC); además de que no todas las funciones API de ODBC se soportan en el sistema de replicación, específicamente solo son soportadas tres: SQLPrepare, SQLExecute y SQLExecDirect. La razón de esto es que en aplicaciones desarrolladas en lenguajes visuales como Visual Basic, son las funciones, más utilizadas; sin embargo existen objetos (controles[16]) propios de los lenguajes que utilizan otra función, SQLBindParameter[17], (como el control DataWindow de Power Builder) en diversos lenguajes, lo cual no funcionaria correctamente en el sistema de replicación propuesto. A pesar de esto, no importa la metodología de conexión de los lenguajes, siempre y cuando sean conexiones ODBC, lo que es un hecho es que en aplicaciones sencillas (desarrollan sus procesos principales exclusivamente con sentencias SQL) el sistema funcionara sin problemas.

[16] Un control, es un objeto visual para el usuario que puede almacenar datos en memoria, y puede utilizarse como instrumento para enviar dichos datos a la base y almacenarlos, modificarlos o eliminarlos, Nota del Autor.

[17] Como se explicó previamente al principio del presente capítulo, la función SQLBindParameter también forma parte de la tarea de procesamiento ODBC, e incluso es la única que no se ve involucrada con el sistema de replicación propuesto; ya que el manejo de esta función es compleja y no se contaba con tiempo suficiente en el desarrollo de la presente Tesis, Nota del Autor.

El último punto del presente capítulo da a conocer cuales son las limitaciones del sistema de replicación propuesto, en el próximo capítulo se mencionarán las ventajas y desventajas del sistema.

CAPÍTULO III EVALUACIÓN DE RESULTADOS

El criterio que se toma para evaluar el sistema de replicación propuesto se basa en los objetivos primordiales de esta tesis que son minimizar costos de software y tiempos de respuesta de transacciones. Para poder realizar la evaluación se tomaron como base solo algunos manejadores de Bases de Datos y sistemas de replicación en el mercado.

3.1 EVALUACIÓN ESTADÍSTICA DE LOS RESULTADOS (FACTOR COSTO)

Hoy en día los costos de los sistemas de replicación son bastante altos, tomando en cuenta que el presente prototipo fuera software libre; es decir que fuera publicado en Internet y cualquier persona pudiera mejorarlo o incluso adaptarlo a sus necesidades. El sistema sería gratis y los costos de replicar información serían prácticamente nulos; la información se muestra la tabla 3.1, representada en la gráfica 3.1, diferenciando los costos entre diferentes software de replicación encontrados; es importante resaltar que para fines de esta tesis se tomaron en cuenta los sistemas más comerciales, ya que existen muchos en el mercado, así también cabe aclarar que existen otros sistemas de replicación que están prácticamente ligados con el manejador, con el hardware de la computadora (número de procesadores), la versión (tanto del software de replicación como del manejador de Base de Datos), la plataforma y muchos factores más que despreciaremos parcialmente para fines de esta tesis; así que mostraremos exclusivamente los costos del software para plataformas Windows, computadoras con procesador Intel y las últimas versiones de software, puesto que es en computadoras con estas características donde corre el sistema de replicación propuesto. Los costos están actualizados hasta Septiembre del 2002 y la fuente de información es Internet (se especifica la fuente en la Tabla); se incluye el costo de una licencia adicional considerando que se necesitará instalar en mínimo dos equipos. Microsoft, en su última versión del manejador de Base de Datos SQL

Server 2000, hasta septiembre del 2002, incluye un software de replicación en tres modalidades distintas, Snapshot para replicaciones totales de los objetos de la Base de Datos, Merge para replicación ocasional y Transactional que copia transacciones y las envía. Esta última modalidad de Replicación solo es permitida al adquirir la versión estándar o la empresarial de Microsoft SQL Server 2000.

Una vez mencionados estos aspectos importantes, se muestra la tabla a continuación.

Todos los precios son en dólares Americanos

Manejador de Base de Datos	Costo Unitario + 1 Licencia	Software de Replicación	Costo Unitario + 1 Lic.	Costo Total	Fuente de Información	Fecha de Consulta
Adaptive Server Enterprise Sybase v. 12.5	$ 4,790.00	Replication Server v. 12.5 Sybase	$ 1,790.00	$6,580.00	www.sybase.com	23/09/02
Adaptive Server Anywhere v. 8.0	$ 399.00 NRL	SQL Remote (I)	-	$ 399.00	www.sybase.com	23/09/02
Microsoft SQL Server Enterprise Edition	$ 2,659.00	(I)	-	$ 2,659.00	www.atomicpark.com	23/09/02
Microsoft SQL Server Standart Edition	$ 3,890.00	(I)	-	$ 3,890.00	www.atomicpark.com	23/09/02
Cualquier Manejador que soporte ODBC*	0	SRP	0	0	-	-

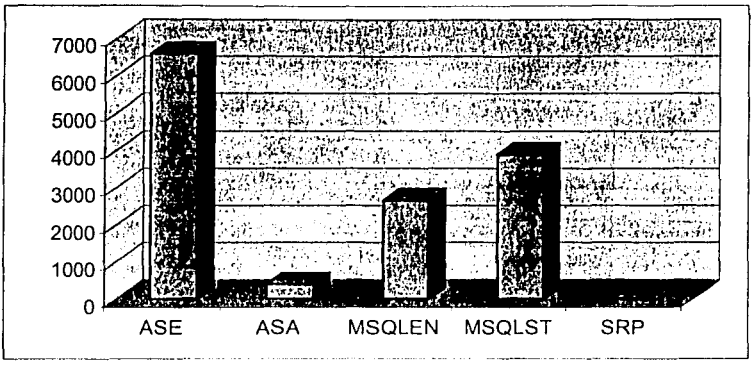
* Nota Soporta todos los mencionados

(NRL) No requiere Licencia

(SRP) Sistema de Replicación Propuesto

(I) Software de Replicación esta incluido en el manejador

Tabla 3.1 Tabla de costos de diferentes sistemas de replicación



Gráfica 3.1 Gráfica de la Tabla de costos de diferentes sistemas de replicación

Existe un software, que no es ODBC, que permite comunicar diferentes plataformas y manejadores con Sybase ASE; el cual se llama Open Server, el precio de este software no fue incluido en la tabla de análisis comparativo pero se muestra a continuación, para diferentes plataformas, como dato adicional.

Todos los precios son en dólares Americanos

	Costo Unitario	Licencia	Fuente de Información	Fecha de Consulta
Open Server 12 para UNIX	$ 3,500.00	$ 600.00	www.sybase.com	23/09/2002
Open Server 12 para Win. NT.	$ 3,500.00	$ 600.00	www.sybase.com	23/09/2002
Open Server 12.5 para LINUX	$ 3,500.00	$ 600.00	www.sybase.com	23/09/2002

Tabla 3.2 Tabla de costos de Open Server

Evaluando los resultados obtenidos se puede apreciar que después del software de replicación, el SQLRemote de Sybase es el segundo más barato seguido por el SQL Server 2000 de Microsoft, y él más caro es el Replication Server de Sybase.

Ambiente	Sybase Adaptive Server Anywhere, SQL Remote	Microsoft SQL Server, Snapshot	MySQL-Access97, Sistema de Replicación Propuesto
Hardware	Pentium 4 / AMD K-6 II	Pentium 4 / AMD K-6 II	Pentium 4 / AMD K-6 II/ Pentium I
Software de Replicación	SQL Remote	Microsoft SQL Server	Sistema de Replicación Propuesto
Plataforma Origen	Workstation NT 4	Windows NT 5	Linux
Plataforma Intermedia	—	—	Windows 2000
Plataforma Destino	Windows 2000	Windows 2000	Windows 95
Manejador Origen	Anywhere Sybase	SQL 2000	MySQL
Manejador Destino	Anywhere Sybase	SQL 2000	Access97

Tabla 3.3 Tabla de Ambientes de Prueba, cada columna representa las características del ambiente

3.2 EVALUACIÓN ESTADÍSTICA DE LOS RESULTADOS (FACTOR TIEMPO)

Para realizar las pruebas de desempeño, analizando el factor tiempo; se hizo una aplicación en Visual Basic, junto con un modelo de Base de Datos. La prueba consiste básicamente en realizar la replicación con 3 distintos software, en diferentes plataformas y diferentes manejadores, cabe señalar que a pesar de que se sabe que existen muchos software's de replicación y manejadores; por falta de tiempo y disposición de los más caros, se consideraron, para la presente prueba, solo algunos de los más comerciales y los más usados. El objetivo es registrar el tiempo de envío en segundos y centésimas de segundo en una muestra de 30 intentos diferentes y comparar el rendimiento entre los mismos; antes de realizar las pruebas se describe brevemente el funcionamiento de la aplicación desarrollada. Posteriormente se explica como funciona cada uno de los software's seleccionados para esta prueba y por último se muestra la tabla de resultados de la prueba y su gráfica respectiva. Las pruebas de desempeño se llevan a cabo entre distintos software's de replicación, plataformas y hardware, los diferentes ambientes quedan compuestos de la siguiente manera:

APLICACIÓN PARA PRUEBAS DE DESEMPEÑO

Estructura de datos de la aplicación

Las Bases de Datos creadas para la aplicación fueron nombradas dependiendo del manejador que las administra por ejemplo la Base de Datos de Access 2000 se llama AeropuertoMia2000; el nombre de la conexión especificado en el origen de datos del administrador de controladores en Windows tiene el mismo que la Base de Datos. La aplicación esta conformada por los siguientes catálogos: C_TIPO_AVION, C_TIPO_VUELO, C_STATUS, C_AEROPUERTOS, C_AEROLINEA; cada uno es una tabla y la tabla principal es itinerarios (INTINERARIOS), a continuación mostramos el diagrama entidad -> relación de la Base de Datos de la aplicación.

Aplicación

La aplicación fue desarrollada para probar el sistema de replicación propuesto no fue creada siguiendo un método de análisis previo, simplemente esta hecha para comprobar y mostrar como se puede realizar la replicación, el nombre de la aplicación es Aeropuerto. La aplicación a grandes rasgos pretende simular, muy vagamente, un sistema de control de vuelos para un conjunto de aeropuertos.

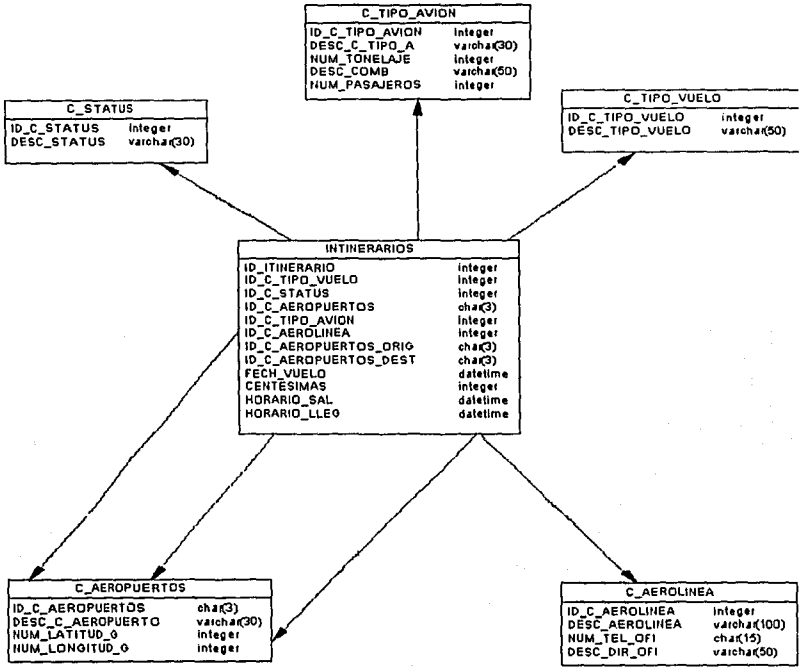


Figura 3.1 Diagrama entidad relación de la aplicación de prueba

Esta hecha en Visual Basic y se anexa el código fuente en el anexo D; el paquete de instalación se anexa en el disco que acompaña esta Tesis, la instalación es muy sencilla y no requiere una explicación amplia. El paquete de instalación dejara el ejecutable de la aplicación en la siguiente ruta, si es que no se cambio el destino, "c:\project\Aeropuerto.exe". El flujo de los procesos que intervienen en la aplicación son muy sencillos y se describen brevemente en la siguiente sección.

¿Como funciona la aplicación?

La aplicación es independiente del sistema de replicación; sin embargo para fines prácticos de la presente tesis es necesario tener instalado el software de replicación. Una vez que se ha configurado dicho software e iniciado la aplicación, no se requiere configuración solo instalarla, se presentara la pantalla del administrador de controladores para seleccionar un origen de datos y conectarse a la Base de Datos indicada en dicho origen, la pantalla es la siguiente:

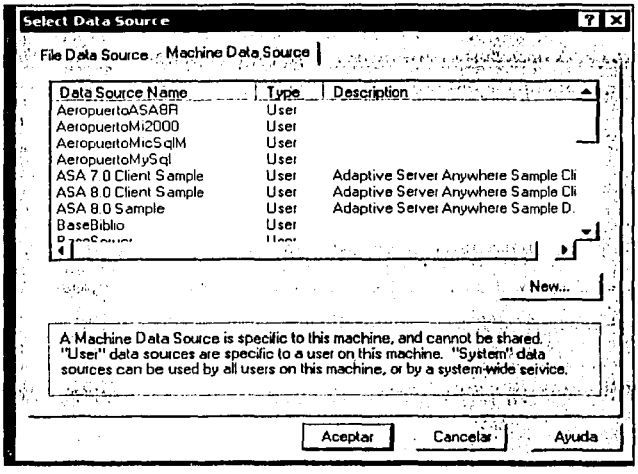


Figura 3.2 Pantalla inicial de la aplicación

Una vez conectada la aplicación a la Base de Datos del origen de datos, se presenta la pantalla principal del sistema; la cual muestra los registros de vuelos programados para el día actual, el sistema cuenta con las funciones básicas de modificación de registros (actualizar, borrar, insertar y seleccionar) tanto para la pantalla principal que controla el itinerario de vuelos como para los catálogos mencionados en el punto anterior. La pantalla principal cuenta con la opción de "refrescar cada segundo" lo cual permite visualizar en pantalla los registros

prácticamente en tiempo real, además de medir el tiempo que tardo en insertarse o borrarse la última transacción en cuestión de centésimas de segundo; cabe señalar que esta acción en particular se realiza registrando la fecha, la hora y las centésimas en el momento que se inicia el proceso de alta los registros (al presionar el botón) y se hace la diferencia de tiempo con la fecha, la hora y las centésimas en el momento que la transacción se realiza satisfactoriamente. A continuación se ilustra la pantalla principal de la aplicación.

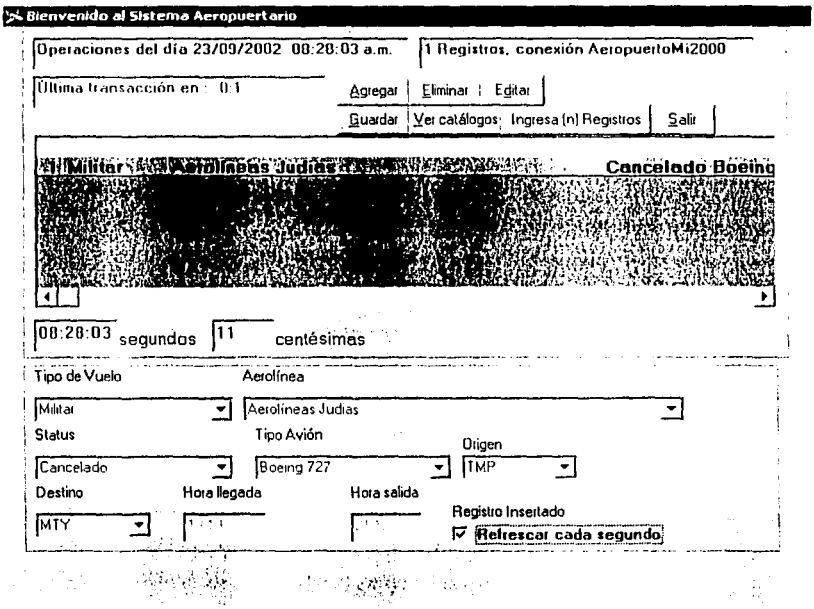


Figura 3.3 Pantalla Principal de la Aplicación

El uso de los catálogos es sencillo ya que su labor es el mantenimiento (Inserción, actualización, eliminación y selección) de las siguientes tablas descritas en la estructura de la aplicación.

Es muy importante mencionar que la técnica de ejecutar las transacciones depende de la metodología de programación y del lenguaje; sin embargo si se usa ODBC, independientemente de este último, los pasos básicos de la aplicación ODBC serán siempre los mismos, es decir el lenguaje mediante alguna instrucción hace referencia y ejecuta cualquiera de las API's de ODBC que se usan en los pasos básicos de ODBC para la ejecución (SQLExecute, SQLExecDirect, SQLPrepare) .

El rol de la aplicación para realizar estas pruebas es primordial, la prueba es sencilla y consiste en, una vez teniendo la instalación de dos aplicaciones en dos máquinas distintas y teniendo un software de replicación funcionando entre dos Bases de Datos de dichas aplicaciones, enviar una transacción n veces (una muestra de 30 para esta prueba), con cierto intervalo de tiempo entre cada envío, y registrar el tiempo que tardó en llegar la transacción. A partir de los resultados obtenidos, se puede hacer una idea de la funcionalidad de los sistemas de replicación. El siguiente paso es describir brevemente como se configuró y el funcionamiento básico de los sistemas de replicación utilizados para las pruebas.

Microsoft SQL Server, Snapshot

El tipo de replicación Snapshot permite replicar de manera periódica y constante pero no en tiempo real, este método copia los datos o los objetos de la Base de Datos exactamente como existan al momento. Este tipo de Replicación esta permitido en cualquier versión de SQL Server 2000. La replicación tipo Merge es usada para replicas ocasionales, puede aplicarse a Bases de Datos móviles. Este tipo de Replicación esta permitido en cualquier versión de SQL Server 2000. La replicación transactional funciona de manera muy similar al Sistema de

Replicación Propuesto, cuando los datos de la publicación se modifican, las transacciones son capturadas y enviadas a los suscriptores.

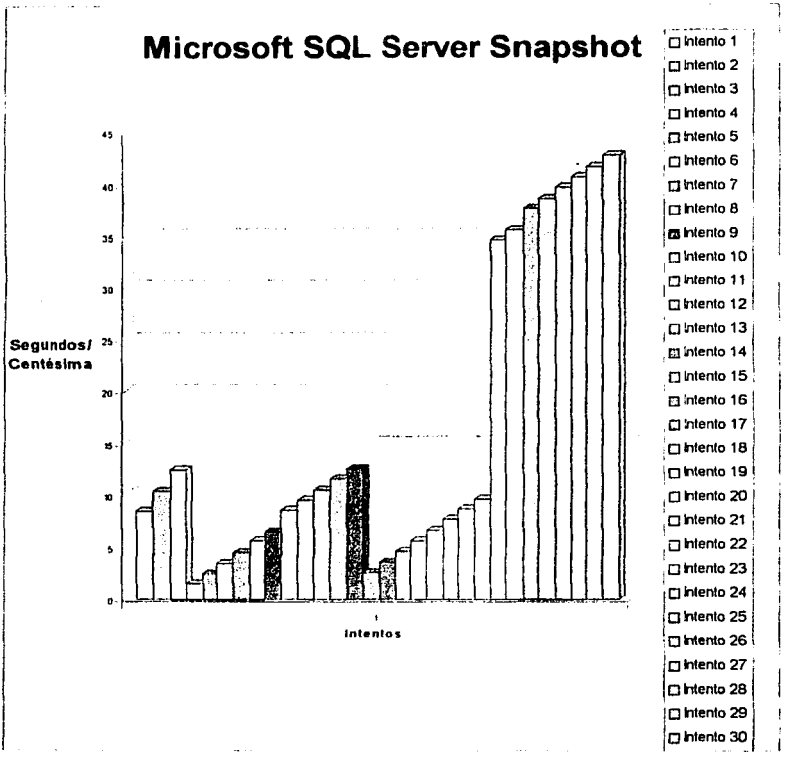
Este tipo de Replicación solo es permitida al adquirir la versión estándar o la empresarial de Microsoft SQL Server 2000. Por esta razón solo se han hecho pruebas con el tipo de replicación snapshot. Es primordial hacer mención que la replicación tipo snapshot se realiza ejecutando un comando en el software del manejador de replicación. Esto es, después de realizar una transacción o un conjunto de transacciones se puede ejecutar el comando mencionado y se realiza la replicación o también puede programarse el comando para realizar la replica en determinados períodos de tiempo.

Los resultados de tiempo para 30 envíos de sentencias SQL de un servidor a otro, entre intervalos variados, mediante el método snapshot se muestran en la tabla 3.4. mostrada a continuación:

Microsoft SQL Server Snapshot

Transacción	Tiempo en Segundos / Centésima
Intento 1	8.34
Intento 2	10.36
Intento 3	12.38
Intento 4	1.39
Intento 5	2.4
Intento 6	3.41
Intento 7	4.42
Intento 8	5.53
Intento 9	6.44
Intento 10	8.46
Intento 11	9.47
Intento 12	10.48
Intento 13	11.49
Intento 14	12.5
Intento 15	2.52
Intento 16	3.53
Intento 17	4.54
Intento 18	5.55
Intento 19	6.56
Intento 20	7.57
Intento 21	8.58
Intento 22	9.59
Intento 23	34.6
Intento 24	35.61
Intento 25	37.63
Intento 26	38.64
Intento 27	39.65
Intento 28	40.66
Intento 29	41.67
Intento 30	42.68
Promedio o media	15.555

Tabla 3.4 Tabla de resultados en segundos / centésima de un total de 30 transacciones enviadas individualmente, mediante el método de replicación Snapshot de SQLServer



Gráfica 3.2 Gráfica de resultados en segundos / centésima de un total de 30 transacciones enviadas individualmente, mediante el método de replicación Snapshot de SQLServer

Sybase Adaptive Server Anywhere, SQL Remote

Al igual que el tipo de replicación Snapshot, SQL Remote permite replicar de manera periódica y constante; el sistema esta diseñado para correr bajo cualquier plataforma Windows y solo funciona con Bases de Datos de Sybase, dependiendo su versión por supuesto, este sistema es económico (como se pudo ver en la sección anterior) y mucho más rápido que el software de Microsoft en su modalidad snapshot como se mostrará a continuación.

Al igual que la replicación tipo snapshot, este sistema realiza la ejecutando un comando en el software del manejador de replicación. Esto es, después de realizar una transacción o un conjunto de transacciones se ejecuta el comando mencionado y se realiza la replicación o también puede programarse el comando para realizar la replica en determinados períodos de tiempo.

Es importante recalcar que tanto para el software SQLRemote como para el de Microsoft se ha despreciado el tiempo que se desperdicia entre enviar la transacción en la aplicación y la ejecución del comando en el software de replicación; que es de aproximadamente 7 segundos para el sistema de Microsoft y 3 segundos para el de Sybase.

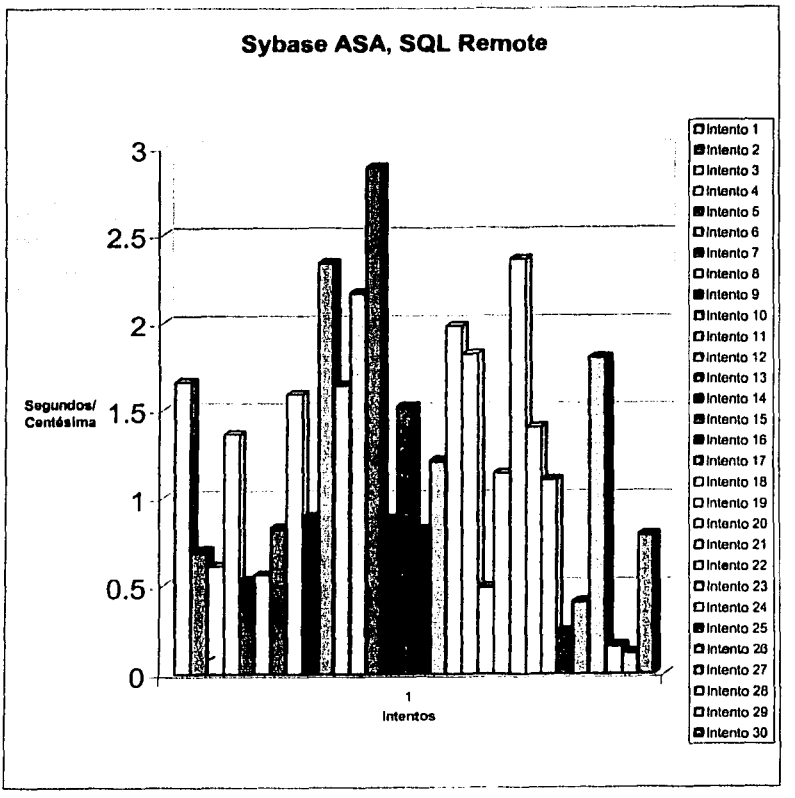
El sistema de replicación, en teoría puede hacerlo entre diferentes plataformas a través de Open Server, pero los costos de este software son elevados como se comento en la sección anterior.

Los resultados de tiempo para 30 envíos de sentencias SQL de un servidor a otro, entre intervalos variados, mediante el método snapshot se muestran en la tabla 3.5. mostrada a continuación:

Sybase ASA, SQL Remote

Transacción	Tiempo en Segundos / Centésima
Intento 1	1.66
Intento 2	0.7
Intento 3	0.61
Intento 4	1.36
Intento 5	0.53
Intento 6	0.56
Intento 7	0.83
Intento 8	1.59
Intento 9	0.89
Intento 10	2.34
Intento 11	1.64
Intento 12	2.17
Intento 13	2.89
Intento 14	0.88
Intento 15	1.52
Intento 16	0.82
Intento 17	1.21
Intento 18	1.98
Intento 19	1.82
Intento 20	0.49
Intento 21	1.14
Intento 22	2.36
Intento 23	1.4
Intento 24	1.1
Intento 25	0.24
Intento 26	0.41
Intento 27	1.8
Intento 28	0.16
Intento 29	0.12
Intento 30	0.79
Media o Promedio	1.200333333

Tabla 3.5 Tabla de resultados en segundos / centésima de un total de 30 transacciones enviadas individualmente, mediante SQLRemote en Adaptive Server Anywhere de Sybase



Gráfica 3.3 Gráfica de resultados en segundos / centésima de un total de 30 transacciones enviadas individualmente, mediante SQLRemote en Adaptive Server Anywhere de Sybase

MySql y Access 97 con el SRP

Para que el sistema de replicación propuesto funcione adecuadamente se requieren ciertos archivos ubicados en localidades especificas dependiendo la modalidad del software que puede ser cliente o servidor.

En su modalidad de cliente, para el sistema de replicación propuesto los siguientes archivos ubicados en el directorio del sistema de Windows de sistema son requeridos: el archivo Loyal.ini, en cuya configuración se debe especificar la dirección IP y el puerto por el cual el servidor espera transacciones, la DLL ObtenIpPuerto.dll y Loyal.dll que se encargan de leer el archivo ini y mandar los mensajes respectivamente.

Es necesario iniciar la opción de seguimiento en el administrador de controladores, antes de ejecutar la aplicación; posteriormente puede cerrarse la ventana del administrador de controladores e iniciar la aplicación, después se presentara la pantalla del administrador de controladores para seleccionar un origen de datos y conectarse a la Base de Datos indicada en dicho origen y seguir el flujo de procesos descrito en la sección de aplicación.

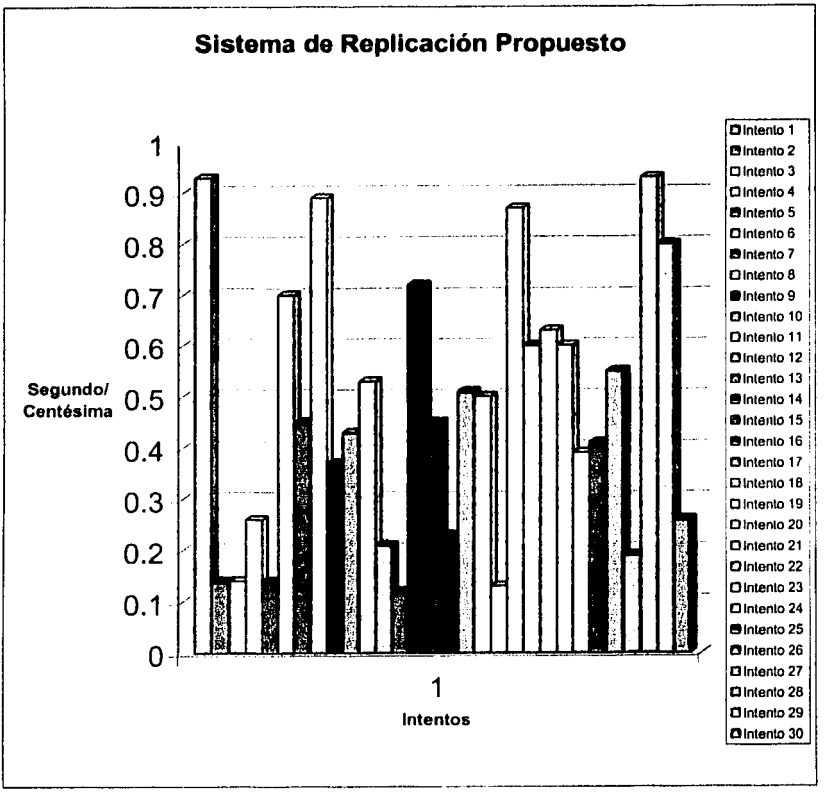
Así de esta forma cada transacción realizada se registrará en el archivo de seguimiento, ya que este hará las llamadas a las API's de ODBC del origen de datos que serán monitoreadas por el administrador de controladores, y dichas transacciones serán enviadas por el sistema de replicación en forma de mensajes.

El sistema de replicación funciona de tal manera que permite la replicación en tiempo real, a continuación se muestran los resultados de tiempo para 30 envíos de sentencias SQL de un servidor a otro, entre intervalos variados:

Sistema de Replicación Propuesto

Transacción	Tiempo en Segundos / Centésima
Intento 1	0.93
Intento 2	0.14
Intento 3	0.14
Intento 4	0.26
Intento 5	0.14
Intento 6	0.7
Intento 7	0.45
Intento 8	0.89
Intento 9	0.37
Intento 10	0.43
Intento 11	0.53
Intento 12	0.21
Intento 13	0.12
Intento 14	0.72
Intento 15	0.45
Intento 16	0.23
Intento 17	0.51
Intento 18	0.5
Intento 19	0.13
Intento 20	0.87
Intento 21	0.6
Intento 22	0.63
Intento 23	0.6
Intento 24	0.39
Intento 25	0.41
Intento 26	0.55
Intento 27	0.19
Intento 28	0.93
Intento 29	0.8
Intento 30	0.26
media o promedio	0.43366667

Tabla 3.6 Tabla de resultados en segundos / centésima de un total de 30 transacciones enviadas individualmente, mediante el Sistema de Replicación Propuesto



Gráfica 3.4 Gráfica de resultados en segundos / centésima de un total de 30 transacciones enviadas individualmente, mediante el Sistema de Replicación Propuesto

Evaluando los resultados obtenidos se puede apreciar un alto rendimiento, en cuestión de velocidad de replica, para el sistema de replicación propuesto seguido por el sistema SQLRemote de Sybase y por último el sistema SQL Server 2000 de Microsoft.

Aunque hay que tomar en cuenta que el sistema de replicación no tiene un control de errores o de concurrencia.

3.3 CUADRO COMPARATIVO DEL SOFTWARE PROPUESTO CONTRA OTROS SOFTWARE'S COMERCIALES

Evaluando los resultados obtenidos en la tabla anterior podemos considerar el software de replicación propuesto como un sistema dentro de los parámetros aceptable; sin embargo para evitar opiniones premeditadas en la siguiente sección se evaluarán las ventajas y desventajas del sistema de replicación propuesto.

Sistema de Replicación	Soporta Multiplataforma	Soporta ODBC	Replica en Tiempo Real	Replica en Ambos Sentidos	Permite Especificar Tablas a Replicar	Tiempo < 2 seg. En Transacción simple	Costo < $2000 dólares
SRP	SI	SI	SI	NO	NO	SI	SI
SQLRemote Sybase	NO	SI	NO	SI	SI	SI	SI
Replication Server	NO	SI	SI	SI	SI	SI	NO
Microsoft SQL Server	NO	SI	SI	SI	SI	NO	NO

Tabla 3.7 Cuadro Comparativo del Sistema de Replicación Propuesto contra otros Software's Comerciales

3.4 VENTAJAS

Tecnología Multihilos en el cliente

La tecnología multihilos consiste en la administración de procesos en una computadora, esta tecnología permite que un proceso se ejecute libremente el tiempo que sea necesario, pero si ese proceso es pospuesto por necesidades del sistema no debe ocupar tiempo de procesamiento adicional (tiempo de CPU) mientras se encuentra en estado latente o pasivo, esta es la tarea principal del procesamiento multihilos además de permitir ejecutar varias instancias de ese mismo proceso (hilos).

El sistema de replicación propuesto, permite ejecutar el seguimiento de transacciones mediante la DLL de seguimiento, cuyo funcionamiento ha sido descrito anteriormente. La DLL es capaz de ejecutar el proceso de envío de mensajes usando la tecnología multihilos; sin embargo esto ocurrirá si existe otra aplicación dentro de la misma máquina que utilicé el mismo proceso (se envíe un mensaje en una segunda aplicación pero en la misma máquina).

Número ilimitado de clientes

Una ventaja importante es que los clientes utilizan ODBC, un programa que es gratuito y que es incluido al obtener la licencia de cualquier sistema operativo Windows; de esta manera el sistema de replicación propuesto puede ser instalado, en su modalidad de cliente o servidor, en cualquier computadora que cuente con cualquier licencia de Windows.

Multiplataforma y Manejadores Distintos

A pesar de que el sistema de replicación propuesto esta diseñado para instalarse en plataformas Windows exclusivamente, como se menciono anteriormente, mediante ODBC puede comunicarse a diferentes manejadores en diferentes plataformas dependiendo del controlador ODBC.

Esto es una gran ventaja ya que es común la necesidad de replicar entre diferentes plataformas, incluso existen programas (puertas de enlace) creados específicamente para replicar entre dos manejadores o plataformas diferentes, como es el caso del gateway Open Server Sybase-Oracle.

Soporta diferentes Arquitecturas de Hardware

El sistema de replicación propuesto puede correr en cualquier arquitectura donde se pueda instalar Windows desde la versión 95 para PC hasta las versiones actuales; no requiere equipos específicos pero es obvio que entre mejor sean las características del Hardware mejor se comportará el sistema.

Uso de Memoria Mínimo

El SRP esta diseñado para utilizar el mínimo de recursos de memoria en la PC, a diferencia de otros sistemas donde el uso de recursos de memoria es alto, esta ventaja permite un funcionamiento optimo entre plataformas de Windows muy distintas; de tal forma que el rendimiento durante la replicación es aceptable.

Costo

Los sistemas de replicación comerciales son caros, el sistema de replicación propuesto puede ahorrar costos de hasta $ 3000.00 dólares por servidor, más las

Si una empresa desarrollara este software al tener número ilimitado de clientes, implica que no hay que pagar ninguna licencia ya que es un software que ha sido desarrollado por la misma.

3.5 DESVENTAJAS

No selecciona Tablas a Replicar

El sistema de replicación propuesto, replica absolutamente todas las transacciones que se ejecuten de una Base de Datos a otra; no permite, como en algunos software's comerciales seleccionar las tablas que se van a replicar.

Replica en un solo Sentido

El sistema de replicación solo replica en un solo sentido. El sistema de replicación esta diseñado para enviar mensajes, en realidad transacciones de un cliente a un servidor, el envió de los mensajes se basa en un sistema de seguimiento de ODBC y si tuviéramos una replicación en dos sentidos deberíamos tener también dos servidores escuchando peticiones y dos clientes enviando mensajes continuamente a través del sistema de seguimiento, esto es cada máquina sería al mismo tiempo servidor y cliente, imaginemos que se ejecuta una transacción en algún cliente y este mediante el sistema de seguimiento envía la petición a un servidor que escucha, en el momento que el servidor ejecuta la transacción contenida en el mensaje recibido el sistema de seguimiento envía de nuevo un mensaje con la transacción ejecutada recientemente a un servidor y repetiría el proceso indefinidamente. Prácticamente ésta es una de las principales desventajas del sistema.

El cliente es quién replica

Tal vez la principal desventaja es ésta, ya que la mayoría de los software's comerciales replican la información de servidor a servidor. El sistema de replicación propuesto esta instalado en un cliente que mediante una aplicación, utilizando ODBC, se conecta y manda peticiones a un servidor llamémosle primario; hasta este punto llevamos el esquema de un sistema cliente - servidor

normal; cuando el cliente envía peticiones al origen de datos especificado en el manejador de controladores de ODBC (concretamente el servidor primario) se realiza un seguimiento de las transacciones que ejecutó el cliente y se envían las transacciones al servidor en el cual se desee replicar la información; esto sucede teniendo previamente instalado el sistema de replicación propuesto en su modalidad de cliente. Así se muestra que es el cliente quien realiza la replicación para el prototipo; esto puede ocasionar problemas si no existe un diseño adecuado en la aplicación; es decir la aplicación debe estar construida considerando que es muy probable que muchos clientes se conecten al servidor primario y envíen transacciones, y por ende repliquen información al servidor secundario. Debe existir un control de concurrencia, no solo el que proporciona el manejador de Base de Datos, sino además de la aplicación; la cual debe estar diseñada para controlar el acceso de manera coherente a la Base de Datos.

No Existe un Control de errores

Los sistemas de replicación actuales cuentan con un control de errores en caso de que alguna transacción no se haya replicado satisfactoriamente, algunos sistemas dan de baja el servidor automáticamente y guardan en el log de transacciones una clave con la cual identifican las transacciones que deben de replicarse una vez que el servidor se haya recuperado, otros sistemas simplemente mandan un mensaje y continúan replicando.

El sistema de replicación propuesto carece de un control de errores, esto es porque diseñar todo un sistema de control de errores implica tiempo dependiendo de que tan complejo se desee hacer con el control de dichos errores y esta fuera del alcance de este prototipo; sin embargo el sistema de replicación propuesto en el cliente hace un registro de las transacciones que se van ejecutando lo cual puede servir para ubicar que transacciones se han ejecutado.

Conclusiones del Capítulo 3

Los resultados de la evaluación plasmados en este último capítulo muestran claramente que el sistema de replicación propuesto esta al nivel de al menos algunas versiones estándar de muchos sistemas de replicación.

Algunas de las ventajas del sistema de replicación propuesto son: número ilimitado de clientes, Multiplataforma y Manejadores Distintos Soporta diferentes Arquitecturas de Hardware, el costo es mínimo.

Las desventajas principales son: No Existe un Control de errores, replica en un solo sentido y no selecciona tablas a replicar. Esto esta fuera del alcance de este trabajo, pero debe ser el siguiente punto a mejorar del software propuesto.

CONCLUSIONES

El sistema de replicación propuesto cumple con la función principal de un sistema de este tipo que es duplicar información; con este diseño se demuestra que el software desarrollado, aun a este nivel, es capaz de resolver muchos problemas reales en múltiples aplicaciones.

En México aun no existe mucho auge en la aplicación de este tipo de sistemas, precisamente porque el costo de adquisición es muy elevado, por eso mismo el sistema de replicación propuesto brinda una ventaja inmensa al ser aplicado a un problema real.

El siguiente paso de este proyecto tiene muchas variantes, sin embargo todas llevan al mismo camino mejorar el sistema de replicación propuesto, puede ser una buena alternativa diseñar el sistema de replicación de tal forma que sea un servidor quien envíe mensajes de datos y sea otro servidor quien los reciba y los procese; además de que los clientes conectados a ese servidor puedan visualizar los datos replicados, o sea que la replicación la realice el servidor, otro aspecto importante seria el diseño de un programa administrador de replica en forma gráfica para el usuario y la replicación en ambos sentidos; es decir que una terminal pudiera ser servidor y cliente al mismo tiempo, esta situación nos llevaría al desarrollo de un nuevo administrador de orígenes de datos a nivel servidor que tendría la capacidad de generar nuevos orígenes de datos apuntando hacia algún cliente en especifico, así cuantos clientes fueran necesarios, y también debe de hacer un seguimiento de las transacciones que se realicen para mandar dichas transacciones al servidor con el cual se desee replicar datos. El prototipo de replicación propuesto puede adaptarse a sistemas donde el número de terminales conectadas a un servidor sea mínimo; a pesar de que el sistema no ha sido probado en una WAN, porque una infraestructura de tal magnitud no estaba a la disposición del autor de la tesis, es muy probable que el funcionamiento sea el mismo que el comportamiento en la infraestructura simulada.

A lo largo del diseño del sistema se investigaron métodos efectivos de diseño como el UML (Lenguaje Unificado de Modelado) el cual sirvió para crear esquemas sencillos de los procesos que se efectúan durante la replicación y dar un seguimiento de estos mismos esto con el fin de generar un sistema lo suficientemente documentado y en cuyos documentos se plasmara todo el análisis; durante el desarrollo se tuvieron que conocer diversas técnicas para enviar datos, para crear una DLL, para acceder bases de datos y se seleccionaron las más adecuadas que satisficieran nuestras expectativas respecto al tiempo de envío, este trabajo enriquece la línea de investigación en el estudio de Bases de Datos y en particular el realizarlo fue una invaluable experiencia profesional.

BIBLIOGRAFÍA

Cisneros González José Luis, Panorama Sobre Bases De Datos (Un Enfoque Practico)
Universidad Nacional Autónoma de Baja California
México, 1998.

Cristopher S. Lester, Bases De Datos Con Visual J++
Prentice Hall
Madrid, 1998.

Deen S.M., Fundamentos De Los Sistemas De Bases De Datos
Editorial Gustavo Gili S.A.
Barcelona, 1987.

Divya Chaturvedi, Paritosh Pathak, Administering SQL Server 7
Mc Graw Hill
E.U.A., 1999

DuBois Paul, Edición Especial MySQL
Prentice Hall
México, 2001

Elmasri y Navathe, Sistemas De Bases De Datos Conceptos Fundamentales
Adison Wesley Iberoamericana S.A.
Wilmington, Delaware. E.U.A.

Garcia Molina Hector, Ullman Jeffrey y Widom, Database System Implementation
Prentice Hall
E.U.A. 2000

Kyle Geiger, Inside Odbc
Microsoft Press
E.U.A. 1995

Pascual, Charte, Segarra, Antonio y Clavijo, Programación Avanzada en Windows 2000
McGraw Hill
España 2000

Ramakrishna y Johannes, Database Management Systems
Mc Graw Hill
E.U.A. 2000

Sanders Roger, Odbc 3.5 Developer's Guide
McGraw Hill
E.U.A. 1998

Sanjiv, Purba, Developing Client/Server Systems using Sybase SQL Server
Wiley Computer Publishing
E.U.A. 1999

Seymour, Lipschutz, Estructura De Datos
Editorial Mc Graw-Hill
México, 1989

Tsai Alice y H., Sistemas De Bases De Datos, Administración Y Uso
Ed. Prentice Hall Hispanoamericana
México, 1990

Manuales Técnicos

SQL SERVER 2000 SYSTEM ADMINISTRATOR
ORACLE 8 CONCEPTS RELEASE 8.0 (AYUDA ORACLE8)
MANUAL MYSQL ADMINISTRATOR
ADMINISTRATOR'S GUIDE FOR INFORMIX DINAMIC SERVER
INSTALATION GUIDE FOR INFORMIX DINAMIC SERVER

Páginas Web
www.mysql.com/doc/M/y/MySQL_Benchmarks.html
www.microsoft.com/latam/sql/evaluation/features/default.asp
www-3.ibm.com/software/data/informix/ids/Enterprise/
www.oracle.com/ip/deploy/database/oracle9i/gsassess.html
Microsoft Data Access Components (MDAC) SDK versión 2.6, extraido de:
http://msdn.microsoft.com/downloads/

GLOSARIO DE TÉRMINOS TÉCNICOS

ANSI (American National Standards Institute)	Uno de los primeros estándares- escenarios para la tecnología computacional en los Estados Unidos.
API (Application programming interface)	Interfaz de programación de aplicaciones, un conjunto de funciones relacionadas que el programador de computadoras usa para obtener algún tipo de servicio desde otra pieza de software. Programadores de aplicaciones basadas en Windows usan la API de Windows para crear ventanas, dibujar texto en pantalla, acceder archivos. Se desconoce el funcionamiento interno de las funciones pero se sabe bien que resultados producirán y se proporciona un punto de entrada para mandar a llamar a la función API.
ASE (Adaptive Server Enterprise)	Se trata de un manejador de base de datos marca Sybase.
ATOMICIDAD	La ejecución de todas o ninguna transacción.
CLI (call level interface)	Interfaz de nivel de llamada, un conjunto de funciones (como API definida previamente). Sin embargo, CLI es usada en los estándares SQL mundiales para describir una interfase que no sea SQL incrustado, el cual también es referenciado como una API.
COMMIT	Se dice así cuando una transacción o grupo de transacciones se comprometen, es decir se realizan.
CLUSTERED	Agrupado, aglomerado; cuando un índice es "clustered" quiere decir que se agrupan varios patrones de orden (campos).
DBMSs (Database Management System)	Plural, en inglés, de sistema manejador de base de datos.

ISO (International Organization for Standardization)	Una federación mundial de estándares internacionales que establecen estándares para una amplia variedad de tecnologías, incluyendo lenguajes de computación.
LAN (Local Area Network)	Red de área local, es una red pequeña que intercomunica varias computadoras ubicadas localmente en un determinado sitio.
METADATOS	El esquema de información para el manejador de base de datos o información específica del mismo.
ODBC (Open Database Connectivity)	Conectividad abierta de bases de datos, este programa sirve como puerta de enlace entre una aplicación y alguna base de datos; su función principal es conectar a una base de datos mediante la programación mediante algún lenguaje, cabe señalar que la base de datos requiere ciertos controladores (drivers) previamente instalados.
ROLLBACK	Se dice así cuando una transacción o grupo de transacciones abortan, es decir no se realizan.
SCHEDULER	Programador de tareas, permite programar una tarea redundante; por ejemplo el respaldo de una base de datos.
THREAD	Hilo, se llama así a un proceso específico independiente, que puede volverse a llamar
TRANSACCIÓN	La unidad de trabajo que debe ejecutarse atómicamente y en aparente aislamiento de otras transacciones
WAN (Wide Area Network)	Red de área amplia, es una red "amplia" que intercomunica varias computadoras ubicadas, tal vez, geográficamente en distintos sitios.

```
/*

TESIS: PROTOTIPO DE REPLICACIÓN DE BASES DE DATOS MEDIANTE ODBC
AUTOR:                DANIEL LEAL LARIS
FECHA DE CREACIÓN:  06/07/2002

NOMBRE:   LoyalServer.exe

Este código hace la función de Servidor y realiza las
siguientes tareas en el siguiente orden:

1.- Acepta los siguentes parámetros:
   a) Dirección IP del Servidor
   b) Puerto de conexión
      (es decir el puerto por el cual el servidor recibirá peticiones )
   c) Nombre del Servidor de Base de Datos local (DBMS)
      de acuerdo con el DNS de ODBC previamente creado
   d) Nombre del usuario con el que se hará la conexión
      de acuerdo con el DNS de ODBC previamente creado
   c) Contraseña del usuario


2.- Verifica que se soporte la tecnologia socket de escuchar peticiones
3.- Se conecta a una base  de datos
4.- Comienza a Escuchar Peticiones de Sentencias SQL
5.- Ejecuta las Sentencias SQL que van llegando


ADVERTENCIA: Para que este programa se pueda compilar, es necesario incluir la librería
Wsock32.lib en project -> settings -> link.


A diferencia de la codificación de la DLL que se  ubica en el cliente, este programa
esta creado bajo los estándares de la programación orientada a objetos.

Para el mejor manejo del archivo ejecutable se puede correr el archivo por lotes (.bat)
LoyalServer.bat donde se pueden especificar los parámetros previamente definidos,
el exe  junto con su archivo por lotes puede estar ubicado en cualquier parte del sistema,
y se ejecutara correctamente siempre y cuando sus parámetros sean validos.

*/

#include <winsock.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <iostream.h>
#include "sql.h"
#include "sqlext.h"
//Librerias Creadas
#include "LoyalServer.h"
#include "OdbcCon.h"
int main(int argv1,  char **argv2)
{

 int t=0,
 int a=0,

 char *stopstring,
 char DatosEntrada[2000],
 char  DatosSalida[2000],

 //El stm (sentencia) se copiara a  esta variable
 char          szBuffer[2000],
 char textopantalla[600]
 char  *pdest = NULL,
 char          AuxszBuffer[2000],
 const char *SrcIPAddr=  argv2[1],
 static int nummsg=0,

 int x=0,
 int y=0,
 int z=0,
 int w=0,
 int  *DatoInt,
 int IPPort,
 int res,
```

```
char peticion='0';
long cuantosLeidos;
IPPort = (int) strtod( argv2[2], &stopstring );
printf("\n %s",SrcIPAddr);
printf("\n %s",IPPort);
textopantalla[500]=0;
DataInt  = (int *) malloc(sizeof(int));
WSADATA wsaData;
WORD     sockVersion = MAKEWORD(1,1);
 //Validando si se soporta la comunicacion mediante SOCKETS
 res = WSAStartup(sockVersion, &wsaData);

if (res == 0)
{
  //Crear una instancia de la clase Servidor
   CServidor servidor(SrcIPAddr,IPPort);
   printf("\n Servidor Escuchando Peticiones del Cliente") ;
   SQLRETURN   rc  = SQL_SUCCESS;
   UCHAR * servidorStr   = (UCHAR *)argv2[3];
   UCHAR * usuario       = (UCHAR *)argv2[4] ;
   UCHAR * contrasenia   = (UCHAR *)argv2[5];

  // Crear una instancia de la clase de ODBC
   ODBC_Class CnBd;
  // Conectarse a la base de datos apropiada - si es satisfactoria,
  // Despliegar el mensaje apropiado

if (CnBd.CnHandle != NULL)
{
  // Inicia Conexion
  rc = SQLConnect(CnBd.CnHandle, servidorStr, SQL_NTS, usuario, SQL_NTS, contrasenia, SQL_NTS);
  if (rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO)
    {
     printf("\n Servidor LoyalServer Conectado Satisfactoriamente a la Base de Datos "
      rc = SQLAllocStmt(CnBd.CnHandle, &CnBd.StmtHandle);
    // Verifica se acepten sentencias SQL en el Manejador de Base de Datos
              if( rc == SQL_SUCCESS)
        {
        do{
         //Inicia servicio de escucha
              if(servidor.servicio(DatosEntrada,2000,cuantosLeidos)==OK)
                {
                 pdest = NULL;
                 DatosEntrada[cuantosLeidos]=0;
                 DatosSalida[cuantosLeidos]=0;
                szBuffer[cuantosLeidos]=0;
                _snprintf(szBuffer,sizeof(szBuffer)," %s \n",DatosEntrada);
                _snprintf(AuxszBuffer,sizeof(AuxszBuffer)," %s \n",DatosEntrada);
                pdest = strstr(strupr(AuxszBuffer),"SQLPREPARE");

                    if (pdest != NULL  || x>0)
                        {
                        x++;
                        if (x<=1)
                        {
                    sprintf(textopantalla,"\n\n Msg(%d) \n\n Preparando /
                    Sentencia SQL [%s]", ++nummsg, szBuffer);
                        printf("%s", textopantalla);

                  // Prepara Sentencia

                        if(SQLPrepare(CnBd.StmtHandle, (UCHAR *)
                    szBuffer, SQL_NTS) != SQL_SUCCESS)
                                {
                                 printf("\n Existio un Problema al Preparar Sentencia");
                                }
                                 else
                                {
                                 printf("\n La Sentencia se Preparo Satisfactoriamente");
                                }
                            x=0;
                        }
              }

              pdest = strstr(strupr(AuxszBuffer),"SQLEXECUTE");

              if (pdest != NULL  || w>0)
              {
                w++;
                if (w>0)
                {
                 t=0;
            a=0;
                printf("*w",w);
                sprintf(textopantalla,"\n\n Msg(%d) \n\n Ejecutando Sentencia SQL ", ++nummsg);
                printf("%s", textopantalla);
                // Ejecuta Sentencia
```

```
    if(SQLExecute(CnBd.StmtHandle) != SQL_SUCCESS)
{
            printf("\n Existio un Problema al Ejecutar Sentencia");
    }
    else
    {
        printf("\n La Sentencia se Ejecuto Satisfactoriamente");
    }
            _flushall();
            w=0;
     }

}
                    pdest = strstr(strupr(AuxszBuffer),"SQLEXECDIRECT");
                if (pdest != NULL || z>0)
    {
                  z++;
                        if(z>1)
        {
                    sprintf(textopantalla,"\r\n Msg(%d). \r\n Sentencia SQL a Ejecutar (%s)",
            ++nummsg, szBuffer);
            printf("%s", textopantalla);
            // Ejecuta Sentencia
              if(SQLExecDirect(CnBd.StmtHandle, (UCHAR *) szBuffer, SQL_NTS) != SQL_SUCCESS)
                {
                        printf("\n Existio un Problema al Ejecutar Sentencia");
                }
                    else
                {
                            printf("\n La Sentencia se Ejecuto Satisfactoriamente");
                    }

                z=0;
        }

         }

     }

    } while(peticion!='d');

   }
  else
{
    printf("\n El sistema operativo no Permite Sentencia (SQLAllocStmt)
a la Base de Datos");
    }
}else
            {
                printf("\n El sistema operativo no Permite Conexión
        (SQLConnect) a la Base de Datos");
            }

    }else

      {
          printf("\n El sistema operativo no Permite Conexión
          (SQLAllowHandle) a la Base de Datos");
       }

          }else

              {
                  printf("\n El sistema operativo no soporta
                  la comunicación por Sockets");
              }

              return 0;
       }
```

```
//Header Servidor  LoyalServer.h

#include <winsock.h>
#include <sys\types.h>
#include <stdio.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <iostream.h>
const int DefaultTTL=1;
const int ERRBASE=65535;
const int OK=0,
const int ESEND          =          ERRBASE+1;
const int ERECV          =          ERRBASE+2;
const int ETIMEOUT  = ERRBASE+3;
class Csockbase
{
 public
  struct sockaddr_in addr;        dirección
  int          lenaddr;       longitud de la direccion
  int          sock,          socket
  int                                                  err;
  int                                                  retval;
};

class CServidor : protected Csockbase
{
 long bytes_recibidos;
 struct sockaddr_in addrClient, *pAddrCliente;

 public:
 CServidor(const char *SrcIPAddr=0,int IPPort=0);
 ~CServidor();

 int servicio(char *buffdatosin, long maxDatosIn, long &datosleidos);

};
//Aqui creo la instancia
CServidor::CServidor(const char *SrcIPAddr,int IPPort)
{

if ((sock = socket(AF_INET, SOCK_STREAM, 0 ) < 0)
            printf("Error creando socket SOCK_STREAM");

            memset( &addr, 0, sizeof(addr) );
            addr.sin_family = AF_INET;

            if(SrcIPAddr)
          addr.sin_addr.s_addr = INADDR_ANY;
            else
              addr.sin_addr.s_addr=inet_addr(SrcIPAddr);
              addr.sin_port=htons(IPPort);
              lenaddr=sizeof(addr);

            //Se asocia el socket TCP a la dirección de recepcion

            if (bind( sock, (struct sockaddr *) &addr, lenaddr) == -1 )
              throw "Error en bind";
                if(getsockname(sock,(struct sockaddr *)&addr,&lenaddr)==-1)
      throw ("error en getsockname");
            listen( sock, 5 );

}
CServidor::~CServidor()
{
  //destruir la conexion que activa y hacer limpieza
  closesocket(sock);
}
int CServidor::servicio(char *buffdatosIn, long maxdatosInOut,
long &datosleidos)
{
 //Primero se construye la estructura de timeouts
 //Un conjunto de sockets a comprobar si tienen peticiones
 fd_set set;
 FD_ZERO(&set);
 FD_SET(sock,&set);
 // No hay tiempo máximo de esperar si hay buffer de entrada
 struct timeval tval;
 tval.tv_sec=0;
 tval.tv_usec=NULL;
 int result=select(0,&set,0,0,&tval);
 if(result==0) return ETIMEOUT; //retornar si cumple tiempo espera
 //hay petición y se procesa
 int sockaceptaccept(sock,(struct sockaddr *) &addr, (int *) &lenaddr);
 if(sockaccept==-1)
   throw "Error en accept";
 //obtenemos datos del socket creado por el sistema para atender
 //el servicio
   int lenaddrservicio = sizeof(addr);
  if( getsockname( sockaccept, (struct sockaddr *) &addrservicio,
&lenaddrservicio ) )
   throw("Error averiguando sockaddr_in del socket asignado para el servicio");
```

```
  //obtenemos los datos del cliente conectado al socket
 int lenaddrcliente= sizeof addrcliente;
   if( getpeername( sockacept, (struct sockaddr *) &addrcliente,
&lenaddrcliente ) ==-1)
   throw("Error averiguando sockaddr_in del socket cliente conectado a este"),

   //Con los datos del cliente se podria decidir si se acepta o no
   //la petición...
   //Se reciben los datos
   retval=recv(sockacept bufdatosin,maxdatosinOut,0),
   if (retval==-1)
    throw("Error en recv"
   datosleidos=retval;
                  //damos por terminada la comunicación

   closesocket(sockacept
   return OK,
 }
```

```
#include <windows h>
#include <sql h>
#include <sqlext h>

//Clase de Conexion a la Base de Datos


// Define la Clase de Conexion ODBC_Class
class ODBC_Class
{
   // Atributos
   public:
      SQLHANDLE   EnvHandle
      SQLHANDLE   ConHandle
      SQLHANDLE   StmtHandle
      SQLRETURN   rc

   // Operaciones
   public
   ODBC_Class();                    Constructor
   ~ODBC_Class();                   Destructor
},

// Define el Constructor de la Clase
ODBC_Class::ODBC_Class()
{
// Inicializa la variable de Codigo de Retorno
rc = SQL_SUCCESS

// Almacena espacio para un Handle de Ambiente
rc = SQLAllocHandle(      SQL_HANDLE_ENV, SQL_NULL_HANDLE, &EnvHandle),

 // Asigna la Aplicacion ODBC a la Version 3 x
 if (rc == SQL_SUCCESS
    rc = SQLSetEnvAttr(EnvHandle, SQL_ATTR_ODBC_VERSION
(SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UINTEGER)

 // Almacena espacio para un Handle de Conexion
   if (rc == SQL_SUCCESS
      rc = SQLAllocHandle( SQL_HANDLE_DBC, EnvHandle, &ConHandle),
}

// Define el Destructor de la Clase
ODBC_Class::~ODBC_Class()
{
   // Libera el Handle de Conexion
   if (ConHandle != NULL
      SQLFreeHandle(SQL_HANDLE_DBC, ConHandle),

   // Libera El Handle de Ambiente
   if (EnvHandle != NULL
      SQLFreeHandle(SQL_HANDLE_ENV, EnvHandle),
}
```

APÉNDICE B

/*

TESIS: PROTOTIPO DE REPLICACIÓN DE BASES DE DATOS MEDIANTE ODBC
AUTOR: DANIEL LEAL LARIS
FECHA DE CREACIÓN: 15/07/2002

NOMBRE: Loyal.dll

init.c - This module contains the DLL initialization functions

Esta DLL fue modificada de su versión Original.

Objetivo:

Realizar un seguimiento, no tan detallado como en la versión original, donde se registren exclusivamente las sentencias SQL que han sido ejecutadas de forma satisfactoria.

En la presente versión de la DLL, solo se incluyen las siguientes funciones para su seguimiento:

TraceSQLExecDirect
TraceSQLPrepare
TraceSQLExecute

Aqui es importante aclarar que, en una aplicación ODBC, para ejecutar transacciones no solo se usan estas funciones, pero para fines de este prototipo se tomaron en cuenta las mas comunes.
El proceso principal se describe a continuación :

a) Inicializar DLL, obtener Dirección Ip del servidor y el puerto
por el cual esté escuchando mensajes
mediante la DLL ObtenIPPuerto.dll

b) Llamar a la función de seguimiento

c) Dentro de la función ProcessTrace:
si la sentencia no es SELECT (puede ser INSERT, UPDATE ó DELETE),
cópiala a un Buffer, escríbela en el Archivo de Seguimiento y
Mándala mediante un socket al Servidor que este Replicando
[LoyalServer] en algun lugar de la RED por la función SendSocket

Puesto que la presente DLL es una adaptación de una DLL que es ejecutada por el manejador de controladores, que no esta bajo el estándar de orientado a objetos, el formato de las funciones ha quedado igual no se ha cambiado mucho, ha excepción de la implementación de una nueva función, y la
modificación de una nueva función:

1.- Una función externa desde otra DLL (ObtenIpPuerto.dll) cargada explícitamente que como lo dice su nombre obtiene la IP y el Puerto del archivo "Loyal.ini"
donde se especifica la dirección y el puerto del servidor a Replicar.
2.- La modificación de la función 'ProcessTrace', para realizar la tarea
descrita anteriormente en el inciso c

ADVERTENCIA: Para que este programa se pueda compilar, es necesario incluir la librería Wsock32.lib en project -> settings -> link.
UNA VEZ GENERADO ESTA DLL ES NECESARIO COPIARLA AL DIRECTORIO DEL SISTEMA DE WINDOWS, JUNTO CON LA DLL ObtenIpPuerto.dll
*/

```
#include "headers.h"
#include <windows.h>
DWORD       *pdwGlobalTraceVariable,
DWORD       *pdwGlobalVistaVariable,
HANDLE      g_hOdbcDM,
CRITICAL_SECTION        g_csWrite,
char    *AddressDest,
int Port,
/* libmain() - Std DLL entry point, called from libentry.asm */
BOOL  WINAPI DllMain(HANDLE hInst,
                                                DWORD ul_reason_being_called,
                                                LPVOID lpReserved)
{
        HINSTANCE hinst = LoadLibrary("ObtenIpPuerto.dll");
  int (*Porto)(void) = (int (*)(void)) GetProcAddress(hinst,"ObtenPuerto");
  char (*AddressD)(int) = (char (*)(int)) GetProcAddress(hinst,"ObtenIp");
  char AddressDest1[16];
  int i;
  switch (ul_reason_being_called)
        {
                case DLL_PROCESS_ATTACH:

   Port = Porto();
        for(i=0;i<15;i++)
        {
          AddressDest1[i] = AddressD(i);
        }
  //AddressDest = (char *) AddressDest1;
   FreeLibrary(hinst);
   AddressDest = (char *) malloc(15);
   AddressDest[15]=0;
   for(i=0;i<15;i++)
    AddressDest[i]=AddressDest1[i];
                        // We do not need the thread attach detach calls  // Who said that?? Daniel
                        DisableThreadLibraryCalls(hInst);
                        g_hOdbcDM = LoadLibraryA("odbc32.dll");
                        if (g_hOdbcDM)
                        {
                                pdwGlobalTraceVariable = (DWORD *) GetProcAddress(g_hOdbcDM, "ODBCSharedTraceFlag");
                                pdwGlobalVistaVariable = (DWORD *) GetProcAddress(g_hOdbcDM, "ODBCSharedVSFlag");
                        }
                        InitializeCriticalSection(&g_csWrite);
                        break;
        case DLL_PROCESS_DETACH:
                        FreeLibrary(g_hOdbcDM);
                        break;

        /* no need for these cases
        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
        */
                default:
                        break;

        } /* switch */
  return 1;
  UNREFERENCED_PARAMETER(lpReserved);
}

// DOTRACE.OBJ                  The main code for handling a trace commands for the
                                                ODBC 3.51 driver manager.
/*
#include "headers.h"
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <share.h>
#include "sql.h"
#include "sqlext.h"
#include "odbcinst.h"
#include "tracecon.h"
#include "sqlucode.h"
#include <tchar.h>
#include <locale.h>
//para el socket
#include <winsock.h>
#include <sys\types.h>
#include <string.h>
#include <iostream.h>
#define     MAX_HANDLES                 200
```

```
static      LPTRACESTR          szHandleMap[MAX_HANDLES];
static      FILE                *fpTraceFile = NULL;
static const wchar_t szTraceFileKey[] = L"TraceFile";
extern CRITICAL_SECTION g_csWrite
//valores del socket
extern char    AddressDest;
extern int Port;
LPCWSTR  szDefaultFileName=                    L"\\SQL.LOG";
LPCWSTR  szODBC=                                                    L"ODBC";
LPCWSTR  szODBCIni=                                                 L"ODBC.INI";
DWORD    *gdwGlobalTraceVariable = global (really global) trace flag
char     *RetToChar(RETCODE   RetCode)
VOID GrabErrors( LPTRACESTR      TraceStr)
BOOL     g_fTrace = FALSE;
BOOL     g_fVS          = FALSE;
BOOL     g_fNuTrace = FALSE;
// Vista variables
DWORD    *gdwGlobalVistaVariable    = global (really global) vista flag

// Members

//          DoTrace()                          Called when entering a traced routine
//          TraceReturn()                      Called when exiting a traced routine
//          ProcessTrace()                     Prints a trace message to the file or stream

VOID        ProcessTrace(
            LPTRACESTR          TraceStr
            BOOL                fEntry,
            RETCODE             RetCode);

//función que manda la cadena de la sentencia SQL al servidor de escucha

VOID   SendSocket(
   LPSTR szBuffer
);

VOID        HandleArguments(
            LPTRACESTR                         TraceStr
                                                        ,pArg
            int
            BOOL                               fEntry,
            LPSTR                              szBuffer
            DWORD                              cbBuffer
            enum ArgTypes          atArgs

VOID DisplayString(
            LPSTR       szBuffer
            DWORD       cbBuffer,
            LPSTR       szData
            DWORD       cbData);

VOID DisplayStringPtr(
            LPSTR       szBuffer
            DWORD       cbBuffer,
            LPSTR       szData
            SDWORD      sdwLen,
            SDWORD      sdwMaxLen,
            BOOL        fUnicode,
            BOOL        fCopy);

VOID        HandleCharData(
            LPSTR                  szBuffer
            DWORD                  cbBuffer,
            LPTRACESTR             TraceStr
            DWORD                  pArg
            BOOL                   fEntry,
            BOOL                   fUnicode
            BOOL                   fCopy);

void HandleDataPtr(
            LPSTR                  szBuffer
            DWORD                  cbBuffer,
            SDWORD                 sdwAttribute
            const struct tagDFTARRAY *pDftList,
            DWORD                  cbDft);
void ExtractDescriptor(
            LPSTR                  szBuffer
            DWORD                  cbBuffer
            LPTRACESTR             TraceStr
            DWORD                  pArg
void GetPtrValue(
            LPSTR                              szBuffer
            DWORD                              cbBuffer
            SDWORD                             sdwAttribute
            LPVOID                             lpvData
            const struct tagDFTARRAY *pDftList
            DWORD                              cbDft,
            BOOL                               fEntry
            LPTRACESTR                         TraceStr
            DWORD                              iArg);

void HandlePtr(
            LPSTR                  szBuffer,
            DWORD                  cbBuffer,
```

```
        LPTRACESTR      TraceStr,
        DWORD           lArg,
        BOOL            fEntry);

enum ArgTypes CTypeToAType(
        SWORD   swCType);

enum ArgTypes GatorToAType(
        SDWORD  sdwFlag,
        BOOL    fUnicode);
//
// DoTrace      Process a trace request
//
// Parameters:
//              TraceStr.       Structure with information about the function
//                                              to trace

VOID    DoTrace(LPTRACESTR  TraceStr)
{

        ProcessTrace(TraceStr,TRUE,0);
        return;
}

//
// TraceReturn          Process a 'post' trace request
//
// Parameters:
//              TraceStr.       saved trace structure
//

VOID    SQL_API TraceReturn(RETCODE     RetHandle, RETCODE RetCode)
{
        LPTRACESTR          TraceStr = NULL;


        if (RetHandle > 0)
                TraceStr = szHandleMap[RetHandle - 1];


        if (TraceStr)
        {
                if (!(g_fNoTrace))
                {
                        ProcessTrace(TraceStr,FALSE, RetCode);
                }
                free(TraceStr);
                _heapmin();
                szHandleMap[RetHandle - 1] = NULL;
        }

        return;
}

// Moved out here because RISC doesn't like huge stacks
static char   szVistaArgs[8 * 1024];

//
// ProcessTrace.        Actually handle a trace message
//
// Parameters:
//              TraceStr        trace structure
//              fEntry                  TRUE if at function entry

VOID    ProcessTrace(
        LPTRACESTR          TraceStr,
        BOOL                fEntry,
        RETCODE             RetCode)
{
        DWORD   i;
        DWORD   dwVistaArgLen = 0;
        DWORD   dwStrLen;

        char    szBuffer[2000];
        char    AuxszBuffer[2000];
        char    *pdest = NULL;

        BOOL    cuatro=FALSE;

        BOOL    fTrace = (g_fTrace || pdwGlobalTraceVariable && *pdwGlobalTraceVariable) && TraceStr;


        if (g_fNoTrace)
        {
                return;
        }

        if (!fTrace)
        {
                return;
```

```
        }



        // tracing
        if (fTrace && !fhTraceFile)
        {
                (void) TraceOpenLogFile(NULL,NULL,0);
                if (!fhTraceFile)
                {
                        fTrace = FALSE;
                }
        }



        if (fTrace)
        {
                EnterCriticalSection(&g_csWrite);
        }

        for (i = 0; i < TraceStr->nArgs; i++)
        {
  if (!fEntry)  //Si es una función de salida, o sea que ya se ejecuto
          {

                        if ((RetCode == SQL_SUCCESS_WITH_INFO) || (RetCode == SQL_SUCCESS))  // si se realizó correctamente la ejecución de la
sentencia
                                {

                                szBuffer[2000]=0;


                                                // Si es cualquiera de estas funciones "SQLPrepare", "SQLExecDirect", el argumento es una cadena "TraceStr-
>atArg[i]==TYP_UCHARPTR)"
                                                // Y no se trata de ninguna cadena vacía "  (UCHAR *) TraceStr->lpvArg[i] != "" "

        if((  TraceStr->szFuncName=="SQLPrepare" || TraceStr->szFuncName=="SQLExecDirect") && ((UCHAR *) TraceStr->lpvArg[i] != "" ) &&(TraceStr-
>atArg[i]==TYP_UCHARPTR))


                                        {

                                                        _snprintf(AuxszBuffer,sizeof(AuxszBuffer),"%s",(UCHAR *) TraceStr->lpvArg[i]);
                                                        _snprintf(AuxszBuffer,sizeof(AuxszBuffer),"%s",strupr(AuxszBuffer));
                                                                pdest = strstr(AuxszBuffer,"SELECT");


                                                // si es cualquier tipo de sentencia excepto de SELECT

                                                                if (pdest == NULL)
                                                                  {


                                                                                //if(i==1)
                                                                                //{
                                                                                        _snprintf(szBuffer,sizeof(szBuffer),"%s
\r\n",(UCHAR *) TraceStr->szFuncName);

                                                                                        dwStrLen = strlen(szBuffer);
                                                                                        SendSocket(szBuffer);
                                                                                //}

                                                                // si la sentencia no es SELECT (puede ser INSERT, UPDATE ó DELETE),
                                                                //              copiala         a un Buffer, escribela en el Archivo de Seguimiento y
                                                                //              Mandala mediante un socket al Servidor que este Replicando
                                                                // [LoyalServer] en algun lugar de la RED por la función SendSocket

                                                                  _snprintf(szBuffer,sizeof(szBuffer),"%s \r\n",(UCHAR *) TraceStr->lpvArg[i]);
                                                                  dwStrLen = strlen(szBuffer);
                                                fwrite(szBuffer,dwStrLen, 1,fnTraceFile),
                                                                  SendSocket(szBuffer);
                                                                  }


                                        }


                                                if(  TraceStr->szFuncName=="SQLExecute"          )
                                                {
                                                                        _snprintf(szBuffer,sizeof(szBuffer),"%s
\r\n",(UCHAR *) TraceStr->szFuncName);

                                                                        dwStrLen = strlen(szBuffer);
```

```
                                                                    SendSocket(szBuffer);
                                                                }

                                    }
                                }
                        }

                                    fflush(fhTraceFile);
                                    LeaveCriticalSection(&g_csWrite);

}

// Función de envio de datos

VOID   SendSocket(LPSTR szBuffer)
{

  struct sockaddr_in    adar;       // dirección
         int            lenaddr,    // longitud de la dirección
         int              sock,     // socket
         int                                                    retval;
  WSADATA               wsaData,
         WORD sockVersion = MAKEWORD(1,1),
         int    res=1,
         DWORD      dwStrLen,
         dwStrLen= strlen(szBuffer),


         res = WSAStartup(sockVersion, &wsaData),


         if (res == 0)
         {

           if((sock = socket( AF_INET, SOCK_STREAM, 0    ==-1) //creamos el socket
           printf ("Error en socket"),
           memset (&addr, 0, sizeof(addr)),
           addr.sin_family = AF_INET;
           addr.sin_addr.s_addr = inet_addr(AddressDest       especificamos dirección IP destino  Dirección del servidor:
                    addr.sin_port = htons((u_short)Port);                                  //especificamos el puerto que esta escuchando en el  servidor
                    lenaddr=sizeof(addr);
                    retval=connect(sock,(struct sockaddr * )&addr,lenaddr),

                    if (retval)
                    printf("connect ")

                    send(sock, szBuffer,dwStrLen, 0),       --andamos el buffer al servidor


         }

}


,, TraceOpenLogFile    Open the log file for a trace
//
// Parameters:
//                      lpwszFileName:              * e name to open or NULL (UNICODE'
//                      pwszOutputMsg:              space for output error message
//                      cbOutputMsg:                ax size of output message
```

```
// Returns:
//                      File handle or NULL
//
// Notes:  If NULL passed in for the file name, looks for the
//                      trace file name in the registry.  If none there, it uses
//                      \SQL.LOG

RETCODE SQL_API TraceOpenLogFile(
        LPWSTR    szFileName,
        LPWSTR    lpwszOutputMsg,
        DWORD     cbOutputMsg)
{
        WCHAR           szTraceFile[_MAX_PATH + 1];
        char            aszTraceFile[_MAX_PATH];

        if (fhTraceFile)
                return SQL_SUCCESS;

        // Get default entry if no filename specified

        if (!szFileName)
        {
                SQLGetPrivateProfileStringW(szODBC,
                                                                        szTraceFileKey,
                                                                        szDefaultFileName,
                                                                        szTraceFile,
                                                                        sizeof(szTraceFile) / sizeof(WCHAR),
                                                                        szODBCIni);

                szFileName = szTraceFile;
        }

        fhTraceFile = _wfsopen(szFileName,L"abc",SH_DENYNO);

        // Maybe we are on the Win95 platform

        if (fhTraceFile == NULL)
        {
                WideCharToMultiByte(CP_ACP,             // XXX Consider: make any cp?
                                                                0,
                                                                szFileName,
                                                                wcslen(szFileName) + 1,
                                                                aszTraceFile,
                                                                sizeof(aszTraceFile),
                                                                NULL,
                                                                FALSE);

                fhTraceFile = _fsopen(aszTraceFile,"abc",SH_DENYNO);

        }

        if (fhTraceFile == NULL)
        {
                char            szBuffer[1000], szFull[_MAX_PATH];

                _fullpath(szFull,aszTraceFile,sizeof(szFull));

                sprintf(szBuffer,
                                        "Unable to open ODBC trace file %s: %s\n",
                                        szFull,
                                        strerror(errno));
                OutputDebugString(szBuffer);

                szBuffer[strlen(szBuffer) - 1] = '\0';

                if (lpwszOutputMsg)
                {
                        MultiByteToWideChar(CP_ACP,
                                                                0,
                                                                szBuffer,
                                                                strlen(szBuffer) + 1,
                                                                lpwszOutputMsg,
                                                                cbOutputMsg / sizeof(WCHAR));
                }
                return SQL_ERROR;
        }

        return SQL_SUCCESS;

}

//
// TraceCloseLogFile:           Close the log file
//
// Parameters: none
//

RETCODE  SQL_API TraceCloseLogFile()
{
        if (fhTraceFile)
```

```
                fclose(fhTraceFile);

        return SQL_SUCCESS;
}

//
// RetToChar: convert a retcode to a character representation
//
// Parameters:
//              RetCode             Retcode value
//
// Returns:
//              String mapping to that value

char    *RetToChar(RETCODE  RetCode)
{
        switch (RetCode)
        {
                case    SQL_SUCCESS:                    return("SQL_SUCCESS");
                case    SQL_ERROR                               return("SQL_ERROR");
                case    SQL_NO_DATA_FOUND return("SQL_NO_DATA_FOUND");
                case    SQL_NEED_DATA                   return("SQL_NEED_DATA");
                case    SQL_INVALID_HANDLE  return("SQL_INVALID_HANDLE");
                case    SQL_SUCCESS_WITH_INFO           return("SQL_SUCCESS_WITH_INFO");
                case    SQL_STILL_EXECUTING             return("SQL_STILL_EXECUTING");
                default
                                        return("UNKNOWN SQL RETURN CODE");
        }
}

//
// GetNextHandle:  get a free trace handle
//


LPTRACESTR      szHandleMap[MAX_HANDLES];

RETCODE  SetNextHandle(LPTRACESTR       lpData)
{
        int     i = 0;

        EnterCriticalSection(&g_csWrite);

        while (i < MAX_HANDLES)
        {
                if (szHandleMap[i] == NULL)
                {
                        szHandleMap[i] = lpData;
                        LeaveCriticalSection(&g_csWrite);
                        return i + 1;
                }
                i++;
        }

        LeaveCriticalSection(&g_csWrite);

        return  0;
}


//
// HandleArgument:      Display an argument
//
// Parameters:
//              TraceStr    trace structure
//              iArg                Argument to handle
//              szBuffer    Return buffer
//              cbBuffer    Max buffer size
//              fEntry              entry to routine (otherwise exit)
//

VOID    HandleArgument(
        LPTRACESTR                      TraceStr,
        int                             i,
        BOOL                            fEntry,
        LPSTR                           szBuffer,
        DWORD                           cbBuffer,
        enum ArgTypes           atArg)
{
        BOOL                    fCCH = FALSE;           // is unicode CCH argument

        switch(TraceStr->dwCallFunc)
        {
                case    SQL_API_SQLPREPARE:
                case    SQL_API_SQLGETDESCREC:
                case    SQL_API_SQLEXECDIRECT:
                case    SQL_API_SQLDESCRIBECOL:
                case    SQL_API_SQLDRIVERCONNECT:
                case    SQL_API_SQLCONNECT:
                case    SQL_API_SQLSETDESCREC:
                case    SQL_API_SQLGETDIAGREC:
                        fCCH = TRUE;
        }
```

```
switch(atArg)
{
        case    TYP_HENV:
        case    TYP_HDBC:
        case    TYP_HSTMT:
        case    TYP_HWND:
        case    TYP_SQLHANDLE:
        case    TYP_SQLHDESC:
        case    TYP_SQLHDBC:
        case    TYP_SQLHENV:
        case    TYP_SQLHLOC:
        case    TYP_SQLHSTMT:
                _snprintf(szBuffer,cbBuffer," 0x%08x\r\n",TraceStr->lpvArg[i]);
                return;

        case    TYP_HENVPTR:
        case    TYP_HSTMTPTR:
        case    TYP_SQLHANDLEPTR:
        case    TYP_HDBCPTR:
        {
                CHAR    szTemp[100];

                if ((!fEntry) || !(TraceStr->lpvArg[i]))
                {
                        _snprintf(szBuffer,cbBuffer," 0x%08x\r\n",TraceStr->lpvArg[i]);
                        return;
                }

                if (!(fEntry) && TraceStr->lpvArg[i])
                {
                        if (((char *)TraceStr->lpvArg[i] < (char *) 0x1000) ||
                                IsBadWritePtr(TraceStr->lpvArg[i],sizeof(UDWORD)))
                                sprintf(szTemp,"BADMEM");
                        else
                                _snprintf(szTemp,cbBuffer," 0x%08x",*(DWORD *)(TraceStr->lpvArg[i]));
                }
                _snprintf(szBuffer,cbBuffer," 0x%08x (%s)\r\n",TraceStr->lpvArg[i],szTemp);
                return;

        }

        case    TYP_PTR:
        case    TYP_PTRPTR:
        case    TYP_SQLPOINTER:
                HandlePtr(szBuffer,cbBuffer,TraceStr,i,fEntry);
                return;

        case    TYP_UNKNOWN:
                _snprintf(szBuffer,cbBuffer," <unknown type>\r\n");
                break;

        case    TYP_SQLUSMALLINT:
        case    TYP_UWORD:
        {
                _snprintf(szBuffer,cbBuffer,"   %8hu ",(UWORD)(TraceStr->lpvArg[i]));
                ExtraInfo(szBuffer + strlen(szBuffer),
                                                        cbBuffer - strlen(szBuffer),
                                                        TraceStr,
                                                        i);
                return;
        }

        case    TYP_BOOL:
        case    TYP_SQLSMALLINT:
        case    TYP_SQLINTEGER:
        case    TYP_SWORD:
        {
                _snprintf(szBuffer,cbBuffer,"   %8hd ",(SWORD)(TraceStr->lpvArg[i]));
                ExtraInfo(szBuffer + strlen(szBuffer),
                                                        cbBuffer - strlen(szBuffer),
                                                        TraceStr,
                                                        i);
                return;
        }

        case    TYP_UDWORD:
        {
                _snprintf(szBuffer,cbBuffer," %8lu\r\n",(UDWORD)(TraceStr->lpvArg[i]));
                return;
        }

        case    TYP_SDWORD:
        {
                _snprintf(szBuffer,cbBuffer," %8ld\r\n",(SDWORD)(TraceStr->lpvArg[i]));
                return;
        }

        case    TYP_SQLINTEGERPTR:
        case    TYP_SDWORDPTR:
        {
                if (((fEntry) || (!(TraceStr->lpvArg[i]))
                {
                        _snprintf(szBuffer,cbBuffer," 0x%08x\r\n",TraceStr->lpvArg[i]);
```

```
                                        return;
                                }

                                if              (IsBadWritePtr(TraceStr->lpvArg[],sizeof(SDWORD)))
                                        _snprintf(szBuffer,cbBuffer," 0x%.2Ex (BADMEM)\r\n",TraceStr->lpvArg[i]);
                                else
                                        _snprintf(szBuffer,cbBuffer," 0x%.2Ex (%ld)\r\n",TraceStr->lpvArg[i], *(SDWORD *)(TraceStr-
>lpvArg[i]));
                                return;

                        }

                        case    TYP_SQLSMALLINTPTR
                        case    TYP_SWORDPTR
                        {
                                if ((fEntry) || (!(TraceStr->lpvArg[i])))
                                {
                                        _snprintf(szBuffer,cbBuffer," 0x%.2Ex\r\n",TraceStr->lpvArg[i]);
                                        return;
                                }

                                if (IsBadWritePtr(TraceStr->lpvArg[],sizeof(SWORD)))
                                        _snprintf(szBuffer,cbBuffer," 0x%.2Ex (BADMEM)\r\n",TraceStr->lpvArg[i]);
                                else
                                        _snprintf(szBuffer,cbBuffer," 0x%.2Ex (%d)\r\n",TraceStr->lpvArg[i], *(SWORD *)(TraceStr-
>lpvArg[i]));
                                return;
                        }

                        case    TYP_UDWORDPTR
                        {
                                if ((fEntry) || (!(TraceStr->lpvArg[i])))
                                {
                                        _snprintf(szBuffer,cbBuffer," 0x%.2Ex\r\n",TraceStr->lpvArg[i]);
                                        return;
                                }

                                if              (IsBadWritePtr(TraceStr->lpvArg[],sizeof(UDWORD)))
                                        _snprintf(szBuffer,cbBuffer," 0x%.2Ex (BADMEM)\r\n",TraceStr->lpvArg[i]);
                                else
                                        _snprintf(szBuffer,cbBuffer," 0x%.2Ex (%lu)\r\n",TraceStr->lpvArg[i], *(SDWORD *)(TraceStr-
>lpvArg[i]));
                                return;

                        }

                        case    TYP_UWORDPTR
                        {
                                if ((fEntry) || (!(TraceStr->lpvArg[i])))
                                {
                                        _snprintf(szBuffer,cbBuffer," 0x%.2Ex\r\n",TraceStr->lpvArg[i]);
                                        return;
                                }

                                if              (IsBadWritePtr(TraceStr->lpvArg[],sizeof(UWORD)))
                                        _snprintf(szBuffer,cbBuffer," 0x%.2Ex (BADMEM)\r\n",TraceStr->lpvArg[i]);
                                else
                                        _snprintf(szBuffer,cbBuffer," 0x%.2Ex (%u)\r\n",TraceStr->lpvArg[i], *(SWORD *)(TraceStr-
>lpvArg[i]));
                                return;
                        }

                        case    TYP_UCHARPTR
                        case    TYP_SQLCHARPTR
                                HandleCharData(szBuffer,cbBuffer,TraceStr,fEntry,FALSE,fCCH);
                                break;
                        case    TYP_WCHARPTR
                        case    TYP_SQLWCHARPTR
                                HandleCharData(szBuffer,cbBuffer,TraceStr,fEntry,TRUE,fCCH);

                                break;

        }

        return;
}


/*
** HandleCharData:              Display a unicode character string
**
** Parameters:
**                      szBuffer        output buffer
**                      cbBuffer        length of output buffer
**                      TraceStr        general trace structure
**                      iItem                   item number
**                      fEntry                  TRUE if function entry
**                      fUnicode        TRUE if unicode
**                      fCCH                    TRUE if unicode con
*/
VOID    HandleCharData(
        LPSTR                   szBuffer,
        SWORD                   cbBuffer,
        LPTRACESTR              TraceStr,
```

```
DWORD               i,
BOOL                fEntry,
BOOL                fUnicode
BOOL                fCCH)
{

    BOOL    fIsOutput = FALSE, fIsStringFunc = FALSE;
    SDWORD  sdwLen = SQL_NTS;
    SDWORD  sdwMaxLen = SQL_NTS;
    SDWORD  iLen;


    _snprintf(szBuffer,cbBuffer," 0x%08x ",TraceStr->lpvArg[i]);
    iLen = strlen(szBuffer);
    szBuffer += iLen;
    cbBuffer -= iLen;

    // Determine length

    if (i + 1 < TraceStr->nArgs)
    {
        switch(TraceStr->atArg[i + 1])
        {
            case    TYP_SWORD:
                    sdwLen = (SDWORD) ((SWORD)(TraceStr->lpvArg[i + 1]));
                    sdwMaxLen = sdwLen;
                    fIsStringFunc = TRUE;
                    break;

            case    TYP_SDWORD:
                    sdwLen = (SDWORD)(TraceStr->lpvArg[i + 1]);
                    sdwMaxLen = sdwLen;
                    fIsStringFunc = TRUE;
                    break;
        }

        // Determine if output parameter
        // If so, current "length" is maximum length

        if ((i + 2 < TraceStr->nArgs) && fIsStringFunc)
        {
            switch(TraceStr->atArg[i + 2])
            {
                case    TYP_SWORDPTR:
                {
                    sdwMaxLen = sdwLen;
                    sdwLen = -9999;
                    fIsOutput = TRUE;

                    if (!(TraceStr->lpvArg[i + 2]))
                            break;

                    if (!(IsBadWritePtr(TraceStr->lpvArg[i + 2], sizeof(SWORD))))
                            sdwLen = (SDWORD) *((SWORD *)(TraceStr->lpvArg[i + 2]));
                    break;
                }

                case    TYP_SQLINTEGERPTR:
                case    TYP_SDWORDPTR:
                {
                    sdwMaxLen = sdwLen;
                    sdwLen = -9999;

                    fIsOutput = TRUE;

                    if (!(TraceStr->lpvArg[i + 2]))
                            break;
                    if (!(IsBadWritePtr(TraceStr->lpvArg[i + 2], sizeof(SDWORD
                            sdwLen = (SDWORD) *((SDWORD *)(TraceStr->lpvArg[i + 2]
                    break;
                }
            }
        }

    }


    if (!(fIsStringFunc))
    {
        _snprintf(szBuffer,cbBuffer," NYI\n");
        return;
    }

    _snprintf(szBuffer,cbBuffer," 0x%08x ",TraceStr->lpvArg[i]);

    // Display input values
    if ((!(fIsOutput) || !(fEntry)) && ((sdwLen > 0) || (sdwLen == SQL_NTS)) && ((sdwMaxLen > 0) || (sdwMaxLen == SQL_NTS)))
    {
        _snprintf(szBuffer,cbBuffer,"[%-8ld]",min(sdwMaxLen,sdwLen));

        DisplayStringPtr(szBuffer + strlen(szBuffer),
                                                    cbBuffer - strlen(szBuffer),
```

```
                                                                    TraceStr->lpvArg[i],
                                                                    sdwLen,
                                                                    sdwMaxLen,
                                                                    fUnicode,
                                                                    fCCH);
        } else
        {
                _snprintf(szBuffer,cbBuffer,"\r\n");

        }

        return;
}

//
// DisplayStringPtr: Display unicode string (in ansi)
//
// Parameters:
//              szBuffer:       output buffer
//              cbBuffer:       length
//              szData                  data string
//              sdwLen:                 length
//              sdwMaxLen:      max length
//              fUnicode        is string unicode?
//              fCCH                    is this a unicode CCH?

VOID DisplayStringPtr(
        LPSTR   szBuffer,
        DWORD   cbBuffer,
        LPSTR   szData,
        SDWORD  sdwLen,
        SDWORD  sdwMaxLen,
        BOOL    fUnicode,
        BOOL    fCCH)
{
        LPSTR   szAnsiData = NULL;

        setlocale(LC_ALL,".ACP");

        if (!(szData))
        {
                _snprintf(szBuffer,cbBuffer,"<empty string>\r\n");
                return;

        }
        if (!(sdwLen))
        {
                _snprintf(szBuffer,cbBuffer,"<zero length>\r\n");
                return;
        }



        if (sdwLen == SQL_NTS)
        {
                sdwLen = 0;
                // Carefully figure out the length
                if (!(fUnicode))
                {
                        while (!(IsBadReadPtr(szData + sdwLen,1)) && *(szData + sdwLen))
                        {
                                sdwLen++;
                        }
                        if (IsBadReadPtr(szData + sdwLen,1))
                        {
                                _snprintf(szBuffer,cbBuffer,"<Bad null-terminated string!>\r\n");
                                return;
                        }
                        sdwLen++;                       // include zero byte
                }
                else
                {
                        // sdwLen needs to be translated into a wide character offset since
                        // *szData is a CHAR -- MDW.  This was a GPF risk
                        while (!(IsBadReadPtr(szData + sdwLen,sizeof(WCHAR)))
                                        && *((WCHAR *)(szData + sdwLen)))
                        {
                                sdwLen += sizeof(WCHAR);
                        }
                        if (IsBadReadPtr(szData + sdwLen,sizeof(WCHAR)))
                        {
                                _snprintf(szBuffer,cbBuffer,"<Bad null-terminated string!>\r\n");
                                return;
                        }
                        sdwLen += sizeof(WCHAR);                        // include zero byte
                        sdwLen /= sizeof(WCHAR);                        // Now Unicode is CCH rather than CB...
                }

                sdwMaxLen = sdwLen;

        }
```

```
        if ((sdwLen < 0) || (sdwMaxLen < 0))
        {
                _snprintf(szBuffer,cbBuffer,"<Invalid string length!>\r\n");
                return;
        }

        if (IsBadReadPtr(szData,min(sdwMaxLen,sdwLen)))
        {
                _snprintf(szBuffer,
                                                cbBuffer,
                                                "<Buffer at 0x%08x isn't %d bytes long>\r\n",
                                                szData,
                                                min(sdwLen,sdwMaxLen));
                return;
        }

        // Display unicode in ANSI format
        if (fUnicode)
        {
                SDWORD    sdwDiv = fCCH ? 1 : sizeof(WCHAR);

                szAnsiData = malloc(min(sdwLen, sdwMaxLen) + 1);
                if (!(szAnsiData))
                {
                        _snprintf(szBuffer,cbBuffer "Out of memory! \r\n");
                        return;
                }

                WideCharToMultiByte(CP_ACP,                     // XXX Consider: make any cp?
                                                                                0,

                                                                                (LPWSTR)(szData),
                                                                                (min(sdwMaxLen,sdwLen) /
sdwDiv),
                                                                                szAnsiData,
                                                                                min(sdwMaxLen,sdwLen),
                                                                                NULL,
                                                                                FALSE);

                DisplayString(szBuffer, cbBuffer, szAnsiData,
                        min(sdwMaxLen/sdwDiv, sdwLen/sdwDiv));
        } else
        {
                DisplayString(szBuffer,cbBuffer,szData,min(sdwMaxLen,sdwLen));
        }

        if (szAnsiData)
                free(szAnsiData);

        return;
}

//
// DisplayString:  display a text string
//
// Parameters:
//
//      szBuffer:   display buffer
//      cbBuffer:   size of buffer
//      szData:             ansi data to display
//      cbData              length of data

VOID DisplayString(
        LPSTR     szBuffer,
        DWORD     cbBuffer,
        LPSTR     szData,
        DWORD     cbData)
{
        BOOL                fLastWasLeadByte = FALSE;

        if (IsBadReadPtr(szData,cbData))
        {
                _snprintf(szBuffer cbBuffer,"...<BAD MEMORY after 0x%08x (len=%ld)!>\r\n",szData, cbData);
                return;
        }


        if (cbBuffer)
        {
                *szBuffer++ = '\';
                cbBuffer--;
        }

        while ((cbBuffer > 5) && (cbData > 0))
        {
                if (isprint(*szData) || fLastWasLeadByte || isleadbyte(*szData))
                {
                        *szBuffer++ = *szData++;
                        cbData--;
```

```
                        cbBuffer--;
                        if (fLastWasLeadByte)
                        {
                                fLastWasLeadByte = FALSE
                        } else if (isleadbyte(*szData))
                        {
                                fLastWasLeadByte = TRUE;
                        }
                } else
                {
                        sprintf(szBuffer, "\\%2x",*szData++);
                        cbData--;
                        cbBuffer -= 3;
                        szBuffer += 3;
                }
        }

        if (cbBuffer)
        {
                if (cbData)
                        _snprintf(szBuffer,cbBuffer,"...\"\r\n");
                else
                        _snprintf(szBuffer,cbBuffer,"\"\r\n");

        }

        return;
}

//
// ExtraInfo:  Decode argument values for certain functions
//
// Parameters:
//      szBuffer            output buffer
//      cbBuffer            length of output buffer
//      TraceStr            trace structure
//      iArg                argument we are on

void ExtraInfo(
        LPSTR           szBuffer,
        DWORD           cbBuffer,
        LPTRACESTR      TraceStr,
        DWORD           iArg)
{

        switch(TraceStr->dwCallFunc)
        {
                case    SQL_API_SQLGETDATA
                case    SQL_API_SQLBINDCOL:
                        if (iArg == 2)
                        {
                                GetDefaultId(szBuffer,
                                                                cbBuffer,
                                                                (SWORD) TraceStr->lpvArg[iArg],
                                                                lpdfCTypes,
                                                                NumItems(lpdfCTypes));

                                return;
                        }
                        break;

                case    SQL_API_SQLBINDPARAM:
                        if (iArg == 2)
                        {
                                GetDefaultId(szBuffer,
                                                                cbBuffer,
                                                                (SWORD) TraceStr->lpvArg[iArg],
                                                                lpdfCTypes,
                                                                NumItems(lpdfCTypes));

                                return;
                        }

                        if (iArg == 3)
                        {
                                GetDefaultId(szBuffer,
                                                                cbBuffer,
                                                                (SWORD) TraceStr->lpvArg[iArg],
                                                                lpdfSqlTypes,
                                                                NumItems(lpdfSqlTypes));

                                return;
                        }

                case    SQL_API_SQLBINDPARAMETER:
                        switch (iArg)
                        {
                                case    2:
                                        GetDefaultId(szBuffer,
                                                                cbBuffer,
                                                                (SWORD) TraceStr->lpvArg[iArg],
                                                                lpdfParamType,
                                                                NumItems(lpdfParamTypes));

                                        return;

                                case    3:
```

```
                        GetDefaultId(szBuffer,
                                                        cbBuffer,
                                                        (SWORD) TraceStr->lpvArg[iArg],
                                                        lpdfCTypes,
                                                        NumItems(lpdfCTypes)),
                        return;
                case    4:
                        GetDefaultId(szBuffer,
                                                        cbBuffer,
                                                        (SWORD) TraceStr->lpvArg[iArg],
                                                        lpdfSqlTypes,
                                                        NumItems(lpdfSqlTypes)),
                        return;
        }
        break;

case    SQL_API_SQLCOLATTRIBUTE
        if (iArg == 2)
        {
                // SQLColAttributes and SQLColAttribute have the same
                // function number
                BOOL fOld = !_tcsicmp(TraceStr->szFuncName, "SQLColAttributes");

                GetDefaultId szBuffer,
                        cbBuffer
                        (SWORD) TraceStr->lpvArg[iArg],
                        fOld ? lpdfDescType : lpdfDescType30,
                        fOld ? NumItems(lpdfDescType) : NumItems (lpdfDescType30));
                return;

        }
        break;


case    SQL_API_SQLGETENVATTR
case    SQL_API_SQLSETENVATTR

        if (iArg == 1)
        {
                GetDefaultId szBuffer
                                                                cbBuffer,
                                                                (UWORD) TraceStr->lpvArg[iArg],
                                                                lpdfSetEnvAttr,
                                                                NumItems(lpdfSetEnvAttr)),
                return;
        }

        break;

case    SQL_API_SQLDATASOURCES

        if (iArg == 1)
        {
                GetDefaultId szBuffer,
                                                                cbBuffer,
                                                                (UWORD) TraceStr->lpvArg[iArg],
                                                                lpdfDataSrcs,
                                                                NumItems(lpdfDataSrcs)),
                return;
        }

        break;

case    SQL_API_SQLDRIVERCONNECT

        if (iArg == 7)
        {
                GetDefaultId szBuffer,
                                                        cbBuffer,
                                                        (UWORD) TraceStr-> pvArg[iArg],
                                                        lpdfDrvConn,
                                                        NumItems(lpdfDrvConn)),
                return;
        }

        break;

case    SQL_API_SQLDRIVERS

        if (iArg == 1)
        {
                GetDefaultId szBuffer,
                                                        cbBuffer,
                                                        (UWORD) TraceStr->pvArg[iArg],
                                                        lpdfDataSrcs,
                                                        NumItems(lpdfDataSrcs)),
                return;
        }

        break;

#ifdef SQL_API_SQLGETLENGTH
case    SQL_API_SQLGETLENGTH
```

```
        case    SQL_API_SQLGETPOSITION:
                if (iArg == 1)
                {
                        GetDefaultId(szBuffer,
                                                        cbBuffer,
                                                        (UWORD) TraceStr->lpvArg[iArg],
                                                        lpdLocfLocatorType,
                                                        NumItems(lpdLocfLocatorType));
                        return;
                }
                break;
#endif

        case    SQL_API_SQLFREEHANDLE:
        case    SQL_API_SQLALLOCHANDLE:
        case    SQL_API_SQLENDTRAN:
                if (iArg == 0)
                {
                        GetDefaultId(szBuffer,
                                                        cbBuffer,
                                                        (UWORD) TraceStr->lpvArg[iArg],
                                                        lpdSQLAllocHandle,
                                                        NumItems(lpdSQLAllocHandle));
                        return;
                }
                break;

        case    SQL_API_SQLEXTENDEDFETCH:
        case    SQL_API_SQLFETCHSCROLL:
                if (iArg == 1)
                {
                        GetDefaultId(szBuffer,
                                                        cbBuffer,
                                                        (UWORD) TraceStr->lpvArg[iArg],
                                                        lpdFetchType,
                                                        NumItems(lpdFetchType));
                        return;
                }
                break;

        case    SQL_API_SQLFREESTMT:
                if (iArg == 1)
                {
                        GetDefaultId(szBuffer,
                                                        cbBuffer,
                                                        (UWORD) TraceStr->lpvArg[iArg],
                                                        lpdFreeStmtOpt,
                                                        NumItems(lpdFreeStmtOpt));
                        return;
                }
                break;

        case    SQL_API_SQLGETCONNECTATTR:
                if (iArg == 1)
                {
                        GetDefaultId(szBuffer,
                                                        cbBuffer,
                                                        (UWORD) TraceStr->lpvArg[iArg],
                                                        lpdGetConAtr,
                                                        NumItems(lpdGetConAtr));
                        return;
                }
                break;

        case    SQL_API_SQLGETFUNCTIONS:
                if (iArg == 2)
                {
                        GetDefaultId(szBuffer,
                                                        cbBuffer,
                                                        (UWORD) TraceStr->lpvArg[iArg],
                                                        lpdGetFunct,
                                                        NumItems(lpdGetFunct));
                        return;
                }
                break;

        case    SQL_API_SQLGETINFO:
                if (iArg == 1)
                {
                        GetDefaultId(szBuffer,
                                                        cbBuffer,
                                                        (UWORD) TraceStr->lpvArg[iArg],
                                                        lpdInfoType,
                                                        NumItems(lpdInfoType));
                        return;
                }
                break;

        case    SQL_API_SQLGETSTMTATTR:
                if (iArg == 1)
```

```
                {
                        GetDefaultId(szBuffer,
                                                                cbBuffer,
                                                                (UWORD) TraceStr->lpvArg[iArg],
                                                                lpdfGetStmtOpt,
                                                                NumItems(lpdfGetStmtOpt));
                        return;
                }
                break;

#ifdef SQL_API_SQLLOCATEDUPDATE

        case    SQL_API_SQLLOCATEDUPDATE:
        case    SQL_API_SQLGETSUBSTRING:
                if ((iArg == 1) || (iArg == 5))
                {
                        GetDefaultId(szBuffer,
                                                                cbBuffer,
                                                                (UWORD) TraceStr->lpvArg[iArg],
                                                                (iArg == 1)? lpdLocfLocatorType :
                                                                                lpdLocfSrcTargType,
                                                                (iArg == 1) ? NumItems(lpdLocfLocatorType) :
NumItems(lpdLocfSrcTargType));
                        return;
                }
                break;

#endif
        case    SQL_API_SQLGETTYPEINFO:
                if (iArg == 1)
                {
                        GetDefaultId(szBuffer,
                                                                cbBuffer,
                                                                (SWORD) TraceStr->lpvArg[iArg],
                                                                lpdfSqlTypes,
                                                                NumItems(lpdfSqlTypes));
                        return;
                }
                break;

#ifdef SQL_API_SQLLOCATOR

        case    SQL_API_SQLLOCATOR:
                if (iArg == 2)
                {
                        GetDefaultId(szBuffer,
                                                                cbBuffer,
                                                                (UWORD) TraceStr->lpvArg[iArg],
                                                                lpdLocfOperation,
                                                                NumItems(lpdLocfOperation));
                        return;
                }
                break;
#endif

        case    SQL_API_SQLSETCONNECTATTR:
                if (iArg == 1)
                {
                        GetDefaultId(szBuffer,
                                                                cbBuffer,
                                                                (SWORD) TraceStr->lpvArg[iArg],
                                                                lpdfSetConAtr,
                                                                NumItems(lpdfSetConAtr));
                        return;
                }
                break;
        case    SQL_API_SQLSETCONNECTOPTION:
                if (iArg == 1)
                {
                        GetDefaultId(szBuffer,
                                                                cbBuffer,
                                                                (SWORD) TraceStr->lpvArg[iArg],
                                                                lpdfSetConOpt,
                                                                NumItems(lpdfSetConOpt));
                        return;
                }
                break;

        case    SQL_API_SQLSETDESCFIELD:
                if (iArg == 2)
                {
                        GetDefaultId(szBuffer,
                                                                cbBuffer,
                                                                (SWORD) TraceStr->lpvArg[iArg],
                                                                lpdSetDescFields,
                                                                NumItems(lpdSetDescFields));
                        return;
                }
```

```
        break;

case    SQL_API_SQLSETDESCREC:
        if (iArg == 2)
        {
                GetDefaultId(szBuffer,
                                                        cbBuffer,
                                                        (SWORD) TraceStr->lpvArg[iArg],
                                                        lpdfCTypes,
                                                        NumItems(lpdfCTypes));
                return;
        }
        break;

case    SQL_API_SQLSETPOS:
        if (iArg == 2)
        {
                GetDefaultId(szBuffer,
                                                        cbBuffer,
                                                        (UWORD) TraceStr->lpvArg[iArg],
                                                        lpdfSetPos
                                                        NumItems(lpdfSetPos));
                return;
        }
        if (iArg == 3)
        {
                GetDefaultId(szBuffer,
                                                        cbBuffer,
                                                        (UWORD) TraceStr->lpvArg[iArg],
                                                        lpdfLock,
                                                        NumItems(lpdfLock));
                return;
        }
        break;

case    SQL_API_SQLSETSCROLLOPTIONS:
        if (iArg == 1)
        {
                GetDefaultId(szBuffer,
                                                        cbBuffer,
                                                        (UWORD) TraceStr->lpvArg[iArg],
                                                        lpdConcurrency,
                                                        NumItems(lpdConcurrency));
                return;
        }
        if (iArg == 2)
        {
                GetDefaultId(szBuffer,
                                                        cbBuffer,
                                                        (SDWORD) TraceStr->lpvArg[iArg],
                                                        lpdCursorType,
                                                        NumItems(lpdCursorType));
                return;
        }
        break;

case    SQL_API_SQLSETSTMTATTR:
        if (iArg == 1)
        {
                GetDefaultId(szBuffer,
                                                        cbBuffer,
                                                        (SDWORD) TraceStr->lpvArg[iArg],
                                                        lpdfSetStmtAtr,
                                                        NumItems(lpdfSetStmtAtr));
                return;
        }
        break;

case    SQL_API_SQLSETSTMTOPTION:
        if (iArg == 1)
        {
                GetDefaultId(szBuffer,
                                                        cbBuffer,
                                                        (SDWORD) TraceStr->lpvArg[iArg],
                                                        lpdfSetStmtOpt,
                                                        NumItems(lpdfSetStmtOpt));
                return;
        }
        break;

case    SQL_API_SQLSPECIALCOLUMNS:
        switch (iArg)
        {
                case    1:
                        GetDefaultId(szBuffer,
                                                        cbBuffer,
                                                        (UWORD) TraceStr->lpvArg[iArg],
                                                        lpdfColType,
                                                        NumItems(lpdfColType));
```

```
                                        return;
                                case    &
                                        GetDefaultId(szBuffer,
                                                                        cbBuffer,
                                                                        (UWORD) TraceStr->lpvArg[iArg],
                                                                        lpdfScope,
                                                                        NumItems(lpdfScope));
                                        return;
                                case    9
                                        GetDefaultId(szBuffer,
                                                                        cbBuffer,
                                                                        (UWORD) TraceStr->lpvArg[iArg],
                                                                        lpdfNullable,
                                                                        NumItems(lpdfNullable));
                                        return;
                        }
                        break;

                case    SQL_API_SQLSTATISTICS
                        switch (iArg)
                        {
                                case    -
                                        GetDefaultId(szBuffer,
                                                                        cbBuffer,
                                                                        (UWORD) TraceStr->lpvArg[iArg],
                                                                        lpdfUnique,
                                                                        NumItems(lpdfUnique));
                                        return;
                                case    8
                                        GetDefaultId(szBuffer,
                                                                        cbBuffer,
                                                                        (UWORD) TraceStr->lpvArg[iArg],
                                                                        lpdfAccuracy,
                                                                        NumItems(lpdfAccuracy));
                                        return;
                        }
                        break;

                case    SQL_API_SQLTRANSACT
                        if (iArg == 2)
                        {
                                GetDefaultId szBuffer
                                                                cbBuffer,
                                                                (UWORD) TraceStr->lpvArg[iArg],
                                                                lpdTransactType,
                                                                NumItems(lpdTransactType));
                                return;
                        }
                        break;


        }
        _snprintf(szBuffer,cbBuffer,"\n");

        return
}

//
// GetDefaultId         Get the default return value
//
// Parameters
//
//      szBuffer    buffer
//      cbBuffer    length of buffer
//      dfltList    list to search
//      cbDft               size of list
//
void GetDefaultId
        LPSTR                           szBuffer,
        DWORD                           cbBuffer,
        SDWORD                          sdwValue,
        const struct tagDFTARRAY *dfltList
        DWORD                           cbDft)

{
        DWORD                   iCount;

        for (iCount = 0; iCount < cbDft; iCount++)
        {
                if (dfltList[iCount].lVal == sdwValue)
                {
                        _snprintf(szBuffer,cbBuffer,"<%s>\n",dfltList[iCount].szOpt);
                        return;
                }
        }

        _snprintf(szBuffer,cbBuffer,"<unknown>\n");
        return;
}

//
// HandlePtr handle PTR data type
```

```
//
// Parameters:
//          szBuffer            output buffer
//          cbBuffer            length of output buffer
//          TraceStr            trace structure
//          iArg                argument we are on

void HandlePtr(
        LPSTR           szBuffer,
        DWORD           cbBuffer,
        LPTRACESTR      TraceStr,
        DWORD           iArg,
        BOOL            fEntry)
{

        switch(TraceStr->dwCallFunc)
        {
                case    SQL_API_SQLGETENVATTR:
                case    SQL_API_SQLSETENVATTR:

                        if (iArg == 2)
                        {
                                GetPtrValue(szBuffer,
                                                                        cbBuffer,
                                                                        (DWORD) TraceStr->lpvArg[1],
                                                                        TraceStr->lpvArg[iArg],
                                                                        lpdfSetEnvAttr,
                                                                        NumItems(lpdfSetEnvAttr),
                                                                        fEntry,
                                                                        TraceStr,
                                                                        iArg);
                                return;
                        }

                        break;

                case    SQL_API_SQLCOLATTRIBUTE:

                        if (iArg == 6)
                        {
                                GetPtrValue(szBuffer,
                                                                cbBuffer,
                                                                (DWORD) TraceStr->lpvArg[5],
                                                                TraceStr->lpvArg[iArg],
                                                                lpdfDescType,
                                                                NumItems(lpdfDescType),
                                                                fEntry,
                                                                TraceStr,
                                                                iArg);
                                return;
                        }

                        break;

                case    SQL_API_SQLGETCONNECTATTR:

                        if (iArg == 4)
                        {
                                GetPtrValue(szBuffer,
                                                                cbBuffer,
                                                                (DWORD) TraceStr->lpvArg[1],
                                                                TraceStr->lpvArg[iArg],
                                                                lpdfGetConAttr,
                                                                NumItems(lpdfGetConAttr),
                                                                fEntry,
                                                                TraceStr,
                                                                iArg);
                                return;
                        }
                        break;

                case    SQL_API_SQLGETDATA:
                        if (iArg == 3)
                        {
                                HandleArgument(TraceStr,
                                                                iArg,
                                                                fEntry,
                                                                szBuffer,
                                                                cbBuffer,
                                                                CTypeToAType((SWORD) TraceStr->lpvArg[2]));
                                return;
                        }
                        break;

                case    SQL_API_SQLGETINFO:

                        if (iArg == 2)
                        {
                                GetPtrValue(szBuffer,
                                                                cbBuffer,
                                                                (DWORD) TraceStr->lpvArg[1],
                                                                TraceStr->lpvArg[iArg],
                                                                lpdfInfoType,
```

```
                                                    NumItems(lpdfInfoType),
                                                    fEntry,
                                                    TraceStr,
                                                    iArg);
                    return;
            }
            break;

    case    SQL_API_SQLGETSTMTATTR:

            if (iArg == 2)
            {
                    GetPtrValue(szBuffer,
                                                    cbBuffer,
                                                    (UWORD) TraceStr->lpvArg[1],
                                                    TraceStr->lpvArg[iArg],
                                                    lpdfGetStmtOpt,
                                                    NumItems(lpdfGetStmtOpt),
                                                    fEntry,
                                                    TraceStr,
                                                    iArg),

                    return;
            }
            break;

    case    SQL_API_SQLSETCONNECTATTR:

            if (iArg == 2)
            {
                    GetPtrValue(szBuffer,
                                            cbBuffer,
                                            (SWORD) TraceStr->lpvArg[1],
                                            TraceStr->lpvArg[iArg],
                                            lpdfSetConAtr,
                                            NumItems(lpdfSetConAtr),
                                            fEntry,
                                            TraceStr,
                                            iArg);

                    return;
            }
            break;

    case    SQL_API_SQLSETCONNECTOPTION:

            if (iArg == 2)
            {
                    GetPtrValue(szBuffer,
                                            cbBuffer,
                                            (SWORD) TraceStr->lpvArg[1],
                                            TraceStr->lpvArg[iArg],
                                            lpdfSetConOpt,
                                            NumItems(lpdfSetConOpt),
                                            fEntry,
                                            TraceStr,
                                            iArg),

                    return;
            }
            break;

    case    SQL_API_SQLSETDESCFIELD:

            if (iArg == 3)
            {
                    GetPtrValue(szBuffer,
                                            cbBuffer,
                                            (SWORD) TraceStr->lpvArg[2]
                                            TraceStr->lpvArg[iArg],
                                            lpdSetDescFields,
                                            NumItems(lpdSetDescFields),
                                            fEntry,
                                            TraceStr,
                                            iArg),

                    return;
            }
            break;

    case    SQL_API_SQLSETSTMTATTR:

            if (iArg == 2)
            {
                    GetPtrValue(szBuffer,
                                            cbBuffer,
                                            (SWORD) TraceStr->lpvArg[1],
                                            TraceStr->lpvArg[iArg],
                                            lpdfSetStmtAtr,
                                            NumItems(lpdfSetStmtAtr),
                                            fEntry,
                                            TraceStr,
                                            iArg),

                    return;
            }
            break;
```

```
                case    SQL_API_SQLSETSTMTOPTION:

                        if (iArg == 2)
                        {
                                GetPtrValue(szBuffer,
                                                                        cbBuffer,
                                                                        (SWORD) TraceStr->lpvArg[1],
                                                                        TraceStr->lpvArg[iArg],
                                                                        fpdfSetStmtOpt,
                                                                        NumItems(lpdfSetStmtOpt),
                                                                        fEntry,
                                                                        TraceStr,
                                                                        iArg);
                                return;
                        }
                        break;


        }

        _snprintf(szBuffer,cbBuffer,"0x%08lx\n", TraceStr->lpvArg[iArg]);
        return;

}
//
// GetPtrValue:         Get the value of a pointer that varies according
                                to some argument (like SQLGetEnvAttr arg 3)
//
// Parameters:
//
//      szBuffer:               buffer
//      cbBuffer:               length of buffer
// sdwAttribute         attribute value
// lpvData              address of data
//      dfltList                list to search
//      cbDft                   size of list
//      fEntry                  is this the entry point or exit?

void GetPtrValue(
        LPSTR                                   szBuffer,
        DWORD                                   cbBuffer,
        SDWORD                                  sdwAttribute,
        LPVOID                                  lpvData,
        const struct tagDFTARRAY *dfltList,
        DWORD                                   cbDft,
        BOOL                                    fEntry,
        LPTRACESTR                              TraceStr,
        DWORD                                   iArg)

{
        DWORD                                   iCount;
        enum ArgTypes           AtThis;

        for (iCount = 0; iCount < cbDft; iCount++)
        {
                if (dfltList[iCount].fOpt == sdwAttribute)
                {
                        AtThis = GetorToAType(dfltList[iCount].uValInfo, TraceStr->dwFlags & TRACESTR_FLAG_UNICODE);
                        switch (AtThis)
                        {
                                case TYP_UDWORD:
                                        AtThis = TYP_UDWORDPTR;
                                        break;
                                case TYP_UWORD:
                                        AtThis = TYP_UWORDPTR;
                                        break;
                        }
                        HandleArgument( TraceStr,iArg,fEntry,szBuffer,cbBuffer,AtThis);
                        return;
                }
        }

        _snprintf(szBuffer,cbBuffer," [Unknown attribute field]=%ld",sdwAttribute);
        return;
}

//
// CTypeToAType:  convert a SQL_C_xxx type to an enum
//
// Parameters:
//      swCType
// returns: enum

enum ArgTypes CTypeToAType(
        SWORD   swCType)
{
        switch (swCType)
        {
                default:
                        return TYP_UNKNOWN;

                case SQL_C_USHORT:
                        return TYP_UWORD;
```

```
                case SQL_C_SLONG:
                        return TYP_SDWORD;

                case SQL_C_ULONG:
                        return TYP_UDWORD;

                case SQL_C_BIT:
                        return TYP_BOOL;

                case SQL_C_TINYINT:
                case SQL_C_SSHORT:
                        return TYP_SQLSMALLINT


                case SQL_C_CHAR:
                        return TYP_SQLCHARPTR


                case SQL_C_WCHAR:
                        return TYP_WCHARPTR
        }
}


//
// GatorToAtype: convert gators odd flags into a type enum
//
// Parameters:
//      swCType
// returns: enum
//
enum ArgTypes GatorToAType(
        SDWORD  sdwFlag,
        BOOL    fUnicode)
{

        if (sdwFlag & PRM_STR)
                return (fUnicode)? TYP_SQLWCHARPTR : TYP_SQLCHARPTR;


        if (sdwFlag & PRM_16BIT)
                return TYP_UWORD;

        if ((sdwFlag & PRM_32MSK) || (sdwFlag & PRM_32HEX))  // forces hex output
                return TYP_HENV;

        if (sdwFlag & PRM_32BIT)
                return TYP_UDWORD

        return TYP_HENV;
}

// Returns the trace API version

DWORD   SQL_API TraceVersion()
{
        return      TRACE_VERSION;
}


void GrabErrors( LPTRACESTR     TraceStr)
{
        LPVOID          lpvHandle = TraceStr->pArg[1];
        SQLSMALLINT     cRec = 0;
        RETCODE         RetCode;
        char            szMessage[2000], szState[1000];
        char            szBuffer[4000];
        SQLINTEGER      NativeError;
        SQLSMALLINT     nType;

        switch (TraceStr->dwCallFunc)
        {
                default:
                        return;

                case    SQL_API_SQLALLOCCONNECT
                case    SQL_API_SQLFREEENV
                case    SQL_API_SQLGETENVATTR
                case    SQL_API_SQLSETENVATTR
                        nType = SQL_HANDLE_ENV;
                        break;


                case    SQL_API_SQLALLOCSTMT
                case    SQL_API_SQLCONNECT
                case    SQL_API_SQLDATASOURCES
                case    SQL_API_SQLDISCONNECT
                case    SQL_API_SQLFREECONNECT
                case    SQL_API_SQLGETCONNECTATTR
                case    SQL_API_SQLGETCONNECTOPTION
```

```
        case    SQL_API_SQLSETCONNECTATTR :
        case    SQL_API_SQLSETCONNECTOPTION :
        case    SQL_API_SQLDRIVERCONNECT :
        case    SQL_API_SQLDRIVERS :
        case    SQL_API_SQLNATIVESQL:
        case    SQL_API_SQLGETINFO :
        case    SQL_API_SQLBROWSECONNECT :
                hType = SQL_HANDLE_DBC;
                break;

        case    SQL_API_SQLTRANSACT :
                if (TraceStr->lpvArg[0])
                {
                        hType = SQL_HANDLE_ENV;
                        lpvHandle = TraceStr->lpvArg[0];
                } else
                {
                        hType = SQL_HANDLE_DBC;
                        lpvHandle = TraceStr->lpvArg[1];
                }

                break;

        case    SQL_API_SQLGETDESCFIELD
        case    SQL_API_SQLGETDESCREC
        case    SQL_API_SQLCOPYDESC
        case    SQL_API_SQLSETDESCFIELD
        case    SQL_API_SQLSETDESCREC
                hType = SQL_HANDLE_DESC;
                break;

        case    SQL_API_SQLBINDCOL
        case    SQL_API_SQLBINDPARAM
        case    SQL_API_SQLCANCEL
        case    SQL_API_SQLCLOSECURSOR
        case    SQL_API_SQLCOLATTRIBUTE
        case    SQL_API_SQLCOLUMNS
        case    SQL_API_SQLDESCRIBECOL
        case    SQL_API_SQLENDTRAN
        case    SQL_API_SQLEXECDIRECT
        case    SQL_API_SQLEXECUTE
        case    SQL_API_SQLFETCH
        case    SQL_API_SQLFETCHSCROLL
        case    SQL_API_SQLFREESTMT
        case    SQL_API_SQLGETCURSORNAME
        case    SQL_API_SQLGETDATA
        case    SQL_API_SQLGETFUNCTIONS
        case    SQL_API_SQLGETSTMTATTR
        case    SQL_API_SQLGETSTMTOPTION
        case    SQL_API_SQLGETTYPEINFO
        case    SQL_API_SQLNUMRESULTCOLS
        case    SQL_API_SQLPARAMDATA
        case    SQL_API_SQLPREPARE
        case    SQL_API_SQLPUTDATA
        case    SQL_API_SQLROWCOUNT
        case    SQL_API_SQLSETCURSORNAME
        case    SQL_API_SQLSETPARAM
        case    SQL_API_SQLSETSTMTATTR
        case    SQL_API_SQLSETSTMTOPTION
        case    SQL_API_SQLSPECIALCOLUMNS
        case    SQL_API_SQLSTATISTICS
        case    SQL_API_SQLTABLES
        case    SQL_API_SQLBULKOPERATIONS
        case    SQL_API_SQLBINDPARAMETER
        case    SQL_API_SQLCOLUMNPRIVILEGES
        case    SQL_API_SQLDESCRIBEPARAM
        case    SQL_API_SQLEXTENDEDFETCH
        case    SQL_API_SQLFOREIGNKEYS
        case    SQL_API_SQLMORERESULTS
        case    SQL_API_SQLNUMPARAMS
        case    SQL_API_SQLPARAMOPTIONS
        case    SQL_API_SQLPRIMARYKEYS
        case    SQL_API_SQLPROCEDURECOLUMNS
        case    SQL_API_SQLPROCEDURES
        case    SQL_API_SQLSETPOS
        case    SQL_API_SQLSETSCROLLOPTIONS
        case    SQL_API_SQLTABLEPRIVILEGES
                hType = SQL_HANDLE_STMT;
                break;
}

do {
        g_fNoTrace = TRUE;
        RetCode = SQLGetDiagRec(hType,
                                                        lpvHandle,
                                                        ++cRec,
                                                        szState,
                                                        &NativeError,
                                                        szMessage,
                                                        (SQLSMALLINT) sizeof(szMessage),
                                                        (SQLSMALLINT *) NULL);

        g_fNoTrace = FALSE;
```

```
                    if (RetCode == SQL_SUCCESS)
                    {
                        sprintf(szBuffer,
                                "\r\n\\\DIAG [%s] %s (%d) \r\n",
                                szState,
                                szMessage,
                                NativeError);
                        fwrite(szBuffer, strlen(szBuffer),1,fhTraceFile);
                    }

            } while (RetCode == SQL_SUCCESS);

            return;
}


RETCODE SQL_API TraceVSControl(DWORD dwControl)
{
            g_fTrace = dwControl & TRACE_ON;
            g_fVS = dwControl & TRACE_VS_EVENT_ON;
            return SQL_SUCCESS;
}



#include "headers.h"

///// Trace function for SQLExecDirect /////

RETCODE SQL_API TraceSQLExecDirect (HSTMT arg0,
                    UCHAR * arg1,
                    SDWORD arg2)
{
            LPTRACESTR          lpCallStr = (LPTRACESTR) malloc(sizeof(TRACESTR));
            if (lpCallStr == NULL)
                    return 0;
            memset(lpCallStr,0,sizeof(TRACESTR));
            lpCallStr->dwCallFunc = SQL_API_SQLEXECDIRECT;
            lpCallStr->szFuncName="SQLExecDirect";
            lpCallStr->lpvArg[0] = (LPVOID) arg0;
            lpCallStr->szArg[0]="HSTMT";
            lpCallStr->atArg[0]=TYP_HSTMT;
            lpCallStr->lpvArg[1] = (LPVOID) arg1;
            lpCallStr->szArg[1]="UCHAR *";
            lpCallStr->atArg[1]=TYP_UCHARPTR;
            lpCallStr->lpvArg[2] = (LPVOID) arg2;
            lpCallStr->szArg[2]="SDWORD";
            lpCallStr->atArg[2]=TYP_SDWORD;
            lpCallStr->nArgs = 3;
            DoTrace(lpCallStr);
            return((RETCODE)SetNextHandle(lpCallStr));
}

/// Trace function for SQLExecute /////

RETCODE SQL_API TraceSQLExecute (HSTMT arg0)
{
            LPTRACESTR          lpCallStr = (LPTRACESTR) malloc(sizeof(TRACESTR));
            if (lpCallStr == NULL)
                    return 0;
            memset(lpCallStr,0,sizeof(TRACESTR));
            lpCallStr->dwCallFunc = SQL_API_SQLEXECUTE;
            lpCallStr->szFuncName="SQLExecute";
            lpCallStr->lpvArg[0] = (LPVOID) arg0;
            lpCallStr->szArg[0]="HSTMT";
            lpCallStr->atArg[0]=TYP_HSTMT;
            lpCallStr->nArgs = 1;
            DoTrace(lpCallStr);
            return((RETCODE)SetNextHandle(lpCallStr));
}

/// Trace function for SQLPrepare /////

RETCODE SQL_API TraceSQLPrepare (HSTMT arg0,
                    UCHAR * arg1,
                    SDWORD arg2)
{
            LPTRACESTR          lpCallStr = (LPTRACESTR) malloc(sizeof(TRACESTR));
            if (lpCallStr == NULL)
                    return 0;
            memset(lpCallStr,0,sizeof(TRACESTR));
            lpCallStr->dwCallFunc = SQL_API_SQLPREPARE;
            lpCallStr->szFuncName="SQLPrepare";
            lpCallStr->lpvArg[0] = (LPVOID) arg0;
            lpCallStr->szArg[0]="HSTMT";
            lpCallStr->atArg[0]=TYP_HSTMT;
            lpCallStr->lpvArg[1] = (LPVOID) arg1;
            lpCallStr->szArg[1]="UCHAR *";
            lpCallStr->atArg[1]=TYP_UCHARPTR;
            lpCallStr->lpvArg[2] = (LPVOID) arg2;
            lpCallStr->szArg[2]="SDWORD";
```

```
        lpCallStr->atArg[2]=TYP_SDWORD;
        lpCallStr->nArgs = 3;
        DoTrace(lpCallStr);
        return((RETCODE)SetNextHandle(lpCallStr));
}

///// Trace function for SQLBindParameter /////

RETCODE SQL_API TraceSQLBindParameter  (HSTMT arg0,
                        UWORD arg1,
                        SWORD arg2,
                        SWORD arg3,
                        SWORD arg4,
                        UDWORD arg5,
                        SWORD arg6,
                        PTR arg7,
                        SDWORD arg8,
                                UNALIGNED SDWORD * arg9)
{
        LPTRACESTR              lpCallStr = (LPTRACESTR) malloc(sizeof(TRACESTR));
        if (lpCallStr == NULL)
                return 0;
        memset(lpCallStr,0,sizeof(TRACESTR));
        lpCallStr->dwCallFunc = SQL_API_SQLBINDPARAMETER;
        lpCallStr->szFuncName="SQLBindParameter";
        lpCallStr->lpvArg[0] = (LPVOID) arg0;
        lpCallStr->szArg[0]="HSTMT";
        lpCallStr->atArg[0]=TYP_HSTMT;
        lpCallStr->lpvArg[1] = (LPVOID) arg1;
        lpCallStr->szArg[1]="UWORD";
        lpCallStr->atArg[1]=TYP_UWORD;
        lpCallStr->lpvArg[2] = (LPVOID) arg2;
        lpCallStr->szArg[2]="SWORD";
        lpCallStr->atArg[2]=TYP_SWORD;
        lpCallStr->lpvArg[3] = (LPVOID) arg3;
        lpCallStr->szArg[3]="SWORD";
        lpCallStr->atArg[3]=TYP_SWORD;
        lpCallStr->lpvArg[4] = (LPVOID) arg4;
        lpCallStr->szArg[4]="SWORD";
        lpCallStr->atArg[4]=TYP_SWORD;
        lpCallStr->lpvArg[5] = (LPVOID) arg5;
        lpCallStr->szArg[5]="UDWORD";
        lpCallStr->atArg[5]=TYP_UDWORD;
        lpCallStr->lpvArg[6] = (LPVOID) arg6;
        lpCallStr->szArg[6]="SWORD";
        lpCallStr->atArg[6]=TYP_SWORD;
        lpCallStr->lpvArg[7] = (LPVOID) arg7;
        lpCallStr->szArg[7]="PTR";
        lpCallStr->atArg[7]=TYP_PTR;
        lpCallStr->lpvArg[8] = (LPVOID) arg8;
        lpCallStr->szArg[8]="SDWORD";
        lpCallStr->atArg[8]=TYP_SDWORD;
        lpCallStr->lpvArg[9] = (LPVOID) arg9;
        lpCallStr->szArg[9]="SDWORD *";
        lpCallStr->atArg[9]=TYP_SDWORDPTR;
        lpCallStr->nArgs = 10;
        DoTrace(lpCallStr);
        return((RETCODE)SetNextHandle(lpCallStr));
}

//
// HEADERS.H        Pre-compiled header file for ODBCTRAC.C
//
// This is the driver manager's helper dll for implementing tracing of
// the ODBC API
//
// (C) 1991 - 1999 Microsoft Corporation  All rights reserved.
//

#define SQL_DESC_BIND_OFFSET                    9999                    // junk
#define SQL_DESC_UPDATEABLE SQL_DESC_UPDATABLE

#include <windows.h>
#include <sql.h>
#include <sqlext.h>
#include <malloc.h>
#include "trace.h"


#pragma hdrstop

//
// TRACE.H
//
// Definitions for the ODBC trace dll
//
// (C) 1991 - 1999 Microsoft Corporation  All rights reserved.
//

#ifndef __TRACEH
#define __TRACEH
```

```
#define MAX_ARGS     20

// Include here just so you can compile if you don't happen to
// have the ODBC 3.51 headers in your path
#ifndef TRACE_VS_EVENT_ON
#define TRACE_ON 1
#define TRACE_VS_EVENT_ON 2
#endif

// Argument type enum

enum        ArgTypes
{

            TYP_UNKNOWN,
            TYP_HENV,
            TYP_HDBCPTR,
            TYP_HENVPTR,
            TYP_HDBC,
            TYP_HSTMTPTR,
            TYP_HSTMT,
            TYP_UWORD,
            TYP_SWORD,
            TYP_PTR,
            TYP_SDWORD,
            TYP_SDWORDPTR,
            TYP_SWORDPTR,
            TYP_UCHARPTR,
            TYP_UDWORDPTR
            TYP_UDWORD,
            TYP_HWND
            TYP_UWORDPTR,
            TYP_PTRPTR,
            TYP_BOOL,
            TYP_SQLSMALLINT,
            TYP_SQLHANDLE,
            TYP_SQLHANDLEPTR,
            TYP_SQLHSTMT,
            TYP_SQLINTEGER,
            TYP_SQLPOINTER,
            TYP_SQLINTEGERPTR,
            TYP_SQLSMALLINTPTR,
            TYP_SQLHDESC,
            TYP_SQLHDBC,
            TYP_SQLCHARPTR,
            TYP_SQLHENV,
            TYP_SQLHLOC,
            TYP_SQLUSMALLINT,
            TYP_WCHARPTR,
            TYP_SQLWCHARPTR
};
#define TRACESTR_FLAG_UNICODE 0x00000001L
#define TRACESTR_FLAG_VISTA           0x00000002L
typedef struct tagTRACESTR {
            DWORD                     dwCallFunc;                   // ODBC ID of function
            DWORD                     nArgs;                                    // How many arguments
            LPVOID                    lpvArg[MAX_ARGS];   // Arguments
            enum ArgTypes       atArg[MAX_ARGS];    // Argument type
            LPCSTR                    szArg[MAX_ARGS];    // Printable name
            DWORD                     dwFlags;                      // flags
            LPCSTR                    szFuncName;                   // Printable function name
} *LPTRACESTR, TRACESTR;

VOID        DeTrace(LPTRACESTR);
RETCODE     SetNewHandle(LPTRACESTR     lpData)
#endif
#define     dCSEG(x)          static const x

#define PRM_16BIT                     0x00000001L
#define PRM_32BIT                     0x00000002L
#define PRM_32MSK                     0x00000004L
#define PRM_32HDL                     0x00000008L
#define PRM_STR                       0x00000010L
#define PRM_CONN_PREFIX                           0x00000020L
#define PRM_STMT_PREFIX                           0x00000040L
#define PRM_HWND                                  0x00000080L
#define PRM_UWORD_PTR                             0x00000100L
#define PRM_SDWORD_PTR                            0x00000200L
#define PRM_UDWORD_PTR                            0x00000400L
#define PRM_PTR                                             0x00000800L
#define PRM_HENV                                  0x00001000L
#define PRM_HDBC                                  0x00002000L
#define PRM_HSTMT                                 0x00004000L
#define PRM_HDESC                                 0x00008000L
#define     PRM_PTR_BIN                                     0x00010000L
#define PRM_PTR_32BIT                       0x00020000L
#define INTERVALVAL                         0x00040000L

#define     NULLPTR                                         9999
#define     VALIDPTR                              9997
#define     SAMEAS_STRLEN                         9996

#define     NumItems(x) (sizeof(x) / sizeof(x[0]))
```

```
//*-------------------------------------------------------------------
//| DFTARRAY.
//|         This structure describes a set of default constants.  Note that an
//|         fOpt may have another array associated with it.  For example,
//|         the SQL_FETCH_DIRECTION fOpt can be any one of the SQL_FD_xxx
//|         flags.  For this case, lpLink points to the linked array of options
//*-------------------------------------------------------------------
typedef struct tagDFTARRAY {
        SDWORD    fOpt,                          // The option ID, note that format is based on following
        SWORD     fCType,                        // The SQL_C_xxx type of fOpt
        LPCTSTR   szOpt,                         // The option name
        UDWORD    uVersion,                      // Major/minor version of this item
        UDWORD    uValInfo,                      // User-defined (eg. could give type for GetInfo)
        UINT      cLinks,                        // Number of items in the linked array
        struct tagDFTARRAY * lpLink,             // Array of linked defaults
        } DFTARRAY,
typedef DFTARRAY *lpDFTARRAY

#include "dlgdfts.h"        // from odbctest


; Nemesis.def
; Libreria de definicion de funciones
; de la DLL LOYAL que mandara a llamar
; el administrador de controladores ODBC
LIBRARY LOYAL
EXPORTS
TraceSQLExecDirect
TraceSQLPrepare
TraceSQLExecute
TraceSQLBindParameter
TraceReturn
TraceOpenLogFile
TraceCloseLogFile
TraceVersion
TraceVSControl
```

APÉNDICE C

```
/*
TESIS: PROTOTIPO DE REPLICACIÓN DE BASES DE DATOS MEDIANTE ODBC
AUTOR:                    DANIEL LEAL LARIS
FECHA DE CREACIÓN:  12/07/2002
NOMBRE:   ObtenIpPuerto.dll
 Este código contiene las funciones para leer el archivo
 de definición 'LOYAL.ini',  que debe estar ubicado en raíz,
 al ser llamada la función  obtiene carácter por carácter la
 dirección IP y regresa dicho carácter a quien
 haya llamado la función, lo mismo sucede con el puerto.
 UNA VEZ GENERADO ESTA DLL ES NECESARIO COPIARLA AL DIRECTORIO DEL SISTEMA
 DE WINDOWS, JUNTO CON LA DLL Loyal.dll
*/

// Dynamic Link Primary
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
#include <stdlib.h>

#include "ObtenIpPuerto.h"


//Obtiene caracter por caracter la Dirección IP del Servidor Contenida en
//el archivo de Definiciones LOYAL.ini

char ObtenIp(int i)
{


  char buffer[16];
  char auxbuffer[16];
  char *backbuffer;

  int fh2;
  int zz;

  fh2 = _open( "LOYAL.ini", _O_RDONLY );

  _lseek( fh2, 9L, SEEK_SET );
  _read( fh2, buffer, 16 );
  _close( fh2 );

              for (zz=0; zz<=16;zz++){
      auxbuffer[zz] = buffer[zz+1];
              }


   return(auxbuffer[i]);


}


int ObtenPuerto(void)
{


              int fh2;
  char buf[5];
  double xx;
  char *stopstring;
  int Puerto;


  fh2 = _open( "LOYAL.ini", _O_RDONLY );
  _lseek( fh2, 9L, SEEK_SET );
  _lseek( fh2, 36L, SEEK_SET );
  _read( fh2, buf, 5 );
  _close( fh2 );

              xx = strtod(buf,&stopstring);
              Puerto = (int) xx;

   return(Puerto);

}

// ObtenIpPuerto.h
```

APÉNDICE C

Código fuente de la DLL de obtención de Ip - Puerto

```
char ObtenIp(int i);
int  ObtenPuerto(void);
```

```
;archivo  def
LIBRARY ObtenIpPuerto
EXPORTS

    ObtenIp
    ObtenPuerto
```

Archivo de configuración Loyal.ini.

```
[Dir Ip]
132.147.160.181
[Puerto]
8181
```

APÉNDICE D

```
Option Explicit
Dim rs As Recordset

Dim inicia As Boolean

Dim fijo As Integer

Dim centesima As Integer


Dim R_TVuelo() As String
Dim R_Stit() As String
Dim R_TAvion() As String
Dim R_TALin() As String
Dim R_CAptoO() As String
Dim R_CAptoD() As String

Private Sub cmbDes_LostFocus()
 If Trim(cmbOrg.Text) = Trim(cmbDes.Text) Then
   MsgBox "Imposible Tener mismo origen que destino"
   cmbOrg.SetFocus

 End If
End Sub

Private Sub cmdAgregar_Click()

   cmbAVTipo.Text = ""
   cmbALna.Text = ""
   cmbStatus.Text = ""
   cmbTAvi.Text = ""
   cmbOrg.Text = ""
   cmbDes.Text = ""
   MaskEdBox1.Mask = "## ##"
   MaskEdBox2.Mask = "## ##"
   MaskEdBox1.Format = "hh:mm"
   MaskEdBox2.Format = "hh:mm"
   prepara

End Sub
Private Sub prepara()
   inicia = True
   cmbAVTipo.Locked = False
   cmbALna.Locked = False
   cmbStatus.Locked = False
   cmbTAvi.Locked = False
   cmbOrg.Locked = False
   cmbDes.Locked = False
   MaskEdBox1.Enabled = True
   MaskEdBox2.Enabled = True
   'MaskEdBox1.Text = ""
   'MaskEdBox2.Text = ""
   cmdEliminar.Enabled = False
   CmdEditar.Enabled = False
   cmdGuardar.Enabled = False
   cmdVC.Enabled = False
   cmbAVTipo.SetFocus
End Sub


Private Sub Command5_Click()
Unload Me
End Sub
Private Sub CmdEditar_Click()
cmdAgregar.Enabled = False
prepara
End Sub

Private Sub cmdEliminar_Click()
Dim q As String
Dim res As Integer
Dim fijo As Integer
Dim i As Integer
On Error GoTo oerr
If Len(Trim(MSFlexGrid1.Text)) <> 0 Then
MSFlexGrid1.Col = 0
q = " Delete from intinerarios where id_itinerario = " & MSFlexGrid1.Text & ""
res = MsgBox("Estas seguro de eliminar el registro ", vbYesNo + vbQuestion + vbDefaultButton2)
 If res = vbYes Then
   cn.Execute q, dbExecDirect
     fijo = MSFlexGrid1.Rows
If fijo > 1 Then
     For i = fijo - 1 To 1 Step -1
       MSFlexGrid1.RemoveItem i
     Next
 End If


   cargaItinerario
```

```
Else
  MsgBox "La operación ha sido cancelada"
 End If
End If
oerr

If Err.Number <> 0 Then
   MsgBox "Ha ocurrido un Error"
   Exit Sub
End If

End Sub

Private Sub cmdguardar_Click()
InsertarRegistro
End Sub


Private Function UbicaVuelo()
Dim firsts As String
firsts = Trim(cmbAVTipo.Text)
cmbAVTipo.ListIndex = 0
Do While firsts <> Trim(cmbAVTipo.Text)
  cmbAVTipo.ListIndex = cmbAVTipo.ListIndex + 1
 Loop
UbicaVuelo = cmbAVTipo.ListIndex + 1
End Function
Private Function UbicaStatus()
Dim firsts As String
firsts = Trim(cmbStatus.Text)
cmbStatus.ListIndex = 0
Do While firsts <> Trim(cmbStatus.Text)
  cmbStatus.ListIndex = cmbStatus.ListIndex + 1
 Loop
UbicaStatus = cmbStatus.ListIndex + 1
End Function
Private Function UbicaTAvion()
Dim firsts As String
firsts = Trim(cmbTAvi.Text)
cmbTAvi.ListIndex = 0
Do While firsts <> Trim(cmbTAvi.Text)
  cmbTAvi.ListIndex = cmbTAvi.ListIndex + 1
 Loop
UbicaTAvion = cmbTAvi.ListIndex + 1
End Function
Private Function UbicaAlin()
Dim firsts As String
firsts = Trim(cmbALna.Text)
cmbALna.ListIndex = 0
 Do While firsts <> Trim(cmbALna.Text)
  cmbALna.ListIndex = cmbALna.ListIndex + 1
 Loop
UbicaAlin = cmbALna.ListIndex + 1
End Function
Private Sub InsertarRegistro()
Dim q As String
Dim IdItinerario As Long
Dim IdAeropuerto As Long
Dim idchpovuelo As Long
Dim idestatus As Long
Dim idcaeropuertos As String
Dim idchpo_avion As Long
Dim idcaerolinea As Long
Dim idcaeropuertosorg As String
Dim idcaeropuertosdest As String
Dim fechvuelo As Date
Dim horariosa As Date
Dim horariofeg As Date
Dim centesimaActual As Integer
Dim i As Integer
Dim fg As Integer
Dim mensaje As String
 idchpovuelo = UbicaVuelo 'cmbAVTipo.ListIndex + 1
 idestatus = UbicaStatus 'cmbStatus.ListIndex + 1
 idchpo_avion = UbicaTAvion 'cmbTAvi.ListIndex + 1
 idcaerolinea = UbicaAlin 'cmbALna.ListIndex + 1
 'idestatus = cmbStatus.ListIndex + 1
 'idchpo_avion = cmbTAvi.ListIndex + 1
 'idcaerolinea = cmbALna.ListIndex + 1
 idcaeropuertosorg = Trim(cmbOrig.Text)
 idcaeropuertosdest = Trim(cmbDes.Text)
 idcaeropuertos = Trim(cmbOrig.Text)
 fechvuelo = Format(Date & " " & Time, "yyyy/mm/dd hh:mm:ss")
 horariosa = MaskEdBox1.Text
 horariofeg = MaskEdBox2.Text
 centesimaActual = Text5.Text
On Error GoTo ern
If (inicia = True And cmdAgregar.Enabled = True) Then
 IdItinerario = DeterMaximo()
 q = "Insert into itinerarios values (" & IdItinerario & "," & idchpovuelo & ","
 q = q + "'" & idestatus & "'," & idcaeropuertosorg & "'," & idchpo_avion & "," & idcaerolinea & ","
 q = q + "'" & idcaeropuertosorg & "'," & idcaeropuertosdest & "'," & Format(fechvuelo, "yyyy/mm/dd hh:mm:ss") & "'," & centesimaActual & "," & Format(Date & " " &
horariosa, "yyyy/mm/dd hh:mm") & "')"
```

```
  q = q + "" & Format(Date & " " & horanolleg, "yyyy/mm/dd hh:mm:ss") & ""
  mensaje = "Registro Insertado"
ElseIf (inicia = True And cmdAgregar.Enabled = False) Then
MSFlexGrid1.Col = 0
IdItinerarioU = MSFlexGrid1.Text
q = "Update itinerarios "
q = q + "    SET "
q = q + "id_c_tipo_vuelo = (select id_c_tipo_vuelo from c_tipo_vuelo where desc_tipo_vuelo like '" & Trim(cmbAVTipo.Text) & "'),"
q = q + "id_c_status    = (select id_c_status from c_status where desc_status like '" & Trim(cmbStatus.Text) & "'),"
q = q + "id_c_aeropuertos = '" & idcaeropuertosorig & "',"
q = q + "id_c_tipo_avion = (select id_c_tipo_avion from c_tipo_avion where desc_c_tipo_a like '" & Trim(cmbTAvi.Text) & "'),"
q = q + "id_c_aerolinea = (select id_c_aerolinea from c_aerolinea where desc_aerolinea like '" & Trim(cmbALna.Text) & "'),"
q = q + "id_c_aeropuertos_orig = '" & idcaeropuertosorig & "',"
q = q + "id_c_aeropuertos_dest = '" & idcaeropuertosdest & "',"
q = q + "fech_vuelo = '" & Format(fechvuelo, "yyyy/mm/dd hh:mm:ss") & "',"
q = q + "horario_sal = '" & Format(horarcsal, "hh:mm") & "',"
q = q + "horario_lleg = '" & Format(horarcolleg, "hh:mm") & "'"
q = q + " WHERE id_itinerario = " & IdItinerarioU & ""
mensaje = "Registro Actualizado"
End If
If inicia = True Then
   cn.Execute q, dbExecDirect
   fijo = MSFlexGrid1.Rows
ObtenCentesima (centesimaActual)
Ibl.Caption = mensaje
If fijo > 1 Then
      For i = fijo - 1 To 1 Step -1
      MSFlexGrid1.RemoveItem i
      Next
End If
   inicia = False
   cargaItinerario
   HabilitaBotones
End If
errY:
If Err.Number <> 0 Then
   MsgBox "Ha ocurrido un error."
    If Err.Number = 3146 Then
      MsgBox "La Sintaxis SQL No es soportada por el control activo de cliente manejador " & cn.Name & " [sintaxis] " & q
   End If
   Exit Sub
End If
End Sub
Private Sub ObtenCentesima(centesimaActual As Integer)
Dim FecPrim As Date
Dim FecLast As Integer
Dim centesimaFinal As Integer
On Error Resume Next
rs.Requery
 rs.MoveLast
 If Not (rs.EOF And rs.BOF) Then
   FecPrim = Format(rs!fech_vuelo, "hh:mm:ss")
   Text6 = Time - FecPrim
   Text6 = Format(CDate(Text6.Text), "ss")
   centesimaActual = rs!centesimas
   centesimaFinal = ((100 - centesimaActual) + Text6)
   If Text6 >= 1 Then
    Text6 = Text6 - 1
   End If
   Do While centesimaFinal >= 100
      centesimaFinal = centesimaFinal - 100
      Text6 = Text6 + 1
   Loop
If (fijo - 1 = rs.AbsolutePosition) Then
   Text3.Text = "Ultima transaccion en " & Text6 & " " & centesimaFinal
End If
añadirArchivo
   SendDataInFile (Text6 & " " & centesimaFinal)
End If
End Sub
Private Sub SendDataInFile(Data As String)
  Const ParaLeer = 1, ParaEscribir = 2, ParaAnexar = 8
  Dim fso, f, Msg
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile("c:\Resultado" & cn.Name & ".txt", ParaAnexar, True)
  f.Write Data
  f.WriteBlankLines 1
  f.Close
End Sub
Private Sub HabilitaBotones()
cmdAgregar.Enabled = True
cmdEliminar.Enabled = True
CmdEditar.Enabled = True
cmdGuardar.Enabled = True
cmdVC.Enabled = True
cmbAVTipo.Locked = True
cmbALna.Locked = True
cmbStatus.Locked = True
cmbTAvi.Locked = True
cmbOrg.Locked = True
cmbDes.Locked = True
MaskEdBox1.Enabled = False
MaskEdBox2.Enabled = False
```

```
End Sub
Private Function ObtenMaximo() As Integer
Dim q As String
Dim Rst As Recordset
q = "Select MAX(Id_Itinerario)+1 AS MAXI  from intinerarios "
Set Rst = cn.OpenRecordset(q, dbOpenDynaset)
If IsNull(Rst.Fields(0)) = False Then
 ObtenMaximo = Rst.Fields(0)
 Else
 ObtenMaximo = 1
End If
Rst.Close
Set Rst = Nothing
End Function
Private Sub cmdIngresa_Click()
Dim k As Integer
Dim n As Integer
n = InputBox("¿Número de Registros a Ingresar?", "Aeropuerto")
For k = 1 To n
  inicia = True
  InsertarRegistro
Next k
inicia = False
End Sub
Private Sub cmdsalir_Click()
rs.Close
Set rs = Nothing
cn.Close
Set cn = Nothing
Unload Me
End Sub
Private Sub cmdVC_Click()
Check1.Value = 0
Load frmseleCata
frmseleCata.Show
End Sub
Private Sub Form_Load()
conectate
formatItinerario
cargaItinerario
inicia = False
centesima = 1
cargaTVuelo
cargaSbt
cargaTAvion
cargaTALin
cargaCAptoO
cargaCAptoD
End Sub
Public Sub cargaCAptoD()
Dim rstCA As Recordset
 Dim q As String
 Dim i As Integer
i = 1
 q = "Select id_c_Aeropuertos from c_aeropuertos"
cmbDes.Clear
 ReDim R_CAptcD(1)
Set rstCA = cn.OpenRecordset(q, dbOpenDynaset)
  While Not rstCA.EOF
    ReDim Preserve R_CAptoD(i)
    R_CAptoD(i) = rstCA.Fields(0)
    cmbDes.AddItem rstCA.Fields(0)
    rstCA.MoveNext
    i = i + 1
  Wend
rstCA.Close
Set rstCA = Nothing
End Sub
Public Sub cargaCAptoO()
 Dim rstCA As Recordset
 Dim q As String
 Dim i As Integer
i = 1
 q = "Select id_c_Aeropuertos from c_aeropuertos"
cmbOrig.Clear
ReDim R_CAptoO(1)
Set rstCA = cn.OpenRecordset(q, dbOpenDynaset)
  While Not rstCA.EOF
    ReDim Preserve R_CAptoO(i)
    R_CAptoO(i) = rstCA.Fields(0)
    cmbOrig.AddItem rstCA.Fields(0)
    rstCA.MoveNext
    i = i + 1
  Wend
rstCA.Close
Set rstCA = Nothing
End Sub
Public Sub cargaTALin()
'R_TALin() As String
 Dim rstCA As Recordset
 Dim q As String
 Dim i As Integer
```

```
 = 1
q = "Select Desc_Aerolinea from c_aerolinea"
 ReDim R_TALin(1)
 cmbALna.Clear
Set rstCA = cn.OpenRecordset(q, dbOpenDynaset)
 While Not rstCA.EOF
   ReDim Preserve R_TALin(i)
   R_TALin(i) = rstCA.Fields(0)
   cmbALna.AddItem rstCA.Fields(0)
   rstCA.MoveNext
    = i + 1
 Wend
rstCA.Close
Set rstCA = Nothing
End Sub
Public Sub cargaTAvion()
R_TAvion() As String
Dim rstCA As Recordset
Dim q As String
Dim i As Integer
 = 1
 = "Select Desc_C_Tipo_A from c_tipo_avion;"
cmbTAvi.Clear
ReDim R_TAvion(1)
Set rstCA = cn.OpenRecordset(q, dbOpenDynaset)
 While Not rstCA.EOF
  ReDim Preserve R_TAvion(i)
  R_TAvion(i) = rstCA.Fields(0)
  cmbTAvi.AddItem rstCA.Fields(0)
  rstCA.MoveNext
   = i + 1
 Wend
rstCA.Close
Set rstCA = Nothing
End Sub
Public Sub cargaStit()
Dim rstCA As Recordset
Dim q As String
Dim i As Integer
 = 1
 = "Select Desc_Status from c_status"
cmbStatus.Clear
ReDim R_Stit(1)
Set rstCA = cn.OpenRecordset(q, dbOpenDynaset)
 While Not rstCA.EOF
  ReDim Preserve R_Stit(i)
  R_Stit(i) = rstCA.Fields(0)
  cmbStatus.AddItem rstCA.Fields(0)
  rstCA.MoveNext
   = i + 1
 Wend
rstCA.Close
Set rstCA = Nothing
End Sub
Public Sub cargaTVuelo()
Dim rstCA As Recordset
Dim q As String
Dim i As Integer
 = 1
 = "Select Desc_Tipo_Vuelo from c_tipo_vuelo"
ReDim R_TVuelo(1)
cmbAVTipo.Clear
Set rstCA = cn.OpenRecordset(q, dbOpenDynaset)
 While Not rstCA.EOF
  ReDim Preserve R_TVuelo(i)
  R_TVuelo(i) = rstCA.Fields(0)
  cmbAVTipo.AddItem rstCA.Fields(0)
  rstCA.MoveNext
   = i + 1
 Wend
rstCA.Close
Set rstCA = Nothing
End Sub
Private Sub conectate()
 Set ws = CreateWorkspace("", "Admin", "", dbUseODBC)
 Set cn = ws.OpenConnection("", dbDriverNoPrompt, , "ODBC;DATABASE=Aeropuerto;UID=dba;PWD=sql;DSN=Aeropuerto")
 Set cn = ws.OpenConnection("", dbDriverPrompt, , "")
End Sub
Private Sub cargaItinerario()
Dim q As String
Dim i As Integer
Dim tempoUno As Integer
On Error Resume Next
i = 0
q = "Select A.Desc_Tipo_Vuelo, B.Desc_Aerolinea, C.Desc_Status, D.Desc_C_Tipo_A,"
q = q & "     E.id_c_aeropuertos_orig , E.id_c_aeropuertos_dest, E.fech_vuelo, E.horario_sal, E.horario_lleg, E.id_itinerario, E.centesimas"
q = q & "     From  c_tipo_vuelo A, c_aerolinea B, c_status C,  c_tipo_avion D,  itinerarios E"
q = q & "     Where A.id_c_tipo_vuelo = E.id_c_tipo_vuelo"
q = q & "     and   B.id_c_aerolinea  = E.id_c_aerolinea"
q = q & "     and   C.id_c_status     = E.id_c_status"
q = q & "     and   D.id_c_tipo_avion = E.id_c_tipo_avion order by id_itinerario ;"
```

```
Set rs = cn.OpenRecordset(q, dbOpenDynaset)
If Not rs.EOF And Not rs.BOF Then
   rs.MoveFirst
End If
If Not rs.EOF Then
'yyyy/mm/dd hh:mm:ss
Text1.Text = "Operaciones del dia " & Date & " " & Time
 Text2.Text = Abs(rs.RecordCount) & " Reg stros, conexión " & cn.Name
 'Text3.Text = "No se ha insertado un ultimo registro "
If Check1.Value <> 1 Then
   cmbAVTipo.Text = rs.Fields(0)
   cmbALna.Text = rs.Fields(1)
   cmbStatus.Text = rs.Fields(2)
   cmbTAvi.Text = rs.Fields(3)
   MaskEdBox1.Mask = ""
   MaskEdBox1.Text = Format(rs.Fields(7), "hh:mm:ss AM/PM")
   MaskEdBox2.Mask = ""
   MaskEdBox2 = Format(rs.Fields(8), "hh:mm:ss AM/PM")
   cmbOrig.Text = rs.Fields(4)
   cmbDes.Text = rs.Fields(5)
 End If
End If
   While Not rs.EOF
    ic = ic + 1
    MSFlexGrid1.AddItem rs.Fields(9) & Chr(9) & rs.Fields(0) & Chr(9) & rs.Fields(1) & Chr(9) & rs.Fields(2) & Chr(9) & rs.Fields(3) & Chr(9) & rs.Fields(4) & Chr(9) &
rs.Fields(5) & Chr(9) & Format(rs.Fields(7), "hh:mm:ss AMPM") & Chr(9) & Format(rs.Fields(8), "hh:mm:ss AMPM")
    ic = ColorRenglon(rs.Fields(2), ic + 1)
    rs.MoveNext
   Wend
End Sub
Private Sub formatItinerario()
Dim i As Integer
   MSFlexGrid1.Clear
   MSFlexGrid1.Cols = 10
   With MSFlexGrid1
    ColWidth(0) = 300
    ColWidth(1) = 1200
    ColWidth(2) = 5000
    ColWidth(3) = 1200
    ColWidth(4) = 1500
    ColWidth(5) = 800
    ColWidth(6) = 800
    ColWidth(7) = 1850
    ColWidth(8) = 1850
    ColWidth(9) = 2500
   End With
End Sub
Private Function ColorRenglon(status As String, rwl As Integer) As Integer
Dim ColorO As ColorConstants
Dim j As Integer
Select Case status
  Case "Demorado"
   ColorO = vbRed
  Case "A tiempo"
   ColorO = vbCyan
  Case "Cancelado"
   ColorO = vbGreen
End Select
'vbRed &HFF Rojo
'vbGreen &HFF00 Verde
'vbYellow &HFFFF Amarillo
'vbBlue &HFF0000 Azul
'vbMagenta &HFF00FF Fucsia
'vbCyan
MSFlexGrid1.Rcw = rwl - 1
For j = 0 To 9
    MSFlexGrid1.Col = j
    MSFlexGrid1.CellBackColor = ColorO
    MSFlexGrid1.CellFontBold = True
    MSFlexGrid1.CellFontSize = 11
Next
ColorRenglon = rwl - 1
End Function
Private Sub MaskEdBox2_LostFocus()
cmdGuardar.Enabled = True
cmdGuardar.SetFocus
End Sub
Private Sub MSFlexGrid1_SelChange()
Dim Valor As Integer
   Valor = Val(MSFlexGrid1.Row - 1)
   MuestraDatosNG
End Sub
Private Sub MuestraDatosNG()
 MSFlexGrid1.Col = 0
 tbInf.Caption = "Información de registro " & MSFlexGrid1.Text
 MSFlexGrid1.Col = 1
 cmbAVTipo.Text = MSFlexGrid1.Text
 MSFlexGrid1.Col = 2
 cmbALna.Text = MSFlexGrid1.Text
 MSFlexGrid1.Col = 3
 cmbStatus.Text = MSFlexGrid1.Text
```

```
MSFlexGrid1.Col = 4
cmbTAvi.Text = MSFlexGrid1.Text

MSFlexGrid1.Col = 7
MaskEdBox1.Mask = ""
MaskEdBox1.Text = MSFlexGrid1.Text
MSFlexGrid1.Col = 8
MaskEdBox2.Mask = ""
MaskEdBox2 = MSFlexGrid1.Text
MSFlexGrid1.Col = 5
cmbOrig.Text = MSFlexGrid1.Text
MSFlexGrid1.Col = 6
cmbDes.Text = MSFlexGrid1.Text
'cmbOrig.Text = rs.Fields(4)
'cmbDes.Text = rs.Fields(5)
End Sub
Private Sub Timer1_Timer()
Dim i As Integer
 If Check1.Value = 1 Then
   fijo = MSFlexGrid1.Rows
MSFlexGrid1.Redraw = False
    If fijo > 1 Then
      For i = fijo - 1 To 1 Step -1
       MSFlexGrid1.RemoveItem i
      Next
        ObtenCentesima Text5.Text
      cargaItinerario
      End If
  MSFlexGrid1.Redraw = True
 End If
End Sub
Private Sub Timer2_Timer()
If centesima < 100 Then
 centesima = centesima + 1
Else
 centesima = 1
If Check1.Value = 1 Then
 If MSFlexGrid1.Rows = 1 Then
     MSFlexGrid1.Redraw = False
     ObtenCentesima Text5.Text
     cargaItinerario
     MSFlexGrid1.Redraw = True
 End If
End If
 End If
Text4.Text = Time
Text5.Text = centesima
End Sub
Dim rs As Recordset
Dim MaxVal As Integer

Private Sub cmdborrar_Click()
Dim q As String
Dim res As Integer
Dim mensaje As String
On Error GoTo Err1
q = " delete from  c_aeropuertos where id_c_aeropuertos = '" & Trim(Text1.Text) & "'"
res = MsgBox("Estas seguro de eliminar el registro ", vbYesNo + vbQuestion + vbDefaultButton2)
 If res = vbYes Then
    cn.Execute q, dbExecDirect
    mensaje = "Registro Eliminado"
    rs.Requery
    Muestra rs
  Else
    mensaje = "La operacion se cancelo"
 End If
MsgBox mensaje
Err1
If Err.Number <> 0 Then
  MsgBox "Ha ocurrido un error " & Err.Description
  Exit Sub
End If
End Sub

Private Sub cmdguardar_Click()
Dim IdApto As String
Dim descApto As String
Dim Latitud As String
Dim Longitud As String
Dim q As String

IdApto = Trim(Text1.Text)
descApto = Trim(Text2.Text)
Latitud = Trim(Text3.Text)
Longitud = Trim(Text4.Text)
On Error GoTo errV
If MaxVal <> -1 Then

 q = " Update c_aeropuertos set desc_c_aeropuerto = '" & descApto & "', num_latitud_g = " & Latitud & ", num_longitud_g= " & Longitud & ""
 q = q + " where id_c_aeropuertos = '" & IdApto & "'"
 mensaje = "El Registro se ha Actualizado Satisfactoriamente"
ElseIf MaxVal = -2 Then
 q = "Insert into c_aeropuertos values('" & IdApto & "','" & descApto & "'," & Latitud & "," & Longitud & ")"
```

```
  mensaje = "El Registro se ha Insertado Satisfactoriamente"
End If

HabilitaBotones

cn.Execute q, dbExecDirect

MsgBox mensaje

rs.Requery

errV

If Err.Number <> 0 Then
  MsgBox "Ocurrio un error " & Trim(Err.Description)
  Exit Sub
End If

End Sub

Private Sub HabilitaBotones()

Text1.Locked = True
Text2.Locked = True
Text3.Locked = True
Text4.Locked = True

cmdNuevo.Enabled = True
cmdmodificar.Enabled = True
cmdborrar.Enabled = True
cmdguardar.Enabled = True
Command1.Enabled = True
Command2.Enabled = True
Command3.Enabled = True
Command4.Enabled = True
cmdsalir.Enabled = True

End Sub

Private Sub cmdmodificar_Click()
MaxVal = 4
 Text2.Locked = False
 Text3.Locked = False
 Text4.Locked = False

 cmdNuevo.Enabled = False
 cmdmodificar.Enabled = False
 cmdborrar.Enabled = False
 cmdguardar.Enabled = False
 Command1.Enabled = False
 Command2.Enabled = False
 Command3.Enabled = False
 Command4.Enabled = False
 cmdsalir.Enabled = False

 Text2.SetFocus
End Sub

Private Sub cmdNuevo_Click()


MaxVal = 2
 Text1.Locked = False
 Text2.Locked = False
 Text3.Locked = False
 Text4.Locked = False
 Text1.Text = ""
 Text2.Text = ""
 Text3.Text = ""
 Text4.Text = ""


cmdmodificar.Enabled = False
cmdborrar.Enabled = False
cmdguardar.Enabled = False
Command1.Enabled = False
Command2.Enabled = False
Command3.Enabled = False
Command4.Enabled = False
cmdsalir.Enabled = False

 Text1.SetFocus

End Sub

Private Sub cmdsalir_Click()
rs.Close
```

```
Set rs = Nothing
frmPrincipal.cargaCAploD
frmPrincipal.cargaCAploD

Unload Me
End Sub

Private Sub Command1_Click()
rs.MoveFirst
Muestra rs
End Sub

Private Sub Command2_Click()
rs.MovePrevious


If Not rs.BOF Then
  Muestra rs
  Else
  rs.MoveFirst
  Muestra rs
End If
End Sub

Private Sub Command3_Click()

rs.MoveNext


 If Not rs.EOF Then
   Muestra rs
  Else
   rs.MoveLast
   Muestra rs
End If
End Sub

Private Sub Command4_Click()
rs.MoveLast
 Muestra rs
End Sub

Private Sub Form_Load()

Dim q As String

q = " Select * from c_aeropuertos "
Set rs = cn.OpenRecordset(q, dbOpenDynaset)

If Not rs.EOF Then
  Muestra rs
End If

End Sub


Private Sub Muestra(rs As Recordset)
    Text1.Text = rs.Fields(0)
    Text2.Text = rs.Fields(1)
    Text3.Text = rs.Fields(2)
    Text4.Text = rs.Fields(3)
End Sub

Private Sub Text4_LostFocus()
cmdguardar.Enabled = True
cmdguardar.SetFocus

End Sub
```

APÉNDICE E

Para configurar e instalar el sistema se requieren los siguientes pasos:

Advertencia: es necesario seguir la secuencia recomendada, es decir primero los pasos del servidor y luego los del cliente.

Servidor

1.- Generar el archivo ejecutable LoyalServer.exe.
2.- Crear un origen de datos ODBC para la base de datos que se necesite replicar (B.D. Secundaria).
3.- Configurar los parámetros de inicio en el archivo LoyalServer.bat de la siguiente manera:

LoyalServer **dirección_ip_del_servidor puerto_por_donde_desee_escuchar nombre_de_origen_de_datos usuario contraseña**

4.- Correr el archivo LoyalServer.bat

Cliente

1.- Generar (si no existen) las DLLs Loyal y ObtenIpPuerto desde el compilador de Visual C++ v.6.0
2.- Una vez generadas copiarlas al directorio del sistema Windows system32/ o system/ según sea el caso, dependiendo del sistema operativo.
3.- Configurar el archivo de Loyal.ini; como se indica en el apéndice C, indicando la dirección y el puerto por el cual el servidor escucha los mensajes (es necesario que sea la misma dirección del servidor).
4.- Crear un origen de datos ODBC para la base de datos con la que se éste trabajando (B.D. Primaria), indicando en la opción de seguimiento (tracing) la DLL Loyal.dll .
5.- Dar clic en "start tracing now" –comenzar seguimiento ahora"
6.- Trabajar normalmente con la aplicación y todas las transacciones serán replicadas